BIOS Function Reference

Bconin()

LONG Bconin(*dev*)
WORD *dev*;

Bconin() retrieves a character (if one is waiting) from the specified device.

OPCODE 2 (0x02)

AVAILABILITY All **TOS** versions.

PARAMETERS *dev* specifies the device to read from as follows:

Name	*dev*	Device
DEV_PRINTER	0	Parallel port
DEV_AUX	1	Auxillary device (normally the RS-232 port, however, **TOS** versions with **Bconmap()** can map in other devices to this handle)
DEV_CONSOLE	2	Console device (keyboard)
DEV_MIDI	3	MIDI Port
DEV_IKBD	4	IKBD Controller (not available as an input device)
DEV_RAW	5	Console device (keyboard)
See Overview	6 –	Additional devices (as available)

BINDING

```
move.w      dev,-(sp)
move.w      #$02,-(sp)
trap        #13
addq.l      #4,sp
```

RETURN VALUE **Bconin()** returns a bit array arranged as follows:

Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0
Shift key status (see **Kbshift()**)	Keyboard Scan Code	Reserved (0)	ASCII value

COMMENTS The shift key status is only returned if the system variable *conterm* (char *(0x484)) has bit 3 set. This is normally disabled.

Non-ASCII keys return 0 in bits 7-0.

SEE ALSO **Bconstat(), Cconin(), Cauxin()**

Bconout()

LONG Bconout(*dev, ch*)
WORD *dev, ch*;

Bconout() outputs a character to a named device.

OPCODE 3 (0x03)

AVAILABILITY All **TOS** versions.

PARAMETERS *dev* specifies the output device as follows:

Name	*dev*	Device
DEV_PRINTER	0	Parallel port
DEV_AUX	1	Auxillary device (see note under **Bconin()**)
DEV_CONSOLE	2	Console device (screen)
DEV_MIDI	3	MIDI port
DEV_IKBD	4	Keyboard (IKBD)
DEV_RAW	5	Raw screen device (control characters and escapes are not processed)
See Overview	6 –	Additional devices (as available)

BINDING

```
move.w      ch,-(sp)
move.w      dev,-(sp)
move.w      #$03,-(sp)
trap        #13
addq.l      #6,sp
```

RETURN VALUE **Bconout()** returns 0 if the character was sent successfully or non-zero otherwise.

SEE ALSO **Bconin(), Cconout(), Cauxout(), Cprnout(), Bcostat()**

Bconstat()

LONG Bconstat(*dev*)
WORD *dev*;

Bconstat() determines whether the specified device is prepared to transmit at least one character.

OPCODE 1 (0x01)

AVAILABILITY All **TOS** versions.

PARAMETERS *dev* specifies the device to check as listed under **Bconin**().

BINDING

```
move.w      dev,-(sp)
move.w      #$01,-(sp)
trap        #13
addq.l      #4,sp
```

RETURN VALUE **Bconstat()** returns 0 if no characters are waiting or -1 if characters are waiting to be received.

SEE ALSO **Bconin(), Cconis(), Cauxis()**

Bcostat()

LONG Bcostat(***dev*** **)**
WORD ***dev*;**

Bcostat() determines if the specified device is prepared to receive a character.

OPCODE 8 (0x08)

AVAILABILITY All **TOS** versions.

PARAMETERS *dev* specifies the device to poll as listed under **Bconout()**.

BINDING

```
move.w      dev,-(sp)
move.w      #$08,-(sp)
trap        #13
addq.l      #4,sp
```

RETURN VALUE **Bcostat()** returns 0 if the device is not ready to receive characters or -1 otherwise.

CAVEATS A bug in **TOS** 1.0 existed that caused the IKBD and MIDI device numbers to become swapped when being handled by the **Bcostat()** call, subsequently returning data for the wrong device. To allow previously written programs to continue operating correctly, this bug has been maintained on purpose in all current versions of **TOS**. You should therefore specify a value of 3 for the IKBD and 4 for MIDI for this call only.

SEE ALSO **Bconout(), Cauxos(), Cconos(), Cprnos()**

Drvmap()

ULONG Drvmap(VOID)

Drvmap() returns a list of mounted drives.

OPCODE 10 (0x0A)

AVAILABILITY All **TOS** versions.

PARAMETERS None.

BINDING

```
move.w      #$0A,-(sp)
trap        #13
addq.l      #2,sp
```

RETURN VALUE **Drvmap()** returns a **ULONG** bitmap of mounted drives. For each drive present, its bit is enabled. Drive ‘A:’ is bit 0, drive ‘B:’ is bit 1, and so on.

COMMENTS Single floppy systems will indicate that two drives are available since both drives can actually be addressed. A request for drive ‘B:’ will simply cause **TOS** to ask the user to insert ‘Disk B’ and provide automatic handling routines for all disk swapping.

SEE ALSO **Dsetdrv()**

Getbpb()

BPB *Getbpb(*dev*)
WORD *dev*;

Getbpb() returns the address of the current **BPB** (Bios Parameter Block) for a mounted device.

OPCODE 7 (0x07)

AVAILABILITY All **TOS** versions.

PARAMETERS *dev* specifies the mounted device (‘A:’ = 0, ‘B:’ = 1) .

BINDING

```
move.w      dev,-(sp)
move.w      #$07,-(sp)
trap        #13
addq.l      #4,sp
```

RETURN VALUE **Getbpb()** returns a pointer to the device's **BPB**. The **BPB** is defined as follows:

```
typedef struct
{
       WORD recsiz;  /* bytes per sector */
       WORD clsiz;   /* sectors per cluster */
       WORD clsizb;  /* bytes per cluster */
       WORD rdlen;   /* sector length of root directory */
       WORD fsiz;        /* sectors per FAT */
       WORD fatrec;  /* starting sector of second FAT */
       WORD datrec;  /* starting sector of data */
       WORD numcl;   /* clusters per disk */
       WORD bflags;  /* bit 0=1 - 16 bit FAT, else 12 bit */
} BPB;
```

CAVEATS A media change *must* be forced after calling this function prior to making any **GEMDOS** calls. Failure to do so may cause **GEMDOS** to become unaware of a disk change causing data loss. Refer to the discussion of forcing a media change earlier in this chapter.

Getmpb()

VOID Getmpb(***mpb*** **)**

Getmpb() returns information regarding **GEMDOS** free and allocated memory blocks.

OPCODE 0 (0x00)

AVAILABILITY All **TOS** versions.

PARAMETERS *mpb* is a pointer to a **MPB** structure which is filled in by the function. The related structures are defined as follows:

```
typedef struct md
{
       struct md *m_link;     /* pointer to next block */
       VOIDP m_start;         /* pointer to start of block */
       LONG m_length;         /* length of block */
       BASEPAGE *m_own;       /* pointer to basepage of owner */
} MD;

typedef struct mpb
{
       MD *mp_mfl;            /* free list */
       MD *mp_mal;            /* allocated list */
       MD *mp_rover;          /* roving pointer */
} MPB;
```

BINDING

```
pea         mpb
clr.w       -(sp)
trap        #13
addq.l      #6,sp
```

CAVEATS

MultiTOS uses a very different method of memory management which makes this call useless.

COMMENTS

An application should *never* attempt to modify any of the returned information nor make any assumptions about memory allocation because of this function.

SEE ALSO

Malloc(), Mfree()

Kbshift()

LONG Kbshift(*mode*)
WORD *mode*;

Kbshift() allows the user to interrogate or modify the state of the keyboard 'special' keys.

OPCODE

11 (0x0B)

AVAILABILITY

All **TOS** versions.

PARAMETERS

mode is -1 to read the state of the keys or a mask of the following values to change the current state:

Name	Mask	Meaning
K_RSHIFT	0x01	Right shift key depressed
K_LSHIFT	0x02	Left shift key depressed
K_CTRL	0x04	Control key depressed
K_ALT	0x08	Alternate key depressed
K_CAPSLOCK	0x10	Caps-lock engaged
K_CLRHOME	0x20	Clr/Home key depressed
K_INSERT	0x40	Insert key depressed

BINDING

```
move.w      mode,-(sp)
move.w      #$0B,-(sp)
trap        #13
addq.l      #4,sp
```

RETURN VALUE

Kbshift() returns the state that the keyboard 'special' keys were in prior to the call.

COMMENTS

Kbshift() is not a particularly fast call. If you are only interested in reading the state a documented macro follows that replaces **Kbshift()** and is much faster. Call the kb_init() function, as shown below, before using:

```
char *p_kbshift;
#define Kbstate()    *p_kbshift

VOID
kb_init(VOID)
{
       /* GetROMSysbase is defined in the BIOS Overview */
       OSHEADER *osheader = GetROMSysbase();

       if ( osheader->os_version == 0x0100 )
             p_kbshift = (char *)0xe1bL;
       else
             p_kbshift = *(char **)osheader->p_kbshift;
}
```

SEE ALSO

evnt_keybd(), evnt_multi(), Cconin(), Bconin()

Mediach()

LONG Mediach(***dev*** **)**
WORD ***dev*****;**

Mediach() inquires as to whether the 'media' has been changed since the last disk operation on a removable block device (floppy, removable hard drive, floptical, etc...).

OPCODE

9 (0x09)

AVAILABILITY

All **TOS** versions.

PARAMETERS

dev specifies the mounted device number to inquire ('A:' = 0, 'B:' = 1, etc.).

BINDING

```
move.w      dev,-(sp)
move.w      #$09,-(sp)
trap        #13
addq.l      #4,sp
```

RETURN VALUE

Mediach() returns one of three values:

Name	Value	Meaning
MED_NOCHANGE	0	Media has not changed
MED_UNKNOWN	1	Media may have changed
MED_CHANGED	2	Media has changed

SEE ALSO **Getbpb()**

Rwabs()

LONG Rwabs(***mode, buf, count, recno, dev, lrecno*** **)**
WORD ***mode*****;**
VOIDP ***buf*****;**
WORD ***count,recno,dev*****;**
LONG ***lrecno*****;**

Rwabs() reads and writes sectors to a mounted device.

OPCODE 4 (0x04)

AVAILABILITY All **TOS** versions. Hard disk access requires the use of a hard disk driver (such as **AHDI**). The long sector offset version is only available as of **AHDI** 3.0. **AHDI** version numbers can be inquired through system variable *pun_ptr* (see discussion earlier in this chapter).

PARAMETERS *mode* is a bit mask which effects the operation to be performed as follows:

Name	Bit	Meaning
RW_READ or **RW_WRITE**	0	0 = Read, 1 = Write
RW_NOMEDIACH	1	Do not read or modify the media change status.
RW_NORETRIES	2	Disable retries
RW_NOTRANSLATE	3	Do not translate logical sectors into physical sectors (*recno* specifies physical instead of logical sectors)

The read or write operation is performed at address *buf*. *buf* must be *count* * bytes per logical sector in logical mode or *count* * 512 bytes in physical mode. *count* specifies how many sectors will be transferred.

dev specifies the index of the mounted device. In logical mode, ‘C:’ is 2, ‘D:’ is 3, etc... In physical mode, devices 2-9 are the ACSI devices and 10-17 are SCSI devices.

recno specifies the first sector to read from. If you need to specify a long offset, set *recno* to -1 and pass the long value in *lrecno*. When using a version of the **AHDI** below 3.0, the parameter *lrecno* should not be passed.

BINDING

```
/* If running AHDI <3.0 omit first parameter */
```

```
move.l       lrecno,-(sp)
move.w       dev,-(sp)
move.w       recno,-(sp)
move.w       count,-(sp)
pea          buf,-(sp)
move.w       mode,-(sp)
move.w       #$04,-(sp)
trap         #13
lea          18(sp),sp
```

RETURN VALUE

Rwabs() returns **E_OK** (0) if successful or a negative **BIOS** error code otherwise.

COMMENTS

Some C compilers (Lattice C in particular) have a secondary binding called **Lrwabs()** used to pass the additional parameter.

This function may invoke the critical error handler (*etv_critic*).

Setexc()

(VOIDP)() Setexc(*num*, *newvec*)
WORD *num*;
VOID (*newvec)();

Setexc() reads or modifies system exception vectors.

OPCODE

5 (0x05)

AVAILABILITY

All **TOS** versions.

PARAMETERS

num indicates the vector number you are interested in. To obtain the vector number divide the address of the vector by 4. Some common vectors are:

Name	*num*	Vector
VEC_BUSERROR **VEC_ADDRESSERROR** **VEC_ILLEGALINSTRUCTION**	0x02 - 0x04	Bomb errors (Bus, Address, Instruction)
VEC_GEMDOS	0x21	Trap #1 (**GEMDOS**)
VEC_GEM	0x22	Trap #2 (**AES/VDI**)
VEC_BIOS	0x2D	Trap #13 (**BIOS**)
VEC_XBIOS	0x2E	Trap #14 (**XBIOS**)
VEC_TIMER	0x100	System timer (*etv_timer*)
VEC_CRITICALERROR	0x101	Critical error handler (*etv_critic*)
VEC_TERMINATE	0x102	Process terminate handle (*etv_term*)

newvec should be the address of your new vector handler. Passing a value of

VEC_INQUIRE ((VOIDP)-1) will not modify the vector.

BINDING

```
pea         newvec
move.w      num,-(sp)
move.w      #$05,-(sp)
trap        #13
addq.l      #8,sp
```

RETURN VALUE The original value of the vector is returned by the call.

COMMENTS You must reinstate old vector handlers you changed prior to your process exiting.

Programs which modify replace system vector code should install themselves following the conventions of the XBRA protocol. For details, consult the overview portion of this chapter.

Tickcal()

LONG Tickcal(VOID)

Tickcal() returns the system timer calibration.

OPCODE 6 (0x06)

AVAILABILITY All **TOS** versions.

PARAMETERS None.

BINDING

```
move.w      #$06,-(sp)
trap        #13
addq.l      #2,sp
```

RETURN VALUE **Tickcal()** returns a **LONG** indicating the number of milliseconds between system clock ticks.