NAME

a.s - details of CO-IDRIS startup

doshdr.86 gives control to IDRIS at line 9 via a call to **_main**. **_main** records the segment registers, sets up a stack, and calls **main** with a pointer to the command line. On return, **_main** clears the dynamic data area and the user area memory, lines 18 to 30. **_main** calls **init** to start processes 0 and 1. **init** places the code between lines 108 and 136 into user memory and executes it as process 1. At some point, the **exec** system call (line 126) or the **/odd/init** program will roadblock. At that point control will resume at line 33 with the resident running as process 0. **swapp** is the process 0 code; it never returns.

exit (lines 37 to 41) is provided for the C library. The quit command, via the **undo** code, calls exit to return to DOS.

vectab, (lines 48 to 54), is a table of **int** locations to be plugged at startup (see the **main.c** manual page in this appendix). The entries are three words in length: the first word is the interrupt number to plug (the location modified is the **int** number * 4); the second word is the address of the routine to be entered (**ip** value); and the third word is the code section to run in at the time of the interrupt (**cs** value). Section -1 is taken to mean the resident data section. All interrupts run in the data section at first. The -1 value at line 54 is the end of the table.

The first entry, line 49, is the clock interrupt. It is initialized to **int** 0 in order to disable it. The startup code in **main** may patch this to another value, in order to enable the clock. For this reason, the clock entry must be first in the table. See the section below on **Adding a Clock** for instructions on implementing your own hardware clock when not running under DOS.

Line 50 is the DOS system call intercept, and must be the second entry. If you are making an IDRIS/86 independent of DOS you will have to change line 49 and delete line 50.

Lines 51 to 53 plug the hardware exception vectors.

User programs enter the resident at **sysent** (line 63). Every user program reserves the first 32 bytes of its text space for system entry code. The system entry code is lines 111 to 122. The instruction at line 12 sets up this code. The **exec** system call checks that the first 4 bytes of the user program matches the 4 bytes at **_usrsvc**, lines 111 to 114. If this is the case, the next 28 bytes (lines 115 to 121) are moved to the user space.

Hardware exceptions overflow, trace trap, and zero divide are dealt with in lines 78 to 94.

The **csw** (get console switches) system call (lines 96 to 100) is redefined for CO-IDRIS to cause a return to DOS.

The IBM serial port interrupts enter at **aca0** and **aca1**. The **int** instructions were plugged by a call to **vector** in **acaop** (open) in the **aca.c** driver file. This interrupt code, lines 139 to 157, should be used as a model for adding other interrupting devices. The hardware interrupt passes control, via the proper **int** location, to **aca0** (line 141). The call to **trapint**, line 142, does several things: saves registers, enters system mode, checks whether the user or the resident was interrupted, sets up a stack, sets **ax** to the return location, sets **dx** to the return segment, sets **cx** to 0 if the system was running, or 0x100 if the user was running, and sets up for a proper return from the interrupt. This interrupt code passes an argument to the C routine in the driver. **aca0** pushes a 0 onto the stack (lines 143 to 145); **aca1** pushes a 1 onto the stack (lines 156, 157, and 145). **ax** is the return location in the resident text space. **ax** is decremented once for each argument pushed; this causes a pop for each argument on return. Up to 3 arguments (and 3 dec **ax's**) may be performed. The C routine is entered via a return intersegment (lines 148 to 151). Remember: interrupts are handled in the data section, but the C code is handled in the text section.

Adding a Clock

CO-IDRIS comes with special code to deal with clock interrupts. This code is entered at **tick**; the hook is made in line 49. To make a clock for an IDRIS running independently of DOS, change line 49 to:

```
    .word   0xVV, mytick, -1
```

and add this code to **a.s**:

```
mytick:
    call    trapint
    push    ax
    push    dx
    lea     ax,_clock
    push    ax
    reti
```

0xVV is the clock interrupt number. The above code is the model for a device interrupt with no arguments passed. Note: consult **main.c** for setting the proper value of **tinc**, the clock frequency indicator.

If you need to adjust the clock hardware, do so just before the **reti** instruction. Registers **ax**, **dx** and **cx** are available. Preserve all other registers.

NAME

a.s code - starting up CO-IDRIS

```
/   CO-IDRIS STARTUP
/   copyright (c) 1984 by Whitesmiths, Ltd.

////////////////////////////
/   BEGINNING OF PROGRAM   /
////////////////////////////
.text
    .public __main      / doshdr gives control here
__main:
    mov     _cseg,cs    / write down resident seg register
    mov     _dseg,ds
    mov     syseg,ds    / system call entry seg
    pop     cx
    pop     cx          / command line ptr
    lea     sp,_u0      / init stack
    push    cx          / pass command line ptr
    call    _main       / do command line
    .cseg
    mov     ax,*2       / DOS top of memory
    mov     _topseg,ax
    lea     di,__edata  / clear memory
    mov     ax,ds
    mov     es,ax
    mov     cx,_systop
    sub     cx,di
    shr     cx
    sub     ax,ax
    cld
    rep
    stos.w  [di]
    cli                 / start procs 0 and 1
    call    _init
    call    _swapp
    jmp.s   .           / should never return

    .public _exit
_exit:
    mov     ax,0x4c     / dos 2.2 exit
    push    ax
    call    _dosy
    jmp.s   .

//////////////////////
/   INTERFACE CODE   /
//////////////////////
.data
    .public _vectab
_vectab:
    .word   0x00,tick,-1    / MUST BE FIRST, shdb 0x08 if used
    .word   0x21,DOSc,-1    / MUST BE SECOND, DOS call intercept
```

```
        .word   0x00,zdiv,-1
        .word   0x01,trace,-1
        .word   0x04,oflo,-1
        .word   -1

SIGDOM = 7
SIGRNG = 6
SIGSVC = 16
SIGTRC = 5

/   supervisor call
        .public sysent
sysent:
        cli
svc:
        call    trapint
        add     cx,SIGSVC
trap1:
        sti
        push    cx
        dec     ax
        push    ax              / trap(USR+sig, ax, dx, cx)
        push    dx
        lea     ax,_trap
        push    ax
        reti

/   overflow trap
oflo:
        call    trapint
        add     cx,SIGRNG
        jmp.s   trap1

/   trace trap
trace:
        call    trapint
        add     cx,SIGTRC
        jmp.s   trap1

/   zero divide trap
zdiv:
        call    trapint
        add     cx,SIGDOM
        jmp.s   trap1

/   undo interrupts and shutdown
        .public _csw
_csw:
        call    _undo
        ret

///////////////////
/   PROCESS ONE   /
///////////////////
```

```
.data
    .public _p1boot, _p1end
    .even
_p1boot:
                                / USER MODE SYSTEM CALL
    .public __usrsvc, syseg
__usrsvc:
    jmp.s       2f              / this area re-written by resident
    .word   0x6969
1:
    pop     cx                  / IDRIS system call (prototype)
    pushf
    push    cs
    push    cx
    jmpi    sysent,*0
syseg = .-2
.   =       __usrsvc+32         / first 32 bytes controlled by
                                / resident
2:
    push    p1vec+2
    push    p1vec
    push    cx          / dummy return link
    mov     ax,11       / exec
    call    1b
    mov     ax,38       / undo()
    call    1b
    jmp.s   .
p1vec = . - _p1boot
    .word   p1path, p1vec+4, p1path+5, 0
p1path = . - _p1boot
    "/odd/init\0"
    .even
_p1end:

/   IBM Asynchronous Communications Adaptor (com1, com2)
.data
    .public _aca0
_aca0:
    call    trapint
    mov     cx,0        / acaint(0)
1:
    push    cx          / push line number
    dec     ax          / do extra pop on return for arg passed
    push    ax          / return loc for acaint()
    push    dx          / cseg for reti
    lea     ax,_acaint
    push    ax          / offset for reti
    reti

    .public _aca1
_aca1:
    call    trapint
    mov     cx,1        / acaint(1)
    jmp.s   1b
```

NAME

aca.c - IBM PC-XT serial line driver

ACAVEC (lines 11 to 13) is the structure where the adaptor I/O registers are located. These registers are, in order of successive port addresses: the character buffer (**cbuf**), the interrupt enable register (**ier**), the interrupt request register (**iir**), the line control register (**lcr**), the modem control register (**mcr**), the line status register (**lsr**), and the modem status register (**msr**).

ACASPEED (lines 15 to 17), is the structure containing the 2-byte speed code register.

The manifest constants, lines 27 to 89, define the I/O and interrupt controller register contents.

The number of adaptors and their addresses is defined in lines 95 to 96. **nmaca** (line 111) defines the names of the adaptor **/dev** entries. Note: this driver can support 2 physical lines, with a primary and alternate minor number for each. The primary number, **com1** or **com2**, is used by and the alternate number, **com1.lnk** or **com2.lnk**, is used by <u>cu</u> to take control of the line away from **paca** (line 107) is a pointer to the line controller currently in effect. INTMASK (line 100) determines the physical interrupt given the minor number; and ADDRMASK (line 101) determines the physical line number given the minor number. The driver is coded to handle the case where there may be multiple adaptors sharing the same interrupt.

The resident knows the entry points to the driver via **acacdev** (line 105) which is referenced in **main.c**.

acastab (lines 117 to 118) is the table of divisor values for each baud rate.

acacl (lines 120 to 147) is called by the resident for every close of an adaptor line. Nothing happens unless this is the last close (lines 135 to 136). The user program roadblocks until output is flushed (line 137). The line controller is reset (lines 138 to 140). If the line controller being closed is currently being used, its alternate is put in use. If the alternate is not open either, the interrupts for the line are turned off; otherwise the baudrate of the alternate line is used. Note: in this context, **com1** and **com1.lnk** are alternates of one another, as are **com2** and **com2.lnk**.

acaop (lines 155 to 228) is called by the resident for each **open** system call (**/dev/com1**, et. al.). **devm**, line 166, is the minor number (0-3); whereas **mdevm**, line 169, is the physical line number (0-1).

acaop ensures that only defined minor numbers can be opened (lines 171 to 175). **pv**, line 176, gets the I/O address of the physical line. **pt** (line 177) goes to the address of the line controller to get the minor number.

On the first **open** to a line (lines 178 to 206), the interrupt vector is plugged and the default line speed and stty-settable parameters are initialized. **dseg**, (lines 160, 183, and 187) is the resident data section (**ds** register). aca0 and aca1 are in module **a.s.** aca0 and aca1 call acaint at line 290.

If the alternate line controller is open, the speed and parameters are copied (lines 197 to 198). If the line is not opened to either line controller (line 209) an attempt is made to set up the hardware for operation (lines 211 to 213). If the hardware is installed (lines 214 to 217) the pointer to the line controller is established, otherwise a failure is reported. If the line is already opened, an **open** to comm1.lnk takes precedence over com1, and com2.lnk takes precedence over com2 (lines 219 to 220).

The line is marked "open" if the hardware appears to be installed (lines 221 to 226).

acard, lines 232 to 238, is called by the resident for every **read** system call. **ttread** does the work, given a pointer to the line controller for the minor number. Note: the program will roadblock in **ttread** until there is input. If the line controller is not current, no input will appear.

acaset, lines 242 to 262, sets baud rate and configuration. Note: speed code 0 means that modem signals are turned off (lines 251 to 252).

acasg, lines 266 to 276, is called by the resident for each **stty** and **gtty** system call. **ttset** does all the work unless the call is an **stty**. For an **stty**, **getfl** is NO, and the driver must set the baudrate.

acawr (lines 280 to 286) is called by the resident for each **write** system call. **ttwrite** queues the characters and calls **acago**. Note: the program will roadblock here when the output queue backs up (reaches **out_hi**).

acaint (lines 290 to 317) is entered from the interrupt code in **a.s.** The interrupt routines pass the adaptor number as an argument (lines 290 to 291). The **for** loop, line 300, really only executes once. The code was constructed to allow for multiple physical lines sharing the same interrupt vector. Nothing is done unless there really is an interrupt request (line 301). The status registers are read at lines 303 to 304. If there is input (line 305) the character is read at line 307. If an overrun is indicated (line 308) a **DEL** code is inserted to indicate that data has been lost. This is necessary for cu to function properly. If a framing error or break is indicated (lines 310 to 311) the character is treated as a **DEL** code. The transmit-go routine, **acago**, is called if there is an output interrupt. The interrupt controller chip is cleared on the way out (line 317). Note: **ttin** will result in a call to **acago** if echoing is being done.

acago (lines 322 to 360) is called by **ttwrite** and **ttin** to start the next character transmission. A pointer to **acago** is established by **acaop** (line 196). **acago** first checks whether the transmitter is busy (line 332). **T_STOP** is set if an **X-OFF** (ctl-s) was received. T_TIMER is set if a delay is in progress (lines 354 to 355). deqc returns the next character (or -1 if none is present) at line 333. In raw mode, 8 bits are transmitted as is (lines 335 to 336). In normal mode, 7 bits and parity is sent (lines 337 to 351). M_EVEN and M_ODD determine 1 parity (line 341 to 343), even parity (lines 344 to 346), odd parity (lines 347 to 348), or 0 parity (default). If the queue is empty or the low-water mark is reached, processes blocked on output are awakened (lines 358 to 359).

NAME

aca.c code - asynchronous communication adapter

```
/*  IBM PC ASYNCHRONOUS COMMUNICATIONS ADAPTER
 *  copyright (c) 1983 by Whitesmiths, Ltd.
 */
#include <std.h>
#include <res.h>
#include <bio.h>
#include <cio.h>

/*  aca registers
 */
typedef struct {
    UTINY cbuf, ier, iir, lcr, mcr, lsr, msr;
    } ACAVEC;

typedef struct {
    UTINY lsbspeed, msbspeed;
    } ACASPEED;

/*  onunit control struct
 */
typedef struct {
    TEXT *next;
    VOID (*on_fn)();
    BYTES on_a1, on_a2;
    } ONUNIT;

/*  8259 interrupt controller
 */
#define INTA00  0x20     /* control port */
#define INTA01  0x21     /* int mask port */
#define EOI     0x20     /* end-of-interrupt */
#define IRQ0    1        /* int mask bits */
#define IRQ1    2
#define IRQ2    4
#define IRQ3    8
#define IRQ4    16
#define IRQ5    32
#define IRQ6    64
#define IRQ7    128

/*  Line-Control Register
 */
#define WLS0    1        /* word length select bit 0 */
#define WLS1    2        /* word length select bit 1 */
#define STB     4        /* number of stop bits */
#define PEN     8        /* parity enable */
#define EPS     16       /* even parity select */
#define STICK   32       /* stick parity */
#define SBREAK  64       /* set break */
#define DLAB    128      /* divisor latch access bit */

#define CONFIG  (WLS1|WLS0) /* 8-bits, no parity */
```

```

 /*  Modem Control Register
  */
 #define DTR      1        /* data terminal ready */
 #define RTS      2        /* request to send */
 #define OUT1     4        /* output 1 */
 #define OUT2     8        /* output 2 */
 #define LOOP     16       /* loopback mode */

 #define LINEON  (OUT2|DTR|RTS)  /* enable ints and signals */
 #define LINEOFF OUT2            /* enable ints, no signals */

 /*  Modem Status Register
  */
 #define DCTS     1        /* delta clear to send */
 #define DDSR     2        /* delta data set ready */
 #define TERI     4        /* trailing edge ring indicator */
 #define DRLSD    8        /* delta receive line signal detect */
 #define CTS      16       /* clear to send */
 #define DSR      32       /* data set ready */
 #define RI       64       /* ring indicator */
 #define RLSD     128      /* receive line signal detect */

 /*  Interrupt Enable Register
  */
 #define ENABLE  0x7       /* modem status|recv status|xmit|recv */
 #define DISABLE 0

 /*  Line Status Register
  */
 #define DR       1        /* data ready */
 #define OR       2        /* overrun error */
 #define PE       4        /* parity error */
 #define FE       8        /* framing error */
 #define BI       16       /* break interrupt */
 #define THRE     32       /* transmitter holding register empty */
 #define TSRE     64       /* transmitter shift register empty */

 /*  define aca addresses and number
  *  NOTE: You make entries here, in acaop(), and in \
     a.s in order
  *  to add more aca lines.
  */
 LOCAL ACAVEC *acaaddr[] {0x3f8, 0x2f8};
 #define NACA (sizeof (acaaddr) / sizeof (acaaddr[0]))

 /*  minor number encoding
  */
 #define INTMASK     1
 #define ADDRMASK    1

 /*  I/O control structure
  */
 CDEVSW acacdev {&acaop, &acacl, &acard, &acawr, \
    &acasg, nmaca};
LOCAL TTY aca[NACA*2] {0};
LOCAL TTY *paca[NACA] {0};

/*  device name displayed by /dev/cnames
 */
LOCAL TEXT nmaca[] {"com1 com2 com1.lnk com2.lnk"};

/*  Speed codes (divisor latch settings)
 *  0, 50, 75, 110, 134.5, 150, 200, 300,
 *  600, 1200, 1800, 2400, 4800, 9600, exta, extb
 */
LOCAL UCOUNT acastab[16] {0, 2304, 1536, 1047, 857, \
    768, 576, 384,
                    192, 96, 64, 48, 24, 12, 0, 0};

/*  close comm line
 */
LOCAL VOID acacl(dev)
    DEV dev;
    {
    IMPORT ACAVEC *acaaddr[];
    IMPORT BITS ttyps;
    IMPORT TTY aca[];
    IMPORT TTY *paca[];
    FAST COUNT devm = dminor(dev);
    FAST ACAVEC *pv = acaaddr[devm & ADDRMASK];
    FAST TTY *pt = &aca[devm];
    BITS ps;
    TTY *pa = &aca[devm < NACA ? devm + NACA : devm - NACA];

    if (--pt->t_open)
        return;
    wflush(pt);
    ps = spl(ttyps);
    pt->t_stat = 0;
    settyp(dev, NO);
    if (pt == paca[devm & ADDRMASK])
        if (!(paca[devm & ADDRMASK] = pa->t_stat ? pa : NULL))
            out(&pv->ier, DISABLE);
        else
            acaset(pa);
    spl(ps);
    }

/*  open comm line
 *  NOTE: You have to make entries in the switch below to add
 *  more aca lines. The first argument to vector() is \
    the interrupt
 *  vector number; the second arg is the interrupt \
    routine addrress
 *  in a.s; the third arg is the resident DS register.
 */
LOCAL VOID acaop(dev)
```

```
    DEV dev;
    {
    IMPORT ACAVEC *acaaddr[];
    IMPORT BITS ttyps;
    IMPORT BYTES dseg;
    IMPORT TTY aca[];
    IMPORT TTY *paca[];
    IMPORT VOID acago();
    IMPORT VOID aca0(), aca1();
    FAST ACAVEC *pv;
    FAST COUNT devm = dminor(dev);
    FAST TTY *pt;
    BITS ps;
    COUNT mdevm = devm & ADDRMASK;

    if (NACA*2 <= devm)
        {
        uerror(ENXIO);
        return;
        }
    pv = acaaddr[mdevm];
    pt = &aca[devm];
    if (!pt->t_go)
        {
        switch (devm & INTMASK)
            {
        case 0:                                  /* /dev/com1,com1.lnk */
            vector(12, &aca0, dseg);
            out(INTA01, in(INTA01) & ~IRQ4);
            break;
        case 1:                                  /* /dev/com2,com2.lnk */
            vector(11, &aca1, dseg);
            out(INTA01, in(INTA01) & ~IRQ3);
            break;
        /* add entries here */
        default:
            uerror(ENXIO);
            return;
            }
        pt->t_dev = dev;
        pt->t_go = &acago;
        if (paca[mdevm])
            movbuf(&paca[mdevm]->t_speeds, &pt->t_speeds, 6);
        else
            {
            pt->t_speeds = 0x0909;               /* 1200 baud */
            pt->t_erase = CERASE;
            pt->t_kill = CKILL;
            pt->t_flag = M_XTABS|M_ECHO|M_CRTLF;
            }
        }
    settyp(dev, YES);
    ps = spl(ttyps);
    if (!paca[mdevm])
```

```
        {
        out(&pv->lcr, CONFIG);
        acaset(pt);
        out(&pv->ier, ENABLE);
        if (in(&pv->lcr) != CONFIG || in(&pv->ier) != ENABLE)
            uerror(ENODEV);
        else
            paca[mdevm] = pt;
        }
    else if (NACA <= devm)
        paca[mdevm] = pt;
    if (!uerror(0))
        {
        if (!(pt->t_stat & T_OPEN))
            pt->t_stat = T_OPEN|T_CARR;
        ++pt->t_open;
        }
    spl(ps);
    }

/*  read comm line
 */
LOCAL VOID acard(dev)
    DEV dev;
    {
    IMPORT TTY aca[];

    ttread(&aca[dminor(dev)]);
    }

/*  set aca speed
 */
LOCAL acaset(pt)
    TTY *pt;
    {
    IMPORT ACAVEC *acaaddr[];
    IMPORT UCOUNT acastab[];
    FAST ACAVEC *pv = acaaddr[dminor(pt->t_dev) & ADDRMASK];
    FAST UCOUNT speed;
    FAST UTINY lcrreg;

    if (!(speed = acastab[pt->t_speeds & 0xf]))
        out(&pv->mcr, LINEOFF);
    else
        {
        lcrreg = in(&pv->lcr);
        out(&pv->lcr, lcrreg | DLAB);
        out(&pv->lsbspeed, speed);
        out(&pv->msbspeed, speed >> 8);
        out(&pv->lcr, lcrreg);
        out(&pv->mcr, LINEON);
        }
    }

```

```
/*  set or get tty mode
 */
LOCAL VOID acasg(dev, getfl)
    DEV dev;
    COUNT getfl;
    {
    IMPORT TTY aca[];
    IMPORT TTY *paca[];
    FAST TTY *pt = &aca[dminor(dev)];

    if (!ttset(pt, getfl) && pt == paca[dminor(dev) & ADDRMASK])
        acaset(pt);
    }

/*  write comm line
 */
LOCAL VOID acawr(dev)
    DEV dev;
    {
    IMPORT TTY aca[];

    ttwrite(&aca[dminor(dev)]);
    }

/*  adapter interrupt
 */
VOID acaint(devm)
    UTINY devm;
    {
    IMPORT ACAVEC *acaaddr[];
    IMPORT TTY aca[];
    IMPORT TTY *paca[];
    FAST ACAVEC *pv;
    FAST TTY *pt;
    UTINY iir, lsr, msr, c;

    for (; devm < NACA; devm =+ 2)
        if ((pt = paca[devm]) && !((iir = in(&(pv = \
            acaaddr[devm])->iir)) & 1))
            {
            lsr = in(&pv->lsr);
            msr = in(&pv->msr);
            if (lsr & DR)
                {
                c = in(&pv->cbuf);
                if (lsr & OR)
                    ttin(CINTR, pt);
                if (lsr & (FE¦BI))
                    c = CINTR;
                ttin(c, pt);
                }
            if (((iir & 7) == 2) && (lsr & THRE))
                acago(pt);
            }
    out(INTA00, EOI);
    }

/*  start transmitter
 */
LOCAL VOID acago(pt)
    FAST TTY *pt;
    {
    IMPORT ACAVEC *acaaddr[];
    IMPORT COUNT out_lo;
    IMPORT UTINY cmaptab[];
    IMPORT VOID ttrstart();
    FAST ACAVEC *pv = acaaddr[dminor(pt->t_dev) & ADDRMASK];
    FAST COUNT c;

    if (!(in(&pv->lsr) & THRE) || pt->t_stat & \
        (T_STOP|T_TIMER) ||
        (c = deqc(&pt->t_outq)) < 0)
        return;
    else if (pt->t_flag & M_RAW)
        out(&pv->cbuf, c);
    else if (c < 0200)
        {
        switch (pt->t_flag & (M_EVEN|M_ODD))
            {
        case M_EVEN|M_ODD:
            c =| 0200;
            break;
        case M_EVEN:
            c =| cmaptab[c] & 0200;
            break;
        case M_ODD:
            c =| cmaptab[c] & 0200 ^ 0200;
            }
        out(&pv->cbuf, c);
        }
    else
        {
        timeout(&ttrstart, pt, c & 0177);
        pt->t_stat =| T_TIMER;
        return;
        }
    if (pt->t_outq.c_num == 0 || pt->t_outq.c_num == out_lo)
        wakeup(&pt->t_outq);
    }
```

NAME

bio.h - driver header file used by all drivers

Each 512-byte buffer is controlled by a buffer controller, which is of type **BUF** (lines 20 to 32). Buffer controllers are maintained as a doubly-linked list when on the free list (**f_next** and **f_prev**) or in the cache (**b_next** and **b_prev**), and as a null-terminated list when enqueued for a device (**d_next** and **d_prev**). Each device driver has a **DEVTAB**, lines 36 to 44, which heads up its chain. The codes for **b_flag**, lines 5 to 18, indicate the state of a buffer, and communicate between the driver and the resident.

B_ASYNC is set for read-ahead and write-behind operations.

B_BUSY is set when I/O is in progress.

B_DIRTY is set when a buffer contents have changed, but have not been written to their block device.

B_ERROR is set by the driver when the I/O has failed.

B_READ is set before calling the driver to indicate a **read** operation, as opposed to a **write**.

B_TAPE is set by the device driver for blocked tape I/O. Its purpose is to defeat read-ahead and write-behind to keep tape blocks in order.

B_VALID is set by a call to **iodone**; it indicates that the contents of a particular buffer are valid.

B_WANT is set by processes waiting on a busy buffer; it causes a wakeup to be done when the I/O completes.

B_CHR is set by **physio** to mark a character special (raw mode) transfer, as opposed to a block special I/O.

B_CTRL is not really a buffer flag, but a parameter sent to **physio** to indicate a control operation (as opposed to a data transfer). It is used to defeat buffer count and address checks, and to keep the requesting process from being locked in memory.

B_MAP and **B_RESRC** are used in the PDP-11 and the VAX to control mapping hardware. **B_MAP** is set for all time for buffers in the resident address space. **B_RESRC** is set when some mapping registers are allocated.

BDEVSW (lines 46 to 54) defines the table of entry points published by each block device driver.

CDEVSW (lines 58 to 65) defines the table of entry points published by each character device driver.

NAME

bio.h code - blocked I/O header file

```
/*  HEADER FOR BLOCKED I/O OPERATIONS
 *  copyright (c) 1979 by Whitesmiths, Ltd.
 */

/*  codes for b_flag
 */
#define B_ASYNC 00001
#define B_BUSY  00002
#define B_DIRTY 00004
#define B_ERROR 00010
#define B_READ  00020
#define B_TAPE  00040
#define B_VALID 00100
#define B_WANT  00200
#define B_CHR   00400
#define B_CTRL  01000
#define B_MAP   02000
#define B_RESRC 04000

struct buf {
    BUF *f_next;
    BUF *f_prev;
    BUF *b_next;
    BUF *b_prev;
    BITS b_flag;
    DEV b_dev;
    BYTES b_count;
    ULONG b_phys;
    TEXT *b_addr;
    BYTES b_resid;
    BLOCK b_blkno;  /* ULONG b_off[2]; */
    };

/*  the device control table
 */
#define DEVTAB  struct devtab
struct devtab {
    BUF *d_next;
    BUF *d_prev;
    BUF *b_next;
    BUF *b_prev;
    TINY d_stat;
    TINY d_nerr;
    };

/*  the block device switch table entry
 */
typedef struct {
    BOOL (*d_open)();
    VOID (*d_close)();
    VOID (*d_strat)();
    DEVTAB *d_tab;
    TEXT *d_bname;
    } BDEVSW;

/*  the character device switch table entry
 */
typedef struct {
    BOOL (*d_open)();
    VOID (*d_close)();
    COUNT (*d_read)();
    COUNT (*d_write)();
    VOID (*d_sgtty)();
    TEXT *d_cname;
    } CDEVSW;
```

NAME

cio.h - driver header file used only by terminal drivers

TTY (lines 69 to 85) controls character I/O to a terminal. The information in the structure is used by both terminal device drivers and the resident. All the information for a particular terminal is in one of these structures.

Certain character codes , lines 7 to 15, are compiled-in. CEOT is the terminal end-of-file character. CDELIM is an internal code. CERASE is the default character-delete character. CGO is the transmit-go character X-ON (ctl-q). CINTR is the keyboard interrupt character (DEL key). CKILL is the default line-delete character. CQUIT is the keyboard abort character (ctl-\). CSTOP is the transmit-stop character X-OFF (ctl-s). CALERT is the audible tone (bell).

t_flag bits, lines 19 to 26, select the mode of the terminal. M_RARE is set for rare mode. M_XTABS controls tab expansion. M_CASE controls upper-case to lower-case mapping. M_ECHO controls full duplex operation (i.e. echoing of input). M_CRTLF controls carriage-return to line-feed mapping. M_RAW is set for raw mode; it supercedes M_RARE. M_ODD and M_EVEN control parity bit generation.

t_speeds fields, lines 30 to 37, hold the terminal's speed codes and some flags. t_speeds is accessed via stty and gtty system calls, the resident, and the terminal driver. S_ISPEED is the mask for the input baud-rate; or the baud-rate for the line if split speeds are not possible or desirable. S_IBREAK is set by the driver when a break key is detected. S_ILOST is set by the driver when input has been lost (data overrun). S_IREADY is set by ttset (invoked by the resident) for each stty if input is ready. S_OSPEED is the output baud-rate code. S_OBREAK is a command to the transmit routine to send a break. The transmit routine resets this flag when the break is sent. S_ONXON is a flag to the resident to inhibit the transmission of X-ON/X-OFF (flow control) when set. S_OREADY is set by ttset (invoked by the resident) for each stty system call when the output queue is empty.

t_stat flags, lines 41 to 48, indicate the state of the terminal. T_BUSY is set by ttin when an X-OFF has been sent. T_CARR is set by the driver when the line is considered connected (data set ready = true). ttin will not operate unless this flag is true. T_ESC is set for escape sequences (\ received). T_OPEN is set by the driver when the line is opened. ttin will not operate unless this flag is true. T_STOP is set when an X-OFF has been received. This flag is honored by the transmit-go routine in the driver. T_TIMER is set by the transmit-go routine in the driver when a delay is in progress. T_CLOS is set by some drivers to indicate that the line is closing. T_WOPEN is set by some drivers when waiting for the line to connect (data set ready = false). When the line is connected (i.e. the terminal in question is powered-up, and/or a modem has answered a call), T_WOPEN is reset and T_OPEN and T_CARR are set.

CLIST is the structure of each 16-byte character buffer (lines 52 to 56). CHQ is the list-head for each null-terminated list of character buffers, (lines 60 to 65). The character queues are maintained by **deqc**, **enqc**, and **popc**.

NAME

cio.h code - character I/O

```
/*  HEADER FOR CHARACTER I/O OPERATIONS
 *  copyright (c) 1979 by Whitesmiths, Ltd.
 */

/*  various paramters
 */
#define CEOT    0004
#define CDELIM  0377
#define CERASE  '\b'
#define CGO     0021
#define CINTR   0177
#define CKILL   '\25'
#define CQUIT   0034
#define CSTOP   0023
#define CALERT  007

/*  codes for t_flag
 */
#define M_RARE  000001
#define M_XTABS 000002
#define M_LCASE 000004
#define M_ECHO  000010
#define M_CRTLF 000020
#define M_RAW   000040
#define M_ODD   000100
#define M_EVEN  000200

/*  code for t_speeds
 */
#define S_ISPEED    0x000f
#define S_IBREAK    0x0010
#define S_ILOST     0x0020
#define S_IREADY    0x0080
#define S_OSPEED    0x0f00
#define S_OBREAK    0x1000
#define S_ONXON     0x2000
#define S_OREADY    0x8000

/*  codes for t_stat
 */
#define T_BUSY  0001
#define T_CARR  0002
#define T_ESC   0004
#define T_OPEN  0010
#define T_STOP  0020
#define T_TIMER 0040
#define T_WCLOS 0100
#define T_WOPEN 0200

/*  the character buffer structure
```

```
 */
#define CLIST   struct clist
struct clist {
    CLIST *c_next;
    TEXT c_info[16 - sizeof (BYTES)];
    };

/*  the character queue structure
 */
#define CHQ struct chq
struct chq {
    COUNT c_num;
    TEXT *c_first;
    TEXT *c_last;
    };

/*  the tty control structure
 */
#define TTY     struct tty
struct tty {
    CHQ t_rawq;
    CHQ t_outq;
    VOID (*t_go)();
    DEV t_dev;
    BITS t_stat;
    COUNT t_ndel;
    COUNT t_col;
    COUNT t_prevcol;
    COUNT t_open;
    COUNT t_nin;
    BITS t_speeds;  /* returned by stty */
    TEXT t_erase;   /* returned by stty */
    TEXT t_kill;    /* returned by stty */
    BITS t_flag;    /* returned by stty */
    };
```

NAME

dos13.s - description of IBM PC hard disk I/O

dos13 (lines 6 to 26) does an **int** 13 call to the IBM BIOS. The register setup is documented in the IBM PC-XT Technical Reference Manual.

The arguments (line 3) are: **nblk,** the number of 512-byte sectors to transfer; **op,** the operation; **offset,** the 16-bit buffer offset from base; **sec,** the sector with 2 bits of cylinder number; **cyl,** the low 8 bits of cylinder number; **unit,** the disk unit with high-bit set; **head,** the disk head (surface) number; and **base,** the 16-bit paragraph. **dos13** returns 0 or an error code (lines 20 to 22).

dos13st (lines 29 to 43) is a special form of the **int** 13 IBM BIOS call which returns the status in the registers to a C structure. The arguments (line 28) are: **unit,** the disk unit with high-bit set; and **stat,** the address of the structure to hold **status. dos13st** returns 0 or an error code, as do the cx and dx registers in the **stat** buffer (lines 36 to 41).

NAME

code for dos13.s - IBM PC hard disk I/O

```
/    IBM PC BIOS hard disk I/O
/    copyright (c) 1983 by Whitesmiths, Ltd.
/    BITS dos13(nblk, op, offset, sec, cyl, unit, head, base)

     .public _dos13
_dos13:
     push    bp
     mov     bp,sp
     push    bx
     push    es
     mov     al,[bp][4]
     mov     ah,[bp][6]
     mov     bx,[bp][8]
     mov     cl,[bp][10]
     mov     ch,[bp][12]
     mov     dl,[bp][14]
     mov     dh,[bp][16]
     mov     es,[bp][18]
     int     0x13
     jc      1f
     sub     ax,ax
1:
     pop     es
     pop     bx
     pop     bp
     ret

/    BITS dos13st(unit, &stat)
     .public _dos13st
_dos13st:
     push    bp
     mov     bp,sp
     mov     dl,[bp][4]
     mov     ah,8
     int     0x13
     jc      1f
     sub     ax,ax
1:
     mov     bp,[bp][6]
     mov     [bp][0],cx
     mov     [bp][2],dx
     pop     bp
     ret
```

NAME

main.c - file of system configuration parameters

The data specific to CO-IDRIS (lines 7 to 16) are used only by **main.c** and some of the CO-IDRIS device drivers. **MAXCMD** (line 9) is the length of the longest command line tolerated. **_pname** (line 16) is printed by the usage message of **getflags**.

The default filesystems (lines 20 to 21) are encoded as a major number in the high 8 bits, and a minor number in the low 8 bits.

If **swapdev** and **rootdev** are the same (lines 25 to 27), **swapadr** is equal to the size of the root filesystem, and **swapsiz** + **swapadr** is equal to the size of the disk. If you put swapping on another block device (i.e. **swapdev** is not the same as **rootdev**), then **swapadr** should be set to 1 and **swapsiz** would be set to the size of the swap device - 1.

Note: a block device can be implemented on any random access media available. It is typically a disk, but could be RAM, ROM, Dectape II, magnetic bubbles, or whatever. You could put the root file-system in ROM, pipes in a RAM disk, and user space in magnetic bubbles. Leave swapping turned off, unless RAM memory is short. It is best to allocate swapping on some fixed media, like a winchester or RAM disk, although diskettes have been used in a pinch.

blkdevs (lines 32 to 38) is the table of linkages to the block device drivers. **chrdevs** (lines 44 to 52) is the table of linkages to the character device drivers. To remove a device from the system , replace the device name with **&nobdev** (for block special devices), or **&nocdev** (for character special devices). Delete the object module for the driver, and remove any interrupt routine calls from **a.s**. It is better to keep the table entries in order because the position of an entry in **blkdevs** or **chrdevs** determines the major number of the device, which has a direct impact on the contents of the /dev directory.

The character buffering limits (lines 58 to 63) are applied on a terminal-by-terminal basis, independent of the number of terminals in the system. **nclist**, line 69, is the number of character buffers available for the system (each 16-byte buffer holds 14 characters). When adjusting these parameters for your system, remember the statistical nature of character I/O. You also have to tune for your mix of line speeds. A printer will consume at most (**out_hi** / 14) buffers while operating: the process outputting to the printer will roadblock when **out_hi** characters are left. A cu packet (using cu or up) is about 200 characters. Only serial lines use these buffers; bit-mapped displays like the DOS console never tie up more than 2 buffers.

in_max is the number of characters the resident will hold onto for input. You may wish to increase **in_max** if you have a data-gathering application with a microprocessor inputting at 120cps or more. Remember - don't count the baud rate; it's the number of characters being inputted per second versus the input consumption rate of the printer that matters.

in_xoff is the number of characters queued on input before an **X-OFF** character is sent. You can disable this feature for the system by setting **in_xoff** equal to or greater than **in_max**. You can disable this feature on a line-by-line basis with an **stty** call. **in_xon** determines the lower threshold for sending **X-ON**, once **X-OFF** has been sent. It needs to be less than **in_xoff**.

in_lim determines the maximum type-ahead in normal mode.

Much of the data area consumed by IDRIS is dynamic and tuneable, (lines 65 to 92).

The biggest user of space is the block device buffer cache. **nbufs**, line 67, determines the number of buffers and buffer controllers. (Figure on its size being about **512 + (24 * nbufs)** bytes). The minimum number of buffers allowed is 2. If you have 6 buffers or less, set **rahead** (line 104) to **NO**. 10 to 20 buffers is a comfortable number for one or two users on a winchester disk.

(**nclist** * 16) is the number of bytes of character queue consumed. **nmap** * 6 is the length of an internal table: if you see the non-fatal message **MAP!**, increase this number. **nheap** is the number of bytes, in addition to the **init** code, used for the heap area. **nheap** can be very small unless the resident is linked for separate code and data space. Then, it must be 1024 to 4096, depending on load. Processes, in-core inodes, mount entries, and open files consume the heap.

tinc (line 97) encodes the clock frequency. The **tinc** number that the user types (**ntinc**) is rather highly encoded (lines 175 to 180).

svdnib (lines 83 to 92) is a table processed by **nibble** (line 199), which actually sets aside the dynamic data area. The **mask** field determines the addressing boundary; the **ptr** field indicates the cell, (or is set to **NULL**), which gets the address of the area. The **m1** field indicates the pointer to the number (or is set to **NULL**) of items to allocate. The **m2** field (meaningful if **m1** is not **NULL**) points to a word holding the size in bytes of each item.

biops and **ttyps** should be set to 0 to disable interrupts, or set to 0x200 to enable interrupts. Interrupts can be enabled for block I/O, if no block device interrupts are used. Interrupts can be enabled for character I/O, if no serial line interrupts are used. Block and terminal I/O are independent, and can interrupt each other. The clock can interrupt both terminal and block I/O processing, but be careful if **timeout** is used.

panic (lines 112 to 119) should not return. Note that for CO-IDRIS it returns to DOS (line 118). A subtlety to watch for: the **sync** (line 116) may not really complete if disk devices are interrupt-driven. There may need to be a respectful delay in **idloop** before quitting.

A command line, as emitted by DOS, is parsed into the usual argument vector and count (lines 147 to 166). This code is valid only for DOS-style command lines: argument 0 (the actual command) is missing, and the first byte contains a byte count of the command line.

getflags (lines 168 to 173) mulls over the parsed command line, converting any values given, and possibly giving a usage message and exiting (exit in a.s).

Read-ahead gets turned off when there are only a few buffers (line 174) because the extra I/O tends to be a waste.

The clock interrupt number is passed to the table in **a.s** at line 182.

Device codes are derived from the names supplied (lines 183 to 193). The names used are the names stored as text strings in the block device drivers.

Initialization code is re-used as heap area, if the resident is not linked for separate code and data space (lines 194 to 198).

Remove line 196 if **getflags** and **gblkdev** are not called (these are in main-sub.o, in the library file).

NAME

main.c code - system configuration parameters

```
/*  GET OPTION FROM DOS COMMAND LINE
 */
#include <std.h>
#include <res.h>
#include <bio.h>

/*  data specific to Co-Idris on DOS
 */
#define MAXCMD  256
BOOL slowclock {NO};       /* clock rate <= 4Hz */
BYTES clkint {0};          /* default clock int# */
TEXT *pipenm {NULL};       /* pipe device name */
TEXT *rootnm {NULL};       /* root device name */
TEXT *swapnm {NULL};       /* swap device name */
BLOCK _rootoff {2};        /* root offset for drivers */
TEXT *_pname {"idris"};

/*  root and pipe filesystems
 */
DEV rootdev     {1<<8 | 1};     /* default is b: */
DEV pipedev     {1<<8 | 1};     /* must be a mounted, \
                                   writeable filesystem */

/*  swap area
 */
DEV swapdev     {NODEV};        /* no swapping */
BLOCK swapadr   {9000};         /* offset to swap area; CANT \
                                   BE ZERO */
BLOCK swapsiz   {0};            /* size of swap area in blocks */

/*  block special devices
 *  replace unused table entries with &nobdev
 */
BDEVSW *blkdevs[] {
    &nobdev,                /* 0: no device */
    &mxbdev,                /* 1: dosmx (a b c ...) */
    &hdbdev,                /* 2: pchd (hdc0 ... hdd0) */
    &iebdev,                /* 3: dosie (idris.0 idris.1) */
    &mdbdev                 /* 4: md (memory disk) */
    };
COUNT nblkdev {sizeof(blkdevs) / sizeof(blkdevs[0])};

/*  character special devices
 *  replace unused table entries with &nocdev
 */
CDEVSW *chrdevs[] {
    &memcdev,               /* 0: (ps myps cnames bnames, et. al.) */
    &concdev,               /* 1: dos console */
    &acacdev,               /* 2: IBM async adaptor (com1, com2) */
    &mxcdev,                /* 3: raw DOS disk (ra rb rc ...) */
    &hdcdev,                /* 4: raw IBM pchd (rhdc0 ... rhdd0) */
    &mdcdev,                /* 5: raw memory disk (rmd0 rmd1) */
    &iecdev                 /* 6: raw DOS file (ridris.0 ridris.1) */
    };
COUNT nchrdev {sizeof(chrdevs) / sizeof(chrdevs[0])};

/*  character buffering limits
 *  values are numbers of characters per line
 */
UCOUNT in_max   {560};      /* max input buffered in raw mode */
UCOUNT in_xoff  {280};      /* hiwater mark when X-OFF sent */
UCOUNT in_xon   {140};      /* lowater mark when X-ON sent */
UCOUNT in_lim   {560};      /* max characters accepted in \
    normal mode */
UCOUNT out_hi   {280};      /* output block hiwater mark */
UCOUNT out_lo   {140};      /* output unblock, lowater mark */

/*  data space allocated at startup time
 */
BYTES nbufs     {10};       /* number of 512-byte disk buffers */
BYTES nrawbs    {3};        /* number of 24-byte raw-buffer \
    controllers */
UCOUNT nclist   {50};       /* number of 16-byte character \
    buffers */
UCOUNT nmap     {20};       /* number of 6-byte map elements */
BYTES nheap     {200};      /* number of bytes to add to \
    system heap */

/*  nibble table for allocating space at startup
 */
typedef struct {
    BYTES mask, *ptr, *m1, *m2;
    } NIBTAB;

BYTES lit1      {1};
BYTES lit16     {16};
BYTES lit512    {512};

NIBTAB svdnib[] {
    {01, &maplist, &nmap, &mapsiz},
    {01, &pbufs, &nbufs, &bufsiz},
    {01, NULL, &nrawbs, &bufsiz},
    {01, &pheap, &nheap, &lit1},
    {017, &eheap, NULL, NULL},
    {017, &pclist, &nclist, &lit16},
    {017, &buffers, &nbufs, &lit512},
    {017, &systop, NULL, NULL},
    {-1}};

/*  clock tick increment
 *  tinc = 614400 / clock-rate-Hz
 */
ULONG tinc      {614400/(1193180/65536)};   /* IBM PC Hz (~18.2) */

```

```
/*  system flags
 */
BITS biops      {0};        /* processing level for block I/O */
BITS ttyps      {0};        /* processing level for async char I/O */
BOOL fppres     {NO};       /* look for 8087 if set to YES */
BOOL rahead     {YES};      /* read next block on sequential \
    file read */
BOOL rootro     {NO};       /* read-only root if set to YES */
BYTES stdheap   {4096};     /* default stack+heap given by exec */
TEXT where[]                /* system id read by /dev/where */
    {"DOS Idris 6/8/84\0 Serial ####-#####"};

/*  put panic message
 */
VOID panic(s)
    TEXT *s;
    {
    putfmt("Idris crash: %s\n", s);
    sync();
    FOREVER
        undo();             /* return to DOS; instead of idloop() */
    }

/*  MAGIC FUNCTION.
 *  free area between _smain() and _emain()
 */
LOCAL VOID _smain(){}

/*  parse command line and call getflags
 */
BOOL main(s)
    FAST TEXT *s;
    {
    IMPORT BOOL fppres, rahead, rootro, silent, slowclock;
    IMPORT BLOCK swapadr, swapsiz, _rootoff;
    IMPORT BYTES clkint, nbufs, vectab[];
    IMPORT DEV pipedev, swapdev, rootdev;
    IMPORT NIBTAB svdnib[];
    IMPORT TEXT *_pname, *swapnm, *pipenm, *rootnm, _memory[];
    IMPORT ULONG tinc;
    IMPORT VOID _emain(), _smain();
    IMPORT VOID _emain1(), _smain1();
    FAST COUNT i;
    FAST TEXT *t;
    BYTES ntinc, pac;
    COUNT ac, n;
    TEXT *av[64], *q, qbuf[MAXCMD+1];
    TEXT **pav;

    for (q = qbuf, n = *s++; 0 < n; )
        {
        while (0 < n && iswhite(*s))
            --n, ++s;
        if (n <= 0)
```

```
                        break;
                for (; 0 < n && !iswhite(*s); --n, ++s)
                        *q++ = *s;
                *q++ = '\0';
                }
        ac = 1;
        av[0] = _pname;
        for (t = qbuf; t != q; ++t)
                {
                av[ac++] = t;
                while (*t)
                        ++t;
                }
        pac = ac;
        pav = av;
        ntinc = 0;
        getflags(&pac, &pav,
                "+fpp,clkint#,tinc#,+1,pipe*,root*,swapoff#,\
                        swapsize#,swap*,s,\
                        +readonly,nbufs#:F",
                &fppres, &clkint, &ntinc, &_rootoff, &pipenm, &rootnm,
                &swapadr, &swapsiz, &swapnm, &silent,
                &rootro, &nbufs);
        rahead = 6 <= nbufs;    /* turn off read-ahead if few bufs */
        if (!ntinc)
                ;
        else if (0xf000 <= ntinc)
                tinc = 614400L / (-ntinc);
        else
                tinc = ntinc;
        slowclock = 153600L <= tinc;
        vectab[0] = clkint;
        if (rootnm)
                {
                rootdev = gblkdev(rootnm, "-root <root-name>\n");
                pipedev = rootdev;
                }
        if (pipenm)
                pipedev = gblkdev(pipenm, "-pipe <block-name>\n");
        if (swapsiz)
                swapdev = swapnm ? gblkdev(swapnm, "-swap \
                        <block-name>\n") : rootdev;
        else if (swapnm)
                usage("-swap <block-name> -swapsize # -swapoff #\n");
        if (cs(-1) == ds(-1))
                {
                free((BYTES)&_emain1 - (BYTES)&_smain1, &_smain1, 0);
                free((BYTES)&_emain - (BYTES)&_smain, &_smain, 0);
                }
        nibble(&_memory, &svdnib);
        }

/* MAGIC FUNCTION. end of init code, keep in order.
 */
```

```
LOCAL VOID _emain(){}
```

NAME

pchd.c - description of IBM PC-XT hard disk driver

This disk driver calls the IBM PC-XT ROM BIOS code, **int 13**, as defined in the *IBM PC-XT Technical Reference Manual*.

This disk has a partition table as part of its boot block . There can be from 1 to 4 partitions on the disk. There are 4 slots in the partition table, as expected. However, the IBM **fdisk** utility uses the slots at the end of the table. Thus, if there are 2 partitions, **fdisk** partition 1 winds up as slot 3, and **fdisk** partition 2 winds up as slot 4 (given that slots 1-4 are available). The following is a table of boot-record slots and the partition numbers they may hold:

	fdisk partition numbers			
slot 1	1			
slot 2	2	1		
slot 3	3	2	1	
slot 4	4	3	2	1

The minor numbers are encoded as shown:

UUUUUPPP	
xxxxx000	refers to the entire disk, block 0 is the boot record
xxxxx001	partition 1, like fdisk
xxxxx010	partition 2
xxxxx011	partition 3
xxxxx100	partition 4
xxxxx101	not used
xxxxx110	not used
xxxxx111	not used

The high 5 bits are the disk number: C or D disk. The entries which use partitions 1 to 4 have an offset (**_rootoff**) added so that the first 2 blocks are skipped. This is to maintain compatibility with the **/dev/[a-h]** interface to the disk.

If the low 3 bits of the minor number are 0 (corresponding to **/dev/hdc0** and **/dev/hdd0**), the entire disk is accessed, and no offsets are applied. Block 0 of **/dev/hdc0** is the boot block.

The other limitation this driver deals with is the fact that DMA operations may not cross a physical 64K boundary.

BIOSRS (line 10) is a **reset** command. **BIOSRD** (line 11) is a **read** command. **BIOSWR** (line 12) is a **write** command. **BIOSRC** (line 13) is a **recalibrate** command (it rewinds the disk). **CYLBITS** (line 14) is a **mask**: the high 2 bits of the sector register is really the high 2 bits of the 10-bit cylinder number.

UNISHFT (line 18) enforces the minor number encoding above. The hard disk unit number is the high 5 bits of the minor number. **NHD** (line 19) controls the number of hard disks supported by this driver. It can be any number, and directly impacts table sizes.) **NPART** (line 20) is the number of partitions the driver controls. Partitions 1 to 4 are **fdisk** partitions, and partition 0 is the entire disk. **NDPART** (line 21) is the number of partition table entries on the disk, referred to as "slots" above.

HDTAB (lines 25 to 27) is the per-disk information the driver maintains. **PARTITION** (lines 29 to 33) is the layout of a partition table entry as it appears on disk and in memory. **pt_secoff** is the sector offset of the start of the partition. **pt_nsec** is the number of sectors in the partition. The sectors are 512 bytes in length.

BOOTREC (lines 35 to 39) is the layout of the boot record on disk. Note that a boot-record consists of a bootstrap program, 4 partition table entries, and 2 signature bytes.

hdbdev (line 41) is the table of entry points for block I/O. **hdcdev** (line 42) is the table of entry points for character (non-buffered) I/O.

hdtab (line 43) is used by the resident to manage the buffer cache for this disk. **DEVTAB** is defined in the manual page on **bio.h** in this appendix. Since this driver is not interrupt-driven, I/O is synchronous, and **enq** and **deq** are not needed. Also, the strategy routine completes the I/O; there is no interrupt or start routine as in other drivers.

hd (line 44) is a table of per-disk information, indexed by hard disk number.

hdp (line 45) is the table of disk partitions managed by the driver. **hdp** is indexed by both the disk number and the in-memory partition number (0-4). **mappart** (lines 205 to 214) is always used to calculate the address of the proper entry in this table, given the minor number.

nmhd and **nmrhd** (lines 46 to 48) are the names of the minor numbers that access this driver. If you change **NHD** (lines 44 and 45) you will have to create or delete disk names from these two strings. Note the extra blanks between "hdc4" and "hdd0": these are significant because they skip the unused minor numbers.

hdclose (lines 50 to 58) counts down the open count for each disk. This open count is used by **hdopen** (lines 80 to 81) as a first time flag. Note that if the hard disk is the root, swap, or pipe disk it will always stay open. If it is not (if the root is on diskette, for instance), the open count may go to 0 and the boot block will be re-read on the next open. This behavior differs from that of DOS, which only accesses the boot-record when the system is initialized. It allows IDRIS to be more responsive, for good or ill, to partition table changes (via **idisk** or **fdisk**).

hdopen (lines 62 to 108) counts opens and does a read of the partition table when the open count is 0.

A check is made for a valid hard disk number at lines 78 to 79. Opens are counted at lines 80 to 81. Nothing else is done unless the open count is 0.

The open count is 0 in this case, so the disk information is obtained from the BIOS (lines 83 to 83). An error is reported if the BIOS call fails, and nothing else is done.

The partition table is read (lines 84 to 91). A buffer is obtained to hold the boot-record using the getblk/brelse machinery. If the read fails or the signature bytes are not right, the partition table is considered invalid. In this case a message is printed (line 88), the buffer is released (line 89), and an error is reported (line 90).

If all is well so far, the partition table actually used by the driver is set up (lines 92 to 108). The per-disk information needed to calculate seek addresses from block numbers is computed (lines 94 to 96). **hd_ncyl** holds the number of cylinders. **hd_nspt** holds the number of sectors per track (per head). **hd_nspc** holds the number of sectors per cylinder. "partition 0" is set up to be the size of the entire disk (line 97). The in-memory partition table is copied from the boot-record and made to line up with the fdisk numbering scheme explained above (lines 98 to 105). Here the offset described above is applied (lines 102 to 103). The test at line 99 skips over empty slots.

The buffer is released and the open is counted (lines 106 to 107).

hdread (lines 111 to 119) is a character (non-buffered) read of the disk. **physio** is called to set up a buffer controller (one of the **rawbufs**) and to re-enter the driver at **hdstrat** to do the I/O. **hdstrat** must be written in anticipation of both a byte count other than 512 and the existence of a buffer address (**b_phys**) in user space.

hdstrat (lines 121 to 167) performs the I/O and returns. The I/O always completes before **hdstrat** returns.

The block number must be within the partition. A check is made in lines 130 to 135. There is an assumption in this driver that all I/O requests are on 512-byte block boundaries and that the byte count is a multiple of 512. The assumption is safe for file and swap I/O, but not for character I/O. Applications that disobey these constraints won't work correctly. The low 9 bits of the seek address are supplied in **b_resid** (or else ignored), and the byte count is in **b_count**.

The outer "for" loop, line 136, controls multiple block transfers . An attempt is made to do the transfer in one operation. However, the BIOS imposes constraints. Only 127 blocks may be transferred, and the I/O may not cross a physical 64K boundary. These contraints are enforced in lines 138 to 142. The low 16 bits of the physical memory address are examined at line 138. If the 64K boundary is in the middle of a block, no I/O can be done (line 139 to 140). The number of sectors tranferred, **j**, is the minimum of the number left in the 64K bank and the number asked for (lines 141 to 142).

The inner "for" loop (line 143) tries 5 times to do a transfer via a BIOS call. **int13** (line 144) does a BIOS call. If all is well, **int13** returns 0 and transfer is complete (line 147). One error condition is a corrected ECC error; that error is really a warning that is taken as success (lines

148 to 152). An I/O error message is printed on the console, unless this is the first error (lines 155 to 156). The error is recovered by issuing a "reset" and the "recalibrate" command to the BIOS, causing the loop to repeat (at line 143).

If the transfer could not complete successfully after 5 tries, the I/O operation is abandoned (lines 160 to 161). Otherwise the rest of the blocks are read (the loop repeats at line 136).

The number of bytes that could not be transferred is reported to the resident in the **b_resid** field (lines 163 to 164). If all the bytes were not transferred, a console message is printed.

The I/O is reported done at line 165.

iotick charges the current process 3/60ths of a second for this I/O, (line 166). This is to help out the IDRIS scheduler in case there is no system clock. This call would not be needed if the driver road-blocks while waiting for an interrupt.

int13 (lines 179 to 203) computes the seek address given the block number and calls the BIOS interface routine, which is written in the assembler module **dos13.s**. **op** is the operation to perform; **dev** has the minor number; **nblk** is the number of 512-byte blocks to transfer; **blkno** is the block number; **offset** is the 16-bit offset within the paragraph; and **base** is the 16-bit paragraph number.

mappart (lines 205 to 214) returns the address of the in-memory partition table entry given the minor number.

NAME

pchd.c code - hard disk handler

```
/*  IBM PC BIOS ROM HARD DISK HANDLER
 *  copyright (c) 1983 by Whitesmiths, Ltd.
 */
#include <std.h>
#include <res.h>
#include <bio.h>

/*  codes
 */
#define BIOSRS      0
#define BIOSRD      2
#define BIOSWR      3
#define BIOSRC      0x11
#define CYLBITS     0xc0

/*  minor number encoding
 */
#define UNITSHFT    3
#define NHD         2
#define NPART       5
#define NDPART      4

/*  disk tables
 */
typedef struct {
    UCOUNT  hd_open, hd_nspt, hd_ncyl, hd_nspc;
    } HDTAB;

typedef struct {
    UTINY pt_boot, pt_head, pt_sec, pt_cyl;
    UTINY pt_sysid, pt_ehead, pt_esec, pt_ecyl;
    ULONG pt_secoff, pt_nsec;
    } PARTITION;

typedef struct {
    UTINY br_bootstrap[0x1be];
    PARTITION br_part[NDPART];
    UTINY br_hex55, br_hexaa;
    } BOOTREC;

BDEVSW hdbdev {&hdopen, &hdclose, &hdstrat, &hdtab, nmhd};
CDEVSW hdcdev {&hdopen, &hdclose, &hdread, &hdwrite, \
    &nulldev, nmrhd};
LOCAL DEVTAB hdtab {0};
LOCAL HDTAB hd[NHD] {0};
LOCAL PARTITION hdp[NHD][NPART] {0};
LOCAL TEXT nmhd[] {"hdc0 hdc1 hdc2 hdc3 hdc4    hdd0 \
    hdd1 hdd2 hdd3 hdd4"};
LOCAL TEXT nmrhd[]
    {"rhdc0 rhdc1 rhdc2 rhdc3 rhdc4    rhdd0 rhdd1 rhdd2 \
        rhdd3 rhdd4"};

/*  close hd
 */
LOCAL VOID hdclose(dev)
    DEV dev;
    {
    IMPORT HDTAB hd[];

    --hd[dminor(dev) >> UNITSHFT].hd_open;
    }

/*  open a disk, get partition table
 */
LOCAL VOID hdopen(dev)
    DEV dev;
    {
    IMPORT BLOCK _rootoff;
    IMPORT BYTES dseg;
    IMPORT PARTITION hdp[][];
    IMPORT HDTAB hd[];
    FAST UCOUNT hdnum = dminor(dev) >> UNITSHFT;
    FAST HDTAB *p = &hd[hdnum];
    FAST BOOTREC *pbr;
    BUF *pb;
    COUNT n, m;
    struct {
        UTINY st_sec, st_cyl, st_hdnum, st_head;
        } st;

    if (NHD <= hdnum)
        uerror(ENXIO);
    else if (p->hd_open)
        ++p->hd_open;
    else if (dos13st(hdnum | 0x80, &st))
        uerror(ENXIO);
    else if (dos13(1, BIOSRD, pbr = getaddr(pb = getblk(NODEV)),
        1, 0, hdnum | 0x80, 0, dseg) ||
        pbr->br_hex55 != 0x55 || pbr->br_hexaa != 0xaa)
        {
        putfmt("bad partition table on hard disk %i\n", hdnum);
        brelse(pb);
        uerror(ENXIO);
        }
    else
        {
        p->hd_ncyl = ((st.st_sec & CYLBITS) << 2 | st.st_cyl) + 1;
        p->hd_nspt = (st.st_sec & ~CYLBITS);
        p->hd_nspc = (st.st_head + 1) * p->hd_nspt;
        hdp[hdnum][0].pt_nsec = (ULONG) p->hd_nspc * p->hd_ncyl;
        for (n = 0, m = 1; n < NDPART; ++n)
            if (pbr->br_part[n].pt_nsec)
                {
                movbuf(&pbr->br_part[n], &hdp[hdnum][m], \
                    sizeof(PARTITION));
                hdp[hdnum][m].pt_secoff =+ _rootoff;
                hdp[hdnum][m].pt_nsec =- _rootoff;
                ++m;
                }
        brelse(pb);
        ++p->hd_open;
        }
    }

/*  unstructured read
 */
LOCAL VOID hdread(dev)
    DEV dev;
    {
    IMPORT VOID hdstrat();

    physio(&hdstrat, dev, B_READ);
    }

/*  perform I/O request
 */
LOCAL VOID hdstrat(pb)
    FAST BUF *pb;
    {
    IMPORT PARTITION *mappart();
    FAST BYTES i, n;
    UCOUNT err, j, nerr;

    if (mappart(pb->b_dev)->pt_nsec < (n = pb->b_count \
        >> 9) + pb->b_blkno)
        {
        pb->b_flag =| B_ERROR;
        iodone(pb);
        return;
        }
    for (i = 0; i < n; i =+ j)
        {
        j = (UCOUNT)pb->b_phys;
        if ((127<<9) < j)
            break;
        j = 128 - ((j + 511) >> 9);
        j = minu(n - i, j);
        for (nerr = 0; nerr < 5; ++nerr)
            if (!(err = int13(pb->b_flag & B_READ ? BIOSRD : \
                BIOSWR, pb->b_dev,
                j, pb->b_blkno + i,
                (BYTES)pb->b_phys & 017, (BYTES)(pb->b_phys >> 4) \
                    + (i << 5))))
                break;
            else if ((err =>> 8) == 0x11)   /* corrected ecc */
                {
                err = 0;
                break;
                }
            else
                {
                if (nerr)
                    deverr(pb, "try #%i: error code %h", nerr + 1, err);
                int13(BIOSRS, pb->b_dev, 0, 0, 0, 0);
                int13(BIOSRC, pb->b_dev, 0, 0, 0, 0);
                }
        if (err)
            break;
        }
    if (pb->b_resid = ((n - i) << 9) + (pb->b_count & 0777))
        deverr(pb, "failed; %i bytes not transferred", \
            pb->b_resid);
    iodone(pb);
    iotick(3);
    }

/*  unstructured write
 */
LOCAL VOID hdwrite(dev)
    DEV dev;
    {
    IMPORT VOID hdstrat();

    physio(&hdstrat, dev, 0);
    }

/*  call bios to do I/O
 */
LOCAL UCOUNT int13(op, dev, nblk, blkno, offset, base)
    UCOUNT op;
    DEV dev;
    UCOUNT nblk, blkno;
    BYTES offset, base;
    {
    IMPORT HDTAB hd[];
    IMPORT PARTITION hdp[][], *mappart();
    FAST UCOUNT hdnum = dminor(dev) >> UNITSHFT;
    FAST HDTAB *p = &hd[hdnum];
    FAST UCOUNT sec;
    UCOUNT head, cyl;
    ULONG lblk;

    lblk = blkno + mappart(dev)->pt_secoff;
    cyl = lblk / p->hd_nspc;
    sec = lblk % p->hd_nspc;
    head = sec / p->hd_nspt;
    sec =% p->hd_nspt;
    ++sec;
    return (dos13(nblk, op, offset, cyl >> 2 & CYLBITS ¦ \
        sec, cyl,
        hdnum ¦ 0x80, head, base));
    }
```

```

/*  map dev minor number to partition table address
 */
LOCAL PARTITION *mappart(dev)
    DEV dev;
    {
    IMPORT PARTITION hdp[][];
    FAST COUNT devm = dminor(dev);

    return (&hdp[devm >> UNITSHFT][devm & ((1<<UNITSHFT)-1)]);
    }
```

NAME

res.h - header file used by all device drivers

res.h is included in all IDRIS C source files. The ordering of **include** files is:

include <std.h>
include <res.h>
<other IDRIS **include** files in alphabetical order>
"local **include** files"

Psuedo types (lines 7 to 19) are:

BLOCK: holds logical block numbers
BUF: structure of a buffer controller
DEV: holds a device code major/minor pair
ERROR: signed, holds an error code
FID: file identification number (small integer)
FVAR: structure of a file variable
INODE: structure of an in-memory inode
INUM: holds an inode number
PID: holds a process id
PROC: structure of a process list entry
UID: holds a user id number
USER: structure of a process table, swapped with the process image
ZLIST: structure of a zombie process

dminor (line 36) is a macro for portably extracting the minor device number from a **DEV** code. **dmajor** (line 37) extracts the major number.

Error codes are defined in lines 46 to 78.

Scheduling priorities are defined in lines 82 to 93. These are the arguments to <u>sleep</u>. Processes sleeping at a negative priority cannot be interrupted. **PITY** and **POTY** are for interactive processes. **PLOCK** is only honored by the <u>nice</u> system call used for locking a process in memory. Use **PBIO** for sleeping on a block I/O.

Signal (software interrupt) numbers are defined in lines 97 to 114.

NAME

res.h code - IDRIS header file

```
/*  HEADER FOR IDRIS SYSTEM
 *  copyright (c) 1979 by Whitesmiths, Ltd.
 */

/*  basics
 */
#define BLOCK   unsigned short
#define BUF     struct buf
#define DEV     unsigned short
#define ERROR   short
#define FID     short
#define FVAR    struct fvar
#define INODE   struct inode
#define INUM    unsigned short
#define PID     short
#define PROC    struct proc
#define UID     char
#define USER    struct user
#define ZLIST   struct zlist

#define I_READ  000400
#define I_WRITE 000200
#define I_EXEC  000100

#define NCRE    01
#define NDEL    02
#define NSYS    04

#define NODEV   0
#define NOSIG   1
#define ROOTINO 1

/*  the I/O device designator
 */
#define AUTOMOUNT 0x80
#define dminor(dev) ((dev) & BYTMASK)
#define dmajor(dev) (((dev) >> 8) & ~AUTOMOUNT)
#define isautomnt(dev) ((dev) & (AUTOMOUNT<<8))
struct {
    UTINY d_minor;
    UTINY d_major;
    };

/*  codes for u_error
 */
#define EPERM   1
#define ENOENT  2
#define ESRCH   3
#define EINTR   4
#define EIO     5
```

```
#define ENXIO   6
#define E2BIG   7
#define ENOEXEC 8
#define EBADF   9
#define ECHILD  10
#define EAGAIN  11
#define ENOMEM  12
#define EACCES  13
#define ENOTBLK 15
#define EBUSY   16
#define EEXIST  17
#define EXDEV   18
#define ENODEV  19
#define ENOTDIR 20
#define EISDIR  21
#define EINVAL  22
#define ENFILE  23
#define EMFILE  24
#define ENOTTY  25
#define ETXTBSY 26
#define EFBIG   27
#define ENOSPC  28
#define ESPIPE  29
#define EROFS   30
#define EMLINK  31
#define EPIPE   32
#define ENOSYS  100
#define EFAULT  106

/*  priorities
 */
#define PLOCK   -20
#define PSWAP   -3
#define PIIO    -2
#define PBIO    -1
#define PITY    0
#define POTY    0
#define PNORM   4
#define PINIT   4
#define PWAIT   10
#define PPIPE   11
#define PSLEEP  20
#define PLOW    20

/*  signals
 */
#define SIGHUP  1
#define SIGINT  2
#define SIGQIT  3
#define SIGINS  4
#define SIGTRC  5
#define SIGRNG  6    /* IOT */
#define SIGDOM  7    /* EMT */
#define SIGFPT  8
```

```
#define SIGKIL  9
#define SIGBUS  10
#define SIGSEG  11
#define SIGSYS  12
#define SIGPIPE 13
#define SIGALRM 14
#define SIGTERM 15
#define SIGSVC  16
#define SIGUSR  17
#define NSIG    18  /* must be last sig + 1 */
```