```
/*
**
**      s a s / f o r m a t / f o r m a t
**
**
**      Standalone format program for XEBEC and ADAPTEC controllers
**      on the Luxor ABC 1600 system.
**
**      Reverse Engineered by hacker@isadora.sypro.se
**
**      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
**      *                                                               *
**      * WARNING: This version not yet complete, differences with     *
**      *          the original does exist.  The goal is to get        *
**      *          equality on the bit level. No guarantees regarding  *
**      *          functionality.                                       *
**      *                                                               *
**      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
**
**      Agenda:
**             1) Weed out the last bits by "unc"-ing the original
**                and a compiled version of this file. Compile without
**                optimization and do not strip the result. Simply use
**                cc format.c -o format.
**                Run diff on the two disassembled files and identify
**.               the offending c line. Fix and repeat until no diffs
**                turn up.
**             2) Fix bugs.
*/

#include <stdio.h>

#define TRUE 1
#define FALSE 0
#define ERROR -1

typedef unsigned char byte;

/*
** macro used to kick the watchdog
*/

#define WATCHDOG {if (*( (char *)0x80007 ));}

/*
** macros used to access the databoard cards
*/

#define DATABOARD(card) (((unsigned)card << 5) | 0x7e000 )
#define INDEX(base,index) (( (byte *)base )[index])

/*
** port numbers on databoard 4105 card
*/

#define SASI_DATA 0
#define SASI_STAT 2
#define SASI_SEL  4
#define SASI_OPS  4
#define SASI_RST  8

/*
```

```
** dir with disk definitions
*/

char diskpar[] = "/sas/format/disktypes";

/*
** controller type, FALSE for XEBEC and TRUE for ADAPTEC
*/

int adaptec = FALSE;

/*
** table of devices that we can format
*/

struct {
        char *prefix;
        int dev;
} devtab[] = {
        "sa", 1,
        "SA", 1,
        0, 0
};

/*
** some variables and strings
*/

int BPT;
int BPB;
char line[100];
char rply[100];

/*
** information about a disk
*/

struct si_info {
        byte cyls_h;
        byte cyls_l;
        byte heads;
        byte reduced_h;
        byte reduced_l;
        byte precomp_h;
        byte precomp_l;
        byte max_ecc;
        byte step_option;
        byte blocks_h;
        byte blocks_l;
        byte blocksize_h;
        byte blocksize_l;
        byte adaptec;
};

/*
** layout of init record
*/

union initrecord {
        struct {
                byte bootrec[512];
                byte sysfptr[512];
```

```
                struct {
                        char name[16];
                        struct si_info info;
                        byte dummy[512-16-sizeof(struct si_info)-1];
                        byte checksum;
                } drvpar;
                byte vutil[512];
        } parts;
        byte whole[2048];
} buff, tbuff;

/*
** buffers and variables
*/

byte fmtb[1024];

int dev;
int lu;
int devunit;

byte sacmd[6];

byte sasense[6];

struct si_info *tp;

byte msel[4+8+10];
int nbadtrk;
int badtrk[100];
byte fmtbuf[16384];
```

```
/*
**      f o r m a t
*/

main ( argc, argv )
int argc;
char *argv[];

{
        int ix;
        int ac;
        char *p1;
        char *p2;
        char *p3;

        setbuf ( stdout, (char *)0 );   /* stdout is unbuffered */

        printf ( "Disk format program\n" );

        ac = argc;
        do {
                do {
                        if ( ac < 2 ) {
                                printf ( "Enter device: " );
                                gets ( line );
                                if ( *line == '\0' ) {
                                        printf ( "Format aborted.\n" );
                                        exit ( 1 );
                                }
                                p3 = line;
                        } else {
                                p3 = argv[1];
                                argv++;
                                argc++;
                        }
                        p1 = p2 = p3;
                        while ( *p1 ) {
                                if ( *p1++ == '/' ) {
                                        p2 = p1;
                                }
                        }
                        if ( ( ix = getdev ( p3 ) ) == ERROR ) {
                                printf ( "Cannot format device %s.\n", p3 );
                                dev = 0;
                                if ( ac != 0 ) {
                                        goto cont;
                                }
                        }
                } while ( dev == 0 );

                if ( ac < 2 ) {
                        printf (
                          "Format device %s, unit %d - OK? ",
                          devtab[ix].prefix, devunit
                        );
                        gets ( rply );
                        if ( *rply != 'y' && *rply != 'Y' ) {
                                printf ( "Format aborted.\n" );
                                exit ( 1 );
                        }
                }
```

```
                switch ( dev ) {
                case 1:
                        fmtsi ( devunit );
                        break;
                }
cont:;
        } while ( argc > 1 );
        exit ( 0 );
}
```

```
/*
** getdev: lookup device name in table
**
**  The format of a device name is  aa(nn,nn) but this code
**  accepts anything that looks like aa?nn where ? is any
**  character.
*/

int
getdev ( s )
char *s;

{
        int i = 0;

        while ( TRUE ) {
                if ( devtab[i].prefix == (char *)0 ) {
                        break;
                }
                if ( equal ( s, devtab[i].prefix ) ) {
                        dev = devtab[i].dev;
                        s += 3;
                        devunit = atoi ( s );
                        return i;
                }
                i++;
        }
        return ERROR;
}

/*
** equal: compare the two first characters of two strings
*/

int
equal ( s1, s2 )
char *s1;
char *s2;

{
        if ( s2[0] != s1[0] ) {
                return FALSE;
        }
        return ( s2[1] == s1[1] ) ? TRUE : FALSE;
}
```

```
/*
** fmtsi: format the si device
*/

fmtsi ( unit )
int unit;

{
        int address;
        int card;
        int interleave;
        int format_skip;
        int status;
        int i;

        /*
        ** get interleave factor
        */

        printf ( "Enter interleave factor (default 4): " );
        gets ( line );
        if ( (unsigned)(line[0] - '0') >= 10 ) {
                line[0] = '4';
                line[1] = '\0';
        }
        interleave = atoi ( line );

        /*
        ** calculate card address and SASI bus address
        **
        ** This calculation is bogous! Only the first card (addr 0)
        ** can be addressed. This gives:
        **      disk 0  -->  controller 0 - LUN 0       OK
        **      disk 1  -->  controller 0 - LUN 1       BAD
        **      disk 2  -->  controller 0 - LUN 0       VERY BAD
        **      disk 3  -->  controller 0 - LUN 1       BAD
        **      disk 4  -->  controller 0 - LUN 0       VERY BAD
        **      disk 5  -->  controller 0 - LUN 1       BAD
        **      disk 6  -->  controller 0 - LUN 0       VERY BAD
        **      disk 7  -->  controller 0 - LUN 1       BAD
        */

        card = ( (unit >> 3) & 0xF ) | 0x20;    /* WRONG */
        address = unit & 0x7;                   /* WRONG */

        /*
        ** reset SASI
        */

        INDEX(DATABOARD(card),SASI_RST) = 0;

        /*
        ** check if device is online
        */

        if ( INDEX(DATABOARD(card),SASI_STAT) == 0xFF ) {
                printf ( "SA off line!\n" );
                return;
        }

        reset ( card );
```

```
        if ( getpar ( -1 ) == 0 ) {
                return 0;
        }

        /*
        ** reset SASI
        */

        INDEX(DATABOARD(card),SASI_RST) = 0;

        /*
        ** ask if we should skip format phase
        */

        printf ( "Skip format? " );
        gets ( line );
        format_skip = ( line[0] == 'Y' || line[0] == 'y' ) ? 1 : 0;

        /*
        ** ask if we have the ADAPTEC controller
        */

        printf ( "Is controller of type 'ADAPTEC'? " );
        gets ( line );
        adaptec = ( line[0] == 'Y' || line[0] == 'y' ) ? 1 : 0;

        sacmd[1] = ( address & 1 ) << 4;
        sacmd[2] = sacmd[3] = 0;

        if ( adaptec ) {
                msel[21] = tp->step_option;
                tp->step_option = 0;

                if ( !format_skip ) {
                        i = badadap ();
                }

                printf ( "Set ADAPTEC parameters, " );

                if ( format_skip ) {
                        goto f_skip;
                }

                /*
                ** mode select
                */

                sacmd[0] = 0x15;        /* MODE SELECT */
                sacmd[4] = 22;          /* number of bytes */
                msel[3] = 8;            /* EXTENT DESCR. LIST SIZE */
                msel[10] = tp->blocksize_h;
                msel[11] = tp->blocksize_l;
                msel[12] = 1;           /* list format code */
                msel[13] = tp->cyls_h;
                msel[14] = tp->cyls_l;
                msel[15] = tp->heads;
                msel[16] = tp->reduced_h;
                msel[17] = tp->reduced_l;
                msel[18] = tp->precomp_h;
                msel[19] = tp->precomp_l;
```

```
                if ( sifl ( card, address, sacmd, msel ) ) {
                        goto sense;
                }

                /*
                ** format
                */

                if ( i > 0 ) {
                        sacmd[1] |= 0x1C;  /* defect list, data 6C */
                        printf ( "Format with defect list, " );
                } else {
                        printf ( "Format, " );
                }
                sacmd[4] = interleave;
                sacmd[0] = 4;           /* FORMAT UNIT */
                if ( sifl ( card, address, sacmd, fmtb ) ) {
                        goto sense;
                }
                sacmd[1] = ( address & 1 ) << 4;
        } else {

                /*
                ** mode select
                */

                printf ( "Set parameters, " );
                sacmd[5] = tp->step_option;
                sasense[5] = tp->step_option;
                sasense[0] = 0xC;
                if ( sifl ( card, address, sasense, tp ) ) {
                        goto sense;
                }
                if ( format_skip ) {
                        goto f_skip;
                }

                /*
                ** recalibrate
                */

                printf ( "Recalibrate, " );
                sacmd[0] = 1;           /* REZERO UNIT */
                if ( sifl ( card, address, sacmd, 0 ) ) {
                        goto sense;
                }

                /*
                ** format
                */

                printf ( "Format, " );
                sacmd[4] = interleave;
                sacmd[0] = 4;           /* FORMAT UNIT */
                if ( sifl ( card, address, sacmd, 0 ) ) {
                        goto sense;
                }
        }
        sacmd[0] = 0;           /* TEST UNIT READY */
f_skip:
```

```
        /*
        ** readcheck
        */

        if ( format_skip ) {
                printf ( "OK\nSkip readcheck? " );
                gets ( line );
                if ( line[0] != 'Y' && line[0] != 'y' ) {
                        goto r_check;
                }
        } else {
r_check:
                printf ( "Readcheck\n" );
                readcheck ( card, address, interleave, format_skip ? 0 : 1 );
        }

        /*
        ** write init record
        */

        if ( format_skip ) {
                printf ( "Skip update of init record? " );
                gets ( line );
                if ( line[0] != 'Y' && line[0] != 'y' ) {
                        format_skip = TRUE;
                }
        }
        if ( adaptec ) {
                buff.parts.drvpar.info.adaptec |= 1;
        }
        printf ( "Update init record, " );
        ccsum ();
        status = siio ( 10, 0, 2048 / BPB, buff.whole, card, address );
        if ( status ) {
                goto error;
        }

        /*
        ** readcheck init record
        */

        printf ( "Check read init record, " );
        status = siio ( 8, 0, 2048 / BPB, tbuff.whole, card, address );
        if ( status ) {
                goto error;
        }
        for ( i = 0; i < 2048; i++ ) {
                if ( buff.whole[i] != tbuff.whole[i] ) {
                        printf ( "ERROR: Invalid init record!\n" );
                        exit ( 1 );
                }
                WATCHDOG;
        }
        printf ( "OK\n" );
        exit ( 0 );

        /*
        ** get status from unit
        */

sense:
        sasense[0] = 3;         /* REQUEST SENSE */
```

```
        if ( sifl ( card, address, sasense, sasense ) ) {
                sasense[0] = 0x3F;
        }
        status = sasense[0];

        /*
        ** display error code
        */

error:
        printf ( "ERROR: stat=0x%x\n", status & 0xFF );
        exit ( 1 );
}
```

```
/*
** sifl: si low level io
*/

int
sifl ( card, address, cmdbuf, buf )
register int card;
int address;
register char *cmdbuf;
register char *buf;

{
        register int status = 0;

        /*
        ** important bits of SASI_STAT port on 4105 card:
        **
        ** bit 3        I / O*
        ** bit 2        BSY
        ** bit 1        C* / D
        ** bit 0        REQ
        */

        card = DATABOARD(card);

        /*
        ** SELECTION PHASE
        **
        ** Whats wrong here: Asserts only one hardcoded target (id 0 = bit 0)
        **                   and does not include its own id (id 7 = bit 7)
        ** This is related to the bogous address calculation in the beginning
        ** of fmtsi() above.
        */

        INDEX(card,SASI_DATA) = 1;      /* WRONG */
        INDEX(card,SASI_SEL) = 1;

        /*
        ** INFORMATION TRANSFER PHASES
        */

        while ( TRUE ) {

                /*
                ** let controller guide us through the different phases
                */

                switch ( INDEX(card,SASI_STAT) & 0xF ) {

                        /*
                        ** DATA IN PHASE
                        */
                case 0x7:                       /* 0111 - data from disk */
                        *buf++ = INDEX(card,SASI_DATA);
                        break;

                        /*
                        ** DATA OUT PHASE
                        */
                case 0xF:                       /* 1111 - data to disk */
                        INDEX(card,SASI_DATA) = *buf++;
```

```
                break;

                /*
                ** COMMAND PHASE
                */
        case 0xD:                       /* 1101 - command to disk */
                INDEX(card,SASI_DATA) = *cmdbuf++;
                break;

                /*
                ** STATUS PHASE
                */
        case 0x5:                       /* 0101 - status from disk */

                /*
                ** status byte layout:
                **
                ** bit 3        busy
                ** bit 2        equal
                ** bit 1        check
                */

                status = INDEX(card,SASI_DATA) & 0x2;
                while ( INDEX(card,SASI_STAT) & 0x04 ) {
                        /* VOID */
                }
                return status;

        /* The following cases not in jump table        */
        /*      case 0x0:                       /* 0000 */
        /*      case 0x1:                       /* 0001 */
        /*      case 0x2:                       /* 0010 */
        /*      case 0x3:                       /* 0011 */
        /*      case 0x4:                       /* 0100 */

        /* The following cases present in jump table    */
                case 0x6:                       /* 0110 */
                case 0x8:                       /* 1000 */
                case 0x9:                       /* 1001 */
                case 0xA:                       /* 1010 */
                case 0xB:                       /* 1011 */
                case 0xC:                       /* 1100 */
                case 0xE:                       /* 1110 */

                default:
                        WATCHDOG;
                        break;
                }
        }
}
```

```
/*
** reset: reset si card
*/

reset ( card )
register int card;

{
        register int st;

        /*
        ** if si bus is busy then issue reset signal
        */

        st = INDEX(DATABOARD(card),SASI_STAT);
        if ( st & 0x04 ) {
                INDEX(DATABOARD(card),SASI_RST) = 0;
        }

        /*
        ** wait until bus becomes not busy
        */

        while ( INDEX(DATABOARD(card),SASI_STAT) & 0x04 ) {
                /* VOID */
        }
}
```

```
/*
** siio: si io operations
*/

int
siio ( op, block, n, buffer, card, address )
int op;
int block;
char *buffer;
int card;
int address;

{
        /*
        ** build command buffer
        */

        sacmd[0] = op;
        sacmd[1] = ( ( block >> 16 ) & 0x1F ) | ( ( address & 1 ) << 4 );
        sacmd[2] =   ( block >> 8 ) & 0xFF;
        sacmd[3] =   ( block >> 0 ) & 0xFF;
        sacmd[4] = n;

        /*
        ** send command
        */

        if ( sifl ( card, address, sacmd, buffer ) == 0 ) {
                return 0;
        }

        /*
        ** got bad status, request sense
        */

        sasense[0] = 3;
        if ( sifl ( card, address, sasense, sasense ) ) {
                sasense[0] = 0x3F;
        }
        return sasense[0];
}
```

```
/*
** badadap: ask for bad track list (adaptec only)
*/

int
badadap ()

{
        register byte *fp = fmtb + 4;
        register char *p;
        register int ix;
        register int cyl;
        register int head;
        register int bfi;

        for ( ix = 0; ; ix++ ) {
                printf ( "Enter Cyl,Head,Index: " );
                gets ( line );

                p = line;
                if ( *p == '\0' ) {
                        break;
                }

                cyl = atoi ( p );

                while ( *p++ != ',' ) {
                        /* VOID */
                }

                head = atoi ( p );

                while ( *p++ != ',' ) {
                        /* VOID */
                }

                bfi = atoi ( p );

                *fp++ = '\0';
                *fp++ = ( cyl << 8 ) & 0xFF;
                *fp++ = cyl & 0xFF;
                *fp++ = head & 0xFF;
                *fp++ = ( bfi << 24 ) & 0xFF;
                *fp++ = ( bfi << 16 ) & 0xFF;
                *fp++ = ( bfi <<  8 ) & 0xFF;
                *fp++ = bfi & 0xFF;
        }
        if ( ix == 0 ) {
                return 0;
        }

        fp = fmtb;
        *fp++ = 0;
        *fp++ = 0;
        ix *= 8;
        *fp++ = ( ix >> 8 ) & 0xFF;
        *fp++ = ix & 0xFF;
        return ix + 4;
}
```

```
/*
** getpar: get disk parameters from file
*/

int
getpar ( x )
int x;

{
        int fd;
        char fyle[100];
        char dirbuff[16];
        char *p;
        char *pf;

again:
        if ( ( fd = open ( diskpar, 0 ) ) == -1 ) {
                printf ( "Unable to open directory %s!\n", diskpar );
                goto inp;
        }
        printf ( "\nSupported disktypes:\n\n" );
        while ( TRUE ) {
                if ( read ( fd, dirbuff, 16 ) != 16 ) {
                        close ( fd );
                        printf ( "\nEnter type (CR for none above): " );
                        gets ( rply );
                        break;
                }
                if ( dirbuff[2] != '.' ) {
                        printf ( " %s\n", &dirbuff[2] );
                }
        }
        if ( *rply == '\0' ) {
inp:
                if ( !inpar () ) {
                        goto again;
                }
                goto err;
        }

        pf = fyle;

        p = diskpar;
        while ( *p ) {
                *pf = *p;
                p++;
                pf++;
        }
        *pf++ = '/';

        p = rply;
        while ( *pf++ = *p++ ) {
                /* VOID */
        }

        if ( ( fd = open ( fyle, 0 ) ) == -1 ) {
                printf ( "Cannot open parameter file %s", fyle );
                goto err;
        }

        lseek ( fd, 0, 0 );
```

```
        if ( read ( fd, &buff.parts.drvpar, 512 ) != 512 ) {
                close ( fd );
                printf ( "Cannot read parameter file %s", fyle );
                goto err;
        }
        close ( fd );

        tp = &buff.parts.drvpar.info;

        if ( outpar () == 0 ) {
                goto again;
        }
err:
        return 1;
}
```

```
/*
** outpar: display disk parameters
*/

int
outpar ()

{
        BPT = tp->blocks_l + ( tp->blocks_h << 8 );
        if ( BPT == 0 ) {
                BPT = 17;
                tp->blocks_l = BPT & 0xFF;
                tp->blocks_h = ( BPT >> 8 ) & 0xFF;
        }
        BPB = tp->blocksize_l + ( tp->blocksize_h << 8 );
        if ( BPB == 0 ) {
                BPB = 512;
                tp->blocksize_l = BPB & 0xFF;
                tp->blocksize_h = ( BPB >> 8 ) & 0xFF;
        }

        printf ( "\nSelected disk parameters\n" );
        printf (
          "\n Disk type (file name):        %s",
          buff.parts.drvpar.name
        );
        printf (
          "\n Number of cylinders:          %d",
          tp->cyls_l + ( tp->cyls_h << 8 )
        );
        printf (
          "\n Number of heads:              %d",
          tp->heads
        );
        printf (
          "\n Start cyl. for reduced write: %d",
          tp->reduced_l + ( tp->reduced_h << 8 )
        );
        printf (
          "\n Start cyl. for write precomp: %d",
          tp->precomp_l + ( tp->precomp_h << 8 )
        );
        printf (
          "\n Max ecc length:               %d",
          tp->max_ecc
        );
        printf (
          "\n Step option:                  %d",
          tp->step_option
        );
        printf (
          "\n Bytes per block (block size): %d",
          BPB
        );
        printf (
          "\n Blocks per track:             %d",
          BPT
        );

        printf ( "\n\nParameters ok? " );
        gets ( rply );
```

```
        if ( rply[0] == 'y' || rply[0] == 'Y' ) {
                return 1;
        } else {
                return 0;
        }
}
```

```
/*
** inpar: ask for disk parameters
*/

int
inpar ()

{
        int x;

        for ( x = 0; x < 2048; x++ ) {
                buff.whole[x] = 0;
                WATCHDOG;
        }

        tp = &buff.parts.drvpar.info;

        printf ( "Do you want to specify parameters? " );
        gets ( rply );
        if ( rply[0] != 'y' && rply[0] != 'Y' ) {
                printf ( "Format aborted!\n" );
                exit ( 1 );
        }

        printf ( "\n\nEnter disk parameters...\n\n" );

        printf ( "Disk type (file name): " );
        gets ( rply );
        rply[15] = '\0';
        for ( x = 0; x < 16; x++ ) {
                buff.parts.drvpar.name[x] = rply[x];
        }

        printf ( "Number of cylinders: " );
        gets ( rply );
        x = atoi ( rply );
        tp->cyls_l = x;
        tp->cyls_h = x >> 8;

        printf ( "Number of heads: " );
        gets ( rply );
        x = atoi ( rply );
        tp->heads = x;

        printf ( "Start cyl. for reduced write: " );
        gets ( rply );
        x = atoi ( rply );
        tp->reduced_l = x;
        tp->reduced_h = x >> 8;

        printf ( "Start cyl. for write precomp: " );
        gets ( rply );
        x = atoi ( rply );
        tp->precomp_l = x;
        tp->precomp_h = x >> 8;

        printf ( "Max ecc length: " );
        gets ( rply );
        x = atoi ( rply );
        tp->max_ecc = x;
```

```
        printf ( "Step option: " );
        gets ( rply );
        x = atoi ( rply );
        tp->step_option = x;

        printf ( "Bytes per block (block size): " );
        gets ( rply );
        x = atoi ( rply );
        tp->blocksize_l = x;
        tp->blocksize_h = x >> 8;

        printf ( "Blocks per track: " );
        gets ( rply );
        x = atoi ( rply );
        tp->blocks_l = x;
        tp->blocks_h = x >> 8;

        if ( outpar () == 0 ) {
                return 0;
        }

        printf ( "\nDo you want to save parameters in parameter file? " );
        gets ( rply );
        if ( rply[0] == 'y' || rply[0] == 'Y' ) {
                savepar ();
        }
        return 1;
}
```

```
/*
** savepar: save disk parameters
*/

savepar ()

{
        int fd;
        char dummy[10];
        char *pf;
        char *p;
        char fyle[100];

        pf = fyle;
        p = diskpar;
        while ( *p ) {
                *pf = *p;
                p++;
                pf++;
        }

        *pf++ = '/';

        p = buff.parts.drvpar.name;
        while ( *pf++ = *p++ ) {
                /* VOID */
        }

        p = fyle;
        if ( ( fd = creat ( p, 644 ) ) < 0 ) {
                printf ( "Cannot create %s - format aborted!\n", p );
                exit ( 1 );
        }

        ccsum ();

        if ( write ( fd, &buff.parts.drvpar, 512 ) != 512 ) {
                printf ( "Write error - format aborted!\n" );
                exit ( 1 );
        }

        sync ();

        printf ( "Parameter file updated.\n" );
        close ( fd );
}
```

```
/*
** ccsum: calculate checksum of init record
*/

ccsum ()

{
        register byte *p;
        register int i;
        register int sum;

        p = (byte *)&buff.parts.drvpar;
        sum = -1;

        for ( i = 0; i < 512; i++ ) {
                sum ^= *p++;
        }

        buff.parts.drvpar.checksum = sum;
}
```

```
/*
** readcheck: readcheck disk
*/

readcheck ( card, address, interleave, format_skip_flag )
int card;
int address;
int interleave;
int format_skip_flag;

{
        register unsigned int cylinder;
        register unsigned int cyls;
        register unsigned int head;
        register unsigned int heads;
        int i;
        int status;
        int blk;
        int altptr;
        int capacity;

        heads = tp->heads;
        cyls = tp->cyls_l + ( tp->cyls_h << 8 );
        capacity = BPT * cyls * heads;

        if ( !adaptec ) {
                readbad ( capacity, heads, BPT );
        }

        printf ( "Read checking disk, hit any key to abort...\n" );

        for ( cylinder = 0; cylinder < cyls; cylinder++ ) {

                if ( ( cylinder % 25 ) == 0 ) {
                        printf ( " Cyl = %d\r", cylinder );
                }

                for ( head = 0; head < heads; head++ ) {
                        blk = BPT * heads * cylinder;
                        status = siio ( 8, blk, BPT, fmtbuf, card, address );
                        if ( status ) {
                                printf ( "Bad track: stat=0x%x, ", status );
                                printf (
                                  "bn=%ld, cyl=%d, head=%d\n",
                                  blk, cylinder, head
                                );
                                markbad ( blk );
                                goto skip;
                        }

                        /*
                        ** this test is weird!
                        */

                        if ( format_skip_flag ) {
                                if ( ( fmtbuf[BPB * BPT] != 0 )
                                  || ( (fmtbuf-1)[BPB * BPT] != 0x6C ) ) {
                                        printf ( "Sector length error" );
                                        exit ( 1 );
                                }
                        }
```

```
skip:
                        if ( testin () ) {
                                printf ( "Read check aborted manually\n" );
                                printf ( "CONTINUE? " );
                                gets ( line );
                                if ( line[0] != 'Y' && line[0] != 'y' ) {
                                        exit ( 1 );
                                }
                                break;
                        }
                }
        }

        if ( !adaptec && nbadtrk ) {
                printf ( "There are bad tracks\n" );
                capacity -= BPT * nbadtrk;
                altptr = capacity;
                for ( i = 0; i < nbadtrk; i++ ) {
                        blk = badtrk[i];
                        printf ( "Bad: " );
                        printtrk ( blk, heads, BPT );
                        sacmd[1] = ( ( blk)>>16 )&0x1F ) || ( (address&1 )<<4 );
                        sacmd[2] = ( blk >> 8 ) & 0xFF;
                        sacmd[3] = ( blk >> 0 ) & 0xFF;
                        sacmd[4] = interleave;
                        if ( blk >= capacity ) {
                                printf ( " format as bad track" );
                                sacmd[0] = 7;
                        } else {
                                while ( isbad ( altptr ) ) {
                                        altptr += BPT;
                                }
                                printf ( " alt: " );
                                printtrk ( altptr, heads, BPT );
                                fmtbuf[0] = ( altptr >> 16 ) & 0x1F;
                                fmtbuf[1] = ( altptr >>  8 ) & 0xFF;
                                fmtbuf[2] = ( altptr >>  0 ) & 0xFF;
                                sacmd[0] = 14;
                                altptr += BPT;
                        }
                        if ( !format_skip_flag ) {
                                printf ( "\nFormat with alt. track? " );
                                gets ( line );
                                if ( line[0] != 'y' && line[0] != 'Y' ) {
                                        goto cont;
                                }
                        }
                        if ( sifl ( card, address, sacmd, fmtbuf ) ) {
                                printf ( " failed\n" );
                                goto sense;
                        }
                        printf ( " ok\n" );
cont:;          }
                while ( isbad ( altptr ) ) {
                        altptr += BPT;
                }
                if ( ( BPT * nbadtrk + capacity ) != altptr ) {
                        printf (
                          "Program error: wrong number of alternates\n"
                        );                              /* WRONG ---^ */
                        exit ( 1 );
                }
```

```
        }
        printf ( "Done - usable disk size: %ldk\n", capacity / 2 );
        return;

sense:
        sasense[0] = 3;
        if ( sifl ( card, address, sasense, sasense ) ) {
                sasense[0] = 0x3F;
        }
        printf ( "ERROR: 0x%x\n", sasense[0] );
        exit ( 1 );
}
```

```
/*
** printtrk: print track info
*/

printtrk ( blk, heads, bpt )
int blk;
int heads;
int bpt;

{
        int cyl;
        int head;

        cyl = ( bpt * heads ) / blk;
        head = bpt / ( ( bpt * heads ) % blk );
        printf ( "cyl=%d, head=%d (bn=%ld)", cyl, head, blk );
}
```

```
/*
** isbad: see if a block has been marked bad
*/

int
isbad ( blk )
int blk;

{
        register int i;

        for ( i = 0; i < nbadtrk; i++ ) {
                if ( badtrk[i] == blk ) {
                        return 1;
                }
        }
        return 0;
}

/*
** markbad: mark a block as bad
*/

markbad ( blk )
int blk;

{
        if ( isbad ( blk ) ) {
                return;
        }

        if ( nbadtrk < 99 ) {
                badtrk[nbadtrk] = blk;
                nbadtrk++;
        } else {
                printf ( "Too many bad tracks - giving up\n" );
                exit ( 1 );
        }
}
```

```
/*
** readbad: read bad track info from user or file
*/

readbad ( capacity, heads, bpt )
int capacity;
int heads;
int bpt;

{
        int blk;
        register int cyl;
        register int head;
        register FILE *f;
        register char *p;

        f = (FILE *)0;

        printf ( "Enter bad track list or filename\n" );

        while ( TRUE ) {
                *line = '\0';
                if ( !f ) {
                        printf ( "cyl,head: " );
                        gets ( line );
                } else {
                        p = line;
                        while ( ( *p = getc ( f ) ) > ' ' ) {
                                p++;
                        }
                        *p = '\0';
                }

                if ( *line == '\0' ) {
                        break;
                }

                if ( *line == ':' ) {
                        if ( f ) {
                                printf ( "%s\n", line );
                        }
                        continue;
                }
                if ( *line > '9' || *line < '0' ) {
                        if ( f ) {
                                fclose ( f );
                        }
                        if ( ( f = fopen ( line, "r" ) ) == (FILE *)0 ) {
                                printf ( "Unable to open %s\n", line );
                        }
                        continue;
                }

                cyl = atoi ( line );

                p = line;
                while ( *p++ != ',' ) {
                        /* VOID */
                }

                head = atoi ( p );
```

```
                blk = bpt * ( ( heads * cyl ) + head );
                if ( head > heads || blk < 0 || blk > capacity ) {
                        printf ( "Cyl=%d, head=%d outside disc\n", cyl, head );
                        if ( f ) {
                                exit ( 1 );
                        } else {
                                break;
                        }
                }
        }
        if ( f ) {
                fclose ( f );
        }
}
```

```
/*
** testin: see if a character is available on the console
**
** The hardware addresses below should point to the console
** serial chip (a Z80 DART.) The addresses used below contains
** no hardware in the ABC 1600 design. Addresserna 0x1FF200 och
** 0x1FF202 borde anv{ndas.
*/

int
testin ()

{
        /*
        ** check the console sio for any key
        **
        ** The numbers below are NOT addresses to valid i/o devices
        ** on an ABC 1600
        */

        if ( ( *((byte *)0x7F6F9) & 0x01 ) == 0 ) {     /* WRONG */
                return 0;
        }
        if ( *((byte *)0x7F6FB) );                      /* WRONG */
        return 1;
}
```