

```

/*
 * Commodore M-series Z8001 boot ROM.
 * This code can run interactively and
 * has commands to boot from disc or serial
 * port or if 'AUTOBOOT' is defined, it
 * will automatically boot.
 */
#include "rom.h"

#define putnhex(i)      puthex((i), 0) /* output nibble in hex*/
#define putbhex(i)     puthex((i), 4) /* output byte in hex */
#define putihex(i)     puthex((i), 12) /* output word in hex */

#define NROW          10                /* Rows to dump */
#define NCOL          14                /* Columns in a row */

#if AUTOBOOT
char *about = "(hd) coherent";
#else
char *about;
#endif

char *gets();
addr_t getaddr();
char *boot();
extern unsigned seg;

char line[100];
static unsigned seqno;
static unsigned nread;
int wdlflag;
extern int typeno; /* drive type for WD driver */
struct romconf romconf; /* Configuration information */
extern char usrabrt; /* <> 0 => user aborted autoboot */
extern char harderr; /* <> 0 => power-up hardware error */
char tryaboot = '\0'; /* <> 0 => auto-boot in progress */
extern char multmic(); /* test multi-micro bit */
extern char kill(); /* stop processor if fatal error */
extern char ramtest(); /* entry point for burn-in diags. */

#ifndef DDT
/*
 * Messages relating to trap types.
 */
char *traps[] = {
    "unrecognisable trap type",
    "extended instruction trap",
    "privileged instruction trap",
    "system call trap",
    "segmentation violation trap",
    "non-maskable interrupt",
    "non-vectorized interrupt"
};
#endif

char *msgs[] = {
    "Command          Function\n",
    "-----\n",
    "(fd)file          boot <file> from floppy disk\n",
    "(hd)file          boot <file> from the SASI hard disk\n",
    "(wd)file          boot <file> from the WD hard disk\n",
    "m                display RAM configuration\n",

```

```

    "FS<unit> <int> format SASI disk <unit> with interleave <int>\n",
    "FF<unit>          format Commodore Floppy disk <unit>\n",
    "D <unit>          run SASI drive diagnostic on <unit>\n",
#ifdef DDT
    "d                run debugger\n",
#else
    "e <address>      examine memory starting at <address>\n",
#endif
    "o                toggle hi-res\n",
    "c                toggle console monitor\n",
    "P <n>            set hard disk parameters to type <n>\n",
    "\n",
    "Note: All numeric input is in HEX !!\n\n"
};

#define NTYPE      7

main(low, hi)
unsigned low, hi;
{
    memdiag();
    port_test(); /* check I/O chip registers */
    configure(&romconf, low, hi);
    if (multmic()) /* if not doing a board burn-in */
        command();
    if (harderr) kill(); /* stop if err on board burn-in */
    ramtest(); /* since board burn-in, restart diagnostics */
}

/*
 * Process commands.
 * Handle automatic commands here as well.
 */
command()
{
    short  retry;
    register int f;
    register char *r;

    if ((aboot != NULL) && /* if auto boot was specified and */
        (usrabrt == 0) && /* the user didn't abort autoboot and */
        (harderr == 0)) /* no power-up hardware errors were detected
    {
        tryaboot = '\077'; /* show auto-boot in progress */
        for ( retry = 0; retry < 8; retry++)
        {
            if (retry == 7) /* issue error messages if fails */
                tryaboot = '\0'; /* on last try */
            r = boot(aboot);
        }
        if (r != NULL)
        {
            puts(r);
            putchar('\n');
        }
    }
    puts("\nCommodore C900 monitor (type ? for commands)\n");
    for (;;) {
        f = 0;
        puts("? ");
        gets(line);
        switch (line[0]) {
            case '\0':

```

```

        break;

    case '(':
        for ( retry = 0;retry< 8;retry++)
            r = boot(line);
            if (r != NULL) {
                puts(r);
                putchar('\n');
            }
            break;

#ifdef DDT
    case 'e':
        examine(&line[1]);
        break;

#endif

    case 'm':
        memory();
        break;

    case 'r':
        run(&line[1], 1);
        break;

    case 'F':
        format(&line[1]);
        break;

    case 'D':
        diagnostic(&line[1]);
        break;

    case 'P':
        parameters(&line[1]);
        break;

    case '?':
        usage();
        break;

#ifdef DDT
    case 'd':
        scall();
        break;

#endif

    case 'o':
        if (romconf.has_hires == '\000')
            romconf.has_hires = '\002';
        else
            romconf.has_hires = '\000';
        /*
        putmsg("\nhires = ");
        putbhex(romconf.has_hires);
        putchar('\n');*/
        break;

    case 'c':
        if (romconf.has_hires == '\001')
            romconf.has_hires = '\002';
        else
            romconf.has_hires = '\001';
        /*

```

```
        putchar('\n');*/
        break;
```

```
    default:
        putchar(line[0]);
        puts(": bad command\n");
    }
}
```

```
/*
 * Start running at the specified address.
 */
```

```
run(cp, f)
char *cp;
int f;
{
    addr_t runaddr;

    runaddr = getaddr(cp, RUNADDR);
    puts("Jumping to address ");
    puthex((int)(runaddr>>24), 2);
    putchar('|');
    puthex((int)(runaddr&0xffff), 4);
    puts(" in segmented mode\n");
    jump(runaddr, f);
}
```

```
/*
 * Read an address in hex and return it.
 * If no address is given, use the default
 * value in 'def'
 */
```

```
addr_t
getaddr(cp, def)
register char *cp;
addr_t def;
{
    register int n;

    while (*cp==' ' || *cp=='\t')
        cp++;
    if (hexdigit(*cp) >= 0) {
        def = 0;
        while ((n = hexdigit(*cp++)) >= 0)
            def = (def<<4) + n;
    }
    return (def);
}
```

```
/*
 * Set parameters in decimal
 * for the hard disc.
 * Used as 'P n' where 'n' indexes
 * the table below.
 */
```

```
struct hdpar {
    char    *hd_type;
    char    hd_param[10];
}
hdpar[] = {
    {"2-head 10MB", 1, 1, 0, 1, 2, 98, 0, 0, 0, 0 },
    {"4-head 10MB", 1, 1, 0, 3, 1, 49, 0, 0, 0, 0 },
    {"4-head 20MB", 1, 1, 0, 3, 2, 98, 0, 0, 0, 0 }.
```

```

parameters(s)
register char *s;
{
    register struct hdpar *hp;
    register int n;

    n = getaddr(s, -1);
    if (n<0 || n>=(sizeof(hdpar)/sizeof(hdpar[0]))) {
        puts("Bad parameter setting\n");
        return;
    }
    hp = &hdpar[n];
    puts("Selecting ");
    puts(hp->hd_type);
    puts(" as hard disc\n");
    hdparams(hp->hd_param);
    typeno = n;
}

```

```

/*
 * Print out memory sizes.
 */
memory()

```

```

{
    register unsigned l, h;

    l = romconf.rom_bram;
    h = romconf.rom_eram;

    puts("RAM base: ");
    puthex(l>>6, 2);
    puts("|");
    puthex(l<<12, 4);
    puts("\n");

    puts("RAM top: ");
    puthex(h>>6, 2);
    puts("|");
    puthex(h<<12, 4);
    puts("\n");
}

```

```

#ifdef DDT

```

```

/*
 * Dump memory.
 */

```

```

examine(cp)
register char *cp;
{
    unsigned s, o;
    register int c;
    register unsigned *bp;
    static addr_t a = 0L;
    static int b[NROW*NCOL];

    a = getaddr(cp, a);
    s = (a>>16)&0xFFFF;
    o = a&0xFFFF;
    ppcopy(a, b, sizeof(b));
    bp = &b[0];
    for (;;) {
        if (bp >= &b[NROW*NCOL])

```

```

        puthex(s>>8, 2);
        putchar('|');
        puthex(o, 4);
        putchar(' ');
        putchar('=');
        for (c=0; c<NCOL; c++) {
            if (bp >= &b[NROW*NCOL]) {
                putchar('\n');
                goto out;
            }
            putchar(' ');
            puthex(*bp++, 4);
        }
        if ((o + NCOL*2) < o)
            s += 0x100;
        o += NCOL*2;
        putchar('\n');
    }

```

```

out:
    a = (addr_t)s << 16;
    a += o;
    return (0);
}

```

```

#endif

```

```

/*
 * Put out a hexadecimal number.
 */

```

```

puthex(u, n)
register unsigned u;
register int n;
{
    if (--n > 0)
        puthex(u>>4, n);
    u &= 0x000F;
    if (u >= 0xA)
        putchar(u+'A'-0xA); else
        putchar(u+'0');
}

```

```

/*
 * Return hex value of a digit. -1 if nothing.
 */

```

```

hexdigit(c)
register int c;
{
    if (c>='0' && c<='9')
        return (c-'0');
    if (c>='A' && c<='Z')
        return (c-'A'+0xA);
    if (c>='a' && c<='z')
        return (c-'a'+0xa);
    return (-1);
}

```

```

/*
 * Put out a string to the console.
 * No added newlines.
 */

```

```

puts(s)
register char *s;
{
    while (*s != '\0')
        putchar(*s++);
}

```

```

/*
 * Get a string from the console
 * No length checking, strip the
 * trailing newline.
 * Arranges to do erase and kill editing.
 */
char *
gets(as)
char *as;
{
    register char *s;
    register int c;

again:
    s = as;
    while ((c = getchar()) != '\n') {
        if (c == '\b') {
            if (s <= as)
                continue;

            s--;
            putchar(' ');
            putchar(c);
            continue;
        }
        if (c == '@') {
            putchar('\n');
            goto again;
        }
        *s++ = c;
    }
    *s = '\0';
    return (s);
}
#endif
/*
 * Handle hardware traps.
 * This is only for traps that happend
 * during the ROM (none of which should happen).
 */
trap(r0, r1, r2, r3, r4, r5, type)
/* ARGSUSED */
register int type;
{
    if (type >= NTYPE)
        type = 0;
    puts("ROM: ");
    puts(traps[type]);
    putchar('\n');
    jump(0L, 1);
}
#endif
/*
 * Print out a simple command summary
 */
usage()
{
    register char **cp;

    puts("\nCommodore C900 Monitor");
    puts(VERS);
    for (cp = msgs; cp < &msgs[(sizeof(msgs)/sizeof(msgs[0]))]; cp++)
        puts(*cp);
}

```

```

/*
 * Format the specified SASI disk (via a controller command).
 */
format(cp)
register char *cp;
{
    register int n, unit, interleave;

    unit = interleave = -1;
    while (*cp == ' ' || *cp == '\t')
        cp++;
    if (hexdigit(*cp) >= 0) {
        unit = 0;
        while ((n = hexdigit(*cp++)) >= 0)
            unit = (unit<<4) + n;
    }
    while (*cp == ' ' || *cp == '\t')
        cp++;
    if (hexdigit(*cp) >= 0) {
        interleave = 0;
        while ((n = hexdigit(*cp++)) >= 0)
            interleave = (interleave<<4) + n;
    }
    if (unit == 0) {
        puts("Do you really want to format SASI hard disk #0 ?");
        switch (getchar()) {
            case 'y':
            case 'Y':
                break;
            default:
                return;
        }
    }
    hdload();
    hdformat(unit, interleave); /* params will be checked */
}

/*
 * Perform a simple diagnostic on the given SASI disk device.
 */
diagnostic(cp)
char *cp;
{
    hdload();
    hddiagnostic((int)getaddr(cp, -1));
}

```



Switch (\*cp) {  
 Case 'y':  
 } all SASI FMT  
 Break;  
 Case 'Y':