CRT.S 5/31/85

```
/ Run-time startup for the Commodore boot ROM - 1K clicks, segmented mode
/ This code does all hardware and much MMU initialisation
/ for the Coherent system that it ultimately boots.
/ March, 1984

rxrdy   =       0       / tty status /
sccbase = 0x0100        / z-scc base address            /
wr0b            = 0x01
wr8b            = 0x11
eschar  =       0x1b    /esc to abort load/send/

SP = rr14                       / Segmented stack pointer
STACK = 0x3F00                  / stack segment
MMU     =       0x00FC                  / Access MMU # 1 (#2 is unused)
MMU1 = 0x00FC
MMU12= 0x00F8
TFCW    = 0xC000                / Trap FCW - segmented, system mode, di VI NV
#ifndef RAMBASE
RAMBASE = 0x02000000            / Base of RAM physical
#endif
ROM     = 0x8000                / Segment of ROM
HIRES = 0x3E00                  / Hi-res screen RAM segment
LORES = 0x3700                  / Lo-res screen RAM segment
ES      = 0x3D3D                / Must be OS-1
OS      = 0x3E3E                / Overlay (driver,buffer,clist,inode) seg
SS      = 0x3F3F
        .globl  SS
SS=0x3F3F
SIGSYS =         7              / Taken from <msig.h>
SIGBPT =        10
SIGSEG =        11
SIGEPA =        12
SIGPRV =        13
SIGNVI =        14
SIGNMI =        15

/ This is location 0 of the data which has a pointer to the ROM
/ configuration table.

        .globl  romconf_, seg_
        .shrd
        .long   romconf_
        .shri

        .globl  ramdone
        .globl  main_, trap_
        .globl  in_, inb_, out_, outb_
        .globl  etext_, end_, edata_

        .shri
/ Power-up reset processor status (4 words)
/ This is also the PSA for ROM only.
start:
        jr      1f                      / dummy word
        .word   0xC000                  / FCW -- Segmented/system mode
        .word   ROM                     / PC segment
        .word   1f                      / PC offset

        .word   0, TFCW, ROM, tepa      / extended instruction
        .word   0, TFCW, ROM, tprv      / privileged instruction
        .word   0, TFCW, ROM, tsys      / System call instruction
        .word   0, TFCW, ROM, tseg      / Segmentation violation
        .word   0, TFCW, ROM, tnmi      / Non-maskable interrupt
```

```
        / No vectored interrupts needed here

/ Possibly put code here to do rudimentary diagnostics on
/ the CPU and halt if any errors found.

/ Before we turn on segmentation
/ we are not allowed to write any memory as we
/ are running out of the ROM.
1:

        ld      r0, $0x80               / Enable MMU1, no translation
        soutb   MMU1+0x0000, rl0        /
        soutb   MMU12+0x1100, rl0       / reset violation type on MMUs
        soutb   MMU12+0x1300, rl0       / reset SWW flag
        soutb   MMU12+0x1400, rl0       / reset FATL flag

        ld      r0, $0x9e00             / Refresh enable
        ldctl   REFRESH, r0

        ldar    rr0, start              / Set up address of...
        ldctl   PSAPSEG, r0             /  PSA segment and ...
        ldctl   PSAPOFF, r1             /  segment

        sub     r2, r2                  / Point all segments ...
        ld      r3, $64                 /  to themselves in
        ldl     rr0, $0x0000FF00        /  physical memory
        soutb   MMU12+0x2000, rl0       / clear descriptor counter
        soutb   MMU12+0x0100, rl2       / Segment number
0:
        soutb   MMU12+0x0F00, rh0       / Load all descriptors ...
        soutb   MMU12+0x0F00, rl0       /  from 0-63 to have ...
        soutb   MMU12+0x0F00, rh1       /  proper base, and ...
        soutb   MMU12+0x0F00, rl1       /  umlimited length
        incb    rh0                     / Next 64K page
        djnz    r3, 0b

        ldb     rl0, $0xC0              / MMU#1 (code), MSEN, TRNS
        soutb   MMU1+0x0000, rl0        / Write mode register
        ld      r0, $1

        jp      ramtest                 / Do power-up RAM test
ramdone:

/ Size and clear all of RAM
/ r12 gets saddr_t of beginning of memory, and
/ r13 gets first unusable saddr_t above RAM
        ldl     rr2, $RAMBASE           / Address of base of RAM
        ldb     rl4, rh3                / Compute saddr_t of ...
        ldb     rh4, rh2                /  base of RAM
        ld      rl2, r4                 /
        srl     rl2, $2                 / base of RAM saddr_t

0:
        ld      (rr2), $0xAAAA          / Set alternating pattern in memory
        cp      (rr2), $0xAAAA          / Test for existence
        jr      ne, 0f                  / Branch if end of (or bad) memory
        ld      (rr2), $0x5555          / Second alternating pattern
        cp      (rr2), $0x5555          / Test for existence
        jr      ne, 0f

        ld      r1, $511                / Size of click in words - 1
        ldl     rr4, rr2                / copy of RAM pointer
        inc     r5, $2                  / next word
```

```
        ldir    (rr4), (rr2), rl        / Clear it
        inc     r3, $2                  / r2 is trailing by one word
        jr      ne, 0b                  / branch if same segment
        incb    rh2                     / go to next segment (stupid z8000)
        jr      0b                      / Do next click

0:
        ldb     rl4, rh3                / First click that is inaccessible
        ldb     rh4, rh2                / represented as a saddr_t
        ld      rl3, r4
        srl     rl3, $2                 / hardware to software click conv.

/ Copy initialised data to top of memory.
/ Do not copy bss but leave room for it.
/ It is zeroed by the previous memory clearing code.
/ Map segments 1 and 63 onto data in RAM, using 63 as the stack seg.
        ld      r0, $end_+1023          / Size of data rounded up
        srl     r0, $10                 / to clicks
        ld      r1, r13                 / End of memory
        sub     r1, r0                  / Start of data segment
        sll     r1, $2                  / Convert to MMU address type

        ld      r0, $0x3F               / Segment 63 (large number)
        soutb   MMU1+0x0100, rl0        / Select segment #
        soutb   MMU1+0x0800, rh1        / Load segment base...
        soutb   MMU1+0x0800, rl1        / saddr_t only

        ld      r0, $edata_             / Number of bytes to copy
        ld      r2, $ROM                / seg # of rr2
        ld      r3, $etext_+1023        / Offset of rr2 ...
        and     r3, $0xFC00             / rounded to 1024 byte click
        ld      r4, $0x3F00             / Segment 63
        sub     r5, r5                  / offset 0

        ldirb   (rr4), (rr2), r0        / Copy from ROM to upper RAM

        ld      r0, $0x01               / Map segment #1...
        soutb   MMU1+0x0100, rl0        / over top of ROM data ...
        soutb   MMU1+0x0800, rh1        / that is in ...
        soutb   MMU1+0x0800, rl1        / RAM

/ test to see if any video boards present

        ldctl   r0,NSPOFF               / Get indicator from register
        cpb     rl0,$1                  / If hi-res
        jr      ne, 0f
        ldb     romconf_+15,$1          /       set "has_hires"
        jr      1f
0:                                      / else
        clrb    romconf_+15             /       clear "has_hires"
1:
        cpb     rl0,$2                  / If lo-res
        jr      ne, 0f
        ldb     romconf_+16,$1          /       set "has_lores"
        jr      1f
0:                                      / else
        clrb    romconf_+16             /       clear "has_lores"
1:
        testb   rh0                     / If hardware error was detected,
        jr      z, 0f
        ldb     harderr_,$0xFF          / Set hardware error flag
0:
        ldctl   rl,NSPSEG               / Store cursor position
```

```
        ldl     curpos,rr0

/       see if user typed an <ESC> key during RAM test to abort autoboot

        ldb     rh1,$10                 / Set loop count
0:      inb     rl1,sccbase+wr0b        / get Rx buffer status
        bitb    rl1,$rxrdy              / If buffer is empty,
        jr      z, 1f                   /       exit loop
        inb     rl1,sccbase+wr8b        / else get character
        resb    rl1,$7                  / clear high order bit of character
        cpb     rl1,$eschar             / if ESC detected
        jr      eq, 0f                  /       set flag
                                                / else
        dbjnz   rh1, 0b                 /       loop up to 10 times
        jr      1f
0:
        ldb     usrabrt_,$0xFF  / Set user abort flag
1:

/ initialise stack pointer to
/ to the data part.
        ld      r14, $STACK
        ld      r15, $estack_

        ld      seg_, $ROM              / Fixed ROM segment number
        push    (SP), r13               / coretop
        push    (SP), r12               / corebot
        call    main_
        jr      start

inb_:
        ld      r1, SP(4)
        inb     rl1, (r1)
        subb    rh1, rh1
        ret

in_:
        ld      r1, SP(4)
        in      r1, (r1)
        ret

outb_:
        ld      r1, SP(4)
        ld      r0, SP(6)
        outb    (r1), rl0
        ret

out_:
        ld      r1, SP(4)
        ld      r0, SP(6)
        out     (r1), r0
        ret

/ Jump to an address.
/ flag is ignored.....
/ jump(addr, flag)
/ long addr;
/ int flag;

        .globl  jump_

jump_:
        ldl     rr2, SP(4)              / Address
```

```
/ Load segment 'n' of MMU for code and
/ load segment 'n+1' of MMU to data
/       segload(n, code, data)
/       unsigned n;
/       saddr_t code, data;
        .globl  segload_

segload_:
        ld      r0, SP(4)               / segment #
        ld      r1, SP(6)               / code bias
        ld      r2, SP(8)               / data bias

        ld      r3, $0xFF03             / 64K, read only, system only
        soutb   MMU1+0x0100, rl0        / set MMU for segment n (CODE)
        soutb   MMU1+0x0F00, rh1        / base high
        soutb   MMU1+0x0F00, rl1        / base low
        soutb   MMU1+0x0F00, rh3        / limits
        soutb   MMU1+0x0F00, rl3        / attributes
                                        / MMU segment n+1 (DATA)
        ld      r3, $0xFF02             / 64K, read/write, system only
        soutb   MMU1+0x0F00, rh2        / data base high
        soutb   MMU1+0x0F00, rl2        /           low
        soutb   MMU1+0x0F00, rh3        / limits
        soutb   MMU1+0x0F00, rl3        / attributes
        ret

/ Copy 'n' bytes from address #1 to address #2
/ ppcopy(p1, p2, n)
/ long p1, p2;
/ unsigned n;
/
/ Clear 'n' bytes starting at a physical address 'p'
/ pclear(p, n)
/ long p;
/ unsigned n;

        .globl  ppcopy_, pclear_

ppcopy_:
        ldl     rr2, SP(4)              / from address
        ldl     rr4, SP(8)              / to address
        ld      r1, SP(12)              / count
        ldirb   (rr4), (rr2), r1
        ret

pclear_:
        ldl     rr2, SP(4)              / destination
        ldl     rr4, rr2                / source
        ld      r1, SP(8)               / count
        clrb    (rr2)                   / clear 1 byte
        inc     r5                      / Point to next byte
        dec     r1
        jr      eq, 1f                  / branch if nothing to do
        ldirb   (rr4), (rr2), r1
1:
        ret

/
/ Simple trap handling routines.
/
tepa:
#ifndef DDT
```

```
#else
        push    (SP), $SIGEPA           / Extended instruction trap
#endif
        jr      trap

tprv:
#ifndef DDT
        push    (SP), $2                / privileged instruction trap
#else
        push    (SP), $SIGPRV           / privileged instruction trap
#endif
        jr      trap

tsys:
#ifndef DDT
        push    (SP), $3                / System call trap
#else
        push    (SP), $SIGSYS           / System call trap
#endif
        jr      trap

tseg:
        soutb   MMU12+0x1100, rl0       / Reset violation
#ifndef DDT
        push    (SP), $4                / Segmentation violation trap
#else
        push    (SP), $SIGSEG           / Segmentation violation trap
#endif
        jr      trap

tnmi:
#ifndef DDT
        push    (SP), $5                / Non-maskable interrupt
#else
        push    (SP), $SIGNMI           / Non-maskable interrupt
#endif
        jr      trap

tnvi:
#ifndef DDT
        push    (SP), $6                / Non-vectored interrupt
#else
        push    (SP), $SIGBPT           / Non-vectored interrupt
#endif
trap:
#ifndef DDT
        dec     r15, $12                / Save registers that ...
        ldm     (SP), r0, $6            /  C doesn't save
#else
        sub     r15, $32                / Save all registers.
        ldm     (SP), r0, $16
        ldctl   r0, NSPSEG              / Save normal mode sp segment
        ldctl   r1, NSPOFF              / Save part of normal mode sp
        pushl   (SP), rr0
        ldl     ddtregp_, SP            / Allows ddt to find registers
        ld      r0, $ES                 / Set ES segment
        soutb   MMU+0x0100, rh0         /   ES + OS maps
        sinb    rh3, MMU+0x0C00         / ES high
        sinb    rl3, MMU+0x0C00         / ES low
        sinb    rh2, MMU+0x0C00         / OS high
        sinb    rl2, MMU+0x0C00         / OS low
        pushl   (SP), rr2               / push ES+OS
/       ei      VI
```

```
#endif
        call    trap_                   / C trap handler
#if DDT
        di      VI                      / Keep out vectored interrupts
        popl    rr0, (SP)               / rr0 = OS+ES
        ld      r2, $ES                 / Restore OS + ES ...
        soutb   MMU+0x0100, rh2         /   segment maps
        soutb   MMU+0x0C00, rh1         / ES high
        soutb   MMU+0x0C00, rl1         / ES low
        soutb   MMU+0x0C00, rh0         / OS high
        soutb   MMU+0x0C00, rl0         / OS low
        popl    rr2, (SP)               / rr2 = NSP
        ldctl   NSPSEG, r2              / Segment of NSP
        ldctl   NSPOFF, r3

        ldm     r0, (SP), $14           / Re-load R0-R13
        add     r15, $32+2              / Pop saved regs + trap type
#else
        ldm     r0, (SP), $6            / Restore registers
        add     r15, $12
#endif
        iret

        .prvd
        / <> 0 means had hardware error during power-up, so don't autoboot
        .globl  ddtregp_,estack_,harderr_,usrabrt_
harderr_: .byte 0
usrabrt_: .byte 0       / <> 0 means user aborted autoboot
        .bssd
ddtregp_: .blkl 1                       / SP for finding registers
seg_:   .blkw   1
stack:  .blkb   512
estack_:
        .blkb   20                              / for safety
```