```
//Mesa.bcpl - BCPL setup for Mesa Emulator - R. Johnsson
//last modified August 10, 1978  9:47 AM

//bldr mesa format mboot mesa-nova1 mesa-nova2 mesaram readpram gp timeconva timeconvb timeio
//files mesaram and later may not be used after FindSpace()

//incompatible microcode with version 15a; February 3, 1977
//ROM compatible microcode with version 16a; February 18, 1977
//incompatible microcode with version 18a; May 18, 1977
//incompatible microcode with version 19a; May 27, 1977
//new version numbering at 23.3
//check file format at version 29; March 1978

manifest [ MajorVersion = 29; MinorVersion = 16 ]
manifest [ ImageVersionID = 03168 ]
manifest [ printversion = true ]

get "SysDefs.d"
get "Streams.d"
get "AltoFileSys.d"

// OS routines
external        [
        Gets; Puts; Resets; Ws; ReadBlock
        OpenFile; Closes; Endofs
        TruncateDiskStream
        MoveBlock; Zero
        RealDA; RealDiskDA
        MyFrame
        Junta; OsFinish
        DisableInterrupts; EnableInterrupts
        OutLd; InLd
        GetCompleteFa; PositionPage
        ]
// OS statics
external        [
        dsp
        keys
        sysDisk
        lvUserFinishProc
        fpComCm; fpSysDir
        ]
manifest        [
        InterruptVector = #501
        DisplayInterruptWord = #421
        WakeupsWaiting = #452
        Active = #453
        PointerToBootMap = #24
        PuntData = #456
        DefaultPriority = 13
        SwatInterruptLevel = 3
        SwatInterruptBit = #10
        TimeoutInterruptLevel = 4
        TimeoutInterruptBit = #20
        ParityInterruptBit = 1
        // These must match SDDefs.mesa
        SystemDispatch = #1060
        sGoingAway = #43
        sFirstProcess = #55
        sLastProcess = #56
        sProcessTrap = #57
        sFirstStateVector = #60
        ]

external        RamImage                                //mesaram

external        [ SetupReadParam;ReadParam ]            //gp
external        [ ReadPackedRAM; LoadPackedRAM ]        //readpram
external        [ FORMATN; CONCATENATE ]                //format
external        [ UNPACKDT; WRITEUDT ]          //ctime
external        [                               //mesa-nova
        EMLOOP
        MesaNova1
        MesaNovaSize1
        AC1Ptr
        AbsoluteTXV
```

```
        CleanUpQueueUser
        RequeueSubUser
        WakeHeadImplementer
        STOPUser
        AdvanceTimerPtr
        processTrapPtr
        firstProcessPtr
        lastProcessPtr
        firstStateVectorPtr
        PScode
        MesaNova2
        MesaNovaSize2
        STOPImplementer
        CleanUpQueueImplementer
        RequeueSubImplementer
        WakeHeadUser
        OSFPtr
        OutLdPtr
        InLdPtr
        FinishPtr
        FinProcPtr
        ]
external        [                               //mboot
        MBOOT
        SwatFlag
        ]

manifest        [ MesaStart=#420] //Starting address in ram
manifest        [ TXV=#25 ] //Transfer Vector for Nova Code
manifest        [ xNovaCode=#174400 ] //Extended Nova Code
manifest        [ MicrocodeOption=#1 ]

static  [ EmulateLoop; BootData ]
static  [ ImageCount=0 ]
static  [ MFileName;IFileName;IFile ]
static  [ ram=true; bootfile=false ]
static  [ PMloc;BLloc;PagesToSkip ]
static  [ FirstBlock; ImageOptions ]
static  [ giveimageversion = false ]

structure string:       [ length byte
                  char↑1,255 byte
                ]

structure BootMap:      [ fp @FP
                  firstpage word
                  address↑0,(255-lFP-1) word
                ]
manifest [ lBootMap = 256 ]

structure BltItem:      [
                  firstSourceM1 word
                  lastDest word
                  minusCount word
                ]

structure BootList:     [
                  pageMap word
                  firstDa word
                  initialState word
                  blt↑0,3 @BltItem // could be any number of these
                  terminator word // = 0
                ]
manifest [ lBootList = size BootList/16 ]

structure StateVector:  [
                  stk↑0,7 word
                  instbyte byte
                  fill bit 4
                  stkptr bit 4
                  X word
                  Y word
                ]
manifest [ lStateVector = size StateVector/16 ]

structure VersionStamp: [
```

```
        zapped bit 1
        net bit 7
        host bit 8
        time: [
                low word
                high word
                ]
        ]

structure ImagePrefix:  [
                  versionident word
                  version @VersionStamp
                  creator @VersionStamp
                  options word
                  leaderDA word
                  state @StateVector
                  loadStateBase word
                  initialLoadStateBase word
                  type bit 2
                  fill bit 5
                  loadStatePages bit 9
                ]
manifest [ lImagePrefix = size ImagePrefix/16 ]
manifest [ FirstImageDataPage = 2 ]
manifest [      // image types
        bootmesa = 0
        makeimage = 1
        checkfile = 2
        other = 3
        ]

structure MapItem: [
                  page byte
                  count bit 7
                  tag bit 1
                  da word
                  base word
                ]
manifest [
        lnormalMapItem = 1
        lchangeMapItem = 3
        ]

let Mesa(layout,up,cfa) be
[Mesa
let f1=vec 20; IFileName=f1; IFileName!0=0
let f2=vec 20; MFileName=f2; MFileName!0=0



if printversion then WriteVersion()

IFile = 0
until up!0 eq 0 do
  [
  if up>>UPE.type eq openStreams then
    [ IFile = up!1; break ]
  up = up + up>>UPE.length
  ]

if IFile eq 0 then SetupParams()


//Load emulation state from image file
let header = vec lImagePrefix-1
ReadImageBlock(header,lImagePrefix)

if giveimageversion then
        [
        WriteStamp(lv header>>ImagePrefix.version)
        Ws(", creator ")
        WriteStamp(lv header>>ImagePrefix.creator)
        KeyboardWait()
        ]
let imageCfa = vec lCFA-1
GetCompleteFa(IFile, imageCfa)
```

```
if header>>ImagePrefix.versionident ne ImageVersionID then
        AbortMsg("*NIncorrect image file format.")
if header>>ImagePrefix.type eq checkfile &
   header>>ImagePrefix.leaderDA ne imageCfa>>CFA.fp.leaderVirtualDa then
        AbortMsg("*NThis CheckPoint file has been tampered with.")

ImageOptions=header>>ImagePrefix.options

//read in the page address-count words until a 0 is found
let PageMap=vec 250
let Maplast=0
for i=0 to 250 do
    [
    PageMap!i=ReadImage()
    if PageMap!i eq 0 then
        [
        Maplast=i
        break
        ]
    ]
PagesToSkip=0   //number of pages to skip in file

LoadMesaMicrocode()

FindSpace(PageMap,Maplast)

//get stuff ready for page zero
let bd = vec 1BootList; BootData = bd; Zero(BootData, 1BootList)
FixMESANOVA()

//find disk address of page FirstImageDataPage of image file
PositionPage(IFile, FirstImageDataPage)
GetCompleteFa(IFile, imageCfa)
RealDiskDA(sysDisk, imageCfa>>CFA.fa.da, 1v FirstBlock)

// set up bootmap
[
@PointerToBootMap = PMloc
MoveBlock(PMloc,1v (imageCfa>>CFA.fp),1FP)
PMloc>>BootMap.firstpage = FirstImageDataPage
for i = 0 to PagesToSkip-1 do
        PMloc>>BootMap.address↑i = PageMap!0&#177400
let a = PagesToSkip;
let nexti = nil
let i = 0
until i eq Maplast do
        [
        let item = 1v (PageMap!i)
        test item>>MapItem.tag
          ifso
            [
            PMloc>>BootMap.address↑a = (item>>MapItem.base lshift 1) + 1
            PMloc>>BootMap.address↑(a+1) = item>>MapItem.da
            a = a + 2
            nexti = i+1changeMapItem
            ]
          ifnot nexti = i+1normalMapItem

        let memaddress = item>>MapItem.page lshift 8
        for j = 1 to item>>MapItem.count do
            [
            PMloc>>BootMap.address↑a = memaddress
            memaddress = memaddress + #400
            a = a + 1
            ]
        i = nexti
        ]
PMloc>>BootMap.address↑a = 0
]

EmulateLoop=TXV
FixAndMoveMBOOT()

let intvec = vec 15
FixInterrupts(intvec)
let initialstate = xNovaCode+MesaNovaSize2
```

```
MakeBltItem(lv BootData>>BootList.blt↑3,
    lv header>>ImagePrefix.state, initialstate, lStateVector)
BootData>>BootList.initialState = initialstate;

@lvUserFinishProc = FinishPtr
Junta(levBasic, Go)

]Mesa

and Go() be
[ BLloc(BootData) ]

and SetupParams() be
[
//Get switches from command line
let StringVec=vec 100
let SwitchVec=vec 100
let GlobalSwitchVec=SwitchVec
let comcm = OpenFile("Com.Cm", ksTypeReadOnly, charItem, verLatest, fpComCm);
SetupReadParam(StringVec,SwitchVec,comcm,GlobalSwitchVec)

ImageCount=0
let done = false

let usename = false

test ImageFile(StringVec)
  ifso usename=true
  ifnot
    if (GlobalSwitchVec!0 ne 0) then for I=1 to GlobalSwitchVec!0 do
        [SwitchLoop
        switchon GlobalSwitchVec!I into
                [SwitchCases
                case $V: case $v:
                        [V
                        if not printversion then WriteVersion()
                        giveimageversion = true
                        endcase
                        ]V
                case $M: case $m:
                        [M
                        test LoadMesaMicrocode()
                          ifso [ Ws( "*NMicrocode loaded"); finish ]
                          ifnot AbortMsg("*NThis machine has extra ROM; can't use RAM")
                        endcase
                        ]M
                case $S: case $s:
                        [S
                        @SwatFlag=#77400
                        endcase
                        ]S
                case $Q: case $q:
                        [C
                        done=true
                        endcase
                        ]C
                case $B: case $b:
                        [B
                        bootfile=true
                        endcase
                        ]B
                default:
                        [Huh
                        Ws("*NBad switch encountered. ")
                        endcase
                        ]Huh
                ]SwitchCases
        ]SwitchLoop

let rewritecomcm = not usename

// read the parameters
test usename
  ifso CONCATENATE(IFileName,StringVec)
  ifnot
    until done do
```

```
    [paramloop
    let p = ReadParam($P,0,0,0,true)
    if (p eq 0)%(p eq -1) then break
    test SwitchVec!0 eq 0
        ifso if IFileName!0 eq 0 then [ CONCATENATE(IFileName,p); break ]
        ifnot
        [localswitches
        for I=1 to SwitchVec!0 do
            switchon SwitchVec!I into
            [
            case $C: case $c:
                [ if IFileName!0 eq 0 then CONCATENATE(IFileName,p);
                  done = true;
                  endcase
                ]
            case $I: case $i:
                [ CONCATENATE(IFileName,p); endcase ]
            case $M: case $m:
                [ CONCATENATE(MFileName,p); endcase ]
            default:
                [
                Ws(FORMATN("*NBad switch '<C>', item will be ignored.", SwitchVec!I))
                KeyboardWait()
                ]
            ]
        ]localswitches
    ]paramloop

if MFileName!0 ne 0 then DefaultName(MFileName,"MESA","PRAM")
if bootfile then
        [
        DefaultName(IFileName,"MESA","SV")
        LoadMesaMicrocode()
        let message = vec 1InLdMessage
        let fp = vec 1FP
        let cfa = vec 1CFA
        let file = OpenFile(IFileName,ksTypeReadOnly,wordItem)
        if ((file eq 0)%(file eq -1)) then
                AbortMsg(FORMATN("*NFile '<S>' not found.",IFileName))
        GetCompleteFa(file,cfa);
        MoveBlock(fp, lv cfa>>CFA.fp, 1FP)
        let realda = 0
        RealDiskDA(sysDisk, cfa>>CFA.fa.da, lv realda)
        fp>>FP.leaderVirtualDa = realda
        message!1 = #377  // level = -1, reason = proceed
        InLd(fp,message)
        ]

DefaultName(IFileName,"Mesa","image")

// maybe rewrite comcm here
test rewritecomcm
    ifso
    [   let newcomcm = OpenFile("Com.Cm", ksTypeWriteOnly, charItem, verLatest, fpComCm)
        for i=1 to IFileName>>string.length do
            Puts(newcomcm,IFileName>>string.char↑i)
        test Endofs(comcm)
          ifso Puts(newcomcm,$*N)
          ifnot Puts(newcomcm,$*S)
        until Endofs(comcm) do
            Puts(newcomcm,Gets(comcm));
        Closes(comcm); Closes(newcomcm);
    ]
    ifnot Closes(comcm)

IFile=OpenFile(IFileName,ksTypeReadOnly,wordItem)
if ((IFile eq 0)%(IFile eq -1)) then
        AbortMsg(FORMATN("*NImage file '<S>' not found.",IFileName))

] // end SetupParams
```

```
and ImageFile(name) = valof
[
        let cap(c) = ((c ge $a) & (c le $z)) ? c+$A-$a, c
        let s = vec 40
        s>>string.length = name!0
        for i = 1 to s>>string.length do
          s>>string.char↑i = name!i
        MoveBlock(name,s,name!0)
        s=".IMAGE"
        let ofs = name>>string.length-s>>string.length
        if ofs ls 0 then resultis false
        for i = 1 to s>>string.length do
          if cap(name>>string.char↑(ofs+i)) ne s>>string.char↑i then resultis false
        resultis true
]

and FixAndMoveMBOOT() be
[
        BootData>>BootList.pageMap = PMloc
        BootData>>BootList.firstDa = FirstBlock

        MoveBlock(BLloc, MBOOT, 256);

]

and FixMESANOVA() be
[
        if AbsoluteTXV ne TXV then
            AbortMsg("Code in Mesa-Nova incorrectly assembled")
        @OSFPtr = OsFinish
        @AC1Ptr = MesaStart
        @OutLdPtr = OutLd
        @InLdPtr = InLd
        @FinProcPtr = SystemDispatch+sGoingAway
        @processTrapPtr = SystemDispatch+sProcessTrap
        @firstProcessPtr = SystemDispatch+sFirstProcess
        @lastProcessPtr = SystemDispatch+sLastProcess
        @firstStateVectorPtr = SystemDispatch+sFirstStateVector
        @STOPUser = STOPImplementer
        @CleanUpQueueUser = CleanUpQueueImplementer
        @RequeueSubUser = RequeueSubImplementer
        @WakeHeadUser = WakeHeadImplementer
        MakeBltItem(lv BootData>>BootList.blt↑0, MesaNova1, TXV, MesaNovaSize1)
//      MoveBlock(TXV, MesaNova1, MesaNovaSize1)
        MakeBltItem(lv BootData>>BootList.blt↑1, MesaNova2, xNovaCode, MesaNovaSize2)
//      MoveBlock(TXV, MesaNova2, MesaNovaSize2)
        @PuntData = 0
]

and MakeBltItem(item, source, dest, count) be
[
item>>BltItem.firstSourceM1 = source-1
item>>BltItem.lastDest = dest+count-1
item>>BltItem.minusCount = -count
]

and FixInterrupts(v) be
[
        let t=@PScode - 2
        for i=0 to 14 do
            [ t=t+1; v!i=t ]
        MakeBltItem(lv BootData>>BootList.blt↑2,        v, InterruptVector, 15)
        DisableInterrupts()
        @DisplayInterruptWord = SwatInterruptBit
        @Active = SwatInterruptBit
        InterruptVector!SwatInterruptLevel = InterruptVector!8  //SWAT and TIMER
        @WakeupsWaiting = 0
        EnableInterrupts()
        v!SwatInterruptLevel = InterruptVector!SwatInterruptLevel  //SWAT and TIMER
        v!TimeoutInterruptLevel = AdvanceTimerPtr
]

and let LoadMesaMicrocode() = valof
[LoadMesaMicrocode
let MBFile=0
```

```
let ramver = 0
if userom() then resultis false
test ImageOptions&MicrocodeOption
    ifso
        [
        let sink=0
        until (ImageCount&#377) eq #377 do sink=ReadImage()
        PagesToSkip=PagesToSkip+10      //10 more pages in microcode
        if MFileName!0 eq 0 then [ MBFile=IFile; MFileName=IFileName ]
        ]
    ifnot if MFileName!0 eq 0 then
        [
        LoadPackedRAM(RamImage, lv ramver)
        resultis true
        ]

if MBFile eq 0 then
  MBFile=OpenFile(MFileName,ksTypeReadOnly,wordItem)
if (MBFile eq 0) then
        AbortMsg(FORMATN("*N<S> not found.",MFileName))

let MBProblems=ReadPackedRAM(MBFile)
if (MBProblems ne 0) then
        [
        Ws(FORMATN("*N<S> had some constant disagreements.",MFileName))
        KeyboardWait()
        ]
if MBFile ne IFile then Closes(MBFile)
]LoadMesaMicrocode

and let userom() = valof
[
    manifest highword = #2000;
    let writeone=table [  // (high,address,low)
        #55001          //STA 3 1 2
        #35003          //LDA 3 3 2
        #61012          //WRTRAM
        #35001          //LDA 3 1 2
        #1401           //JMP 1 3
        ]
    let readone=table [  // (pointer,address)
        #55001          //STA 3 1 2
        #115000         //MOV 0 3
        #61011          //RDRAM
        #41400          //STA 0 0 3
        #35001          //LDA 3 1 2
        #1401           //JMP 1 3
        ]
    let testcode = table [
        #14030; #102020;        // x1: AC0←L,:START;
        #20; #170776;           //     L←ONE,SWMODE,:x1;
        ]
    let vers=table [
        #61014          //VERS
        #1401           //JMP 1 3
        ]
    let altotype = vers() rshift 12
    if altotype eq 4 % altotype eq 5 then resultis true
    let saveram = vec 3
    readone(saveram,highword+#776);   readone(saveram+1,#776);
    readone(saveram+2,highword+#777); readone(saveram+3,#777);
    writeone(testcode!0,#776,testcode!1);
    writeone(testcode!2,#777,testcode!3);
    let isram = ((table [ #61010; #1401 ])(0,#777));
    writeone(saveram!0,#776,saveram!1);
    writeone(saveram!2,#777,saveram!3);
    if isram then resultis false
    resultis true
]
```

```
and let FindSpace(PageMap,Maplast) be
[FindSpace
//find space for boot loader and pagemap which does not interfere
//with this program or the image to be loaded
let InUse=vec 15
Zero(InUse, 16)
let bootListSize = 0
let nexti = nil
let i = 0
until i eq Maplast do
    [
    let item = lv (PageMap!i)
    test item>>MapItem.tag
        ifso
            [
            nexti = i+lchangeMapItem
            bootListSize = bootListSize + 2
            ]
        ifnot nexti = i+lnormalMapItem
    let page, count = item>>MapItem.page, item>>MapItem.count
    bootListSize = bootListSize + count
    for j=page to page+count-1 do
        [
        let wd = j rshift 4
        InUse!(wd) = InUse!(wd) % (#100000 rshift (j&#17))
        ]
    i = nexti
    ]
let lastpage=(MyFrame()-256) rshift 8
let firstpage=(RamImage+255) rshift 8
let p = AllocPages(InUse, firstpage, lastpage, 1)
BLloc = p lshift 8
if p ne 0 then
    p = AllocPages(InUse, p+1, lastpage, (bootListSize+256) rshift 8)
PMloc = p lshift 8
if PMloc eq 0 then
        AbortMsg("*NCan't find enough space for loader.")
]FindSpace

and AllocPages(map, first, last, npages) = valof
[
let n = 0
for i = first to last do
    [
    test (map!(i rshift 4) &  (#100000 rshift (i & #17))) eq 0
      ifso
        [ n = n + 1; if n eq npages then resultis i-n+1 ]
      ifnot n = 0
    ]
resultis 0
]

and DefaultName(name,defname,defext) be
[DefaultName
if name!0 eq 0 then
    [
    CONCATENATE(name,defname,".",defext)
    return
    ]
for I=1 to name>>string.length do
    [
    if name>>string.char↑I eq $. then return
    ]
CONCATENATE(name,name,".",defext)
]DefaultName

and let KeyboardWait() be
[KeyboardWait

Ws(" [] ")
let Char=Gets(keys)
Puts(dsp,Char)
switchon Char into
    [
    case $F: case $f: finish
```

```
    ]
]KeyboardWait


and AbortMsg(s) be
[AbortMsg
        Ws(s)
        Resets(keys)
        KeyboardWait()
        finish
]AbortMsg

and ReadImage()=valof
[ReadImage
ImageCount=ImageCount+1
if Endofs(IFile) then AbortMsg("Premature end of image file.")
resultis Gets(IFile)
]ReadImage

and ReadImageBlock(p,n) be
[ReadImageBlock
ImageCount=ImageCount+n
if ReadBlock(IFile,p,n) ne n then
        [
        AbortMsg("Premature end of image file.")
        ]
]ReadImageBlock

and WriteVersion() be
[WriteVersion
Ws(
FORMATN("RunMesa <D>.<D>, microcode <D>*N", MajorVersion, MinorVersion, @RamImage))
]WriteVersion

and WriteStamp(v) be
[WriteStamp
  let uv = vec 6
  let dv = vec 1
  dv!0, dv!1 = v>>VersionStamp.time.high, v>>VersionStamp.time.low // reverse
  UNPACKDT(dv, uv)
  WRITEUDT(dsp,uv)
  Ws(FORMATN(" <B>#<B>#",v>>VersionStamp.net,v>>VersionStamp.host))
  if v>>VersionStamp.zapped ne 0 then
    Ws(" zapped!")
]WriteStamp
```