```
SERIES-III PL/M-86 X19A COMPILATION OF MODULE CBBOOTSTRAP
OBJECT MODULE PLACED IN :F1:BOOTB.OBJ
COMPILER INVOKED BY:  :F2:PLM86.86 :F1:BOOTB.P86 OPTIMIZE(3) COMPACT DEBUG ROM


            $set(Rv)
            $nocond
            $if Rv
            $title('PARTIAL MODEL BOOTSTRAP PROGRAM')
            $elseif Fv
            $endif
            $nointvector
   1        CB$bootstrap:
            do;
            $include (:f1:prop.lit)
        =   /*
        =       Intel Corporation Proprietary Information. This listing is
        =       supplied under the terms of a license agreement with Intel
        =       Corporation and may not be copied nor disclosed except in
        =       accordance with the terms of the agreement.
        =   */
            $nolist include(:f1:common.lit)
            $nolist include(:f1:kaos.dcp)
            $nolist include(:f1:dll.dcp)
            /*
              declare literals
            */
 100   1    declare /*
                        declare commands first
                    */
                    C$Presence literally '1',
                    C$Local$load$go literally '2',
                    C$Remote$boot literally '3',
                    C$MIP$init$go literally '4',
                    C$Local$load literally '5',
                    C$Local$dump literally '6',
                    C$Go literally '7',
                    C$Remote$forced$boot$go literally '9',
                    C$do$echo$request  literally '8',
                    C$echo$request literally '10',
                    C$Remote$dump$request literally '11',
                    C$do$reset literally '12',
                    /*
                      now declare responses
                    */
                    CMD$OK literally '1',
                    No$response literally '2',
                    E$EOF literally '3',
                    No$echo literally '2',
                    Abort$boot literally '4',

                    RL$no$response literally '2',
                    Bad$command literally '0FFH',
                    /*
                      declare NML commands that are processed by boot
                    */
            NML$type literally '500CH',
            NML$type$r literally '0050h',
            NML$remote$request literally '1',
            NML$remote$reply literally '2',
            NML$forced$boot$go literally '3',
            NML$boot$cmd$request literally '4',
            NML$boot$cmd$response literally '5',
            NML$boot$data$request literally '6',
            NML$boot$data$response literally '7',
            NML$echo$request literally '8',
            NML$echo$response literally '9',
            NML$remote$dump$recuest literally '10',
            NML$remote$dump$response literally '11',
            NML$reset literally '15',
            /*
              declare Mip limits
            */
            MIP$devices literally '6',
            MIP$ids$s literally '8',
            /*
              declare OMF bits
            */
            Morebit literally '1H',
            Start$bit literally '2H',
            /*
              declare LSI device addresses
            */
            PIC$PB literally '0E1H',
            PIC$PC literally '0E2H',
            PIC$CMD literally '0F0H',
            PIC$DATA literally '0F1H',
            PIC$MASK literally '0F1H',
            /*
              declare PIC cmds
            */
            PIC$ICW1 literally '0001$0011B',
            PIC$ICW2 literally '0001$0000B',
            PIC$ICW4 literally '0000$0011B',
            POLL$PIC literally '0CH';
        $eject
```

```
          /*
            declare external variables. These will all be in other CB firmware
            modules

            for Mip first
          */
101   1   declare CQ$MIP$Ids$bases (Mip$idss) structure (
                    base byte,
                    length byte)  external,

                  CQ$MIP$Device$info (Mip$devices) structure(
                    devid byte,
                    Status byte,
                    RQDin pointer,
                    RQDout pointer,
                    Int$type byte,
                    Time$to$wait byte,
                    Int$adr word ) external,

                  CQ$Thisdevice byte external;
          $eject
```

```
              /*
                declare external procedures

                first the confidence test routines
              */
 102    1     CQ$ram$test:
 103    2       procedure byte external; end CQ$ram$test;

 104    1     CQ$device$test:
 105    2       procedure byte external; end CQ$device$test;
              /*
                now init routine
              */
 106    2     CQ$CB$init: procedure external; end CQ$CB$init;
 108    2     CQ$Hdw$init: procedure external; end CQ$Hdw$init;
              /*
                now for special routines for accessing other routines
              */
 110    1     Long$goto: procedure(SA) external;
 111    2       declare SA pointer; end Long$goto;

 113    1     Short$call: procedure(SA) external;
 114    2       declare SA word; end Short$call;

 116    1     Subsystem$call: procedure(Para,Proc) external;
 117    2       declare (Para,Proc) word; end Subsystemcall;
              /*
                MIP now
              */
 119    2     CQ$MIP$init: procedure external; end CQ$MIP$init;

 121    2     CQ$MIP$Intask: procedure external; end CQ$MIP$Intask;

 123    2     CQmiphalt: procedure external; end CQmiphalt;
              /*
                now to external data link
              */
 125    1     EDL$start:
 126    2       procedure external; end EDL$start;
              /*
                declare restart point
              */
 127    1     declare CQBoot$dmt$entry address external;

 128    1     declare Currentversion byte external;

 129    1     declare CQmipdevcnt byte external,
                      CQmipdevtoentry (8) byte external;

 130    1     declare CQ_DLL_hostid (6) byte external;

              $eject
```

```
declare /*
          declare structures to get command block address
        */
        Copyright (*) byte data('(C) 1981 INTEL CORP'),

        Cmd$block$ptr pointer public,
        CBptr structure (off word, base word) at (@Cmd$block$ptr),
        Cmd$block$ptr$o word at (@CBptr.off),
        Cmd$block$ptr$b word at (@CBptr.base),
        Cmd$block$addresses (8) word public
           data(0F69H,1F00H,2041H,100H,800H,1000H,2000H,2F00H),
        /*
          declare variables for control purposes
        */
        First$boot$cmd byte  at (0F3FFFH),
        Miprunning boolean  at (0F3FFEH),
        Ran$RT boolean at (0F3FFDH),
        /*
          declare variables for buffering purposes in remote loading
        */
        Num$remaining word,
        Last$buffer boolean,
        Bufptr pointer,
        Bufadr structure(
         Offset word,
         Base   word) at (@Bufptr),
        /*
          declare local variables for various purposes
        */
        CTresult byte public ,
        Run boolean,
        Cmd byte public,
        NML$entry word public,
        Execution$SA word public,
        Mip$def$ptr pointer public,
        DLL$bufptr pointer,
        DLL$bufptr$o word public at (@DLLbufptr),
        /*
          declare data structures for CA handling
        */
        CQ$boot$cmd$mb (16) byte external,
        Local$boot$cmd$msg structure (
          link pointer,
          Length word,
          dll$filler (12) byte,
          type word ) public ,
        /*
          declare structure for remote comm
        */
        CQ$Remote$waiting$mb (16) byte external,
        CQ$Wait$acb (16) byte external,
        Tries byte public,
        /*
          to convert from local to remote commands
        */
        Remote$to$local$cmd (16) byte data (0,0,0,C$Remote$Forced$boot$go,
          C,0,C,0,C$echo$request,0,C$remote$dump$request,0,0,0,C,C$do$reset),
            /*
              data structures for remote booting
            */
            Remote$booter$adr (6) byte data (01H,0AAH,0,0FFH,0FFH,0FFH),
            Temp$buf structure (
              Srcadr (6) byte,
              Type word,
              Cmd byte,
              Start$adr pointer,
              Length word ) public,
            Remote$server$adr (6) byte at (@Temp$buf.srcadr),
            Reqclass word at (@Temp$buf.Start$adr),
            Next$Remote$block word,
            /*
              structure for Series IV reporting
            */
            S4 based Cmd$block$ptr structure(deviceid byte,result byte),
            Status$report structure (
              done byte,
              semaphore byte,
              result$blk$ptr word) at (2F0C0H),
            /*
              declare data structures for loading code remotely
            */
            Rload structure (
              Lcmd byte,
              Load$sa pointer,
              Length word,
              Exec$sa word) ;

        $eject
```

```
                    /*
                      declare based structures for local commands
                    */
 132   1        declare Prc$cmd based Cmd$block$ptr structure
                    ( Cmd byte,
                      Response byte,
                      Diagnostic$code byte,
                      Version byte,
                      Hostid (6) byte),

                    LL$area based Cmd$block$ptr structure
                    ( Cmd byte,
                      Response byte,
                      From$area pointer,
                      To$area pointer,
                      Length word,
                      Exec$SA word,
                      Mip$def$area byte ),

                    Remote$boot based Cmd$block$ptr structure
                    ( Cmd byte,
                      Response byte,
                      Class$code word),

                    Go$area based Cmd$block$ptr structure
                    ( Cmd byte,
                      Response byte,
                      Exec$SA pointer),

                    Mip$sizes based Mip$def$ptr structure
                    ( Devcnt byte,
                      Ids$cnt byte,
                      This$dev byte,
                      Rsrved byte,
                      Mip$bases (8) word),

                    Mip$dev$def based Mip$def$ptr (1) structure
                    ( Dev$id byte,
                      Status byte,
                      RQD$to$CB pointer,
                      RQD$from$CB pointer,
                      Int$type byte,
                      Time$to$wait byte,
                      Int$adr word  ),

                    Echo$req based Cmd$block$ptr structure (
                      Cmd byte,
                      Response byte,
                      Dest$adr (6) byte,
                      info word,
                      Reply word  );

                /*
                  declare structures for remote commands
                */
 133   1        declare
                    Remote$cmd based DLL$bufptr$o structure(
               Link pointer,
               P$Length word,
               DA$1 pointer,
               DA$2 word,
               SA$1 pointer,
               SA$2 word,
               Type word,
               Cmd byte,
               Info  word ),

             Remote$dump based DLL$bufptr$o structure(
               Link pointer,
               P$Length word,
               DA$1 pointer,
               DA$2 word,
               SA$1 pointer,
               SA$2 word,
               Type word,
               Cmd byte,
               Start$adr pointer,
               Length word,
               Info  byte),

             Remote$data based DLL$bufptr$o structure(
               Link pointer,
               P$Length word,
               DA$1 pointer,
               DA$2 word,
               SA$1 pointer,
               SA$2 word,
               Type word,
               Cmd byte,
               Block word ,
               Lcmd byte,
               Load$SA pointer,
               Length word,
               Exec$SA word,
               Info(1) byte);
         $eject
```

```
                /*
                  declare some utility routines

                  declare the routine to get the Cmd block address
                */
 134     1      Load$Cmd$block$ptr:
                $if Rv
                  procedure external;
                $else
                $endif
 135     2      end Load$Cmd$block$ptr;
                /*
                  declare procedures to load comm memory to/from host
                */
 136     1      Local$move:
                $if Rv
                  procedure external;
                $else
                $endif
 137     2      end Local$move;
                /*
                  short form for DLL Send
                */
 138     1      DLLsend:
                $if Rv
                  procedure external;
                $else
                $endif
 139     2      end DLLsend;
                /*
                  short form for dll return buffer
                */
 140     1      DLLretbuf:
                $if Rv
                  procedure external;
                $else
                $endif
 141     2      end DLLretbuf;
                /*
                  boot interrupt routine
                */
 142     1      CQ$CA$int$routine:
                $if Rv
                  procedure external;
                $else
                $endif
 143     2      end CQ$CA$int$routine;
                /*
                  set LED on or off depending on results of CT
                */
 144     1      Set$LED:
                $if Rv
                  procedure external;
                $else
                $endif
 145     2      end Set$LED;
                /*
                routine for NML to use when present
              */
 146   1      Boot$register:
              $if Rv
                procedure external;
              $else
              $endif
 147   2      end Boot$register;
              /*
                routine to save src adr and other info of a msg
              */
 148   1      Save$rcvd$info:
              $if Rv
                procedure external;
              $else
              $endif
 149   2      end Save$rcvd$info;

 150   1      Bumpandchecktries:
              $if Rv
                procedure boolean external;
              $else
              $endif
 151   2      end Bumpandchecktries;
              $eject
              /*
                first define DLL interface routines
              */
 152   1      Transmit:
              $if Rv
                procedure(DAptr,Cmd,Info) external;
              $else
              $endif
 153   2        declare DAptr pointer,
                        Cmd byte,
                        Info word;

              $if not Rv
              $endif
 154   2      end Transmit;
              /*
                this routine does waiting for a reply
              */
 155   1      Wait$for$reply:
                procedure boolean;
                /*
                  send msg via DLL
                */
 156   2        call DLLsend;
                /*
                  start timer and then wait for a reply
                */
 157   2        call CQ$set$alarm(@CQ$Wait$acb,.CQ$Remote$waiting$mb,100,0);
 158   2        DLL$buf$ptr = CQ$receive(.CQ$Remote$waiting$mb);
                /*
                  have something, see if timeout or msg
                */
 159   2        call CQ$Clear$alarm(@CQ$Wait$acb);
                /*
                  check if timeout
                */
 160   2        return (DLL$buf$ptr$o <> .CQ$Wait$acb);
 161   2      end Wait$for$reply;

              $eject
```

```
            /*
              declare buffering routines for remote loading
            */
 162   1    Get$buffer:
              procedure byte;
                /*
                return current buffer
                */
 163   2        Tries = 0;
 164   2    L0: call DLLretbuf;
                /*
                  if the previous buffer was the last one,return EOF
                */
 165   2        if Last$buffer then return E$EOF;
                /*
                  get transmit buffer to send request
                */
 167   2    L1: call Transmit(@Remote$server$adr,NML$boot$data$request,Next$remote$block);
                /*
                  send to DLL and wait for a reply
                */
 168   2        if not Wait$for$reply then
 169   2        do;
 170   3          if Bumpandchecktries then return No$response;
 172   3          else goto L1;
 173   3        end;
                /*
                  have reply, make sure it is what we want
                */
 174   2        if (Remote$data.cmd <> NML$boot$data$response) or
 175   2           (Remote$data.block <> Next$remote$block) then  go to L0;
                /*
                  it is, set up pointer and things
                */
 176   2        Next$remote$block = Next$remote$block + 1;
 177   2        if (Num$remaining:= Remote$data.P$Length -17) <> 1497 then
 178   2           Last$buffer=TRUE;
 179   2        Bufptr = @Remote$data.lcmd;
 180   2        return CMD$OK;
 181   2    end Get$buffer;

 182   1    Read$bin:
              procedure(Dptr,Count) byte public;
 183   2      declare Dptr pointer, Count word,
                      Cadr structure (offset word, base word) at (@Dptr),
                      To$move word, Status byte;

 184   2      do while (Count <> 0);
 185   3        if Num$remaining = 0 then
 186   3        do;
                  /*
                    buffer is empty, get PRM to  refill it
                  */
 187   4          if (Status := Getbuffer) <> CMD$OK then return Status;
 189   4        end;
                /*
                  transfer what is needed or what we have, whatever is less
```

```
                 */
      3          if Count > Numremaining then To$move = Num$remaining;
      3          else To$move = Count;
      3          call Movb(Bufptr,Dptr,To$move);
      3          Dadr.offset = Dadr.offset + To$move;
      3          Bufadr.offset  = Bufadr.offset + To$move;
      3          Count = Count - To$move;
      3          Num$remaining = Num$remaining - To$move;
      3        end;
      2        return CMD$OK;
      2      end Read$bin;

             $eject
```

```
/*
  declare remote loading routine
*/
Do$remote$load:
  procedure(Class)  byte;
  declare  Class word, Status byte;

  Lastbuffer = FALSE;
  Status = CMD$OK;
  Tries, Numremaining = 0;
  /*
    send the request and wait for reply
  */
L0: if Bumpandchecktries then return No$response;
  call Transmit(@Remote$booter$adr,NML$boot$cmd$request,Class);
  /*
    send and wait for a reply
  */
  if not Wait$for$reply then goto L0;
  /*
    have reply. hopefully it is what we want
  */
  call Save$rcvd$info;
  if Temp$buf.cmd <> NML$Boot$cmd$response  then
  do;
    goto L0;
  end;
  /*
    see if boot request was  accepted
  */
  Next$remote$block = 0;
  /*
    begin processing load module(s).
  */
L1:
  if (Status:=Read$bin(@Rload.Lcmd,9)) = CMD$OK then
  do;
    if (Status:=Read$bin(Rload.Load$sa,Rload.length)) = CMD$OK then
    do;
      if (Rload.lcmd and Start$bit) <> 0 then call Short$call(Rload.Exec$sa);
      else Execution$SA = Rload.Exec$sa;
      if (Rload.lcmd and More$bit) <> 0 then goto L1;
    end;
  end;
  call DLLretbuf;
  return Status;
end Do$remote$load;

$eject
```

```
                /*
                  declare remote dumping procedure
                */
 231    1       Do$remote$dump:
                  procedure;
                    /*
                      first make copy of important fields in rcv'ed buffer and then
                      release original
                    */
 232    2           call Save$rcvd$info;
                    /*
                      get transmit buffer to send back ack and data
                    */
 233    2           call Transmit(&Temp$buf.srcadr,NML$remote$dump$response,0);
                    /*
                      put data into it
                    */
 234    2           if Temp$buf.length > 1493 then Temp$buf.length=1493;
 236    2           call Movb(Temp$buf.start$adr,@Remote$dump.info,
                             Temp$buf.length);
 237    2           Remote$dump.P$length = Temp$buf.length + 21;
 238    2           Remote$dump.startadr = Temp$buf.startadr;
 239    2           Remote$dump.length = Temp$buf.length;
                    /*
                      give to DLL to send
                    */
 240    2           call DLLsend;
 241    2       end Do$remote$dump;

                $eject
```

```
              /*
                declare routine tto handle MIP things
              */
  242    1    Handle$mip:
              $if Rv
                procedure external;
              $else
              $endif
  243    2    end Handle$mip;

              $eject
```

```
          /*
            define routine that processes most commands
          */
  1       Process$boot$cmd:
            procedure;
  2         declare Response byte;
            /*
              init a few variables
            */
  2         Run = FALSE;
            /*
              check for legal cmd and if ok, then set up to execute each
            */
  2         if Cmd > C$do$Reset then
  2         do;  /* illegal cmd */
  3           LL$area.Response = Bad$command;
  3         end;
  2         else
            do;
              /*
                good command value
              */
  3           Response = CMD$OK;
              /*
                now process each type of command
              */
  3           do case Cmd;

  4             return;

  4             do;   /* Presence */
  5               Prc$cmd.diagnostic$code = CTresult;
  5               Prc$cmd.version = Current$Version;
  5               call movb(@CQ_DLL_hostid,@Prc$cmd.hostid,6);
  5             end;

  4             do;  /* local load and go */
  5               Run = TRUE;
  5               Mipdefptr = @LL$area.Mip$def$area;
  5               Execution$SA = LL$area.Exec$SA;
  5               call Handle$mip;
  5               call Local$move;
  5             end;

  4             do;  /* Remote load */
  5               Response = Do$remote$load(Remote$boot.class$code);
  5             end;

  4             do;  /* MIP Init */
  5               Mipdefptr = @LL$area.From$area;
  5               call Handle$mip;
  5               Run = TRUE;
  5             end;

  4             do;  /* local load */
  5               call Local$move;
  5             end;
```

```
 4           do;  /* Local Dump */
 5             call Local$move;
 5           end;

 4           do;  /* Go */
 5             GO$area.response = CMD$OK;
 5             Output(7H) = 1;
 5             call Long$goto(Go$area.Exec$SA);
 5           end;

 4           do;  /* generate echo request */
 5    Lecho:   call Transmit(@Echoreq.Dest$adr,NML$echo$request,Echoreq.info);
 5             if Wait$for$reply then
 5             do;
 6               call Save$rcvd$info;
 6               Echoreq.reply = Req$class;
 6               if Temp$buf.cmd = NML$echo$response then Response = CMD$OK;
 6               else goto Lecho;
 6             end;
 5             else Response = No$echo;
 5           end;

 4           do;  /* Remote forced boot and go */
               /*
                 start remote load sequence with class code that was passed
               */
 5             call Save$rcvd$info;
 5             Response = Do$remote$load(Reqclass);
 5             return;
 5           end;

 4           do;  /* echo request */
 5             call Save$rcvd$info;
 5             call Transmit(@Remote$server$adr,NML$echo$response,Req$class);
 5             call DLLsend;
 5             return;
 5           end;

 4           do;  /* remote dump request */
 5             call Do$remote$dump;
 5             return;
 5           end;

 4           do;  /* remote reset */
 5             call Long$go$to(@CCbootdmtentry);
 5           end;

 4         end; /* of case stmt */
           /*
             update cmd block
           */
 3         LL$area.Response = Response;
 3         Output(7H)=1;
           /*
             if requested to start system (and boot task) then do it
           */
 319   3           if not Run then return;
 321   3       Dorun:
                   call Short$call(Execution$SA);
 322   3         end;  /* of good cmd */
 323   2       end Process$boot$cmd;
               $eject
```

```
             /*
               define boot task
             */
 324   1     CQ$boot$task:
               procedure public;

 325   2       Local$boot$cmd$msg.type = 0;
 326   2       NMLentry = .CQDLLrxretbuf;
               /*
                 clear out MIP data bases
               */                                    ouch!
 327   2       call setb(0,@CQmipdevcnt,0D6H);
               /*
                 this is the command execution loop. If there is already
                 a command pending, then post dummy msg.
               */
 328   2       do forever;
 329   3     Cmd$loop:
 330   3         if First$boot$cmd then call CQCA$int$routine;     send msg to CQBOOTCMDMB for local boot
                 /*
                   must wait for command
                 */
 331   3         Cmd = CQ$DLL$connect(NML$type,.CQ$boot$cmd$mb);
 332   3         DLL$buf$ptr,Cmd$block$ptr = CQ$Receive(.CQ$boot$cmd$mb);  read that sucker if local boot
                 /*
                   have a command. If the msg is of type local then
                   it came from the host.
                 */
 333   3         if Remote$cmd.type = 0 then
 334   3         do;
                   /*
                     command is local, get cmdblockptr and Cmd
                   */
 335   4           call Load$Cmd$block$ptr;
 336   4         end;
 337   3         else
                 do;
                   /*
                     cmd came from data link. If remote dump then
                     handle it immediately, else let processbootcmd
                     do it.
                   */
 338   4           Cmd = Remote$to$local$cmd(Remote$cmd.Cmd and 0FH);
 339   4           if Cmd = 0 then
 340   4           do;   /* we don't handle this */
 341   5             call Subsystemcall(DLL$buf$ptr$o,NMLentry);
 342   5           end;
 343   4         end;
                 /*
                   we now have either a local command or a non-remote dump
                   command. See if the RAM test has been run
                   and if not, run it. If it has, then execute command
                 */
 344   3         if Ran$RT then
 345   3         do;
                   /*
                     have run RAM test, so execute command;
```

exit boot-KAOS state

```
              */
              call Process$boot$cmd;
              First$boot$cmd = FALSE;
            end;
            else
            do;
              /*
                execute RAM test. this will wipe all all memory
                except for First$boot$cmd. After it, go restart CMX
              */
              disable;
              if CTresult = 0 then
                CTresult = CQ$ram$test;
              Ran$RT = TRUE;
              call CQCBinit;
            end;
          end;  /* forever loop */

        end CQ$boot$task;
        $eject
```

```
            $if not Rv
            $endif

 358   1    end CB$bootstrap;


MODULE INFORMATION:

     CODE AREA SIZE     = 0451H   1105D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 004EH     78D
     MAXIMUM STACK SIZE = 002AH     42D
     1139 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```