```
//packmu pool.mb PoolMC.BR
//bldr pool pool2 poolsetup poolcheck PoolPutPt PoolMC.BR ReadPram
get "bcpl.head"

//incoming procedures
external [ LoadPackedRAM                //from ReadPram
           Col                  //from Pool2
           Setup;InitBall       //from PoolSetup
           CheckCollisions      //from PoolCheck
           Moveball;Abs         //from PoolPutPt
         ]

//outgoing statics
external [ XPos;YPos;XVel;YVel;Mass;Radius;R;Ball;XPosold
           YPosold;Balls;x;y
         ]
static [ XPos=nil       // x position
         YPos=nil       // y position
         XVel=nil       // x velocity
         YVel=nil       // y velocity
         Mass=1 // ball mass
         Radius=5       // ball radius
         R=50           // coefficient of restitution times 64
         x=160          // x offset
         y=100          // y offset
         XPosold=nil    // old x position
         YPosold=nil    // old y position
         Ball=nil       // bit pattern for ball
         Balls=15       // number of balls
       ]

//incoming statics
external [ Screen
           RamImage
         ]

//internal statics
static [ Ball1;Ball2;Ball3;Ball4;Ball5;Ball6;Ball7;Ball8;Ball9;Ball10
         Ball11;Ball12;Ball13;Ball14;Ball15     //bit patterns pre-shifted
         BallSelect
         DelayCount=50
         MinBallsInMotion=5
       ]

let Start() be
  [ LoadPackedRAM(RamImage)
    let RamCall=table [ #61010;#1401]   //JMPRAM;JMP 1,3
    let xp=vec(16)
    let yp=vec(16)
    let xv=vec(16)
    let yv=vec(16)
    let xo=vec(16)
    let yo=vec(16)
    let count=vec(16);Zero(count,16)
    let ball=vec(16);Ball=ball
    Ball!0=table  [ #370;#1406;#1002;#2001;#2001;#2001;#2001;#2001;#1002;#1406;#3
                                        **70]
    Ball!1=table  [ #370;#1406;#1002;#2141;#2041;#2041;#2041;#2161;#1002;#1406;#3
                                        **70]
    Ball!2=table  [ #370;#1406;#1002;#2161;#2021;#2161;#2101;#2161;#1002;#1406;#3
                                        **70]
    Ball!3=table  [ #370;#1406;#1002;#2161;#2021;#2161;#2021;#2161;#1002;#1406;#3
                                        **70]
    Ball!4=table  [ #370;#1406;#1002;#2241;#2241;#2371;#2041;#2041;#1002;#1406;#3
                                        **70]
```

```
Ball!5=table  [ #370;#1406;#1002;#2161;#2101;#2161;#2021;#2161;#1002;#1406;#3
                                          **70]
Ball!6=table  [ #370;#1406;#1002;#2161;#2101;#2161;#2121;#2161;#1002;#1406;#3
                                          **70]
Ball!7=table  [ #370;#1406;#1002;#2161;#2021;#2021;#2021;#2021;#1002;#1406;#3
                                          **70]
Ball!8=table  [ #370;#1776;#1776;#3617;#3657;#3617;#3657;#3617;#1776;#1776;#3
                                          **70]
Ball!9=table  [ #370;#1406;#1002;#2161;#2121;#2161;#2021;#2021;#1002;#1406;#3
                                          **70]
Ball!10=table [ #370;#1406;#1002;#2271;#2251;#2251;#2251;#2271;#1002;#1406;#3
                                          **70]
Ball!11=table [ #370;#1406;#1002;#2121;#2121;#2121;#2121;#2121;#1002;#1406;#3
                                          **70]
Ball!12=table [ #370;#1406;#1002;#2271;#2211;#2271;#2241;#2271;#1002;#1406;#3
                                          **70]
Ball!13=table [ #370;#1406;#1002;#2271;#2211;#2271;#2211;#2271;#1002;#1406;#3
                                          **70]
Ball!14=table [ #370;#1406;#1002;#2251;#2251;#2271;#2211;#2211;#1002;#1406;#3
                                          **70]
Ball!15=table [ #370;#1406;#1002;#2271;#2241;#2271;#2211;#2271;#1002;#1406;#3
                                          **70]

InitBall(lv Ball1,1)
InitBall(lv Ball2,2)
InitBall(lv Ball3,3)
InitBall(lv Ball4,4)
InitBall(lv Ball5,5)
InitBall(lv Ball6,6)
InitBall(lv Ball7,7)
InitBall(lv Ball8,8)
InitBall(lv Ball9,9)
InitBall(lv Ball10,10)
InitBall(lv Ball11,11)
InitBall(lv Ball12,12)
InitBall(lv Ball13,13)
InitBall(lv Ball14,14)
InitBall(lv Ball15,15)

let bs=vec 16
BallSelect=bs
BallSelect!0=Ball15
BallSelect!1=Ball14
BallSelect!2=Ball13
BallSelect!3=Ball12
BallSelect!4=Ball11
BallSelect!5=Ball10
BallSelect!6=Ball9
BallSelect!7=Ball8
BallSelect!8=Ball7
BallSelect!9=Ball6
BallSelect!10=Ball5
BallSelect!11=Ball4
BallSelect!12=Ball3
BallSelect!13=Ball2
BallSelect!14=Ball1
BallSelect!15=Ball
```

```
XPos=xp
YPos=yp
XVel=xv
YVel=yv
XPosold=xo
YPosold=yo
let mbs=nil
let velflag=0
let speed=0
let vmax=0
Setup(BallSelect)

[ Wl("Push top button to place cue ball")
  let frct=1
    [ mbs=((not @#177030)&7)
      switchon mbs into        // place cue ball
          [ default: loop
            case 4: let x1= (@#424-x)*64
                    let y1= (@#425-y)*64
                    Moveball(0,x1,y1)
                    XPos!0=x1
                    YPos!0=y1
                    XPosold!0=x1
                    YPosold!0=y1
                    XVel!0=0
                    YVel!0=0
          ]
    ] repeatuntil mbs eq 4

  [ mbs=((not @#177030)&7)] repeatuntil mbs eq 0
  Wl("Point arrow in direction of velocity")
  Wl("Push top button for high speed, middle for medium speed")
  Wl("and bottom for slow speed")

  [ mbs=((not @#177030)&7)
    if (mbs ne 0)&(XVel!0 eq 0)&(YVel!0 eq 0) then
      [ speed=selecton mbs into        // determine velocity
           [ case 1: 128
             case 2:  64
             case 4:  256
           ]
        let chx=(@#424)-(XPos!0 rshift 6)-x   // get mouse coordinates
        let chy=(@#425)-(YPos!0 rshift 6)-y
        while (Abs(chx) gr 64)%(Abs(chy) gr 64) do
           [ chx=chx/2
             chy=chy/2
           ]

         let sx=(chx gr 0)?1,-1
         let sy=(chy gr 0)?1,-1

         test (Abs(chx) gr Abs(chy)) ifso // determine which has larger magni
                                          **tude
           [ XVel!0=speed*sx              // chx has larger magnitude
             YVel!0=(speed*sx*chy)/chx
           ]
         ifnot                          // chy has larger magnitude
           [ YVel!0=speed*sy
             XVel!0=(speed*sy*chx)/chy
           ]
      ]
```

```
      velflag=0
      for n=0 to Balls do              // begin loop to check each ball
          [ if XPos!n eq 0 then loop      // skip if ball isn't on table
            let XVelN=XVel!n
            let YVelN=YVel!n
            if (XVelN eq 0) & (YVelN eq 0) then loop
CheckXCushion:
            if (XPos!n le 1728 & XVelN ls 0) %
                (XPos!n ge 16448 & XVelN gr 0) then
                   [ test (YPos!n le 2304)%(YPos!n ge 31168)%
                     ((YPos!n ge 16256)&(YPos!n le 17280)) ifso
                        [ Moveball(n,XPosold!n,YPosold!n)
                          XPos!n=0;YPos!n=0
                          loop
                        ]
                     ifnot XVelN = -(XVelN)
                   ]
CheckFriction:
            if frct eq 0 then
                [ test Abs(XVelN) gr Abs(YVelN) ifso
                    vmax=(Abs(XVelN) rshift 5)+56
                  ifnot vmax=(Abs(YVelN) rshift 5)+56
                  if XVelN ne 0 then XVelN=(vmax*(XVelN))/66
                  if YVelN ne 0 then YVelN=(vmax*(YVelN))/66
                  if (Abs(XVelN) le 1)&(Abs(YVelN) le 1) then
                    [ XVel!n=0
                      YVel!n=0
                      loop
                    ]
                ]
CheckYCushion:
            if (YPos!n le 1728 & YVelN ls 0) %
               (YPos!n ge 31872 & YVelN gr 0) then
                  [ test (XPos!n le 1900)%(XPos!n ge 16200) ifso
                       [ Moveball(n,XPosold!n,YPosold!n)
                         XPos!n=0;YPos!n=0
                         loop
                       ]
                    ifnot YVelN=-YVelN  // bounce off cushion
                  ]
            velflag=velflag+1
            XVel!n=XVelN;YVel!n=YVelN

            CheckCollisions(n)

            if XVel!n eq 0 & YVel!n eq 0 then loop     // if velocity = 0 don't b
                                        **other to change position

CheckChange:
            let newposx = (XPos!n)+(XVel!n)     // calculate new x position
            let newposy = (YPos!n)+(YVel!n)     // calculate new y position
            if (newposx&#177700) ne ((XPos!n)&#177700) %        // if actual posi
                                        **tion changes moveball
                 (newposy&#177700) ne ((YPos!n)&#177700) then
               [ Moveball(n,newposx,newposy)
                 Moveball(n,XPosold!n,YPosold!n)
                 XPosold!n=newposx
                 YPosold!n=newposy
               ]

            XPos!n=newposx
            YPos!n=newposy
          ]     //end of "for n=0 to Balls do"
 for i=velflag to MinBallsInMotion do
    for j=1 to DelayCount do Idle()
```

```
        frct = frct + 1
        if frct ge 30 then frct=0
    ] repeatuntil (XPos!0 eq 0)&(velflag eq 0)
  ] repeat
]
```

```
get "bcpl.head"

external [ XPos;YPos;XVel;YVel;Mass;Radius;R;Ball;Col ]

let Col(n,m) be
  [ let Rshft=table [ #61010;#1401]
    let V1xit,V1yit,V1xip,V1yip,V1xfp,V1yfp=nil,nil,nil,nil,nil,nil
    let V2xit,V2yit,V2xip,V2yip,V2xfp,V2yfp=nil,nil,nil,nil,nil,nil
    let sin=(YPos!n-YPos!m)/(Radius*2)
    let cos=(XPos!n-XPos!m)/(Radius*2)
    let V1xi,V1yi,V2xi,V2yi=XVel!n,YVel!n,XVel!m,YVel!m
    let Vxc,Vxs,Vyc,Vys,Vxsc,Vysc=nil,nil,nil,nil,nil,nil
    Vxc=Rshft(V1xi*cos,6)
    Vxs=Rshft(V1xi*sin,6)
    Vyc=Rshft(V1yi*cos,6)
    Vys=Rshft(V1yi*sin,6)
    Vysc=Rshft(Vys*cos,6)
    Vxsc=Rshft(Vxs*cos,6)
    V1xit=Rshft(Vxs*sin,6)-Vysc
    V1yit=Rshft(Vyc*cos,6)-Vxsc
    V1xip=Rshft(Vxc*cos,6)+Vysc
    V1yip=Rshft(Vys*sin,6)+Vxsc
    Vxc=Rshft(V2xi*cos,6)
    Vxs=Rshft(V2xi*sin,6)
    Vyc=Rshft(V2yi*cos,6)
    Vys=Rshft(V2yi*sin,6)
    Vysc=Rshft(Vys*cos,6)
    Vxsc=Rshft(Vxs*cos,6)
    V2xit=Rshft(Vxs*sin,6)-Vysc
    V2yit=Rshft(Vyc*cos,6)-Vxsc
    V2xip=Rshft(Vxc*cos,6)+Vysc
    V2yip=Rshft(Vys*sin,6)+Vxsc
    V1xfp=V1xip*Mass-Rshft(V1xip*R,6)*Mass
    V1xfp=V1xfp+Rshft(V2xip*(R+64),6)*Mass
    V1xfp=V1xfp/(Mass+Mass)
    V1yfp=V1yip*Mass-Rshft(V1yip*R,6)
    V1yfp=V1yfp+Rshft(V2yip*(R+64),6)*Mass
    V1yfp=V1yfp/(Mass+Mass)
    V2xfp=Rshft(V1xip*(R+64),6)*Mass
    V2xfp=V2xfp+V2xip*Mass-Rshft(V2xip*R,6)*Mass
    V2xfp=V2xfp/(Mass+Mass)
    V2yfp=Rshft(V1yip*(R+64),6)*Mass
    V2yfp=V2yfp+V2yip*Mass-Rshft(V2yip*R,6)*Mass
    V2yfp=V2yfp/(Mass+Mass)
    let chx=XPos!n - XPos!m
    let chy=YPos!n - YPos!m
    let chvx=V1xfp - V2xfp
    let chvy=V1yfp - V2yfp
    if (chvx eq 0)&(chvy eq 0) then return
    if ((chx xor chvx) ls 0)&((chy xor chvy) ls 0) then return
    XVel!n=V1xfp + V1xit
    YVel!n=V1yfp + V1yit
    XVel!m=V2xfp + V2xit
    YVel!m=V2yfp + V2yit
  ]

//and Rshft(divisor,shifts) =
//    valof [ test divisor gr 0 ifso resultis divisor rshift shifts
//              ifnot resultis -1*((-1*divisor)rshift shifts)
//          ]
```

```
//outgoing procedures
external CheckCollisions

//incoming routines
external Col

//incoming statics
external [ XPos;XVel;YPos;YVel
           Balls
         ]

let CheckCollisions(n) be
 [ let RamCall=table [ #61010;#1401]
   //note for RamCall: n is FP!4
   let m=Balls
   [ m=RamCall(m,2)
     if m ls 0 then return

//     let chy=YPos!n - YPos!m
//     let chx=XPos!n - XPos!m
//     let sx = (chx gr 0)?1,-1
//     let sy = (chy gr 0)?1,-1
//      [ let chvx=XVel!n-XVel!m
//        let chvy=YVel!n-YVel!m
//        let svx=(chvx gr 0)?1,-1
//        let svy=(chvy gr 0)?1,-1
//        if (sx ne svx) % (sy ne svy) then Col(n,m)
//      ]
Col(n,m)
     m=m-1
    ] repeat
 ]
```

```
;Microcode for Pool ball motion

;SYMBOL DEFINITIONS FOR ALTO

$MOUSE  $L0,14006,100;
$DISP$L0,14007,120;
$MD$L26006,14005,124100;
$DDR$L26010,0,124100;
$XPREG  $L26010,0,124000;
$CSR    $L26011,0,124000;
$TASK$L16002,0,0;
$BLOCK$L16003,0,0;
$MAR$L20001,0,144000;
$LLCY8$L0,22006,200;
$LRSH1$L0,22005,200;
$LLSH1$L0,22004,200;
$BUS=0$L24001,0,0;
$SH<0$L24002,0,0;
$SH=0$L24003,0,0;
$BUS$L24004,0,0;
$ALUCY$L24005,0,0;
$IDISP$L24015,0,0;
$BUSODD $L24010,0,0;
$LMRSH1 $L0,62005,200;  MAGIC RIGHT SHIFT
$LMLSH1 $L0,62004,200;  MAGIC LEFT SHIFT
$EVENFIELD$L24010,0,0;
$SETMODE$L24011,0,0;
$IR$L26014,0,124000;
$ACDEST$L30013,32013,60100;
$DNS$L30012,0,60000;
$ACSOURCE$L0,32016,100;
$L$L40001,36001,144200;
$HALT$L42001,0,0;
$BREAK$L42003,0,0;
$WENB$L42005,0,0;
$READY?$L42006,0,0;
$NOVA$L44002,46003,124100;
$ORT$L0,50002,2;
$ANDT$L0,50003,2;
$XORT$L0,50004,2;
$+1$L0,50005,2;
$-1$L0,50006,2;
$+T$L0,50007,2;
$-T$L0,50010,2;
$-T-1$L0,50011,2;
$+INCT$L0,50012,2;      SYNONYM FOR +T+1
$+T+1$L0,50012,2;
$+SKIP$L0,50013,2;
$IF$L52001,54001,124040;
$END$L34000,0,0;
$.T     $L0,50014,2;
$AND NOT T$L0,50015,2;
```

```
;DEFINITIONS FOR EMULATOR TASK
$STARTF $L16017,0,0;    NDF1=17
$RSNF           $L0,70016,100;  NDF1=16

;DEFINITIONS FOR CONTROL RAM
$SWMODE $L16010,0,0;            NDF1=10  (EMULATOR)
$WRTRAM $L16011,0,0;            NDF1=11
$RDRAM  $L16012,0,0;            NDF1=12

;DISK DEFINITIONS
$KSTAT  $L20012, 14003, 124100 ;        DF1=12 (LHS) BS=3 (RHS)
$RWC    $L24011, 0, 0;                  NDF2=11
$RECNO  $L24012, 0, 0;                  NDF2=12
$INIT   $L24010, 0, 0;                  NDF2=10
$CLRSTAT$L16014, 0, 0;                  NDF1=14
$KCOMM  $L20015, 0, 124000;             DF1=15 (LHS ONLY) REQUIRES BUS DEF
$SWRNRDY$L24014, 0, 0;                  NDF2=14
$KADR   $L20016, 0, 124000;             DF1=16 (LHS ONLY) REQUIRES BUS DEF
$KDATA  $L20017, 14004, 124100;         DF1=17 (LHS)  BS=4 (RHS)
$STROBE $L16011, 0, 0;                  NDF1=11
$NFER   $L24015, 0, 0;                  NDF2=15
$STROBON$L24016, 0, 0;                  NDF2=16
$XFRDAT $L24013, 0, 0;                  NDF2=13
$INCRECNO$L16013, 0, 0;                 NDF1=13
$SINK   $L44000, 0, 124000;             DF3=0  FAKE TO ALLOW BUS SOURCE WITH
;                                       NO DESTINATION
$NOP    $L42000, 0, 0;                  NDF3=0  ANOTHER FAKE


;***x14 change:  added mesa subroutine return constants sr20-37, and -13D

;***X13 CHANGE:  REPLACED CONSTANT MEMORY WITH LISP VERSION
;       THE ALTO CONSTANT MEMORY
;  constants.mc - special for Lisp microcode
;
;  last modified 28 mar 75 @ 13:40
;
$0      $L0,12000,100;  CONSTANT 0 IS SUPER-SPECIAL
$ALLONES4       $M4:177777;     CONSTANT NORMALLY ANDED WITH KSTAT
$ALLONES5       $M5:177777;     CONSTANT NORMALLY ANDED WITH MD
$M17    $M6:17; CONSTANT NORMALLY ANDED WITH MOUSE
$ALLONES7       $M7:177777;     CONSTANT NORMALLY ANDED WITH DISP
$M177770        $M7:177770;     MASK FOR DISP
$M7     $M7:7;  MASK FOR DISP
$X17    $M7:17;         MASK FOR DISP
$ONE    $1;             THE CONSTANT 1
$2      $2;
$-2     $177776;        - DISK HEADER WORD COUNT
$3      $3;
$4      $4;
$5      $5;
$6      $6;
$7      $7;
$10     $10;
$-10    $177770;        - DISK LABEL WORD COUNT
$17     $17;
$20     $20;
$37     $37;
$ALLONES        $177777;        THE REAL -1 (NOT A MASK)
$40     $40;
$77     $77;
$100    $100;
$177    $177;
$200    $200;
$377    $377;
```

```
$177400$177400;
$-400   $177400;        - DISK DATA WORD COUNT
$2000   $2000;
$PAGE1  $400;
$DASTART$420;           MAIN MEMORY DISPLAY HEADER ADDRESS
$KBLKADR$521;           MAIN MEMORY DISK BLOCK ADDRESS
$MOUSELOC$424;          MAIN MEMORY MOUSE BLOCK ADDRESS
$CURLOC $426;           MAIN MEMORY CURSOR BLOCK ADDRESS
$CLOCKLOC$430;
$CON100 $100;
$CADM   $7772;          CYLINDER AND DISK MASK
$SECTMSK$170000;        SECTOR MASK
$SECT2CM$40000;         CAUSES ILLEGAL SECTORS TO CARRY OUT
$-4     $177774;        CURRENTLY UNUSED
$177766 $177766;        CURRENTLY UNUSED
$177753 $177753;        CURRENTLY UNUSED
$TOTUWC $44000;         NO DATA TRANSFER, USE WRITE CLOCK
$TOWTT  $66000;         NO DATA TRANSFER, DISABLE WORD TASK
$STUWC  $4000;          TRANSFER DATA USING WRITING CLOCK
$STRCWFS$10000;         TRANSFER DATA USING NORMAL CLOCK, WAIT FOR SYNC
$177000 $177000;
$77777  $77777;
$77740  $77740;
$LOW14  $177774;
$77400  $77400;
$-67D   $177675;
$7400   $7400;
$7417   $7417;
$170360 $170360;
$60110  $60110;
$30000  $30000;
$70531  $70531;
$20411  $20411;
$65074  $65074;
$41023  $41023;
$122645 $122645;
$177034 $177034;
$37400  $37400;
$BIAS   $177700;        CURSOR Y BIAS
$WWLOC  $452;   WAKEUP WAITING IN PAGE 1
$PCLOC  $500;   PC VECTOR IN PAGE 1
$100000 $100000;
$177740 $177740;
$COMERR1        $277;   COMMAND ERROR MASK
$-7     $177771;        CURRENTLY UNUSED
$177760 $177760;
$-3     $177775;
$4560   $4560;
$56440  $56440;
$34104  $34104;
$64024  $64024;
$176000 $176000;
$177040 $177040;
$177042 $177042;
$203    $203;
$360    $360;
$177600 $177600;
$174000 $174000;
$160000 $160000;
$140000 $140000;
$777    $777;
$1777   $1777;
$3777   $3777;
$7777   $7777;
$17777  $17777;
```

```
$37777  $37777;
$1000   $1000;
$20000  $20000;
$40000  $40000;
$-15D   $177761;
$TRAPDISP       $526;
$TRAPPC $527;
$TRAPCON        $470;
$JSRC   $6000;  JSR@ 0
$MASKTAB        $460;   MASK TABLE STARTING ADDRESS FOR CONVERT
$SH3CONST       $14023; DESTINATION=3, SKIP IF NONZERO CARRY, BASE CARRY=0

$EIPOSTLOC      $600;   ETHERNET CONSTANTS
$EIIBIT $601;
$EOPOSTLOC      $602;
$EOIBIT         $603;
$EIEOTLOC       $604;
$EOEOTLOC       $605;
$EOLOADLOC      $606;
$EISERLOC       $607;
$EIBCLOC        $610;
$EOBCLOC        $612;

$ITQUAN         $422;
$ITIBIT         $423;
$402    $402;           LOCATION INTO  WHICH THE LABEL BLOCK WILL BE STORED ON BO
                                **OT
$M177760        $M7:177760;     MASK FOR DISP.  FOR I/O INSTRUCTIONS
$JSRCX  $4000;  JSR 0
$KBLKADR2       $523;
$KBLKADR3       $524;

$MFRRDL $177757;        DISK HEADER READ DELAY IS 21 WORDS
$MFROBL $177744;        DISK HEADER PREAMBLE IS 34 WORDS
$MIRRDL $177774;        DISK INTERRECORD READ DELAY IS 4 WORDS
$MIROBL $177775;        DISK INTERRECORD PREAMBLE IS 3 WORDS
$MRPAL  $177775;        DISK READ POSTAMBLE LENGTH IS 3 WORDS
$MWPAL  $177773;        DISK WRITE POSTAMBLE LENGTH IS 5 WORDS
$BDAD   $12;    ON BOOT, DISK ADDRESS GOES IN LOC 12

$REFMSK$77740;
$X37$M7:37;     NOPAR MASK
$M177740$M7:177740;     DITTO
$EIALOC$177701; LOCATION OF EIA INPUT HARDWARE
;  constants for Lisp microcode
$7000   $7000;          mapbase
$176    $176;           mapmask
$177576 $177576;                mapmask3
$30     $30;            reprobinc
$15             $15;            wrt-1
$1770           $1770;          ciad
$101771         $101771;                cilow
$175777         $175777;        for resetting fbn
$11     $11;            just to have small integers
$13     $13;
$14     $14;
$16     $16;            for 2CODE
$60     $60;            low R to high R bus source
$776    $776;
$177577         $177577;        -129
$100777         $100777;
$177677         $177677;
$177714         $177714;        (-2fvar+14)
;
;  CONSTANTS ADDED BY LEO
```

```
$2527   $2527;
$101    $101;
$630    $630;
$631    $631;
$642    $642;
$lgm1   $M7:1;
$lgm3   $M7:3;
$lgm10  $M7:10;
$lgm14  $M7:14;
$lgm20  $M7:20;
$lgm40  $M7:40;
$lgm100 $M7:100;
$lgm200 $M7:200;
;
;  add new constants below this line only!!!
;
$disp.300               $M7:300;
$-616           $177162;
$-650           $177130;
$22             $22;
$24             $24;
$-20            $177760;
$335            $335;           endcode for getframe
$1377           $1377;          smallnzero
$401            $401;
$2001           $2001;
$21             $21;            just to have them
$23             $23;
$25             $25;
$26             $26;
$27             $27;
$31             $31;
$1675           $1675;
$736            $736;
$-660           $177120;
$300            $300;
$disp.377               $M7:377;
$6001           $6001;          f.e. flg,quick flg, use count
$disp.3         $M7:3;

;***x13 change declares the following constants for subroutine returns.
;some are added
$sr1    $60110;
$sr0    $70531;
$sr2    $61000; added
$sr3    $61400; added
$sr4    $62000; added
$sr5    $62400; added
$sr6    $67000; added-value of 16b mapped to 6 by disp prom
$sr7    $63400; added
$sr10   $64024;
$sr11   $64400; added
$sr12   $65074;
$sr14   $66000; added
$sr15   $66400; added
$sr16   $63000; added- value of 6 mapped to 16b by disp prom
$sr17   $77400;
```

```
;new constants sr20-sr37 use the sr13 IDISP slot to 'return' to a
;subroutine which does: SINK←DISP,BUS; return20;
;added 23 oct 75. by CPT

$sr20    $65400;
$sr21    $65401;
$sr22    $65402;
$sr23    $65403;
$sr24    $65404;
$sr25    $65405;
$sr26    $65406;
$sr27    $65407;
$sr30    $65410;
$sr31    $65411;
$sr32    $65412;
$sr33    $65413;
$sr34    $65414;
$sr35    $65415;
$sr36    $65416;
$sr37    $65417;
$-13D    $177763;

; Alto II constants added for compatibility - X23 - August 11, 1976
$177024 $177024;
$177025 $177025;
$177026 $177026;
$7774    $7774;

; New stuff added for X21 - May 10, 1976
; Re-arranged to correspond to Alto II order - August 11, 1976
$2377    $2377;  Added for changed Ethernet microcode
$2777    $2777;
$3377    $3377;
$477     $477;   Added for BitBlt
$576     $576;   Added for Ethernet boot
$177175 $177175;


;Dispatch definitions:

!17,20,Moveball,Init,CheckCollisions,Init2,Init3,,Rshift6;
!20,1,START;     return address for emulator restart

;REGISTERS USED BY NOVA EMULATOR
$AC0     $R3;    ac's are backwards because the hardware supplies
                 ;   the complement address when addressing from ir
$AC1     $R2;
$AC2     $R1;
$AC3     $R0;
$NWW     $R4;
$SAD     $R5;
$PC      $R6;
$XREG    $R7;
$LastL   $R40; not a real S register, but rather L gated to the bus
;Clock (in refresh task) R11,R37
;Ethernet R12,R13
;Display controller: R20-R30
;Disk Controller: R31-R34

;Available: R5,R10,R14-17,R35-36

rett:    TASK;   most general return (Return&TASK)
retn:    NOP;    return, do nop first (prev inst has task)
ret:     SWMODE;
         :START; back to ROM
```

```
;---------------------------------------------------------------
;Rshift6(num,6) arithmetic right shift 6
;---------------------------------------------------------------
!1,2,pos6,neg6;

Rshift6: L←AC0;
        NOP,SH<0;
        T←0,:pos6;
neg6:   T←ONE;
pos6:   AC0←L MRSH 1;
        L←AC0;
        AC0←L MRSH 1;
        L←AC0;
        AC0←L MRSH 1;
        L←AC0;
        AC0←L MRSH 1;
        L←AC0;
        AC0←L MRSH 1;
        L←AC0;
        AC0←L MRSH 1,:rett;

;---------------------------------------------------------------
;Init(XPos,1,YPos)     //initialize S registers for XPos,YPos,set MinDist=11*64=7
                                          **04=#1300
;---------------------------------------------------------------
$XPos           $R50;
$YPos           $R51;
$MinDist        $R52;
$CollDist       $R53;

Init:   T←3;    YPos at AC2!3
        MAR←AC2+T;
        L←AC0;
        XPos←L;
        L←MD;
        YPos←L;

        T←100;
        L←200+T;
        T←1000;
        L←LastL+T,TASK;
        MinDist←L;

        T←200;  CollDist=13*64=#1500
        L←MinDist+T;
        CollDist←L,:rett;

;---------------------------------------------------------------
;Init2(XVel,3,YVel)     //and NWDS=18.=#22
;---------------------------------------------------------------
$XVel           $R54;
$YVel           $R55;
$NWDS   $R45;   words per scan line

Init2:  T←3;
        MAR←AC2+T;
        L←AC0;
        XVel←L;
        L←MD;
        YVel←L;

        T←2;
        L←20+T;
        NWDS←L,:rett;
```

```
;-------------------------------------------------------------
;Init3(BallSelect,4,Screen)
;-------------------------------------------------------------
$BallSelect     $R56;
$Screen $R57;
$ninety $R74;

Init3:  T←3;
        MAR←AC2+T;
        L←AC0;
        BallSelect←L;
        L←MD;
        Screen←L;

;90.=#132
        T←2;
        L←10+T;
        T←20;
        L←LastL+T;
        T←100;
        L←LastL+T;
        ninety←L,:rett;

;-------------------------------------------------------------
;MoveBall(n,xa,ya) XOR's 11 or 22 words of ball pattern to screen
;-------------------------------------------------------------
$ya             $R16;   for shift
$xWord  $R17;   for shift
$ya4            $R36;   for shift
$count          $R41;   counter for number of words to put out
$ScreenBits     $R42;   previous screen contents
$BallBits       $R43;   bits to XOR onto screen
$next           $R44;   temp to store next ScreenAddr in
;$NWDS  $R45;   words per scan line
$locat          $R46;
$bits           $R47;
$ScreenAddr     $R76;
$n              $R75;
$BallVec        $R73;

;MoveBall(n,xa,ya)
;and Moveball(n,xa,ya) be
;  [ let Rshift=table [ #61010;#1401]
;    let xWord=Rshift(xa,6)
;    let locat=18*Rshift(ya,6)+(xWord rshift 4)
;    let bits=(xWord&#17)-11
;    if bits ls 0 then [ locat=locat-1;bits=bits+16]
;    let BallVec=(BallSelect!bits)!n
;    let ScreenAddr=Screen+1+locat-18*5

;    let RamCall=table [ #61010;#1401]
;    RamCall(ScreenAddr,0,BallBits)
;    if bits ls 10 then RamCall(ScreenAddr-1,0,BallBits+11)
; ]
```

```
;n in AC0, xa in AC3, ya in AC2+3
Moveball:       T←3;
        MAR←AC2+T;
        L←AC0;
        n←L;
        T←BIAS; BIAS=177700
        L←MD AND T;
        ya←L RSH 1;
        L←ya,TASK;
        ya4←L RSH 1;    (ya rshift 6)*16
        L←ya4,TASK;
        ya←L RSH 1;
        L←ya,TASK;
        ya←L RSH 1;
        L←ya,TASK;
        ya←L RSH 1; (ya rshift 6)*2
        T←ya4;
        L←ya+T,TASK;
        ya←L;   //(ya rshift 6)*18

;       let xWord=xa rshift 6
        L←AC3,TASK;
        AC3←L RSH 1;
        L←AC3,TASK;
        AC3←L RSH 1;
        L←AC3,TASK;
        AC3←L RSH 1;
        L←AC3,TASK;
        AC3←L RSH 1;
        L←AC3,TASK;
        AC3←L RSH 1;
        L←AC3,TASK;
        AC3←L RSH 1;
        L←AC3,TASK;
        AC3←L RSH 1;    //xa rshift 6

;    let locat=18*Rshift(ya,6)+(xWord rshift 4)

        L←AC3,TASK;
        xWord←L RSH 1;
        L←xWord,TASK;
        xWord←L RSH 1;
        L←xWord,TASK;
        xWord←L RSH 1;
        L←xWord;
        xWord←L RSH 1;  xWord rshift 4
        T←ya;
        L←xWord+T,TASK;
        locat←L;

;    let bits=(xWord&#17)-11

!1,2,bitsPositive,bitsNegative;

        T←17;
        L←AC3 AND T;
        T←BDAD+1;
        L←LastL-T;
        bits←L,SH<0;

;       if bits ls 0 then [ locat=locat-1;bits=bits+16]

        L←locat-1,:bitsPositive;        ***bitsPositive,bitsNegative
bitsNegative: locat←L;
        T←20;
        L←bits+T;
        bits←L;
```

```
;    let BallVec=(BallSelect!bits)!n

bitsPositive:   T←bits;
        MAR←BallSelect+T;
        T←n;
        L←MD;

        MAR←LastL+T;
        NOP;
        L←MD,TASK;
        BallVec←L;
        NOP;            next instruction uses ALU←SReg after TASK

;    let ScreenAddr=Screen+1+locat-18*5 //18*5=90=#132

        L←Screen+1;
        T←locat;
        L←LastL+T;
        T←ninety;
        L←LastL-T,TASK;
        ScreenAddr←L;

!1,2,loop,break;

        L←BDAD; BDAD=12 (10.)
        count←L;

        L←BallVec;
        AC1←L;

        L←ScreenAddr;
        AC0←L;

loop:   MAR←T←AC0;      ScreenAddr
        L←NWDS+T;
        next←L;
        L←MD;

        MAR←AC1;
        ScreenBits←L;
        L←AC1+1;
        AC1←L;
        L←MD,TASK;
        BallBits←L;

        L←next;
        T←ScreenBits;
        MAR←AC0;
        AC0←L;
        L←BallBits XOR T;
        MD←LastL;

        L←count-1;
        count←L,SH<0;
        NOP,:loop;      ***loop,break
```

```
;    if bits ls 10 then RamCall(ScreenAddr-1,0,BallBits+11)

!1,2,Only11,Do22;

break:  T←BDAD; BDAD=#12=10.
        L←bits-T;
        NOP,SH<0;
        NOP,:Only11;     ***Only11,Do22;

Only11: NOP,:rett;

!1,2,loop2,break2;

Do22:   L←T←BDAD;        BDAD=12 (10.)
        count←L;

        L←BallVec+T+1;   //#12+1=#13=11.
        AC1←L;

        L←ScreenAddr-1;
        AC0←L;

loop2:  MAR←T←AC0;       ScreenAddr
        L←NWDS+T;
        next←L;
        L←MD;

        MAR←AC1;
        ScreenBits←L;
        L←AC1+1;
        AC1←L;
        L←MD,TASK;
        BallBits←L;

        L←next;
        T←ScreenBits;
        MAR←AC0;
        AC0←L;
        L←BallBits XOR T;
        MD←LastL;

        L←count-1;
        count←L,SH<0;
        NOP,:loop2;      ***loop2,break2;

break2: NOP,:rett;
```

```
;-----------------------------------------------------------
;CheckCollisions(m,2)   //m in AC0, n in AC2!4
;-----------------------------------------------------------
$XPosN  $R60;
$YPosN  $R61;
$chy            $R14;   for Shift
$chx            $R15;   for Shift
$m              $R62;
$sy             $R64;
$sx             $R65;
$dist           $R66;

CheckCollisions:        T←4;    offset of first param (n)
        MAR←L←AC2+T;
        L←AC0;
        m←L;
        L←T←MD;
        n←L;

        MAR←XPos+T;
        L←YPos+T;
        next←L;
        L←MD,TASK;
        XPosN←L;

        MAR←next;
        NOP;
        L←MD;
        YPosN←L;

;for m=m to 15 do
;     let chy=YPosN - YPos!m
;     if Abs(chy) gr 11*64 then loop    //too much y separation, no collision
;     let chx=XPosN - XPos!m
;     if Abs(chx) gr 11*64 then loop    //too much x separation, no collision
;    if m eq n then loop                //same ball

!1,2,MoreBalls,NoMoreBalls;
!1,2,pos1,neg1;
!1,2,pos2,neg2;
!1,2,YSep,NoYSep;
!1,2,XSep,NoXSep;
!1,2,PossCollision,NoCol;
!1,2,CheckDir,NoColl;
!1,2,CollY,NoCollY;
!1,2,CollX,NoCollX;

        T←m;
MoreBalls:      MAR←YPos+T;
        L←0;
        sy←L;
        T←MD;
        L←YPosN-T,TASK;
        chy←L;

        L←chy;
        T←ALLONES,SH<0;
        L←chy XOR T,:pos1;
neg1:   chy←L;
        L←100000;
        sy←L;
```

```
pos1:   T←MinDist;
        L←chy-T;
        NOP,SH<0;
        T←m,:YSep;      ***Ysep,NoYSep -- go to YSep if Abs(chy) gr 11*64 (i.e.,
                                          **loop)

NoYSep: MAR←XPos+T;
        L←0;
        sx←L;
        T←MD;
        L←XPosN-T,TASK;
        chx←L;

        L←chx;
        T←ALLONES,SH<0;
        L←chx XOR T,:pos2;
neg2:   chx←L;
        L←100000;
        sx←L;
pos2:   T←MinDist;
        L←chx-T;
        NOP,SH<0;
        T←n,:XSep;      go to XSep if Abs(chx) gr 11*64 (i.e., loop)

NoXSep: L←m-T;
                NOP,SH=0;
                NOP,:PossCollision;     ***PossCollision,NoCol;

NoCollX:        L←m-1,:cont;
NoColl: L←m-1,:cont;
NoCol:  L←m-1,:cont;
XSep:   L←m-1,:cont;
YSep:   L←m-1;
cont:   m←L,SH<0;
        T←m,:MoreBalls;
NoMoreBalls: AC0←L,:rett;

;     let dist = nil
;     test Abs(chx) gr Abs(chy) ifso dist=Abs(chx) + (Abs(chy)) rshift 1
;     ifnot dist = Abs(chy) + (Abs(chx)) rshift 1
;     if dist le 13*64 then     //collision
;      [ let chvx=XVel!n-XVel!m
;        let chvy=YVel!n-YVel!m
;        let svx=(chvx gr 0)?1,-1
;        let svy=(chvy gr 0)?1,-1
;        if (sx ne svx) % (sy ne svy) then Col(n,m)
;      ]

!1,2,BigX,BigY;
$tempVel        $R70;
$tempSign       $R71;

PossCollision:
        T←chy;
        L←chx-T;
        NOP,SH<0;
        NOP,:BigX;      ***BigX,BigY
BigX:   L←chy,TASK;
        dist←L RSH 1;
        T←dist;
        L←chx+T;
        dist←L,:HaveDist;
```

```
BigY:   L←chx,TASK;
        dist←L RSH 1;
        T←dist;
        L←chy+T;
        dist←L;
HaveDist:       T←dist;
        L←CollDist-T;           13*64-dist
        NOP,SH<0;               false if dist le 13*64
        L←m,:CheckDir;  ***CheckDir,NoColl

CheckDir:
        T←n;
        MAR←YVel+T;
        NOP;
        L←MD,TASK;
        tempVel←L;

        T←m;
        MAR←YVel+T;
        NOP;
        T←MD;
        L←tempVel-T;
        T←100000;       we only care about the sign bit
        L←LastL AND T;
        T←sy;
        L←LastL-T;
        NOP,SH=0;
        NOP,:CollY;     ***CollY,NoCollY;

NoCollY:        T←n;
        MAR←XVel+T;
        NOP;
        L←MD,TASK;
        tempVel←L;

        T←m;
        MAR←XVel+T;
        NOP;
        T←MD;
        L←tempVel-T;
        T←100000;
        L←LastL AND T;
        T←sx;
        L←LastL-T;
        NOP,SH=0;
        NOP,:CollX;     ***CollX,NoCollX;

CollY:  L←m,:DoCol;
CollX: L←m,:DoCol;
DoCol: AC0←L,:rett;
```

```
                ;  file  PutPt.AS


        AC0  = 0
        AC1  = 1
        AC2  = 2
        AC3  = 3


        .ENT  PutPt
        .ENT  Abs
        .ENT  ZeroPt
        .ENT  UGr
        .ENT  UGe
        .ENT  ULs
        .ENT  ULe
        .ENT    Moveball        ;set up for RamCall 0

        .BEXT  Screen



        .SREL
PutPt:  PUTPT
Abs:    ABSVAL
ZeroPt: ZEROPT
UGr:    UGR
UGe:    UGE
ULs:    ULS
ULe:    ULE
Moveball:       .Moveball


        .NREL


        ;Moveball(n,xa,ya)
.Moveball:
        STA 3,My3
        MOV 1,3
        SUB 1,1 ;AC1 has RamCall number (0)
        61010           ;JMPRAM
        LDA 3,My3
        JMP 1,3
My3:    0

        ; Abs(x)
        ;
        ;  returns the absolute value of x
        ;
ABSVAL:
        MOVL# AC0  AC0,SZC      ;negative ?
        NEG   AC0  AC0
        JMP   1,AC3
```

```
        ; UGr(x,y)
        ;
        ; x and y are unsigned 16 bit numbers.
        ; UGr(x,y) returns true if x greater than y; otherwise, false.
        ;
        ;
UGR:
        SGTU    AC0 AC1
        SUB     AC0 AC0 SKP
        ADC     AC0 AC0
        JMP     1 AC3



        ; UGe(x,y)
        ;
        ; x and y are unsigned 16 bit numbers.
        ; UGe(x,y) returns true if x greater or equal than y; otherwise, false.
        ;
        ;
UGE:
        SGEU    AC0 AC1
        SUB     AC0 AC0 SKP
        ADC     AC0 AC0
        JMP     1 AC3



        ; ULs(x,y)
        ;
        ; x and y are unsigned 16 bit numbers.
        ; ULs(x,y) returns true if x less than y; otherwise, false.
        ;
        ;
ULS:
        SLTU    AC0 AC1
        SUB     AC0 AC0 SKP
        ADC     AC0 AC0
        JMP     1 AC3



        ; ULe(x,y)
        ;
        ; x and y are unsigned 16 bit numbers.
        ; ULe(x,y) returns true if x less or equal than y; otherwise, false.
        ;
        ;
ULE:
        SLEU    AC0 AC1
        SUB     AC0 AC0 SKP
        ADC     AC0 AC0
        JMP     1 AC3
```

```
        ; ZeroPt(x,y)
        ;
        ;  clears appropriate point into Screen
        ;
ZEROPT:
        STA  AC3  ret
        STA  AC2  savefp
        STA  AC0  x
        MOVZR  AC0  AC0
        MOVZR  AC0  AC0
        MOVZR  AC0  AC0
        MOVZR  AC0  AC0          ; x rshift 4
        LDA  AC2  NWDS
        MUL
        MOVL# AC1 AC1,SZC        ; sword ls 0?
        JMP  return              ; yes - return
        LDA  AC0  ScreenMax
        SGE  AC0  AC1            ; sword gr ScreenMax?
        JMP  return              ; yes - return
        STA  AC1  sword
        LDA  AC0  x
        LDA  AC1  C017           ; #17
        AND  AC1  AC0            ;  x & #17
        SUB  AC0  AC1            ;  #17  - (x & #17)
        JSR  BITTABLE
        ADD  AC1  AC3            ;  bittable!(#17 - (x & #17))
        LDA  AC0  0,AC3
        LDA  AC2  @SCRN          ; Screen
        LDA  AC1  sword
        ADD  AC1  AC2            ; lv (Screen!sword)
        COM  AC0  AC3            ; not bit
        LDA  AC1  0,AC2          ; Screen!sword
        AND  AC3  AC1            ; & not bit
        STA  AC1  0,AC2          ; into Screen!sword
        JMP  return


        ; PutPt(x,y)
        ;
        ;  stores appropriate point into Screen
        ;
PUTPT:
        STA  AC3  ret
        STA  AC2  savefp
        STA  AC0  x
        MOVZR  AC0  AC0
        MOVZR  AC0  AC0
        MOVZR  AC0  AC0
        MOVZR  AC0  AC0          ; x rshift 4
        LDA  AC2  NWDS
        MUL
        MOVL# AC1 AC1,SZC        ; sword ls 0?
        JMP  return              ; yes - return
        LDA  AC0  ScreenMax
        SGE  AC0  AC1            ; sword gr ScreenMax?
        JMP  return              ; yes - return
        STA  AC1  sword
        LDA  AC0  x
        LDA  AC1  C017           ; #17
        AND  AC1  AC0            ;  x & #17
        SUB  AC0  AC1            ;  #17  - (x & #17)
        JSR  BITTABLE
        ADD  AC1  AC3            ;  bittable!(#17 - (x & #17))
        LDA  AC0  0,AC3
```

```
        LDA   AC2  @SCRN          ; Screen
        LDA   AC1  sword
        ADD   AC1  AC2            ; lv (Screen!sword)
        COM   AC0  AC3            ; not bit
        LDA   AC1  0,AC2          ; Screen!sword
        AND   AC3  AC1            ; & not bit
        ADD   AC0  AC1            ; + bit
        STA   AC1  0,AC2          ; into Screen!sword
return:
        LDA   AC3  ret
        LDA   AC2  savefp
        JMP   1,AC3

NWDS:           18.     ;Words per scanline
ScreenMax:      9468.   ;18.*526.
CO17:           17

SCRN:           Screen

ret:            0
x:              0
sword:          0
savefp:         0

BITTABLE:
        JSR   0,AC3               ; return with address of table in AC3
        1
        2
        4
        10
        20
        40
        100
        200
        400
        1000
        2000
        4000
        10000
        20000
        40000
        100000


        .END
```

```
get "bcpl.head"

//outgoing procedures
external [ Setup;InitBall;PutLine
         ]

//incoming procedures
external [ Moveball
           PutPt;Abs   //from PutPt (assembly coded)
         ]

//outgoing statics
external Screen
static Screen

//incoming statics
external [ XPos;XPosold;YPos;YPosold;XVel;YVel
           x;y;Balls;Ball
         ]

//internal manifest and structure declarations
manifest [ XLen=283;YLen=525
             WordsPerLine=18    //(283/16)=17; WordsPerLine must be even
             ScanLines=YLen+(YLen&1)    //must be even
           ]

let InitBall(BallVeclv,numShifts) be
 [ let ballvec=GetFixed(15)-1
   for i=0 to 15 do
    [ let ballbits=GetFixed((numShifts gr 5)?21,10)-1
      ballvec!i=ballbits
      let balli=Ball!i  //template of ith ball (11 word vector)
      for j=0 to 10 do ballbits!j=(balli!j) lshift numShifts
      if numShifts gr 5 then
        for j=11 to 21 do ballbits!j=(balli!(j-11)) rshift (16-numShifts)
     ]
   @BallVeclv=ballvec
 ]

and Setup(BallSelect) be
  [ let RamCall=table [ #61010;#1401]

    let topDCB=GetFixed(5)
    topDCB=topDCB+(topDCB&1)    //must be even

    Screen=GetFixed(WordsPerLine*ScanLines+5)
    let ScreenDCB=Screen+(Screen&1)     //must be even
    Screen=ScreenDCB+4
    Zero(Screen,WordsPerLine*ScanLines)

    let bottomDCB=GetFixed(5)
    bottomDCB=bottomDCB+(bottomDCB&1)   //must be even

    topDCB!0=ScreenDCB
    topDCB!1=0
    topDCB!2=0
    topDCB!3=y/2

    ScreenDCB!0=bottomDCB
    ScreenDCB!1=WordsPerLine+((x/16) lshift 8)
    ScreenDCB!2=Screen
    ScreenDCB!3=ScanLines/2
```

```
bottomDCB!0=@#420
bottomDCB!1=0
bottomDCB!2=0
bottomDCB!3=(700-(ScanLines+y))/2

RamCall(XPos,1,YPos)        //initialize S regs for XPos,YPos,MinDist
RamCall(XVel,3,YVel)
RamCall(BallSelect,4,Screen)

PutLine(0,0,282,0)                // draw outer border of pool table
PutLine(282,0,282,524)
PutLine(282,524,0,524)
PutLine(0,524,0,0)

PutLine(20,20,262,20)     // draw inner border of pool table
PutLine(262,20,262,504)
PutLine(262,504,20,504)
PutLine(20,504,20,20)

let pocket=254
while pocket le 270 do                    // draw side pockets
   [ PutLine(10,pocket,20,pocket)
     PutLine(262,pocket,272,pocket)
     pocket=pocket+1
   ]
let lx=10                                 // draw corner pockets
let rx=21
let incx=1
let line=10
let incy=1
CnPckt(lx,rx,line,incx,incy)
lx=261
rx=272
incx=-1
CnPckt(rx,lx,line,incx,incy)
lx=261
rx=272
incy=-1
line=514
CnPckt(rx,lx,line,incx,incy)
lx=10
rx=21
incx=1
CnPckt(lx,rx,line,incx,incy)
let ball=1                                  // draw rack
let xp=141*64
let yp=383*64
let clen=1
let cntr=1
let inc=64
while clen le 5 do
    [ while (cntr le clen)&(ball le Balls) do
        [ Moveball(ball,xp,yp)
          XPos!ball=xp
          XPosold!ball=xp
          YPos!ball=yp
          YPosold!ball=yp
          XVel!ball=0
          YVel!ball=0
          cntr=cntr+1
          ball=ball+1
          xp=xp+12*inc
        ]
      yp=yp+704
      xp=xp-6*inc
```

```
          inc=(-1)*inc
          clen=clen+1
          cntr=1
        ]
    @#420=topDCB
  ]

and CnPckt(x1,x2,line,incx,incy) be
  [ let linemark=line+(11*incy)
    while line ne linemark do
    [ PutLine(x1,line,x2,line)
      line=line+incy
      x2=x2+incx
    ]
    x2=x2-2*incx
    PutLine(x1,line,x2,line)
    line=line+incy
    x2=x2-incx
    x1=x1+incx
    linemark=line+(10*incy)
    while line ne linemark do
    [ PutLine(x1,line,x2,line)
      x1=x1+incx
      x2=x2-incx
      line=line+incy
    ]
  ]

and PutLine(x1,y1,x2,y2) be
 [ let hstep=x1-x2
   let vstep=y1-y2
   let hsign=(hstep ge 0)?1,-1
   let vsign=(vstep ge 0)?1,-1
   let absvstep=vsign*vstep
   let abshstep=hsign*hstep
   let count=0
   PutPt(x1,y1)

   until (Abs(x1-x2) le 1)&(Abs(y1-y2) le 1) do
    [ PutPt(x2,y2)
      test count ge absvstep then //go horizontal
        [ x2=x2+hsign
          count=count-absvstep
        ]
      or
        [ y2=y2+vsign
          count=count+abshstep
        ]
    ]
   PutPt(x2,y2)
  ]
```