```
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Module: Servo
;>
;>      This module contains all the specifications and source code for
;>      controlling the Widget Servo Board ( i.e., communication
;>      protocol, seek and head positioning, spare table lookup, etc. )
;>
;>      PROCEDURE OverLap( Response : BYTE { !rA }
;>                         CommandType : BYTE { !r9 }
;>                         BlockType : BYTE { !r8 }
;>                         BlockNumber : 3 BYTES { !rC:E }
;>                         Random      : BOOLEAN { !r7/Bit 7 }
;>                         Offset      : 3 BITS  { !r7/Bits 2:0 }
;>                        )
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Function: OverLap       { OverLapped Seek }
;>
;>      This function allows the drive to begin a seek operation
;>      ( if one is needed ) before getting tied up with the SOS
;>      driver.
;>
;>      Inputs:
;>              Response    : BYTE { !rA }
;>              CommandType : BYTE { !r9 }
;>              BlockType   : BYTE { !r8 }
;>              BlockNumber : 3 BYTES { !rC:E }
;>              Random      : BOOLEAN { !r7/bit 7 }
;>              Offset      : 3 BITS { !r7/bit 2:0 }
;>
;>      Outputs:
;>              BlockStatus  : BYTE { !r0 }
;>
;>      Local Variables:
;>              Retry   : BYTE { !r4 }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       IF ( Srch_Cache( BlockNumber, Random ) )
;>        THEN
;>         IF ( Srch_Cache.Head <> Head )
;>          THEN
;>              SelectHead( Srch_Cache.Head )
;>              Head := Srch_Cache.Head
;>         Sector := Srch_Cache.Sector
;>         Ack_Read
;>        ELSE
;>         Retry := 4
;>         Temp := False
;>         REPEAT
;>
```

```
;>                Retry := Retry - 1
;>                Temp := PositionHeads( NoWait, Dmt_OverLap, BlockNumber )
;>
;>          UNTIL NOT( Recovery ) OR Temp OR ( Retry = 0 )
;>
;>          IF NOT( Temp ) THEN Abort
;>          BlockStatus := PositionHeads.Status
;>          TempCyl := PositionHeads.Cylinder
;>          TempHead := PositionHeads.Head
;>          TempSector := PositionHeads.Sector
;>          IF ( CommandType = Write )
;>           THEN Get_Write_Data( Response )
;>           ELSE Ack_Read
;>          Load_Cache
;>          Seek( TempCyl, TempHead, TempSector )
;>       END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

OverLap:
                Call    Srch_Cache
                Jr      Nz,OvrLapSeek

                Tm      BlkStat,#CachHdChg ;check for a head change
                Jr      Z,Ovr_Ld_Sctr

                Xor     Head,#01 ;complement the head value

                Push    !r0 ;save Srch_Cache result
                 Ld     !rE,Head
                 Ld     !r0,#.HIBYTE. SelectHead
                 Ld     !r1,#.LOWBYTE. SelectHead
                Call    Bank_Call

                 Ld     !r0,#.HIBYTE. Load_Cache
                 Ld     !r1,#.LOWBYTE. Load_Cache
                Call    Bank_Call ;update the cache
                Pop     !r0 ;retrieve srch_cache result

Ovr_Ld_Sctr:    And     !r0,#$1F ;mask off all but sector info
                Ld      Sector,!r0
                Jr      OvrLp_End

OvrLapSeek:      Ld     !r0,#.HIBYTE. CnvrtLogical
                 Ld     !r1,#.LOWBYTE. CnvrtLogical
                Call    Bank_Call

                Ld      !r4,#4  ;Retry := 4

OvrLp_Lp:        Clr    !r8 ;no wait

                 Ld     !r0,#.HIBYTE. PosHeads
                 Ld     !r1,#.LOWBYTE. PosHeads
                Call    Bank_Call
                Jr      Nz,OvrLp_2

                Tm      Excpt_Status,#Recovery
```

```
                Jr      Z,OvrLp_1

                Djnz    !r4,OvrLp_Lp

OvrLp_1:         Clr    !r0 ;byte 0
                 Ld     !r1,#Stat_Srvo
                Call    SetStatus
                Call    Abort

OvrLp_2:         Ld     !r0,#.HIBYTE. Ext_Push
                 Ld     !r0,#.LOWBYTE. Ext_Push
                Call    Bank_Call
                 Ld     !r0,#.HIBYTE. Load_Cache
                 Ld     !r1,#.LOWBYTE. Load_Cache
                Call    Bank_Call ;update the cache
                 Ld     !r0,#.HIBYTE. Load_Logical
                 Ld     !r1,#.LOWBYTE. Load_Logical
                Call    Bank_Call
                Call    Srch_Cache ;make certain block is in cache
                Jr      Z,OvrLp_S_1

                Call    Abort

OvrLp_S_1:      Clr     Cache_Index
                 Ld     !r0,#.HIBYTE. Ext_Pop
                 Ld     !r1,#.LOWBYTE. Ext_Pop
                Call    Bank_Call
                 Ld     !r0,#.HIBYTE. Seek
                 Ld     !r1,#.LOWBYTE. Seek
                Call    Bank_Call

OvrLp_End:      Ret

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Procedure: Seek
;>
;>      This procedure accepts cylinder, head, and sector values
;>      and then calls position heads.
;>
;>      Inputs:
;>              Cylinder: WORD { !!rC }
;>              Head    : BYTE { !rE }
;>              Sector  : BYTE { !rF }
;>
;>      Outputs: { none }
;>
;>      Global Varibles Changed:
;>              Cylinder, Head, Sector
;>
;>      Algorithm:
;>
;>      BEGIN
;>       i := 4
;>       WHILE ( i > 0 ) AND NOT( PositionHeads( Wait,
;>                               Dmt_Seek, Cylinder, Head, Sector ) DO
;>
;>              i := i -1
;>
```

```
;>       IF ( i = 0 )
;>        THEN Abort
;>        ELSE
;>             GlobalCylinder := Cylinder
;>             GlobalHead     := Head
;>             GlobalSector   := Sector
;>             SectorCount := SectorCount + 1
;>             DiskStat.Parked := False
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

Seek:
                Ld      !r1,#2

Seek_Lp:        Push    !r1     ;save counter

                 Ld     !r8,#Wait
                 Ld     !r1,#Dmt_Seek
                Call    PosHeads

                Pop     !r1 ;get counter back
                Jr      Nz,Seek_End

                Push    !r1 ;save counter
                 Ld     !r2,#.HIBYTE. 20 ;wait 200 ms before retrying
                 Ld     !r3,#.LOWBYTE. 20
                Call    MsWait
                Pop     !r1 ;get counter back

                Djnz    !r1,Seek_Lp

                 Ld     !r0,#0 ;byte 0
                 Ld     !r1,#Stat_Srvo
                Call    SetStatus
                Call    Abort

Seek_End:       Or      DiskStatus,#On_Track
                Ld      Cylinder,!rC
                Ld      Cylinder+1,!rD
                Ld      Sector,!rF
                Incw    SeekCount
                And     DiskStat,#$FF-Parked

                Jp      Bank_Ret

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Function: PosHeads  { Position Heads }
;>
;>      This function is responsible for postioning the heads of
;>      the drive. There are two ways of positioning the heads via this
;>      routine: 1) Specify a logical block number ( where all the
```

```
;>      USER accessable blocks are numbered from 0 to n-1 ) or 2)
;>      by specifying the Cylinder, and Head number of any track
;>      on the disk surface.
;>
;>      The way that PositionHeads interprets the block number is
;>      controlled by the input parameter Logical ( i.e., IF Logical
;>      THEN BlockNumber is a Logical BlockNumber ELSE BlockNumber
;>      is Cylinder and Head ).
;>
;>      The Spare Table is searched ONLY if Logical is true, this is
;>      consistent with allowing a diagnostic program to get to any
;>      spot on the disk independent of the data stored there.
;>
;>      Inputs:
;>              Wait        : BOOLEAN { !r8/bit 6 }
;>              Parent      : BYTE    { !rB }
;>              Cylinder    : WORD { !!rC }
;>              Head        : BYTE { !rE }
;>              Sector      : BYTE { !rF }
;>
;>      Outputs:
;>              PositionHeads : BOOLEAN { Zero Flag; i.e., Zero is True if
;>                                        PositionHeads is True }
;>      Global Variables Changed:
;>              Cylinder, Head, Sector, Cur_Cyl
;>
;>      Local Variables:
;>              Seek, PositionDone : BOOLEAN { !r4, !r3 }
;>              Cyl, Magnitude     : WORD    { ( !rC, !rD ), ( !r5, !r6 }
;>              Head, Sector,                { !rE, !rF }
;>              PosRetry, SioRetry,          { !r7, !rA }
;>              Direction          : BYTE    { !r9 }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       SetDeadManTimer( PositionHeads, Parent )
;>       SelectHead( Head )
;>       Seek := CalcMagnitudeDirection( Cyl, Magnitude, Direction )
;>       IF ( GlobalHead <> Head ) AND DiskStatus.Write THEN Seek := True
;>       GlobalHead := Head
;>       GlobalSector := Sector
;>       PosRetry := 2
;>       REPEAT
;>        IF Seek AND ServoOk( RecalcMagAndDir { Boolean returned in !r0 } )
;>         THEN
;>              IF RecalcMagAndDir THEN Seek := CalcManitudeDirection( Cyl.
;>                                              Magnitude, Direction )
;>              SioRetry := 2
;>              WHILE ( SioRetry > 0 ) AND
;>                 NOT( ServorStore( AccessVar+Direction+
;>                                   Magnitude[ 1 ], Magnitude[ 2 ],
;>                                   AutoOffset, 0 ) DO
;>                   IF Recovery
;>                    THEN
;>                         ResetServo
;>                         SioRetry := SioRetry - 1
;>                    ELSE
;>                         SioRetry := 0
;>              IF Wait AND ( SioRetry > 0 )
;>               THEN WHILE NOT( ServoReady ) DO BEGIN END
;>          PositionDone := NOT( ServoError )
;>          PosRetry := PosRetry - 1
```

```
;>       UNTIL NOT( Recovery ) OR PositionDone OR ( PosRetry = 0 )
;>       ClearDeadManTimer
;>       IF PositionDone THEN Cur_Cyl := LocalCylinder
;>       PositionHeads := PositionDone
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

PosHeads:
                Call    Ext_Push ;save callers variables

                 Ld     !r0, #Dmt_PosHeads
                Call    Set_Dmt                 ;set Dead Man Timer

Clc_MagDir:     Call    CalcMagDir              ;get relative position

                Cp      Head,!rE        ;IF GlobalHead <> Head..
                Jr      Z,P_Ld_Retry
                Tm      DiskStat,#Wr_Op
                Jr      Z,P_Ld_SelHead
                Ld      !r4,#1

P_Ld_SelHead:   Call    SelectHead

P_Ld_Retry:     Ld      Sector,!rF

                Ld      !r7,#2 ;Retry := 2

PosHds_Rpt:     Or      !r4,!r4 ;test for Seek On
                Jr      Z,PosHds_4      ;if no seek then get out
                Call    ServoOk ;test if Servo is in a reasonable state
                Jr      Z,PosHds_3 ;if not ok then retry
                Tm      DiskStatus,#On_Track ;check for recalc of magnitude and direction
                Jr      Nz,PosHds_While
                Call    CalcMagDir

PosHds_While:    Ld     !r0,Seek_Type
                 Or     !r0,!r9 ; Plus Direction
                 Or     !r0,!r5 ; Plus Ms Magnitude
                 Ld     !r1,!r6 ; Ls Magnitude
                 Ld     !r2,#Off_Auto
                 Ld     !r3,#S_Rate_57_6
                Call    ServoCmnd
                Jr      Nz,PosHds_4

                Tm      Excpt_Status,#Recovery ;IF recovery
                Jr      Z,PosHds_5

                Call    ResetServo
                Call    UpDate_Hdr
                Call    CalcMagDir
                Djnz    !rA,PosHds_While

PosHds_4:       Call    LoadStatus      ;sample ServoError
                Tm      !r0,#ServoErr
                Jr      Z,PosHds_3      ;if servo error then retry
                Djnz    !r7,PosHds_Rpt
```

```
                Jr      PosHds_5

PosHds_3:       Tm      !r8,#Wait       ;IF Wait AND NOT( ServoRdy )
                Jr      Z,PosHds_5
                Tm      !r0,#ServoRdy   ; THEN loop until timeout OR ServoRdy
                Jr      Z,PosHds_4

PosHds_5:       Call    Clr_Dmt
                Call    LoadStatus      ;sample ServoError
                Tm      !r0,#ServoErr   ;check if position done
                Jr      Nz,PosHds_6

                Ld      Cur_Cyl,!rC
                Ld      Cur_Cyl+1,!rD

PosHds_6:       Push    !r0 ;save status
                Call    Ext_Pop ;get caller's variables back
                Pop     !r0
                Tcm     !r0,#ServoErr   ;return Z = true if servo error
                Ret

SelectHead:     Or      !rE,!rE ;test for Head0 or Head1
                Jr      Z,Sel_Head0
                Or      Port3,Hs0       ;select head 1
                Jr      SelHd_End

Sel_Head0:      And     Port3,#$FF-Hs0   ;select head 0
SelHd_End:      Ld      Head,!rE

                Jp      Bank_Ret

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Function: CalcMagDir  { Calculate Magnitude and Direction }
;>
;>      This function takes the current cylinder number ( from the
;>      conversion of the logical block ) and generates a magnitude
;>      and direction from the the position that the heads are
;>      currently at ( the servo needs to know a RELATIVE distance,
;>      not an absolute distance ).
;>
;>      Inputs:
;>              LocalCylinder : WORD { !rC, !rD }
;>
;>      Outputs:
;>              CalcMagDir : BOOLEAN { !r4 }
;>              Magnitude : WORD { !r5, !r6 }
;>              Direction : BYTE { !r9 }
;>
;>      Local Variables :
;>              Temp : BOOLEAN { }
;>              GlobalPTR : PTR { !r2, !r3 }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       IF LocalCylinder <> GlobalCylinder
;>        THEN
;>              IF LocalCylinder > GlobalCylinder
```

```
;>                THEN
;>                     Direction := Positive
;>                     Magnitude := LocalCylinder - GlobalCylinder
;>                ELSE
;>                     Direction := Negative
;>                     Magnitude := GlobalCylinder - LocalCylinder
;>               Temp := True
;>         ELSE
;>               Temp := False
;>        CalcMagDir := Temp
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

CalcMagDir:
                Push    !rE     ;save !rE

                 Ld      !rE,!rD ;move cylinder down one byte to use Sub3
                 Ld      !rD,!rC
                 Clr     !rC

                 Ld     !r1,Cur_Cyl
                 Ld     !r2,Cur_Cyl+1
                 Clr    !r0
                Call    Sub3    ;get Cylinder - CurrentCylinder

                Ld      !rC,!rD ;replace cylinder variable
                Ld      !rD,!rE
                Pop     !rE     ;replace Head
                Clr     !r4     ;assume no seek needed

                Jr      Lt,C_MagDir_Else

                Clr     !r5     ;assume 0 track seek
                Clr     !r6
                Ld      !r9,#0 ;set direction negative

                Clr     !r3     ;check for no seek condition
                Or      !r3,!r2
                Or      !r3,!r1
                Or      !r3,!r0
                Jr      Z,C_MagDir_Ret
                Jr      S_Glbl_Cyl

C_MagDir_Else:  Com     !r2     ;2's complement sutraction result
                Com     !r1
                Add     !r2,#1
                Adc     !r1,#0

                Ld      !r9,#Hd_Dir_Frwd     ;set direction positive

S_Glbl_Cyl:     Ld      !r5,!r1 ;load magnitude
                Ld      !r6,!r2

                Ld      !r4,#01 ;set Seek
C_MagDir_Ret:   Jp      Bank_Ret
```

```
                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Function: ServoOk
;>
;>      This function is responsible for determining if the
;>      servo is in a reasonable state to perform commands. In
;>      other words if ServoReady and NOT( ServoError ) then
;>      the servo is healthy, wealthy and wise. If, on the other
;>      hand, ServoError is active then the state of ServoReady
;>      determines the type of action to perform to try to get
;>      the servo back to a nice state:
;>
;>      ServoError and ServoReady: Read Status
;>      ServorError and Not( ServoReady ): Data Recal
;>
;>      Inputs: { none }
;>
;>      Outputs:
;>              ServOk: BOOLEAN { Zero Flag, True if NOT( ServoOk ) }
;>              RecalcMagDir: BYTE { !r0 }
;>
;>      Local Variables:
;>              Retry   : BYTE { !rF }
;>              SioRetry: BYTE { !r8 }
;>              Done    : BOOLEAN { }
;>              ControllerStatusPTR : PTR { !!rC }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       RecalcMagDir := False
;>       IF Recovery AND ServoError
;>        THEN
;>              Retry := 4
;>              WHILE ( Retry > 0 ) AND ServoError DO
;>                 IF ServoReady
;>                  THEN ServoStatus( ReadStatus, x
;>                                                x, NormalServoStatus )
;>                  ELSE ServoCmnd( DataRecal, x, x, x )
;>                 Retry := Retry - 1
;>       IF NOT( On_Track ) THEN RecalcMagDir := True
;>       ServoOk := NOT( ServoError )
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

ServoOk:
                Push    !r8
                Push    !rC     ;save registers
                Push    !rD

                Ld      !rC,#.HIBYTE. Status_Port
                Ld      !rD,#.LOWBYTE. Status_Port
```

```
                Tm      Excpt_Status,#Recovery          ;IF Recovery
                Jr      Z,S_Ok_End
                Lde     !r0,@!!rC       ;  AND ServoError
                Tm      !r0,#ServoErr
                Jr      Z,S_Ok_End

                Ld      !rF,#2          ;Retry := 2

;>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>              Or      !rF,!rF         ;THEN check for retry > 0
;>              Jr      Z,S_Ok_End
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>

S_Ok_Lp1:       Lde     !r0,@!!rC       ;check for servo error
                Tm      !r0,#ServoErr
                Jp      Z,S_Ok_End

                Tm      !r0,#ServoRdy
                Jr      Z,S_Ok_Else

                 Ld     !r0,#ReadStatus ;Load servo command buffer
                 Clr    !r1
                 Clr    !r2
                 Ld     !r3,#S_Norm_Status + S_Rate_57_6
                Call    ServoStatus     ; IF NOT( ServoStore(..)

S_Ok_Dec1:      Djnz    !rF,S_Ok_Lp1
                Jr      S_Ok_End

S_Ok_Else:       Ld     !r0,#DataRecal
                Call    Restore

                Jr      S_Ok_Dec1

S_Ok_UpDate:     Ld      !r1,#Dmt_ServoOk
                Call    ReadHdr
                Jr      Z,S_Ok_St_Rst

                Call    UpDate_Cur_Cyl
                Jr      S_Ok_Dec1

S_Ok_St_Rst:    Call    ResetServo
                Jr      S_Ok_Dec1

S_Ok_End:       Lde     !r0,@!!rC       ;sample ServoError
                Tcm     !r0,#ServoErr   ;return z = true if servo error
                Pop     !rD     ;restore registers
                Pop     !rC
                Pop     !r8

                Jp      Bank_Ret

                .LSTOFF
```