```
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Module: Spare1  { a continuation of Spare }
;>
;>      FUNCTION SrchSpTabl( LogicalBlock : 3 BYTES { !rC:E }
;>                           BlockType : 3 BITS { !r8/bits 3:1 } ) :
;>                         PhysicalBlock : 3 BYTES { !rC:E }
;>                         Status : BYTE { !r0 }
;>                         ElementPtr : BYTE { !r1 }
;>      FUNCTION GetNewSpare( BlockNumber : 3 BYTES { !rC:E } ) : BYTE { !r0 }
;>      PROCEDURE AddSpare( BlockType : 3 BITS { !r8/bits 3:1 }
;>                          SpareType : BIT { !r8/bit 4 }
;>                          Location  : BYTE { !rF }
;>                          LogicalBlock : 3 BYTES { !rC:E }
;>                        )
;>      PROCEDURE DeleteSpare( Location : BYTE { !rF }
;>                             LogicalBlock : 3 BYTES { !rC:E }
;>                           )
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Function: SrchSpTabl  { Search Spare Table }
;>
;>      This function is responsible for checking to see if the
;>      block number that is passed into it is currently in the
;>      spare table. If the block is found to be in the spare table
;>      then the physical block number of the spare block is passed
;>      back to the caller, as well as the PTR to the location of
;>      spared block's element within the spare table. In any case,
;>      a byte of status is always passed back to the caller describing
;>      the state of the logical block within the spare table.
;>
;>      Inputs:
;>              LogicalBlockNumber: 3 BYTES { !rC, !rD, !rE }
;>              ElementType       : BYTE { !rF }
;>
;>      Outputs:
;>              SrchSpTabl : BOOLEAN { zero flag set if not found in table }
;>              PhysicalBlockNumber: 3 BYTES { !rC, !rD, !rE }
;>              Status             : 1 BYTE { !r0 }
;>              ElementPtr         : 1 BYTE { !r1 }
;>
;>      Local Variables:
;>              HeadPtr : 1 BYTE { !r0 }
;>              Ptr     : 1 BYTE { !r0, offset; !!r8, actual Ptr }
;>              Found   : 1 BYTE { !r7 }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       CASE DiskCapacity OF
;>         10MB: k := 256; m := 256
;>         20MB: k := 128; m := 512
;>         40MB: k := 64;  m := 1024
```

```
;>       HeadPtr :=  Get_HeadPtr
;>       IF HeadPtr.Nil
;>        THEN PhysicalBlockNumber := LogicalBlockNumber +
;>                                    LogicalBlockNumber DIV k
;>        ELSE
;>              Ptr := HeadPtr.Ptr
;>              Ptr := Ptr * 4  { calc offset into spare table }
;>              Done := False
;>              Found := False
;>              WHILE NOT( Done ) DO
;>               IF ( Ptr^.Used ) AND ( Ptr^.Useable ) AND
;>                     ( Ptr^.Type = ElementType ) AND
;>                     ( Ptr^.Token = LogicalBlockNumber/bits 0:9 )
;>                THEN
;>                     PhysicalBlock := ( Ptr^.Location ) * m
;>                     Done := True
;>                     Found := True
;>                ELSE
;>                     Ptr := ( Ptr^.Ptr ) * 4
;>                     IF Ptr^.Nil
;>                      THEN Done := True
;>       Status := Ptr^.Status
;>       ElementPtr := Ptr
;>       IF NOT( Found ) THEN SrchSpTable := False
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

SrchSpTabl:
                Call    Get_HeadPtr
                Jr      Z,NotInTabl

                Clr     !r7             ;Found := False

SrchLp:         Ld      ScrReg1,!r0     ;save current ptr
                Call    Get_Ptr

                Lde     !r1,@!!r2       ;get element status
                Ld      ScrReg0,!r1     ;save element.status
                Tm      !r1,#Used       ;IF Used
                Jr      Z,SrchLpElse
                Tm      !r1,#Useable    ;  AND Useable
                Jr      Z,SrchLpElse
                Ld      !r0,Data_Type
                Tm      !r1,!r0         ;AND Type is correct
                Jr      Z,SrchLpElse
;                                               AND ( Token =
                Incw    !!r2
                Lde     !r1,@!!r2
                And     !r1,#$03         ;         LogicalBlockNumber/bits 0:9 )
                Ld      !r0,!rD
                And     !r0,#03
                Cp      !r0,!r1
                Jr      Nz,SrchLpElse
                Incw    !!r2            ;point to bits 0:7 of token
                Lde     !r0,@!!r2
                Cp      !r0,!rE
```

```
                Jr      Nz,SrchLpElse

                Tm      ScrReg0,#Spare ;check if BadBlock
                Jr      Nz,Srch_Spare

                Or      !r7,#Found
                Jr      Srch_Sp1

Srch_Spare:      Ld     !r0,ScrReg1
                 Inc    !r0 ;number spare blocks 1..76

;****** INLINE: MulR0_m *****
                Clr     !rE     ;Result := !r0 * 256
                Ld      !rD,!r0
                Clr     !rC

                .DO     W_20MB + W_40MB
                Rlc     !rE     ;Result := Result * 2
                Rlc     !rD
                Rlc     !rC
                .FIN
                .DO     W_40MB
                Rlc     !rE     ;Result := Result * 2
                Rlc     !rD
                Rlc     !rC
                .FIN

                Or      !r7,#Found
                Jr      SrchDone

SrchLpElse:     Tm      ScrReg0,#Nil        ;test if element.Ptr = Nil
                Jr      Nz,NotInTabl

                 Ld     !r0,ScrReg1 ;get address to current element
                Call    Get_Ptr
                Add     !r3,#3  ;point to next Ptr
                Adc     !r2,#0
                Lde     !r0,@!!r2
                Jr      SrchLp

NotInTabl:      Clr     !r7             ;return Not_Found status

Srch_Sp1:       Ld      ScrReg0,!rD

;***** INLINE: Div3_k *****
                Ld      !r2,!rD         ;shift right 1 byte
                Ld      !r1,!rC
                Clr     !r0

                .DO     W_20MB
                Ld      !rF,1           ;shift left once for DIV 128
                .FIN
                .DO     W_40MB
                Ld      !rF,2           ;shift left twice for DIV 64
                .FIN

                .DO     W_20MB + W_40MB
                Ld      !r3,!rE         ;shift left 1 bit
Div3_k_Lp:      Rlc     !r3             ;move !r3 bit 7 into carry flag
                Rlc     !r2             ;then shift all 3 bytes
                Rlc     !r1
                Rlc     !r0
                Djnz    !rF,Div3_k_Lp
```

```
                .FIN
                Add     !rE,!r2         ;PhysicalBlock := LogicalBlock +
                Adc     !rD,!r1         ;                 LogicalBlock DIV k
                Adc     !rC,!r0
                Ld      !r0,!rD ;save rollover byte

                .LSTOFF
                .DO     Internal
                .LSTON
                .FIN
                .DO     W_10MB
                Cp      !r0,ScrRegD ;check for rollover
                .LSTOFF
                .FIN
                .DO     W_20MB
                And     !r0,#$FE ;mask off MOD 512 bits
                And     ScrRegD,#$FE
                Cp      !r0,ScrRegD ;check for rollover
                .LSTOFF
                .FIN
                .DO     W_20MB
                And     !r0,#$FC ;mask off MOD 512 bits
                And     ScrRegD,#$FC
                Cp      !r0,ScrRegD ;check for rollover
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .FIN

                Jr      Z,SrchDone
                Add     !rE,#1 ;otherwise account for rollover
                Adc     !rD,#0
                Adc     !rC,#0

SrchDone:       Or      !r7,!r7 ;set status
                Ld      !r0,ScrReg0         ;return status
                Ld      !r1,ScrReg1     ;return ptr to element
                Jp      Bank_Ret

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Function: GetNewSpare
;>
;>      This function accepts either a physical block number or
;>      a logical block number and returns a one byte index into
;>      the Spare Table's bit map describing the location of the
;>      block that can be used as a spare.
;>
;>      Inputs:
;>              BlockNumber : 3 BYTES { !rC:E }
;>
;>      Outputs:
;>              GetNewSpare : BYTE { !r0 }
;>
;>      Global Variables Used:
;>              SpareBitMap
;>
;>      Local Variables:
```

```
;>              Bit   : ScrReg0
;>              Temp1 : ScrReg1
;>              Temp2 : ScrReg2
;>              NoHis : ScrReg3/bit 7
;>              NoLos : ScrReg3/bit 6
;>
;>      Algorithm:
;>
;>      BEGIN
;>       Bit := SrchSpTabl Div k { get physical blocknumber divided by
;>                                 the number of blocks between spares }
;>       IF ( SpareBitMap[ Bit ] = 0 )
;>        THEN GetNewSpare := Bit
;>        ELSE
;>              NoHis := False
;>              NoLos := False
;>              Temp1 := Bit
;>              WHILE NOT( NoHis ) AND ( SpareBitMap[ Temp1 ] = 1 ) DO
;>                      Temp1 := Temp1 + 1
;>                      IF ( Temp1 >= 76 ) THEN NoHis := True
;>              Temp2 := Bit
;>              WHILE NOT( NoLos ) AND ( SpareBitMap[ Temp2 ] = 1 DO
;>                      Temp2 := Temp2 - 1
;>                      IF ( Temp2 < 0 ) THEN NoLos := True
;>              IF ( NoHis AND NoLos )
;>               THEN ABORT
;>               ELSE
;>                IF NoHis
;>                 THEN GetNewSpare := Temp2
;>                 ELSE
;>                  IF ( Temp1 - Bit ) > ( Bit - Temp2 )
;>                   THEN GetNewSpare := Temp2
;>                   ELSE GetNewSpare := Temp1
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

GetNewSpare:
                Call    ExtPush_Vector ;save state

                .DO     W_10MB
                Ld      !r4,!r0 ;Div3_k, store result in !r4
                .FIN

                 Ld     !r0,!r4
                 Ld     !rC,#TestBitMap
                Call    TSC_BitMap      ;IF BitMap[ Bit ] = 0 ...
                Ld      !r0,!r4     ;assume Bit is unused
                Jr      Z,Gns_End       ;Jump if THEN

                Ld      !r5,!r4 ;ELSE ..
                Clr     !r7

Gns_Lp1:         Ld     !r0,!r5     ;test for bit map location = 0
                 Ld     !rC,#TestBitMap
                Call    TSC_BitMap
                Jr      Z,Gns_Lp1End
```

```
                Inc     !r5 ;bump Temp1
                Cp      !r5,#76
                Jr      Lt,Gns_Lp1
                Or      !r7,#$80    ;NoHis := True

Gns_Lp1End:     Ld      !r6,!r4

Gns_Lp2:         Ld     !rD,!r6     ;test for bit map location = 0
                 Ld     !rC,#TestBitMap
                Call    TSC_BitMap
                Jr      Z,Gns_Lp2End
                Dec     !r6
                Jr      Gt,Gns_Lp2
                Or      !r7,#$40    ;NoLos := True

Gns_Lp2End:     Tcm     !r7,#$C0    ;IF NoHis AND NoLos ..
                Jr      Nz,Gns_Chk_Hi

                 Ld     !r0,#1 ;byte 1
                 Ld     !r1,#SprBlk_Hard
                Call    SetStatus
                Call    Abort

Gns_Chk_Hi:     Ld      !r0,!r6     ;assume NoHis
                Tm      !r7,#$80    ;test for NoHis
                Jr      Nz,Gns_End

                Ld      !r0,!r5 ;assume NoLos
                Tm      !r7,#$40 ;test for NoLos
                Jr      Nz,Gns_End

                Ld      !r1,!r5
                Sub     !r1,!r4 ;otherwise find out which is closer
                Sub     !r4,!r6

                Ld      !r0,!r6 ;assume Temp2 is closer
                Cp      !r1,!r4
                Jr      Uge,Gns_End
                Ld      !r0,!r5 ; otherwise Temp1 is closer

Gns_End:        Push    !r0
                Call    ExtPop_Vector
                Pop     !r0
                Jp      Bank_Ret

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Procedure:  AddSpare { add an element to the spare table }
;>
;>      This function is responsible for adding an element to the spare
;>      table. It accepts a 1 byte value describing the location within
;>      the spare table { it is assumed that the caller has already used
;>      GetNewSpare } as well as the LogicalBlockNumber and whether the
;>      block being added to the table is a Spare block or a Bad Block.
;>
;>      Inputs:
;>              BlockType         : 3 BITS { !r8/bits 3:1 }
;>              SpareType         : BOOLEAN { !r8/bit 4 }
;>              Location          : BYTE { !rF }
```

```
;>              LogicalBlockNumber: 3 BYTES { !rC, !rD, !rE }
;>
;>      Outputs: { none }
;>
;>      Local Variables:
;>              HeadPtr : 1 BYTE { !r0 }
;>              Ptr     : 1 BYTE { !r0, offset; !!r8, actual Ptr }
;>
;>      Global Variables Changed:
;>              SpareCount
;>
;>      Algorithm:
;>
;>      BEGIN
;>       HeadPtr :=  Get_HeadPtr( LogicalBlockNumber )
;>       IF HeadPtr.Nil
;>        THEN
;>              HeadPtr.Nil := False
;>              HeadPtr.Ptr := Location
;>              SegPtrArray[ LogicalBlockNumber/bits 10:15 ] := HeadPtr
;>        ELSE
;>              Ptr := HeadPtr.Ptr
;>              Ptr := Get_EoList( Ptr )
;>              Ptr^.Nil := False
;>              Ptr^.Ptr := Location
;>       Ptr := Get_Ptr( Location )
;>       Ptr^.Nil := True
;>       Ptr^.Used := True
;>       Ptr^.Useable := True
;>       Ptr^.Spare := Spare
;>       Ptr^.Type := SpareType
;>       Ptr^.Token := LogicalBlockNumber/bits 0:9
;>       TCS_BitMap( Set, Location ) { set the bit map location for the add }
;>       IF ( SpareType = Spare )
;>        THEN SpareCount := SpareCount + 1
;>        ELSE BadCount := BadCount + 1
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

AddSpare:
                Call    Get_HeadPtr     ;IF HeadPtr.Nil ...
                Jr      Nz,ADS_Else1

                And     !r0,#$FF-Nil    ;THEN HeadPt.Nil := False
                Or      !r0,!rF         ;     HeadPtr.Ptr := Location
                Lde     @!!r2,!r0       ; create link
                Jr      ADS_UpDate

ADS_Else1:      Call    Get_EoList      ;search 'til end of list
                And     !r1,#$FF-Nil    ;Ptr^.Nil := False
                Lde     @!!r2,!r1       ;update table

                Add     !r3,#3          ;get Ptr^.Ptr
                Adc     !r2,#0
                Lde     @!!r2,!rF       ;create link
```

```
ADS_UpDate:      Ld      !r0,!rF          ;get structure ptr
                 Call    Get_Ptr          ;create a real ptr out of it

                 Ld      !r0,#Nil+Used+Useable
                 Or      !r0,!r8 ;merge Spare/Bad Block/Type Info
                 Or      !r0,Data_Type
                 Lde     @!!r2,!r0
                 Incw    !!r2             ;point to element.HiToken
                 Ld      !r0,!r0          ;get HiToken
                 And     !r0,#$03
                 Lde     @!!r2,!r0
                 Incw    !!r2             ;point to element.LoToken
                 Lde     @!!r2,!rE        ;store LoToken

                  Ld     !rC,#SetBitMap
                  Ld     !rD,!rF
                 Call    TSC_BitMap       ;update the bit map
                 Jp      Bank_Ret

                 .LSTOFF
                 .DO     External
                 .LSTON
                 .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Procedure:  DeleteSpare { add an element to the spare table }
;>
;>      This function is responsible for deleting an element from the spare
;>      table. It accepts a 1 byte value describing the location within
;>      the spare table { it is assumed that the caller has already used
;>      GetNewSpare } as well as the LogicalBlockNumber.
;>
;>      Inputs:
;>              Location          : BYTE { !rF }
;>              LogicalBlockNumber: 3 BYTES { !rC, !rD, !rE }
;>
;>      Outputs: { none }
;>
;>      Local Variables:
;>              Ptr1^.Status : BYTE { ScrRegF }
;>
;>      Global Variables Changed:
;>              SpareCount
;>
;>      Algorithm:
;>
;>      BEGIN
;>       Location := SrchSpTabl( Load_Logical )
;>       IF NOT( SrchSpTabl.Found ) THEN Abort
;>       IF ( Get_Head( LogicalBlockNumber ).Ptr = Location )
;>        THEN Head.Nil = True
;>        ELSE
;>             Ptr1 := Get_Ptr( Location )
;>             IF ( Ptr1^.Nil = True )
;>              THEN Ptr( PreviousElement )^.Nil = True
;>              ELSE Ptr( PreviousElement )^.Ptr = Ptr1^.Ptr
;>       Zero Out the Deleted Element
;>       TCS_BitMap( Clear, Location ) { free up the bit map location }
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                 .LSTOFF
                 .FIN
```

```
                .DO     Internal
                .LSTON
                .Page
                .FIN

DeleteSpare:
                Call    Load_Logical
                Call    SrchSpTabl
                Ld      !rF,!r1
                Jr      Nz,Chk_HdPtr

                Call    Abort

Chk_HdPtr:      Call    Load_Logical
                Call    Get_HeadPtr     ;IF ( Get_HeadPtr... )
                Ld      ScrRegE,!r0
                Ld      !r0,!rF
                Call    Get_Ptr
                Lde     !r0,@!!r2
                Tm      !r0,#Nil
                Jr      Z,Chk_Chain
                Cp      ScrRegE,!rF
                Jr      Nz,Chk_Chain    ;ELSE ...

                Call    Get_HeadPtr
                Ld      !r0,#Nil        ;THEN Head.Nil := True
                Lde     @!!r2,!r0
                Jr      Zero_Element

Chk_Chain:      Ld      ScrRegF,!r0
                Or      !r0,#Nil
                Lde     @!!r2,!r0       ;break the chain

                 Ld      !r0,ScrRegE
                Call    Get_EoList      ;get Ptr( Previous ) in ScrReg0,1

                Ld      !r0,ScrRegF     ;get original status back
                Tm      !r0,#Nil        ;IF Ptr1^.Nil
                Ld      !r2,ScrReg0     ;get Ptr( Previous )
                Ld      !r3,ScrReg1
                Jr      Z,D_Chk_Else    ; ELSE...

                Lde     !r0,@!!r2
                Or      !r0,#Nil
                Lde     @!!r2,!r0
                Jr      Zero_Element

D_Chk_Else:     Cp      !rF,ScrRegE
                Jr      Nz,Get_Previous
                Call    Get_HeadPtr
                Jr      Save_Previous

Get_Previous:   Add     !r3,#3          ;get to Ptr( Previous )^.Ptr
                Adc     !r2,#0
Save_Previous:  Push    !r2
                Push    !r3

                 Ld      !r0,!rF
                Call    Get_Ptr
                Add     !r3,#3          ;get to Ptr1^.Ptr
                Adc     !r2,#0
                Lde     !r0,@!!r2
```

```
                Pop     !r3
                Pop     !r2
                Lde     @!!r2,!r0       ;Ptr( Previous )^.Ptr := Ptr1^.Ptr

Zero_Element:    Ld      !r0,!rF         ;get Ptr1 once more
                Call    Get_Ptr

                Ld      !r0,#$FF ;initiale element
                Ld      !r1,#4          ;zero 4 bytes
Zero_E_Lp:      Lde     @!!r2,!r0
                Incw    !!r2
                Djnz    !r1,Zero_E_Lp

                 Ld     !rC,#ClearBitMap
                 Ld     !rD,!rF
                Call    TSC_BitMap
                Jp      Bank_Ret

                .LSTOFF
```