THE TEXT OF OSPub

by

Christopher Strachey

and

Joseph Stoy

Oxford University

Technical Monograph PRG-9(t)
July 1972
Oxford University Computing Laboratory,
Programming Research Group,
45, Banbury Road,
Oxford.
ABSTRACT

The two volumes of this monograph comprise the complete text of an experimental operating system, and a commentary on it. It is published to illustrate the authors' papers describing the system (Stoy, J.E., and Strachey, C.: OSO - an experimental operating system for a small computer; Comp. J. 15, Nos. 2 and 3, 1972), to give an example of an operating system written in a high level language, and to provide material for discussion about matters of style in programming.
FOREWORD

These books are a supplement to the authors' two papers on OS6 (a full reference is given in the abstract). Though the general design of the system is the work of the authors, other people have, of course, assisted with its implementation. The authors wish to make grateful acknowledgement to

Julia Bayman, Bijit Biswas, Malcolm Harper, Clifford Hones, Peter McGregor and Peter Mosses,

who have all (to a greater or lesser extent) helped with the writing of this system, Julia Bayman and Malcolm Harper have given particularly valuable assistance in preparing the system and these documents for publication. Nevertheless, the design errors which remain are the sole responsibility of the authors.
## CONTENTS

(Not: items enclosed in square brackets appear only in the commentary; round brackets denote either a further level of subheading, or that the item also appears again, and is described elsewhere.)

<table>
<thead>
<tr>
<th>Foreword</th>
<th>[I. Introduction</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. The text of the system</td>
<td>Aims of publication, Differences from 'OS6' papers, Style, Guide to these books.</td>
<td>8</td>
</tr>
<tr>
<td>1. Load-go loop, Run and Load</td>
<td>LGLOOP</td>
<td>8</td>
</tr>
<tr>
<td>1.1 LGLOOP</td>
<td>LoadGoLoop, LGLoop, [Prog,] DefaultProg.</td>
<td>8</td>
</tr>
<tr>
<td>1.2 RUN</td>
<td>Run, PrepareforRun, TerminateRun, ClearUp, [ClearUpChain,] LogIn, Finish.</td>
<td>9</td>
</tr>
<tr>
<td>1.3 LOAD</td>
<td>[Note on binary format, Format of compiled BCPL segment,] Load, [SectId, IBlock, Format of loaded section, CPtr, CFirst,] LoadSection, Unload.</td>
<td>11</td>
</tr>
<tr>
<td>1.4 SETLAB</td>
<td>SetLabels, SetGlobals.</td>
<td>15</td>
</tr>
</tbody>
</table>

| II. Post-mortem arrangements and interrupts | 11 |
| 2.3 MPM | ManualPM, Exchange, Return, JumpTo, AnythingTyped, MPMNextN, MPMNextU. | 23 |
| 2.4 INTERRUPT | [PPtr, PrivateStack,] Interrupt. | 26 |
3. Storage allocation and PutBack

3.1 FREESTORE
[FS,] NewVec, ReturnVec, RVGiveUp, NewWord, 
ReturnWord, MaxVecSize, NewFreeStore, 
RestoreFreeStore.

3.2 PUTBACK
[PBChain,] PutBack, NextPB, FNextPB, ClosePB, 
FClosePB, ResetPB, FResetPB, EndoPB.

4. Stream primitives and I/O routines

4.1 STRPRIMITIVES
Close, Reset, Source, State, ResetState, 
StreamError.

4.2 OUTS
OutS, WriteS, ReportS, OutString, GetC, 
OutJustify, Size, OSReport, OSReportN.

4.3 OPRTS
OutP, OutN, OutG, OutO, OutByte, OutAddr, 
Write, WriteN, WriteG, WriteO, WriteByte, WriteAddr.

4.4 NEXTN
NextN, NextO, NextCh.

5. Permanent input/output streams

5.1 TELETYPE
[Teletype,] NextTT, OutNewLine, OutTT, 
ResetTT, EndofTT, StateTT, ResetStateTT.

5.2 INTTY
[ExecConsole,] IntcodeToandfromTeletype, 
NextTele, EvenParity, OutTele, OutCRLF, 
OutNewLines, CloseTele, ResetTele, EndofTele, 
(Source).

5.3 XFER
InitiateTransfer.

5.4 READER
[BytesfromPT, ReaderDev,] ReadBuff, 
FirstNextBFPT, NextFillBFPT, ResetBFPT, 
NextWaitBFPT, StateBFPT, ReaderOffline, 
EndoBFPT, TryAgain.

5.5 DIADS
[Note on diad format, DiadRead,] 
WordsFromDiads, NextBlock, Next Byte, NextCh, 
EndofW, ResetW, CloseW, TryDiadAgain, 
ReadBackwards.
6. Miscellaneous facilities

6.1 CLOCK
   OutDateandTime, OutDate, OutTime, TimeOfDay,
   RestartClock, AskTime.

6.2 MISC
   AddressZero, Copy, EqS, NullProgram, Wait.

7. Disc routines (1)

7.1 DISCFER
   [DiscPage,] CoretoDisc, DiscToCore,
   DiscTransfer, Message.

7.2 DISCS
   [FSE,] NewDiscBlock, ReturnDiscBlock.

7.3 DISCIN
   CheckType, InfromFile, NextBuffIF, EndofIF,
   CloseIF, ResetIF, EntriesFromFile, NextEF,
   EndoEF, CloseEF, ResetEF.

7.4 DISCOUT
   ReturnChain, CheckPerm, DeleteBody,
   OuttoFile, OutBuffOF, TurnPage, CloseOF,
   ResetOF, FailClose.

8. Disc routines (2)

8.1 CHARGE
   Charge.

8.2 UPDATE
   Update.

8.3 UPDATEHEAD
   UpdateHead.

8.4 FINDHEADING
   LookUpinMFL, FindHeading.

8.5 LOOKUP
   LookUp.

8.6 LOADFILE
   LoadFile, LoadSystemFile.

9. Special functions in WIC
   FetchCode, StoreCode, Exec [TRANSFER (the
   executive teletype, the paper tape reader,
   the paper tape punch, the line printer, the
   disc, the clock, remote consoles), CANCEL,
   LOOKATRDR, READABS], Sumcheck, Next, Out,
   Endof, TransferIn, TransferInC, TransferOut.
III. Set-up programs

1. System set-up

1.1 SYSSETUP

[FSLim, CPages, DPages,] PInterrupt, SetUpFS,
SetUpDiscPage, SetUpPMStacks
SetUpSundryItems, SetUpTimeOfDayClock,
SetUpRunBlock, SetUpStackBase, CheckDiscOn,
ReleaseNonSystemGlobals, FSFtoCore, LogIn.

1.2 SETDANDT

SetUpDateAndTime, Date, Wrong, Leap, NextLetter.

1.3 QUICKVAL

QuickValFSF, ReportMessage.

2. Set up streams

2.1 SETUPSTR

SetUpStreams, [Parity,] SetUpParityTable,
[Output, ReportStream, Console, In].

2.2 SETUPTT

SetUpExecConsole, TT.

2.3 SETUPRDR

SetUpReader, RFPT.

IV. Declarations

1. DECLARATIONS

1.1 GLOBALS

1.2 PRIVATE GLOBALS

1.3 CONSTANTS

General, Machine constants, Exec commands,
Exec block for TRANSFER command, Standard
contents of TRANSFER block elements, Device
numbers, Code segment addresses
(INTERRUPTINHIBITED, REASONFORINTERRUPT,
MAXD, MAXC, SUMCHINHIBITED,
DISCWRITEMERMITTED, MFIRSTPAGE, DATE,
RBLOCK, TIMEOFDAYCLOCK), Reasons for
interrupt, Clock, Run block, Information
block, Free store, ClearUpChain, Stream
vector, Stream elements used by
InitiateTransfer, Fast stream block, Teletype
stream, Reader stream, PutBack vector,
Internal character code, Teletype character
code, Stack element, Filing system, Headings,
Index entries, Some file types, File
permissions, Master file list, Free store
file, InfoFromFile, OuttoFile.
V. Segmentation of the system for compilation
OS/1 - OS/8, OS/SU, OS/SUS.

VI. List of failure reports

VII. The system index
SystemIndex, Files with second name 'IC',
Files with second name 'Index', Other files.

VIII. Some library files
1. LogIn
   [User, CurrentIndex,] Prog, LookUpUser,
   String.
2. MakeNewFile
   MakeNewFile, LoadDiscRtsifNec, NewMFLentry,
   NewMFLPage, CreateNewHead, UpdatePermission,
   DeleteFile, CheckLegality.
3. Index Ops
   Enter, AddEntry, Link, CheckLinkDoesntLoop,
   Check, AddLinkedEntry, (LoadDiscRtsifNec,)
   CheckPermission, Size, DeleteEntry.
4. File Vectors
   VectorToFile, VectorFromFile,
   AddMoreVectorFile, (LoadDiscRtsifNec,)
   CheckPerm.
5. DiscRts
   NewLocation, MakeOnePageBody, AddVectorFile,
   TurnPage.
6. LinePrinter
   LinePrinter, GeneralLinePrinter, ByteStoLP,
   OutLP, CloseLP, ClearULP, Cancel,
   GeneralIntcodeToLP, OutLP, OutputUnderlines,
   TrapPageThrow, CloseLP, ResetLP,
   StandardErrorFn, PrinterReport, (Source).

References
Appendix: BCPL
Index
II: THE TEXT OF THE SYSTEM

II:1. The load-go loop, Run and Load

II:1.1 LLOAD

1 This section defines the global routines
1 LoadGoLoop, LLoop and DefaultProg.

5 let LoadGoLoop[] be
\$ Run[LLoop] repeat \$

10 let LLoop[] be
\$L
Reset[In]
Prog := DefaultProg

15 OutS[Console, "nOSPub"]
OutDateTime[Console]
Load[1]
Run[Prog]

20 Reset[Output]
Reset[ReportStream]
\$L

25 let DefaultProg[] be
\$ DSReport[111, "global 1 not set"]

****
This section declares the global routines Run, TerminateRun and Finish.

```plaintext
let Run[Program] be

PreparoRoeRun[]
Program[]
TerminateRun[] repeatuntil Terminated

and PreparoRoeRun[] be

let R = NewVec[RSize]           // New RunBlock :

RPRF := FetchCode[RBlock]      // Predecessor
RPTR := PPtr                 // Procedure Pointer
RIBLK := IBlock               // Information Block
RCPTR := CPtr                 // Code Pointer
RFsv := FS                   // FreeStore Vector

RINPT := In                   // Input Stream
ROUTP := Output               // Output Stream
RCON := Console               // Console Stream
RREP := ReportStream          // Report Stream
RGU := GiveUp                 // GiveUp Routine

RGUS := GiveUpStack           // Stack for GiveUp
RGUSS := GiveUpStackSize      // Size of stack needed
RCUC := ClearUpChain          // Used by ClearUp

NewFreeStore[]
CoretoDisc[FSF, FSPAGE]       // Update last page of FSFile

unless ClearUpChain = ENDCUCHN do
    $ ClearUpChain := CPre := lv RRU CU
        ClearUpChain := ENDCUCHN
    $

StoreCode[RBlock, R]

PR
```
and TerminateRun[ ] be
$TR$ Terminated := false
ClearUp[ ]

§ let $R = $FetchCode[RBLOCK]
GiveUpStackSize := $GUSS
GiveUpStack := $GUS
GiveUp := $GU
ReportStream := $REP
Console := $CON
Output := $OUTP
In := $INPT

Unload[R$CPTR]
RestoreFreeStores[R$FSV]
StoresCode[RBLOCK, $RPRE]
Terminated := true

ClearUpChain := $CUC
unless ClearUpChain = ENDCUCHN do
  ClearUpChain$CPRE := lv ClearUpChain

unless (IBlock = $IBLK)$CPtr = $CPTR) do
  § OSReport[121, "TerminateRun wrong:*n
  IBlock, CPtr = *N, *N', IBlock, CPtr]
  GiveUp[R] $§$

unless $RPRE = $ do ReturnVec[R, RSIZE]

if User = NULL do Run[LogIn]
$TR$

§CU

and ClearUp[ ] be
$CU$ until ClearUpChain = ENDCUCHN do
  §u let $E_1 = $ClearUpChain
  ClearUpChain := ClearUpChain$CSUC
  (E1$ROUTINE)[E1]

$CU
and LogIn[] be
$LI OutS[Console, 'Please log in *n']
   LoadSystemFile['LogIn']
   Prog[]

let Finish[] be
   $F PPtr := (FetchCode[RBLOCK])↓PPTR
   return
$F

****
II:1.3 LOAD

|| This section defines the global routines
|| Load and Unload.

5 manifest || Loader Warning Characters
  § CODE = 1
  INTERLUDE = 2
  BINARY = 3
  DATA = 4
10 TITLE = 5
  NEWSECTION = 6
  ENDLOAD = 7

§

15 manifest || Interpreted code instruction
  $ GLOBJUMP = 8140127 $ || 0021 (Set Globals); GOTO

20 static
  § SectId = 100
  GlobalsUnset = true

§

25 let Load[NumberOfEndloads] be
  §L
  $r
  let WCh = Next[In] || Warning character
  30 let Length = Next[In]
  switch WCh into
  $s case TITLE: || Skip Title
    for i=1 to Length do WCh := Next[In]
  endcase

35 case NEWSECTION:
  unless Length = 0 do
    $ OSReportN[131]; GiveUp[Length] $
    LoadSection[SectId]
  SectId := SectId+1
  endcase
60 and LoadSection[Ident] be
§LS let WCh, Length = 0, 0

let I = NewVec[ISIZE]  || New Information Block:
ICP :: CPtr  || Code Pointer
I\(\text{CL}\) :: 0  || Code Length
I\(\text{DP}\) :: NOTSET  || Data Pointer
I\(\text{DL}\) :: 0  || Data Length
I\(\text{IPRE}\) :: IBlock  || Predecessor
I\(\text{ISUC}\) :: NONE  || Successor
I\(\text{ID}\) :: Ident  || Section Identifier

Storecode[CPtr, I]
CPtr :: CPtr+1
CFirst :: CPtr
IBlock\(\text{ISUC}\) :: I
IBlock :: I
GlobalsUnset :: true
§

WCh := Next[In]
switchon WCh into
§§ case CODE:
   case INTERLUDE:
      Length :: Next[In]
   if (CPtr+Length) > FetchCode[MAXC] do
      § OSReport[I34, 'Too much code!']
      GiveUp[I]  §
GlobalsUnset := true
TransferInC[In, CPtr, Length]

test WCh = CODE
  then CPtr := CPtr + Length
  or  if I↑CL := CPtr - (I↓CP + 1)

  CPtr[]  ||  enter interlude
  GlobalsUnset := false

endcase

case DATA:
  unless I↓DP = NOTSET do
    $ OSReportN[135];  GiveUp[I]  $
    ||  more than one DBlock

  Length := Next[In]
  $ let D = NewVec[Length]  ||  Data Block
  D↓O := I
  I↓DP := D
  I↓DL := Length
  TransferIn[In, D+1, Length]
endcase

default :
  I↓CL := CPtr - (I↓CP + 1)
  PutBack[In, WCh]
  return  ||  to Load

repeat


let Unload[c] be

if c < (FetchCode[RBLOCK])\$CPTR do
  \$ OSReport[136, 'Unload[*N]', c]
  GiveUp[o]  

while c < IBlock\$CP do
  \$w
  if (FetchCode[(IBlock\$CP+1)=GLOBJUMP]~\$GlobalsUnset
      do SetGlobals[]  || to denest

  unless IBlock\$DP = NOTSET do
    ReturnVec[IBlock\$DP, IBlock\$DL]  || return DBlock

  IBlock := IBlock\$IPRE
  ReturnVec[IBlock\$ISUC, ISIZE]  || return IBlock

  \$w

  CPTR := IBlock\$CP + IBlock\$CL + 1
  SectId := IBlock\$ID + 1
  IBlock\$ISUC := NONE

135  \$w

140  \$w

145  \$w

150  \$w

155  \$w

****
II:1.4 SETLAB

|| This section defines the routines SetLabels and SetGlobals, which are normally called only by
|| the interlude of a program.

5 manifest || Relocating information
§
| SEGBIT = $100000
| SEGMASK = $77777
| ENDLABCHN = $77776
10 || BINARY is declared in 'LOAD'
§

let SetLabels[] be
15 §SL
  let C = IBlock\CP + 1 || Start of program's Code segment
  let D = IBlock\DP + 1 || Start of program's Data segment

let WCh = Next[In]
20 unless WCh = BINARY do
  § OSReportN[141]; GiveUp[WCh]
§

for i=1 to Next[In]/2 do
  §f let A = Next[In]
25 let R = Next[In]
  let ASeg = A\SEGBIT
  let RSeg = R\SEGBIT
  let Val = (A\SEGMASK + (ASeg\CSEG + C, D)
  let Ref = (R\SEGMASK)
30 test RSeg = CSEG
  then §C
  until Ref = ENDLABCHN do
    §u let Ptr = FetchCode[Ref+C]
35 StoreCode[Ref+C, Val]
    Ref := Ptr
  §C
  or §D
40 if IBlock\DP = NOTSET unless Ref = ENDLABCHN do
  § OSReportN[141]; GiveUp[IBlock]
  || DLabels but no DBlock
\[
\text{until Ref = ENDLABCHN do} \\
\text{let Ptr = rv(Ref+D)} \\
\text{rv(Ref+D) := Val} \\
\text{Ref := Ptr}
\]

|| SetGlobals nests the declared globals \\
|| It is also called by Unload to de-nest them.

let SetGlobals[] be \\
\text{let p = IBlock} \backslash CP + IBlock \backslash CL \\
| last word of program code \\
let n = FetchCode[p] \\
| number of globals to be set \\
for i=1 to n do \\
\text{let Address = FetchCode[p-1]} \\
\text{let Contents = FetchCode[p-2]} \\
\text{StoreCode[p-2, rv Address]} \\
rv Address := Contents \\
p := p-2
11:2, Post-mortem arrangements and interrupts

11:2.1 GIVEUP

|| This section defines the global routines

manifest || Dumping information
5 $ PRNVEC $ 33
VEC $ 10
C $ 3
D $ 2
DUMPSTART $ 3776
10 $

let ForceGiveUp[] be
§FGU

$ test InStack[PrivateStack\0, GiveUpStack]
15 then PrivateStack\0 := GiveUpStack\0
or GiveUpStack\0 := PrivateStack\0

§1 let R := FetchCode[RBLOCK]
while InStack[R\PPTR, GiveUpStack] || Unravel
20 do $ TerminateRun[] || Runs in
R := FetchCode[RBLOCK] § || GiveUpStack,

§1

PPtr := GiveUpStack
25 ReportS['Forced GiveUp']

if GiveUp $ StandardGiveUp do
unless GiveUpStackSize < GiveUpStack\LENGTH do
§SS let g = GiveUp
30 GiveUp := StandardGiveUp
|| in case FreeStore runs out
§ let V = NewVec[GiveUpStackSize+1]
V\0 := GiveUpStack\0
V\LENGTH := GiveUpStackSize+1
35 GiveUp := g
GiveUpStack := V
PPtr := V
§SS

40 GiveUp[0]
§FGU
let StandardGiveUp[n] be

let Ch = 0
Outs[Console, 'GiveUp *N*0, Dump *q*, n, n]
Ch := Next[Console]
repeat until (Ch='y') v (Ch='Y') v (Ch='n') v (Ch='N')
test (Ch='y') v (Ch='Y')
then Dump[]
or StandardPM[]
EndGiveUp['GiveUp', n]

and Dump[] be

let PRNVec = FetchExecWord[PRNVec]
let PRN8 = FetchExecWord[PRNVec + 8]
let CBounds = FetchExecWord[PRN8 + C]
let DBounds = FetchExecWord[PRN8 + D]
let UC, LC = CBounds + 377, CBounds rshift 8
let UD, LD = DBounds + 377, DBounds rshift 8
let CSize = (UC-LC)XPAGESIZE
let DSize = (UD-LD)XPAGESIZE

SetUpDummyExecStructure[CBounds, DBounds]
DumpSegment[CODESEG, DUMPSTART + LC, CSize]
DumpSegment[DATASEG, DUMPSTART + LD, DSize]
ReportS['Dumped']

and FetchExecWord[Addr] = valu

let Ans = 0
Exec[READABS, L, lv Ans, Addr]
L: resultis Ans
and SetUpDummyExecStructure[CBounds, DBounds] be

- DiscPage\[PRNVEC := 0 \] dummy PRN vector at 0
- DiscPage\[8 := VEC \] PRN8
- DiscPage\[9 := VEC \] PRN9
- DiscPage\[(VEC+C) := CBounds \]
- DiscPage\[(VEC+D) := DBounds \]

95 \$ let x = FetchCode[DISCWRITEPERMITTED]
    StoreCode[DISCWRITEPERMITTED, true]
    CoretoDisc[DiscPage, DUMPSTART]
    StoreCode[DISCWRITEPERMITTED, x]

100 and DumpSegment[Seg, Page, Size] be

- unless Page > DUMPSTART do
  - \$ OSReportN[211]; Wait[ ] $

105 $ giving

- let E = vec E\[SIZE := 0 \] Exec Block
  - E\[DEVICE := DISCWRITE \]
  - E\[BUFFER := 0 \]
  - E\[BUFFSIZE := Size \]
  - E\[PAGENUMB := Page \]
  - E\[SEG := Seg \]
  - E\[ENDMODE := QUIETEND \]
  - E\[COMPLETED := false \]
  - E\[SELFPTR := E \]

- The other two elements are not used.

110 Exec[TRANSFER, Rj, E\[PAGENUMB := Page \]]

120 \$ $ repeat until E\[COMPLETED \]

125 if E\[PAGENUMB = Page \] break
- If the call is rejected or terminates
  - successful transfer

125 $ if E\[PAGENUMB != Page \]
- unsuccessful, Exec overwrites E\[PAGENUMB \]
  - with a failure number.

130 $ repeat
let EndGiveUp[String, n] be
  Out$[\text{Console}, \text{"S N O"}, \text{String, n, n}]
  \text{Wait[\text{]}]

and ForcedFinish[] be
  \text{Reset[DiadRead]}
  \text{Reset[ExecConsole]}
  \text{Finish[\text{]}]

\text{****}
This section defines the global routines StandardPM, ReportCallTrace, and ReportFreeStoreState.

Let StandardPM[] be

\[ \text{let StandardPM[]} \text{ be } \]
\[ \text{let ReportCallTrace} \text{[2, 10] } \]
\[ \text{let ReportFreeStoreState} \text{[FS]} \]

Let ReportCallTrace[n, Max] be

\[ \text{let P = Pptr} \]
\[ \text{let RP = (FetchCode[RBLOCK]) WPTR} \]
\[ \text{ReportS[' P LINK' ]} \]
\[ \text{for i = 1 to n do P := rv P} \]
\[ \text{don't trace most recent n calls} \]
\[ \text{until P > RP do P := rv P} \]
\[ \text{don't trace in GiveUpStack or PrivateStack} \]
\[ \text{for i = 1 to Max do} \]
\[ \text{if P < RP break} \]
\[ \text{ReportS["N *N", P, rv(P+1)]} \]
\[ \text{P := rv P} \]
\[ \text{ReportS["P at last Run: *N", RP]} \]

And ReportFreeStoreState[f] be

\[ \text{ReportS["FreeStore area *N to *N", f\text{"LB, f\text{"UB] } \]
\[ \text{ReportBlocks["Free", f\text{"FBC]} \]
\[ \text{ReportWords["Free", f\text{"FWC]} \]
\[ \text{ReportBlocks["Pending", f\text{"PBC]} \]
\[ \text{ReportWords["Pending", f\text{"PW]} \]

\[ \text{ReportS} \]
end ReportBlocks[\text{String}, \text{b}] \text{ be}

\begin{align*}
45 \quad & \text{let } n, w, m = 0, 0, 0 \\
& \text{if } b = \text{END} \quad \text{return} \\
& \text{until } b = \text{END} \quad \text{do} \\
& \quad n := n + 1 \\
& \quad \text{if } b.\text{SIZE} > m \quad \text{then} \quad m := b.\text{SIZE} \\
& \quad w := w + (b.\text{SIZE} + 1) \\
& \quad b := b.\text{NXTB} \\
& \quad \text{ReportS}[\text{"N S S s n *N words* n Largest block size *N"}, \\
& \quad n, \text{String}, (n=1 \rightarrow \text{"Block"}, \text{"Blocks"}), w, m] \\
\end{align*}

\begin{align*}
60 \quad & \text{and ReportWords[\text{String}, \text{w}] be} \\
\text{let } n = 0 \\
& \text{if } w = \text{END} \quad \text{return} \\
& \text{until } w = \text{END} \quad \text{do} \\
& \quad n := n + 1 \\
& \quad w := \text{rv} \quad w \\
& \quad \text{ReportS}[\text{"N S S s s", n, \text{String}, (n=1 \rightarrow \text{"Word"}, \text{"Words"})}] \\
\end{align*}

70 ****
This section defines ManualPM (used by Interrupt)

```plaintext
static $ Con = 0$

let ManualPM[] be

let Ch, A, N = 0, 0, 0

let Exch, CSeq, Both, Ret

Duts[Con, '*nMPM ']

| Con is set in Interrupt |

| §r2 |

Ch := Next[Con]

switchon Ch into

| case 'N': |

| case 'n': ForcedFinish[] |

| case 'E': |

| case 'e': Exch := true endcase |

| case 'Y': |

| case 'y': |

| case 'c': CSeq := true |

| case 'B': |

| case 'b': Both := true endcase |

| case 'R': |

| case 'r': Ret := true |

break || out of §r2

§r2 repeatuntil ('o'<Ch<'9') \(\lor\) (Ch='0')

test Ret then Return[] or
\[ A := \text{MPMNextN}[\text{Con}, \text{Ch}] \]
\[ \text{ResetState}[\text{Con}] \]
\[ N := \text{CSeg} \rightarrow \text{FetchCode}[A], \text{rv} A \]

\[ \text{if } N \neq 0 \text{ unless Exch do} \]
\[ \text{Out}[\text{Con}, "^n"] \]
\[ A := A + 1 \]
\[ N := \text{CSeg} \rightarrow \text{FetchCode}[A], \text{rv} A \]
\[ \text{repeat until } N \neq 0 \]

\[ \text{OutS}[\text{Con}, (\text{Both} \rightarrow "^N \ast A \ast N", "^N \ast N"), A, N, N] \]

\[ \text{test Exch then} \]
\[ \text{OutS}[\text{Con}, "^n"] \]
\[ \text{Ch := Next}[\text{Con}] \text{ repeatwhile } \text{Ch} = "^a" \]
\[ \text{if } (\text{Ch} = "^N") \lor (\text{Ch} = "^n") \text{ break } || \text{ out of } \text{sr} 3 \]
\[ \text{Exchange}[A, \text{Ch}, \text{CSeg}] \]

\[ \text{or} \]
\[ \text{Out}[\text{Con}, "^n"] \]
\[ \text{if AnythingTyped}[\text{Con}] \text{ break } || \text{ out of } \text{sr} 3 \]

\[ A := A + 1 \]
\[ \text{repeat} \]
\[ \text{repeat} \]
\[ \text{and Exchange}[\text{Addr}, \text{Ch}, \text{CSeg}] \text{ be} \]
\[ \text{let } N = 0 \]
\[ \text{if } (\text{Ch} = "^n") \text{ return} \]
\[ \text{if } (\text{"O"} \leq \text{Ch} \leq "9") \lor (\text{Ch} = "-" ) \text{ do} \]
\[ \text{N := MPMNextN}[\text{Con}, \text{Ch}] \]
\[ \text{break } || \text{ out of } \text{sr} \]
\[ \text{if } (\text{Ch} = "B") \lor (\text{Ch} = "b") \text{ do} \]
\[ \text{N := MPMNextO}[\text{Con}, \text{Ch}] \]
\[ \text{break } || \text{ out of } \text{sr} \]
ch := Next[Con]

repeat

test CSeg then StoreCode[Addr, N]
or rv Addr := N

and Return[] be

Outs[Con, 'Return *q *']

let ch = Next[Con]
unless (Ch='Y') then (Ch='y') return \| to ManualPM

Outs[Con, 'P, Link, Result (in Decimal) := '] 

let P = MPMNextN[Con, NULL]
let Link = MPMNextN[Con, NULL]
let Result = MPMNextN[Con, NULL]

let PP = PPtr
let RunLevel = (FetchCode[RBLOCK]) + PPtr
until (PP=P) or (PP=RunLevel) do PP := rv PP
unless PP=P do

Outs[Con, 'Wrong p*n']

return \| to ManualPM

let Dummy = JumpTo[P, Link, Result]

and JumpTo[P, Link, Result] = valof

let rv PPtr := P \| fiddle stack
let rv (PPtr+1) := Link
resultis Result \| Does the actual return

and AnythingTyped[S] = valof

let x = state[S]
if x = NOTHING_TYPED resultis false
if (x=RETURN) then (S$#execConsole) do

resultis false

resultis true
and $\text{MPMNextN}[S, \text{FirstCh}] = \text{valof}$

\[ S_{NN} \]

135 let $Ch, n, Neg = \text{FirstCh, 0, false}$

\[ \text{S}_1 \text{ if '0'<} Ch<='9' \text{ break } || \text{ out of } \text{S}_1 \]

\[ \text{unless } (Ch='*3') \lor (Ch='*4') \]

\[ \text{do } Neg := (Ch='*') \]

140 $Ch := \text{Next}[S]$ \n
\[ \text{S}_1 \text{ repeat} \]

\[ \text{S}_2 n := 10 \times n + (Ch='0') \]

$Ch := \text{Next}[S]$ \n
145 $\text{S}_2 \text{ repeatwhile } '0'<Ch<='9' \]

resultis Neg $\rightarrow$ $n, n$

|| loses the terminating character

\[ S_{NN} \]

150 and $\text{MPMNextO}[S, \text{FirstCh}] = \text{valof}$

\[ S_{NO} \]

155 let $Ch, m, n = \text{FirstCh, 0, 0}$

\[ \text{until } '0'<Ch<='7' \text{ do } Ch := \text{Next}[S] \]

160 \[ \text{S}_1 n := 8 \times n + (Ch='0') \]

$Ch := \text{Next}[S]$ \n
\[ \text{S}_1 \text{ repeatwhile } '0'<Ch<='7' \]

\[ \text{until } Ch = '*n' \text{ do } || \text{ look for second byte} \]

\[ \text{S}_U \text{ if '0'<} Ch<='7' \text{ do} \]

\[ \text{S}_B \text{ S}_2 m := 8 \times m + (Ch='0') \]

$Ch := \text{Next}[S]$ \n
\[ \text{S}_2 \text{ repeatwhile } '0'<Ch<='7' \]

165 $n := (n \text{ ishift 8}) + m$

\[ \text{break } || \text{ out of } \text{S}_U \]

\[ \text{S}_B \]

$Ch := \text{Next}[S]$ \n
170 $\text{S}_U \text{ resultis } n$

|| loses the terminating character

\[ S_{NO} \]

****
II:2.4 INTERRUPT

This section defines the global routine Interrupt

5 manifest $\text{UNUSED} = 0$

static $\text{TempP} = \text{UNUSED}$

let Interrupt[] be

15 if TempP = UNUSED do TempP := PPtr

PPtr := PrivateStack

unless InStack[TempP, PrivateStack]

do PrivateStack[0 := TempP

TempP := UNUSED

let RFI = FetchCode[REASONFORINTERUPT]

StoreCode[REASONFORINTERUPT, NOREASON]

Con := Console \| Con is a \textit{static} declared in MPM

Reset[Con]

25 switchon RFI into

§§ case POWERON:\n
\| auto start

Out[ExecConsole, '*n']

ForcedFinish[]

30 case NOREASON:\n
\| probably violation

OutS[Con, '*nVIOLATED;']

endcase

35 case EXECCONXON:\n
\| X-ON from ExecConsole

Con := ExecConsole

endcase

default : \| Remote console X-ON

40 $\text{OutS}[\text{Con}, '*n\text{RUN} 8;']$
let Ch = 0

let Out[Con, BELL] ResetState[Con] Ch := State[Con] repeat while Ch = NOTHINTYPED

Ch := Ch ^ PARITYMASK

switchon Ch into
  \$ case 'L': || LogIn
  case 'l':
    User, CurrentIndex := NULL, NULL
    || run on into next case

  case 'G': || Go
  case 'g':
    Out[Con, 'C*n', Ch] ForcedFinish[]

  case 'F': || ForcedGiveUp
  case 'f':
  case 'M': || ManualPM
  case 'm':
    \$ let c = Console
    Console := Con
    Out[Con, Ch]
    OutDateandTime[Con]
    Console := c \$

70 \$ test (Ch='M') v (Ch='m')

\$ then ManualPM[]

\$ or ForcedGiveUp[]

\$ repeat

***
II:3. Storage allocation and PutBack

II:3.1 FREESTORE

This section defines the global routines and functions NewVec, ReturnVec, NewWord, ReturnWord, MaxVecSize, NewFreeStore and RestoreFreeStore.

5  let NewVec[n] = value  
SNV
   if n < 0 do
      OSReport[311, "NewVec[*N]", n]
   GiveUp[0]
   if (n=0)^(FS↑FWC ≠ END) do
      let w = FS↑FWC  \* first word in FreeWordChain
      FS↑FWC := rv w
   result is w

5  let BP = lv FS↑FBC  \* Block Pointer
let B = rv BP  \* Block

20  if B = END do
      OSReport[312, "NewVec[*N] too big", n]
      GiveUp[FS]

25  if B↓SIZE > n break
BP := lv B↓NXTB
B := rv BP

repeat

30  test B↓SIZE ≤ n+1
   then
      rv BP := B↓NXTB
      if B↓SIZE > n do ReturnWord[lv(B↓(n+1))]

35  or
   let SB = lv(B↓(n+1))  \* SurplusBlock
   SB↓NXTB := B↓NXTB
   SB↓SIZE := (B↓SIZE)-(n+1)
   \* The order of the last two assignment statements
   \* is important if n=0. Here NXTB > SIZE.
rv BP := SB

result is B

let ReturnVec[V, n] be

unless FS'LB ≤ V ≤ V+n ≤ FS'UB do

§ Pend

if n < 0 do RVGiveUp[313, V, n, 0]

unless (V>FS'UB) ∨ (V+n < FS'LB)

do RVGiveUp[314, V, n, FS]

§ let f = FS

while f > f'FPRE do f := f'FPRE || seek first FS

unless f = f'FPRE do RVGiveUp[315, V, n, FS]

unless (f'LB ≤ V) ∧ (V+n ≤ f'UB)

do RVGiveUp[316, V, n, f]

if n = 0 then

§ rv V := FS'FWC

FS'FWC := V || pending word

§

or

§ V'SIZE, V'NXTB := n, FS'PBC

FS'PBC := V || pending block

§

return

§ Pend

§ let PW = rv FS'FWC || Previous Word

let W = rv PW || Word

until (W=END) ∨ (W ≥ V-1) do

§ PW := W

W := rv PW

§

unless W = END do

§1 if W = V-1 do

§ W := rv W

rv PW := W

V := V-1

n := n+1

§
if $V \leq W \leq V+n$ do RVGiveUp[317, $V$, $n$, $FS$]

if $W-1 = V+n$ do
\[ PV := PV \]
\[ n := n+1 \]

let $BP = lv FS\#FBC$ \hspace{1cm} \text{// Block Pointer}
let $B = rv BP$ \hspace{1cm} \text{// Block}

until $(B-END) \lor (B+B\downarrow SIZE \geq V-1)$ do
\[ BP := lv B\downarrow NXTB \]
\[ B := rv BP \]

unless $B = END$ do
\[ DV \downarrow SIZE = V-1 \text{ do} \]
\[ B\downarrow SIZE := B\downarrow SIZE + (n+1) \]
\[ DV \downarrow SIZE := B\downarrow SIZE + ((B\downarrow NXTB)\downarrow SIZE +1) \]
\[ B\downarrow NXTB := (B\downarrow NXTB)\downarrow NXTB \]

return

if $B-1 = V+n$ do
\[ DV \downarrow SIZE := n + (B\downarrow SIZE +1) \]
\[ V\downarrow NXTB := B\downarrow NXTB \]
\hspace{1cm} \text{// The order of the last two assignment statements is important if $n=0$.}
\[ rv BP := V \]

return

unless $B > V+n$ do RVGiveUp[318, $V$, $n$, $FS$]

\[ test n = 0 \]
\hspace{1cm} then $\downarrow V := W$
\hspace{1cm} $\downarrow PV := V$

\[ or \]
\hspace{1cm} $\downarrow V\downarrow NXTB := B$
\hspace{1cm} $\downarrow V\downarrow SIZE := n$
\hspace{1cm} $\downarrow PV := V$

\[ RV \]
and RVGiveUp[r, V, n, g] be
§ OSReport[r, 'ReturnVec[*N, *N]', V, n]  
GiveUp[g]  

140 let NewWord[] = NewVec[0]

let ReturnValue[w] be ReturnVec[w, 0]

let MaxVecSize[] = valof §MVS

150 let M = (FS↓FWC = END)→ NOSTORE, 0
let B = FS↓FBC
until B = END do
§ if B↓SIZE > M do M := B↓SIZE
155 B := B↓NXTB
§ result is M
§MVS

160

let NewFreeStore[] be §NFS

165 let f = NewVec[FSVECSIZE]  || new FSBlock
let MVS = MaxVecSize[]
let A = NewVec[MVS]  || new area

f↓FBC, f↓FWC := A, END
f↓FBC, f↓FWC := END, END
170 f↓LB, f↓UB := A, (A+MVS)
f↓FPRE := FS
A↓NXTB := END
FS := f
175 §NFS
let RestoreFreeStore[Fstore] be
  §RFs
  §1 let f = FS
  until (f<FS)Ú(f<FPRE) do f := f<FPRE
  unless f=FS do
    § DSReportN[319] ; GiveUp[FS]  §
  §1
  until FS=FS do
  §U test FS<lb<PBchain<FS<ub
  185 then || PutBack is in current area
    §2 let s = PBchain<STREAM
    let V = (S<o)Ú (~s), s
    test FS<lb<v<FS<ub
    then || Stream also in current area
  190  § PBchain := PBchain<PBPre
       RestoreFreeStore[FS<FPRE]
    §
    or
    § let Ob = Next[s] || to remove PutBack
  195 RestoreFreeStore[FS<FPRE]
       PutBack[s, Ob]
    §2
    or
    §3 let f = FS
  200 FS := FS<FPRE
       ReturnVec[f<lb, (f<ub - f<lb)]
  § let x = f<PBC || Pending Blocks
  205 until x = END do
    § let b = x
    x := b<NXTB
       ReturnVec[b, b<SIZE]
    §
    x := f<PWC || Pending Words
  210 until x = END do
    § let w = x
    x := rv w
       ReturnWord[w]
    §
  215 ReturnVec[f, FSVECSIZE]

§RFs
****
II:3.2 PUTBACK

This section defines the global routine PutBack.

5 let PutBack[Str, x] be
   $PB$
   let $p = \text{NewVec}[\text{PBSIZE}]$
   let $s = (\text{Str} > 0) \rightarrow \text{Str}, (~\text{Str}) \rightarrow \text{SLOWBLOCK}$

10 $p \downarrow \text{OB} := x$
   $p \downarrow \text{STREAM} := \text{Str}$

   $p \downarrow \text{PBNEXT}, p \downarrow \text{PENDING} := s \downarrow \text{NEXT}, s \downarrow \text{ENDOF}$
   $p \downarrow \text{PCLOSE}, p \downarrow \text{PRESET} := s \downarrow \text{CLOSE}, s \downarrow \text{RESET}$

15 $p \downarrow \text{PSTR} := s \downarrow \text{STR}$

   if (Str < 0) do
      $p \downarrow \text{PTR} := (~\text{Str}) \downarrow \text{INBFFPTR}$
      $(~\text{Str}) \downarrow \text{INBFFPTR} := ((~\text{Str}) \downarrow \text{INBFFEND}) + 1$

20  

   $p \downarrow \text{PEPRE} := \text{PBChain}$
   $\text{PBChain} := p$

25 $s \downarrow \text{STR}, s \downarrow \text{ENDOF} := p, \text{EndofPB}$

   test (Str > 0)
   then $s \downarrow \text{NEXT}, s \downarrow \text{CLOSE}, s \downarrow \text{RESET} := \text{NextPB}, \text{ClosePB}, \text{ResetPB}$
   or $s \downarrow \text{NEXT}, s \downarrow \text{CLOSE}, s \downarrow \text{RESET} := \text{FNextPB}, \text{FClosePB}, \text{FResetPB}$

$\text{PB}$

30 end NextPB[S] = valof

$\text{SNPB}$

35 let $p = s \downarrow \text{STR}$

36 let $x = p \downarrow \text{OB}$

   $s \downarrow \text{NEXT}, s \downarrow \text{ENDOF} := p \downarrow \text{PBNEXT}, p \downarrow \text{PENDING}$
   $s \downarrow \text{CLOSE}, s \downarrow \text{RESET} := p \downarrow \text{PCLOSE}, p \downarrow \text{PRESET}$
   $s \downarrow \text{STR} := p \downarrow \text{PSTR}$

40 if $p \downarrow \text{STREAM} < 0$ do
   $(~(p \downarrow \text{STREAM})) \downarrow \text{INBFFPTR} := p \downarrow \text{PTR}$


\$ \text{let } BP = \text{lv PBChain} \quad \text{// Block Pointer.} \\
\text{let } B = \text{rv BP} \quad \text{// Block.} \\
\text{until } (B = \text{END}) \lor (B = \text{P}) \text{ do} \\
\quad \\$ \text{BP} := \text{lv BP} \oplus \text{PBPRE} \\
\quad \text{B} := \text{rv BP} \\
\text{\$} \\
\text{if } B = \text{END} \text{ do} \\
\quad \$ \text{OSReportN[32i]; GiveUp[FS]} \text{ \$} \\
\text{rv BP} := \text{BP} \oplus \text{PBPRE} \\
55 \text{ReturnVec}[P, \text{PBSIZE}] \\
\text{resultis } x \\
\$NPB \\
\text{and } \text{FNextPB}[FB] = \text{NextPB}[FB \downarrow \text{SLOWBLOCK}] \\
60 \text{and } \text{ClosePB}[S] \text{ be} \\
\$C \text{let } x = \text{NextPB}[S] \\
\text{Close}[S] \\
65 \$C \\
\text{and } \text{FClosePB}[FB] \text{ be} \\
\$FC \text{let } x = \text{FNextPB}[FB] \\
\text{Close}[\sim FB] \\
70 \$FC \\
\text{and } \text{ResetPB}[S] \text{ be} \\
\$R \text{let } x = \text{NextPB}[S] \\
75 \text{Reset}[S] \\
\$R \\
\text{and } \text{FResetPB}[FB] \text{ be} \\
\$FR \text{let } x = \text{FNextPB}[FB] \\
80 \text{Reset}[\sim FB] \\
\$FR \\
\text{and } \text{EndofPB}[S] = \text{false} \\
85 ****
II:4. Stream primitives and I/O routines

II:4,1 STRPRIMITIVES

This section defines the global routines and functions Close, Reset, State, ResetState and StreamError.

let Close[S] =
  \$C
  test S > 0
  then (S\$CLOSE)[S]
  or \$ S := ~S
  ((S\$SLOWBLOCK)\$CLOSE)[S]

let Reset[S] =
  \$R
  test S > 0
  then (S\$RESET)[S]
  or \$ S := ~S
  ((S\$SLOWBLOCK)\$RESET)[S]

let Source[S] = valof
  \$
  if S < 0 do S := (~S)\$SLOWBLOCK
  result (S\$SOURCE)

let State[S] = valof
  \$
  let Sce = Source[S]
  result (Sce\$STATE)[Sce]

let ResetState[S] =
  \$
  let Sce = Source[S]
  (Sce\$RESETSTATE)[Sce]

let StreamError[S] =
  \$
  OSReport[411, 'StreamError']
  GiveUp[S]

***
This section defines the global output routines OutS, WriteS and ReportS, and also the system routines OSReport and OSReportN.

let OutS[S, String, a,b,c,d,e,f,g,h,i,j,k,l,m, n,o,p,q,r,s,t,u,v,w,x,y,z] be OutString[S, String, lv a]

and WriteS[String, a,b,c,d,e,f,g,h,i,j,k,l,m, n,o,p,q,r,s,t,u,v,w,x,y,z] be OutString[Output, String, lv a]

and ReportS[String, a,b,c,d,e,f,g,h,i,j,k,l,m, n,o,p,q,r,s,t,u,v,w,x,y,z] be OutString[ReportStream, String, lv a]

and OutString[S, String, ParamList] be

let Ptr, Ch = 0, 0
let i = 0
let Length = (String\0) rshift 8

until Ptr = Length do
\$$u$$

\$$\text{Ptr := Ptr+1}$$
\n\$$\text{Ch := GetC[String, Ptr]}$$
\n\$$\text{test Ch \# "\text{"*"}}$$
\n\$$\text{then Out[S, Ch]}$$
\n\$$\text{or}$$
\n\$$\text{if Ptr > Length do}$$
\n\$$\text{OSReportN[421]; \text{return}}$$
\n\$$\text{Ch := GetC[String, Ptr]}$$
\n\$$\text{test Ch = "\text{"*"}}$$
\n\$$\text{then Out[S, "\text{"*"}] or}$$
§2  if i > 25  do
    return $  

    test Ch != 'I'
    then
    $3  let Function = valof
        switch Ch into
        $ case 'A': resultis OutAddr
        case 'B': resultis OutByte
        case 'C': resultis Out
        case 'N': resultis OutN
        case 'O': resultis OutO
        case 'S': resultis OutString
        default:
            OSReport[423,'OutString: ** *B',Ch]

    return $V
    Function[S, ParamList\$i]
    $3
    or
    $4  Ptr := Ptr+1
    if Ptr > Length do
        $ OSReportN[424]; return $
        Ch := GetC[String, Ptr]
        unless '0' < Ch < '9'  do
            $ OSReport[425,'OutString: **I *B', Ch]
            return $
            OutJustify[S, ParamList\$i, (Ch-"0")]
    $4
    i := i+1
    $u
    gas

and GetC[String, n] = (n rem 2 = 0) →
    (String\$(n/2))rshift 8, (String\$(n/2))^LMAEK
and OutJustify[S, n, FieldSize] be

let Sp = FieldSize - Size[n]
for i = 1 to Sp do Out[S, '*s']
OutN[S, n]

and Size[n] = (n > 9) → 1 + Size[n/10],
(n > 0) → 1,
(n / -32768) → 1 + Size[-n],

|| -32768 must be a special case
|| because of the machine representation.

and OSReport[n, String, a,b,c,d] be

Outs[ReportStream, 'OSReport *N: ',n]
Reports[String, a,b,c,d]

and OSReportN[n] be

Reports['OS Report *N', n]

****
This section defines the global output routines OutAddr, OutByte, OutN, OutO, Write, WriteAddr, WriteByte, WriteN and WriteO.

let OutP[S, n] be || outputs positive number
\[\text{if } n > 9 \text{ do } \text{OutP}[S, n/10] \]
\[\text{Out}[S, '0' + n \text{ rem } 10]\]

let OutN[S, n] be
\[\text{if } n < 0 \text{ do } \text{Out}[S, '0'] \]
\[\text{if } n = -32768 \text{ do } \text{OutString}[S, '32768']; \text{return } \]
\[\text{OutP}[S, n] \]
\[n := -n\]

let OutO[S, x] be || outputs 9 bits as octal number
\[\text{Out}[S, '0' + ((x \text{ rshift } 6)^7)]\]
\[\text{Out}[S, '0' + ((x \text{ rshift } 3)^7)]\]
\[\text{Out}[S, '0' + (x^7)]\]

let OutO[S, n] be
\[\text{OutO}[S, n \text{ rshift } 9] \]
\[\text{OutO}[S, n]\]

let OutByte[S, b] be
\[\text{Out}[S, b \text{ ^ } 8 \text{^ } 7]\]
let OutAddr[S, a] be
  $\text{OutByte}[S, a \text{ rshift } 8]$
  Out[S, 'a']
  OutByte[S, a]

let Write[x] be Out[Output, x]
let WriteN[n] be OutN[Output, n]
let WriteO[n] be OutO[Output, n]
let WriteByte[b] be OutByte[Output, b]
let WriteAddr[a] be OutAddr[Output, a]

****
This section defines the global functions \texttt{NextN} and \texttt{NextD}.

5 \texttt{let NextN[S] = valof}
4
\texttt{let Ch, n, Neg = 0, 0, false}
3

\texttt{repeat}
2
\texttt{if ('0' < Ch < '9') break}
1
\texttt{unless (Ch = 's' || Ch = '*')}
0
\texttt{do Neg := (Ch = '-')}

10 \texttt{repeat}

15 \texttt{repeatwhile ('0' < Ch < '9')} 

\texttt{PutBack[S, Ch]}

20 \texttt{resultis Neg -> n, n}

\texttt{and NextD[S] = valof}

25 \texttt{let Ch, n = 0, 0}

24 \texttt{until ('0' < Ch < '7') do Ch := NextCh[S]}

20 \texttt{repeatwhile ('0' < Ch < '7')}

25 \texttt{PutBack[S, Ch]}

30 \texttt{resultis n}

35 \texttt{and NextCh[S] = valof}

34 \texttt{if Endof[S] do}
33 \texttt{OSReportN[1441]; GiveUp[S]}

39 \texttt{resultis Next[S]}

40 \texttt{****}
II:5. Permanent input/output streams

II:5.1 TELETYPEx

II | This section defines the routines and functions
II | which are used in 'SETUPTT'.

5

let NextTT[S] = valof
SN let p = S↓LINERFPTR
S↓LINERFPTR := p+1
if p < S↓LINERFPTR result is rv p

10 II Otherwise refill the buffer.
§ let BuffStart = lv S↓LINEBUFFER
let BuffEnd = lv S↓TELESIZE
let x = 0
p := BuffStart
Out[S, RETLt]
ResetState[S]

§r x := State[S] repeat while x = NOTHINGTYPEd
20 ResetState[S]
switch on x into
§s default:
Out[S, x] || echo
rv p := x
p := p+1
if p > BuffEnd do
§ OutNewline[S]
break
§
30 endcase

35 case RETURNt:
case LINEFEEDt:
OutNewline[S]
rv p := '*n'
p := p+1
break

35 case ESCAPt: || terminate input
break || without 'newline'.
case RUBOUTt:
  unless p = BuffStart do
    \$ p := p - 1 \htrailing erase.\$
    Out[S, QUERYt]
  endcase

case XDFFt:
  p := BuffStart \hkill line.\n  Out[S, SHARPt]
  \hrun on into next case\n
case VTABt: \hdoes 'newline' on paper,\n  OutNewline[S] \hbut doesn't input it.\nendcase

\r
repeat

S\LINEBFFLAST := p
  \helement of buffer after last character\nS\LINEBFFPTR := BuffStart
resultis Next[S]
\n  \n
and OutNewline[S] be
\$
  Out[S, RETURNt]
  Out[S, LINEFEEDt]
\$

let OutTT[S, x] be
\$0 let i:=0
  S\OUTBUFF := x
  RjiExec[TRANSFER, Rj, lv i, 0, lv S\WRITEBLOCK]
\$0

let ResetTT[S] be
  S\LINEBFFPTR := S\LINEBFFLAST +1

let EndofTT[S] = false
\[
\text{let stateTT}[s] = \text{S\textbackslash{}INBUFF} \\
\text{let ResetStateTT}[s] be} \\
\;
\text{let } i \vdash \text{S\textbackslash{}INBUFF := NOTHING_TYPED} \\
\text{Exec[TRANSFER, L, lv i, o, lv S\textbackslash{}READBLOCK]} \\
\text{L: return} \\
\text{\textbf{end}}
\]

****
This section defines the global device stream function IntcodeToAndFromTeletype.

manifest

\[ TTSIZE = 8 \]
\[ TTSTR = 7 \]
\[ FLAG = 8 \]

let IntcodeToAndFromTeletype[Str] = valof

\[ let S = NewVec[TTSIZE] \]
\[ S\{NEXT\} := NextTele \]
\[ S\{OUT\} := OutTele \]
\[ S\{CLOSE\} := CloseTele \]
\[ S\{RESET\} := ResetTele \]
\[ S\{ENDOF\} := EndofTele \]
\[ S\{TTSTR\} := Str \]
\[ S\{SOURCE\} := Source[Str] \]

Resultis S

and NextTele[S] = valof

\[ let x = 0 \]
\[ \text{repeat until EvenParity}[S, x] \]
\[ x := x \wedge \text{PARITYMASK} \]

if x = NEWLINE then Resultis '*n'
if x = PRIME then Resultis PRIME
if x = UPARROW then Resultis '↑'
if ('*' < x < LEFTARROW) || ('a' < x < 'z') then Resultis x
if x = BELL then Resultis BELL

Resultis S

and EvenParity[S, x] = valof

\[ \text{if Parity}(x \text{ rightshift} 4) = \text{Parity}(x \wedge 817) \text{ then Resultis true} \]

\[ \text{OSReport}[521, 'Wrong TT parity'] \]

Resultis false
and OutTele[S, Ch] be
NOT if S = ExecConsole do
  $Reset[S]
  if Ch = '*n' do
    $OutCRLF[S, TTSTR]
    return
  $S$OUT, $S$FLAG := OutNewLines, false
  return
if Ch = '*4' do
  $for$ i=1 to 4 do Out[S, '*s']
  return
Ch := Ch $^ Ul\text{MASK}$ $/$ ignore underlining
unless ('*' < Ch < '.') $\lor$ ('a' < Ch < 'z') do
  Ch := $\text{valof}$
  $v$ switch on Ch into
  \begin{align*}
  \text{case } & \text{RETURN :} & \text{result} & \text{is Ch} \\
  \text{case } & \text{LEFTARROW :} & \text{result} & \text{is Ch} \\
  \text{case } & \text{DELETE :} & \text{result} & \text{is Ch} \\
  \text{case } & \text{PRIME :} & \text{result} & \text{is Ch} \\
  \text{case } & \text{GRAVE :} & \text{result} & \text{is Ch} \\
  \text{case } & \text{'^' :} & \text{result} & \text{is Ch} \\
  \text{case } & \text{'X' :} & \text{result} & \text{is Ch} \\
  \text{case } & \text{'|' :} & \text{result} & \text{is Ch} \\
  \text{case } & \text{DIV :} & \text{result} & \text{is Ch} \\
  \text{case } & \text{'^' :} & \text{result} & \text{is Ch} \\
  \text{case } & \text{RUNOUT :} & \text{result} & \text{is Ch} \\
  \text{default :} & \text{result} & \text{is Ch} \\
  \end{align*}
$v$
if Parity$(Ch \text{ shift 4}) \not= Parity$(Ch $^817$) do
  Ch := Ch $\lor$\text{ PARITYBIT}
  Out[S$\downarrow TTSTR$, Ch]
$\not=$

and OutCRLF[S] be
$\not=$ Out[S, RETURN$\not=$]
  Out[S, LINEFEED$\not=$]
$\not=$
and OutNewLines[S, Ch] be

\[
\text{if } \text{Ch} = "*n" \text{ do}
\]
\[
\text{S}\downarrow\text{FLAG} := \text{true}
\]
\[
\text{OutCRLF[S\downarrow\text{TTSTR}]}
\]
\[
\text{return}
\]
\[
\text{if S\downarrow\text{FLAG for } i = 1 \text{ to } 40 \text{ do Out[S\downarrow\text{TTSTR}, \text{RUNOUT]}}
\]
\[
\text{OutCRLF[S\downarrow\text{TTSTR}]}
\]
\[
\text{S\downarrow\text{OUT}} := \text{OutTele}
\]
\[
\text{OutTele[S, Ch]}
\]

and CloseTele[S] be

\[
\text{if S\downarrow\text{OUT} = OutNewLines do OutCRLF[S\downarrow\text{TTSTR}]}
\]
\[
\text{Close[S\downarrow\text{TTSTR}]}
\]
\[
\text{ReturnVec[S, TTSIZE]}
\]

and ResetTele[S] be

\[
\text{if S\downarrow\text{OUT} = OutNewLines do}
\]
\[
\text{S\downarrow\text{OUT}} := \text{OutTele}
\]
\[
\text{OutCRLF[S\downarrow\text{TTSTR}]}
\]
\[
\text{Reset[S\downarrow\text{TTSTR}]}
\]

and EndofTele[S] = Endof[S\downarrow\text{TTSTR}]

and source[Str] = valof

\[
\text{if Str < 0 do Str := (~ Str)\downarrow\text{SLOWBLOCK}}
\]
\[
\text{resultis Str\downarrow\text{SOURCE}}
\]

****
II:5.3 XFER

|| This section defines the routine InitiateTransfer.

\[
\text{manifest}
\]

\[
\begin{align*}
\text{||} & \quad \text{PAUSEO} = 200 \\
& \quad \text{PAUSE1} = 200
\end{align*}
\]

10 let InitiateTransfer[S, n, Message] be
\[
\begin{align*}
\text{let } i &= \text{PAUSEO} \\
\text{let } \text{Last} &= \text{PAUSE1} \times S/PINGS \\
\text{let HeldUp} &= \text{false} \\
\text{let } E &= \text{LV } S/\text{EXECBLOCK} \\
\text{\hline}
\end{align*}
\]

\[
\begin{align*}
\text{E\text{BUFFSIZE}} & \equiv n \\
\text{\hline}
\end{align*}
\]

\[
\begin{align*}
\text{let } r &= 0 \\
\text{\hline}
\end{align*}
\]

\[
\begin{align*}
\text{let } \text{HeldUp} &= \text{false} \\
\text{\hline}
\end{align*}
\]

\[
\begin{align*}
\text{\hline}
\end{align*}
\]

20 \text{break} \quad || \text{if transfer accepted.}

\[
\begin{align*}
\text{\hline}
\end{align*}
\]

\[
\begin{align*}
\text{Rj: test } 0 < i < \text{Last} \\
\text{then if } (i \text{ rem PAUSE1}) = 0 \quad \text{do Out[ExecConsole, "HELL"]} \quad \text{or } \quad \text{§ if } i = \text{Last do} \\
\text{§ HeldUp} & \equiv \text{true} \\
\text{§ ResetState[ExecConsole]} \\
\text{\hline}
\end{align*}
\]

\[
\begin{align*}
\text{\hline}
\end{align*}
\]

30 \text{§ if HeldUp} \wedge (\text{State[ExecConsole]} \equiv \text{NOTHINGTYPEDE}) \\
\text{do} \\
\text{§ Out[ExecConsole, } 's*n', \text{ Message]} \\
\text{§ ResetState[ExecConsole]} \\
\text{§§ i} \equiv i + 1 \\
\text{§§}
\]

35 \text{§§ repeat}

\[
\begin{align*}
\text{§§ E\text{BUFFER}} & \equiv (E\text{BUFFER})\downarrow(-1) \\
\text{§§}
\end{align*}
\]

40 \text{****}
This section defines the routines and functions used in 'SETUPPRDR' and the global routine TryAgain.

5 manifest
§ OPERABLE = 0
OFFLINE = -1
ONLINE = 0
PAUSE2 = 100

let ReadBuff[S] be
InitiateTransfer[S, READBBFSIZE+1, "Reader"]

and FirstNextBFPT[FB] = valof
§F let S = FB\SLOWBLOCK
Let E = lv S\EXECBLOCK
DEVICE := ReaderDev
ENDSTOPCH := NOTBYTE
ReadBuff[S]
S\NEXT := NextFillBFPT
FB\INBFPTR, FB\INBBFEND := EMPTY, EMPTY

let Rubbish = Next[-FB]
resultis Next[-FB]
§F

and NextFillBFPT[FB] = valof
§N let S = FB\SLOWBLOCK
ReadBuff[S]
FB\INBFPTR := S\(EXECBLOCK+BUFFER)
FB\INBBFEND := FB\INBFPTR + READBBFSIZE
resultis Next[-FB]
§N

and ResetBFPT[FB] be
§R let S=FB\SLOWBLOCK
FB\INBFPTR, FB\INBBFEND := EMPTY, EMPTY
S\NEXT := ReaderOffLine[] → FirstNextBFPT, NextWaitBFPT
§R
II:5.4

45 \textbf{and} \textsf{NextWaitBFPT[FB]} = \textsf{valof} \newline
\hspace{1em} \textsf{let} \ S = \textsf{FB\Downarrow\textit{SLOWBROWK}} \newline
\hspace{1em} \textsf{let} \ i, \ Ch = 0, 0 \newline
\hspace{1em} \textsf{let} \ \textsf{NeverOff} = \textsf{true} \newline
\hspace{1em} \textsf{ResetState[ExecConsole]} \\
50 \textsf{let} \ \textsf{test ReaderOffLine[]} \newline
\hspace{1em} \textsf{then} \ \textsf{NeverOff} := \textsf{false} \newline
\hspace{1em} \textsf{or unless} \ \textsf{NeverOff} \ \textsf{break} \newline
\hspace{1em} \textsf{Ch} := \textsf{State[ExecConsole]} \\
55 \hspace{1em} \textsf{if} \ ((\textsf{Ch=RETURN}) \lor (\textsf{Ch=LINEFEED})) \ \textsf{^ NeverOff} \ \textsf{break} \\
\hspace{1em} \textsf{test} \ O < i < \textsf{PAUSE2} \times \textsf{SPINGS} \newline
\hspace{1em} \textsf{then if} \ (i \ \textsf{rem} \ \textsf{PAUSE2}) \Rightarrow \textsf{do} \textsf{Out[ExecConsole, BELL]} \newline
\hspace{1em} \textsf{or if} \ \textsf{Ch} \Rightarrow \textsf{NOTHINGTYPED} \newline
60 \hspace{1em} \textsf{do} \textsf{Outs[ExecConsole, 'Reader*nt']}
\hspace{1em} \textsf{if} \ \textsf{Ch} \Rightarrow \textsf{NOTHINGTYPED} \textsf{do} \textsf{ResetState[ExecConsole]}
\hspace{1em} \textsf{i} := (i < 0) \Rightarrow 0, i+1 \\
\hspace{1em} \textsf{let} \ \textsf{TryAgain[FB]} \newline
\hspace{1em} \textsf{repeat} \\
65 \hspace{1em} \textsf{resultis} \textsf{FirstNextBFPT[FB]} \\
\hspace{1em} \textsf{and} \ \textsf{StateBFPT[S]} = \textsf{ReaderOffLine[]} \Rightarrow \textsf{OFFLINE}, \textsf{ONLINE} \\
70 \hspace{1em} \textsf{and} \ \textsf{ReaderOffLine[]} = \textsf{valof} \newline
\hspace{1em} \textsf{let} \ \textsf{Status} := \textsf{0} \\
\hspace{1em} \textsf{Exec[LOOKATRDR, L, lV Status]} \\
75 \hspace{1em} \textsf{resultis} \textsf{Status} \neq \textsf{OPERABLE} \\
\hspace{1em} \textsf{L: resultis true} \\
\hspace{1em} \textsf{let} \ \textsf{EndofBFPT[FB]} = \textsf{false} \\
80 \hspace{1em} \textsf{let} \ \textsf{TryAgain[F]} \ \textsf{be} \newline
\hspace{1em} \textsf{sta unless} \ F = \textsf{BytesfromPT} \ \textsf{do} \\
\hspace{2em} \textsf{OSReport[541, 'TryAgain on wrong stream']} \\
\hspace{2em} \textsf{GiveUp[F]} \\
\hspace{1em} \textsf{}}
§ let FB = ~F

let S = FB: SLOWBLOCK
let E = lv S: EXECBLOCK

unless S: CLOSE = S: RESET do

§ OSReportN[542]; GiveUp[F] § || PutBack on BFPT

OutS[ExecConsole, 'Try again*n']

§ let Ptr, End = FB: INBBFPTR, FB: INBBFEND

§ NEXT := FirstNextBFPT

FB: INBBFPTR, FB: INBBFEND := EMPTY, EMPTY

§ DEVICE := RDRDEVSUM - ReaderDev || change direction

§ let k = End - Ptr

let First = (k < 1) -> End-5,
     (Ptr-3 < E: BUFFER) -> E: BUFFER, Ptr-3

for i = First to First+4 do

§ if OutByte[ExecConsole, ry i]

if i = Ptr-1 do OutS[ExecConsole, * wrong*]

Out[ExecConsole, *n*]

§ let x = ry (Ptr-1) || offending character

let Message = 'Backread'

InitiateTransfer[S, 2, Message]

InitiateTransfer[S, READBFFSIZE+1, Message]

§ let y = 0

let Ly = E: BUFFER + k

ry Ly := NOTBYTE

InitiateTransfer[S, k+2, Message]

y := ry Ly repeatwhile y = NOTBYTE

test x = y

then OutS[ExecConsole, 'Char still same*']

or OutS[ExecConsole, 'Char now *B*, y]

Wait[]

§TA

****
This section defines the global stream function WordsfromDiads, and the global routine TryDiadAgain.

**manifest**  

```plaintext
DSIZE = 5  
DIADBFSIZE = 30  
DIADBFF = 6
```

**Diad Characters**

- STOPCODE = 4  
- ESC = 833  
- ERASE = 8377  
- WCH = 8125

```plaintext
let WordsfromDiads[Str] = valof
let FB = NewVec[FBSIZE]
let S = NewVec[DSIZE + DIADBFSIZE]

S↓NEXT := NextBlock
S↓ENDOF := EndofW
S↓CLOSE := CloseW
S↓RESET := ResetW
S↓STR := Str
S↓OUT := StreamError

FB↓SLOWBLOCK := S
FB↓INBBPPTR, FB↓INBBFEND := EMPTY, EMPTY
FB↓OUTBBPPTR, FB↓OUTBBFEND := EMPTY, EMPTY
FB↓INESC, FB↓OUTESC := NOTUSED, NOTUSED
```

**resultis** ~FB
and NextBlock[FB] = valof
let S = FB\$SLOWBLOCK
let B = lv S\$DIADBFF
let I = S\$STR

let Checksum, Blocksize = 0, 0
let Ch, LByte, RByte = 0, 0, 0

Ch := NextCh[I] repeatwhile Ch=NULL
if Ch = WCH break
test Ch = STOPCODE
then Reset[I]
or § OSReport[551, 'Wrong diad WCH']
TryAgain[I]

Blocksize := NextByte[I]
if Blocksize > DIADBFFSIZE do
§ OSReportN[552]; GiveUp[Blocksize] §
Checksum := Blocksize

for i = 0 to Blocksize-1 do
§f LByte := NextByte[I]
RByte := NextByte[I]
B[i] := (LByte lshift 8)\$ RByte
Checksum := Checksum + LByte + RByte
§f

Ch := NextByte[I]
Ch := (Ch lshift 8)\$ NextByte[I]
if Ch = Checksum break
OSReport[553, 'Diad sumcheck failed']
TryDiadAgain[I, Ch, Checksum]

FB\$INFFPTR := B
FB\$INBEND := B + Blocksize
resultis Next[~FB]
and NextByte[S] = valof
\$ let x = NextCh[S]
\$ resultis (x = ESC) \rightarrow (NextCh[S]+1), x

and NextCh[S] = valof
\$ let x = 0
\$ x := Next[S] \text{ repeatwhile} x = ERASE
\$ resultis x

and EndofW[FB] = Endof[(FB↓SLOWBLOCK)↓STR]

and ResetW[FB] be
\$R let \ S = FB↓SLOWBLOCK
\ RFB↓INBFPPTR, FB↓INFFEND := EMPTY, EMPTY
\ R Reset[\ S↓STR]

and CloseW[FB] be
\$C let \ S = FB↓SLOWBLOCK
\ C Close[\ S↓STR]
\ C ReturnVec[\ S, DSIZE + DIADBFFSIZE]
\ C ReturnVec[FB, FBSIZE]
\$C
let TryDiadAgain[F, Read, Comp] be

unless F = BytesfromPT do
  $DSreportN[554]; GiveUp[F]$

let FB = ~F
let $S = FB\downarrow SLOWBLOCK$

OutS[ExecConsole, "Try diad again n"]

let k = FB\downarrow INBFEND - FB\downarrow INBFPTR
$\downarrow NEXT := FirstNextBFPT$
$FB\downarrow INBFPTR, FB\downarrow INBFEND := EMPTY, EMPTY$

ReadBackwards[S, 3]
ReadBackwards[S, READBFFSIZE]
ReadBackwards[S, k+1]

E\downarrow STOPCH := WCH
$SR S\downarrow BFF\downarrow ENDE := false$
$S\downarrow BFF\downarrow END := false$
ReadBackwards[S, READBFFSIZE]
$SR$ repeatwhile S\downarrow BFF\downarrow ENDE v S\downarrow BFF\downarrow END
E\downarrow STOPCH := NOTBYTE

OutS[ExecConsole, "Sum read *0, computed *0", Read, Comp]

and ReadBackwards[S, n] be

let Message = "Backread"
let E = lv S\downarrow EXCBLOCK

E\downarrow DEVICE := RDRDEVSUM - ReaderDev
E\downarrow COMPLETED := false

InitiateTransfer[S, n+1, Message]
$SR$ repeatuntil E\downarrow COMPLETED

****
This section defines the global routine OutDateAndTime, and the global function TimeOfDay.

static §§
    ClockRestarted = false
 §§

let OutDateAndTime[S] be
    $DT Out[S, "*********** '"]
    OutDate[S, FetchCode[DATE]]
    OutTime[S]
    Out[S, "*n'"]

and OutDate[S, d] be
20    Out[S, "*N*N*N'", d rshift 11, (d rshift 7) rshift 17, d rshift 77]

and OutTime[S] be
25     $DT let t = TimeOfDay[]
      if ClockRestarted ^ (S = Console) return
      let h, m = t / 60, t rem 60
      Out[S, "*N*N*N'", h, m / 10, m rem 10]
     $DT

let TimeOfDay[] = valof
8T     let Clock = FetchCode[TIMEOFDAYCLOCK]
35     ClockRestarted := false
     RestartClock[Clock]
     result is MINS PERDAY - (Clock \ BUFFER) \ TIME
8T
and RestartClock[E] be
§R let i=0
    Exec[TRANSFER, Rj, lv i, 0, E]
    ClockRestarted := true
    (E\BUFFER)\TIME := MINSECOND - AskTime[Console]
Rj:|| assumed already going.
§R

and AskTime[Con] = valof
  $A OutS[Con, "*ntime *q"]
  § let Hrs = NextN[Con]
  let Mins = NextN[Con]
  resultis Hrs X 60 + Mins

****
This section defines the global routines AddressZero, Copy, EqS, NullProgram, and Wait.

5 let AddressZero[] be
   OSReport[621, 'Jump to address Zero']
   GiveUp[0]

10 let Copy[V1, V2, n] be
   let FB = vec FBSIZE
   FB\INBFFP, FB\INRFEND := V1, V1+n
   FB\INESC := NOTUSED
   TransferIn[~FB, V2, n]

20 let EqS[p, q] = valof
   unless p\0 = q\0 result is false
   for i = 1 to (p\0 rshift 8)/2 do
     unless p\i = q\i result is false
     result is true

25 let NullProgram[] be $$

30 let Wait[] be
   let Ch = 0
   Reset[Console]
   Ch := Next[Console] repeat until Ch = "*n"
This section defines the global routines CoretoDisc and DiscToCore.

manifest \$ OFFSET = -10 \$

let CoretoDisc[v, p] be
\$CD\$ unless \( v = \text{DiscPage} \) do Copy[v, DiscPage, PAGESIZE]
\$CD\$
\( \text{DiscPage} \text{SUMCH} := \text{Sumcheck[\]} \)
\( \text{unless} \text{FetchCode[DISCWRITEPERMITTED]} \text{ return} \)
\( \text{DiscTransfer[DISCWRITE, p]} \)

and DiscToCore[v, p] be
\$DC\$
\( \text{\textbf{Sr}} \text{ DiscTransfer[DISCREAD, p]} \)
\( \$ \text{let} x, y = \text{Sumcheck[\]} , \text{DiscPage} \text{SUMCH} \)
\( \text{if} (x=y) \text{vFetchCode[SUMCHINHIBITED]} \text{ break} \)
\( \text{Outs[Console, 'Sumcheck page *N: computed *O,*s} \)
\( \text{recorded *O '}, p, x, y \}
\( \text{Reset[Console]} \)
\( \text{\textbf{Sr}} \text{ repeatuntil Next[Console] = 'I'} \)
\( \text{unless} \text{v = DiscPage do Copy[DiscPage, v, PAGESIZE]} \)
\$DC$

and DiscTransfer[Dev, p] be
\$ST\$
\( \text{let} E = \text{DiscPage} + \text{OFFSET} \text{ } \text{ExecBlock for Disc} \)
\( \text{EDEVICE} := \text{Dev} \)
\( \text{\textbf{Sr}} \text{ for i = 1 to 2 do} \)
\( \$ \text{E\text{PAGENUMB} := p} \)
\( \text{E\text{COMPLETED} := false} \)
\( \text{Exec[TRANSFER, Rj, l v E\text{PAGENUMB, E\text{PAGENUMB, E}}} \)
\( \text{\$ \$ repeatuntil E\text{COMPLETED} } \)
40 if $E\uparrow$PAGENUMB = $p$ return  || successful transfer

|| If the transfer ends unsuccessfully, Exec
|| overwrites $E\uparrow$PAGENUMB with a failure number.

45 if $i = 2$ break  || out of $§f$
Message[ExecConsole, $E\uparrow$PAGENUMB, $p$]
OutDateandTime[ExecConsole]

§f
50 Rj: Message[Console, $E\uparrow$PAGENUMB, $p$]
Reset[Console]
§r repeatuntil Next[Console] = 'I'
§t

55 and Message[S, $f$, $p$] be
OutS[S, "Disc failure *D page *N ", $f$, $p$]

****
This section defines the routines NewDiscBlock and ReturnDiscBlock.

```
let NewDiscBlock[] = valof

let Scan = FSF\SCAN.
  test Scan = PAGEEND
  then
  test FSF\BACKLINK = ENDBODY
  then $ OSReport[721, 'Disc full']
  GiveUp
  or
  let Addr = FSF\PAGE

  let Link = FSF\BACKLINK
  DisctoCore[FSF, Link]
  FSF\NXTPAGE := ENDBODY
  CoretoDisc[FSF, Link]
  FSF\PAGE := Link
  FSF\SCAN := FIRSTENTRY

  let d = FetchCode[DATE]
  UpdateHead[FSF\THISFILE, SAME, FSF\SERIAL, Link, SAME, d, d]
  resultis Addr

  or
  let Addr = FSF\Scan
  FSF\SCAN := Scan + 1
  unless Addr = CLAIMED do
  FSF\Scan := CLAIMED

  DisctoCore[DiscPage, Addr]
  if DiscPage\O = FREE resultis Addr

  repeat
```
let ReturnDiscBlock[Addr] be

let Scan = FFS\downarrow SCAN
test Scan = FIRSTENTRY

then

\[\text{FSF}\downarrow \text{NXTPAGE} := \text{Addr}\]
\[\text{CoretoDisc}[\text{FSF}, \text{FSF}\downarrow \text{PAGE}]\]
\[\text{FSF}\downarrow \text{SERIAL} := \text{FSF}\downarrow \text{SERIAL} + 1\]
\[\text{FSF}\downarrow \text{NXTPAGE} := \text{ENDBODY}\]
\[\text{FSF}\downarrow \text{BACKLINK} := \text{FSF}\downarrow \text{PAGE}\]
\[\text{FSF}\downarrow \text{PAGE} := \text{Addr}\]
\[\text{FSF}\downarrow \text{SCAN} := \text{PAGEEND}\]

for i = FIRSTENTRY to LASTENTRY do FFS\downarrow i := CLAIMED

CoretoDisc[FSF, Addr]

let d = FetchCode[DATE]

UpdateHead[FSF\downarrow \text{THISFILE}, \text{SAME}, FFS\downarrow \text{SERIAL},
\text{Addr, SAME, d, d}]

or

\[\text{FSF}\downarrow (\text{Scan} - 1) := \text{Addr}\]
\[\text{FSF}\downarrow \text{SCAN} := \text{Scan} - 1\]
\[\text{DiscPage}\downarrow 0 := \text{FREE}\]

CoretoDisc[DiscPage, Addr]

****
\[\text{manifest} \ $EFSIZE = 5$ \]

\[\text{let CheckType}[f, h] = \text{valof} \]
\[\text{let OuttoFile}[f] = \text{valof} \]
\[\text{let EntriesfromFile}[f] = \text{valof} \]

\[\text{let CheckType}[f, h] \text{ be} \]
\[\text{if } h = \text{NULL do} \]
\[\text{OSReport}[731, 'File deleted'] \]
\[\text{GiveUp}[f] \]$\]
\[\text{unless } (h \uparrow \text{TYPE} \land \text{STREAMABLE}) = \text{STREAMABLE do} \]
\[\text{OSReportN}[732] \]
\[\text{GiveUp}[f] \]
\$\text{IfNUMB} = f
(S \downarrow \text{IFBUFFER}) \downarrow \text{NXTPAGE} := \text{FirstPage}

\text{resultis} \sim FB
\text{IF}

50 \text{and NextBuffIF}[FB] = \text{valof}
\text{SN if EndofIF}[FB] \text{resultis} \text{ENDOFSTREAMCH}
\quad || \text{EndofIF refills buffer}
\text{resultis} \text{Next}[-FB]
\text{SN}

55 \text{and EndofIF}[FB] = \text{valof}
\text{SE let} S = FB \downarrow \text{SLOWBLOCK}
\text{let} B = 1v S \downarrow \text{IFBUFFER}
60 \text{if} (B \downarrow \text{NXTPAGE} = \text{NULLBODY}) \vee (B \downarrow \text{NXTPAGE} = \text{ENDBODY}) \text{then}
\quad \text{DIsctoCore}[B, B \downarrow \text{NXTPAGE}]
\quad FB \downarrow \text{INBFPPTR} := 1v B \downarrow \text{FIRSTDATA}
\quad FB \downarrow \text{INBFEND} := (B \downarrow \text{NXTPAGE} = \text{ENDBODY}) \rightarrow
\quad 1v B \downarrow (B \downarrow \text{ENDOFDATAPTR}), (1v B \downarrow \text{LASTDATA}) + 1
\text{resultis} (FB \downarrow \text{INBFPPTR} = FB \downarrow \text{INBFEND})
\text{SE}

70 \text{and CloseIF}[FB] be
\text{Sec ReturnVec}[FB \downarrow \text{SLOWBLOCK}, \text{IFSIZE}]
\text{ReturnVec}[FB, \text{FB SIZE}]
\text{Sec}

75 \text{and ResetIF}[FB] be
\text{SR let} S = FB \downarrow \text{SLOWBLOCK}
\text{let} h = \text{FindHeading}[S \downarrow \text{FILENUMB}]
80 (S \downarrow \text{IFBUFFER}) \downarrow \text{NXTPAGE} := h \downarrow \text{FIRSTPAGE}
\text{ReturnVec}[h, h \downarrow \text{^ IMASK}]
\quad FB \downarrow \text{INBFPPTR}, FB \downarrow \text{INBFEND} := \text{EMPTY}, \text{EMPTY}
\text{SR}

85
let EntriesFromFile[f] = valof 

90  let S = NewVec[EFSIZE]
    let h = FindHeading[f]
    unless h\TYPE = INDEX do
      OSReport[733, 'Not index file']
      GiveUp[f] §
      ReturnVec[h, h:0 ^ LMASK]

S\NEXT := NextEF
S\ENDOF := EndofEF
S\CLOSE := CloseEF
S\RESET := ResetEF
S\OUT := StreamError
S\STR := InfromFile[f]

resultis S

end NextEF[S] = valof

$N  let F = S\STR \[ | ] = InfromFile[f]
110  let n = Next[F]
    if n = ENDOFSTREAMCH resultis n
      let Entry = NewVec[n]
      Entry\FO := n
      TransferIn[F, ly(Entry\FO), n]
115  resultis Entry

$N

end EndofEF[S] = Endof[S\STR]

120

end CloseEF[S] be

$C  Close[S\STR]
    ReturnVec[S, EFSIZE]
125 $C

end ResetEF[S] be Reset[S\STR]

****
This section defines the global Stream function
OuttoFile, and also the global routine DeleteBody.

They are defined in the same section because they
both use the non-global routines ReturnChain
and CheckPerm.

let ReturnChain[f, Page] be
$\mathcal{RC}
\begin{align*}
\text{DiscToCore[DiscPage, Page]} & \text{ unless DiscPage.THISFILE = f do} \\
\text{OSReport[741, 'Page chain wrong - file *N', f]} & \\
\text{GiveUp[Page]} & \\
\text{let p = DiscPage.NXTPAGE} & \\
\text{ReturnDiscBlock[Page]} & \\
\text{Page := p} & \\
\text{repeat until Page = ENDBODY} & \\
\end{align*}$

let CheckPerm[f, h] be
$\mathcal{GC}
\begin{align*}
\text{let p = h.PERM} & \text{ unless (p=UNRESTRICTED) \land (p=OWNERONLY \land User=h.OWNER) do} \\
\text{OSReport[742, 'Protected file']} & \\
\text{GiveUp[f]} & \\
\end{align*}$

let DeleteBody[f] be
$\mathcal{GD}
\begin{align*}
\text{let h = FindHeading[f]} & \text{ if h = NULL do} \\
\text{OSReport[743]; GiveUp[f]} & \\
\text{CheckPerm[f, h]} & \\
\text{let p = h.FIRSTPAGE} & \text{ unless p = NULLBODY do} \\
\text{UpdateHead[f, NULLBODY, 0, NULLBODY, SAME, FetchCode[DATE], SAME]} & \\
\text{ReturnChain[f, p]} & \\
\end{align*}$
62

Charge[\(h \downarrow \text{OWNER}, - h \downarrow \text{NUMPAGES}\)]

§1

ReturnVec[\(h, h \downarrow 0 \wedge \text{LMASK}\)]

§D

50 let OutToFile[\(f\)] = valof

let FB = NewVec[\(\text{FSIZE}\)]
let S = NewVec[\(\text{FSIZE}\)]
let B = lv S \downarrow \text{BUFFER}

let \(h\) = FindHeading[\(f\)]
CheckType[\(f, h\)]
CheckPerm[\(f, h\)]

FB \downarrow \text{SLOWBLOCK} := S
FB \downarrow \text{OUTBFPPTR} := lv B \downarrow \text{FIRSTDATA}
FB \downarrow \text{OUTBFPEND} := (lv B \downarrow \text{LASTDATA}) + 1
FB \downarrow \text{INBFPPTR}, FB \downarrow \text{INBFPEND} := \text{EMPTY, EMPTY}
FB \downarrow \text{INESC}, FB \downarrow \text{OUTBSC} := \text{NOTUSED, NOTUSED}

S \downarrow \text{NEXT} := \text{StreamError}
S \downarrow \text{OUT} := \text{OutBufOf}
S \downarrow \text{CLOSE} := \text{CloseOf}
S \downarrow \text{ENDOF} := \text{StreamError}
S \downarrow \text{RESET} := \text{ResetOf}

S \downarrow \text{FIENUMB} := f
S \downarrow \text{FILEOWNER} := h \downarrow \text{OWNER}
S \downarrow \text{OLDBODY} := h \downarrow \text{FIRSTPAGE}
S \downarrow \text{OLDSIZE} := h \downarrow \text{NUMPAGES}
S \downarrow \text{OFPAGE} := \text{NewDiscBlock[\(\)]}
S \downarrow \text{NEWBODY} := S \downarrow \text{OFPAGE}

ReturnVec[\(h, h \downarrow 0 \wedge \text{LMASK}\)]

80

B \downarrow \text{SERIAL} := 1
B \downarrow \text{THISFILE} := f
B \downarrow \text{NXTPAGE} := \text{ENDBODY}
CoreToDisc[B, S \downarrow \text{OFPAGE}] || for safety

§ let \(C = lv S \downarrow \text{UCHAIN}\)
CoreToRoutine := FailClose
C↓CSUC := ClearUpChain
C↓CPRE := l↓ ClearUpChain

ClearUpChain := C
unless C↓CSUC = ENDCUCHN do (C↓CSUC)↓CPRE := C

resultis ~FB

end

and OutBuffOF [FB] be

let S = FB↓SLOWBLOCK

let B = l↓ S↓CFBUFFER

TurnPage[S, NewDiscBlock[], B]

FB↓OUTBFPTR := l↓ B↓FIRSTDATA

done

and TurnPage[S, NextPage, B] be

B↓NXTPAGE := NextPage
CoretoDisc[B, S↓OFFPAGE]

S↓OFFPAGE := NextPage

B↓SERIAL := B↓SERIAL + 1
done

and CloseOF [FB] be

let S = FB↓SLOWBLOCK

let C = l↓ S↓CUCHAIN

(C↓CPRE)↓CSUC := C↓CSUC
unless C↓CSUC = ENDCUCHN do (C↓CSUC)↓CPRE := C↓CPRE

ResetOF[FB]
ReturnDiscBlock[S↓OFFPAGE]
ReturnVec[S, OFSIZE]
ReturnVec[FB, FBSIZE]
done

and ResetOF[FB] be

let S = FB↓SLOWBLOCK
let B = lv S↓OFBUFFER

135 if (S↓OFPAGE = S↓NEWBODY) && (FB↓OUTBFFPTR = lv B↓FIRSTDATA)
   || i.e., if nothing output
   do
      S↓OLDBODY := NULLBODY
      S↓OLDSIZE := 0
      DeleteBody[S↓FILENAME]
      return
   end

§ let NumbPages = B↓SERIAL
145 Charge[S↓FILEOWNER, NumbPages - S↓OLDSIZE]

B↓ENDOFDATAPTR := FB↓OUTBFFPTR - B
B↓NEXTPAGE := ENDBODY
CoretoDisc[B, S↓OFPAGE] || write last page

150 UpdateHead[S↓FILENAME, S↓NEWBODY, NumbPages, S↓OFPAGE,
               SAME, FetchCode[DATE], SAME]

unless S↓OLDBODY = NULLBODY do
   ReturnChain[S↓FILENAME, S↓OLDBODY]

   S↓OLDBODY := S↓NEWBODY
   S↓OLDSIZE := NumbPages
   S↓OFPAGE := NewDiscBlock[]

160 S↓NEWBODY := S↓OFPAGE
   B↓SERIAL := 1
   FB↓OUTBFFPTR := lv B↓FIRSTDATA
   and FailClose[C] be
§
   let S = C - CUCHAIN
   let B = lv S↓OFBUFFER
170 B↓NEXTPAGE := ENDBODY
   CoretoDisc[B, S↓OFPAGE] || Complete the chain,
   ReturnChain[S↓FILENAME, S↓NEWBODY]|| and return it.
§

****
II:8. Disc routines (2)

II:8.1 CHARGE

\[
\text{let } \text{Charge}[	ext{User, NumberofPages}] \text{ be } \$ \text{return } \$
\]

****
```
let Update[f, Word, Value] be
let Incrementing = (Word < 0)
let Element = Incrementing → -Word, Word

let h = FindHeading[f]
if h = NULL do
  § OSReport[821, 'Heading deleted']
  GiveUp[f] §

§ let HdLength = h↓0 ^ LMASK
unless Element ≤ HdLength do
  § OSReportN[822]
  GiveUp[Word] §
  ReturnVec[h, HdLength]

§ let HdAddr = LookUpinMFL[f]
let DiscWord = HdAddr↓1WORD + Element
DiscCore[DiscPage, HdAddr↓HPAGE]
if DiscWord > LASTDATA do
  §1 let NextPage = DiscPage↓NPAGE
  if NextPage = ENDBODY do
    § OSReportN[823]
    GiveUp[f] §
    DiscCore[DiscPage, NextPage]
  HdAddr↓HPAGE := NextPage
  DiscWord := DiscWord - DATASIZE
  §1

DiscPage↓DiscWord :=
  Incrementing → (DiscPage↓DiscWord)↓Value, Value
CoreToDisc[DiscPage, HdAddr↓HPAGE]
ReturnVec[HdAddr, ENTRYSIZE]
§

***
```
This section defines the global routine UpdateHead.

5 let UpdateHead[f, FirstPage, NumPages, LastPage, Type, DateLastChanged, DateLastRead] be

§UH

let HdAddr = LookupMFL[f]

let ParamList = lv f

DiscToCore[DiscPage, HdAddr↓HPAGE]

for i=1 to 6 unless ParamList↓i = SAME do

§E

let Element = valof

$V switchon i into

§ case 1 : resultis FIRSTPAGE

case 2 : resultis NUMPAGES

case 3 : resultis LASTPAGE

case 4 : resultis TYPE

case 5 : resultis DLW

case 6 : resultis DLR

$V

§ let DiscWord = HdAddr↓WORD + Element

if DiscWord > LASTDATA do

§ let NextPage = DiscPage↓NEXTPAGE

if NextPage = ENDBODY do

§ OSReportN[831]

GiveUp[f] §

CoreToDisc[DiscPage, HdAddr↓HPAGE]

DiscToCore[DiscPage, NextPage]

HdAddr↓HPAGE := NextPage

HdAddr↓WORD := HdAddr↓WORD - DATASIZE

DiscWord := DiscWord - DATASIZE

§ let DiscPage↓DiscWord := ParamList↓i

$E

CoreToDisc[DiscPage, HdAddr↓HPAGE]

ReturnValue[ H dAddr, ENTRYSIZE ]

40 §UH

****
This section defines the global functions FindHeading and LookUpinMFL.

```ml
let LookUpinMFL f =
  valof
  if f < 0 result  is NULL
  DiscToCore [DiscPage, FetchCode [MFLFIRSTPAGE]]

let i = 0
  until f ≤ i + ENTRIESPERPAGE do
    if DiscPage\NXTPAGE = ENDBODY result  is NULL
    DiscToCore [DiscPage, DiscPage\NXTPAGE]
    i := i + ENTRIESPERPAGE

let w = (ENTRYSIZE+1)×((f-1)-i) + FIRSTDATA
  if DiscPage\w = NUNENTRY result  is NULL

let Entry = NewVec [ENTRYSIZE]
  Copy [lv (DiscPage\w), Entry, ENTRYSIZE+1]
  result  is Entry

let FindHeading f =
  if
    let HdAddr = LookUpinMFL f
    if HdAddr = NULL do
      OSReport [841, "No entry in MFL"]
      GiveUp f
  then
    DiscToCore [DiscPage, HdAddr\HPAGE]
    let Word = HdAddr\HWORD
    ReturnVec [HdAddr, ENTRYSIZE]
```
\$ \textbf{let} \ Length = \text{DiscPage} \downarrow \text{Word} \wedge \text{LMASK} \\
\textbf{let} \ h = \text{NewVec}[\text{Length}] \\
\textbf{let} \ n = \text{LASTDATA} - \text{Word} \\
\text{test} \ Length < n \\
\text{then} \ \text{Copy}[\downarrow \text{DiscPage} \downarrow \text{Word}, \ h, \ Length+1] \\
\text{or} \ \$1 \\
\text{let} \ \text{NextPage} = \text{DiscPage} \downarrow \text{NXTPAGE} \\
\text{if} \ \text{NextPage} = \text{ENDBODY} \ \text{do} \\
\$ OSReportN[842] \\
\text{GiveUp}[f] \ \$ \\
\text{Copy}[\downarrow \text{DiscPage} \downarrow \text{Word}, \ h, \ n+1] \\
\text{DiscToCore}[\text{DiscPage}, \ \text{NextPage}] \\
\text{Copy}[\downarrow \text{DiscPage} \downarrow \text{FIRSTDATA}, \ downarrow \text{h} \downarrow (n+1), \ \text{Length}-n] \\
\$1 \\
\text{if} \ \text{h \downarrow \text{TYPE} = \text{DELETED \ do} \\
\$2 \ \text{ReturnVec}[h, \ \text{Length}] \\
\text{resultis NULL} \\
\$2 \\
\text{resultis h} \\
65 \ \$F \\
\text{****}
This section defines the global function LookUp

\[ \text{manifest } \text{ENTRYLENGTH} = 20 \]

\[ \text{let } \text{LookUp}[\text{Name1, Name2, i}] = \text{valof} \]

\[ \text{if } i = \text{NULL } \text{result is NULL} \]
\[ \text{let } h = \text{FindHeading}[i] \]
\[ \text{if } h = \text{NULL } \text{do} \]
\[ \text{OSReport}[\text{"Index deleted"}] \]
\[ \text{GiveUp}[i] \]
\[ \text{unless } h \text{\#TYPE = INDEX } \text{do} \]
\[ \text{OSReport}[\text{"File not index"}] \]
\[ \text{GiveUp}[i] \]
\[ \text{ReturnVec}[h, h \text{\#0 } \text{\# LMASK}] \]
\[ \text{let } s = \text{InfromFile}[i] \]
\[ \text{let } \text{Length} = \text{ENTRYLENGTH} \]
\[ \text{let } v = \text{NewVec}[\text{Length}] \]
\[ \text{until } \text{Endof}[s] \text{ do} \]
\[ \text{let } n = \text{Next}[s] \]
\[ \text{if } n > \text{Length } \text{do} \]
\[ \text{ReturnVec}[v, \text{Length}] \]
\[ \text{Length} := n \]
\[ \text{v} := \text{NewVec}[\text{Length}] \]
\[ \text{TransferIn}[s, v+i, n] \]
\[ \text{unless } v \text{\#STATUS = DELETED } \text{do} \]
\[ \text{if } \text{EqS}[\text{Name1}, v + v \text{\#N1}] \land \text{EqS}[\text{Name2}, v + v \text{\#N2}] \text{ do} \]
\[ \text{Close}[s] \]
\[ \text{let } \text{NotLinked} = (v \text{\#LINKING} > 0) \]
\[ \text{let } f = \text{NotLinked } \rightarrow v \text{\#FILE}, \]
\[ \text{LookUp}[v + v \text{\#N3}, v + v \text{\#N4}, \]
\[ \text{LookUp}[v + v \text{\#N5}, v + v \text{\#N6}, \text{SystemIndex}]] \]
\[ \text{ReturnVec}[v, \text{Length}] \]
\[ \text{result is } f \]
\[ \text{Close}[s] \]
\[ \text{ReturnVec}[v, \text{Length}] \]
\[ \text{result is NULL} \]

****
This section defines the global routines LoadFile and LoadSystemFile.

let LoadFile[f] be

let i = In
let n = MaxVecSize[] - INFILESIZE
let v = NewVec[n]

In := InFromFile[f]
ReturnVec[v, n]

Load[i] ;; from In
Close[In]
In := i

let LoadSystemFile[String] be

let f = LookUp[String, 'IC', SystemIndex]
if f = NULL do
  OSReport[501, '*'*5*'*IC'* not system file', String]
  GiveUp[0]

LoadFile[f]

****
II:9. Special functions in WIC

DEC 6; DEC 0;  \| NEWSECTION

DEC 1; DEC 17;  \| CODE, 17 words:

*12; IP2; LDI PRG; OSY; ST; EXIT;  \| FetchCode (global 2)
5 *13; OP3; LDC; IP2; STHR; EXIT;  \| StoreCode (global 3)
  *14; EXEC; EXIT;  \| Exec (global 4)

*15; OG335; SUMCK;  \| SumCheck (global 15)
  \| (global 335 = DiscPage)

10 *17; NEXT;
  *16; OUTZ;

*18; ENDOC;
  *13; GET; *PUTZ;

15 *14; GET; *PUTY;
  *17; TROT;

*100; DEC 2; DEC 21;  \| INTERLUDE, 21 words:
  *101; OG8; LDC; OG2; ST;  \| OG8 = CFirst (global 8)

20 ON3; ADDT; OG3; ST;
  ON3; ADDT; OG4; ST;
  ON1; ADDT; OG15; ST;
  ON2; ADDT; OG17; ST;
  ON1; ADDT; OG16; ST;

25 ON1; ADDT; OG20; ST;
  ON1; ADDT; OG13; ST;
  ON2; ADDT; OG14; ST;
  ON2; ADDT; OG7; ST;

30 *L999; END;

35 \| Note that if the method of setting globals is changed,
  \| then the system dumper must also be changed.
let SumCheck[] = valof
\[S\]
let n = 0
\[S\]
for i=0 to LASTDATA do n := AddwithEAC[r, DiscPage\[i\] \[results n\]
\[S\]
and AddwithEAC[a,b] = a+b+((a+b)<0<(a+b)) \rightarrow 1, 0
\[S\]

let Next[S] = valof
\[SN\]
if S>0 \[results (S\[\text{NEXT}[S]\]
S := \~S
\[SN\]
if S\[INBBFPTR]>S\[INBBFEND \[results ((S\[SLOWBLOCK]\[\text{NEXT}[S]\]
unless S\[INESC = NOTUSED do
\[S\]
if rv(S\[INBBFPTR])= S\[INESC \[results ((S\[SLOWBLOCK]\[\text{TRAP}[S]\]
\[S\]
let Ch = rv(S\[INBBFPTR])
S\[INBBFPTR] := S\[INBBFPTR] + 1
\[SN\]
\[SN\]

let Out[S, x] be
\[SO\]
\[test S>0 \[results (S\[OUT][S, x])
\[S\]
or
\[S\]
\[test S\[OUTBFPFTR] > S\[OUTBFEND \[results ((S\[SLOWBLOCK]\[OUT][S, x])
\[S\]
\[S\]
or
\[S\]
rv (S\[OUTBFPFTR]) := x
S\[OUTBFPFTR] := S\[OUTBFPFTR] + 1
\[test S\[OUTBFPFTR] > S\[OUTBFEND \[results ((S\[SLOWBLOCK]\[OUT][S, x])
\[S\]
or
\[S\]
unless S\[OUTESC = NOTUSED do
\[S\]
if S\[OUTESC=x \[results ((S\[SLOWBLOCK]\[OUT][S, x])
\[S\]
\[S\]
let Endof[S] = valof
  \[S\] if \(S > 0\) \[result[Endof[S]]\] \[S := \neg S\] if \(S \uparrow \text{INBBFPTR} \geq S \uparrow \text{INBFEND}\) then \[result[\text{SLOWBLOCK}\uparrow \text{ENDF}][S]\] unless \(S \uparrow \text{INESC} = \text{NUTUSED}\) do
  if \(\text{rv}(S \uparrow \text{INBBFPTR}) = S \uparrow \text{INESC}\) then \[result[\text{SLOWBLOCK}\uparrow \text{ENDF}][S]\] \[result[false]\]

let TransferIn[S, v, n] be
  for \(i = 0\) to \(n - 1\) do \(v[i] := \text{Next}[S]\)

let TransferInC[S, v, n] be
  for \(i = 0\) to \(n - 1\) do StoreCode[v+i, Next[S]]

let TransferOut[S, v, n] be
  for \(i = 0\) to \(n - 1\) do Out[S, v+i]
III: SET-UP PROGRAMS

III:1 System set-up

III:1,1 SYS SETUP

|| This section defines the routine PInterrupt, which
|| is entered when the System is first loaded from Disc.

5 manifest
§ INVALIDPAGE = 9999
SYSTEMINDEX = 4
SYSTEM = 1 || User code
STARTOFMFL = 2
10 GUSSIZE = 110
PSSIZE = 110

OUTOFRANGE = 5 || Exec reject qualifier
FIRSTNSG = 401
LASTNSG = 500
OFFSET = -(ESIZE+1)
§

static §§
20 DiscUsable = false
DeadArea = 0; DALength = 0
§§

25 let PInterrupt[] be
§§
StoreCode[DISCWRITEPERMITTED, false]

|| The order of this sequence is important
30 SetupFS[]
SetUpDiscPage[]
SetUpPMStacks[]
SetUpStreams[]
SetUpSundryItems[]
35 SetUpTimeofDayClock[]
SetDateandTime[]
SetUpRunBlock[]
SetStackBase[]
40 Discusable := CheckDiscOn[]
   test Discusable
      then Discusable := QuickValFSF[]
      or OutS[Console, 'DISC OFF: ']

45 StoreCode[REASONFORINTERRUPT, NOREASON]
StoreCode[INTERRUPTADDRESS, lv Interrupt]
ReleaseNonSystemGlobals[]

50 test Discusable
   then $ FSToCore[]
      Run[LogIn]
      StoreCode[DISCWRITEPERMITTED, true]
   $
   or $ FSF\PAGE := INVALIDPAGE
      User, CurrentIndex := SYSTEM, SystemIndex
      OutS[Console, 'Abnormal set-up*n']
   $

60 return || enters LoadGoLoop (see SetStackBase)
   $PI

65 and SetUpFS[] be
   $FS let f = FS + (FSVECSIZE + 1)
   $FSF\FBC, FSF\FWC := f, END
   FS\PBC, FS\PWC := END, END
60 FS\LB, FS\UB := f, FSLim
   FS\FPRE := FS
   f\SIZE := FSLim - f
   f\NXTB := END
75 $FS

and SetUpDiscPage[] be
   $DP let Blocksize = ESIZE + 1
80 let v = NewVec[Blocksize + 2\*PAGESIZE]
let Endv = lv v(\Blocksize + 2\*PAGESIZE)
let D = ((v-1)/\pagesize + 1)\*\pagesize
D now points to the first Hardware page in the range (v, Endv).

\[
\text{if } D - \text{Blocksize} < v \text{ do } D := D + PAGESIZE
\]
allow room for ExecBlock.

\[
\text{DeadArea} := v \\
\text{DALength} := ((D - \text{Blocksize}) - v) - 1
\]
The DeadArea is returned after the PM stacks are set-up.

\[
\text{ReturnVec}[(D+PAGESIZE), (Endv - (D+PAGESIZE))]
\]

\[
\text{let } E := D - \text{Blocksize} \quad \text{so } \text{OFFSET} := -\text{Blocksize}
\]

\[
\begin{align*}
\text{BUF} & := \text{DiscPage} \\
\text{BUFFSIZE} & := PAGESIZE \\
\text{SEG} & := \text{DATASEG} \\
\text{ENDMODE} & := \text{QUIETEND} \\
\text{SELFPTR} & := E
\end{align*}
\]

DEVICE, \text{PAGENUMB} and \text{COMPLETE} are set before use (e.g. in DiscTransfer). The other two elements of the ExecBlock are not used in this command.

\[
\text{SetUpPMStacks[] be}
\]

\[
\begin{align*}
\text{GiveUpStack} & := \text{NewVec}[GUSSIZE] \\
\text{GiveUpStack\LENGTH} & := GUSSIZE \\
\text{GiveUpStack\Size} & := GUSSIZE - 1
\end{align*}
\]

\[
\begin{align*}
\text{PrivateStack} & := \text{NewVec}[PSSIZE] \\
\text{PrivateStack\LENGTH} & := PSSIZE
\end{align*}
\]

\[
\text{ReturnVec}[\text{DeadArea, DALength}] \quad \text{claimed in SetUpDiscPage}
\]

\[
\text{SetUpSundryItems[] be}
\]

\[
\begin{align*}
\text{GiveUp} & := \text{StandardGiveUp} \\
\text{PBChain} & := \text{END} \\
\text{ClearUpChain} & := \text{ENDCUCHN}
\end{align*}
\]

\[
FSF := \text{NewVec}[FSFSIZE]
\]
$	ext{StoreCode}[\text{MFIFIRSTPAGE}, \text{STARTOFMFIL}]$

130 $	ext{SystemIndex := SYSTEMINDEX}$

$	ext{StoreCode}[\text{SUMCHINHIBITED}, \text{false}]$

1Block\$ISUC := \text{NONE}$

135 § let MaxC = $(\text{CPages} - 1) \times \text{PAGESIZE} - \text{OFFSETC}$

$\quad$ [] allows for multiplexor page

§ let MaxD = $\text{DPages} \times \text{PAGESIZE}$

StoreCode[MaxC, MaxC]

140 $\text{StoreCode}[\text{MAXD}, \text{MAXD}]$

§§

145 and SetUpTimeofDayClock[] be

§ let $E = \text{NewVec}[\text{ESIZE}]$

§ let $B = \text{NewVec}[\text{CLOCKBUFFSIZE} - 1]$

$\begin{array}{ll}
\text{EDEVICE} & := \text{CLOCK} \\
\text{EBUFFER} & := B \\
\text{EBUFFSIZE} & := \text{CLOCKBUFFSIZE} \\
\text{EBSEG} & := \text{DATASEG} \\
\text{EENDMODE} & := \text{QUIETEND} \\
\text{ECOMPLETED} & := \text{false} \\
\end{array}$

150 $\quad$ || The other three elements of the ExecBlock

$\quad$ || are not used in this device and command.

$\begin{array}{ll}
\text{ETIME} & := \text{NULL} \quad || \text{set in AskTimeofDay} \\
\text{EPERIOD} & := \text{ONEMINUTE} \\
\end{array}$

160 $\text{StoreCode}[\text{TIMEOFDAYCLOCK}, E]$

§§

165 and SetUpRunBlock[] be

§§ let $R = \text{NewVec}[\text{RSIZE}]$

170 $\quad$ || Reset FS so that RunBlock is outside FreeStore:

NewFreeStore[]

$\begin{array}{ll}
\text{FS\$FPRE} & := \text{FS} \\
\end{array}$
R\[RPRE := R
R\[PPTR := FSLim + 1 \| start of stack
R\[IBLK := IBlock
R\[CPTR := CPtr
R\[FSV := FS
180 R\[INPT := In
R\[OUTP := Output
R\[CON := Console
R\[REP := ReportStream
R\[GU := GiveUp
185 R\[GUS := GiveUpStack
R\[GUSS := GiveUpStackSize
R\[CUC := ClearUpChain

\hspace{1cm} \text{StoreCode[RBLOCK, R]}

\hspace{1cm} \text{and SetStackBase[]} be
\hspace{1cm} \text{let StackBase = FSLim + 1}
195 \hspace{1cm} \text{let Link = StackBase + 1}
\hspace{1cm} \text{rv StackBase := StackBase}
\hspace{1cm} \text{rv Link := LoadGLoop}
\hspace{1cm} \text{§S}

\hspace{1cm} \text{and CheckDiscOn[]} = valof
\hspace{1cm} \text{let E = DiscPage + OFFSET}
\hspace{1cm} \text{E\[DEVICE := DISCREAD}
\hspace{1cm} \text{E\[PAGENUMB := INVALIDPAGE}
\hspace{1cm} \text{Exec[TRANSFER, Rj, lv E\[PAGENUMB, E\[PAGENUMB, E]}
\hspace{1cm} \text{Rj: || Exec will reject the command, either}
\hspace{1cm} \text{|| because the Disc-Controller is off-line,}
\hspace{1cm} \text{|| or because the page is out of range,}
\hspace{1cm} \text{|| The reason is placed in E\[PAGENUMB.}
\hspace{1cm} \text{resultis (E\[PAGENUMB = OUTFRANGE)}
\hspace{1cm} \text{§DO}

\hspace{1cm} \text{and ReleaseNonSystemGlobals[]} be
\hspace{1cm} \text{global § GlobalZero : 0 §}
\hspace{1cm} \text{let GlobVec = lv GlobalZero}
\hspace{1cm} \text{for g = FIRSTNSG to LASTNSG do GlobVec\[g := NULL}
\hspace{1cm} \text{§RG}
and FSFtoCore[] be
\[\text{let } H = \text{FindHeading[LookUp["FSF", "SYS", SystemIndex]]}\]
\[\text{DisttoCore[FSF, H\{LASTPAGE\}]}\]
\[\text{FSF\{SCAN \:=\} FIRSTENTRY}\]
225 \[\text{FSF\{PAGE \:=\} H\{LASTPAGE\}}\]
\[\text{ReturnVec[H, (H\{O \^\{LMASK\})]}\]

230 and LogIn[] be
\[\text{LoadSystemFile["LogIn"]}\]
\[\text{Prog[]}\]

***
This section defines the routine `SetDateandTime`, which is called by `SYSSETUP`.

manifest

JAN = 1; FEB = 2; MAR = 3; APR = 4; MAY = 5; JUN = 6
JUL = 7; AUG = 8; SEP = 9; OCT = 10; NOV = 11; DEC = 12
IMPOSS = -1

let SetDateandTime[] be

\$SDT\$

let t = TimeofDay[]
\$SDT\$

let d = Data[]
StoreCode[DATE, d]
reset[Console]
\$SDT\$

and Date[] = valof

\$D\$

outS[Console, 'Date *q ']
\$D\$

let Day = NextN[Console]
\$D\$

let Month = NextN[Console]
\$D\$

let Year = NextN[Console] rem 1900

unless 1 ≤ Year < 99
\$D\$

wrong['Year']; resultant Date[]\$D\$

unless JAN ≤ Month ≤ DEC
\$D\$

wrong['Month']; resultant Date[]\$D\$

\$D\$

let DaysinMonth, Fudge = 0, 0

switch Month into

\$S\$

case JAN: DaysinMonth := 31
Fudge := (Leap[Year]→ 6, 0); endcase
\$S\$

case FEB: DaysinMonth := (Leap[Year]→ 29, 28)
Fudge := (Leap[Year]→ 2, 3); endcase
\$S\$

case MAR: DaysinMonth, Fudge := 31, 3; endcase
\$S\$

case APR: DaysinMonth, Fudge := 30, 6; endcase
\$S\$

case MAY: DaysinMonth, Fudge := 31, 1; endcase
\$S\$

case JUN: DaysinMonth, Fudge := 30, 4; endcase
\$S\$

case JUL: DaysinMonth, Fudge := 31, 6; endcase
case AUG: DaysinMonth, Fudge := 31, 2; endcase
case SEP: DaysinMonth, Fudge := 30, 5; endcase
case OCT: DaysinMonth, Fudge := 31, 0; endcase
case NOV: DaysinMonth, Fudge := 30, 3; endcase
case DEC: DaysinMonth, Fudge := 31, 5; endcase

unless 1 < Day < DaysinMonth
do $ Wrong["Day"]; resultis Date[] $

$ let DayofWeek = (Year + Year/4 + Day + Fudge)rem 7
|| Fudge is a correction because of the Month

$ let Ch = NextLetter[Console]
if Ch="S" v Ch="T" do

$ let x = NextLetter[Console]
unless x="U" do Ch := x

$ unless DayofWeek = (Ch="S" -> 0, Ch="M" -> 1, Ch="T" -> 2,
Ch="W" -> 3, Ch="H" -> 4, Ch="P" -> 5,
Ch="A" -> 6, IMPOSS)
do $ Wrong["Day of week"]; resultis Date[] $

resultis (Day lshift 11)+(Month lshift 7)+ Year

and Wrong[String] be
Outs[Console, "*s wrong *n", String]

and Leap[Year] := (Year rem 4 = 0)

80 and NextLetter[s] = valof
$NL let Ch = 0
Ch := Next[s] repeatuntil ("A"<Ch<"Z")v("a"<Ch<"z")
resultis Ch ^ CASEMASK || translate into capitals

85

****
manifest $\text{MAXPAGE = 3000}$

static $\gamma$

Result = true

let QuickVa1F[] = valof

let $f = \text{LookUp["FSF", "SYS", SystemIndex]}$

if $f = \text{NULL}$ do

ResultS["FreeStoreFile not found"]

resultis false $\gamma$

let $h = \text{FindHeading}[f]$

let $\text{Serial, Page, PreviousPage} = 1, "\text{FIRSTPAGE, ENDBODY}"

unless $1 < \text{Page} < \text{MAXPAGE}$ do

ResultS["FSF: forward link $\text{Page, PreviousPage}]$

resultis false $\gamma$

DisctoCore[DiscPage, Page]

unless DiscPage$\text{SERIAL} = \text{Serial}$ do

ReportMessage["FSF: page *N wrong serial", Page]

unless DiscPage$\text{THISFILE} = f$ do

ReportMessage["FSF: page *N wrong file no.", Page]

unless DiscPage$\text{BACKLINK} = \text{PreviousPage}$ do

ReportMessage["FSF: page *N backlink wrong", Page]

let $n = 0$

for $i = \text{FIRSTENTRY to LASTENTRY}$ do

unless $1 < \text{DiscPage}i < \text{MAXPAGE}$ do

if DiscPage$\text{i + CLAIMED}$ do $n := n + 1$

if $n > 0$ do ReportMessage["FSF: page *N, *N *S*s outside page limits", Page, n, (n > 1 $\rightarrow$ 'entries', 'entry')]

if DiscPage$\text{NEXTPAGE} = \text{ENDBODY}$ break

Serial, PreviousPage$= \text{Serial} + 1$, Page

Page $= \text{DiscPage} \text{NEXTPAGE}$

repeat
unless Serial = h\$NUMPAGES do
  ReportMessage["FSF: no. of pages wrong"]
unless Page = h\$LASTPAGE do
  ReportMessage["FSF: last page number wrong"]
ReturnVec[h, h\$O^LMASK]
result is Result

and ReportMessage[String, x, y, z] be
\% Reports[String, x, y, z]
Result := false \%
III:2. Set up streams

III:2.1 SETUPSTR

```
manifest
§ VECSIZE = 15   || 16 possible values
     PARBITS = 864626   || for 4 bits.
§

let SetUpStreams[] be
§ SUS
   SetUpParityTable[]
   SetUpExecConsole[]
   Console := ExecConsole
   Output := ExecConsole
   ReportStream := ExecConsole
   SetUpReader[]
   In := DiadRead
§ SUS

and SetUpParityTable[] be
§ P
   let P = NewVec[VECSIZE]
   let PBits = PARBITS
   for i = 0 to VECSIZE do
§ P\i := PBits ^ 1
   PBits := PBits rshift 1
§
   Parity := P
§ P

****
```
III:2.2 SETUPTT

let SetupExecConsole[] be
\[\begin{align*}
\text{Teletype} & := \text{TT[]} \\
\text{ExecConsole} & := \text{IntcodeToAndFromTeletype[Teletype]} \\
\text{ExecConsole\textbackslash CLOSE} & := \text{ExecConsole\textbackslash RESET}
\end{align*}\]

end $\text{TT[]} = \text{valof}$ $\text{S}$

let $S = \text{NewVec[TELESIZE]}$
\[\begin{align*}
\text{S\textbackslash NEXT} & := \text{NextTT} \\
\text{S\textbackslash OUT} & := \text{OutTT} \\
\text{S\textbackslash CLOSE} & := \text{ResetTT} \\
\text{S\textbackslash STR} & := \text{NULL} \\
\text{S\textbackslash ENDF} & := \text{EndOfTT} \\
\text{S\textbackslash RESET} & := \text{ResetTT} \\
\text{S\textbackslash SOURCE} & := S \\
\text{S\textbackslash STATE} & := \text{StateTT} \\
\text{S\textbackslash RESETSTATE} & := \text{ResetStateTT}
\end{align*}\]

let $R = \text{lv S\textbackslash READBLOCK}$
\[\begin{align*}
\text{R\textbackslash DEVICE} & := \text{TIREAD} \quad \| \quad \text{without echo} \\
\text{R\textbackslash BUFFER} & := \text{lv S\textbackslash INBUFF} \\
\text{R\textbackslashBUFFSIZE} & := 2 \quad \| \quad \text{one word is required by Exec} \\
\text{R\textbackslash STOPCH} & := \text{NOBYTE} \\
\text{R\textbackslash SEG} & := \text{DATASEG} \\
\text{R\textbackslashENDMODE} & := \text{QUIETEND} \\
\text{R\textbackslash COMPLETED} & := \text{false} \\
\text{R\textbackslashSELFPTR} & := R
\end{align*}\]

|| The other two elements of the ExecBlock
|| are not used in this command.
§ let \( w = lv \) WRITEBLOCK

\[ W^\downarrow \text{DEVICE} := \text{TTWRITE} \]

\[ W^\downarrow \text{BUFFER} := lv \ 6^\downarrow \text{OUTBUFF} \]

\[ W^\downarrow \text{BUFFSIZE} := 1 \]

\[ W^\downarrow \text{STOPCH} := \text{BYTE} \]

\[ W^\downarrow \text{SEG} := \text{DATASEG} \]

\[ W^\downarrow \text{ENDMODE} := \text{QUIETEND} \]

\[ W^\downarrow \text{COMPLETED} := \text{false} \]

\[ W^\downarrow \text{SELFPTR} := w \]

\[ \text{resultis} \ S \]

\[ S^\downarrow \text{LINDERFPTR}, S^\downarrow \text{LINDERFLAST} := \text{EMPTY}, \text{EMPTY} \]

\[ \text{resultis} \ S \]
manifest $ \text{RPINGS} = 3 $

let SetUpReader[] be $\text{ReaderDev} := \text{READERLEFTTORIGHT}$
$\text{BytesfromPT} := \text{BFPT}[]$
$\text{DiadRead} := \text{WordsfromDiads[BytesfromPT]}$

let Diad = (~\text{DiadRead})$\uparrow$\text{SLOWBLOCK}$
$\text{Diad} \uparrow \text{CLOSE} := \text{Diad} \uparrow \text{RESET}$

and $\text{BFPT}[] = \text{valof}$
$\text{FB}$

let $\text{FB} = \text{NewVec[FB\text{SIZE}]}$

let $\text{S} = \text{NewVec[BFPT\text{SIZE}]}$

let $\text{E} = \text{lv S} \uparrow \text{EXEBCBLOCK}$

$\text{FB} \uparrow \text{SLOWBLOCK} := \text{S}$
$\text{FB} \uparrow \text{INBFPPTR}, \text{FB} \uparrow \text{INBFEND} := \text{EMPTY, EMPTY}$

$\text{FB} \uparrow \text{OUTBFPPTR}, \text{FB} \uparrow \text{OUTBFEND} := \text{EMPTY, EMPTY}$
$\text{FB} \uparrow \text{INESC}, \text{FB} \uparrow \text{OUTESC} := \text{NOTUSED, NOTUSED}$

$\text{S} \uparrow \text{NEXT} \text{is set by the call of ResetBFPT below}$

$\text{S} \uparrow \text{OUT} := \text{StreamError}$

$\text{S} \uparrow \text{CLOSE} := \text{ResetBFPT}$

$\text{S} \uparrow \text{ENDIF} := \text{EndofBFPT}$

$\text{S} \uparrow \text{RESET} := \text{ResetBFPT}$

$\text{S} \uparrow \text{SOURCE} := \text{S}$

$\text{S} \uparrow \text{STATE} := \text{StateBFPT}$

$\text{S} \uparrow \text{RESETSTATE} := \text{NullProgram}$

$\text{S} \uparrow \text{RPINGS} := \text{RPINGS}$

let $\text{Buff1}, \text{Buff2} = \text{lv S} \downarrow \text{BUFF1}, \text{lv S} \downarrow \text{BUFF2}$

$\text{Buff1} \downarrow (-1) := \text{Buff2}$

$\text{Buff2} \downarrow (-1) := \text{Buff1}$
E\$DEVICE := ReaderDev
E\$BUFFER := Buff1
E\$BUFFSIZE := 0 || set in InitiateTransfer
E\$STOPCH := NOTBYTE
E\$SEG := DATASEG
E\$ENDMODE := QUIETEND
E\$COMPLETED := false
E\$SELFPTR := E
   || The other two elements of the ExecBlock
   || are not used in this command.

55 ResetBFPT[FB]  || to set S\$NEXT

result is ~FB

60 ****
IV: DECLARATIONS

IV: 1 DECLARATIONS

get 'GLOBALS'
get 'PRIVATE GLOBALS'
get 'CONSTANTS'

5

****
IV:1.1 GLOBALS

```
global §G

BytesfromP: 387

5 Close: 18; Console: 27; Copy: 36; CPtr: 9
   CurrentIndex: 300
   DeleteBody: 326; DefaultProg: 398; DiadRead: 338

10 EndGiveUp: 373; Endof: 20; EntriesfromFile: 320
   EqS: 352; ExecConsole: 383
   FetchCode: 2; FindHeading: 318; Finish: 395

15 GiveUp: 29; GiveUpStack: 363; GiveUpStackSize: 353
   In: 25; InfromFile: 321
   Load: 397; LoadFile: 378; LoadSystemFile: 305

20 LookUp: 317
   MaxVecSize: 37
   NewVec: 60; NewWord: 62; Next: 17; NextN: 358
25 NextO: 357; NullProgram: 359

   Out: 16; OutAddr: 347; OutByte: 346
   OutDateandTime: 348; OutN: 379; OutO: 380
   Output: 26; OutS: 381; OutToFile: 322

30 Prog: 1; PutBack: 391

   ReportCallTrace: 375; ReportFreeStoreState: 374
   Reports: 30; ReportStream: 28; Reset: 68; ResetState: 349
35 ReturnVec: 61; ReturnWord: 63; Run: 399

   StandardGiveUp: 394; StandardPM: 393; Start: 1
   State: 350; StoreCode: 3; StreamError: 69
   SystemIndex: 303

40 TimeOfDay: 364; TransferIn: 13; TransferInC: 14
   TransferOut: 7; TryAgain: 355
```
Unload: 396; User: 310;

Wait: 372; Words from Diads: 340; Write: 343
Write Addr: 345; Write Byte: 344; Write N: 45; Write Q: 38

Write S: 44

****

&c
IV:1.2 PRIVATE GLOBALS

_**global**_ $G$

AddressZero: 371  
CFirst: 8; Charge: 306; ClearUpChain: 332; CoreToDisc: 334  
DiscPage: 335; DiscToCore: 333; Dump: 66  
Exec: 4  
ForcedGiveUp: 32; FS: 24; FSF: 329  
IBlock: 23; InitiateTransfer: 10  
IntcodeToAndFromTeletype: 79; Interrupt: 351  
LLoop: 64; LoadGoLoop: 370; LookUpInMFL: 319  
NewDiscBlock: 330; NewFreeStore: 390  
OSReport: 33; OSReportN: 34  
Parity: 67; PBChain: 337; PPtr: -3; PrivateStack: 356  
ReaderDev: 382; RestoreFreeStore: 380  
ReturnDiscBlock: 331  
SetGlobals: 21; SetLabels: 11; SumCheck: 15  
Teletype: 376; TerminateRun: 392; TryDiadAgain: 336  
Update: 312; UpdateHead: 311

20

```
    || temporary globals:
     
CPages: 416  
DPages: 417  
EndOfBFPT: 401; EndOfTT: 402  
FSLim: 412  
NextTT: 403  
OutTT: 404  
PInterrupt: 411  
QuickValFSF: 414  
ResetBFPT: 405; ResetTT: 406; ResetStateTT: 407  
SetDateAndTime: 413; SetUpExecConsole: 409  
    SetUpReader: 410; SetUpStreams: 415  
    StateBFPT: 418; StateTT: 408
```

35

_**§G**_

****
IV:1.3 CONSTANTS

manifest || General
§ NULL = 0
§ UNDEFINED = 0
§

manifest || Machine constants
§ PAGE SIZE = 256
§ OFFSET = 30
§ LMASK = 8377

manifest || Exec commands
§ CANCEL = 108
§ LOOKATRDR = 248
§ READABS = 240
§ TRANSFER = 28
§

manifest || Exec block for TRANSFER command
§ ESIZE = 9

DEVICE = 0
BUFFER = 1
BUFF SIZE = 2

PAGENUMB = 3 || for disc transfers
STOPCH = 3 || for other transfers
SEG = 4

| ENDMODE = 6 || element 5 is never used
| COMPLETED = 7 || used if endmode is QUIETEND
| LFPTR = 8
| INTREASON = 9 || used if endmode is INTERRUPT

manifest || Standard contents of Exec block elements
§ MODIFY = -1 || stopch
§ CODESEG = 1 || seg
§ DATASEG = 0 || seg
§ INTERRUPT = 3 || endmode
§ QUIETEND = 2 || endmode
§
manifest || Device numbers

45 $TTWRITE = 1
TTREAD = 2 || without echo
READERLEFTRIGHT = 4
RDRDEVSUM = 7
DISCREAD = 26
50 DISCWRITE = 27
CLOCK = 29

manifest || Code segment addresses

55 $INTERRUPTINHIBITED = -29
REASONFORINTERRUPT = -28
MAXD = -8
MAXC = -7
60 SUMCHINHIBITED = -6
DISCWRITEPERMITTED = -5
MFLFIRSTPAGE = -4
DATE = -3
RBLOCK = -2
65 TIMEOFDAYCLOCK = -1

interruptaddress = 3

manifest || Reasons for interrupt

$NOREASON = 0
EXECCONXON = 1
POWERON = 2

manifest || Clock

$CLOCKHUFFSIZE = 2

80 TIME = 0
PERIOD = 1
ONEMINUTE = 120
85 MINSPERDAY = 60x24
manifest \| Run block

\[ \text{manifest} \| \text{Run block} \]

\[ \text{RPRE} = 0 \| \text{Predecessor of block} \]
\[ \text{PPTR} = 1 \| \text{Procedure pointer} \]
\[ \text{IBLK} = 2 \| \text{Information block} \]

\[ \text{manifest} \| \text{Information block} \]
\[ \text{ISIZE} = 6 \]
\[ \text{CP} = 0 \| \text{Code pointer} \]
\[ \text{CL} = 1 \| \text{Code length} \]
\[ \text{DP} = 2 \| \text{Data pointer} \]
\[ \text{DL} = 3 \| \text{Data length} \]

\[ \text{manifest} \| \text{Free store} \]
\[ \text{FSVECSIZE} = 6 \]
\[ \text{FBC} = 0 \| \text{Free block chain} \]
\[ \text{FWC} = 1 \| \text{Free word chain} \]
\[ \text{PBC} = 2 \| \text{Pending block chain} \]
\[ \text{FWC} = 3 \| \text{Pending word chain} \]
\[ \text{LB} = 4 \| \text{Lower bound of current area} \]
\[ \text{UB} = 5 \| \text{Upper bound of current area} \]
\[ \text{FPRE} = 6 \| \text{Predecessor of vector} \]
\[ \text{SIZE} = 0 \]
\[ \text{NXTB} = 1 \]

|| SIZE and NXTB cannot be interchanged without
\| alteration to the program. (See 'FREESTORE').

\[ \text{END} = 0 \]
\[ \text{NUSTORE} = -1 \]

\[ \text{manifest} \quad || \quad \text{ClearUp Chain} \]
\[ \text{CSUC} = 0 \quad || \quad \text{Successor of block} \]
\[ \text{CPRE} = 1 \quad || \quad \text{Predecessor of block} \]
\[ \text{ROUTINE} = 2 \quad || \quad \text{ClearingUp routine} \]

\[ \text{ENDCUCHN} = 877774 \]

\[ \text{manifest} \quad || \quad \text{Stream vector} \]
\[ \text{NEXT} = 0 \quad || \quad \text{except source or bilateral streams} \]
\[ \text{OUT} = 1 \]
\[ \text{CLOSE} = 2 \]

\[ \text{STR} = 3 \quad || \quad \text{only source streams} \]
\[ \text{ENDOF} = 4 \]
\[ \text{RESET} = 5 \]
\[ \text{SOURCE} = 6 \]
\[ \text{STATE} = 7 \quad || \quad \text{only source streams} \]

\[ \text{RESETSTATE} = 8 \quad || \quad \text{only source streams} \]

\[ \text{NOTHINGTYPED} = -1 \]

\[ \text{manifest} \quad || \quad \text{Stream elements used by InitiateTransfer} \]
\[ \text{PINGS} = 3 \]
\[ \text{EXECBLOCK} = 9 \]
manifest || Fast stream block
180 \$ FBSIZE = 6

  SLOWBLOCK = 0 || pointer to stream vector
  INBFPPTR = 1
  INBFEND = 2
185
  INESC = 3
  OUTBFPPTR = 4
  OUTBFEND = 5
  OUTESC = 6

190
  EMPTY = 0
  NOTUSED = 0

\$

manifest || Teletype stream
195 \$ TELESIZE = 105

  READBLOCK = 9
  WRITEBLOCK = 19
  INBUFF = 29
  OUTBUFF = 31
  LINEBFPPTR = 32
  LINEBFFLAST = 33
200
  LINEBUFFER = 34

\$

manifest || Size of reader buffer
210 \$ READBFFSIZE = 30 \$

manifest || Reader stream
215 \$ BFPTSIZE = 22 + 2\times READBFFSIZE

215
  BUFF1 = 20
  BFF1END = 50
  BUFF2 = 52
  BFF2END = 82
220 \$
225 manifest || PutBack vector
§ PBSIZE = 8

PBEXT = 0
PBCLOSE = 1
PBSTR = 2
PBENDOF = 3
PBRESET = 4
OB = 5 || Object put back
STREAM = 6
PTR = 7
PBPRE = 8 || Predecessor

§

manifest || Internal Character Code
240 § ULMASK = 8177
CASEMASK = 8137

BELL = 7
DELETE = 8177
DIV = 830
GRAVE = 8140
LEFTARROW = 8137
PRIME = 824
RETURN = 815
RUNOUT = 3

§

manifest || Teletype Character Code
255 § PARITYMASK = 8177
PARITYBIT = 8200

ANDERST = 846
BELLt = 8207
ESCAPET = 833
LINEFEEDt = 812
NEWLINEt = 812
PRIMEt = 847
QUERYt = 877
RETURNt = 8215

RUBOUTT = 8377
RUNOUTT = 0
SHARPT = 8243
SLASHt = 857
manifest || stack element
LENGTH = 1

manifest || Filing System

SERIAL = 0
THISFILE = 1
NXTPAGE = 2
FIRSTDATA = 3
LASTDATA = 254
ENDOFDATAPTR = 254 || on last page of streamable files
SUMCH = 255

DATASIZE = 252
FREE = 0
ENDBODY = 877776

manifest || Headings

FIRSTPAGE = 1
NUMBPGES = 2
LASTPAGE = 3
TYPE = 4
OWNER = 5
CREATED = 6 || Data created
DLW = 7 || Date Last Written
DLR = 8 || Date Last Read
PERM = 9 || Permission
MFLNUMB = 10 || File number
SAME = -9
NEVER = 0
NULLBODY = -1
manifest || Index Entries

ENTRYEND = 0
STATUS = 1
FILE = 3
LINKING = 3
N1 = 1
N2 = 2
N3 = 3
N4 = 4
N5 = 5
N6 = 6

manifest || Some file types

DELETED = 0
INDEX = 813
STREAMABLE = 810

manifest || File permissions

UNRESTRICTED = 0
OWNERONLY = 1
RESTRICTED = 2

manifest || MasterFile List

ENTRYSIZE = 1

HPAGE = 0
HWORD = 1

METAENTRY = 0
ENTRIESPERPAGE = 126
360
\[
\text{manifest } \| \text{ Free Store File}
\]
\[
\text{FSIZE} = 257
\]
365
\[
\begin{align*}
\text{BACKLINK} &= 3 \\
\text{FIRSTENTRY} &= 4 \\
\text{LASTENTRY} &= 254 \\
\text{PAGEEND} &= 255 \\
\text{SCAN} &= 256 \\
\text{PAGE} &= 257
\end{align*}
\]
370
\[
\text{CLAIMED} = 5000
\]
375
\[
\text{manifest } \| \text{ InfromFile}
\]
\[
\text{FSIZE} = 261
\]
\[
\begin{align*}
\text{FILENAMEB} &= 3 \quad \| \text{ also used by OuttoFile} \\
\text{IFBUFFER} &= 6
\end{align*}
\]
380
\[
\begin{align*}
\text{ENDOFSTREAMCH} &= -1 \\
\text{INFILESIZE} &= 269 \quad \| (\text{FSIZE}+1)+(\text{FSIZE}+1)
\end{align*}
\]
385
\[
\text{manifest } \| \text{ OuttoFile}
\]
\[
\text{FSIZE} = 269
\]
\[
\begin{align*}
\| \text{ FILENUMB} &= 3 \quad \text{(defined above)}
\end{align*}
\]
390
\[
\begin{align*}
\text{CUCHAIN} &= 6 \\
\text{FILEOWNER} &= 9 \\
\text{OLDBODY} &= 10 \\
\text{OLDSIZE} &= 11 \\
\text{OFPAGE} &= 12 \\
\text{NEWBODY} &= 13 \\
\text{OFBUFFER} &= 14
\end{align*}
\]
400
\[****\]
V: SEGMENTATION OF THE SYSTEM FOR Compilation

OS/1

§Pub1

get 'DECLARATIONS'
get 'IG LOOP'
get 'RUN'
get 'LOAD'
get 'SETLAB'

§Pub1

OS/2

§Pub2

get 'DECLARATIONS'
get 'GIVEUP'
get 'PM'
get 'MPM'
get 'INTERRUPT'

§Pub2

OS/3

§Pub3

get 'DECLARATIONS'
get 'FREESTORE'
get 'PUTBACK'

§Pub3
$\textit{OS/4}$

\begin{verbatim}
Pub4
get 'DECLARATIONS'
get 'STRPRIMITIVES'
get 'OUTS'
get 'OPRTS'
get 'NEXTN'
\end{verbatim}

$\textit{OS/5}$

\begin{verbatim}
Pub5
get 'DECLARATIONS'
get 'TELETYPEx'
get 'INTTI'
get 'XFER'
get 'READER'
get 'DIADS'
\end{verbatim}

$\textit{OS/6}$

\begin{verbatim}
Pub6
get 'DECLARATIONS'
get 'CLOCK'
get 'MISC'
\end{verbatim}
§Pub7

get 'DECLARATIONS'
get 'DISCXRER'
get 'DISCFS'
get 'DISCIN'
get 'DISCOUT'

§Pub7

05/8

§Pub8

get 'DECLARATIONS'
get 'CHARGE'
get 'UPDATE'
get 'UPDATEHEAD'
get 'FINDHEADER'
get 'LOOKUP'
get 'LOADFILE'
OS/SU

get "DECLARATIONS"
get "SYSETUP"
get "SETDANDT"
get "QUICKVAL"

OS/SUS

get "DECLARATIONS"
get "SETUPSTR"
get "SETUP1"
get "SETUPRDR"

OS/SUS
VIII: SOME LIBRARY FILES

VIII:1 logIn

get 'DECLARATIONS'

manifest ||| elements of Usercode entry :
§ ENTSIZE = 0
5
USERNAME = 1
NAME1 = 2
NAME2 = 3
USERNUMB = 4
§

manifest
§ MAXSTRINGSIZE = 127 §

15 let Prog[] be
§P User, CurrentIndex := NULL, NULL
Outs[Console, 'Name *q ']
§ let Name = String[Console]
let v = LookUpUser[Name]
20 ReturnVec[Name, MAXSTRINGSIZE]
if v = NULL do
§ Reports['No such user']
    return
25 §
User := v[USERNUMB]
CurrentIndex := LookUp[v[NAME1], v[NAME2], SystemIndex]
ReturnVec[v, v[ENTSIZE]]
§P

30 and LookUpUser[Name] = valof
§L let UserCodes = LookUp['UserCodes', 'SYS', SystemIndex]
let S = EntriesfromFile[UserCodes]
35 until Endof[S] do
§u let v = Next[S]
if EqS[Name, v[USERNAME]] do
§ Close[S]
    resultis v §
40 ReturnVec[v, v[ENTSIZE]]
§u
Close[S]  
resultis NULL  

§L  
and String[S] = valof  
§S  
let v = NewVec[MAXSTRINGSIZE]  
let n, i = 0, 0  
let Ch = Next[S]  

v\$0 := 0  
until Ch='n' do  
§u  
n := n + 1  

test n rem 2 = 1  
then v\$i := (v\$i) \lor Ch  
or § i := i + 1  

v\$i := Ch lshift 8 §  
Ch := Next[S]  
§u  
v\$0 := (n lshift 8) \lor v\$0  
§S  
resultis v  

****
VIII:2 MakeNewFile

get 'DECLARATIONS'

```plaintext
VIIH2 MakeNewFi1e

~ Declared in this segment
5 $ DeleteFile = 327
    MakeNewFile = 328
    UpdatePermission = 313

$ Defined in 'DiscRts'
10 $ AddVectoFil.e = 307
    NewLocation = 308

manifest
15 $ ASIZE = 1
    APAGE = 0
    AWORD = 1
20 UNlilADED = 0
    TITLE = 11

manifest
25 $ MAXHDSIZE = 127 + TITLE

static
20 HeadingFile = UNDEFINED
    MFLFile = UNDEFINED
30 StartMFLScan = UNDEFINED

35 let MakeNewFile[Title, Type] = valof

S1 if StartMFLScan = UNDEFINED do
  $i HeadingFile := LookUp['Headings', 'SY$3, SystemIndex]
    MFLFile := LookUp['MFL', 'SY$3, SystemIndex]
    StartMFLScan := FetchCode[MFLFIRSTPAGE]
LoadDiscRtsifNec[]
§ let HdAddr = NewLocation[HeadingFile]
  let File = NewMFILEntry[HdAddr\PAGE, HdAddr\AWORD]
CreateNewHead[Title, Type, File, User, HdAddr]
ReturnVec[HdAddr, ASIZE]
resultis File
§M
and LoadDiscRtsifNec[] be
50  if NewLocation=UNLOADED do LoadSystemFile["DiscRts"]
and NewMFILEntry[Page, Word] = valof
§N let p = vec (_PAGESIZE-1)
$M DisctoCore[p, StartMFLScan]
55  § let i = FIRSTDATA
  let f = 1 + ENTRIESPERPAGEX((p+SERIAL)-1)
  until i > LASTDATA do
§u if p+i = NOTENTRY do
  §1 p+i\PAGE := Page
  p+i\WORD := Word
  CoreoDisc[p, StartMFLScan]
  resultis f
  i := i + (ENTRYSIZE+1)
  f := f+1
§u if p\NXTPAGE = ENDBODY do
  §2 p\NXTPAGE := NewMFILEPage[(p+SERIAL)+1]
  CoreoDisc[p, StartMFLScan]
70  §2 StartMFLScan := p\NXTPAGE
$M repeat
§N
75 and NewMFILEPage[Serial] = valof
§P let Page = NewDiscBlock[]
  DiscPage\SERIAL := Serial
  DiscPage\THISFILE := MFLFfile
  DiscPage\NXTPAGE := ENDBODY
80  for i=FIRSTDATA to LASTDATA do DiscPage\i := NOTENTRY
  CoreoDisc[DiscPage, Page]
  UpdateHead[MFLFile, SAME, Serial, Page,
              SAME, FetchCode[DATE], SAME]
resultis Page
§P
and CreateNewHead[Title, Type, File, Owner, HdAddr] be

[\text{let } v = \text{vec MAXHDSIZE}]
[\text{let TitleLength} = (\text{Title} \downarrow 0 \text{ rshift } 8)/2]

\[v\downarrow 0 := (\text{TITLE } lshift 8)v(\text{TITLE} + \text{TitleLength})\]
\[v\downarrow \text{FIRSTPAGE} := \text{NULLBODY}\]
\[v\downarrow \text{NUMPAGES} := 0\]
\[v\downarrow \text{LASTPAGE} := \text{NULLBODY}\]

\[v\downarrow \text{TYPE} := \text{Type}\]
\[v\downarrow \text{OWNER} := \text{Owner}\]
\[v\downarrow \text{CREATED} := \text{FetchCode[DATE]}\]
\[v\downarrow \text{DLW} := \text{NEVER}\]
\[v\downarrow \text{DLR} := \text{NEVER}\]

\[v\downarrow \text{PERM} := \text{OWNRONLY}\]
\[v\downarrow \text{FILENUM} := \text{File}\]
\[\text{Copy}[\text{Title}, lv v\downarrow \text{TITLE}, \text{TitleLength}+1]\]

\[\text{AddVecToFile}[\text{HeadingFile, HdAddr, } v, (\text{TITLE}+\text{TitleLength})]\]

\[\text{let UpdatePermission}[f, \text{Perm}] \text{ be}\]
[\text{CheckLegality}[f, \text{update permission}]]
\[\text{Update}[f, \text{PERM}, \text{Perm}]\]

\[\text{and DeleteFile}[f] \text{ be}\]
[\text{CheckLegality}[f, \text{delete file}]]
\[\text{DeleteBody}[f]\]
\[\text{Update}[f, \text{TYPE, DELETED}]\]

\[\text{and CheckLegality}[f, \text{Description}] \text{ be}\]
[\text{let } h = \text{FindHeading}[f]]
[\text{if } (h=\text{NULL})\downarrow (\text{User} + h\downarrow \text{OWNER}) \text{ do}]
[\text{ReportS["Illegal attempt to *S", Description]}]
[\text{GiveUp}[f]]
\[\text{ReturnVec}[h, h\downarrow 0 \wedge \text{LMASK}]\]

****
VIII:3 Index Ops

get 'DECLARATIONS'

---

**global** \|\| defined in this segment

5  §  DeleteEntry : 314  
    Enter : 316  
    Link : 315  


**global** \|\| defined in 'DiscRts'

§  AddVectorFile : 307  
    NewLocation : 308

§

**manifest**

§  ASIZE = 1  
    UNLOADED = 0

§

**static** \|\| used by Link

§  p=0; q=0; i=0  

**let** Enter[f, Name1, Name2, Ind] be

30  \$  if f<0 do  
    §  Reports[Enter[*N, *S, *S, *N], f, Name1, Name2, Ind]  
    GiveUp[f] §  
    CheckPermission[Ind]

§  if g=NULL do DeleteEntry[Name1, Name2, Ind]

§  AddEntry[f, Name1, Name2, Ind]  

§§
AddEntry[f, Name1, Name2, Ind] be

\[
\text{LoadDiskSurfNec[]}
\]

\[
\text{let } L1, L2 = \text{Size[Name1]}, \text{Size[Name2]}
\]

\[
\text{let } n = 3 + L1 + L2
\]

\[
\text{let } v = \text{NewVec[n]}
\]

\[
v\downarrow \text{ENTRYEND} := n
\]

\[
v\downarrow N1 := 4
\]

\[
v\downarrow N2 := 4 + L1
\]

\[
v\downarrow \text{FILE} := f
\]

\[
\text{Copy[Name1, v+v\downarrow N1, L1]}
\]

\[
\text{Copy[Name2, v+v\downarrow N2, L2]}
\]

\[
\text{let Addr = NewLocation[Ind]}
\]

\[
\text{AddVectorFile[Ind, Addr, v, n]}
\]

\[
\text{ReturnVec[Addr, ASIZE]}
\]

\[
\text{ReturnVec[v, n]}
\]

Link[Name1, Name2, Na, Nb, Nc, Nd, Ind] be

\[
\text{CheckPermission[Ind]}
\]

\[
\text{CheckLinkDoesntLoop[Name1, Name2, Na, Nb, Nc, Nd, Ind]}
\]

\[
\text{let } g = \text{LookUp[Name1, Name2, Ind]}
\]

\[
\text{if } g \neq \text{NULL do DeleteEntry[Name1, Name2, Ind]}
\]

\[
\text{AddLinkedEntry[Name1, Name2, Na, Nb, Nc, Nd, Ind]}
\]

\[
\text{CheckLinkDoesntLoop[Name1, Name2, Na, Nb, Nc, Nd, Ind] be}
\]

\[
\text{Check[Name1, Name2, Ind]} = \text{valof } || \text{ version of LookUp}
\]

\[
\text{if } Eqs[Name1, p] \land Eqs[Name2, q] \land (\text{Ind}=i) \text{ do}
\]

\[
\text{ReportS[Attempted loop in Link']}
\]

\[
\text{GiveUp[Ind]}
\]
if \text{Ind} = \text{NULL} \text{ result is NULL}

\text{let } s = \text{EntriesFromFile}[\text{Ind}]

\text{until EndOf}[s] \text{ do}
\text{let } v = \text{Next}[s]
\text{if } v\downarrow \text{STATUS} \neq \text{DELETED} \text{ do}
\text{if } \text{EqS}[\text{Name1}, v\downarrow v\downarrow N1] \land \text{EqS}[\text{Name2}, v\downarrow v\downarrow N2] \text{ do}
\text{let NotLinked} = (v\downarrow \text{LINKING} > 0)
\text{let } f = \text{NotLinked} \rightarrow v\downarrow \text{FILE},
\text{Check}[v\downarrow v\downarrow N3, v\downarrow v\downarrow N4, v\downarrow v\downarrow N5, v\downarrow v\downarrow N6, \text{SystemIndex}]]
\text{ReturnVec}[v, v\downarrow \text{ENTRYEND}]
\text{result is } f
\text{ReturnVec}[v, v\downarrow \text{ENTRYEND}]
\text{Close}[s]
\text{result is NULL}

\text{let AddLinkedEntry}[\text{Name1}, \text{Name2}, \text{Na}, \text{Nb}, \text{Nc}, \text{Nd}, \text{Ind}] \text{ be}
\text{let } L1, L2 = \text{Size}[\text{Name1}], \text{Size}[\text{Name2}]
\text{let } La, Lb = \text{Size}[\text{Na}], \text{Size}[\text{Nb}]
\text{and } Lc, Ld = \text{Size}[\text{Nc}], \text{Size}[\text{Nd}]
\text{let } n = 6+L1+L2+La+Lb+Lc+Ld
\text{let } v = \text{NewVec}[n]

v\downarrow \text{ENTRYEND} ::= n
v\downarrow N1 ::= 7
v\downarrow N2 ::= 7+L1
v\downarrow N3 ::= v\downarrow N2 + L2
v\downarrow N4 ::= v\downarrow N3 + La
v\downarrow N5 ::= v\downarrow N4 + Lb
v\downarrow N6 ::= v\downarrow N5 + Lc
\text{Copy}[\text{Name1}, v\downarrow v\downarrow N1, L1]
\text{Copy}[\text{Name2}, v\downarrow v\downarrow N2, L2]
\text{Copy}[\text{Na}, v\downarrow v\downarrow N3, La]
\text{Copy}[\text{Nb}, v\downarrow v\downarrow N4, Lb]
\text{Copy}[\text{Nc}, v\downarrow v\downarrow N5, Lc]
\text{Copy}[\text{Nd}, v\downarrow v\downarrow N6, Ld]
\text{v\downarrow \text{LINKING} ::= } -(v\downarrow \text{LINKING})
\quad \text{|| negative sign indicates linked entry}
§ let Addr = NewLocation[Ind]
     AddVectorFile[Ind, Addr, v, n]
135  ReturnVec[Addr, ASIZE]
     ReturnVec[v, n]

§A

and LoadDiscRtsIfNec[] be
    if NewLocation = UNLOADED do LoadSystemFile["DiscRts"]

and CheckPermission[f] be
145  $P let h = FindHeading[f]
    let Perm = h\$PERM and Owner = h\$OWNER
    ReturnVec[h, h\$0\$AMASK]
    if (Perm = RESTRICTED) or (Perm = OWNERONLY) or (User = Owner) do
        $ Reports["Illegal attempt to write to file *N", f]
    $P

and Size[String] = 1 + ((String\$0) \text{ rshift } 8)/2
155

let DeleteEntry[Name1, Name2, Ind] be
§D
160  let Input = EntriesFromFile[Ind]
    let Output = OutToFile[Ind]

    until Endof[Input] do
        §u let v = Next[Input]
        let Size = v\$ENTRYEND
        if v\$STATUS = DELETED do
            unless EqS[Name1, v+v\$N1] and EqS[Name2, v+v\$N2] do
                TransferOut[Output, v, Size+1]
        ReturnVec[v, Size]
    $u
    Close[Output]

    Close[Input]
§D

****
VIII:4  File Vectors

get 'DECLARATIONS'

global || defined in this segment
§ AddMoreVectorFile : 325
5 VectorFromFile : 323
VectorToFile : 324

§

global || defined in 'DiscRts'
10 § AddVectorFile : 307
NewLocation : 308

§

manifest
15 § ASIZE = 1
UNLOADED = 0

§

let VectorToFile[v, f] be
20 §V let S = OuttoFile[f]
TransferOut[S, lv(v'1), v'0]
Close[S]

§V

25 let VectorFromFile[f] = valof
§V let h = FindHeading[f]
if h = NULL do
§ ReportS['Vectorfrom (deleted) file']
GiveUp[f]

30 §

DiscCo tore[DiscPage, h'LASTPAGE]
§ let Unused = LASTDATA - (DiscPage'ENDOFDATAPTR - 1)
let n = (DATASIZE X h'NUMPAGES) - Unused
ReturnVec[h, h'0^LMASK]

35 § let v = NewVec[n]
v'0 := n
§ let S = InfromFile[f]
TransferIn[S, lv(v'1), v'0]
Close[S]

resultis v
let AddMoreVectoFile[ v, f] be
  let h = FindHeading[ f]
  CheckPerm[ f, h]
  ReturnVec[ h, h\uparrow 0 \uparrow 1 \uparrow \text{IMASK}]
  LoadDiscRtsifNec[]

let Addr = NewLocation[ f]
AddVectoFile[ f, Addr, lv( v\downarrow 1), ( v\downarrow 0 - 1)]
ReturnVec[ Addr, ASIZE]

and LoadDiscRtsifNec[] be
  if NewLocation = UNLOADED do LoadSystemFile['DiscRts']

and CheckPerm[ f, h] be
  let Perm = h\downarrow 0 \downarrow \text{PERM} and Owner = h\downarrow 0 \downarrow \text{OWNER}
  if (Perm = \text{RESTRICTED}) \lor ((Perm = \text{OWNERONLY}) \land (User \neq Owner)) do
    ReportS['Illegal attempt to AddMoreVectoFile *N*, f]
  GiveUp[Owner]

****
VIII:5 DiscRts

get 'DECLARATIONS'

global // defined in this segment

AddVectorFile : 307
NewLocation : 308

manifest // the size and elements of the location vector
ASIZE = 1
APAGE = 0
AWORD = 1

let NewLocation[f] = valof
let Addr = NewVec[ASIZE]

let h = FindHeading[f]
let Owner = h\OWNER
let Page = h\LASTPAGE and Word = 0
ReturnVec[h, h\O\IMASK]

test Page = NULLBODY
then $Page := MakeOnePageBody[f, Owner]
Word := FIRSTDATA

or $DisctoCore[DiscPage, Page]
Word := DiscPage\ENDOFDATAPTR

Addr\APAGE, Addr\AWORD := Page, Word
resultis Addr

and MakeOnePageBody[f, Owner] = valof

let Page = NewDiscBlock[]
DiscPage\SERIAL := 1
DiscPage\THISFILE := f
DiscPage\NXTPAGE := ENDBODY
DiscPage\ENDOFDATAPTR := FIRSTDATA
CoretoDisc[DiscPage, Page]
let AddVectorFile[f, Addr, v, n] be

let Page, Word = Addr\PAGE, Addr\WORD
Vec := NewVec[PAGESIZE-1]
DiscToCore[Vec, Page]
for i=0 to n do
  if Word > LASTDATA do
    Page := TurnPage[Page]
    Word := FIRSTDATA
  VecWord := v\i
  Word := Word+1

if Word > LASTDATA do
  Page := TurnPage[Page]
  Word := FIRSTDATA
Vec\ENDEDATAPTR := Word
CoreToDisc[Vec, Page]

let h = FindHeading[f]
UpdateHead[f, SAME, Vec\SERIAL, Page, SAME, FetchCode[DATE], SAME]
Charge[owner, i]
ReturnVec[h, h\NUMBPAGES]
ReturnVec[Vec, (PAGESIZE-1)]

and TurnPage[p] = value
let Page = NewDiscBlock[]
Vec\NXTPAGE := Page
CoreToDisc[Vec, p]
Vec\SERIAL := Vec\SERIAL + 1
Vec\NXTPAGE := ENDBODY
resultis Page
VIII:6 LinePrinter

get 'DECLARATIONS'

global || defined in this segment
5 $ GeneralLinePrinter : 80
   LinePrinter : 98
$

10 manifest || BytestoLP stream
$ BIPSIZE = 23
    CLEARUP = 19
    BUFF = 23
15 LPPINGS = 5
    PRINTER = 28
    PAUSE = 1000
$

20 manifest || GeneralIntcodoetoLP
$ LPSIZE = 10
    ULBUFF = 7
    ULPTR = 8
    ERROR = 9
25 PAGETHROWS = 10

LPLINLEN = 80
    FLAG = 0
$

30 manifest || Line Printer Character Code
$ BARp = 8 41 ; PAGETHROWp = 8 14
   DOWNARROWp = 8 40 ; PRIMEp = 8 47
35 GRAVEp = 8 43 ; RETURNp = 8 15
   LAMBDAp = 8 134 ; ROUNDop = 8 60
   LOGORp = 8 42 ; RTARROWp = 8 100
   MULTp = 8 140 ; SHARPp = 8 174
   NARROWOp = 8 117 ; SPACEp = 8 40
40 NEQp = 8 44 ; ULP = 8 137
   NEWLINEp = 8 12 ; UPARROWp = 8 45
$
manifest || Extra Internal Character Code

§
ULBIT = 8200 ; EXCLAM = 841
ACUTE = 847 ; HOOK = 822
AT = 8100 ; LAMBDA = 834
BACKSTROKE = 8134 ; LETTERO = 8117
DIGITO = 8460 ; SHARP = 643
DOLLAR = 644 ; STOPCODE = 4

manifest || everything else

§
DEFAULT = 0
MASKUL = E8177577
COMPLETE = 0
UNPRINTABLE = 1
OVERFLOW = 2
PAPERWASTE = 3
IRRELEVANT = 0

let LinePrinter[] = GeneralLinePrinter[DEFAULT]

and GeneralLinePrinter[ErrorFunction]
= GeneralIntcodetoLP[BytestoLP[], ErrorFunction]

and BytestoLP[] = valof
§8
let v = NewVec[BLP.SIZE]

v\_NEXT := StreamError
v\_OUT := OutBLP
v\_CLOSE := CloseBLP
v\_PINGS := LPPINGS
v\_ENDOF := StreamError
v\_RESET := NullProgram
v\_SOURCE := v
v\_STATE := StreamError
v\_RESETSTATE := StreamError
§ let E = lv \( \triangleright DioEXECBLOCK \\
E \downarrow DEVICE := PRINTER \\
E \downarrow BUFFER := lv \downarrow BUFF \\
E \downarrow BUFSIZE := 1 \\
E \downarrow STOPCH := NOTBYTE \\
E \downarrow SEG := DATASEG \\
E \downarrow ENDMODE := QUIETEND \\
E \downarrow SELFPTR := E \\
\] The other three elements of the Execblock are not used

§ let c = lv \( \triangleright DioCLEARUP \\
c \downarrow CSUC := ClearUpChain \\
c \downarrow CPRE := lv \downarrow ClearUpChain \\
c \downarrow ROUTINE := ClearUpLP \\
ClearUpChain := c \\
un\( \downarrow e \downarrow c \downarrow CSUC=\downarrow END\downarrow CUCHN do (c \downarrow CSUC) \downarrow CPRE := c \\
\]

\( v \downarrow (BUFF+1) := lv \downarrow BUFF \] \( \triangleright \) used by InitiateTransfer

resultis v

\§ \$8

and OutBLP[v, x] be
\$OB \( v \downarrow BUFF := x \\
InitiateTransfer[v, 1, 'printer'] \\
\]

\$OB

and CloseBLP[v] be
\$CB let c = lv \( \triangleright DioCLEARUP \\
(c \downarrow CPRE) \downarrow CSUC := c \downarrow CSUC \\
un\( \downarrow e \downarrow c \downarrow CSUC=\downarrow END\downarrow CUCHN do (c \downarrow CSUC) \downarrow CPRE := c \downarrow CPRE \\
\] for i = 1 to PAUSE do \( § \;
\]

Cancel[]

\$CB

\$CB

130 and ClearUpLP[c] be Cancel[]
and Cancel[] be
  %C let i = 0
135  Exec[CANCEL, L, iv i, 0, PRINTER]
  %C

and GeneralIntcodeToLP[Str, ErrorFn] = valof
140  %GI let S = NewVec[LPSIZE]
    %S NEXT  ::= StreamError
    %S OUT   ::= TrapPageThrow || top of page assumed
    %S CLOSE ::= CloseLP
    %S STR   ::= Str
    %S ENDOF ::= StreamError
    %S RESET ::= ResetLP
    %S SOURCE ::= Source[Str]

    %S PAGETHROWS ::= 1
    %S ERROR ::= (ErrorFn=DEFAULT) -> StandardErrorFn, ErrorFn

  % let B = NewVec[LPLINELENGTH] || buffer for underlining
  %S ULBUFFER ::= B
  %S ULPTR  ::= 0
155  %S FLAG ::= false

  resultis S
  %GI

and OutLP[S, x] be
  %O let B = %S ULBUFFER
165  let CharUnderlined = ((x^ULBIT)#0)
  let y = x^MASKUL
  let Ch = valof
  %S switchon y into
    %S default :
5 170    test '()' < y < DELETE
5      then resultis y
5      or endcase || unprintable character

  case "*4":
175  for i=1 to 4 do
5    OutLP[S, CharUnderlined -> '_' , '*a']
5  resultis COMPLETE
case 'n' :
    OutputUnderlines[S]
    Out[S\STR, NEWLINEp]
    S\ULPTR := 0
    result is COMPLETE

case RETURN :
    OutputUnderlines[S]
    Out[S\STR, RETURNp]
    S\ULPTR := 0
    result is COMPLETE

case 'p' :
    OutputUnderlines[S]
    Out[S\STR, PAGETHROWp]
    S\ULPTR := 0
    S\OUT := TrapPageThrow
    S\PAGETHROWS := 1
    result is COMPLETE

case 'b' :
    if S\ULPTR = 0 result is COMPLETE
    OutputUnderlines[S]
    Out[S\STR, RETURNp]
    S\ULPTR := S\ULPTR-1
    for i=1 to S\ULPTR do
        Out[S\STR, SPACEp]
        S\ULPTR := S\ULPTR-1
    result is COMPLETE

210

215

case 's' :
    result is SPACEp

220

case PRIME :
    result is PRIMEp

case 'p' :
    result is RTARROWp

case 'v' :
    result is LOGORp

case 'x' :
    result is MULTp

case LAMBDA :
    result is LAMBDAp

225

case 'a' :
    result is NBQp

case 'A' :
    result is UPARROWp

case 'q' :
    result is DOWNARROWp

case EXCLAM :
    OutLP[S, '*']

230

case SHARP :
    result is SHARPPp
case DOLLAR: OutLP[S, 'S']
               result is BARp
               OutLP[S, '*b']

               result is BARp
225
               result is NARROWOp
               result is ROUNDOP
               result is GRAVEp
               result is BARp

230
               result is BARp
               result is GRAVEp
               result is BARp

235
               result is BARp
               result is GRAVEp
               case AT : case BACKSTROKE :
               case LEFTARROW : endcase || all unprintable
               result is COMPLETE

               result is COMPLETE
               x := (S\ERROR)[S, UNPRINTABLE, x]
               unless x = COMPLETE do OutLP[S, x]
               result is COMPLETE
               "v

240
               if s\ULPTR > ILINELENGTH do
               Ch := (S\ERROR)[S, OVERFLOW, Ch]

               if Ch = COMPLETE return
               Out[S\STR, Ch]
               S\ULPTR := S\ULPTR + 1
               B\((S\ULPTR) := CharUnderlined \rightarrow ULp, SPACEp
               B\FLAG := CharUnderlined \lor B\FLAG

               result is COMPLETE
245
               OutputUnderlines[S] be
               \[SUL let B = S\ULBUFF
               unless B\FLAG return
               B\FLAG := false
               Out[S\STR, RETURNp]
               TransferOut[S\STR, lv (B\1), S\ULPTR]
               \]SUL

250

255 and OutputUnderlines[S] be
               \[SUL let B = S\ULBUFF
               unless B\FLAG return
               B\FLAG := false
               Out[S\STR, RETURNp]
               TransferOut[S\STR, lv (B\1), S\ULPTR]
               \]SUL

260

265 and TrapPageThrow[S, x] be
               \[STP test x = '*p'
               then S\PAGETHROWS := S\PAGETHROWS + 1
               \]STP
or § let p = $\downarrow$PAGETHROWS
$\downarrow$PAGETHROWS := 0
$\downarrow$OUT := OutLP

\[ \text{if } p > 1 \text{ do } p := (S\downarrow ERROR)[S, \text{ PAPERWASTE}, p] \]
OutLP[S, x]

§TP

and CloseLP[S] be
§C
\[ \text{let } B = S\downarrow ULBUFF \]
\[ \text{let } p = S\downarrow PAGETHROWS \]
OutputUnderlines[S]

\[ \text{if } p > 1 \text{ do } p := (S\downarrow ERROR)[S, \text{ PAPERWASTE}, p] \]
if $\downarrow$PAGETHROWS = 0 do Out[S\downarrow STR, PAGETHROWp]
Close[S\downarrow STR]
ReturnVec[S, LPSIZE]

§C

and ResetLP[S] be
§R
unless $\downarrow$ULPTR = 0 do Out[S, "*n"]

\[ \text{let } p = S\downarrow PAGETHROWS \]

\[ \text{if } p > 1 \text{ do } \]
\[ \text{let } (S\downarrow ERROR)[S, \text{ PAPERWASTE}, p] \]
$\downarrow$PAGETHROWS := 1

§R

and StandardErrorFn[S, Reason, x] = valof

\[ \text{switch } \text{Reason } \text{into } \]
\[ \text{case } \text{UNPRINTABLE: } \]
\[ \text{let } y = x\uparrow \text{MASKUL } \]
\[ \text{if } y = \text{ACUTE } \text{result is } \text{PRIME}(x\uparrow \text{ULBIT}) \]
\[ \text{if } (x = \text{RUNOUT}) \text{v}(x = \text{STOPCODE}) \text{ result is } \text{COMPLETE} \]

\[ \text{test } \text{HOOK < } y < \text{DELETE } \]
then § PrinterReport["*O unprintable", x]
\[ \text{result is } *[S \text{v}(x\uparrow \text{ULBIT})] \]

§R

or § PrinterReport["*O invalid", x]
\[ \text{result is } \text{COMPLETE} \]
case OVERFLOW:
    Out[S, "n"]
    PrinterReport["Line too long"]
    result is x

case PAPERWASTE:
    PrinterReport['N consecutive page throws', x]
    result is IRRELEVANT

default:
    PrinterReport['invalid call of StandardErrorFn']
    result is NULL

and PrinterReport[String, a, b, c, d] be

ReportS[ReportStream, '*Printer: ']

and Source[str] = valov

if str< do str := (~str)\$SLOWBLOCK
result is Str\$SOURCE

***
<table>
<thead>
<tr>
<th>Command</th>
<th>Text</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULLProgram</td>
<td>53</td>
<td>66</td>
</tr>
<tr>
<td>NUMFACES</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>NTH</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>NOTPAGE</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>CE</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>OFFSET</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>OPACES</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>CEFSIZE</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>CLOUDBY</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>PALM</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>CONTINUE</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>System text</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Operator</td>
<td>21</td>
<td>26</td>
</tr>
<tr>
<td>Operator's</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>Operator's</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>ODB -</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Operator's</td>
<td>33</td>
<td>42</td>
</tr>
<tr>
<td>Operator's</td>
<td>33</td>
<td>42</td>
</tr>
<tr>
<td>OUTF</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td>73</td>
<td>98</td>
</tr>
<tr>
<td>OUTADD</td>
<td>35</td>
<td>43</td>
</tr>
<tr>
<td>OUTAPPEND</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>OUTBPPR</td>
<td>128</td>
<td>137</td>
</tr>
<tr>
<td>OUTLUP</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>OUTLIBDEF</td>
<td>53</td>
<td>80</td>
</tr>
<tr>
<td>OUTLIB</td>
<td>33</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBTIME</td>
<td>71</td>
<td>64</td>
</tr>
<tr>
<td>OUTLIBTIME+</td>
<td>51</td>
<td>64</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>OUTLIBTID</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBTIDE</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBTIDE+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBTID+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBTID</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBTIDE</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBTIDE+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBTID+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT+</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>OUTLIBT</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>Text</td>
<td>Code</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>30 39</td>
<td></td>
</tr>
<tr>
<td>STATE</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>StateBit</td>
<td>45 57</td>
<td></td>
</tr>
<tr>
<td>status</td>
<td>39 48</td>
<td></td>
</tr>
<tr>
<td>STATUS</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>STORCH</td>
<td>94 95</td>
<td></td>
</tr>
<tr>
<td>Storage allocation (core)</td>
<td>23 29</td>
<td></td>
</tr>
<tr>
<td>Storage allocation (disc)</td>
<td>56 70</td>
<td></td>
</tr>
<tr>
<td>StoreCode</td>
<td>72 90</td>
<td></td>
</tr>
<tr>
<td>STR</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>STREAM</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>Stream elements used by</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>Stream primitives</td>
<td>39 38</td>
<td></td>
</tr>
<tr>
<td>Stream vector constants</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>STREAMABLE</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>StreamError</td>
<td>30 40</td>
<td></td>
</tr>
<tr>
<td>Streams from files</td>
<td>58 74</td>
<td></td>
</tr>
<tr>
<td>STREAM</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>Stream index</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>STREAM</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>System index</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>System set-up</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>SYSTEM</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>SystemIndex</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>TELERIC</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>Teletype</td>
<td>37 45</td>
<td></td>
</tr>
<tr>
<td>Teletype character code</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>Teletype stream constants</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>TerminateRun</td>
<td>3 9</td>
<td></td>
</tr>
<tr>
<td>Text of the system</td>
<td>1 8</td>
<td></td>
</tr>
<tr>
<td>TIMEFILE</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>TIMEOFDAYCLOCK</td>
<td>95 112</td>
<td></td>
</tr>
<tr>
<td>TRANSFER</td>
<td>94 91</td>
<td></td>
</tr>
<tr>
<td>TransferIn</td>
<td>74 96</td>
<td></td>
</tr>
<tr>
<td>TransferInC</td>
<td>74 96</td>
<td></td>
</tr>
<tr>
<td>TransferOut</td>
<td>74 96</td>
<td></td>
</tr>
<tr>
<td>Traps and Traps</td>
<td>125 141</td>
<td></td>
</tr>
<tr>
<td>Traps (See Permissions)</td>
<td>125 141</td>
<td></td>
</tr>
<tr>
<td>TrapAgain</td>
<td>45 57</td>
<td></td>
</tr>
<tr>
<td>TryAgain</td>
<td>50 61</td>
<td></td>
</tr>
<tr>
<td>TT</td>
<td>86 107</td>
<td></td>
</tr>
<tr>
<td>TThread</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>THREAD</td>
<td>95 92</td>
<td></td>
</tr>
<tr>
<td>THREAD</td>
<td>95 92</td>
<td></td>
</tr>
<tr>
<td>Threads</td>
<td>119 135</td>
<td></td>
</tr>
<tr>
<td>TurnPage (AddVectorFile)</td>
<td>119 135</td>
<td></td>
</tr>
<tr>
<td>TurnPage (addToFile)</td>
<td>63 80</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Types of file</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>US</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>UMASK</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>UNTYPED</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>Upload</td>
<td>8 44</td>
<td></td>
</tr>
<tr>
<td>UNRESTRICTED</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>UP-AGAIN</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Update</td>
<td>66 84</td>
<td></td>
</tr>
<tr>
<td>UpdateDiscVectorElement</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>UpdateHead</td>
<td>67 83</td>
<td></td>
</tr>
<tr>
<td>UpdatePermission</td>
<td>111 125</td>
<td></td>
</tr>
<tr>
<td>User</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>User details</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>Validator</td>
<td>83 105</td>
<td></td>
</tr>
<tr>
<td>Vector allocation</td>
<td>23 29</td>
<td></td>
</tr>
<tr>
<td>vectorFromFile</td>
<td>110 133</td>
<td></td>
</tr>
<tr>
<td>vectorToFile</td>
<td>110 133</td>
<td></td>
</tr>
<tr>
<td>VTA#</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>wait</td>
<td>53 66</td>
<td></td>
</tr>
<tr>
<td>WIC</td>
<td>72 89</td>
<td></td>
</tr>
<tr>
<td>WordsFromFile</td>
<td>47 60</td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td>35 43</td>
<td></td>
</tr>
<tr>
<td>WriteAddr</td>
<td>35 43</td>
<td></td>
</tr>
<tr>
<td>WRITELOCK</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>WriteByte</td>
<td>35 43</td>
<td></td>
</tr>
<tr>
<td>writer</td>
<td>35 43</td>
<td></td>
</tr>
<tr>
<td>writer</td>
<td>35 43</td>
<td></td>
</tr>
<tr>
<td>writers</td>
<td>31 41</td>
<td></td>
</tr>
<tr>
<td>written</td>
<td>72 89</td>
<td></td>
</tr>
<tr>
<td>Wrong</td>
<td>82 104</td>
<td></td>
</tr>
<tr>
<td>MIPS</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>