

*

* REGISTER DECLARATIONS.

CONFIGURE APL;

* GENERAL REGISTERS:

DR ZERO ← 0;
DR CONDITIONS ← 1; DR COUNTER ← 1;
DR LINK ← 2;
DR FLAGS ← 3; * ASSORTED FLAG BITS
DR MEMADDR ← 4; * ADDRESS FOR MEMORY ACCESS
DR MEMDATAL ← 5; * LEFT HALF OF MEMORY DATUM
DR IOSTATUS ← 6; * INPUT/OUTPUT STATUS
DR MEMDATAR ← 7; * RIGHT HALF OF MEMORY DATUM
DR PFIRST ← 8; * BASE OF PROCEDURE SEGMENT
DR PNLEN ← 9; * 2'S COMPLEMENT OF PSEG LENGTH
DR LFIRST ← 10; * BASE OF LOCAL SEGMENT
DR LNLEN ← 11; * 2'S COMPLEMENT OF LSEG LENGTH
DR GFIRST ← 12; * BASE OF GLOBAL SEGMENT
DR GNLEN ← 13; * 2'S COMPLEMENT OF GSEG LENGTH
DR LBASE ← 14; * LOCAL ENVIRONMENT POINTER
DR LTOP ← 15; * TOP-OF-STACK POINTER
DR PBASE ← 16; * PROCEDURE BASE POINTER
DR PCTR ← 17; * CURRENT CODE BYTE POINTER (PBASE RELATIVE)
DR AL ← 18; * AL,AR IS THE
DR AR ← 19; * PRIMARY ACCUMULATOR
DR BL ← 20; * BL,BR IS THE
DR BR ← 21; * SECONDARY ACCUMULATOR
DR T0 ← 22; DR FIRSTBYTE ← 22; * USUALLY HOLDS CURRENT SYL.
DR T1 ← 23; DR SEG ← 23;
DR T2 ← 24; DR ADDRESS ← 24;
DR T3 ← 25;
DR T4 ← 26;
DR T5 ← 27;
DR T6 ← 28;
DR T7 ← 29;
DR T8 ← 30;
DR T9 ← 31; DR TEMP ← 31;

* REGISTERS FOR GENSCALAR:

DR OPNO ← 22; * INDEX IN GSTAB (SAME REGISTER AS FIRSTBYTE)
DR APTR ← 23; * CURSOR IN FIRST OPERAND
DR BPTR ← 24; * CURSOR IN SECOND OPERAND
DR NELTS ← 25; * COUNTER (SAME AS MKNELTS BELOW)
DR RPTR ← 26; * CURSOR IN RESULT ARRAY
DR OPRET ← 27; * RETURN ADDRESS FOR OPERATOR ROUTINE

* REGISTERS FOR MONADIC AND DYADIC SCALAR OPERATOR ROUTINES:

DR AEXP ← 28; * EXPONENT OF FLOATING-POINT RESULT
DR ASIGN ← 29; * SIGN OF RESULT
DR FT1 ← 30; * TEMP
DR FT2 ← 31; * TEMP

* REGISTERS FOR MAKE():

DR MKNELTS ← 25; * NUMBER OF ELEMENTS IN ARRAY TO ALLOCATE
DR MKRANK ← 26; * RANK FOR NEW ARRAY BLOCK
DR MKP ← 27; * POINTER TO NEW ARRAY (ON RETURN FROM MAKE())
DR MKSIZE ← 28; * SIZE OF NEW ARRAY BLOCK (ON RETURN FROM MAKE())

```
DR MKT1      ← 29; * TEMP
DR MKT2      ← 30; * TEMP
DR MKT3      ← 31; * TEMP
* REGISTERS FOR FREE():
DR FRP       ← 28; * POINTER TO ARRAY BLOCK TO FREE
DR FRSIZE    ← 29;
DR FRT1      ← 30;
DR FRT2      ← 31;
```

*

* FLAG BITS

* HERE ARE MACROS TO SET AND CLEAR BITS OF THE FLAG REGISTER --

* THE ARGUMENT FOR EACH IS A MASK WITH A 1 FOR EACH BIT TO BE AFFECTED.

DM CLEARFLAG(BITS) ← FLAGS←FLAGS AND (0-(BITS)-1);

DM SETFLAG(BITS) ← FLAGS←FLAGS OR BITS;

* HERE ARE THE DEFINITIONS OF THE BIT POSITIONS.

DC PCB13 ← 0;

* THESE THREE BITS

DC PCB14 ← 1;

* MUST BE LEFTMOST

DC PCB15 ← 2;

DC PCB15BIT ← 020000B;

* IN FLAGS

DC PCLOWB ← 160000B;

*

DC NUMOP ← 3; DC NUMOPB ← 010000B;

DC MONOP ← 4; DC MONOPB ← 004000B;

DC ASCALAR ← 5; DC ASCALARB ← 002000B;

DC BSCALAR ← 6; DC BSCALARB ← 001000B;

DC RARRAY ← 7; DC RARRAYB ← 000400B;

DC INTRPT ← 8; DC INTRPTB ← 000200B;

DC OPFLAGSB ← 017600B;

*

DC FCNCTRC ← 12;

DC FCNRTRC ← 13;

DC STEP ← 14;

*

DC IORG ← 15; DC IORGB ← 000001B; * THIS FLAG MUST BE RIGHTMOST

*
* INTERPROCESSOR COMMUNICATION

*
* IPC FLIPFLOPS (IOSTATUS BIT POSITIONS)

DC FFSAREAD ← 10; * READ FF[SA]
DC FFSARST ← 12; DC FFSARSTB ← 000010B; * RESET FF[SA]
DC FFASSET ← 13; DC FFASSETB ← 000004B; * SET FF[AS]

*
* ADDRESSES OF COMMUNICATION WORDS

DC APLCOMWD ← 457B; * SIMPLE-TO-APL COMMAND WORD
DC APLSTATWD ← 460B; * APL-TO-SIMPLE STATUS WORD
DC APLDSEGWD ← 461B; * DATA SEGMENT DESCRIPTOR WORD
DC APLPSEG0WD ← 462B; * PROCEDURE SEGMENT DESCRIPTOR WORD
DC APLPSEG1WD ← 463B; * ALTERNATE PROCEDURE SEGMENT DESCRIPTOR WORD

*
* STATUS CODES ISSUED BY THE APL PROCESSOR

DC STATWHOOPS ← 1; * "WHOOPS, STARTED WHILE RUNNING"
DC STATOK ← 2; * "OK, STOPPED"
DC STATTRAP ← 3; * "TRAP"

*

* PROCEDURE BLOCKS

* 1ST WORD, LEFT HALF:

DC CALLTRACE ← 1;

DC RINTRACE ← 2;

DC RESULT ← 3;

DC NLINESB ← 007777B;

* 1ST WORD, RIGHT HALF:

DC NARGSB ← 170000B;

DC NLOCALSB ← 007777B;

* PROCEDURE BLOCK-RELATIVE ADDRESS OF FIRST CODE SYLLABLE:

DC CODEOFFSET ← 2;

*

* DATA SEGMENT STATE AREA

DC SBSP ← 0; * CONTAINS SBASE, SPTR
DC FLLB ← 1; * CONTAINS FLAGS, LBASE
DC TRAPINFO ← 2; * GETS TRAP CLASS, NUMBER
DC SAVENO ← 3; * SAVE NELTS, OPNO AFTER INTERRUPT OF GENOP
DC SAVEAB ← 4; * " APTR, BPTR "
DC SAVERX ← 5; * " RPTR "
DC ROVER ← 6; * FREE POINTER FOR ARRAY SPACE
DC BLOCKPTR ← 7; * ADDRESS OF LAST ARRAY BLOCK ALLOCATED
*
DC GLOBALOFFSET ← 8; * G-RELATIVE ADDRESS OF FIRST GLOBAL VARIABLE


```

*
* DESCRIPTOR TYPES
DC FLOTYPE ← 1; * NUMERIC SCALAR IN FLOATING-POINT FORMAT
DC INTTYPE ← 2; * NUMERIC SCALAR IN INTEGER FORMAT
DC CHTYPE ← 3; * CHARACTER SCALAR
DC ARYTYPE ← 4; * ARRAY
DC REFTYPE ← 5; * INDIRECT PARAMETER WORD
DC UNDFTYPE ← 6; * UNSPECIFIED VALUE
*
* DESCRIPTOR TEMPLATES
DC FLOTYPEB ← 040000B;
DC INTTYPEB ← 020000B;
DC CHTYPEB ← 010000B;
DC ARYTYPEB ← 004000B;
DC REFTYPEB ← 002000B;
DC UNDFTYPEB ← 001000B;
*
* MASKS FOR INTERESTING SETS OF TYPES
DC NUMTYPEB ← 060000B; * NUMERIC SCALAR TYPES
DC SCATYPEB ← 070000B; * SCALAR TYPES
*
* FIELDS OF A FLOTYPE VALUE (FLOATING-POINT NUMBER).
* LEFT HALF:
DC SIGN ← 0; DC SIGNB ← 100000B; DC SIGNC ← 077777B;
DC COEFB1 ← 1;
* RIGHT HALF:
DC COEFB23 ← 7; DC COEFB23BIT ← 000400B;
DC ROUND1 ← 8;
DC ROUND2 ← 9; DC ROUND2B ← 000100B;
DC STICKY ← 10; DC STICKYB ← 000040B;
DC EXPB ← 000377B; DC EXPC ← 177400B;
*
DC EXPBIAS ← 200B;
*
DC FLPT1L ← 040000B; DC FLPT1R ← 000200B; * FLOATING-POINT ONE
*
* OPCODE BITS
DC OPCB0 ← 8; * 1XXXMMMM
DC OPCB1 ← 9; * X1XXMMMM
DC OPCB2 ← 10; * XX1XMMMM
DC OPCB3 ← 11; * XXX1MMMM
DC OPCB4 ← 12; * XXXX1XXX
DC OPCB5 ← 13; * XXXXX1XX
DC OPCB6 ← 14; * XXXXXX1X
DC OPCB7 ← 15; * XXXXXXXX1
*
* ARRAY BLOCKS
* MACRO TO SET NEXT POINTER IN FREE BLOCK.
DM SETNEXT(PTR,VAL) ← TESTGRMW(PTR P1). WAITREAD.
MEMDATAR←MEMDATAR. MEMDATAL←VAL ANDWRITE;
* MACRO TO SET BACK POINTER IN FREE BLOCK.
DM SETBACK(PTR,VAL) ← TESTGRMW(PTR P1). WAITREAD.
MEMDATAR←VAL. MEMDATAL←MEMDATAL ANDWRITE;

```

```

* FIELDS IN 1ST HEADER WORD:
DC THISFREE ← 0;      DC THISFREEB ← 100000B;      DC THISFREEC ← 077777B;
DC PREVFREE ← 1;      DC PREVFREEB ← 040000B;      DC PREVFREEC ← 137777B;
                        DC SLOPB      ← 037400B;
                        DC RANKB      ← 000377B;
* MAXIMUM SLOP ALLOWED. THIS SYMBOL IS REFERENCED IN EXACTLY ONE PLACE,
* IN THE ROUTINE MAKE().
DC MAXSLOP ← 4;
* ARRAY BLOCK-RELATIVE ADDRESS OF FIRST SHAPE WORD:
DC SHAPEOFFSET ← 2;
*
* MISC.
DC PSEGSEL ← 4;      DC PSEGSELB ← 004000B;
                        DC FCNOB   ← 003777B;
*
DC LSEG      ← 0;      * LOCAL SEGMENT
DC GSEG      ← 1;      * GLOBAL SEGMENT
*
DC ONEINST ← 2;      * ONE MICROINSTRUCTION OCCUPIES TWO WORDS
*
DC RMASK1 ← 000001B;
DC RMASK4 ← 000017B;
DC RMASK6 ← 000077B;
DC RMASK8 ← 000377B;
DC RMASK12 ← 007777B;

```

```

*
*
* TRAP HANDLING.

*
* CLASS 1 -- STORAGE OVERFLOW

STKOFI:    LTOP←LTOP-1;
STKOF:     T2←1; GOTO STGCLASS;
MKGOF:     T2←2;
STGCLASS:  T1←1; GOTO TRAP;

*
* CLASS 2 -- DEBUG

STEPTRAP:  T2←1; GOTO DBG1CLASS;
CALLTRAP:  T2←2;
DBG1CLASS: T1←2; GOTO TRAP1;
*
RTNTRAP:   T2←3; GOTO DBGCLASS;
BKPTTRAP:  T2←4;
DBGCLASS:  T1←2; GOTO TRAP;
*
* THESE TRAPS OCCUR AT THE
* COMPLETION OF THE INSTR.
* DON'T DECREMENT PCTR

* AS USUAL, THESE TRAPS OCCUR
* AT THE BEGINNING OF THE
* INSTRUCTION

*
* CLASS 3 -- SYSTEM AND/OR COMPILER ERRORS

PERR:      T2←1; GOTO SCCLASS;
LERR:      T2←2; GOTO SCCLASS;
GERR:      T2←3; GOTO SCCLASS;
*
STKUFERR:  T2←4; GOTO SCCLASS;
*
WHATERR:   T2←5; GOTO SCCLASS;
*
REFERR:    T2←6; GOTO SCCLASS;
*
MRSYLERR:  T2←7; GOTO SCCLASS;
MRUNDERR:  T2←8; GOTO SCCLASS;
*
GSUNDERR:  T2←9; GOTO SCCLASS;
GOUNDERR:  T2←10; GOTO SCCLASS;
*
ETCUNDERR: T2←11; GOTO SCCLASS;
*
RETURNERR: T2←12;
SCCLASS:   T1←3; GOTO TRAP;

*
* CLASS 4 -- ATTENTION

ATTNTRAP:  T1←4; GOTO TRAPN0;

*
* CLASS 5 -- RANK ERROR

```

```
RANKERROR: T1+5; GOTO TRAPN0;

*
* CLASS 6 -- LENGTH ERROR

LENGTHERROR:T1+6; GOTO TRAPN0;

*
* CLASS 7 -- TYPE ERROR

TYPEERROR: T1+7; GOTO TRAPN0;

*
* CLASS 8 -- DOMAIN ERRORS

IOF:          T2+1; GOTO DOMCLASS;
*
FEXPUF:      T2+2; GOTO DOMCLASS;
FEXPOF:      T2+3; GOTO DOMCLASS;
*
DIV0ERR:     T2+4; GOTO DOMCLASS;
*
DOMAINERROR:T2+5;
DOMCLASS:    T1+8; GOTO TRAP;

*
* CLASS 9 -- INDEX ERRORS

IXRNARGERR:T2+1; GOTO IDXCLASS;
INDEXERROR:T2+2;
IDXCLASS:    T1+9; GOTO TRAP;

*
* CLASS 10 -- VALUE ERROR

VALUEERROR:T1+10; *GOTO TRAPN0;

* TRAP OF CLASS WHICH TAKES NO NUMBER.
TRAPN0:      T2+0;
              *GOTO TRAP;
```

```

*
* HANDLE A TRAP. (T1,T2) CONTAIN THE TRAP (CLASS,NUMBER), WHICH WILL
* BE STORED IN THE STATE AREA OF THE DATA SEGMENT. THE PCTR MUST BE
* DECREMENTED BY 1+FLAGSSPCLOW . IF FLAGSSRARRAY=1, BLOCKPTR CONTAINS
* THE ADDRESS OF AN ARRAY WHICH SHOULD BE FREED. THEN THE APL PROCESSOR
* IS READY TO STOP. TRAPS WHICH SHOULDN'T DECREMENT THE PCTR (STEPTRAP
* AND CALLTRAP) CAN BE HANDLED BY ENTERING THROUGH TRAP1.

* SET T3 TO VALUE OF LEFTMOST 3 BITS OF FLAGS (I.E. FLAGSSPCLOW).
TRAP:      T3←FLAGS R8;
           T3←T3 R1; T3←T3 R1; T3←T3 R1; T3←T3 R1; T3←T3 R1;
* SET PCTR TO PCTR-(T3+1).
           T3←T3 EOR -1;
           PCTR←PCTR+T3;
* STORE TRAP INFORMATION IN STATE AREA OF DATA SEGMENT.
TRAP1:    MEMDATAL←T1; MEMDATAR←T2;
           WRITEG( TRAPINFO );
* SET T1 TO STATUS CODE TO BE ISSUED BY STOP.
           T1←STATTRAP;
           GOTO STOP IF FLAGSSRARRAY=0;
* RELEASE THE WOULD-BE RESULT ARRAY, AFTER CLEARING FLAGSSRARRAY
* TO MAKE A "TRAP LOOP" TOTALLY IMPOSSIBLE).
           READG( BLOCKPTR );
           CLEARFLAG( RARRAYB );
           FRP←MEMDATAL PAUSE;
           CALL FREE;
* SET FLAGSSRARRAY TO 1 AGAIN (TO HELP DEBUGGING) AND STOP.
           SETFLAG( RARRAYB );
           GOTO STOP;

```

```
*  
* SPECIAL TRAP HANDLERS FOR CONDITIONS ENCOUNTERED DURING START UP.  
* PROCESSOR STATE HASN'T BEEN LOADED YET, SO THE ORDINARY TRAP LOGIC  
* (WHICH SAVES THE STATE) CAN'T BE USED. ALL ARE CLASS 3!
```

```
*  
* "LERR"
```

```
STLERR: T2+2; GOTO STSCCLASS;
```

```
*  
* "GERR"
```

```
STGERR: T2+3; GOTO STSCCLASS;
```

```
*  
* "STKUFERR"
```

```
STSTKUFERR: T2+43; *GOTO STSCCLASS;
```

Bug 2



```
STSCCLASS: MEMDATAL+3; MEMDATAR-T2;  
WRITEG( TRAPINFO );  
* SET T1 TO STATUS CODE FOR STOPSTOP.  
T1+STATTRAP;  
*  
GOTO STOPSTOP;
```

*
* THE APL PROCESSOR RECEIVED A COMMAND WHILE RUNNING, AND WILL STOP
* REGARDLESS OF WHETHER THE COMMAND WAS STOP OR START (THE ONLY
* DIFFERENCE IS THE STATUS RETURNED). FIRST THE SIMPLE-TO-APL FLIPFLOP
* MUST BE RESET. THEN THE STATUS CODE WILL BE DETERMINED.

STOPPED: IOSTATUS←FFSARSTB; IOSTATUS←0;

*

READ(APLCOMWD);

* COMMANDS ARE 0 FOR STOP, 1 FOR START.

* SET T1 TO STATUS TO BE RETURNED:

* STATWHOOFS FOR "WHOOFS, STARTED WHILE RUNNING"

* STATOK FOR "OK, STOPPED"

WAITREAD;

T1←STATOK;

GOTO STOP IF MEMDATAR\$15=0;

T1←STATWHOOFS;

*GOTO STOP;

*
* THE PROCESSOR IS TO STOP. T1 SHOULD CONTAIN A STATUS CODE TO BE
* ISSUED. STORE PCTR IN THE CURRENT STACK FRAME; STORE LTOP, LBASE,
* AND FLAGS IN THE STATE AREA OF THE DATA SEGMENT.

STOP: RMWL(LBASE P1);
 WAITREAD;
 MEMDATAR←PCTR; MEMDATAL←MEMDATAL ANDWRITE;
*
 MEMADDR←GFIRST+SBSP MR MW; * RMWG(SBSP)
 WAITREAD;
 MEMDATAR←LTOP; MEMDATAL←MEMDATAL ANDWRITE;
*
 MEMDATAL←FLAGS; MEMDATAR←LBASE;
 WRITEG(FLLB);
* ISSUE STATUS CODE.
STOPSTOP: MEMDATAL←0; MEMDATAR←T1;
 WRITE(APLSTATWD);
 IOSTATUS←FFASSETB; IOSTATUS←0;
* ENTER IDLE STATE.
 *GOTO IDLE;

*
* THE APL PROCESSOR STAYS IN ITS IDLE LOOP UNTIL IT RECEIVES A START
* COMMAND. (STOP COMMANDS RECEIVED WHILE IDLING ARE IGNORED.)

IDLE: TEMP←IOSTATUS;
 GOTO IDLE IF TEMP\$FFSAREAD=0; * NO COMMAND
* THE SIMPLE-TO-APL FLIPFLOP IS SET. RESET IT AND EXAMINE THE
* ASSOCIATED COMMAND WORD IN CENTRAL MEMORY.
 IOSTATUS←FFSARSTB; IOSTATUS←0;

*
 READ(APLCOMWD);
* ONLY THE LOW ORDER BIT OF THE COMMAND WORD HAS MEANING:
* 0 => STOP,
* 1 => START.
 WAITREAD;
 GOTO IDLE IF MEMDATAR\$15=0; * ALREADY STOPPED
*GOTO START;

```

*
* LOAD THE APL PROCESSOR STATE AND BEGIN RUNNING.
*
* THE CELLS APLDSEGWD, APLPSEG0WD, AND APLPSEG1WD CONTAIN THE
* (BASE ADDRESS, LENGTH) OF THE DATA SEGMENT, PROCEDURE SEGMENT, AND
* ALTERNATE PROCEDURE SEGMENT, RESPECTIVELY.
*
* THE DATA SEGMENT CONTAINS: SOME STATE WORDS, THE GLOBAL VARIABLES, AND
* ARRAY STORAGE SPACE (COLLECTIVELY KNOWN AS THE "GLOBAL SEGMENT"
* THROUGHOUT THE APL PROCESSOR); AND THE STACK (KNOWN AS THE "LOCAL
* SEGMENT").
*
* THE FIRST WORD OF THE DATA SEGMENT HOLDS TWO 16-BIT QUANTITIES, SBASE
* AND SPTR. SBASE IS THE LENGTH OF THE GLOBAL SEGMENT, WHICH IS
* EQUAL TO THE DATA SEGMENT-RELATIVE ADDRESS OF THE LOCAL SEGMENT.
* SPTR IS THE SBASE-RELATIVE ADDRESS OF THE FIRST AVAILABLE STACK
* CELL, OR ALTERNATIVELY IS THE LENGTH OF THE IN-USE PART OF THE STACK.
*
* FROM THESE QUANTITIES WE COMPUTE THE VALUES FOR THE BASE-BOUND
* REGISTERS GFIRST, GNLEN, LFIRST, AND LNLEN, AND THE VALUE FOR LTOP.

START:      READ( APLDSEGWD );
* THE BASE OF THE DATA SEGMENT IS THE BASE OF THE GLOBAL SEGMENT.
      GFIRST←MEMDATAL PAUSE;
      TEMP←MEMDATAR EOR -1;          * TEMP←-1-(DATA SEGMENT LEN)
* READ WORD OF DATA SEGMENT WITH SBASE, SPTR.
      READG( SBSP );
* SET GNLEN TO -SBASE.
      GNLEN←MEMDATAL EOR -1 PAUSE; GNLEN←GNLEN+1;
* SET LFIRST TO SBASE+(BASE OF DATA SEGMENT).
      LFIRST←MEMDATAL+GFIRST;
* SET LNLEN TO -(LENGTH OF DATA SEGMENT - SBASE)
      LNLEN←MEMDATAL+TEMP P1;
* SET LTOP TO SPTR.
      LTOP←MEMDATAR;
*
* THE SECOND WORD OF THE DATA SEGMENT CONTAINS VALUES FOR THE REGISTERS
* FLAGS AND LBASE.
      READG( FLLB );
      FLAGS←MEMDATAL PAUSE;
      LBASE←MEMDATAR;
*
* NOW WE MUST CHECK THAT GNLEN, LNLEN, LTOP, AND LBASE ARE
* "REASONABLE". THE FOLLOWING CONDITIONS MUST BE SATISFIED:
* GLOBALOFFSET≤(-GNLEN)
* LBASE<LTOP<(-LNLEN)
      ZERO←GNLEN+(GLOBALOFFSET-1);
      GOTO STGERR IF CARRY;          * GLOBALOFFSET>(-GNLEN)
*
      TEMP←LTOP EOR -1;
      ZERO←LBASE+TEMP P1;
      GOTO STSTKUFERR IF CARRY;     * LBASE>=LTOP
*

```

```
ZERO←LNLEN+LTOP;
GOTO STLERR IF CARRY;          * LTOP>=(-LNLEN)
*
* SET PFIRST, PNLEN, PBASE, AND PCTR.
  CALL LOADP;
*
* NORMALLY EXECUTION BEGINS AT CYCLE1. HOWEVER IF FLAGSSINTRPT=1,
* GENOP WAS INTERRUPTED AND SHOULD BE CONTINUED.
  GOTO GOCONT IF FLAGSSINTRPT=1;
  CLEARFLAG( PCLOWB+OPFLAGSB );
  GOTO CYCLE1;
```

```

*
*
* LOADP()
*
* SETS PFIRST, PNLEN, PBASE, AND PCTR USING THE INFORMATION IN THE
* CURRENT STACK FRAME.
*
* USES:      T8, T9, MEMORY REGISTERS.

LOADP:      READL( LBASE P1 );
* SET T8 TO ADDRESS OF SELECTED PROCEDURE SEGMENT DESCRIPTOR, EITHER
* APLPSEG0WD OR APLPSEG1WD.
          T8←APLPSEG0WD;
          GOTO LP1 IF MEMDATAL$PSEGSEL=0 PAUSE;
          T8←APLPSEG1WD;
* SET T9 TO FUNCTION NUMBER.
LP1:        T9←MEMDATAL AND FCNOB;
* SET PCTR TO VALUE SAVED IN CURRENT STACK FRAME.
          PCTR←MEMDATAR;
* READ THE SELECTED PROCEDURE SEGMENT DESCRIPTOR.
          READ( T8 );
* SET PFIRST, PNLEN FROM THE DESCRIPTOR.
          PFIRST←MEMDATAL PAUSE;
          PNLEN←MEMDATAR EOR -1; PNLEN←PNLEN+1;
* SET PBASE FROM PROCEDURE\DIRECTORY[FCNO] AND RETURN.
          T9←T9+T9;
          READTESTP( T9 );
          PBASE←MEMDATAL PAUSE ANDRETURN;

```

```

*
*
* NEXTBYTE()
* SETS:    TEMP = NEXT BYTE OF CODE.
* UPDATES: FLAGSPCLOW, PCTR.
* USES:    MEMORY REGISTERS
* NOTE:    NEXTBYTE() ASSUMES MEMDATA L,R HAVEN'T BEEN USED SINCE
*          THE FETCH AT "CYCLE".

```

```

NEXTBYTE:  FLAGS←FLAGS+PCB15BIT;
           GOTO NB1X IF PCTR$14=1;
           GOTO NB01 IF PCTR$15=1;

```

```

* FIRST BYTE.

```

```

NB00:     TEMP←PCTR R1; TEMP←TEMP R1;
           TEMP←PBASE+TEMP;
           READTESTP( TEMP );
           PCTR←PCTR+1;
           TEMP←MEMDATAL PAUSE R8 ANDRETURN;

```

```

* SECOND BYTE.

```

```

NB01:     TEMP←MEMDATAL AND RMASK8;
           PCTR←PCTR+ZERO P1 ANDRETURN;

```

```

*

```

```

NB1X:     PCTR←PCTR+1;
           GOTO NB11 IF PCTR$15=0;           * 11+1=100 (BASE 2)

```

```

* THIRD BYTE.

```

```

NB10:     TEMP←MEMDATAR R8 ANDRETURN;

```

```

* FOURTH BYTE.

```

```

NB11:     TEMP←MEMDATAR AND RMASK8;
           RETURN;

```

```

*
*
* NEXT2()
* SETS:      TEMP = THE NEXT 16 BITS (TWO BYTES) OF CODE
* UPDATES:   FLAGS$PCLOW, PCTR.
* USES:      T8, T9=TEMP, MEMORY REGISTERS.
* NOTE:      NEXT2() ASSUMES MEMDATA L,R HAVEN'T BEEN DISTURBED SINCE
*            THE FETCH AT "CYCLE".

NEXT2:      FLAGS←FLAGS+2*PCB15BIT;
            GOTO NT1X IF PCTR$14=1;
            GOTO NT01 IF PCTR$15=1;
* FIRST & SECOND BYTES.
NT00:      TEMP←PCTR R1; TEMP←TEMP R1;
            TEMP←PBASE+TEMP;
            READTESTP( TEMP );
            PCTR←PCTR+2;
            TEMP←MEMDATAL PAUSE ANDRETURN;
* SECOND & THIRD BYTES.
NT01:      PCTR←PCTR+2;
            T8←MEMDATAL L8;
            TEMP←MEMDATAR R8;
            TEMP←T8 OR TEMP ANDRETURN;
*
NT1X:      PCTR←PCTR+2;
            GOTO NT11 IF PCTR$15=1;
* THIRD & FOURTH BYTES.
NT10:      TEMP←MEMDATAR ANDRETURN;
* FOURTH & FIRST BYTES.
NT11:      T8←MEMDATAR L8;
            TEMP←PCTR R1; TEMP←TEMP R1;
            TEMP←PBASE+TEMP;
            READTESTP( TEMP );
            TEMP←MEMDATAL PAUSE R8;
            TEMP←T8 OR TEMP ANDRETURN;

```

```
*
*
* TOP2()
* SETS:    AL,AR,BL,BR = THE TOP TWO ELEMENTS OF THE STACK.
* USES:    TEMP, MEMORY REGISTERS.
* ABNORMAL EXIT: STKUFERR IF LTOP<LBASE+3.
```

```
TOP2:    TEMP←LTOP-1;
          READL( TEMP );
          TEMP←LBASE+3;
          TEMP←TEMP EOR -1;
          ZERO←LTOP+TEMP P1;
          GOTO STKUFERR IF NOCARRY;          * LTOP<LBASE+3
          BL←MEMDATAL PAUSE;
          BR←MEMDATAR;
          *GOTO TOP;                          * CALL TOP & RETURN
```

```
*
* TOP()
* SETS:    AL,AR AND MEMDATA L,R = THE TOP ELEMENT OF THE STACK.
* USES:    TEMP, MEMORY REGISTERS.
* ABNORMAL EXIT: STKUFERR IF LTOP<LBASE+2.
```

```
TOP:     READLTOP;
          TEMP←LBASE+2;
          TEMP←TEMP EOR -1;
          ZERO←LTOP+TEMP P1;
          GOTO STKUFERR IF NOCARRY;          * LTOP<LBASE+2
          AL←MEMDATAL PAUSE;
          AR←MEMDATAR ANDRETURN;
```

```

*
*
* DECRFC()
* CALL:    READ( <DESCRIPTOR ADDRESS> );
*          CALL DECRFC;
*   OR:    FRP←<ARRAY ADDRESS>;
*          CALL DECRFCF;
* USES:    TEMP, FRP, FRSIZE, FRT1, FRT2, MEMORY REGISTERS
**
** FUNCTION DECRFC(X);
**   IF XSTYPE=ARRAYTYPE DO;
**     XSADDR.RFC←XSADDR.RFC-1;
**     IF XSADDR.RFC>0 DO;
**       FREE( XSADDR );
**     ENDIF;
**   ENDIF;
**   RETURN;
** ENDFUNC;

DECRFC:    TEMP←MEMDATAL EOR ARYTYPEB PAUSE;
           RETURNIF( TEMP$#0 );
           FRP←MEMDATAR;

DECRFCF:   TESTGRMW( FRP P1 );
           TEMP←MEMDATAL-1 PAUSE;           * TEMP←(REFERENCE COUNT)-1
           GOTO DRFREE IF TEMP=0;
           MEMDATAR←MEMDATAR;             * UPDATE ARRAY.RFC &
           MEMDATAL←TEMP ANDWRITE ANDRETURN;* RETURN

*
DRFREE:    MEMDATAL←0 ANDWRITE;           * FINISH RMW CYCLE
           GOTO FREE;                     * CALL FREE & RETURN

```

1 [in case FREE() traps]

Bug 4

* PROCESSOR MAIN LOOP

```
**
** WHILE TRUE DO;
**   FIRSTBYTE←NEXTBYTE();
**   IF FIRSTBYTE$OPCB0=0 DO;
**     IF FIRSTBYTE$OPCB1=0 DO;
**       SHORTLOCAL()
**       ... 00SSSSSS
**     ELSE DO;
**       IF FIRSTBYTE$OPCB2=0 DO;
**         LONGLOGLOB();
**         ... 010XLLLL LLLLLLLL
**       ELSE DO;
**         MEMREF()
**         ... 011XMMMM <LOC OR GLB SYL>
**       ENDIF;
**     ENDIF;
**   ELSE DO;
**     IF FIRSTBYTE$OPCB1=0 DO;
**       IF FIRSTBYTE$OPCB2=0 DO;
**         IF FIRSTBYTE$OPCB3=0 DO;
**           GENSCALAR();
**           ... 1000MMMM
**         ELSE DO;
**           INDEX();
**           ... 1001MMMM
**         ENDIF;
**       ELSE DO;
**         SHORTCONST();
**         ... 101NNNNN NNNNNNNN
**       ENDIF;
**     ELSE DO;
**       ETC();
**       ... 11SSSSSS [ ... ]
**     ENDIF;
**   ENDIF;
** ENDWHILE;
```

```

*
* MANY PLACES IN THE MICROCODE NEED TO STORE A VALUE ON THE TOP OF THE
* STACK BEFORE BRANCHING TO CYCLE. THEY CAN ACCOMPLISH THIS BY
* LOADING MEMDATA L,R AND BRANCHING TO WLCYCLE.
*

WLCYCLE:   WRITELTOP;

* HERE BEGINS THE INSTRUCTION CYCLE.
* THE CALL TO NEXTBYTE() IN THE MAIN LOOP IS CODED OPEN, AND OF COURSE
* MUST ALWAYS FETCH THE CURRENT INSTRUCTION WORD.

* SET EACH OF THE "TEMPORARY" FLAG BITS TO ZERO.
CYCLE:    CLEARFLAG( PCLOWB+OPFLAGSB );
* GENERATE A STEP TRAP IF THE STEP MODE BIT IS SET.
        GOTO STEPTRAP IF FLAGS$STEP=1;
* STOP THE PROCESSOR IF AN EXTERNAL COMMAND HAS ARRIVED.
CYCLE1:   TEMP←IOSTATUS;
        GOTO STOPPED IF TEMP$FFSAREAD=1;
* FETCH THE CURRENT INSTRUCTION WORD.
        TEMP←PCTR R1; TEMP←TEMP R1;
        TEMP←PBASE+TEMP;
        READ( PFIRST+TEMP );           * READP( TEMP )
* INCREMENT THE PCTR.
        PCTR←PCTR+1;
* TEST THAT TEMP CONTAINS A VALID PROCEDURE SEGMENT ADDRESS.
        ZERO←PNLEN+TEMP;
        GOTO PERR IF CARRY;
* NOW DISPATCH ON THE BOTTOM 2 BITS OF PCTR, WHICH HAS BEEN INCREMENTED.
        GOTO FB1X IF PCTR$14=1;           * BITS WERE 01 OR 10
*
        GOTO FB01 IF PCTR$15=1;           * BITS WERE 00
* FOURTH BYTE.
FB00:    WAITREAD;
        FIRSTBYTE←MEMDATAR AND RMASK8;
        GOTO DECODE;
* FIRST BYTE.
FB01:    FIRSTBYTE←MEMDATAL PAUSE R8;
        GOTO DECODE;
*
FB1X:    GOTO FB11 IF PCTR$15=1;           * BITS WERE 10
* SECOND BYTE.
FB10:    FIRSTBYTE←MEMDATAL AND RMASK8 PAUSE;
        GOTO DECODE;
* THIRD BYTE.
FB11:    WAITREAD;
        FIRSTBYTE←MEMDATAR R8;
        *GOTO DECODE;
DECODE:  GOTO OPC1XXX IF FIRSTBYTE$OPCB0=1;
        GOTO OPC01XX IF FIRSTBYTE$OPCB1=1;
        *GOTO SHORTLOCAL;

```

```

**
** OPCODE SHORTLOCAL;
**   STACKIT( LSEG, LBASE+FIRSTBYTE$SHORTOFFSET-64 )
** ENDOP;

SHORTLOCAL:ADDRESS+FIRSTBYTE OR MASK(0,OPCB1);* SIGN EXTEND
LOCAL:      ADDRESS+LBASE+ADDRESS;
           SEG+LSEG;

**
** FUNCTION STACKIT( SEG, ADDRESS );
**   LTOP+LTOP+1;
**   LSEG[LTOP]+SEG[ADDRESS];
**   INCRFC();
**   RETURN;
** ENDFUNC;

STACKIT:   CALL CEA;                * MEMDATA+SEG[ADDRESS]
           TESTNOTEQUAL( MEMDATAL, UNDFTYPEB, VALUEERROR );
           AL+MEMDATAL; AR+MEMDATAR;
           INCLTOP;
           MEMDATAL+AL; MEMDATAR+AR;
           WRITELTOP;                * LSEG[LTOP]+MEMDATA
           *GOTO INCRFC;              * CALL INCRFC & GOTO CYCLE

**
** FUNCTION INCRFC()
**   IF LSEG[LTOP]$TYPE=ARRAYTYPE DO;
**     LSEG[LTOP]$ADDR.RFC+LSEG[LTOP]$ADDR.RFC+1;
**   ENDIF;
**   RETURN;
** ENDFUNC;

INCRFC:   TESTEQUAL( AL, ARYTYPEB, CYCLE );
           TESTGRMW( AR P1 );        * MODIFY 2ND HEADER WORD
           WAITREAD;
           MEMDATAR+MEMDATAR;
           MEMDATAL+MEMDATAL+1 ANDWRITE;
           GOTO CYCLE;

```

```

*
OPC01XX:  GOTO MEMREF IF FIRSTBYTE$OPCB2=1;
**
**  OPCODE LONGLOGGLOB;
**    ADDRESS←(FIRSTBYTE LSH 8 OR NEXTBYTE())$LONGOFFSET;
**    IF FIRSTBYTE$OPCB3=0 DO;
**      STACKIT( LSEG, LBASE+ADDRESS-4096 );
**    ELSE DO;
**      STACKIT( GSEG, ADDRESS );
**    ENDIF;
**    RETURN;
**  ENDOP;

LONGLOGGLOB:CALL NEXTBYTE;                * TEMP←NEXTBYTE()
            ADDRESS←FIRSTBYTE L8;
            ADDRESS←ADDRESS OR TEMP;
            GOTO LONGGLOBAL IF FIRSTBYTE$OPCB3=1;
LONGLOCAL:  ADDRESS←ADDRESS OR MASK(0,3);  * SIGN EXTEND LOCAL ADDRESS
            GOTO LOCAL;
LONGGLOBAL:ADDRESS←ADDRESS AND RMASK12;
            SEG←GSEG;
            GOTO STACKIT;

```

```

**
** OPCODE MEMREF;
** DECLARE DCBYTE, DCOPC;
** DCBYTE←NEXTBYTE();
** DCOPC←DCBYTE$OPCODE;
** IF DCOPC<4 DO;
**     SEG←LSEG; ADDRESS←LBASE+DCBYTE$SHORTOFFSET-64;
** ELSE IF DCOPC=4 DO;
**     SEG←LSEG;
**     ADDRESS←LBASE+(DCBYTE LSH 8 OR NEXTBYTE())$LONGOFFSET-4096;
** ELSE IF DCOPC=5 DO;
**     SEG←GSEG; ADDRESS←DCBYTE LSH 8 OR NEXTBYTE())$LONGOFFSET;
** ELSE DO;
**     GOTO MRSYLERR;
** ENDIF;
** IF FIRSTBYTE$OPCB3=0 DO;
**     ASSIGNIDX(); * 0110 MMMM
** ELSE DO;
**     IF FIRSTBYTE$OPCB4=0 DO;
**         IF FIRSTBYTE$OPCB5=0 DO;
**             IF FIRSTBYTE$OPCB6=0 DO;
**                 ASSIGN(); * 0111 000X
**             ELSE DO;
**                 IF FIRSTBYTE$OPCB7=0 DO;
**                     REFERENCE(); * 0111 0010
**                 ELSE DO;
**                     TESTDEF(); * 0111 0011
**                 ENDIF;
**             ENDIF;
**         ELSE DO;
**             MRUNDERR(); * 0111 01XX
**         ENDIF;
**     ELSE DO;
**         MRUNDERR(); * 0111 1XXX
**     ENDIF;
** RETURN;
** ENDOP;

```

```

MEMREF:    CALL NEXTBYTE;                * TEMP←NEXTBYTE()
           GOTO MRSYLERR IF TEMPSOPCB0=1; * ADDR. OPCODES ARE 00XX,010X
           GOTO MRLONG   IF TEMPSOPCB1=1;
* SHORT LOCAL = 00XX
           ADDRESS←TEMP OR MASK(0,OPCB1); * SIGN EXTEND
MRLOCAL:   SEG←LSEG;
           ADDRESS←LBASE+ADDRESS;
           LINK←MRDCDED; GOTO CEA;        * CALL CEA & GOTO MRDCDED
*
MRLONG:    GOTO MRSYLERR IF TEMPSOPCB2=1;
           ADDRESS←TEMP L8;
           CALL NEXTBYTE;                * TEMP←NEXTBYTE()
           ADDRESS←ADDRESS OR TEMP;
           GOTO MRGLOBAL IF ADDRESS$3=1;
* LONG LOCAL = 0100
           ADDRESS←ADDRESS OR MASK(0,3); * SIGN EXTEND
           GOTO MRLOCAL;
* LONG GLOBAL = 0101
MRGLOBAL:  ADDRESS←ADDRESS AND RMASK12;
           SEG←GSEG;
           CALL CEA;                      * MEMDATA←GSEGI[ADDRESS]
*
MRDCDED:   GOTO MRSWITCH IF FIRSTBYTESOPCB3=1;
**
** SUBOP ASSIGNIDX;

ASSIGNIDX: FIRSTBYTE←FIRSTBYTE AND RMASK4; * EXPECTED RANK FOR INDEXER()
           CALL ACOPY;                    * SETS AR TO ARRAY ADDRESS
           T1←AR;                          * FOR INDEXER()
           T2←LTOP;                        * ADDRESS OF FIRST SUBSCRIPT
           CALL INDEXER;                   * SETS T3 TO ADDR OF ELEMENT
* FETCH THE VALUE TO BE ASSIGNED.
* IF AN "ASSIGN RAVELED" INSTRUCTION IS ADDED, IT SHOULD END WITH
* A TRANSFER TO HERE:
AIY:       READL( T2 );
           TEMP←LBASE+2; TEMP←TEMP EOR -1; ZERO←T2+TEMP P1;
           GOTO STKUFERR IF NOCARRY;
* DISPATCH ON TYPE OF THIS VALUE.
           TEMP←MEMDATAL AND SCATYPEB PAUSE;
           GOTO AISCA   IF TEMP#0;
           GOTO RANKERROR IF MEMDATAL$ARYTYPE=1;
           GOTO WHATERR;
* PERFORM THE ASSIGNMENT.
AISCA:     MEMDATAL←MEMDATAL; MEMDATAR←MEMDATAR;
           TESTGWRITE( T3 );
           LTOP←T2;                        * POP SUBSCRIPTS
           GOTO CYCLE;

```

```

*
*
* ACOPY() -- ENSURE THAT THE VALUE OF A GIVEN VARIABLE IS AN ARRAY
*           WITH REFERENCE COUNT OF ONE.
* CALL:    CALL CEA; * SETS T1=SEG, T2=ADDRESS, MEMDATA L,R
*           CALL ACOPY .
* RETURN:  AR CONTAINS THE ADDRESS OF THE ARRAY WHICH IS (NOW) THE
*           VALUE OF THE VARIABLE AT SEGIADDRESS].
* USES:    EVERYTHING, EXCEPT T0=FIRSTBYTE.

* CHECK THAT THE VALUE OF THE VARIABLE IS AN ARRAY.
ACOPY:     TEMP←MEMDATAL AND SCATYPEB;
           GOTO RANKERROR IF TEMP#0;
           GOTO WHATERR IF MEMDATALSARYTYPE=0;

* IF THE REFERENCE COUNT IS ONE, JUST RETURN A POINTER TO THIS ARRAY.
           AR←MEMDATAR;
           READTESTG( AR P1 ); * READ 2ND HEADER WORD
           TEMP←MEMDATAL EOR 1 PAUSE;
           RETURNIF( TEMP=0 );

* THE REFERENCE COUNT IS NOT ONE. COPY THE ARRAY.
           MKNELTS←MEMDATAR;
           READTESTG( AR ); * READ 1ST HEADER WORD
           AL←LINK;
           MKRANK←MEMDATAL AND RANKB PAUSE;
           CALL MAKE; * MKP←MAKE(MKRANK, MKNELTS)
           LINK←AL;

* WRITE A DESCRIPTOR FOR THE NEW ARRAY AT SEGIADDRESS].
           MEMDATAL←ARYTYPEB; MEMDATAR←MKP;
           GOTO ACGLO IF SEG$15=GSEG;
ACLOC:     WRITEL( ADDRESS ); GOTO ACY;
ACGLO:     WRITEG( ADDRESS );

* COPY THE SHAPE AND RAVEL WORDS FROM THE OLD ARRAY TO THE NEW ONE.
ACY:       NELTS←MKNELTS+MKRANK;
           BPTR←AR+SHAPEOFFSET;
           RPTR←MKP+SHAPEOFFSET;
ACLOOP:    READTESTG( BPTR ); BPTR←BPTR+1;
           TEMP←MEMDATAL PAUSE;
           MEMDATAL←TEMP;
           MEMDATAR←MEMDATAR;
           WRITEG( RPTR ); RPTR←RPTR+1;
           NELTS←NELTS-1;
           GOTO ACLOOP IF NELTS#0;

* RETURN POINTER FOR NEW ARRAY.
           FRP←AR;
           AR←MKP; * CALL DECRFCF(FRP) &
           GOTO DECRFCF; * RETURN

```

3.13

GET VARIABLE MEMDATAL \$UNIT = 1

```

*
MRSWITCH:  GOTO MRUNDERR IF FIRSTBYTE$OPCB4=1;
            GOTO MRUNDERR IF FIRSTBYTE$OPCB5=1;
            GOTO MRS001X  IF FIRSTBYTE$OPCB6=1;

**
** SUBOP ASSIGN;
**   DECRFC( SEG[ADDRESS] ) IF SEG[ADDRESS]$TYPE=ARRAYTYPE;
**   SEG[ADDRESS]+LSEG[LTOP];
**   IF FIRSTBYTE$OPCB7=0 DO; * ASSIGN PROPER
**     INCRFC();
**   ELSE DO; * ASSIGN, NO RESULT
**     LTOP+LTOP-1;
**   ENDIF;
**   RETURN;

ASSIGN:    CALL DECRFC;
            CALL TOP; * AL,AR + LSEG[LTOP]
            MEMDATAL+AL; MEMDATAR+AR;
            GOTO ASGLOBAL IF SEG$15=GSEG;

ASLOCAL:  WRITEL( ADDRESS );
            GOTO ASY;

ASGLOBAL: WRITEG( ADDRESS );

ASY:      GOTO INCRFC IF FIRSTBYTE$OPCB7=0; * ASSIGN PROPER ELSE NO RES
            LTOP+LTOP-1; * DECREMENT STACK POINTER
            GOTO CYCLE;

*
MRS001X:  GOTO TESTDEF IF FIRSTBYTE$OPCB7=1;
*
* MAKE A REFERENCE DESCRIPTOR FOR @SEG[ADDRESS].

REFERENCE: INCLTOP;
            TEMP+SEG AND RMASK1; * EXTRACT SEG BIT
            MEMDATAL+TEMP OR REFTYPEB;
            MEMDATAR+ADDRESS;
            GOTO WLCYCLE; * WRITELTOP & GOTO CYCLE

*
* TEST WHETHER THE VALUE OF SEG[ADDRESS] IS DEFINED OR UNDEFINED.

TESTDEF:  INCLTOP;
            MEMDATAL+INTTYPEB;
            MEMDATAR+1;
* RECALL MEMDATAL IS A DOUBLE-SIDED REGISTER.
            TESTEQUAL( MEMDATAL, UNDFTYPEB, WLCYCLE );
            MEMDATAR+0; * UNDEFINED
            GOTO WLCYCLE;

```



```

*
*
* CEA() -- COMPUTE EFFECTIVE ADDRESS.
* CALL:   ADDRESS←SEGMENT-RELATIVE ADDRESS OF A VARIABLE;
*         SEG←LSEG IF LOCAL ELSE GSEG IF GLOBAL;
*         CALL CEA;
* SETS:   MEMDATA TO THE CONTENTS OF THE ADDRESSED VARIABLE;
*         ADDRESS, SEG$15 TO THE EFFECTIVE ADDRESS, SEGMENT;
*         SEG$REFTYPE TO THE NUMBER OF INDIRECTIONS REQUIRED (<2).
* USES:   TEMP, MEMORY REGISTERS.

CEA:      GOTO CEAG IF SEG$15=GSEG;
CEAL:     READTESTL( ADDRESS ); GOTO CEAY;
CEAG:     READTESTG( ADDRESS );
CEAY:     TEMP←MEMDATAL EOR REFTYPEB PAUSE;
          RETURNIF( TEMP$L#0 );           * "DIRECT" TYPE
          GOTO REFERR IF SEG$REFTYPE#0;   * REF TO A REF (SEE BELOW)
          ADDRESS←MEMDATAR;               *
          SEG←MEMDATAL;                   * SEG$REFTYPE←1 (SEE ABOVE)
          GOTO CEA;

```

```

*
* GENSCALAR = 1000MMMM.

* M IS THE OPERATOR NUMBER, AN INDEX IN GSTAB. FIRST COME THE
* MONADIC OPERATORS, THEN THE DYADIC OPERATORS. NON-SCALAR
* OPERANDS CAUSE ELEMENT-BY-ELEMENT REPETITION OF THE GIVEN SCALAR
* OPERATION.

DC GSTAB ← *;
* MONADIC OPERATORS:
DC IDOPNO ← *-GSTAB;
    DATA IDENTITY;
    DATA NEGATIVE;
    DATA FLOOR;
    DATA CEILING;
    DATA MAGNITUDE;
    DATA NOT;
* THE FOLLOWING MONADIC OPERATORS ACCEPT CHARACTER OR NUMERIC
* OPERANDS.
DC GSCHMONOPN ← *-GSTAB;
    DATA TESTNUM;
    DATA CONVERT;
* DYADIC OPERATORS:
DC GSDYOPN ← *-GSTAB;
    DATA SUM;
    DATA DIFFERENCE;
    DATA PRODUCT;
    DATA QUOTIENT;
    DATA BAND;
    DATA BOR;
    DATA LESS;
* THE NEXT OPERATOR ("EQUAL") IS THE ONLY DYADIC ONE ACCEPTING
* CHARACTER OPERANDS, ALTHOUGH MORE COULD BE ADDED FOLLOWING IT.
DC GSCHDYOPN ← *-GSTAB;
    DATA EQUAL;
* THERE ARE NO MORE SCALAR OPERATORS.
DC GSUNDOPN ← *-GSTAB;
* IMPORTANT:
* THE VALUE "GSUNDOPN" MUST BE NOT GREATER THAN 16.
*
DC IGOPNO ← *-GSTAB;
    DATA IGELT;
DC GOUNDOPN ← *-GSTAB;
    DATA GOUNDERR;
*
OPC1XXX:   GOTO ETC           IF FIRSTBYTESOPCB1=1;   * OP CODE 11--
           GOTO SHORTCONST IF FIRSTBYTESOPCB2=1;   * OP CODE 101-
           GOTO INDEX       IF FIRSTBYTESOPCB3=1;   * OP CODE 1001
           *GOTO GENSCALAR;   * OP CODE 1000

```

```

*
* BEGIN GENSCALAR.

* OPNO=FIRSTBYTE GETS OPERATOR NUMBER.
GENSCALAR: OPNO←FIRSTBYTE AND RMASK4;
* THE OPERAND FLAG BITS WILL BE SET FOR GENOP.
* OPERATOR IS MONADIC IF ITS NUMBER IS LESS THAN GSDYOPN.
    TESTLESS( OPNO, GSDYOPN, GSTESTDY );
* MONADIC OPERATOR.  FETCH ITS OPERAND.
    SETFLAG( MONOPB );
    TESTLESS( OPNO, GSCHMONOPN, GSTOP );
    SETFLAG( NUMOPB );
GSTOP:    LINK←GSDISPA;                * CALL TOP &
    GOTO TOP;                          * GOTO GSDISPA
* OPERATOR IS DYADIC IF ITS NUMBER IS LESS THAN GSUNDOPN.
GSTESTDY: TESTLESS( OPNO, GSUNDOPN, GSUNDERR );
* DYADIC OPERATOR.  FETCH ITS OPERANDS.
    TESTLESS( OPNO, GSCHDYOPN, GSTOP2 );
    SETFLAG( NUMOPB );
GSTOP2:   CALL TOP2;
* DISPATCH ON TYPE OF FIRST (OR ONLY) OPERAND.
GSDISPA:  TEMP←AL AND NUMTYPEB;
    GOTO GSSCAXXX IF TEMP#0;
    GOTO GSARRXXX IF AL$ARYTYPE=1;
    GOTO WHATERR IF AL$CHTYPE=0;
* FIRST OPERAND IS A SCALAR CHARACTER.
    GOTO TYPEERROR IF FLAGSS$NUMOP=1;
* FIRST OPERAND IS SCALAR.  DISPATCH ON TYPE OF SECOND OPERAND (IF
* ANY).
GSSCAXXX: GOTO GSSCASCA IF FLAGSS$MONOP=1; * NO SECOND OPERAND
    TEMP←BL AND NUMTYPEB;
    GOTO GSSCASCA IF TEMP#0;
    GOTO GSSCAARR IF BL$ARYTYPE=1;
    GOTO WHATERR IF BL$CHTYPE=0;
* SECOND OPERAND IS A SCALAR CHARACTER.
    GOTO TYPEERROR IF FLAGSS$NUMOP=1;
* BOTH OPERANDS ARE SCALAR.
GSSCASCA: SETFLAG( ASCALARB+BSCALARB );
    LINK←OPNO+GSTAB;
    LINK←ZERO LOAD 0;                * LINK← OP. ROUTINE ADDRESS
    OPRET←GOOPRET;
    GOTO 0 IX;
* OPERANDS ARE SCALAR, ARRAY.
GSSCAARR: SETFLAG( ASCALARB );
    APTR←BR;                          * APTR←ARRAY ADDRESS
GSONEARR: READTESTG( APTR );
    MKRANK←MEMDATAL AND RANKB PAUSE; * MKRANK←RANK OF ARRAY
    GOTO GSALLOC;
*
* FIRST OPERAND IS ARRAY.  DISPATCH ON TYPE OF SECOND OPERAND (IF ANY).
GSARRXXX: GOTO GSARRSCA IF FLAGSS$MONOP=1;
    TEMP←BL AND SCATYPEB;
    GOTO GSARRSCA IF TEMP#0;

```

```

        GOTO GSARRARR IF BLSARYTYPE=1;
        GOTO WHATERR;
* OPERANDS ARE ARRAY, SCALAR.
GSARRSCA: SETFLAG( BSCALARB );
        APTR←AR;
        GOTO GSONEARR;
* BOTH OPERANDS ARE ARRAYS.
GSARRARR: READTESTG( AR );
        MKRANK←MEMDATAL AND RANKB PAUSE; * MKRANK←A.RANK
        READTESTG( BR );
* COMPARE RANKS OF A AND B.
GSCHKRANK: TEMP←MEMDATAL AND RANKB PAUSE; * TEMP←B.RANK
        TEMP←TEMP EOR -1;
        T1←MKRANK+TEMP P1; * T1←A.RANK-B.RANK
        GOTO GSDIFRANK IF T1#0;
* A.RANK=B.RANK=N, SAY. IS A.SHAPE[I]=B.SHAPE[I] FOR 1<=I<=N ?
        T7←MKRANK; * T7←N
        T8←AR+SHAPEOFFSET; * T8←@A.SHAPE[I]
        T9←BR+SHAPEOFFSET; * T9←@B.SHAPE[I]
GSCHKSHP: READTESTG( T8 ); * READ A.SHAPE[I]
        T8←T8+1;
        WAITREAD;
        T6←MEMDATAR; * NOTE MEMDATAL=INTTYPEB
        READTESTG( T9 ); * READ B.SHAPE[I]
        T9←T9+1;
        WAITREAD;
        T6←MEMDATAR EOR T6;
        GOTO GSDIFLEN IF T6#0; * A.SHAPE[I]#B.SHAPE[I]
        T7←T7-1;
        GOTO GSCHKSHP IF T7#0;
* SHAPES ARE EQUAL.
        APTR←AR;
* CREATE A NEW ARRAY BLOCK (TO HOLD RESULT) WITH THE SAME SHAPE AS
* THE ARRAY POINTED TO BY APTR.
GSALLOC: READTESTG( APTR P1 ); * READ 2ND HEADER WORD
        WAITREAD;
        MKNELTS←MEMDATAR; * NUMBER OF ELEMENTS
* THE ROUTINE MAKE() ALLOCATES AN ARRAY BLOCK, INITIALIZES ITS TWO
* HEADER WORDS, AND RETURNS A POINTER TO THE NEW BLOCK IN MKP AND IN
* THE GLOBAL CELL BLOCKPTR.
        CALL MAKE; * MKP←MAKE(MKRANK, MKNELTS)
        SETFLAG( RARRAYB );
* FILL IN THE SHAPE WORDS OF THE NEW ARRAY.
        COUNTER←MKRANK; * RANK
        T8←APTR+SHAPEOFFSET; * T8←@A.SHAPE[I]
        TEMP←MKRANK+SHAPEOFFSET;
        APTR←AR+TEMP;
        BPTR←BR+TEMP;
        RPTR←MKP+SHAPEOFFSET; * RPTR←@RESULT.SHAPE[I]
        LINK←GSCOPSHP;
GSCOPSHP: READTESTG( T8 );
        TEMP←MEMDATAL PAUSE;
        MEMDATAL←TEMP;
        MEMDATAR←MEMDATAR;

```

```

WRITEG( RPTR );
T8←T8+1;
RPTR←RPTR+ZERO P1 D J;
* NOW OPNO, NELTS=MKNELTS, APTR, BPTR, AND RPTR ARE INITIALIZED
* FOR GENOP. APPLY THE OPERATOR ELEMENT-BY-ELEMENT TO ITS OPERAND(S).
GOTO GENOP;
*
* OPERANDS ARE ARRAYS WITH DIFFERENT RANKS.
GSDIFRANK: LINK←RANKERROR;
GOTO GSCHKNELT;
* OPERANDS ARE RANK-N ARRAYS (N>0) WITH DIFFERENT LENGTHS.
GSDIFLEN: LINK←LENGTHERROR;
* CHECK FOR A 1-ELEMENT OPERAND. RECALL T1=A.RANK-B.RANK.
GSCHKNELT: READTESTG( AR P1 ); * READ 2ND HEADER WORD OF A
WAITREAD;
T2←MEMDATAR-1; * T2←A.NELTS-1
READTESTG( BR P1 ); * READ 2ND HEADER WORD OF B
WAITREAD;
T3←MEMDATAR-1; * T3←B.NELTS-1
GOTO GSAIS1ELT IF T2=0; * A.NELTS=1
GOTO GSPEELB IF T3=0; * ONLY B HAS 1 ELEMENT
GOTO 0 IX; * RANKERROR OR LENGTHERROR
* A.NELTS=1. IS B.NELTS=1 ALSO ?
GSAIS1ELT: GOTO GSPEELA IF T3#0; * ONLY A HAS 1 ELEMENT
* BOTH A AND B ARE 1-ELEMENT ARRAYS. TREAT THE ONE WITH SMALLER RANK
* AS SCALAR.
GOTO GSPEELB IF T1$0=0; * B.RANK<=A.RANK
* "PEEL" A (SET A←(RAVEL A)[IORG]) AND RESTART GENSCALAR.
GSPEELA: FRP←AR; * ARRAY TO PEEL
T1←LTOP; * PLACE FOR RESULT
LINK←GENSCALAR; * CALL PEEL(FRP, T1) &
GOTO PEEL; * GOTO GENSCALAR
* "PEEL" B AND RESTART GENSCALAR. (PEEL() SAVES FIRSTBYTE.)
GSPEELB: FRP←BR;
T1←LTOP-1;
LINK←GENSCALAR; * CALL PEEL(FRP, T1) &
GOTO PEEL; * GOTO GENSCALAR

```

```

*
* GENOP -- GENERALIZED OPERATOR APPLICATION.
*
* DRIVEN BY THE FOLLOWING BITS IN THE REGISTER "FLAGS":
*   NUMOP=1   IFF THE OPERATOR WILL NOT ACCEPT CHARACTER OPERANDS;
*   MONOP=1   IFF THE OPERATOR IS MONADIC;
*   ASCALAR=1 IFF THE FIRST (OR ONLY) OPERAND IS SCALAR;
*   BSCALAR=1 IFF THE SECOND OPERAND (IF ANY) IS SCALAR;
*   RARRAY=1  IFF THE RESULT IS AN ARRAY.
*
* THE FOLLOWING REGISTERS SHOULD ALSO BE INITIALIZED BEFORE ENTERING
* GENOP.
*   NELTS = NUMBER OF ITERATIONS;
*   APTR  = CURSOR IN FIRST OPERAND, IF FLAGSSASCALAR=0;
*   BPTR  = CURSOR IN SECOND OPERAND, IF FLAGSSBSCALAR=0;
*   RPTR  = CURSOR IN RESULT, IF FLAGSSRSCALAR=0;
*   OPNO  = INDEX OF OPERATOR TO BE ITERATIVELY APPLIED.
GENOP:   GOTO GENOPFIN IF NELTS=0;
* CHECK FOR EXTERNAL COMMAND.
        TEMP←IOSTATUS;
        GOTO GOINTRPT IF TEMP$FFSAREAD=1;
* LOAD FIRST OPERAND FROM STACK OR ARRAY INTO AL,AR.
        GOTO GOARRA IF FLAGSSASCALAR=0;
GOSCAA:  READLTOP; GOTO GOWAITA;
GOARRA:  READTESTG( APTR );
GOWAITA: NELTS←NELTS-1;
        OPRET←GOOPRET;
        LINK←OPNO+GSTAB; LINK←ZERO LOAD 0;
        AL←MEMDATAL PAUSE; AR←MEMDATAR;
        GOTO GOA IF FLAGSSNUMOP=0;
* OPERATOR REQUIRES NUMERIC OPERANDS -- CHECK A.
        TEMP←AL AND NUMTYPEB;
        GOTO TYPEERROR IF TEMP=0;
* APPLY THE OPERATOR NOW IF IT IS MONADIC.
GOA:     GOTO 0 IF FLAGSSMONOP=1 IX;
* LOAD SECOND OPERAND FOR DYADIC OPERATOR INTO BL,BR.
        GOTO GOARRB IF FLAGSSBSCALAR=0;
GOSCAB:  TEMP←LTOP-1; READL( TEMP ); GOTO GOWAITB;
GOARRB:  READTESTG( BPTR );
GOWAITB: BL←MEMDATAL PAUSE; BR←MEMDATAR;
        GOTO 0 IF FLAGSSNUMOP=0 IX;
* OPERATOR REQUIRES NUMERIC OPERANDS -- CHECK B.
        TEMP←BL AND NUMTYPEB;
        GOTO 0 IF TEMP#0 IX;
        GOTO TYPEERROR;
* THE OPERATOR ROUTINE RETURNS HERE.
GOOPRET: MEMDATAL←AL; MEMDATAR←AR;
        GOTO GENOPFIN IF FLAGSSRARRAY=0; * RESULT IS SCALAR
        TESTGWRITE( RPTR );
        APTR←APTR+1; BPTR←BPTR+1; RPTR←RPTR+1;
        GOTO GENOP;

```

*
* A COMMAND WAS RECEIVED IN THE MIDST OF GENERALIZED OPERATOR
* APPLICATION. SAVE THE REGISTERS NELTS, APTR, BPTR, RPTR, AND OPNO
* IN THE STATE AREA OF THE DATA SEGMENT; SET FLAGSSINTRPT TO 1, AND
* TRANSFER TO STOPPED. IF THIS PROCESS IS LATER RESTARTED, CONTROL WILL
* RETURN TO GOCONT BELOW.

GOINTRPT: MEMDATAL←NELTS; MEMDATAR←OPNO;
WRITEG(SAVENO);

*
MEMDATAL←APTR; MEMDATAR←BPTR;
WRITEG(SAVEAB);

*
MEMDATAL←RPTR; *MEMDATAR←ANYTHING;
WRITEG(SAVERX);

*
SETFLAG(INTRPTB);
GOTO STOPPED;

*
* CONTINUE PROCESS INTERRUPTED IN GENOP. CLEAR FLAGSSINTRPT AND
* RESTORE THE REGISTERS, THEN CHECK THAT OPNO<GOUNDOPN.

GOCONT: CLEARFLAG(INTRPTB);

*
READG(SAVENO);
NELTS←MEMDATAL PAUSE; OPNO←MEMDATAR;

*
READG(SAVEAB);
APTR←MEMDATAL PAUSE; BPTR←MEMDATAR;

*
READG(SAVERX);
RPTR←MEMDATAL PAUSE;

*
TESTLESS(OPNO, GOUNDOPN, GOUNDERR);
GOTO GENOP;

*

*

* GENOPFIN MUST NOT BE CALLED WITH (FLAGSS)RARRAY=0 UNLESS ASCALAR=1
* AND BSCALAR=1.

GENOPFIN: GOTO GOFCHKB IF FLAGSSASCALAR=1;

READLTOP;

CALL DECRFC;

GOFCHKB: GOTO GOFSTK IF FLAGSSMONOP=1;

LTOP←LTOP-1;

GOTO GOFSTK IF FLAGSSBSCALAR=1;

READLTOP;

CALL DECRFC;

GOFSTK: GOTO WLCYCLE IF FLAGSSRARRAY=0;

READG(BLOCKPTR);

MEMDATAR←MEMDATAL PAUSE; MEMDATAL←ARYTYPEB;

GOTO WLCYCLE;


```
*****
* MONADIC SCALAR OPERATOR ROUTINES *
*****
```

```
*****
* FLOOR *
*****
```

```
* (FLOOR A)=A IF A IS AN INTEGER.
FLOOR:   GOTO IDENTITY IF AL$FLOTYPE=0;
*
```

```
* (FLOATING-POINT) FLOOR.
* SETS:   AL,AR TO THE FLOOR OF THE FLOATING-POINT NUMBER IN AL,AR.
*         RESULT IS OF INTEGER TYPE WHENEVER POSSIBLE (I.E.
*         WHENEVER EXPONENT OF OPERAND IS <= 22).
* USES:   COUNTER, LINK, AEXP, ASIGN, FT1.
```

```
* UNPACK A.
```

```
* AEXP GETS (A$EXP-EXPBIAS).
```

```
FFLO:   AEXP←AR AND EXPB; AEXP←AEXP-EXPBIAS;
```

```
* IF AEXP>22 THEN A IS ALREADY "INTEGRAL"; JUST RETURN IT.
```

```
FT1←AEXP-(22+1);
```

```
GOTO IDENTITY IF FT1$0=0; * 23<=AEXP
```

```
* ASIGN$SIGN GETS ASIGN.
```

```
ASIGN←AL AND SIGNB;
```

```
* AL,AR GET ASCOEF.
```

```
AL←AL AND SIGNC; AR←AR AND EXPC;
```

```
* IF AEXP<0 THEN RETURN 0 (IF A>0) OR -1 (IF A<0).
```

```
GOTO FFSHIFT IF AEXP$0=0; * 0<=AEXP
```

```
LINK←OPRET;
```

```
AL←ASIGN OR INTTYPEB;
```

```
ZERO←ASIGN L1 SO;
```

```
* NOTE SIGN = BIT 0
```

```
AR←ZERO SI L1 ANDRETURN;
```

```
* SHIFT AL,AR RIGHT (22-AEXP) PLACES THROUGH A STICKY BIT.
```

```
FFSHIFT: COUNTER←FT1 EOR -1; * COUNTER←22-AEXP
```

```
GOTO FFPACK IF COUNTER$R=0; * AEXP=22
```

```
LINK←FFLOOP;
```

```
FT1←0;
```

```
* FT1$COEFB23 IS STICKY BIT
```

```
FFLOOP: FT1←FT1 OR AR;
```

```
AL←AL R1 SO; AR←AR SI R1 D J;
```

```
* IF ASIGN#0 THEN ADD STICKY BIT TO AL,AR (ROUND UP).
```

```
GOTO FFPACK IF ASIGN=0;
```

```
FT1←FT1 AND COEFB23BIT;
```

```
AR←AR+FT1; AL←AL+ZERO CI;
```

```
* RETURN INTEGER WITH SIGN FROM ASIGN, MAGNITUDE FROM AL,AR.
```

```
FFPACK: FT1←AL L8;
```

```
* SHIFT AL,AR
```

```
AL←AL R8;
```

```
* RIGHT
```

```
AR←AR R8; AR←FT1 OR AR;
```

```
* 8 PLACES
```

```
GOTO IPACK;
```

* CEILING *

* (CEILING A)=A IF A IS AN INTEGER.
CEILING: GOTO IDENTITY IF ALSFLOTYPE=0;
* (CEILING A)=-(FLOOR -A) IN ANY CASE.
AL←AL EOR SIGNB;

* OK SINCE A#0

*

FT2←OPRET;
OPRET←CEILNEG;
GOTO FFLO;

* OPCALL(FFLO)
* (SAVES FT2)

*

CEILNEG: OPRET←FT2;
*GOTO NEGATIVE;

* OPCALL(NEGATIVE) &
* RETURN

* NEGATIVE *

NEGATIVE: GOTO NEGANZ IF ALSFLOTYPE#0;
GOTO NEGANZ IF ALSR#0;
GOTO IDENTITY IF AR=0;

* (-0)=0

* A#0. TOGGLE ASSIGN.
NEGANZ: AL←AL EOR SIGNB;
GOTO IDENTITY;

***** *****
* MAGNITUDE * , * IDENTITY *
***** *****

MAGNITUDE: AL←AL AND SIGNC;
IDENTITY: LINK←OPRET;
RETURN;

* CLEAR SIGN BIT

* NOT *

NOT: CALL TESTLOGA;
AR←AR EOR 1;
GOTO IDENTITY;

* ENSURE A IS INTEGER 0 OR 1
* 0->1, 1->0

* TESTNUM *

* RETURN 1 FOR NUMERIC OPERAND, 0 FOR CHARACTER OPERAND.
TESTNUM: GOTO OPRETURN1 IF ALSFLOTYPE=1;
GOTO OPRETURN1 IF ALSINTTYPE=1;
GOTO OPRETURN0 IF ALSCHTYPE=1;
GOTO WHATERR;

* CONVERT *

* CONVERT CHARACTER TO INTEGER, INTEGER TO CHARACTER.
CONVERT: GOTO CNVAFLO IF ALSFLOTYPE=1;
GOTO CNVAINT IF ALSINTTYPE=1;
GOTO CNVACH IF ALSCHTYPE=1;
GOTO WHATERR;

* A IS FLOATING-POINT.

CNVAFLO: CALL FIXA;

* A IS (NOW) AN INTEGER.

CNVAINT: GOTO DOMAINERROR IF ALSSIGN#0;
GOTO DOMAINERROR IF ALSR#0;
GOTO DOMAINERROR IF ARSL#0;
AL←CHTYPEB;
GOTO IDENTITY;

* A IS A CHARACTER.

CNVACH: AL←INTTYPEB;
GOTO IDENTITY;

```
*****
* DYADIC SCALAR OPERATOR ROUTINES *
*****
```

```
*****
* DIFFERENCE *
*****
```

```
* DIFFERENCE(A,0)=A.
DIFFERENCE:GOTO DIFBNZ   IF BL$FLOTYPE=1;
             GOTO DIFBNZ   IF BR#0;
             GOTO IDENTITY IF BL$R=0;           * B=0
* DIFFERENCE(A,B)=SUM(A,-B).
DIFBNZ:     BL←BL EOR SIGNB;                   * B←-B
             *GOTO SUM;
```

```
*****
* SUM *
*****
```

```
* PERFORM INTEGER ADDITION IF BOTH OPERANDS ARE INTEGERS.
SUM:       FT1←AL OR BL;
           GOTO SUMFLOAT IF FT1$FLOTYPE=1;
* INTEGER+INTEGER.
* UNPACK A AND B.
*   ASIGN GETS $ASSIGN; FT1 GETS (ASSIGN EOR B$SIGN).
           ASIGN←AL AND SIGNB;
           FT1←AL EOR BL;
*   AL,AR AND BL,BR GET $MAG AND B$MAG RESPECTIVELY.
           AL←AL AND RMASK8;
           BL←BL AND RMASK8;
* THE TREATMENT OF LIKE AND UNLIKE SIGNS DIVERGES HERE.
           GOTO IASUB IF FT1$SIGN=1;
* ADD MAGNITUDES.
           AR←AR+BR; AL←AL+BL CI;
           GOTO IOF IF AL$8=1;                   * INTEGER OVERFLOW
* PACK ASIGN,AL,AR INTO INTEGER FORMAT AND RETURN.
IPACK:    LINK←OPRET;
           AL←AL OR INTTYPEB;
           AL←ASIGN OR AL ANDRETURN;
* SUBTRACT MAGNITUDES.
IASUB:    BR←BR EOR -1; AR←AR+BR P1;
           BL←BL EOR -1; AL←AL+BL CI;
           GOTO IASUBNZ IF AR#0;
           GOTO OPRETURN0 IF AL=0;               * RESULT IS ZERO
IASUBNZ:  GOTO IPACK IF AL$0=0;
* RECOMPLEMENT -- B HAD THE LARGER MAGNITUDE.
           AR←AR EOR -1; AR←AR+ZERO P1;
           AL←AL EOR -1; AL←AL+ZERO CI;
           ASIGN←ASIGN EOR SIGNB;
           GOTO IPACK;
*
* AT LEAST ONE OF A,B IS FLOATING-POINT.
```

```

SUMFLOAT:  GOTO SUMCHKB IF ALS$FLOTYPE=1;  * A IS FLOATING-POINT
* A IS AN INTEGER.
          CALLFC( FLOATA, OPRETURNB );      * FLOATA() FAILS IF A=0
SUMCHKB:   GOTO FADD IF BL$FLOTYPE=1;      * B IS FLOATING POINT
* B IS AN INTEGER.
          CALLFC( FLOATB, IDENTITY );      * FLOATB() FAILS IF B=0
          *GOTO FADD;

*
* FLOATING-POINT ADDITION.
* USES:    COUNTER, LINK, BL, BR, ASIGN, AEXP, FT1, FT2.

* UNPACK A AND B.
* AEXP GETS A$EXP (STILL BIASED) AND FT1 GETS (A$EXP-B$EXP).
FADD:     AEXP←AR AND EXPB;
          FT1←BR AND EXPB; FT1←FT1 EOR -1; FT1←AEXP+FT1 P1;
* IF A$EXP<B$EXP, INTERCHANGE A,B AND RESTART.
          GOTO FAUNPACK2 IF FT1$0=0;
          FT1←AL; AL←BL; BL←FT1;
          FT1←AR; AR←BR; BR←FT1;
          GOTO FADD;
* ASIGN (TENTATIVE RESULT SIGN) GETS A$SIGN AND FT2$SIGN GETS
* (A$SIGN EOR B$SIGN).
FAUNPACK2: ASIGN←AL AND SIGNB;
           FT2←AL EOR BL;
* AL,AR AND BL,BR (BITS 1 THROUGH 23) GET A$COEF AND B$COEF
* RESPECTIVELY.
           AL←AL AND SIGNC; AR←AR AND EXPC;
           BL←BL AND SIGNC; BR←BR AND EXPC;
* ALIGN THE RADIX POINTS OF THE COEFFICIENTS.
           GOTO FAADDSUB IF FT1=0;          * POINTS ALREADY ALIGNED
* SHIFT BL,BR (FT1) PLACES RIGHT THROUGH ROUND1, ROUND2 AND STICKY BITS.
           COUNTER←FT1;                    * FT1<256
           LINK←FASHIFT1;
FASHIFT1:  GOTO FASHIFT2 IF BR$STICKY=0;
           BR←BR OR STICKYB*2;            * STICKY BIT STICKS
FASHIFT2:  BL←BL R1 S0; BR←BR SI R1 D J;
* TREATMENT OF LIKE AND UNLIKE SIGNS DIVERGES HERE.
FAADDSUB:  GOTO FASUB IF FT2$SIGN=1;
* ADD THE COEFFICIENTS OF A AND B, THEN TEST FOR OVERFLOW (NO OTHER
* NORMALIZING IS NECESSARY).
           AR←AR+BR; AL←AL+BL CI;
           GOTO FROUND IF NOOVERFLOW;
* COEFFICIENT OVERFLOW: SHIFT AL,AR RIGHT 1 PLACE (THROUGH ROUND AND
* STICKY BITS) AND INCREMENT AEXP.
FCOEFOF:  AEXP←AEXP+1;
           FT1←AR AND STICKYB;
           AL←AL R1 S0; AR←AR SI R1;
           AR←AR OR FT1;
           GOTO FROUND;
* ADD A$COEF AND THE COMPLEMENT OF B$COEF. A NEGATIVE RESULT REQUIRES
* RECOMPLEMENTATION; A SUM OF ZERO SHOULD BE DETECTED BEFORE
* NORMALIZING.
FASUB:    BR←BR EOR -1; AR←AR+BR P1;
           BL←BL EOR -1; AL←AL+BL CI;

```

```

FT1←AL OR AR;
GOTO OPRETURN0 IF FT1=0;           * RESULT IS ZERO
GOTO FASNORM IF AL$0=0;           * NO RECOMPLEMENTATION NEEDED
AR←AR EOR -1; AR←AR+ZERO P1;
AL←AL EOR -1; AL←AL+ZERO CI;
ASIGN←ASIGN EOR SIGNB;
* SHIFT AL,AR LEFT JUST ENOUGH PLACES SO AL$COEFB1=1, AND DECREMENT
* AEXP BY THE NUMBER OF SHIFTS REQUIRED.
FASNORM:  GOTO FROUND IF AL$COEFB1=1;
FASNORML: AEXP←AEXP-1;
          AR←AR L1 S0; AL←AL SI L1;
          GOTO FASNORML IF AL$COEFB1=0;
* ROUND THE COEFFICIENT IN AL,AR TO 23 BITS (COEFB1 THROUGH COEFB23)
* USING ROUND1, ROUND2 AND STICKY:
*   ROUND DOWN (TRUNCATE) IF ROUND1=0;
*   ROUND TO NEAREST EVEN IF ROUND1=1, ROUND2=STICKY=0;
*   ROUND UP (ADD 1) IF ROUND1=1, ROUND2=1 AND/OR STICKY=1.
FROUND:   GOTO FPACK   IF AR$ROUND1=0;
*
          FT1←AR AND (ROUND2B+STICKYB);
          GOTO FROUNDUP IF FT1#0;
*
          GOTO FPACK   IF AR$COEFB23=0;
* ROUND UP BY ADDING 1 IN COEFB23 AND CHECKING FOR COEFFICIENT OVERFLOW.
FROUNDUP: AR←AR+COEFB23BIT; AL←AL+ZERO CI;
          GOTO FPACK IF NOOVERFLOW;
* COEFFICIENT ROUNDING OVERFLOW. NOTE AR=0 AND AL=1000000B.
          AL←0400000B;
          AEXP←AEXP+1;
* CHECK FOR EXPONENT (UNDER/OVER)FLOW AND PACK ASIGN,AL,AR,AEXP INTO
* AL,AR.
FPACK:   GOTO FEXPUF IF AEXPS0#0;           * EXPONENT UNDERFLOW;
          FT1←AEXP AND EXPC;
          GOTO FEXPOF IF FT1#0;           * EXPONENT OVERFLOW
          LINK←OPRET;
          AR←AR AND EXPC;
          AR←AR OR AEXP;
          AL←ASIGN OR AL ANDRETURN;

```

* PRODUCT *

* PERFORM INTEGER MULTIPLICATION IF BOTH OPERANDS ARE INTEGERS.

PRODUCT: FT1←AL OR BL;
GOTO PRODFLOAT IF FT1\$FLOTYPE=1;

* INTEGER*INTEGER.

* UNPACK A AND B.

* ASIGN GETS (ASIGN EOR BSSIGN).

ASIGN←AL EOR BL; ASIGN←ASIGN AND SIGNB;

* AL,AR HOLD ASMAG (THE MULTIPLIER).

* BL,BR GET BSMAG (THE MULTIPLICAND).

BL←BL AND RMASK8;

* THE PRODUCT WILL BE DEVELOPED IN FT1,FT2,AL,AR. AS LOW-ORDER BITS
* OF THE MULTIPLIER SPILL FROM THE RIGHT END OF AR, LOW-ORDER BITS OF
* THE PRODUCT WILL SPILL INTO THE LEFT END OF AL.

FT1←0; FT2←0;

COUNTER←23;

LINK←IMLOOP;

AL←AL R1 S0; AR←AR SI R1 S0; * INITIALIZE CONDSSHIFT

IMLOOP: FT2←BR*FT2; FT1←BL*FT1 CI R1 S0;

FT2←FT2 SI R1 S0; AL←AL SI R1 S0; AR←AR SI R1 S0 D J;

* THE HIGH-ORDER 23 BITS OF THE 46-BIT PRODUCT ARE IN FT1,FT2. THEY
* MUST BE ALL ZEROS.

FT1←FT1 OR FT2;

GOTO IOF IF FT1#0; * INTEGER OVERFLOW

* SHIFT THE LOW-ORDER 23 BITS NINE PLACES RIGHT IN AL,AR.

AL←AL R1 S0; AR←AR SI R1;

FT1←AL L8;

AL←AL R8;

AR←AR R8; AR←FT1 OR AR;

GOTO IPACK;

* PACK RESULT AND RETURN

*

* AT LEAST ONE OF A,B IS FLOATING-POINT.

PRODFLOAT: GOTO PRODCHKB IF ALS\$FLOTYPE=1;

* A IS AN INTEGER.

CALLF(FLOATA, OPRETURN0); * FLOATA() FAILS IF A=0

PRODCHKB: GOTO FMUL IF BLS\$FLOTYPE=1;

* B IS AN INTEGER.

CALLF(FLOATB, OPRETURN0); * FLOATB() FAILS IF B=0

*GOTO FMUL;

```

*
* FLOATING-POINT MULTIPLICATION.
* SETS:    AL,AR TO THE PRODUCT OF AL,AR AND BL,BR.
* USES:    COUNTER, LINK, BL, BR, ASIGN, AEXP, FT1, FT2.

* UNPACK A AND B.
*    AEXP GETS (A$EXP+B$EXP-EXPBIAS).
FMUL:    AEXP←AR AND EXPB;
          FT1←BR AND EXPB; AEXP←AEXP+FT1; AEXP←AEXP-EXPBIAS;
*    ASIGN GETS (ASSIGN EOR B$SIGN).
          ASIGN←AL EOR BL; ASIGN←ASIGN AND SIGNB;
*    BL,BR GET B$COEF RIGHT JUSTIFIED (MULTIPLIER FOR "*"
*    MICRO-INSTRUCTION).
          FT1←BL L8;
          BL←BL R8;
          BR←BR R8;
          BR←FT1 OR BR;
*    FT1,FT2 GET A$COEF (MULTIPLICAND).
          FT1←AL AND SIGNC; FT2←AR AND EXPC;
* THE PRODUCT IS FORMED BY SUMMING 23 TERMS. ALL BUT THE LAST ADDITION
* IS PERFORMED IN THE FOLLOWING LOOP, WHICH RIGHT SHIFTS THE SUM
* (KEPT IN AL,AR) 1 PLACE (THROUGH THE ROUND AND STICKY BITS) EACH TIME.
          AL←0; AR←0;
          COUNTER←22;
          LINK←FMLOOP;
FMLOOP:  BL←BL R1 S0; BR←BR SI R1 S0;
          AR←FT2*AR; AL←FT1*AL CI R1 S0;
          GOTO FMSHIFTAR IF AR$STICKY=0;
          AR←AR OR STICKYB*2;
FMSHIFTAR: AR←AR SI R1 D J;
* PERFORM LAST MULTIPLICATION STEP WITHOUT SHIFTING SUM.
          BR←BR R1 S0;
          AR←FT2*AR; AL←FT1*AL CI;
* ROUND AND PACK THE RESULT IF NO COEFFICIENT OVERFLOW.
          GOTO FROUND IF NOOVERFLOW;
          GOTO FCOEFOF;
* RIGHT-SHIFT COEF 1 PLACE

```

* QUOTIENT *

* ALL QUOTIENTS ARE FLOATING-POINT.

QUOTIENT: GOTO QUOCHKB IF AL\$FLOTYP=1; * A IS FLOATING-POINT
CALLF(FLOATA, OPRETURN0); * FLOATA() FAILS IF A=0
QUOCHKB: GOTO FDIV IF BL\$FLOTYP=1; * B IS FLOATING-POINT
CALLF(FLOATB, DIV0ERR); * FLOATB() FAILS IF B=0
*GOTO FDIV;

*

* FLOATING-POINT DIVISION.

* SETS: AL,AR TO THE QUOTIENT OF AL,AR DIVIDED BY BL,BR (ASSUMED #0).
* USES: COUNTER, LINK, BL, BR, ASIGN, AEXP, FT1, FT2.

* UNPACK A AND B.

* AEXP GETS (A\$EXP-B\$EXP+EXPBIAS).

FDIV: FT1←BR AND EXPB; FT1←FT1 EOR -1;
AEXP←AR AND EXPB; AEXP←AEXP+FT1 P1; AEXP←AEXP+EXPBIAS;

* ASIGN GETS (A\$SIGN EOR B\$SIGN).

ASIGN←AL EOR BL; ASIGN←ASIGN AND SIGNB;

* AL,AR GET ASCOEF (DIVIDEND).

AL←AL AND SIGNC; AR←AR AND EXPC;

* BL,BR GET THE 1'S COMPLEMENT OF B\$COEF (DIVISOR).

BL←BL AND SIGNC; BR←BR AND EXPC;

BL←BL EOR -1; BR←BR EOR -1;

* THE QUOTIENT IS FORMED BIT-BY-BIT BY REPEATED TRIAL SUBTRACTIONS

* OF THE DIVISOR FROM THE DIVIDEND. A SUCCESSFUL SUBTRACTION

* CORRESPONDS TO A "1" IN THE QUOTIENT; AN UNSUCCESSFUL SUBTRACTION

* CORRESPONDS TO A "0". ACTUALLY THE 1'S COMPLEMENT OF THE QUOTIENT

* IS DEVELOPED (IN FT1,FT2).

FT1←-1; FT2←-1; * COMPLEMENTED ZEROES

COUNTER←23;

LINK←FDLOOP;

* IF THE FIRST TRIAL SUBTRACTION WOULD FAIL WE "PRENORMALIZE" THE

* DIVIDEND BY SHIFTING IT 1 PLACE LEFTWARD.

ZERO←AR+BR P1 L1 S0; AL←AL/BL CI SI L1 S0;

GOTO FDSTART IF NOSPILL; * NONNEGATIVE DIFFERENCE

AR←AR L1 S0; AL←AL SI L1; * LEFT SHIFT FRACTION AND

AEXP←AEXP-1; * DECREMENT EXPONENT

FDLOOP: ZERO←AR+BR P1 L1 S0; AL←AL/BL CI SI L1 S0;

FDSTART: FT2←FT2 SI L1 S0; FT1←FT1 SI L1;

GOTO FDQ0 IF FT2\$15=1; * NEGATIVE DIFFERENCE

AR←AR+BR P1 L1 D J; * FINISH THE SUBTRACTION

GOTO FDLAST;

FDQ0: AR←AR L1 S0; AL←AL SI L1 D J;

* COMPLEMENT FT1,FT2 TO OBTAIN TRUE, UNROUNDED QUOTIENT.

FDLAST: FT1←FT1 EOR -1; FT2←FT2 EOR -1;

* PRODUCE A 24-TH QUOTIENT BIT (WITHOUT INSERTING IT IN FT1,FT2)

* FOR ROUNDING PURPOSES. (SEE COMMENTS ON ROUNDING AT FROUND.)

ZERO←AR+BR P1 L1 S0; AL←AL/BL CI SI L1 S0;

GOTO FDPACK IF SPILL; * ROUND BIT = 0

AR←AR+BR P1 L1; * FINISH THE SUBTRACTION

```
* NOW AL,AR = 0 IFF STICKY BIT = 0.  
  AL←AL OR AR;  
  GOTO FDROUNDUP IF AL#0;  
* ROUND=1, STICKY=0. ROUND QUOTIENT TO NEAREST EVEN.  
  GOTO FDPACK IF FT2$15=0;      * EVEN ALREADY  
FDROUNDUP: FT2←FT2+1; FT1←FT1+ZERO C1;  
  GOTO FDPACK IF FT1$8=0;      * NO OVERFLOW  
* COEFFICIENT ROUNDING OVERFLOW. NOTE FT1=000200B AND FT2=000000B.  
  FT1←000100B;  
  AEXP←AEXP+1;  
* POSITION QUOTIENT COEFFICIENT CORRECTLY AND TRANSFER TO FPACK.  
FDPACK:  AL←FT2 R8; FT1←FT1 L8; AL←FT1 OR AL;  
  AR←FT2 L8;  
  GOTO FPACK;
```

* AND *

BAND: CALL TESTLOGAB; * SETS A,B TO INTEGER 0 OR 1
 LINK←OPRET;
 AR←AR AND BR ANDRETURN;

* OR *

BOR: CALL TESTLOGAB;
 LINK←OPRET;
 AR←AR OR BR ANDRETURN;

*
*
* TESTLOGAB()
* CALL: CALL TESTLOGAB .
* SETS: A←FLOOR(A), PROVIDED A=0 OR A=1;
 B←FLOOR(B), PROVIDED B=0 OR B=1.
* USES: TEMP.

TESTLOGAB: GOTO TLABFLO IF BL\$FLOTYPE=1;
* B IS AN INTEGER.
 GOTO DOMAINERROR IF BL\$R#0;
 GOTO TESTLOGA IF BR=0; * TESTNOTEQUAL(BR,0,TESTLOGA)
 TESTNOTEQUAL(BR, 1, TESTLOGA);
 GOTO DOMAINERROR;
* B IS FLOATING-POINT.
TLABFLO: TESTEQUAL(BR, FLPT1R, DOMAINERROR);
 TESTEQUAL(BL, FLPT1L, DOMAINERROR);
 BL←INTTYPEB; BR←1; * B=1
 *GOTO TESTLOGA; * CALL TESTLOGA & RETURN

*
* TESTLOGA()

TESTLOGA: GOTO TLAFLO IF AL\$FLOTYPE=1;
* A IS AN INTEGER.
 GOTO DOMAINERROR IF AL\$R#0;
 RETURNIF(AR=0); * A=0
 TEMP←AR EOR 1;
 RETURNIF(TEMP=0); * A=1
 GOTO DOMAINERROR;
* A IS FLOATING-POINT.
TLAFLO: TESTEQUAL(AR, FLPT1R, DOMAINERROR);
 TESTEQUAL(AL, FLPT1L, DOMAINERROR);
 AL←INTTYPEB; AR←ZERO+ZERO P1 ANDRETURN; * A=1

* LESS *

* THIS OPERATOR ROUTINE RETURNS DIRECTLY TO GOOPRET.
* A<B IF AND ONLY IF SIGNUM(A-B)=-1.

LESS: OPRET←**+4;
 GOTO DIFFERENCE; * OPCALL(DIFFERENCE)
 OPRET←GOOPRET;
 GOTO OPRETURN1 IF AL\$\$SIGN=1; * A<B
 GOTO OPRETURN0; * A>=B

* EQUAL *

* THIS OPERATOR ROUTINE RETURNS DIRECTLY TO GOOPRET.
* IF A\$TYPE=B\$TYPE, THEN A=B IFF A\$VAL=B\$VAL.

EQUAL: FT1←AL EOR BL;
 GOTO EQDIFTYPES IF FT1\$L#0; * SIGNS OR TYPES DIFFERENT
 GOTO OPRETURN0 IF FT1\$R#0; * A#B
 TESTNOTEQUAL(AR, BR, OPRETURN1); * IS A#B ?

*

OPRETURN0: LINK←OPRET; AL←INTTYPEB; AR←ZERO ANDRETURN;

*

* A\$TYPE#B\$TYPE OR AT LEAST ONE IS FLOATING-POINT. IF EITHER IS CHTYPE,
* THEN A#B.

EQDIFTYPES: TESTNOTEQUAL(AL, CHTYPEB, OPRETURN0);
 TESTNOTEQUAL(BL, CHTYPEB, OPRETURN0);

* A AND B ARE NUMBERS. A=B IFF SIGNUM(A-B)=0=(INTTYPEB,0).

OPRET←**+4;
GOTO DIFFERENCE; * OPCALL(DIFFERENCE)
OPRET←GOOPRET;
TESTEQUAL(AL, INTTYPEB, OPRETURN0);
GOTO OPRETURN0 IF AR#0; * TESTEQUAL(AR, 0, OPRETURN0)

*

OPRETURN1: LINK←OPRET; AL←INTTYPEB; AR←ZERO+ZERO P1 ANDRETURN;

*

OPRETURNB: LINK←OPRET; AL←BL; AR←BR ANDRETURN;

```

*
*
* FLOATA() (FLOATB())
* CONVERTS AL,AR (BL,BR) FROM INTEGER TO FLOATING FORMAT.
* FAILS: IF AL,AR (BL,BR) IS ZERO.
* USES: ASIGN,AEXP,FT1,FT2.

* BEGIN FLOATB().
* FLOATB() INTERCHANGES A AND B, CALLS FLOATA(), AND AGAIN INTERCHANGES
* A AND B.
FLOATB: GOTO FLBICH IF BR#0;
        GOTO FRETURN IF BLSR=0;

*
FLBICH: FT1←AL; AL←BL; BL←FT1;
        FT1←AR; AR←BR; BR←FT1;

*
        FT2←LINK;
        CALL FLOATANZ;           * SKIP ZERO TEST
        LINK←FT2;

* RE-INTERCHANGE A AND B, THEN RETURN FROM FLOATB().
        FT1←AL; AL←BL; BL←FT1;
        FT1←AR; AR←BR; BR←FT1 ANDRETURN;

* BEGIN FLOATA().
* CHECK THAT A#0.
FLOATA: GOTO FLOATANZ IF AR#0;
        GOTO FRETURN IF ALSR=0;

* UNPACK ASIGN AND SHIFT AL,AR LEFT 8 PLACES.
FLOATANZ: ASIGN←AL AND SIGNB;
          AL←AL L8; FT1←AR R8; AL←AL OR FT1;
          AR←AR L8;

* INITIALIZE AEXP TO PLACE RADIX POINT AT RIGHT EDGE OF COEFFICIENT.
          AEXP←EXPBIAS+22;

* TEST FOR SHIFT BY 16.
          GOTO FLT8 IF AL#0;
          GOTO FLS8 IF AR$0#0;           * CAN AT LEAST SHIFT BY 8
          AEXP←AEXP-16;
          AL←AR; AR←0;

* TEST FOR SHIFT BY 8.
FLT8: FT1←AL AND MASK(1,8);
      GOTO FLT421 IF FT1#0;

FLS8: AEXP←AEXP-8;
      AL←AL L8; FT1←AR R8; AL←AL OR FT1;
      AR←AR L8;

* TEST FOR SHIFT OF LESS THAN 8 (DONE IN LOOP).
FLT421: GOTO FLPACK IF AL$COEFB1=1;
FLS421: AEXP←AEXP-1;
        AR←AR L1 S0; AL←AL SI L1;
        GOTO FLS421 IF AL$COEFB1=0;

* PACK SIGN, COEFFICIENT, AND EXPONENT INTO AL,AR AND RETURN.
FLPACK: AL←AL OR ASIGN;
        AR←AR OR AEXP ANDRETURN;

*
FRETURN: LINK←LINK-ONEINST; RETURN;

```

```

*
* INDEX = 1001MMMM. M IS THE EXPECTED RANK.

INDEX:    FIRSTBYTE←FIRSTBYTE AND RMASK4; * EXPECTED RANK
          CALL TOP;
* FIRST OPERAND MUST BE AN ARRAY.
          TEMP←AL AND SCATYPEB;
          GOTO RANKERROR IF TEMP#0;
          GOTO WHATERR IF ALSARYTYPE=0;
* FIRST OPERAND IS AN ARRAY -- INDEXER() EXPECTS ITS ADDRESS IN T1.
          T1←AR;
          T2←LTOP-1; * ADDRESS OF 1ST SUBSCRIPT
          CALL INDEXER; * COMPUTE INDEX IN T3
* IF AN "INDEX RAVELED" INSTRUCTION IS ADDED, IT SHOULD END WITH
* A TRANSFER TO HERE:
INDEXY:   READTESTG( T3 );
          LTOP←T2+1; * POP OPERANDS EXCEPT LAST
          FRP←T1; * ADDRESS OF ARRAY TO FREE
          WAITREAD;
          MEMDATAL←MEMDATAL; MEMDATAR←MEMDATAR;
          WRITELTOP;
          LINK←CYCLE; * CALL DECRFCF(FRP) &
          GOTO DECRFCF; * GOTO CYCLE

*
*
* INDEXER() -- DO RANK-N ARRAY INDEXING WITH SCALAR SUBSCRIPTS.
*
* CALL:    T0=FIRSTBYTE CONTAINS THE EXPECTED RANK;
*          T1 CONTAINS THE G-RELATIVE ADDRESS OF THE ARRAY;
*          T2 CONTAINS THE L-RELATIVE ADDRESS OF THE FIRST
*          (LEFTMOST) SUBSCRIPT (SUCCEEDING SUBSCRIPTS ARE AT
*          LOWER ADDRESSES).
* RETURN:  T1 IS UNCHANGED;
*          T2 CONTAINS (L-RELATIVE ADDRESS OF THE LAST SUBSCRIPT)-1;
*          T3 CONTAINS THE G-RELATIVE ADDRESS OF THE SELECTED
*          ARRAY ELEMENT.
* USES:    EVERYTHING, INCLUDING AL,AR,BL,BR.

* CHECK THAT GIVEN ARRAY IS OF THE EXPECTED RANK AND SET UP THE
* REGISTERS FOR THE MAIN LOOP.
INDEXER:  READTESTG( T1 ); * READ 1ST HEADER WORD
* T3 ACCUMULATES THE INDEX IN ARRAY.RAVEL;
* T4 SCANS THROUGH ADDRESSES IN ARRAY.SHAPE;
* BL WILL CONTAIN INDEX ORIGIN, 1'S COMPLEMENTED;
* T5 SAVES THE RETURN ADDRESS.
          T3←0;
          T4←T1+SHAPEOFFSET;
          BL←FLAGS AND IORGB; BL←BL EOR -1;
          T5←LINK;

*
          TEMP←MEMDATAL EOR FIRSTBYTE PAUSE;
          GOTO IXRNARGERR IF TEMPSR#0; * WRONG NO. OF SUBSCRIPTS
* BEGIN MAIN LOOP.

```

```

IXRLOOP:  READTESTG( T4 );          * READ NEXT ARRAY SHAPE WORD
          T4←T4+1;
* PREPARE TO MULTIPLY T3 BY THE NEXT SHAPE NUMBER.
          WAITREAD;
          BR←MEMDATAR;              * BR←SHAPE NUMBER
          GOTO IXRREADSUB IF T3=0;  * (S TIMES 0)=0
* TEMP WILL HOLD THE HIGH-ORDER BITS OF THE PRODUCT, WHICH ON
* COMPLETION WILL BE ALL ZEROS.
* T3 IS THE MULTIPLIER, AND ON COMPLETION WILL CONTAIN (THE LOW-ORDER
* BITS OF) THE PRODUCT.
          MULTIPLY( T3, BR, TEMP );
* FETCH THE NEXT SUBSCRIPT, WHICH SHOULD BE A NON-NEGATIVE INTEGER
* LESS THAN 2+16.
IXRREADSUB:READL( T2 );
          TEMP←LBASE+2; TEMP←TEMP EOR -1; ZERO←T2+TEMP P1;
          GOTO STKUFERR IF NOCARRY;  * STACK UNDERFLOW
          T2←T2-1;
          AL←MEMDATAL PAUSE; AR←MEMDATAR;
*
          LINK←IXRINT;
          GOTO FIXA IF AL$FLOTYPE=1; * CALL FIXA IF AL$FLOTYPE=1
          GOTO TYPEERROR IF AL$CHTYPE=1;
          GOTO RANKERROR IF AL$ARYTYPE=1; * ONLY SCALAR SUBSCRIPTS OK
          GOTO RANKERROR IF AL$UNDFTYPE=1; * NO "MISSING SUBSCRIPTS"
          GOTO WHATERR IF AL$INTTYPE=0;
* SUBSCRIPT IS (NOW) AN INTEGER. CHECK IT AGAINST THE ARRAY BOUNDS.
IXRINT:  GOTO INDEXERROR IF AL$SIGN=1; * SUBSCRIPT<0
          GOTO INDEXERROR IF AL$R#0;  * 2+16<=SUBSCRIPT
          AR←AR+BL P1;                 * SUBTRACT INDEX ORIGIN
          GOTO INDEXERROR IF NOCARRY;  * SUBSCRIPT<INDEX ORIGIN
          TEMP←BR EOR -1; ZERO←AR+TEMP P1;
          GOTO INDEXERROR IF CARRY;    * UPPER BOUND<=SUBSCRIPT
*
          T3←T3+AR;                    * ACCUMULATE RESULT
*
          FIRSTBYTE←FIRSTBYTE-1;
          GOTO IXRLOOP IF FIRSTBYTE#0;
* NOW T4 CONTAINS @ARRAY.RAVEL[FIRST] AND T3 CONTAINS THE DESIRED
* RESULT LESS T4.
          T3←T4+T3;
*
          LINK←T5;
          RETURN;

```

```

*
* SHORTCONST = 101LLLLLLLLLLLLLLL.

* L IS A SIGNED-MAGNITUDE INTEGER TO BE PUSHED ONTO THE STACK.

SHORTCONST:CALL NEXTBYTE;           * TEMP←NEXTBYTE()
      MEMDATAL←INTTYPEB;
      GOTO SCY IF FIRSTBYTE$OPCB3=0; * NUMBER IS POSITIVE
      MEMDATAL←SIGNB+INTTYPEB;      *          NEGATIVE
SCY:   FIRSTBYTE←FIRSTBYTE AND RMASK4;
      FIRSTBYTE←FIRSTBYTE L8;
      MEMDATAR←FIRSTBYTE OR TEMP;
      INCLTOP;
      GOTO WLCYCLE;                 * WRITELTOP & GOTO CYCLE

```


*
* ETC = 11NNNNNN.

*
* TRANSFER VECTOR FOR ETC.

| | | |
|-----------------------------|-------------------|-------------------------------|
| ETCTAB: | DATA UNDEFINED; | * PUSH NOTYPE VALUE |
| | DATA EAT1; | * POP VALUE ON TOP OF STACK |
| | DATA INTERCHANGE; | * INTERCHANGE A AND B |
| | DATA SETORIGIN; | * INTERCHANGE A AND ORIGIN |
| | DATA GETORIGIN; | * PUSH ORIGIN |
| * APL MIXED OPERATORS NEXT: | | |
| | DATA SHAPE; | * MONADIC RHO |
| | DATA RESHAPE; | * DYADIC RHO |
| | DATA RAVEL; | * MONADIC COMMA |
| | DATA CATENATE; | * DYADIC COMMA |
| | DATA INDEXGEN; | * MONADIC IOTA |
| * * | | |
| | DATA CONSCALAR; | * (IMMEDIATE) CONSTANT SCALAR |
| | DATA CONVEC; | * VECTOR |
| * * | | |
| | DATA BRANCH; | * MONADIC RIGHT-ARROW |
| | DATA GO; | * (UNCONDITIONAL JUMP) |
| | DATA GOTRUE; | * (JUMP ON TRUE) |
| | DATA GOFALSE; | * (JUMP ON FALSE) |
| * * | | |
| | DATA RETURNF; | * (RETURN FROM FUNCTION) |
| | DATA CALLFCN; | * CALL FUNCTION |
| | DATA CYCLE; | * NO OPERATION |
| | DATA BKPTRAP; | * CAUSE BREAKPOINT TRAP |
| | DATA ATNTRAP; | * CAUSE ATTENTION TRAP |

* RESERVED FOR FUTURE EXPANSION:

DATA ETCUNDERR;
DATA ETCUNDERR;
DATA ETCUNDERR;
DATA ETCUNDERR;
DATA ETCUNDERR;
DATA ETCUNDERR;
DATA ETCUNDERR;

*
DC ETCUNDOPN ← *-ETCTAB;
* IMPORTANT:
* THE VALUE "ETCUNDOPN" MUST BE EVEN AND NOT GREATER THAN 64.
*
* BEGIN ETC.

ETC: FIRSTBYTE←FIRSTBYTE AND RMASK6;
* THE FOLLOWING TEST IS NOT NECESSARY IF ETCUNDOPN=63.
TESTLESS(FIRSTBYTE, ETCUNDOPN, ETCUNDERR);
LINK←FIRSTBYTE+ETCTAB;
LINK←ZERO LOAD 0;
GOTO 0 IX;

```

*
* PUSH AN "UNSPECIFIED" VALUE ONTO THE STACK.

UNDEFINED: INCLTOP;
           MEMDATAL←UNDFTYPEB; MEMDATAR←0;
           GOTO WLCYCLE;

*
* REMOVE ONE ELEMENT FROM THE STACK.

EAT1:     CALL TOP;
           LTOP←LTOP-1;
           LINK←CYCLE;           * CALL DECRFC &
           GOTO DECRFC;         *     GOTO CYCLE

*
* INTERCHANGE THE TOP TWO ELEMENTS OF THE STACK.

INTERCHANGE:CALL TOP2;
            TEMP←LTOP-1;
            MEMDATAL←AL; MEMDATAR←AR;
            WRITEL( TEMP );
            MEMDATAL←BL; MEMDATAR←BR;
            GOTO WLCYCLE;           * WRITELTOP & GOTO CYCLE

*
* EXCHANGE THE TOP ELEMENT OF THE STACK WITH THE INDEX ORIGIN BIT.

* THE OPERAND MUST BE A "LOGICAL" (=0 OR 1) SCALAR OR 1-ELEMENT ARRAY.
SETORIGIN: CALL PSEUDOINT;
           CALL TESTLOGA;           * ENSURES INTEGER 0 OR 1
           MEMDATAL←INTTYPEB; MEMDATAR←FLAGS AND IORGB;
           CLEARFLAG( IORGB );     * CLEAR INDEX ORIGIN &
           SETFLAG( AR );          *     SET IT FROM BIT IN AR
           GOTO WLCYCLE;

*
* PUSH THE INDEX ORIGIN BIT ONTO THE STACK AS AN INTEGER VALUE.

GETORIGIN: INCLTOP;
           MEMDATAL←INTTYPEB; MEMDATAR←FLAGS AND IORGB;
           GOTO WLCYCLE;

```

```

*
* SHAPE ("MONADIC RHO")
* FOR ALL A, (SHAPE A) RETURNS A VECTOR OF LENGTH A.RANK WHOSE
* ELEMENTS ARE THE WIDTHS OF THE COORDINATES OF A.

* LOAD OPERAND INTO AL,AR.
SHAPE:    CALL TOP;
* MKNELTS WILL CONTAIN THE LENGTH OF THE RESULT VECTOR.
    MKNELTS←0;                * RANK OF SCALAR IS ZERO
    TEMP←AL AND SCATYPEB;
    GOTO SHPALLOC IF TEMP#0;    * OPERAND IS SCALAR
    GOTO WHATERR IF ALSARYTYPE=0;
* OPERAND IS AN ARRAY. SET MKNELTS TO ITS RANK.
    READTESTG( AR );          * READ 1ST HEADER WORD
    MKNELTS←MEMDATAL AND RANKB PAUSE;
* CREATE A VECTOR OF LENGTH (MKNELTS).
SHPALLOC: CALL MAKEVEC;        * RPTR←MAKEVEC(MKNELTS)
* SET OPFLAGS FOR USE BY THE GENERALIZED OPERATOR APPLICATION
* MECHANISM.
    SETFLAG( MONOPB+RARRAYB );
* FINISH SETTING UP FOR GENERALIZED OPERATOR APPLICATION.
    APTR←AR+SHAPEOFFSET;      * APTR←@A.SHAPE[FIRST]
    OPNO←IDOPNO;
    GOTO GENOP;

```

```

*
* RESHAPE ("DYADIC RHO")
* ROUGHLY SPEAKING, (A RESHAPE B) RETURNS AN ARRAY WITH SHAPE A AND
* RAVEL B. A MUST BE A VECTOR (SPECIAL CASE: A SCALAR) WHOSE ELEMENTS
* ARE NON-NEGATIVE INTEGERS LESS THAN 2+16 AND WHOSE PRODUCT IS LESS
* THAN 2+16. THE ONLY CONSTRAINT ON B IS THAT IT MAY BE EMPTY ONLY IF
* SOME ELEMENT OF A IS ZERO.

```

```

* LOAD OPERANDS INTO AL,AR AND BL,BR.

```

```

RESHAPE: CALL TOP2;

```

```

* DISPATCH ON TYPE OF A.

```

```

    GOTO RSAFLO      IF AL$FLOTYPE=1;

```

```

    GOTO RSAINT      IF AL$INTTYPE=1;

```

```

    GOTO TYPEERROR  IF AL$CHTYPE=1;

```

```

    GOTO RSAARR      IF AL$ARYTYPE=1;

```

```

    GOTO WHATERR;

```

```

* A IS FLOATING-POINT.

```

```

RSAFLO: CALL FIXA;

```

```

* CONVERT TO INTEGER

```

```

* A IS (NOW) AN INTEGER.

```

```

RSAINT: GOTO DOMAINERROR IF AL$SIGN=1;

```

```

* SHAPE NUMBER<0

```

```

    GOTO DOMAINERROR IF AL$R#0;

```

```

* 2+16<=SHAPE NUMBER

```

```

    MKRANK+1;

```

```

* RANK OF RESULT;

```

```

    MKNELTS+AR;

```

```

* LENGTH OF RESULT

```

```

    GOTO RSCHECKB;

```

```

* A IS AN ARRAY. IT MUST BE OF RANK ONE.

```

```

RSAARR: READTESTG( AR );

```

```

* READ 1ST HEADER WORD

```

```

    TEMP+MEMDATAL EOR 1 PAUSE;

```

```

    GOTO RANKERROR IF TEMP$R#0;

```

```

* IF A IS EMPTY THE RESULT WILL BE SCALAR.

```

```

    READTESTG( AR P1 );

```

```

* READ 2ND HEADER WORD

```

```

    WAITREAD;

```

```

    MKRANK+MEMDATAR;

```

```

    GOTO RSRSCA IF MKRANK=0;

```

```

* RESULT WILL BE SCALAR

```

```

* OTHERWISE A MUST HAVE FEWER THAN 256 ELEMENTS.

```

```

    GOTO LENGTHERROR IF MKRANK$L#0;

```

```

* THE PRODUCT OF THE ELEMENTS OF A WILL BE COMPUTED IN MKNELTS.

```

```

    MKNELTS+1;

```

```

* APTR WILL SCAN BACKWARDS THROUGH A.RAVEL.

```

```

    T0+AR+SHAPEOFFSET; APTR-T0+MKRANK;

```

```

RSSHPLOOP: READTESTG( APTR );

```

```

* READ NEXT SHAPE NUMBER

```

```

    AL+MEMDATAL PAUSE; AR+MEMDATAR;

```

```

    GOTO RSSFLO      IF AL$FLOTYPE=1;

```

```

    GOTO RSSINT      IF AL$INTTYPE=1;

```

```

    GOTO TYPEERROR  IF AL$CHTYPE=1;

```

```

    GOTO WHATERR;

```

```

* SHAPE NUMBER IS FLOATING-POINT.

```

```

RSSFLO: CALL FIXA;

```

```

* CONVERT TO INTEGER

```

```

    MEMDATAL+AL; MEMDATAR+AR;

```

```

* AND

```

```

    WRITEG( APTR );

```

```

* UPDATE A!

```

```

* SHAPE NUMBER IS (NOW) AN INTEGER.

```

```

RSSINT: GOTO DOMAINERROR IF AL$SIGN=1;

```

```

* SHAPE NUMBER<0

```

```

    GOTO DOMAINERROR IF AL$R#0;

```

```

* 2+16<=SHAPE NUMBER

```

```

* SET MKNELTS TO (MKNELTS) TIMES (AR).

```

```

    MULTIPLY( MKNELTS, AR, TEMP );

```

```

        GOTO DOMAINERROR IF TEMP#0;          * 2+16<=PRODUCT
*
        APTR←APTR-1;                          * STEP BACK ONE
        TESTEQUAL( APTR, T0, RSSHPLOOP );    * MORE ELEMENTS OF A.RAVEL
* RESTORE THE ARRAY DESCRIPTOR IN AL,AR.
        AL←ARYTYPEB; AR←APTR-SHAPEOFFSET; * (APTR)=@A.SHAPE[FIRST]
* B MAY BE EMPTY ONLY IF THE RESULT ARRAY WILL BE EMPTY.
RSCHECKB: GOTO RSALLOC IF MKNELTS=0;        * RESULT WILL BE EMPTY
        TEMP←BL AND SCATYPEB;
        GOTO RSALLOC IF TEMP#0;            * B IS SCALAR -- NOT EMPTY
        GOTO WHATERR IF BLSARYTYPE=0;
* B IS AN ARRAY.
        READTESTG( BR P1 );
        WAITREAD;
        GOTO LENGTHERROR IF MEMDATAR=0;
* CREATE THE RESULT ARRAY BLOCK.
RSALLOC:  CALL MAKE;                          * MKP←MAKE(MKRANK,MKNELTS)
        SETFLAG( RARRAYB );
* CATCOPY() WILL BE USED TO FILL IN THE SHAPE AND RAVEL OF THE NEW
* ARRAY.
        NELTS←MKRANK+MKNELTS;                * NELTS IS MAX NO. TO COPY
        RPTR←MKP+SHAPEOFFSET;                * RPTR←@RESULT.SHAPE[FIRST]
*
        CALL CATCOPY;                          * FILL IN THE SHAPE WORDS
*
        AL←BL; AR←BR;
RSRAVLOOP: CALL CATCOPY;                       * FILL IN SOME RAVEL WORDS
        GOTO RSRAVLOOP IF NELTS#0;          * NOT FINISHED YET
        GOTO GENOPFIN;
*
* A IS AN EMPTY VECTOR, SO THE RESULT WILL BE SCALAR. B MUST BE
* NON-EMPTY.
RSRSCA:  TEMP←BL AND SCATYPEB;
        GOTO RSRBSCA IF TEMP#0;
        GOTO WHATERR IF BLSARYTYPE=0;
* B IS AN ARRAY.
        READTESTG( BR P1 );
        WAITREAD;
        GOTO LENGTHERROR IF MEMDATAR=0;    * B IS EMPTY
*
        FRP←BR;
        T1←LTOP-1;
        LINK←RESHAPE;                          * CALL PEEL &
        GOTO PEEL;                              * GOTO RESHAPE
* B IS SCALAR.
RSRBSCA: MEMDATAL←BL; MEMDATAR←BR;
        LTOP←LTOP-1;
        WRITELTOP;
        FRP←AR;
        LINK←CYCLE;                              * CALL DECRFCF(FRP) &
        GOTO DECRFCF;                            * GOTO CYCLE

```

```

*
* RAVEL ("MONADIC COMMA")
* IF A IS ANY VALUE, THEN (RAVEL A) RETURNS A VECTOR WHOSE LENGTH IS
* THE PRODUCT REDUCTION OF (SHAPE A) AND WHOSE ELEMENTS ARE THE
* ELEMENTS OF A, ORDERED LEXICOGRAPHICALLY BY SUBSCRIPTS.

* LOAD OPERAND INTO AL,AR.
RAVEL:    CALL TOP;
* DISPATCH ON TYPE OF OPERAND.
* MKNELTS WILL CONTAIN THE LENGTH OF THE RESULT VECTOR.
    MKNELTS←1;
*
    TEMP←AL AND SCATYPEB;
    GOTO RAVALLOC IF TEMP#0;          * OPERAND IS SCALAR
    GOTO WHATERR IF ALSARYTYPE=0;    *
* OPERAND IS AN ARRAY.
* IF ITS RANK IS ONE, (RAVEL A) IS JUST A.
    READTESTG( AR );                * READ 1ST HEADER WORD
    TEMP←MEMDATAL AND RANKB PAUSE;
    TESTNOTEQUAL( TEMP, 1, CYCLE );
*
    READTESTG( AR P1 );              * READ 2ND HEADER WORD
    WAITREAD;
    MKNELTS←MEMDATAR;
* CREATE THE RESULT VECTOR.
RAVALLOC: CALL MAKEVEC;
* SET OPFLAGS FOR GENOPFIN.
    SETFLAG( MONOPB+RARRAYB );
* USE CATCOPY() TO FILL IN THE ELEMENTS OF THE NEW VECTOR.
* MAKEVEC() SET RPTR TO @RESULT.RAVEL[FIRST].
    LINK←GENOPFIN;                  * CALL CATCOPY &
    GOTO CATCOPY;                   * GOTO GENOPFIN

```

```

*
* CATENATE ("DYADIC COMMA") -- FOR OPERANDS OF RANK LESS THAN TWO.
* LOAD OPERANDS INTO AL,AR AND BL,BR.
CATENATE: CALL TOP2;
* DISPATCH ON TYPE OF FIRST OPERAND.
* THE LENGTH OF THE RESULT WILL BE COMPUTED IN MKNELTS.
      MKNELTS+1;
      TEMP+AL AND SCATYPEB;
      GOTO CATDISPB IF TEMP#0;          * FIRST OPERAND IS SCALAR
      GOTO WHATERR IF ALSARYTYPE=0;
* FIRST OPERAND IS AN ARRAY. ITS RANK MUST BE ONE.
      READTESTG( AR );                * READ 1ST HEADER WORD
      TEMP+MEMDATAL EOR 1 PAUSE;
      GOTO RANKERROR IF TEMP$R#0;
*
      READTESTG( AR P1 );              * READ 2ND HEADER WORD
      WAITREAD;
      MKNELTS+MEMDATAR;
* DISPATCH ON TYPE OF SECOND OPERAND.
CATDISPB: TEMP+BL AND SCATYPEB;
      GOTO CATBSCA IF TEMP#0;          * SECOND OPERAND IS SCALAR
      GOTO WHATERR IF BLSARYTYPE=0;
* SECOND OPERAND IS AN ARRAY. ITS RANK MUST BE ONE.
      READTESTG( BR );                * READ 1ST HEADER WORD
      TEMP+MEMDATAL EOR 1 PAUSE;
      GOTO RANKERROR IF TEMP$R#0;
*
      READTESTG( BR P1 );              * READ 2ND HEADER WORD
      WAITREAD;
      MKNELTS+MEMDATAR+MKNELTS;
*
      GOTO CATALLOC;
* SECOND OPERAND IS SCALAR.
CATBSCA: MKNELTS+MKNELTS+1;
* CREATE VECTOR TO HOLD RESULT.
CATALLOC: GOTO LENGTHERROR IF OVERFLOW; * RESULT LENGTH MUST BE <2+16
          CALL MAKEVEC;                * RPTR+MAKEVEC(MKNELTS)
* SET OPFLAGS FOR GENOPFIN.
          SETFLAG( RARRAYB );
* FILL IN THE ELEMENTS OF THE RESULT.
* CATCOPY() COPIES ELEMENTS OF OBJECT DESCRIBED BY AL,AR INTO RESULT
* VECTOR AND INCREMENTS RPTR. NELTS IS AN UPPER BOUND ON THE
* NUMBER OF ELEMENTS COPIED.
          NELTS+2+16-1;                * RESULT.NELTS<=2+16-1
          CALL CATCOPY;
          AL+BL; AR+BR;
          LINK+GENOPFIN;              * CALL CATCOPY &
          *GOTO CATCOPY;              * GOTO GENOPFIN;

```

```

*
*
* CATCOPY() -- COPY ELEMENTS OF A INTO RESULT VECTOR.
* CALL:    AL,AR+DESCRIPTOR (FOR SCALAR OR ARRAY);
*          RPTR←INITIAL ADDRESS IN RAVEL OF RESULT VECTOR;
*          NELTS←MAXIMUM NUMBER OF ELEMENTS TO COPY;
*          CALL CATCOPY .
* SETS:    GSEG[RPTR],...,GSEG[RPTR+N-1] TO APPROPRIATE ELEMENTS OF A;
*          RPTR TO (RPTR+N);
*          NELTS TO (NELTS-N);
*          WHERE N = MINIMUM(NELTS,A.NELTS).
* USES:    APTR,RPTR,T8,T9=TEMP.

CATCOPY:  RETURNIF( NELTS=0 );          * NOTHING TO COPY
* DISPATCH ON RANK OF OPERAND.
  TEMP←AL AND SCATYPEB;
  GOTO CATCOPSCA IF TEMP#0;
  GOTO WHATERR  IF ALSARYTYPE=0;
* OPERAND IS AN ARRAY.
  READTESTG( AR );                    * READ 1ST HEADER WORD
  APTR←AR+SHAPEOFFSET;
  TEMP←MEMDATAL AND RANKB PAUSE;      * TEMP←A.RANK
  APTR←APTR+TEMP;                    * APTR←A.RAVEL[FIRST]
*
  READTESTG( AR P1 );                 * READ 2ND HEADER WORD
  WAITREAD;
  T8←MEMDATAR;                        * T8←A.NELTS
*
CATCOPLOOP:RETURNIF( NELTS=0 );
  RETURNIF( T8=0 );
  NELTS←NELTS-1;
  T8←T8-1;
  READTESTG( APTR );
  APTR←APTR+1;
  WAITREAD;
  MEMDATAL←MEMDATAL; MEMDATAR←MEMDATAR;
  TESTGWRITE( RPTR );
  RPTR←RPTR+1;
  GOTO CATCOPLOOP;
* OPERAND IS SCALAR.
CATCOPSCA: MEMDATAL←AL; MEMDATAR←AR;
  TESTGWRITE( RPTR );
  NELTS←NELTS-1;
  RPTR←RPTR+ZERO P1 ANDRETURN;

```



```

*
* INDEX GENERATOR ("MONADIC IOTA")
* IF L IS A NON-NEGATIVE PSEUDO-INTEGERS, (INDEXGEN L) RETURNS AN
* L-ELEMENT VECTOR CONTAINING THE INTEGERS IORG, IORG+1, ..., IORG+L-1.

INDEXGEN: CALL PSEUDOINT;           * LOAD INTEGER ARG INTO AL, AR
* THE OPERAND MUST BE NON-NEGATIVE AND LESS THAN 2+16.
      GOTO DOMAINERROR IF AL$SIGN=1; * ARG<0
      GOTO DOMAINERROR IF AL$R#0;   * 2+16<=ARG
* CREATE THE RESULT VECTOR.  AR CONTAINS THE NUMBER OF ELEMENTS.
      MKNELTS←AR;
      CALL MAKEVEC;                 * RPTR←MAKEVEC(MKNELTS)
* USE GENOP TO FILL IN THE ELEMENTS OF THE NEW VECTOR.
      SETFLAG( MONOPB+ASCALARB+RARRAYB );
      APTR←FLAGS AND IORGB;        * APTR←INDEX ORIGIN
      OPNO←IGOPNO;
      GOTO GENOP;
* ROUTINE FOR I-TH STEP:
IGELT:  LINK←OPRET;
        AL←INTTYPEB; AR←APTR ANDRETURN;

```

```

*
* CONSCALAR -- (IMMEDIATE) CONSTANT SCALAR. A 32-BIT SCALAR DESCRIPTOR
*           FOLLOWS IN THE INSTRUCTION SEQUENCE.

```

```

CONSCALAR: CALL NEXT2;           * TEMP←NEXT2()
           AL←TEMP;
           CALL NEXT2;         * TEMP←NEXT2()
           MEMDATAL←AL; MEMDATAR←TEMP;
           INCLTOP;
           GOTO WLCYCLE;      * WRITELTOP & GOTO CYCLE

```

```

*
* CONVEC -- (IMMEDIATE) CONSTANT VECTOR.
*
* FIRST BYTE OF CONVEC INSTRUCTION IS ITS ETC-CLASS OPCODE;
* SECOND BYTE IS ELEMENT COUNT (0≤COUNT≤255);
* THE ZERO TO THREE BYTES UP TO THE NEXT FULLWORD BOUNDARY ARE IGNORED;
* FINALLY COME THE 32-BIT SCALAR DESCRIPTORS, ONE TO A FULLWORD.

```

```

* SET MKNELTS←NEXTBYTE().
CONVEC:   CALL NEXTBYTE;       * TEMP←NEXTBYTE()
           MKNELTS←TEMP;
* CREATE RESULT VECTOR, SETTING MKP TO ITS BASE ADDRESS AND RPTR TO
* ADDRESS OF FIRST WORD OF ITS RAVEL.
           CALL MAKEVEC;       * MKP,RPTR←MAKEVEC(MKNELTS)
           SETFLAG( RARRAYB ); * FOR TRAP HANDLING
* SET T1 TO PBASE-RELATIVE FULLWORD ADDRESS OF NEXT CODE WORD.
           T1←PCTR+3; T1←T1 R1; T1←T1 R1; * T1←CEIL(PCTR/4)
** WHILE MKNELTS#0 DO;
**   MKNELTS←MKNELTS-1;
**   GSEG[RPTR]←PSEG[PBASE+T1];
**   T1←T1+1;
**   RPTR←RPTR+1;
** ENDWHILE;

```

```

CVLOOP:   GOTO CVFIN IF MKNELTS=0;
           MKNELTS←MKNELTS-1;
           TEMP←PBASE+T1;
           READTESTP( TEMP );
           WAITREAD;
           MEMDATAL←MEMDATAL; MEMDATAR←MEMDATAR;
           TESTGWRITE( RPTR );
           T1←T1+1;
           RPTR←RPTR+1;
           GOTO CVLOOP;

```

```

* INCREMENT LTOP, SET LSEG[LTOP] TO A DESCRIPTOR FOR THE NEW ARRAY, AND
* SET PCTR←4*T1.

```

```

CVFIN:   INCLTOP;
           MEMDATAL←ARYTYPEB; MEMDATAR←MKP;
           PCTR←T1+T1 L1;
           GOTO WLCYCLE;      * WRITELTOP & GOTO CYCLE

```



```

*
* RETURNF -- RETURN FROM FUNCTION

* SAVE THE LEFT HALF OF THE FIRST HEADER WORD OF THE CURRENT FUNCTION
* BLOCK IN T0.
RETURNF:  READTESTP( PBASE );          * READ 1ST PROC. HEADER WORD
          T0←MEMDATAL PAUSE;

* GENERATE A TRAP IF THE GLOBAL FUNCTION TRACE MODE IS TURNED ON
* AND THE RETURN TRAP BIT OF THE CURRENT FUNCTION IS SET.
          GOTO RF1      IF FLAGSS$FCNRTRC=0; * NO RETURN TRACING
          GOTO RTNTRAP IF T0$RTNTRACE=1;

* SET T1 TO THE NUMBER OF LOCALS OF THE CURRENT FUNCTION, I.E. TO THE
* NUMBER OF ARGUMENT VARIABLES PLUS THE NUMBER OF LOCAL VARIABLES.
RF1:      T1←MEMDATAR R8;
          T1←T1 R1; T1←T1 R1; T1←T1 R1; T1←T1 R1;
          TEMP←MEMDATAL AND NLOCALSB;
          T1←T1+TEMP;

* SET T2 TO THE LBASE OF THE FUNCTION BEING RETURNED TO. UNLESS THIS
* IS LESS THAN THE CURRENT LBASE, GENERATE A "RETURN FROM LEVEL ZERO"
* TRAP.
          READL( LBASE );
          T2←MEMDATAL PAUSE;
          TEMP←LBASE EOR -1;
          ZERO←T2+TEMP P1;
          GOTO RETURNERROR IF CARRY;      * T2>=LBASE

* CHECK THAT THE RESULT VALUE IS DEFINED AND SAVE IT IN AL,AR, PROVIDED
* THE CURRENT FUNCTION HAS A RESULT. IN ANY CASE SET T3 TO THE ADDRESS
* OF THE "TOPMOST" LOCAL TO BE RELEASED.
          T3←LBASE-1;
          GOTO RF2      IF T0$RESULT=0; * NO RESULT
          GOTO LERR     IF NOCARRY;
          READTESTL( T3 );          * READ RESULT VARIABLE
          T3←T3-1;
          T1←T1-1;                * ONE LESS LOCAL TO RELEASE
          AL←MEMDATAL PAUSE; AR←MEMDATAR; * SAVE RESULT VALUE
          TESTNOTEQUAL( AL, UNDFTYPEB, VALUEERROR );

* RELEASE THE LOCAL VALUES.
RF2:      LTOP←T3;                  * ADDRESS OF FIRST TO RELEASE
RFRLSLOOP: GOTO RF3 IF T1=0;        * ALL RELEASED
          T1←T1-1;
          READTESTL( LTOP );
          LTOP←LTOP-1;
          LINK←RFRLSLOOP;          * CALL DECRFC &
          GOTO DECRFC;            * GOTO RFRLSLOOP

* STACK THE RESULT VALUE, IF ANY.
RF3:      GOTO RF4 IF T0$RESULT=0; * NO RESULT
          LTOP←LTOP+1;
          MEMDATAL←AL; MEMDATAR←AR;
          WRITELTOP;

* UPDATE THE REGISTERS LBASE, PFIRST, PNLEN, PBASE, AND PCTR.
RF4:      LBASE←T2;
          LINK←CYCLE;              * CALL LOADP() &
          GOTO LOADP;              * GOTO CYCLE

```

```

*
* CALL FUNCTION
*
* FIRST BYTE OF CALL INSTRUCTION IS ITS ETC-CLASS OPCODE;
* SECOND & THIRD BYTES CONTAIN FOLLOWING FIELDS IN INDICATED ORDER:
*   NUMBER OF ARGUMENTS = NARGS (4 BITS)
*   FUNCTION DESCRIPTOR = FDESCR:
*     PROCEDURE SEGMENT SELECTOR = PSEGSEL (1 BIT)
*     FUNCTION NUMBER = FCNO (11 BITS)

* FETCH INSTRUCTION BYTES CONTAINING NARGS, PSEGSEL, AND FCNO.
CALLFCN:  CALL NEXT2;                * TEMP←NEXT2()
* SET T0←FUNCTION DESCRIPTOR (MAINTAINED THROUGHOUT CALLFCN); AND
*   T1←NARGS.
        T0←TEMP AND RMASK12;
        T1←TEMP R8;
        T1←T1 R1; T1←T1 R1; T1←T1 R1; T1←T1 R1;
* CHECK THAT THERE ARE AT LEAST NARGS TEMPORARY VALUES ON TOP OF THE
* STACK, I.E. THAT (LBASE+2)+(NARGS-1)=(LBASE+NARGS+1)≤LTOP.
        TEMP←LBASE+T1 P1;                * TEMP←LBASE+NARGS+1
        TEMP←TEMP EOR -1;                * TEMP←-(LBASE+NARGS+1)-1
        ZERO←LTOP+TEMP P1;
        GOTO STKUFERR IF NOCARRY;
* SET PFIRST,PNLEN TO THE BASE,(-LENGTH) OF THE CALLEE'S PROCEDURE
* SEGMENT.
        TEMP←APLPSEG0WD;
        GOTO CF0 IF T0$PSEGSEL=0;
        TEMP←APLPSEG1WD;
CF0:    READ( TEMP );
        PFIRST←MEMDATAL PAUSE;
        PNLEN←MEMDATAR EOR -1; PNLEN←PNLEN+1;
* SET T1 TO CALLEE'S PBASE. THE PROCEDURE SEGMENT BEGINS WITH A
* SERIES OF TWO-WORD "DIRECTORY ENTRIES". THE LEFT HALF OF THE FIRST
* WORD OF A DIRECTORY ENTRY IS THE SEGMENT ADDRESS OF THE CORRESPONDING
* FUNCTION BODY.
        TEMP←T0 AND FCNOB;
        TEMP←TEMP+TEMP;
        READTESTP( TEMP );
        T1←MEMDATAL PAUSE;
* READ THE FIRST WORD OF THE CALLED FUNCTION BLOCK.
        READTESTP( T1 );
* SET T3 TO LEFT HALF OF FIRST HEADER WORD, T2 TO CALLEE.NLOCALS .
        T3←MEMDATAL PAUSE;
        T2←MEMDATAR AND NLOCALSB;
* CHECK THAT ROOM ON THE STACK EXISTS FOR THE LOCAL VARIABLES AND
* TWO-WORD STACK FRAME OF THE CALLED FUNCTION.
        TEMP←LTOP+T2;    GOTO STKOF IF CARRY;
        TEMP←LTOP+2;    GOTO STKOF IF CARRY;
        ZERO←LNLEN+TEMP; GOTO STKOF IF CARRY;
* SAVE THE CALLER'S PROGRAM COUNTER IN THE CURRENT STACK FRAME.
        RMWL( LBASE P1 );
        WAITREAD;
        MEMDATAR←PCTR; MEMDATAL←MEMDATAL ANDWRITE;

```

```

* INITIALIZE THE CALLEE'S LOCAL VARIABLES WITH UNDFTYPE VALUES.
CFINITLOOP:GOTO CF2 IF T2=0;
    T2←T2-1;
    LTOP←LTOP+1; * OVERFLOW IS IMPOSSIBLE (?)
    MEMDATAL←UNDFTYPEB; MEMDATAR←0;
    WRITELTOP;
    GOTO CFINITLOOP;
* CREATE THE TWO-WORD STACK FRAME.
CF2:    LTOP←LTOP+1;
    MEMDATAL←LBASE; MEMDATAR←0;
    WRITELTOP;
*
    LTOP←LTOP+1;
    MEMDATAL←T0; MEMDATAR←0; * FDESCR (,PCTR)
    WRITELTOP;
* UPDATE THE REGISTERS LBASE, PBASE, AND PCTR.
    LBASE←LTOP-1;
    PBASE←T1;
    PCTR←CODEOFFSET*4;
* THE CALL IS COMPLETE; A FUNCTION CALL TRAP SHOULD BE GENERATED IF
* THE GLOBAL AND LOCAL CALL TRACE MODE BITS ARE SET.
    GOTO CYCLE IF FLAGSS$FCNCTRC=0;
    GOTO CYCLE IF T3$CALLTRACE=0;
    GOTO CALLTRAP;

```

```

*
*
* PSEUDOINT()
* LOADS AL,AR WITH TOP STACK ELEMENT, AND:
*   (1) CONVERTS FLOATING-POINT NUMBER TO INTEGER;
*   (2) "PEELS" A 1-ELEMENT ARRAY.
* USES:   T0,T1,T6,T7,T8,T9, COUNTER, MEMORY REGISTERS

PSEUDOINT: T1←LINK;
           CALL TOP;
           LINK←T1;
* DISPATCH ON TYPE OF OPERAND.
           GOTO FIXA      IF ALSFLOTYPE=1; * CALL FIXA & RETURN
           RETURNIF( ALSINTTYPE=1 );
           GOTO TYPEERROR IF ALSCHTYPE=1;
           GOTO WHATERR   IF ALSARYTYPE=0;
* OPERAND IS AN ARRAY. IT MUST BE A 1-ELEMENT ARRAY.
           READTESTG( AR P1 );           * READ 2ND HEADER WORD
           WAITREAD;
           TESTEQUAL( MEMDATAR, 1, LENGTHERROR );
* "PEEL" THE ARRAY AND START OVER.
           FRP←AR;
           T1←LTOP;
           T0←LINK;
           CALL PEEL;
           LINK←T0;
           GOTO PSEUDOINT;

```



```

*
*
* FIXA()
* CONVERTS FLOATING-POINT NUMBER IN AL,AR TO INTEGER.
* ERROR IF FRACTIONAL PART OF ARGUMENT IS NONZERO OR IF 2+23<=ABS(ARG).
* USES:    COUNTER,ASIGN,AEXP,FT1.

* UNPACK A.
*   ASIGN GETS ASSIGN.
FIXA:   ASIGN←AL AND SIGNB;
*   AEXP GETS A$EXP-EXPBIAS.
        AEXP←AR AND EXPB; AEXP←AEXP-EXPBIAS;
*   AL,AR GETS ASMAGNITUDE.
        AL←AL AND SIGNC;
        AR←AR AND EXPC;
        GOTO DOMAINERROR IF AEXP$0#0;    * AEXP<0
        TESTLESS( AEXP, 23, DOMAINERRR ); * IS AEXP<23 ?
        FT1←AEXP-(30+1);                  * YES
        COUNTER←FT1 EOR -1;                * COUNTER←30-A$EXP
        FT1←LINK;
        LINK←FXLOOP;
FXLOOP: GOTO DOMAINERROR IF AR$15#0;    * A$FRACTION#0
        AL←AL R1 S0; AR←AR SI R1 D J;
        LINK←FT1;
FXPACK: AL←AL OR INTTYPEB;
        AL←ASIGN OR AL ANDRETURN;

```

```

*
*
* PEEL()
* CALL: FRP←<ADDRESS OF ARRAY TO PEEL>
*       T1←<LFIRST-RELATIVE ADDRESS TO PUT PEELED SCALAR>
*       CALL PEEL .
* SETS: L[T1] TO FRP.RAVEL[1] AND DECREMENTS FRP.RFC .
* USES: TEMP,FRP,FRSIZE,FRT1,FRT2, MEMORY REGISTERS

PEEL:  READTESTG( FRP );
       TEMP←MEMDATAL AND RANKB PAUSE;
       TEMP←TEMP+SHAPEOFFSET;
       TEMP←FRP+TEMP;                * TEMP←@ARRAY.RAVEL[FIRST]
       READTESTG( TEMP );
       TEMP←MEMDATAL PAUSE; MEMDATAL←TEMP; MEMDATAR←MEMDATAR;
       TESTLWRITE( T1 );
       GOTO DECRFCF;                * CALL DECRFCF & RETURN

```

```

*
*
* MAKEVEC()
* CALL:      MKNELTS←NUMBER OF ELEMENTS;
*           CALL MAKEVEC .
* SETS:      MKP, GSEG[BLOCKPTR] TO ADDRESS OF NEW VECTOR WITH ITS HEADER
*           AND SHAPE INITIALIZED;
*           RPTR TO ADDRESS OF FIRST ELEMENT OF RAVEL OF NEW VECTOR.
* FAILS:     IF ARRAY STORAGE IS FULL.
* USES:      EVERYTHING EXCEPT T0 (=FIRSTBYTE) AND T1.

MAKEVEC:     MKRANK←1;
             T2←LINK;
             CALL MAKE;
             LINK←T2;
* INITIALIZE THE SHAPE WORD.
             TEMP←MKP+SHAPEOFFSET;
             MEMDATAL←INTTYPEB; MEMDATAR←MKNELTS;
             WRITEG( TEMP );
* SET RPTR AND RETURN.
             RPTR←MKP+SHAPEOFFSET+1;
             RETURN;

```

```

*
*
* MAKE() -- ALLOCATE AN ARRAY BLOCK AND INITIALIZE ITS TWO HEADER WORDS.
* CALL:   MKNELTS ← NUMBER OF ELEMENTS IN NEW ARRAY;
*         MKRANK ← RANK OF NEW ARRAY;
*         CALL MAKE .
* SETS:   MKP = ADDRESS OF NEW BLOCK;
*         MKSIZE = LENGTH OF NEW BLOCK EXCLUDING SLOP;
*         GSEG[BLOCKPTR] = MKP;
*         GSEG[ROVER] = ADDRESS OF "NEXT" FREE BLOCK.
* USES:   MKSIZE, MKT1, MKT2, MKT3 AND MEMORY REGISTERS.
**
** FUNCTION MAKE( NELTS, RANK );
**   GLOBAL CELL ROVER;
**   POINTER P; INTEGER N;
**   P←ROVER;
**   N←2+RANK+NELTS;

MAKE:   READG( ROVER );
        MKSIZE←MKRANK+SHAPEOFFSET;
        MKSIZE←MKSIZE+MKNELTS;
        MKT2←MEMDATAL PAUSE;
        MKP←MKT2;
        * LENGTH OF BLOCK TO BE
        * MKT2←ROVER

**
**   WHILE P.SIZE<N DO;
**     P←P.NEXT;
**     FRETURN IF P=ROVER;
**   ENDWHILE;

MKSEARCH: READTESTG( MKP );
          MKT1←MKSIZE EOR -1;
          WAITREAD;
          MKT1←MEMDATAR+MKT1 P1;
          GOTO MKFOUND IF CARRY;
          * MKT1←P.SIZE-N
          * P.SIZE>=N

          READTESTG( MKP P1 );
          MKP←MEMDATAL PAUSE;
          * P←P.NEXT

          MKT1←MKP EOR MKT2;
          GOTO MKSEARCH IF MKT1#0;
          * P#ROVER
* THERE IS NO BLOCK CONTAINING N OR MORE WORDS.
  GOTO MKGOF;

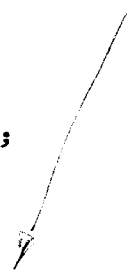
**
**   (P+P.SIZE).PREVFREE←FALSE;
**   ROVER←P.NEXT;

MKFOUND: MKT2←MEMDATAR+MKP;
          * MKT2←P+P.SIZE
          GOTO GERR IF CARRY;
          TESTGRMW( MKT2 );
* NOW WE KNOW ALL ADDRESSES FROM (MKP) THROUGH (MKT2) INCLUSIVE
* ARE WITHIN THE GLOBAL SEGMENT.
  WAITREAD;
  MEMDATAR←MEMDATAR;

```

*To print out of memory free list (pg 51)
the search loop should end after 20 iterations.*

Bug 1



```

MEMDATAL←MEMDATAL AND PREVFREEC ANDWRITE;

READG( MKP P1 );           * READ 2ND HEADER WORD OF P
MKT2←MEMDATAL PAUSE;      * MKT2←P.NEXT
MKT3←MEMDATAR;           * MKT3←P.BACK
MEMDATAL←MKT2; *MEMDATAR←ANYTHING;
WRITEG( ROVER );

**
** IF P.SIZE-N≤MAXSLOP DO;
**   P.BACK.NEXT←P.NEXT;
**   P.NEXT.BACK←P.BACK;
**   P.SLOP←P.SIZE-N;
**   P.THISFREE←FALSE;
**   P.PREVFREE←FALSE;
**   P.RANK←RANK;

* A FREE BLOCK WITH MORE THAN MAXSLOP WORDS OF SLOP WILL BE SPLIT.
  TESTLESS( MKT1, (MAXSLOP+1), MKSPLIT );

    SETNEXT( MKT3, MKT2 );           * P.BACK.NEXT←P.NEXT
    SETBACK( MKT2, MKT3 );          * P.NEXT.BACK←P.BACK

    MKT1←MKT1 L8;
    MEMDATAL←MKT1 OR MKRANK;        * P.FLAGS←...

**
** ELSE DO;
**   P.SIZE←P.SIZE-N;
**   (P+P.SIZE-1).SIZE←P.SIZE
**   P←P+P.SIZE;
**   P.THISFREE←FALSE;
**   P.PREVFREE←TRUE;
**   P.SLOP←0;
**   P.RANK←RANK;

MKSPLIT:   GOTO MKY;
            MEMDATAL←THISFREEB;      * P.FLAGS ARE UNCHANGED
            MEMDATAR←MKT1;          * RECALL MKT1=P.SIZE-N
            WRITEG( MKP );

            MKP←MKP+MKT1;           * P←P+P.SIZE
            MKT2←MKP-1;
            MEMDATAL←0;
            MEMDATAR←MKT1;
            WRITEG( MKT2 );

            MEMDATAL←MKRANK OR PREVFREEB; * P.FLAGS←...

**
**   ENDIF;
**   P.SIZE←N; P.RFC←1; P.NELTS←NELTS; BLOCKPTR←P;
**   SRETURN;
** ENDFUNCTION;

MKY:       MEMDATAR←MKSIZ;          * P.SIZE←N

```

```
WRITEG( MKP ); * WRITE 1ST HEADER WORD OF P
MEMDATAL←1; MEMDATAR←MKNELTS;
WRITEG( MKP P1 ); * WRITE 2ND HEADER WORD OF P
MEMDATAL←MKP; *MEMDATAR←ANYTHING;
WRITEG( BLOCKPTR );
RETURN;
```

```

*
*
* FREE() -- RETURN A GIVEN BLOCK OF STORAGE TO THE FREE POOL.
* CALL:   FRP←POINTER; CALL FREE.
* USES:   FRSIZE (TO HOLD A BLOCK SIZE)
*         FRT1   (TO HOLD A POINTER TO AN ADJACENT BLOCK, OR...)
*         FRT2   (IN CONJUNCTION WITH FRT1, TO HOLD A PAIR OF POINTERS)
*
* THERE ARE FOUR CASES TO CONSIDER WHEN FREEING A BLOCK:
* 1. NEITHER THE PRECEDING NOR THE FOLLOWING BLOCK IS FREE;
* 2. THE FOLLOWING BLOCK IS FREE;
* 3. THE PRECEDING BLOCK IS FREE;
* 4. BOTH ADJACENT BLOCKS ARE FREE.
**
** FUNCTION FREE( P );
**   DECLARE Q;
**   P.SIZE←P.SIZE+P.SLOP;
**   Q←P+P.SIZE;

FREE:   READTESTG( FRP );                * READ 1ST HEADER WORD OF P
        FRSIZE←MEMDATAL AND SLOPB PAUSE; * MEMDATAL$SLOP = P.SLOP &
        FRSIZE←FRSIZE R8;                *   MEMDATAR = P.SIZE
        FRSIZE←MEMDATAR+FRSIZE;
        FRT1←FRP+FRSIZE;                 * FRT1←P+P.SIZE

**
**   IF P.PREVFREE=FALSE DO;
**   IF Q.THISFREE=FALSE DO;
**   Q.PREVFREE←TRUE;

        GOTO FRCASE34 IF MEMDATAL$PREVFREE=1; * MEMDATAL$1=P.PREVFREE
        READTESTG( FRT1 );                * READ 1ST HEADER WORD OF Q
        GOTO FRCASE2 IF MEMDATAL$THISFREE=1 PAUSE;
FRCASE1: MEMDATAR←MEMDATAR;                * Q.SIZE IS UNCHANGED
        MEMDATAL←MEMDATAL OR PREVFREEB ANDWRITE; * Q.PREVFREE←1

**
**   P.NEXT←ROVER.NEXT;
**   P.BACK←ROVER;
**   P.BACK.NEXT←P;
**   P.NEXT.BACK←P;

        READG( ROVER );
        FRT1←MEMDATAL PAUSE;                * FRT1←ROVER
        TESTGRMW( FRT1 P1 );                * SETNEXT( FRT1, FRP )
        FRT2←MEMDATAL PAUSE;                * FRT2←ROVER.NEXT
        MEMDATAR←MEMDATAR;                * ROVER.BACK IS UNCHANGED
        MEMDATAL←FRP ANDWRITE;            * ROVER.NEXT←P

        SETBACK( FRT2, FRP );                * ROVER.NEXT.BACK←P

        MEMDATAL←FRT2; MEMDATAR←FRT1;      * SET P.NEXT AND P.BACK
        TESTGWRITE( FRP P1 );                * WRITE 2ND HEADER WORD OF P

**
**   ELSE DO;

```

```

**          P.SIZE←P.SIZE+Q.SIZE;
**          P.NEXT←Q.NEXT;
**          P.BACK←Q.BACK;

          GOTO FRCASE12Y;
FRCASE2:    FRSIZE←MEMDATAR+FRSIZE;          * MEMDATAR CONTAINS Q.SIZE
          READTESTG( FRT1 P1 );           * READ 2ND HEADER WORD OF Q
          FRT1←MEMDATAL PAUSE;           * FRT1←Q.NEXT
          FRT2←MEMDATAR;                 * FRT2←Q.BACK
          MEMDATAL←FRT1; MEMDATAR←FRT2;   * SET P.NEXT AND P.BACK
          TESTGWRITE( FRP P1 );          * WRITE 2ND HEADER WORD OF P

**
**          P.NEXT.BACK←P;
**          P.BACK.NEXT←P;
**          ROVER←P;

          SETBACK( FRT1, FRP );          * Q.NEXT.BACK←P
          SETNEXT( FRT2, FRP );          * Q.BACK.NEXT←P

          *MEMDATAL←FRP; *MEMDATAR←ANYTHING;
          WRITEG( ROVER );              * ROVER←P

**
**          ENDIF;
**          P.THISFREE←TRUE;

FRCASE12Y: MEMDATAL←THISFREEB; MEMDATAR←FRSIZE;
          WRITEG( FRP );                * WRITE 1ST HEADER WORD OF P

**
**          ELSE DO;
**          IF Q.THISFREE=FALSE DO;
**          Q.PREVFREE←TRUE;

          GOTO FRALLY;
FRCASE34:  READTESTG( FRT1 );           * READ 1ST HEADER WORD OF Q
          GOTO FRCASE4 IF MEMDATAL$THISFREE=1 PAUSE;

          MEMDATAR←MEMDATAR;            * Q.SIZE IS UNCHANGED
          MEMDATAL←MEMDATAL OR PREVFREEB ANDWRITE; * Q.PREVFREE←1

**
**          ELSE DO;
**          P.SIZE←P.SIZE+Q.SIZE;
**          Q.BACK.NEXT←Q.NEXT;
**          Q.NEXT.BACK←Q.BACK;

          GOTO FRCASE34Y;
FRCASE4:  FRSIZE←MEMDATAR+FRSIZE;          * MEMDATAL CONTAINS Q.SIZE

          READTESTG( FRT1 P1 );           * READ 2ND HEADER WORD OF Q
          FRT1←MEMDATAL PAUSE;           * FRT1←Q.NEXT
          FRT2←MEMDATAR;                 * FRT2←Q.BACK

          SETNEXT( FRT2, FRT1 );          * Q.BACK.NEXT←Q.NEXT

```



```

                SETBACK( FRT1, FRT2 );                * Q.NEXT.BACK←Q.BACK
**
**      ENDIF;
**      P←P-(P-1).SIZE;
**      P.SIZE←P.SIZE+(P+P.SIZE).SIZE;
**      ROVER←P;

FRCASE34Y: FRT1←FRP-1;                * FRT1←P-1
          READTESTG( FRT1 );
          WAITREAD;
          FRT1←MEMDATAR EOR -1;        * MEMDATAR HAS (P-1).SIZE
          FRP←FRP+FRT1 P1;            * FRP←P-(P-1).SIZE

          TESTGRMW( FRP );            * READ 1ST HDR WORD OF NEW P
          WAITREAD;
          FRSIZE←MEMDATAR+FRSIZE;
          MEMDATAR←FRSIZE;
          MEMDATAL←MEMDATAL ANDWRITE; * P.FLAGS ARE UNCHANGED

          MEMDATAL←FRP; *MEMDATAR←ANYTHING;
          WRITEG( ROVER );

**
**      ENDIF;
**      (P+P.SIZE-1).SIZE←P.SIZE;
**      RETURN;
** ENDFUNCTION;

FRALLY:   FRT1←FRP+FRSIZE;
          FRT1←FRT1-1;                * FRT1←P+P.SIZE-1
          MEMDATAL←0; ; MEMDATAR←FRSIZE;
          TESTGWRITE( FRT1 ) ANDRETURN;

*
*
*
*
          END;

```