Name:

Project: 1 Programmer: FOO

File Name: XXX[1,FOO]

File Last Written: 4:56 2 May 1974

Time: 4:59 Date: 2 May 1974

Stanford University
Artificial Intelligence Laboratory
Computer Science Department
Stanford, California

```
COMMENT    VALID 00021 PAGES
C REC  PAGE   DESCRIPTION
C00001 00001
C00002 00002     '
C00003 00003     "       Stack Discipline...
C00008 00004
C00010 00005
C00013 00006       END
C00014 00007
C00016 00008
C00017 00009
C00019 00010
C00020 00011
C00022 00012
C00024 00013
C00025 00014
C00028 00015
C00031 00016        END
C00032 00017
C00035 00018
C00037 00019
C00039 00020
C00040 00021
C00041 ENDMK
C;
```

```
'
"       ECL MODEL
';

'       Preliminary setup
"
';

POKE(OPENCG, 1);

UX = NOTHING -> LOADB "AIDS";

EDOPCHANGE("@", "@@");

"@@".SBLK ← "@".SBLK;

SET\UP(77, 1, 0, 27);

↑;

'
"       GENERAL CONVENTIONS:
"
"       Registers...
"
"           A,B,...,E denote hardware registers or scratchpad registers
"           (in a micro-programmed inplementation) for the machine used
"           for the implementation of ECL.  Their length is the length
"           of the storage word on the former machine.  They are used
"           for passing arguments and results between routines and
"           and their callers.
"
```

```
"       Stack Discipline...
"
"           The reason for a stack discipline is principally to allow the
"           garbage collector to function properly.  The rules are:
"
"              Name Stack:  All entries are two WORD records consisting
"                     of the name (SYMBOL), mode (MODE), and
"                     location (XWD) of the object denoted by the
"                     entry.
"
"              Control Stack:  Entries are one of the forms of control
"                     record understood by the garbage collector (so
"                     that they can be traced and marked properly) or
"                     a HWD whose LH part is zero and RH part is
"                     convertible to the mode FORM, or a HWD whose
"                     LH part is non-zero (the word will not be traced).

"
"              Value Stack:  Entries are reached during garbage collec-
"                     tion from the ATOMS, NS, or CS.  If items on
"                     the VS are to be traced, a mode must be attri-
"                     buted to them which is proper for this purpose.
"                     Otherwise, the mode discovered by the garbage
"                     collector must have EPFLG=FALSE to prevent
"                     tracing.
"
"       DECLs in primitives...
"
"           A,B,...,E must be preserved on NS or CS if a routine called
"           somehow invokes the garbage collector.  Hence, any of these
"           with live content will be explicitly pushed onto a stack
"           when necessary to allow correct garbage collection.
"
"           For mnemonic value in reading primitve routines, a DECL of
"           the form DECL <mnemonic>:ANY SHARED <register>; is used.
"           Necessary temporaries not vulnerable to garbage collection
"           will be declared using DECL.
"
"       Routine Exits...
"
"           Some routines in the model have two exits.  These corres-
"           pond to the non-skip and skip return in PDP-10 machine-language
"           routines.  In these cases, the result type is a BOOL and TRUE
"           denotes success (skip return) in all cases.
";

↑;
```

```
'
"       REAL MACHINE IMPLEMENTATIONS:
"
"       The mode compiler in the model and the primitives assume an ECL
"       machine (i.e., that on which the model proper and primitive
"       routines are runnable) whose storage word is large enough for
"       two storage addresses.  Specifically, the PDP-10 is assumed.
"
"       However, if WORD\SIZE=HWORD\SIZE below, a machine such
"       as the PDP-11 is assumed.  In the former case, the algorithm
"       for packing STRUCTs packs half-word objects from right to left
"       in the machine word.  It is fortuitous, but extremely pleasant,
"       that when  a machine such as the PDP-11 is considered
"       this is the right order of packing from the point of view of
"       storage addressing on such a machine.
"
"       In the primitives, when the comment /* "special packing"; occurs
"       it is this set of facts that is referred to.  Also note that
"       in the user select function for the mode XWD, special packing
"       is assumed.
';

↑;
```

```
'
"       User mode functions for modes used in the primitive routines
';

'       Supply conversions between the modes HWD, HWORD, SITE, and the
"       various arithmetic modes.  Conversion of a half-word mode to
"       XWD clears the LH part of the XWD, while the opposite conversions
"       just select the RH part.
';

HWDCFN ←
 EXPR(S:ONEOF(SITE, HWORD), M:MODE; ANY)
  BEGIN
   DECL SS:ANY LIKE LOWER(S, VECTOR(HWORD\SIZE, BOOL));
   OR(M = INT, M = ARITHNONE, M = ARITH) => 0 ← SS;
   LIFT(SS, HWORD)           /* 'S WAS A SITE';
  END;

HWDPFN ←
 EXPR(S:ONEOF(SITE, HWORD), P:PORT; ANY)
  BEGIN
   DECL T:VECTOR(6, INT)   /* 'OCTAL DIGITS';
   DECL SS:ANY BYVAL S;
   FOR I FROM 6 BY - 1 TO 1 REPEAT T[I] ← SS - SS / 8 * 8; SS ← SS / 8 END;
   FOR I TO 6 REPEAT PRINT(T[I], P) END;
   S;
  END;

XWDCFN ←
 EXPR(X:XWD, M:MODE; ANY)
  BEGIN
   M = INT OR M = ARITH => X.LH * 262144 + X.RH;
   M = SITE OR M = HWORD => X.RH;
   M = NONE => NOTHING;
   ERROR1();
  END;

XWDAFN ←
 EXPR(T:XWD, S:ANY; XWD)
  BEGIN
   DECL MS:MODE LIKE MD(S);
   DECL TT:XWD.UR LIKE LOWER(T);
   BEGIN
    MS = XWD => TT ← LOWER(S);
    MS = INT =>
     BEGIN
      T.LH ← S / 262144;
      T.RH ← [) S LT 262144 => S; S - S / 262144 * 262144 (];
      T;
     END;
    MS = SITE OR MS = HWORD OR MS = HWORD.UR => [) T.LH ← 0; T.RH ← S; T (];
 ERROR1('TYPE FAULT');
END;
T;
```

```
  END;

XWDSFN <-
 EXPR(X:XWD, IX:ONEOF(INT, SYMBOL); ANY)
  [) MD(IX) = SYMBOL => LOWER(X)[IX]; IX = 1 => LOWER(X)[2]; LOWER(X)[1] (];

XWDPFN ←
 EXPR(S:XWD, P:PORT)
  BEGIN
   PRINT('XWD::(', P);
   PRINT(S.LH, P);
   PRINT(',,', P);
   PRINT(S.RH, P);
   PRINT(%), P);
   S;
  END;

↑;
```

```
'
"       PRIMITIVE ROUTINES AND DATA
"
"       The following comprise modes and data which define the machine
"       on which the primitive routines run.
"
';

WORD\SIZE ← 36              /* 'STORAGE WORD SIZE';

CHAR\SIZE ← 7               /* '# BITS PER CHARACTER';

HWORD\SIZE ← 18             /* 'SIZE OF STORAGE ADDRESS (MAY = WORD\SIZE)';

PAGE\SIZE ← 128      /* 'STORAGE PAGE SIZE (WORDS)';

HIPAG ← 32                  /* 'NUMBER OF PAGES AVAILABLE FOR ALLOCATION';

MAXPAG ← 0                  /* 'LAST PAGE ACTUALLY ALLOCATED';

WORD ←
 WORD ::
  VECTOR(WORD\SIZE, BOOL)  /* 'DEFINES THE MAIN STORAGE WORD';

'       Force compactification so CORE won%'t move later.
';

POKE(CGCCOUNT, 1);

RECLAIM(1, DTPR);

CORE ←
 ALLOC(VECTOR(HIPAG * PAGE\SIZE, WORD))
                         /* 'SIMULATES MAIN STORAGE';

HWORD ← QL("HWORD", HWDCFN, NIL, NIL, HWDPFN) :: VECTOR(HWORD\SIZE, BOOL);

SITE ←
 QL("SITE", HWDCFN, NIL, NIL, HWDPFN) ::
  HWORD                  /* 'USED TO SIMULATE A STORAGE ADDRESS';
```

```
XWD ←
 QL("XWD", XWDCFN, XWDAFN, NIL, XWDPFN) ::
  STRUCT(RH:SITE, LH:HWORD)
                        /* 'SPECIAL PACKING';

/* 'Simulated machine registers or scratchpad registers for global use
   and argument passing';

A ← CONST(XWD);

B ← CONST(XWD);

C ← CONST(XWD);

D ← CONST(XWD);

E ← CONST(XWD);

'
"       Data used for storage management
';

UHEAP :: NONE;

USTK :: NONE;

INTX ← 1                /* 'SPACE INDICES FOR INT PAGES, ETC.';

REALX ← 2;

REFX ← 3;

DTPRX ← 4;

ATOMX ← 5;

DDBX ← 6;

STKX ← 7;

UHEAPX ← 8;
```

```
FLOOR\LOG <-
 EXPR(N:HWORD; INT)
  [) DECL I:INT; SHARED I ← 1 REPEAT N[I] => HWORD\SIZE - I END (];

UHBMAX ←
 FLOOR\LOG(HIPAG * PAGE\SIZE)
                        /* ' # BUCKETS FOR UHEAP FREE LISTS';

SMCODE ←
 CONST(VECTOR(UHEAPX, HWORD) OF
       INTX, REALX, REFX, DTPRX, ATOMX, DDBX, STKX, UHEAPX)
                        /* 'CODES USED IN QUANTUM MAP';

SMHEAD ←
 CONST(VECTOR(UHEAPX, STRUCT(FIRST\PAGE:SITE, FIRST\FREE:SITE)))
                        /* 'CURRENT BEGINNING OF EACH SPACE AND HEAD OF ITS FREELIST (SPECIA
L PACKING)';

BHEAD ←
 CONST(VECTOR(UHBMAX, SITE))
                        /* 'BUCKET HEADS FOR UHEAP FREE-LISTS';

SMFREE ←
 CONST(VECTOR(UHEAPX, INT))
                        /* '# REMAINING FREE WORDS';

SMRQST ← CONST(XWD)       /* 'STORES CURRENT SPACE REQUEST';

ZERO ← CONST(HWORD)       /* 'ZERO HALF-WORD, FOR TESTING';

NO\SITE ← CONST(SITE)     /* 'NULL SITE, FOR TESTING';

QMAP ←
 CONST(VECTOR(HIPAG, STRUCT(LINK:HWORD, CODE:MODE)))
                        /* 'QUANTUM MAP';

'LEFT HALFWORDS FOR BYTE POINTERS TO HALFWORD AND WORD OBJECTS';

LHPTR ← CONST(HWORD BYVAL 74880);

RHPTR ← CONST(HWORD BYVAL 1152);
```

```
WDPTR ← CONST(HWORD BYVAL 2304);

'Same integer goes into both halves of a WORD';

BHC <- EXPR(N:INT; XWD) CONST(XWD OF N, N);

↑;
```

```
'       PDP-10 machine language routines which are compiled to produce
"       CEXPRs.
';

PREFIX("@");

BLT <-
 EXPR(S:SITE, T:SITE, N:INT)
  BEGIN
   NORENT;
   HRLZ(A, @ X(NP) - 4);
   HRR(A, @ X(NP) - 2);
   MOVE(B, @ X(NP));
   ADDI(B, X(A) - 1);
   BLT(A, X(B));
   MOVEI(A, NONEB);
   SETZM(B);
   RET();
  END;

POP <-
 EXPR(P:XWD, S:ANY)
  BEGIN
   NORENT;
   MOVE(A, @ X(NP) - 2);
   POP(A, @ X(NP));
   MOVEM(A, @ X(NP) - 2);
   MOVEI(A, NONEB);
   SETZM(B);
   RET();
  END;

PUSH <-
 EXPR(P:XWD, S:ANY)
  BEGIN
   NORENT;
   MOVE(A, @ X(NP) - 2);
   PUSH(A, @ X(NP));
   MOVEM(A, @ X(NP) - 2);
   MOVEI(A, NONEB);
   SETZM(B);
   RET();
  END;
```

```
'       Use of the following routine is extremely dangerous.  It will
"       fetch the storage address of any ECL object.  If a compactification
"       which relocates the object occurs later, the SITE will no longer
"       be valid.
';

SITE\OF <-
 EXPR(X:ANY; SITE)
  BEGIN
   NORENT;
   HRRZ(A, RLITQ(SITE));
   HRRZ(B, X(NP));
   PUSH(VP, B);
   HRRZI(B, X(VP));
   HRLI(B, 1152);
   RET();
  END;

'       This routine attributes a mode to the object starting at site
"       S.  Its intended use is to access words in CORE.
';

!! <-
 EXPR(M:MODE, S:SITE; ANY)
  [) NORENT; LDB(A, X(NP) - 2); LDB(B, X(NP)); HLL(B, X(A) + 5); RET() (];

INFIX("!!");

COMPILE <-
 EXPR()
  BEGIN
   COMPILE\CEXPRS = NOTHING => LOADB "CEXPRS";
   DECL F:FORM BYVAL QL(SITE\OF, !!, PUSH, POP, BLT);
   DUMPB = NOTHING -> LOADB "SYS:DUMPB";
   GETP = NOTHING -> [) LOADB "143:COMMON"; LOADB "143:PASS3" (];
   PUT("BLT", "OPD", MKFM(86528));
   DECL P3:ANY LIKE VAL(PASS3\HANDLE[1]);
   REPEAT
    DECL R:ANY LIKE VAL(F.CAR.TLB);
    R ← P3(R);
    (F ← F.CDR) = NIL => NOTHING;
   END;
   "@".TLB ← "!!".TLB;
   "@".SBLK ← "!!".SBLK;
   FLUSH("!!");
   DUMPB("CEXPRS", QL(@, SITE\OF, PUSH, POP, BLT), 1);
   VAL(PASS3\HANDLE[2])();
   FLUSH(DUMPB);
  END;
```

```
COMPILE();

↑;
```

```
'       GARBAGE COLLECTOR
"
"       Reclaim heap storage not in use by any path of the current job.
"
"       Arguments:
"
"               A/      # words needed
"               B/      Index of space requesting more core
"
';

GCOL ←
 EXPR()
  BEGIN
   SMRQST.LH ← A.RH          /* 'SAVE REQUEST SIZE AND SPACE INDEX';
   SMRQST.RH ← B;
   FASTGC() => NOTHING             /* 'TRUE IF SPACE OBTAINED';
   ERROR1('GC NOT IMPLEMENTED YET');
  END;

'       FASTGC attempts to get page of a space without doing a regular
"       garbage collection if less than
"       without an intervening GC.  If it fails, then a regular GC will
"       be invoked.  Otherwise, the new page is linked into the freelist
"       for that space.
';

FASTGC ←
 EXPR(; BOOL)
  BEGIN
   A ←
    (A + PAGE\SIZE - 1) /
     PAGE\SIZE               /* 'ROUND UP TO # PAGES';
   A GT HIPAG - MAXPAG =>
    FALSE              /* 'NO WAY, CHARLIE';
   B ← SMCODE[B]             /* 'GET SPACE CODE';
   NOT GETPAG() => FALSE   /* '  AND TRY TO GET A PAGE';
   B ← SMRQST.RH;
   SMFREE[B] ← SMFREE[B] - (XWD @ A).LH;
   BEGIN
    DECL X:INT LIKE B;
    X = UHEAPX => X ← COUNT((XWD @ A).LH) + UHEAPX;
    X = DDBX => NOTHING;
    A ← B;
    GCLKWD()                 /* 'LINK WORDS';
   END;
   DECL H:ANY BYVAL
    SMHEAD[X]                /* 'GET OLD HEAD';
   SMHEAD[X] ← B             /* '  AND STORE NEW ONE';
   (XWD @ S).RH ← B          /* '  AND PUT OLD HEAD AT END';
   TRUE;
 END;
↑;
```

```
'        DATA SPACE EXTENSION ROUTINES
"
"        Expand data spaces to meet required or desired minima.
';

'        Satisfy hard request for core.  Request sorted in SMRQST as
"        <size of block,,space index>.  Returns TRUE if request is
"        satisfied.
';

GCHARD ←
 EXPR(; BOOL)
  QL("EXIT", BOOL) <<
   BEGIN
    DECL N:ANY SHARED A    /* 'SIZE OF BLOCK NEEDED';
    DECL SPACE:ANY SHARED
     B                     /* 'SPACE INDEX';
    DECL K:ANY SHARED D    /* 'ANOTHER COUNT';
    N ← SMRQST.LH;
    SPACE ← SMRQST.RH;
    K ← SMFREE[B]          /* 'INITIALLY TOTAL FREE COUNT FOR SPACE';
    /* 'FIRST, TRY TO USE AN EXISTING SPACE';
    N GT K ->
     BEGIN
      DECL BLOCK:ANY SHARED
       C                   /* 'SITE OF NEXT BLOCK TO TRY';
      BLOCK ←
       SMHEAD[SPACE].FIRST\FREE
                           /* 'START AT HEAD OF FREE LIST';
      ANY <<
       REPEAT
        BLOCK.RH = NO\SITE ->
         RETURN()          /* 'AT END OF LIST';
        K ←
         (XWD @ BLOCK).LH  /* 'SIZE OF THIS BLOCK';
        N LE K ->
         RETURN(TRUE, "EXIT")
                           /* 'FOUND IT!';
        BLOCK ←
         (XWD @ BLOCK).RH  /* 'ADVANCE IN LIST';
       END;
     END;
    /* 'FAIL...GET ENOUGH NEW PAGES';
    N ←
     (N + PAGE\SIZE - 1) /
      PAGE\SIZE            /* 'ROUND UP TO NEXT PAGE';
    SPACE ← SMCODE[SPACE]  /* 'CODE FOR GETPAGE';
    NOT GETPAGE() ->
     RETURN(FALSE)         /* 'NOT ENOUGH PAGES THERE';
    SPACE ← SMRQST;
    SMFREE[SPACE] ←
 SMFREE[SPACE] + N      /* 'ADD # WORDS GOTTEN';
GCSIFT()                /* 'SIFT INTO FREE LIST';
RETURN(TRUE);
```

```
  END;
↑;
```

```
'       Sift new blocks into the free list of a space.
"
"       Assumes free list is in descending order of site.  Sorts new
"       blocks into the free list, merging new blocks with any old
"       blocks found to be contiguous.
"
"       Register usage:
"
"       A/      new block site
"       B/      space index (returned unchanged)
';

GCSIFT ←
 EXPR()
  BEGIN
   DECL PRED:SITE                  /* 'SUPER-RETARDED POINTER';
   DECL RET:SITE BYVAL
    SMHEAD[B]                      /* 'RETARDED POINTER';
   DECL ADV:SITE BYVAL
    XWD @ RET                      /* 'ADVANCED POINTER';
   DECL BLOCK:ANY SHARED A /* 'CURRENT BLOCK';
   /* 'LINK THE NEW BLOCK INTO THE FREE LIST (IN DESCENDING ORDER BY SITE)';
   (INT <<
    REPEAT
     ADV = NO\SITE ->
      RETURN(0)            /* 'GOES AT THE END OF THE LIST';
     /* 'IF BLOCK BELOW ADV, THEN STEP DOWN LIST';
     BLOCK.RH LT ADV => [) PRED ← RET; RET ← ADV; ADV ← XWD @ ADV (];
     /* 'INSERT NEW BLOCK INTO FREE LIST AND TRY TO MERGE WITH SUCCESSOR';
     DECL S:SITE BYVAL
      (XWD @ ADV).LH +
       (XWD @ ADV).RH              /* 'SITE OF WORD AFTER BLOCK';
     S # ADV -> RETURN(0)          /* 'RETURN ZERO IF NOT CONTIGUOUS';
     /* 'MERGE BLOCK WITH SUCCESSOR';
     (XWD @ ADV).LH ←
      (XWD @ ADV).LH +
       (XWD @ BLOCK).LH           /* 'NEW SIZE OF BLOCK';
     BLOCK ← ADV;
     ADV ← XWD @ ADV               /* 'GET SUCCESSOR, SINCE ABSORBED';
     RETURN(1)                     /* 'RETURN, INDICATING NO RELINKING NECESSARY';
    END) = 0 ->
    (XWD @ BLOCK).RH ← ADV /* 'LINK NEW BLOCK TO SUCCESSOR';
   /* 'TRY TO MERGE BLOCK INTO ITS PREDECESSOR';
   BEGIN
    RET # (XWD @ BLOCK).LH + (XWD @ BLOCK).RH =>
     PRED ← RET            /* 'IF CANNOT MERGE';
    BLOCK.LH ←
     (XWD @ RET).LH +
      BLOCK.LH                     /* 'EXTEND NEW BLOCK';
   END;
   /* ((XWD @ PRED).RH ←
     BLOCK.RH          /* 'LINK TO NEW BLOCK');
END;
```

```
'       Link each space into form for efficient heap generation
"
"       INT,REAL,REF,DTPR linked by word in ascending order
"       DDB,ATOM variable-sized blocks linked in ascending order
"       UHEAP variable-sized blocks sorted by size into HWORD\SIZE
"       ascending lists.
';

GCLINK ← EXPR() FOR I TO UHEAPX REPEAT A ← I; SMLINK[I]() END;

'       Link routines for each space
"
"               A/      Space index
';

GCLKIR ←
 EXPR()
  BEGIN
   B ← SMHEAD[A].FIRST\FREE;
   DECL WL:SITE        /* 'INITIAL WORD LIST';
   ANY <<
    REPEAT
     NOT GCLKWD() =>
      RETURN()                  /* 'DO IT TILL DONE';
     XWD @ (D - 1) ← W          /* 'LINK TO NEXT GROUP';
     W ← B              /* 'CURRENT BLOCK SITE IS LINK';
     B ← E              /* 'GET NEXT BLOCK';
    END;
   SMHEAD[A].FIRST\FREE ←
    W                           /* 'PLANT NEW FREE LIST';
  END;

GCLKAT ←
 EXPR()
  BEGIN
   DECL X:INT BYVAL A           /* 'SPACE INDEX';
   A ← SMHEAD[A].FIRST\FREE;
   GCRVRS()                     /* 'REVERSE THE LIST TO BE ASCENDING';
   SMHEAD[X].FIRST\FREE ←
    B                   /* 'SET NEW LIST HEAD';
  END;
```

```
GCLKUH ←
 EXPR()
  BEGIN
   FOR I TO UHBMAX
    REPEAT
     BHEAD[I] ← NO\SITE    /* 'SET ALL FREE LISTS TO NIL';
    END;
   DECL NEXT:SITE BYVAL B;
   ANY <<
    REPEAT
     NEXT = NO\SITE => NOTHING;
     DECL HD:XWD SHARED XWD @ NEXT;
     NEXT ← HD.RH;
     DECL I:INT LIKE COUNT(HD.LH);
     HD.RH ← BHEAD[I];
     BHEAD[I] ← SITE\OF(HD);
    END;
  END;

'        Internal routine to link words of a variable length block.
"
"        B/      Site of block
"        Returns TRUE unless B contains NO\SITE on entry.
"        B/      Unchanged
"        D/      Site of word just beyond block
"        E/      Value of block link field
';

GCLKWD ←
 EXPR(; BOOL)
  BEGIN
   B = NO\SITE => FALSE;
   E ← B                 /* 'BLOCK LINK FIELD';
   D ← B.RH + 1;
   TO B.RH - 1 REPEAT XWD @ D ← D; D ← D.RH + 1 END;
   TRUE;
  END;

'        Reverse a linked list of blocks
"
"        A/      Input list head...new list head returned in A
';
```

```
GCRVRS <-
 EXPR()
  BEGIN
   DECL J1:SITE;
   XWD <<
    REPEAT
     A.RH = NO\SITE => A ← J1;
     J1.LH ← (XWD @ A).LH  /* 'SIZE OF NEW BLOCK,,SITE OF OLD BLOCK';
     DECL T:ANY BYVAL XWD @ A;
     XWD @ A ← J1;
     J1 ← A;
     A ← T;
    END;
  END;

ERROR1 <-
 EXPR(S:STRING) BREAK([) S = ” => 'AN IMPOSSIBLE ERROR HAS OCCURRED'; S (]);

SMLINK ←
 CONST(VECTOR(UHEAPX, ROUTINE) OF
       GCLKIR, GCLKIR, GCLKIR, GCLKIR, GCLKAT, GCLKAT, ERROR1, GCLKUH);
```