TR 79-009

REDMACH PROGRAM LOGIC MANUAL

by

Frances T. Kerr

University of North Carolina
Chapel Hill
May, 1979

Appendix C to
SIMULATION OF A REDUCTION MACHINE

TABLE CF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

This manual describes the module division, external data structures, subroutines, and some of the algorithms used in REDMACH, a system which simulates a reduction machine. We assume knowledge of the reduction machine and reduction languages. For more information about them, we refer the reader to the master's thesis, "Simulation of a Reduction Machine".

The program documented in this manual is not a completed system. It runs in batch mode and reads a set of user options that cannot be changed during execution. It gathers no statistics.

In Chapter 2 we describe the module division of REDMACH. In Chapter 3 we list the external data structures used in REDMACH. In Chapter 4 we list all the internal and external procedures and entry points. There are some difficult sections of code in REDMACH. In the Chapter 5, we explain what these sections do.

Throughout this manual, when an internal procedure is listed, it is enclosed in parentheses. In the module section, members are listed in order of their static nesting. That is, if a list includes

A,(B,(C))

it means that A is an external procedure, B is internal to A, and C is internal to B.

REDMACH was written in PL/I and was compiled using the PL/I Optimizing Compiler, Version 1, Release 3.0, PTF 64, at TUCC, the Triangle Universities Computation Center, under IBM OS/360, with MVT, Release 21.8. The catalogued procedure that executes REDMACH is stored in

UNC.CS.F233S.KERR.REDS.CNTL(REDMACH).

The contents of this procedure are

```
//REDMACH   PROC OPTIONS=
//STEP1     EXEC PGM=REDMACH,REGION=500K,PARM='/&OPTIONS'
//STEPLIB   DD   DSN=UNC.CS.F233S.KERR.REDS.LOAD,DISP=SHR
//CRT       DD   DUMMY
//SYSPRINT  DD   SYSOUT=A
//TABLES    DD   DSN=UNC.CS.F233S.KERR.TABLES.DATA,DISP=SHR
//INPUT     DD   DSN=&INPUT,DISP=SHR
//          DD   DSN=UNC.CS.F233S.KERR.MASTER.DATA,DISP=SHR.
```

The symbolic parameter &OPTIONS in the operand of the keyword parameter PARM can be specified by the user as a keyword parameter and operand (OPTIONS='option string') on his EXEC card. If he does not specify this parameter, the default value is null. The PARM operand is passed to the main procedure of REDMACH. The OPTIONS field on the user's EXEC card is a symbolic parameter in the parameter PARM, which is passed to the main procedure. Dataset UNC.CS.F233S.KERR.REDS.LOAD(REDMACH) contains the REDMACH load module. Dataset UNC.CS.F233S.KERR.TABLES.DATA contains the system tables (micro-opcodes and formats, registers, and constants). This dataset must be unnumbered. INPUT is the DD name of the input dataset. The user supplies the dataset name of his input dataset as a symbolic parameter on his EXEC card. Dataset UNC.CS.F233S.KERR.MASTER.DATA is the master library of definitions and microprograms. It is catenated to the user's input dataset.

Chapter 2

MODULE DIVISION

We use the term "module" to denote one or more subroutines that together perform a clearly defined function. REDMACH is composed of eight modules. They are listed in Table 1 with the external subroutines that belong to each module. Figure 1 shows the connections among the modules. Only the Main Control module can invoke the modules on the level beneath it. The two utility modules shown on the lowest level can be accessed by any other module.

TABLE 1

The Modules of REDMACH and Their Members

Module	Procedures
Main Control Module	REDMACH
Input Module	SETUP, SETOPTS
Update List of RA's	FINDRAS
Output Module	PRINTL, PRTNSTS, TALK
Interpreter Module	RAS, PRIM, ASSIGN, ARITH, SEND, COMPARE, POP, SETREG, INSERT
Storage Management Module	STORAGE
Symbol Table Manager	HASHSYM
Error Message Routine	ABEND

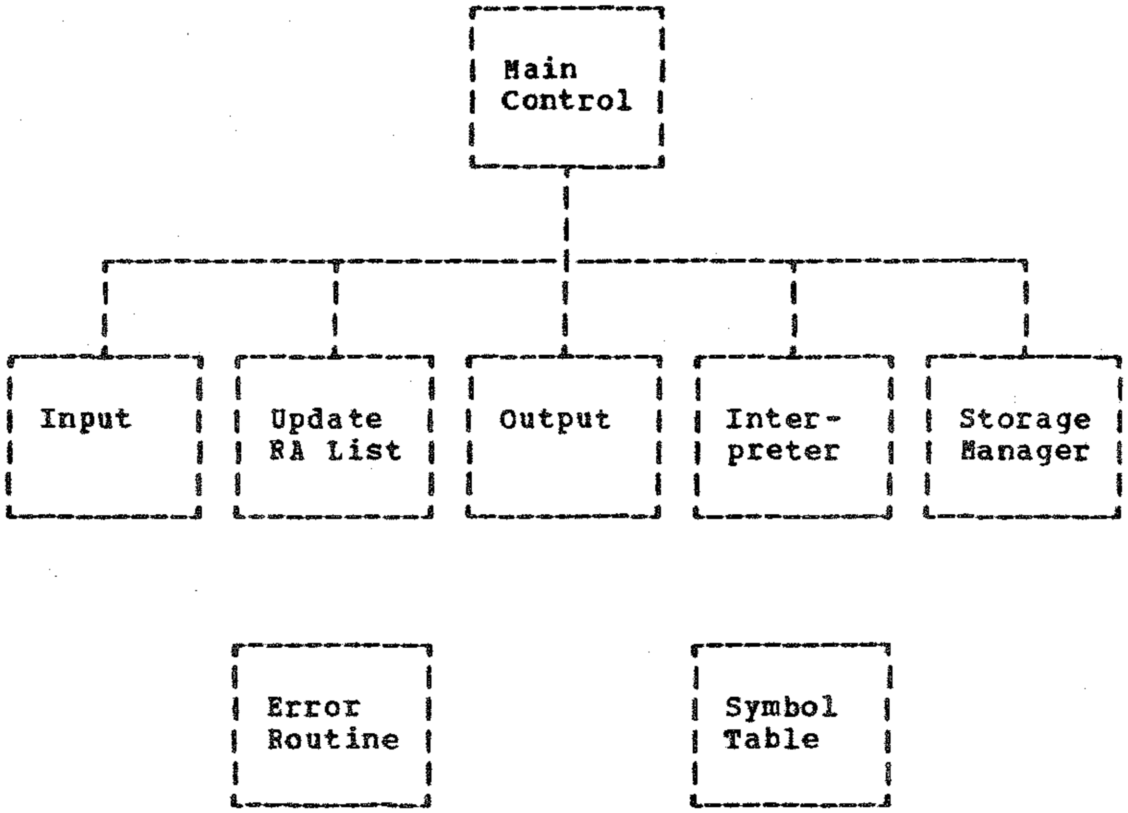

Figure 1: The Modules of REDMACH

2.1 MODULE DESCRIPTIONS

Main Control Module

PURPOSE: Call procedures to execute the interpreter system.

MEMBERS: REDMACH.

Input Module

PURPOSE: Read initial user options; read programs to be executed; initialize L array; read and translate to internal representation the defined operators and microprograms for primitives needed for execution.

MEMBERS: SETUP, (INIT), (GETOPS, (GETDEF, GETPRIM, (EXORDER, INSERT))), (GETPROG), (SCANTOK, (GETCHAR, NEWTOK)), (GETOPTS), SETOPTS.

Update List of RA's

PURPOSE: Find new Reducible Applications (RA's); build syntax trees; determine class and status of new RA's; determine top and height of areas of new and old RA's.

MEMBERS: FINDRAS, (PARSE), (FINDTOP).

Output Module

PURPOSE: Print the L array.

MEMBERS: PRINTL, PRNTSTS, TALK.

Interpreter Module

PURPOSE: Reduce or request storage for all RA's in L; compute time available for data movement.

MEMBERS: RAS, (INITRAS), (RAFINI), (DMSTAT), PRIM, (EXSEG, (EXSTMT)), ASSIGN, ARITH, INSERT, SEND, (ARITHOP), (BOOLOP), (MINMAX), POP, COMPARE, (EXCOMP), SETREG.

Storage Management Module

PURPOSE: Cancel storage requests when necessary; perform storage management on L.

MEMBERS: STORAGE, (HALF1, (CANCEL)), (HALF2), (MOVEIT), (RAFINI).

Error Message Routine

PURPOSE: When any error is encountered, print an error message and terminate abnormally.

MEMBERS: ABEND.

Symbol Table Manager

PURPOSE: Enter symbols and numbers in symbol table; determine symbol table address of a symbol.

MEMBERS: HASHSYM, HASHVAL, SEARCH, (SCANTAB), (INSERT).

Chapter 3

EXTERNAL DATA STRUCTURES

This chapter descibes the external data stuctures used by REDMACH. It includes PL/I structures and arrays, fixed and float binary variables, pointers, and files.

3.1 STRUCTURES AND ARRAYS

L Array

```
1 L(*)          CTL EXT,     /* L ARRAY */
   2 S          FIXED BIN    /* SYMBOL TABLE ADDRESS OF S */
   2 ALN        FIXED BIN,   /* ABSOLUTE LEVEL NUMBER */
   2 RLN        FIXED BIN,   /* RELATIVE LEVEL NUMBER */
   2 NEWPOS     FIXED BIN,   /* CELL'S INDEX IN L AFTER */
                             /* STORAGE MANAGEMENT */
   2 S2         FIXED BIN,   /* SYMBOL TABLE ADDRESS OF S' */
                             /* (SYMBOL AFTER REDUCTION) */
   2 RLN2       FIXED BIN,   /* RLN AFTER REDUCTION */
   2 STATUS     FIXED BIN,   /* CELL'S STATUS */
   2 IRSR       FIXED BIN,   /* INSERTION REQUESTS TO */
                             /* CELL'S RIGHT */
   2 IRSL       FIXED BIN,   /* INSERTION REQUESTS TO */
                             /* CELL'S LEFT */
   2 SNAPIT     FIXED BIN,   /* FLAG IF SYMBOL CAUSES A */
                             /* 'SNAP' */
```

Structure simulates the cells of L. It is indexed from SIZE to 2*SIZE. Values of S and S2 are symbol table addresses. S2 is the address of the symbol printed by the output module; it is the cell's contents after reduction is complete, that is, after data movement is over. ALN is absolute level number; RLN is relative level number; RLN2 is relative level number after reduction is complete; IRSL and IRSR are the number of insertion requests to the left and right of a cell; NEWPOS is a cell's index in L after storage management (BL and BR are not necessary); STATUS shows what situation a cell is in. Size of L is determined by SIZE option.

Used by: ABEND, FINDRAS, INSERT, PRIM, PRINTL, RAS, SEND,
SETREG SETUP, STORAGE.

Reducible Applications

```
COUNT               FIXED BIN,      /* NUMBER OF SYMBOLS IN RA */
1 RA                BASED(P1),      /* LIST OF RA'S */
   2 OPIND          FIXED BIN,      /* INDEX OF OPERATOR IN L */
   2 TOP            FIXED BIN,      /* RA'S TOP IN T */
   2 HT             FIXED BIN,      /* HEIGHT OF AREA */
   2 IRSTOT         FIXED BIN,      /* TOTAL INSERTION REQUESTS */
   2 DML            FIXED BIN,      /* LENGTH OF DATA MOVEMENT */
   2 CLASS          FIXED BIN,      /* CLASS OF RA */
   2 MARK(5,2)      FIXED BIN,      /* TREE INDICES OF MARKED */
                                    /* EXPRESSIONS */
   2 #SYMS          FIXED BIN,      /* NUMBER OF SYMBOLS IN RA */
   2 TREE(COUNT REFER(#SYMS)),
                                    /* SYNTAX TREE */
      3 IND         FIXED BIN,      /* INDEX OF SYMBOL IN L */
      3 BRO         FIXED BIN,      /* INDEX OF SYMBOL'S BROTHER */
                                    /* IN TREE */
      3 SON         FIXED BIN,      /* INDEX OF SYMBOL'S SON IN */
                                    /* TREE */
   2 INS_FLAGS(15,COUNT REFER(#SYMS))
                    BIT(1) UNALIGNED,
                                    /* FLAGS WHERE INSERTIONS */
                                    /* WILL OCCUR */
   2 NEXTRA         POINTER,        /* NEXT RA IN LIST */
```

Structure holds information about each active Reducible Application (RA) in L. When a new RA is allocated, its fields are initialized and its syntax tree is built. Each node of the tree contains 3 items: the index in L of the non-empty cell of L it represents, the tree index of its son (next non-empty cell in L if it has a larger level number than the node; value is zero if no son exists), and the tree index of the node's brother or father (next non-empty cell of L with level number less than or equal to the node's level number).

Used by: ASSIGN, FINDRAS, INSERT, PRIM, RAS, SEND,
         SETREG, STORAGE.

Microprograms for Primitives

```
(N1,                            /* NUMBER OF DESTINATIONS */
 N2,                            /* NUMBER OF FRAGMENTS */
 N3)           FIXED BIN,       /* NUMBER OF INSTRUCTION */
                                /* BYTES */
1 PRIMITIVE    BASED(P6),       /* PRIMITIVE MICROPROGRAM */
   2 #DESTS    FIXED BIN,       /* NUMBER OF DESTINATIONS */
   2 DEST(N1 REFER(#DESTS),2)   /* DESTINATION LIST */
               FIXED BIN,
   2 #FRAGS    FIXED BIN,       /* NUMBER OF FRAGMENTS */
   2 FRAG(N2 REFER(#FRAGS),2)   /* FRAGMENT LIST */
               FIXED BIN,
   2 LENTOT    FIXED BIN,       /* NUMBER OF INSTUCTION */
                                /* BYTES */
   2 STMT(N3 REFER(LENTCT))     /* INSTRUCTION ARRAY */
               FIXED BIN,
   2 STARTIRS  FIXED BIN,       /* FRAGMENT # OF FIRST */
                                /* INSERT INSTRUCTION */
```

Each allocation contains a microprogram for a primitive operator. Destination expression list (DEST) contains one entry for each destination expression. Each destination contains 2 items: a 0 (for S) or 1 (for E) and RLN. Fragment list (FRAG) contains one entry for each fragment. Each fragment contains 2 items: a destination number (index in destination list) and a starting instruction counter value. The instruction array (STMT) consists of all the microprogram bytes, and is sorted in execution order. The next instruction is located by incrementing an instruction counter the length of the last instruction executed. Each copy is allocated and initialized when primitives are read from the input file. The address of a copy is stored as a PL/I pointer in the primitive's copy of OPTAB.

USED BY: ARITH, ASSIGN, COMPARE, INSERT, PRIM,
         SEND, SETUP.

Symbol Table

```
1 SYMTAB(*)     CTL EXT,       /* SYMBOL TABLE */
   2 SYM        CHAR(8),       /* SYMBOL */
   2 USES       FIXED BIN,     /* NUMBER OF OCCURRENCES OF */
                               /* SYMBOL */
   2 NUMERIC    BIT(1),        /* FLAG IF SYMBOL IS A NUMBER */
   2 VAL        FLOAT BIN,     /* NUMERIC VALUE OF ATOM */
   2 OPPTR      POINTER,       /* POINTER TO OPERATOR TABLE */
```

All program symbols are stored in this table. Table is indexed (0:TABSIZE+1). Constants are read from a file (TABLES), so they can be changed without recompiling any programs. Addresses of symbols are determined by hashing.

Used by: ARITH, ASSIGN, COMPARE, FINDRAS, HASHSYM, INSERT,
         PRIM, PRINTL, RAS, SEND, SETUP, STORAGE.

Defined Operator Table

```
N               FIXED BIN,       /* LENGTH OF DEFINITION */
1 DEFTAB        BASED(P5),       /* DEFINITION TABLE */
   2 DEFLEN     FIXED BIN,       /* LENGTH OF DEFINITION */
   2 DEFINITION(N REFER(DEFLEN)),
      3 S       FIXED BIN,       /* SYMBOL */
      3 RLN     FIXED BIN,       /* RELATIVE LEVEL NUMBER */
      3 SNAPIT  FIXED BIN,       /* FLAG FOR SNAPSHOTS */
```

Table contains definitions of all user-defined operators. The length of the definition is followed by the definition in internal representation.

Used by: RAS, SETUP.

Operator Table

```
1 OPTAB         BASED(P4),   /* TABLE OF OPERATORS */
   2 CLASS      FIXED BIN,   /* CLASS OF OPERATOR */
   2 MPBITS     FIXED BIN,   /* NUMBER OF MICROPROGRAM BITS */
                             /* FOR PRIMITIVES, DEFINITION */
                             /* LENGTH FOR DEFINED OPERATORS */
   2 MPCALLED   BIT(1),      /* FLAG IF MICROPROGRAM HAS */
                             /* ALREADY BEEN CALLED */
   2 DEFPTR     POINTER,     /* POINTER TO DEFINITION TABLE */
                             /* OR PRIMITIVE MICROPROGRAM */
   2 NEXTOP     POINTER,     /* NEXT OPERATOR IN LIST */
```

Each operator (primitive and defined) owns one copy. Contains information about the operator and a pointer to its definition or microprogram.

Used by: FINDRAS, RAS, SETUP.

Messages

```
1 MESSAGE        CTL EXT,      /* MESSAGES AND MESSAGE */
                               /* CONTROLS */
   2 MSTART(2)   FIXED BIN,    /* STARTING TREE INDEX OF */
                               /* MESSAGE */
   2 MEND(2)     FIXED BIN,    /* LAST TREE INDEX OF MESSAGE */
   2 MESS#(2)    FIXED BIN,    /* INDEX OF MESSAGE */
   2 MESS(2,4,#SYMS)           /* MESSAGES */
                 FLOAT BIN,
                               /* INDICES FOR INSTRUCTIONS */
                               /* REFERENCING MESSAGES: */
   2 M#          FIXED BIN,    /* INDEX OF MESSAGE REFERENCED */
   2 PARM#       FIXED BIN,    /* OPERAND NUMBER OF MESSAGE */
                               /* COMPONENT REFERENCED */
   2 MIND        FIXED BIN,    /* INDEX OF MESSAGE IN MESSAGE */
                               /* CONTROL (1 OR 2) */
   2 MESSFLAG(2,#SYMS)         /* FLAG IF MESSAGE WAS SENT */
                 BIT(1) UNALIGNED,
   2 SFLAG(3)    BIT(1),       /* FLAG IF COMPARISON USED */
                               /* MESSAGES */
   2 MFLAG(3)    BIT(1),       /* FLAG IF COMPARISON WAS */
                               /* SIMPLE */
   2 SMESS(2,0:4)              /* MESSAGE COMPONENTS THAT ARE */
                 BIT(1),       /* SYMBOL TABLE ADDRESSES */
```

Allocated before reducing an RA to keep track of two rounds of messages (necessary because a new message may be started before the last can be erased). Messages are stored in the array MESS; its first dimension is to keep two rounds of messages; its second corresponds to the component number of the four possible message components; its third corresponds to the indices of the messages themselves - the third dimension indices of messages sent by an SI instruction are the same as the tree indices of the target expression which sent them;. these indices are stored in MSTART and MEND. The single result of a component of an SC is stored in the entire cross section corresponding to the component number; MSTART and MEND for an SC are 1. MESS# contains the indices of up to two rounds of messages; e.g., the messages from sendi have MESS#=i. MESSFLAG indicates whether a cell sent a message or not. M#, MIND and PARM# refer to a current instruction referencing a message that has already been sent.

Used by: ARITH, ASSIGN, COMPARE, POP, PRIM, SEND

Register Table

```
    REGTAB(*)       CHAR(6) CTL EXT, /* TABLE OF REGISTERS */
```

Table of registers. Each element is the mremonic name of a register and is initialized from system tables file (TABLES). Indexed (NOREG:1).

Used by: ARITH, ASSIGN, COMPARE, INSERT, PRIM, SEND,
         SETREG, SETUP

Micro Opcode Table

```
1 MICRTAB(*)    CTL EXT,          /* TABLE OF MICRO-OPCODES */
   2 MNEM       CHAR(4),          /* MNEMONIC OPCODE */
   2 LEN        FIXED BIN,        /* INSTRUCTION LENGTH */
   2 R          FIXED BIN,        /* REGISTER POSITIONS IN */
                                  /* INSTRUCTION */
```

Table of micro opcodes. Initialized from system tables file (TABLES). R indicates what fields in an instruction with this opcode contain register values.

Used by: ARITH, ASSIGN, COMPARE, INSERT, PRIM, SEND, SETUP

Program List

```
    PROG(10)        FIXED BIN EXT,  /* LIST OF USER PROGRAMS */
```

List of indices in L of first symbol of each user prograr being executed. Not used in current version; will be used to gather statistics.

Used by: SETUP, STORAGE

3.2 FIXED BINARY EXTERNAL

Status variables:

EMPTY Cell is empty; also is symbol table address of blank.

Used by: ARITH, FINDRAS, INSERT, PRIM, RAS, SEND, SETREG, SETUP, STORAGE.

NOTRA Cell is not in an RA.

Used by: FINDRAS, PRINTL, RAS, SETUP, STORAGE.

REQUEST Cell is in RA ready to request storage.

Used by: FINDRAS, PRINTL, RAS, SETUP.

REDUCE Cell is in RA that is ready to be reduced.

Used by: FINDRAS, PRINTL, RAS, SETUP.

CANCLED Cell is in RA whose insertion requests were canceled.

Used by: FINDRAS, PRINTL, RAS, SETUP, STORAGE.

Data Movement Status Variables:

EBFA Cell was empty before data movement, full after data movement.

Used by: PRINTL, RAS, SETUP.

FBEA Cell was full before data movement, empty after data movement.

Used by: PRINTL, RAS, SETUP.

FBFA Cell was full before data movement, full after data movement.

Used by: PRINTL, RAS, SETUP.

RA Class variables:

ACLASS Class A: RA requires no storage management and no data movement.

Used by: FINDRAS, RAS, SETUP.

BCLASS Class B: RA requires data movement but no storage management.

Used by: FINDRAS, RAS, SETUP.

CCLASS Class C: RA requires both storage management and data movement.

Used by:FINDRAS, RAS, SETUP.

DEFCLAS RA's operator is a user-defined operator.

Used by: FINDRAS, RAS, SETUP.

METCLAS RA has composite operator and requires meta composition.

Used by: FINDRAS, RAS, SETUP.

Program Constants:

APPL Symbol table address of application symbol; symbol and address are initialized from system tables file (TABLES).

Used by: FINDRAS, SETUP.

DDTOT Number of output files.

Used by: REDMACH, SETUP.

FALSE Symbol table address of 'F'; initialized from system tables file (TABLES).

Used by: SEND, SETUP.

LENL Index of rightmost cell of L array; initialized from value of SIZE.

Used by: FINDRAS, RAS, SETUP, STORAGE.

MESSREG Index of first message register; initialized as NOREG-1.

Used by: COMPARE, PRIM, SETUP.

NOREG Index that indicates no register for microprogram instruction register fields; initialized from number of registers when REGTAB is initialized.

Used by: COMPARE, HASHSYM, PRIM, SEND, SETREG, SETUP.

PAREN Symbol table address of sequence symbol; initialized from system tables file (TABLES); note that name does not correctly match current syntax -- this is for historical reasons.

Used by: FINDRAS, RAS, SETUP.

POS# Index of POS# in REGTAB.

Used by: INSERT, SETUP.

ROOT Index of root cell of T.

Used by: FINDRAS, RAS, SETUP, STORAGE.

SIZE Number of cells of L; T is indexed (ROOT:SIZE-1); L is indexed (SIZE:2*SIZE); initialized from user options.

Used by: FINDRAS, PRINTL, RAS, SETOPTS, SETUP, STCRAGE.

TABSIZE High bound of symbol table minus 1; initialized from user options; value should be a prime number because hashing algorithm uses it to divide to get symbol table addresses.

Used by: HASHSYM, SETOPTS, SETUP.

THT Height of T; initialized from value of SIZE.

Used by: FINDRAS, RAS, SETUP.

TRUE Symbol table address of 'T'; initialized from system tables file (TABLES).

Used by: SEND, SETUP.

ZS Symbol table address of 'ZZZZZZZZ'; initialized from system tables file (TABLES); used to initialize a message to 'infinity'.

Used by: SEND, SETUP.

3.3 FLOAT BINARY EXTERNAL

P Percent of L that must always remain empty; initialized from user options.

Used by: RAS, SETOPTS, SETUP, STORAGE.

TBIT Time it takes to move a bit through the root of T; initialized from user options.

Used by: RAS, SETOPTS, SETUP.

TLEV Time it takes to move an atom one level in T; initialized from user options.

Used by: RAS, SETOPTS, SETUP.

3.4 POINTER EXTERNAL

OPHEAD Head of operator list (see OPTAB).

Used by: RAS, SETUP.

RATAIL Tail of RA list (see RA).

Used by: FINDRAS, RAS, STORAGE.

3.5 FILE EXTERNAL

CRT DD name of terminal display; not used in current system.

Used by: SETUP.

INPUT DD name of input libraries.

Used by: SETUP.

SYSPRINT DD name of output print file.

Used by: SETUP, PRINTL.

TABLES DD name of system tables file; contains registers for REGTAB, constants for SYMTAB, and micro instruction opcodes and formats for MICRTAB.

Used by: SETUP.

Chapter 4

PROCEDURES

ABEND (MESSAGE)

MODULE: Error Message Module.

SCOPE: External.

PARAMETERS: MESSAGE CHAR(*);

PURPOSE: Print an error message and abend.

METHOD: This subroutine is not fully implemented; it receives a character string as a parameter, prints it, and stops execution.

CALLED BY: Almost every subroutine.

CALLS: None.

ARITH (IC, TREEIND, RT, TOP, P6, OPERAND, COND, MATCH, GRO)

MODULE: Interpreter Module.

SCOPE: External.

PARAMETERS:

IC FIXED BIN;
Instruction counter.

TREEIND FIXED BIN;
Leftmost syntax tree index of target expression.

RT FIXED BIN;
Rightmost syntax tree index of target expression.

TOP FIXED BIN;
Index of top entries in COND and MATCH stacks.

P6 POINTER;
Base pointer for PRIMITIVE data structure.

OPERAND(*,*) FLOAT BIN;
Instruction's operands.

COND(*,*) BIT(*) UNALIGNED;
Results of previous comparisons for each syntax tree node in target expression.

MATCH(*,*,*) BIT(*) UNALIGNED;
Results of previous comparisons involving messages for each syntax tree node in target expression and each syntax tree node of message (see ALGORITHMS).

GRO(*) FLOAT BIN;
Register of temporary results.

PURPOSE: Execute an arithmetic microinstruction.

METHOD: Loop through cells of target expression; for each cell, perform operation specified by instruction's opcode and store result in temporary register.

CALLED BY: (EXSTMT).

CALLS: None.

ARITHOP

MODULE: Interpreter Module.

SCOPE: Internal to SEND.

PARAMETERS: None.

PURPOSE: Execute a send-and-combine instruction with arithmetic combining operator.

METHOD: Loop through each cell of target expression; combine operands of each cell with messages according to combining operator.

CALLED BY: SEND.

CALLS: None.

ASSIGN (TREEIND, RT, IC, P1, P6, OPERAND, COND, MATCH, TOP, GRO)

MODULE: Interpreter Module.

SCOPE: External.

PARAMETERS:

TREEIND FIXED BIN;
Leftmost syntax tree index of target expression.

RT FIXED BIN;
Rightmost syntax tree index of target expression.

IC FIXED BIN;
Instruction counter.

P1 POINTER;
Base pointer for RA data structure.

P6 POINTER;
Base pointer for PRIMITIVE data structure.

OPERAND(*,*) FLOAT BIN;
Instruction's operands.

COND(*,*) BIT(*) UNALIGNED;
Results of previous comparisons for each syntax tree node in target expression.

MATCH(*,*,*) BIT(*) UNALIGNED;
Results of previous comparisons involving messages for each syntax tree node in target expression and each syntax tree node of message (see ALGORITHMS).

TOP FIXED BIN;
Index of top entries in COND and MATCH stacks.

GRO (*) FLOAT BIN;
Register of temporary results.

PURPOSE: Execute an assign microprogram instruction.

METHOD: Loop through target expression in syntax tree; make assignments to S or RLN.

CALLED BY: (EXSTMT).

CALLS: HASHVAL, ABEND.

BOOLOP

MODULE: Interpreter Module.

SCOPE: Internal to SEND.

PARAMETERS: None.

PURPOSE: Execute a send-and-combine instruction with Boolean combining operator.

METHOD: Loop through each cell of target expression; combine operands of each cell with messages according to combining operator.

CALLED BY: SEND.

CALLS: None.

CANCEL

MODULE: Storage Management Module.

SCOPE: Internal to (HALF1).

PARAMETERS: None.

PURPOSE: Cancel storage requests.

METHOD: Move down through T cancelling areas' requests until all remaining requests can be satisfied.

CALLED BY: (HALF1).

CALLS: None.

COMPARE (IC, TREEIND, RT, TOP, P6, OPERAND, COND, MATCH)

MODULE: Interpreter Module.

SCOPE: External.

PARAMETERS: IC FIXED BIN;
Instruction counter.

TREEIND FIXED BIN;
Leftmost syntax tree index of target expression.

RT FIXED BIN;
Rightmost syntax tree index of target expression.

TOP FIXED BIN;
Index of top entries in COND and MATCH stacks.

P6 POINTER;
Base pointer for PRIMITIVE data structure.

OPERAND(*,*) FLOAT BIN;
Instruction's operands.

COND(*,*) BIT(*) UNALIGNED;
Results of previous comparisons for each syntax tree node in target expression.

MATCH(*,*,*) BIT(*) UNALIGNED;
Results of previous comparisons involving messages for each syntax tree node in target expression and each syntax tree node of message (see ALGORITHMS).

PURPOSE: Execute a compare microprogram instruction.

METHOD: Determine whether comparing immediate operands or symbol table addresses; store results of comparison of each node of target expression in syntax tree in COND. See ALGORITHMS.

CALLED BY: (EXSTMT).

CALLS: ABEND, (EXCOMP).

DMSTAT

MODULE: Interpreter Module.

SCOPE: Internal to RAS.

PARAMETERS: None.

PURPOSE: Set status of cells in an RA undergoing data movement.

METHOD: Examine each cell in RA and set its status according to its contents before and after data movement.

CALLED BY: RAS.

CALLS: None.

EXCOMP

MODULE: Interpreter Module.

SCOPE: Internal to COMPARE.

PARAMETERS: None.

PURPOSE: Execute a comparison on elements of a target expression.

METHOD: For each node of in syntax tree of target expression, determine whether comparison is true or false and store result in COND and MATCH. See ALGORITHMS.

CALLED BY: COMPARE.

CALLS: None.

EXFRAG

MODULE: Interpreter Module.

SCOPE: Internal to PRIM.

PARAMETERS: None.

PURPOSE: Execute each microprogram instruction in a fragment.

METHOD: Initialize each instruction's operands; if before storage requests have been filled, execute all instructions; if after storage has been received, execute insert instructions only; increment instruction counter.

CALLED BY: PRIM.

CALLS: (EXSTMT), SETREG, ABEND.

EXORDER

MODULE: Input Module.

SCOPE: Internal to (GETPRIM).

PARAMETERS: None.

PURPOSE: Control when fragments are to be inserted in fragment list.

METHOD: Insert last fragment if not yet inserted. Insert current fragment.

CALLED BY: (GETPRIM).

CALLS: (INSERT).

EXSTMT

MODULE: Interpreter Module.

SCOPE: Internal to (EXFRAG).

PARAMETERS: None.

PURPOSE: Execute a microprogram instruction.

METHOD: GO TO a label that either executes the instruction or calls a subroutine to execute the instruction.

CALLED BY: (EXFRAG).

CALLS: COMPARE, INSERT, POP, ASSIGN, SEND, ARITH, ABEND.

FINDRAS (APPLTOT, SNAP)

MODULE: Update List of RA's.

SCOPE: External.

PARAMETERS: APPLTOT FIXED BIN;
Total number of applications in L.

SNAP BIT(*);
Flag indicating whether any operator in operator position has 'snap' flag enabled.

PURPOSE: Locate new innermost applications (RA's); build their syntax trees; determine their class and status; insert them in list of RA's. For each RA, find height and top of RA's area in T.

METHOD: A single scan of L in which Magó test for finding RA's is applied.

CALLED BY: REDMACH.

CALLS: (FINDTOP), (PARSE).

FINDTOP (PTR)

MODULE: Update List of RA's.

SCOPE: Internal to FINDRAS.

PARAMETERS: PTR POINTER;
Pointer to a member of the RA list.

PURPOSE: Compute top in T and height of area belonging to RA.

METHOD: Find indices in T of ancestor nodes of RA's application symbol, of the cell to its left, and of its right neighbor, until the lowest common ancestor is found.

CALLED BY: FINDRAS.

CALLS: None.

GETCHAR (NEWCHAR, CLASS)

MODULE: Input Module.

SCOPE: Internal to (SCANTOK).

PARAMETERS: NEWCHAR CHAR(*);
Character scanned.

CLASS FIXED BIN;
Lexical class of NEWCHAR.

PURPOSE: Find next character of token; determine its lexical class.

METHOD: Scan input card for next non-blank character; look up class in table of legal characters and corresponding lexical classes.

CALLED BY: (SCANTOK).

CALLS: None.

GETDEF

MODULE: Input Module.

SCOPE: Internal to (GETOPS).

PARAMETERS: None.

PURPOSE: Read and translate to internal representation the definition of a user defined operator.

METHOD: Scan each symbol in the definition; enter them in symbol table; calculate RLN's; enter symbol table addresses, RLN's, and 'snap' indicators in definition.

CALLED BY: (GETOPS).

CALLS: HASHSYM, (SCANTOK).

GETOPS

MODULE: Input Module.

SCOPE: Internal to SETUP.

PARAMETERS: None.

PURPOSE: Read and translate to internal representation all operators needed for execution.

METHOD: Read each operator in input file; if the operator is in the symbol table and is not yet defined, translate the definition or microprogram.

CALLED BY: SETUP.

CALLS: SEARCH, (GETDEF), (GETPRIM).

GETOPTS

MODULE: Input Module.

SCOPE: Internal to SETUP.

PARAMETERS: None.

PURPOSE: Initialize user options.

METHOD: Read in each option card, starting with options on EXEC card and then options in user's library; assign operand values to corresponding option variables if not yet specified.

CALLED BY: SETUP.

CALLS: SETOPTS.

GETPRIM

MODULE: Input Module.

SCOPE: Internal to (GETOPS).

PARAMETERS: None.

PURPOSE: Read and translate to internal representation a primitive operator's microprogram; sort the micro-instructions into execution order.

METHOD: Scan each input card for target expression and micro-instruction; translate components; insert instruction's fragment control information in execution order in fragment list; copy instructions in execution order to PRIMITIVE data structure. See ALGORITHMS.

CALLED BY: (GETOPS).

CALLS: SEARCH, (EXORDER).

GETPROG

MODULE: Input Module.

SCOPE: Internal to SETUP.

PARAMETERS: None.

PURPOSE: Read programs to be executed; translate to internal representation; initialize L array.

METHOD: Read each program selected by PROG option from user file; translate each symbol to symbol table address; initialize L cells; use scale control and blank count symbols for spacing control in L.

CALLED BY: SETUP.

CALLS: HASHSYM, (SCANTOK), ABEND.

HALF1

MODULE: Storage Management.

SCOPE: Internal to STORAGE.

PARAMETERS: None.

PURPOSE: Compute PT and NT values in T; cancel insertion requests as needed.

METHOD: One full cycle (up and down) in T. See ALGORITHMS.

CALLED BY: STORAGE.

CALLS: (CANCEL).

HALF2

MODULE: Storage Management.

SCOPE: Internal to STORAGE.

PARAMETERS: None.

PURPOSE: Compute new position of each cell of L.

METHOD: Move up in T computing new PT values; move down through T computing BL and BR values in cells of T, and then new positions of cells of L.

CALLED BY: STORAGE.

CALLS: None.

HASHSYM (TOKEN, ADDRESS)

MODULE: Symbol Table Manager.

SCOPE: External.

PARAMETERS: TOKEN CHAR(*); Token to be entered in symbol table.

ADDRESS FIXED BIN; Symbol table address of TOKEN.

PURPOSE: Enter a symbol in symbol table.

METHOD: Determine symbol's table address; insert symbol in table.

CALLED BY: (GETDEF), (GETPROG).

CALLS: (SCANTAB), (INSERT).

HASHVAL (VALUE, ADDRESS)

MODULE: Symbol Table Manager.

SCOPE: External entry point in HASHSYM.

PARAMETERS: VALUE FLOAT BIN;
Value to be entered in table.

ADDRESS FIXED BIN;
Symbol table address of VALUE.

PURPOSE: Convert numeric value to character string; enter in symbol table.

METHOD: Assign VALUE to picture variable; remove nonsignificant zeros to get unique character representation; determine symbol's address in table; insert symbol in table.

CALLED BY: ASSIGN.

CALLS: (SCANTAB), (INSERT).

INIT

MODULE: Input Module.

SCOPE: Internal to SETUP.

PARAMETERS: None.

PURPOSE: Initialize Micro-operation table, Register table, and symbol table.

METHOD: Read bounds of tables; allocate tables; read table values.

CALLED BY: SETUP.

CALLS: None.

INITRAS

MODULE: Interpreter Module.

SCOPE: External entry point in RAS.

PARAMETERS: None.

PURPOSE: Initialize branch mechanism for RAS; initialize list of RA's.

METHOD: Allocate and assignment.

CALLED BY: REDMACH.

CALLS: None.

INSERT (TREEIND, RT, IC, P1, P6, OPERAND, TOP, COND, TIMER, INS#)

MODULE: Interpreter Module.

SCOPE: External.

PARAMETERS:

TREEIND FIXED BIN;
Leftmost syntax tree index of target expression.

RT FIXED BIN;
Rightmost syntax tree index of target expression.

IC FIXED BIN;
Instruction counter.

P1 POINTER;
Base pointer for RA data structure.

P6 POINTER;
Base pointer for PRIMITIVE data structure.

OPERAND(*,*) FLOAT BIN;
Instruction's operands.

TOP FIXED BIN;
Index of top entries in COND and MATCH stacks.

COND(*,*) BIT(*) UNALIGNED;
Results of previous comparisons for each syntax tree node in target expression.

TIMER FIXED BIN;
Tells whether RA is before or after storage management.

INS# FIXED BIN;
Number of insert instruction within microprogram.

PURPOSE: Execute an insert microprogram instruction.

METHOD: If before storage management, determine number of cells to be inserted; if after storage management, locate cells to be inserted and insert them.

CALLED BY: (EXSTMT).

CALLS: SETREG, ABEND.

INSERT

MODULE: Input Module.

SCOPE: Internal to (GETPRIM).

PARAMETERS: None.

PURPOSE: Insert a fragment in fragment list.

METHOD: Find fragment's slot in list (sorted by DEST# within PGM#); insert fragment. See ALGORITHMS.

CALLED BY: (EXORDER).

CALLS: None.

INSERT

MODULE: Symbol Table Manager.

SCOPE: Internal to HASHSYM.

PARAMETERS: None.

PURPOSE: Insert a symbol in symbol table.

METHOD: If counter of address's uses is zero, insert in table; increment uses counter.

CALLED BY: HASHSYM, HASHVAL.

CALLS: None.

MINMAX

MODULE: Interpreter Module.

SCOPE: Internal to SEND.

PARAMETERS: None.

PURPOSE: Execute a send-and-combine microinstruction with MIN or MAX combining operator.

METHOD: Loop through each cell of target expression; if value larger than message is encountered and combining operator is MAX, save value in message; if value smaller than message is encountered and combining operator is MIN, save value in message.

CALLED BY: SEND.

CALLS: None.

MOVEIT

MODULE: Storage Management Module.

SCOPE: Internal to STORAGE.

PARAMETERS: None.

PURPOSE: Move each cell of L to its new postion; update program list and RA's syntax tree.

METHOD: Use list of cells moving in same direction to move each cell without erasing previous contents of cell. See ALGORITHMS.

CALLED BY: STORAGE.

CALLS: None.

NEWTOK (NEWCHAR, CLASS)

MODULE: Input Module.

SCOPE: Internal to (SCANTOK).

PARAMETERS: NEWCHAR CHAR(*);
Character scanned.

CLASS FIXED BIN;
Lexical class of NEWCHAR.

PURPOSE: Scan first character of new token and determine its lexical class.

METHOD: Find next non-blank character, reading new input if necessary; check if character is in special lexical class; if not, look up lexical class in tables of legal characters and corresponding lexical classes.

CALLED BY: (SCANTOK).

CALLS: None.

PARSE

MODULE:	Update List of RA's.
SCOPE:	Internal to FINDRAS.
PARAMETERS:	None.
PURPOSE:	Build syntax tree for an RA, with each node containing the index of a non-empty cell of the RA in L, and the indices in the tree of its son and brother.
METHOD:	Stack next non-empty cell if it's a son; pop the stack if it's a father or brother.
CALLED BY:	FINDRAS.
CALLS:	None.

POP (TREEIND, RT, TOP, MATCH, COND, TEST)

MODULE: Interpreter Module.

SCOPE: External.

PARAMETERS:

TREEIND FIXED BIN;
Leftmost syntax tree index of target expression.

RT FIXED BIN;
Rightmost syntax tree index of target expression.

TOP FIXED BIN;
Index of top entries in COND and MATCH stacks.

MATCH(*,*,*) BIT(*) UNALIGNED;
Results of previous comparisons involving messages for each syntax tree node in target expression and each syntax tree node of message (see ALGORITHMS).

COND(*,*) BIT(*) UNALIGNED;
Results of previous comparisons for each syntax tree node in target expression.

TOP FIXED BIN;
Boolean value with which 2 top stack entries are to be combined (0=OR, 1=AND, -1=pop top entry off stack).

PURPOSE: Combine comparison stack entries.

METHOD: Assign results to COND depending on values of MATCH and operands involved in previous comparisons. See ALGORITHMS.

CALLED BY: (EXSTMT).

CALLS: ABEND.

PRIM (P1, P6, TIMER)

MODULE: Interpreter Module.

SCOPE: External.

PARAMETERS: P1 POINTER;
Base pointer for RA data structure.

P6 POINTER;
Base pointer for PRIMITIVE data structure.

TIMER FIXED BIN;
Tells whether before or after storage management for this RA.

PURPOSE: Reduce an RA with a primitive in the operator position.

METHOD: For each microprogram fragment locate corresponding target expression; execute fragment's instructions.

CALLED BY: RAS.

CALLS: (EXFRAG), ABEND.

PRINTL (LOPT, OUTPUT)

MODULE: Output Module.

SCOPE: External.

PARAMETERS: LOPT(*) CHAR(*);
Options specifying what type of snapshot to print.

OUTPUT FILE VARIABLE;
DD name of output file.

PURPOSE: Print a snapshot of the L array.

METHOD: Examine options for output file; print snapshot according to options.

CALLED BY: REDMACH.

CALLS: None.

PRNTSTS (OUPTUT)

MODULE: Output Module.

SCOPE: External.

PARAMETERS: OUTPUT FILE VARIABLE;
DD name of output file.

PURPOSE: Print statistics.

METHOD: RETURN.

CALLED BY: REDMACH.

CALLS: None.

REMARKS: Not implemented.

RAFINI

MODULE: Interpreter Module.

SCOPE: Internal to RAS.

PARAMETERS: None.

PURPOSE: Show RA is reduced.

METHOD: Reinitialize status of each cell in RA to NOTRA; delete RA from list of RA's.

CALLED BY: RAS.

CALLS: None.

RAFINI

MODULE: Storage Management Module.

SCOPE: Internal to STORAGE.

PARAMETERS: None.

PURPOSE: Show RA is reduced.

METHOD: Reinitialize status of each cell in RA to NOTRA; delete RA from list of RA's.

CALLED BY: STORAGE.

CALLS: None.

RAS (DMLEVELS)

MODULE: Interpreter Module.

SCOPE: External.

PARAMETERS: DMLEVELS FIXED BIN;
Time available for data movement in current machine cycle.

PURPOSE: Process each RA in list; compute time available for data movement.

METHOD: GO TO label array for each possible CLASS-STATUS combination. Add operator's I/O time to data movement time if appropriate; reduce or request storage for each RA.

CALLED BY: REDMACH.

CALLS: PRIM, (DMSTAT), (RAFINI).

REDMACH (PARMS)

MODULE: Main Control Module.

SCOPE: External Main.

PARAMETERS: PARMS CHAR(100) VARYING;
Input options string from user's EXEC card.

PURPOSE: Execute the simulator.

METHOD: Call subroutines; keep track of how many cycles have executed and call output routines when options specify that it is time.

CALLED BY: JCL.

CALLS: SETUP, INITRAS, FINDRAS, RAS, STORAGE, PRINTL, PRNTSTS, TALK, ABEND.

SCANTAB

MODULE: Symbol Table Manager.

SCOPE: Internal to HASHSYM.

PARAMETERS: None.

PURPOSE: Determine a symbol's symbol table address.

METHOD: Hash symbol to find address using mid-squares algorithm; if collision occurs, probe table linearly until empty address is located.

CALLED BY: HASHSYM, HASHVAL, SEARCH.

CALLS: None.

SCANTOK (TOKEN, STATE)

MODULE: Input Module.

SCOPE: Internal to SETUP.

PARAMETERS: TOKEN CHAR(*) VARYING;
Token scanned.

STATE FIXED BIN;
TOKEN's recognize state.

PURPOSE: Read next token from input and determine its lexical class.

METHOD: Table driven scanner.

CALLED BY: (GETDEF), (GETPROG).

CALLS: (GETCHAR), (NEWTCK).

SEARCH (TOKEN, ADDRESS)

MODULE: Symbol Table Manager.

SCOPE: External entry point in HASHSYM.

PARAMETERS: TOKEN CHAR(*) VARYING;
Token being sought in symbol table.

ADDRESS FIXED BIN;
Symbol table address of TOKEN.

PURPOSE: Determine whether a symbol is in symbol table; if it is, return its symbol table address; if not, return address indicating 'not found'.

METHOD: Determine symbol table address; if that address is empty, return NOREG, indicating that symbol is not in table.

CALLED BY: (GETOPS), (GETPRIM).

CALLS: (SCANTAB).

SEND (TREEIND, RT, IC, TOP, P1, P6, OPERAND, COND)

MODULE: Interpreter Module.

SCOPE: External.

PARAMETERS:

TREEIND FIXED BIN;
Leftmost syntax tree index of target expression.

RT FIXED BIN;
Rightmost syntax tree index of target expression.

IC FIXED BIN;
Instruction counter.

TOP FIXED BIN;
Index of top entries in COND and MATCH stacks.

P1 POINTER;
Base pointer for RA data structure.

P6 POINTER;
Base pointer for PRIMITIVE data structure.

OPERAND(*,*) FLOAT BIN;
Instruction's operands.

COND(*,*) BIT(*) UNALIGNED;
Results of previous comparisons for each syntax tree node in target expression.

PURPCSE: Execute a send microprogram instruction.

METHOD: If instruction is SI, send the messages and add the number of messages sent to the data movement counter of the RA; if SC, combine the operands according to first send operand (combining operator).

CALLED BY: (EXSTMT).

CALLS: (BOOLOP), (ARITHOP), (MINMAX), ABEND.

SETOPTS (CARD, FREQOPT, LOPT, STATOPT, PGM, OPTFLAG)

MODULE: Interpreter Module.

SCOPE: External.

PARAMETERS:

CARD CHAR(*);
Input card containing user options.

FREQOPT(*,*) CHAR(*);
Output frequency controls.

LOPT(*,*) CHAR(*);
Output format controls.

STATOPT(*) CHAR(*);
Option controlling printing of statistics.

PGM(*) CHAR(*);
Program selector list.

OPTFLAG(*) BIT(*);
Flag telling whether or not to set an option.

PURPOSE: Assign values to option variables.

METHOD: Parse option card for each option keyword and operand values; check if option is to be specified; if so, set value of option variable and assign value '1'B to corresponding option flag.

CALLED BY: (GETOPTS).

CALLS: ABEND.

SETREG (P1, LEFT, RIGHT, REG#, OP, GRO)

MODULE: Interpreter Module.

SCOPE: External.

PARAMETERS:

P1 POINTER;
Base pointer for RA data structure.

LEFT FIXED BIN;
Leftmost syntax tree index of target expression.

RIGHT FIXED BIN;
Rightmost syntax tree index of target expression.

REG# FIXED BIN;
Number of register to be initialized.

OP(*) FIXED BIN;
Operand to which register values are to be assigned.

GRO(*) FLOAT BIN;
Contents of register of temporary results.

PURPOSE: Assign to an operand values of a register of L.

METHCD: Calculate register values indicated by register number for each syntax tree node in target expression.

CALLED BY: PRIM, INSERT.

CALLS: None.

SETUP (FREQOPT, LOPT, STATOPT, DDNAME, PARMS)

MODULE: Input Module.

SCOPE: External.

PARAMETERS: FREQOPT(*,*) CHAR(*);
Options specifying when to print the L array.

LOPT(*,*) CHAR(*);
Options specifying format in which L is to be printed.

STATOPT(*) CHAR(*);
Options specifying when statistics are to be printed.

DDNAME(*) FILE VARIABLE;
DD names of output files.

PARMS CHAR(*) VARYING;
User option string passed to main procedure from EXEC card.

PURPOSE: Initialize user options, L array, symbol table, micro-opcode table, and register table; read defined and primitive operators and translate them to internal representation.

METHOD: Call internal subroutines.

CALLED BY: REDMACH.

CALLS: HASHSYM, SEARCH, ABEND, (INIT), (GETOPTS), (GETPROG), (GETOPS).

STORAGE (DMLEVELS)

MODULE: Storage Management Module.

SCOPE: External.

PARAMETERS: DMLEVELS FIXED BIN;

PURPOSE: Fill as many storage requests as possible.

METHOD: Calculate total storage requests, necessary cancellations, and cells' new positions by moving up and down in a binary tree (T).

CALLED BY: REDMACH.

CALLS: (HALF1), (HALF2), (MOVEIT), (RAFINI).

TALK (FREQOPT, LOPT, STATOPT)

MODULE: Output Module.

SCOPE: External.

PARAMETERS: FREQOPT(*,*) CHAR(*);
Options specifying when to print the L array.

LOPT(*,*) CHAR(*);
Options specifying format in which L is to be printed.

STATOPT(*) CHAR(*);
Cptions specifying when statistics are to be printed.

PURPOSE: Initiate an interactive conversion with user.

METHOD: RETURN.

CALLED BY: REDMACH.

CALLS: None.

REMARKS: Not implemented.

Chapter 5

ALGORITHMS

5.1 GETPRIM

Microprogramming is documented in the REDMACH User's Guide. The microprogram format used by the Interpreter Module differs from what the user codes in two ways: instructions are stored in a fixed binary array (one byte per field), and they are sorted into the order in which they are to be executed.

The translation of the fields of a microinstruction is straightforward. The first field of an instruction is the opcode, which is translated by table lookup in MICRTAB. The opcode determines what type of operand each subsequent field in the instruction contains. The register bytes, the number of which depends on the opcode and is determined by the value in MICRTAB.R, are also translated by table lookup in REGTAB, except if the register is a message or a program symbol. Messages are translated as follows. MESSREG is the low bound of REGTAB minus one. So, the four components of message #1 are numbered MESSREG, MESSREG-1, MESSREG-2, and MESSREG-3. The components of message #2 start at MESSREG-4. In this way, each message component can be uniquely identified. A program symbol is translated to the symbol's address in the symbol table. If a register field is not used (e.g. in send statements), it is translated to NOREG. Immediate data are translated by converting from character string to binary. The conditional operator is translated as follows:

1) AND -> 1

2) OR -> 0

3) STACK -> -2

4) THEN -> 1

5) ELSE -> 0

6) if omitted -> -1

Instruction sequencing is crucial; if the interpreter tried to assign a message to a cell before the message had been sent, incorrect results would occur. There are four requirements of the ordering process.

1) Instructions that are grouped into a fragment by the assembler instructions BEG and END must remain contiguous and in their original order relative to each other.

2) A message with index i must not be referenced before all send instructions with index i have been executed. No send instruction with index i+1

can be executed before all send instructions with index i have been executed. However, one or more conditionals may compare a message with index i and then execute a send instruction with index i+1. That is, the following instruction group is legal:

```
(CER,S,M1(1))
(SI,T,,,,2,THEN)
(CER,N1,M2(1))
(SI,F,,,,2,THEN).
```

3) Insert statements must be executed last.

4) If all other considerations are equal, instructions must be sorted by destination number; that is, they must go from left to right within the RA.

The solution to these requirements is as follows.

1) Index microprogram's destination expressions in destination list from left to right.

2) Assign to each instruction a program number (PGM#) as follows:

   a) no messages referenced, not an insert:
      PGM#=1,

   b) send instruction with index i:
      PGM#=2*i-1,

   c) message referenced, index i (e.g. M2(i)),
      PGM#=2*i,

   d) insert instruction:
      PGM#=100.

3) If the instruction is part of an explicit fragment (part of a BEG-END group) save the largest PGM# of all instructions in the fragment so that the fragment will be grouped together correctly. Do not go to 4 until END is encountered.

4) Insert destination number (DEST#), PGM#, and instruction counter starting and ending values in a list sorted by DEST# within PGM#. There is one entry in the list for each fragment, explicit or implicit. All instructions that are contiguous in

the original microprogram and which either have the same DEST# and PGM# or are grouped together with BEG and END form a fragment.

5) Copy instructions into PRIMITIVE.STMT in fragment order.

The result of this algorithm is a series of fragments that can be executed in order.

5.2 PRIM

This procedure controls the interpretive loop that executes microprograms. See GETPRIM for an explanation of the internal representation of microprograms. This is the basic loop.

1) Determine starting fragment number (depends on whether before or after storage management).

2) Find target expression of fragment.

3) Execute fragment's instructions (CALL EXSEG).

4) EXSEG calls EXSTMT for each instruction in a fragment.

EXSTMT uses a GO TO label array to branch to a section that executes the correct microinstruction or calls a procedure to execute it. There are two interrelated mechanisms involved that are complicated: conditionals and messages.

Two rounds of messages are saved so that one can be referenced while the next is being sent. Other information must be saved also: the index of the message, whether it was sent by an SI or SC instruction, whether each component is a symbol table address or immediate data, and the syntax tree bounds of the target expression that sent the message (this is not needed for SC messages since a single result is computed). If sending the message is the arm of a conditional, it is necessary to remember which cells sent a message and which did not; this is stored in MESSFLAG.

Now we will consider conditionals. If no messages are involved, the comparison part of a conditional can be evaluated using a stack of results, in which each element is an array containing one result for each cell of the target expression.

If messages are involved, comparisons are harder. What are the complicating factors?

1) The cells in which registers other than messages are to be referenced are the cells of the current target expression.

2) The syntax tree bounds of messages referenced were set when the message was sent and thus are the bounds of another target expression in the RA.

3) Not every cell of the message's target expression may have sent a message.

Two loop controls are needed to resolve points 1 and 2. Furthermore, a single stack of results no longer suffices. Suppose these instructions are executed:

```
(CER,S,M1(1))
(CER,N1,M2(1),AND)
(ADR,S,M1(1),THEN).
```

Suppose furthermore that the messages with index 1 were the result of an SI instruction, so that there are n messages, where n is the number of non-empty cells in the target expression from which the messages originated. Then for each cell of the current target expression, we need to know not only whether S equals one of the messages, but also which message it equals. What the instructions really say is: "initialize register 0 to S; whenever S equals M1(1) and RLN equals M2(1), add M1(1) to the total in register 0".

REDMACH uses a pair of bit arrays to resolve this problem. COND is a two-dimensional bit array that is the result stack mentioned earlier. For comparisons not involving messages, this array holds the result of the comparison for each cell of the target expression. MATCH is a three-dimensional bit array that shows for each cell of a target expression where the comparison was true, if it was true at all. TOP is the index of the top elements of both COND and

MATCH. For comparisons not involving messages, the result for a cell with syntax tree index I is stored in the entire corresponding cross section of the array, i.e. MATCH(TOP,I,*). For comparisons involving messages, the result of comparing an operand of a cell with syntax tree index I to a message sent by a cell with syntax tree index J is stored in MATCH(TOP,I,J). So, if in the previous example the values of S and RLN of syntax tree node I equal the message values sent by syntax tree node J, then MATCH(TOP,I,J) will be true.

REDMACH must also combine the results of multiple comparisons, e.g.

```
(CEI,N1,1)
(CER,S,M1(1),AND).
```

The following algorithm is used to evaluate comparisons.

1) Set TEST to instruction's comparison operator. See GETPRIM for values.

2) For each symbol in target expression,

   a) assign symbol's syntax tree index to I,

   b) initialize temporary bit array, TEMP, to false,

      i) if comparison does not involve a message, evaluate comparison and store result in TEMP(*),

ii) if comparison involves messages, evaluate comparison for each message and store result in TEMP(J), where J is the index in the syntax tree of the cell that sent the message,

c) if TEST=-1 or TEST=-2, assign results in TEMP to MATCH(TOP,I,*): if any TEMP(J) is true, assign true to COND(TOP,I),

d) if TEST=1, execute bitwise AND of TEMP(*) and MATCH(TOP,I,*), storing results in MATCH(TOP,I,*); if any element in MATCH(TOP,I,*) is set to true, assign true to COND(TOP,I),

e) if TEST=0, execute bitwise OR of TEMP(*) and MATCH(TOP,I,*), storing results in MATCH(TOP,I,*); if any element in MATCH(TOP,I,*) is set to true, assign true to COND(TOP,I).

Branches of conditionals are chosen for each cell of a target expression using the results stored in COND and MATCH. The algorithm for branches follows.

1) Set TEST to instruction's branch operator value. See GETPRIM for values.

2) For each cell in the current target expression,

   a) if TEST=-1, evaluate instruction,

   b) if TEST=1, execute instruction only if comparison was true,

   c) if TEST=0, execute instruction only if comparison was false.

If an instruction with a branch operator references a message, it must get the one that matches correctly. That is, if TEST=1 and MATCH(TOP,I,J)='1'B, then use MESS(J), and if TEST=0 and ¬MATCH(TOP,I,J)='0'B then use MESS(J).

5.3 STORAGE

Preparation for storage management is performed as Mago$' describes in [3]. The implementation used in REDMACH is as follows.

1) Cells of T are indexed 1:SIZE-1, where SIZE is the number of cells of L.

2) Cells of L are indexed SIZE:2*SIZE-1.

3) To move up in T from a son to a father node, divide the son's index by 2.

4) To move down in T from a father node to his son nodes, if the father node's index is i, his left son's is i*2 and his right son's is 2*i+1.

Storage management is performed in one pass through L. Instead of calculating BL and BR values, the program calculates each cell's new index in L (NEWPOS). A linked list (STORMAN) keeps track of each contiguous group of cells that will move in the same direction. Using this list, groups of cells can move without overwriting the contents of a cell that has not yet moved. If a group is moving to the left, the leftmost cell in the group moves first, if to the right, the rightmost cell moves first.