Mimer/DB

Mimer Database Manager User Guide

Updated for Mimer/DB Version 3.2.8

9740-00

Diab Data AB

Box 2029 S-183 02 TÄBY
SWEDEN

MIMER DATA BASE MANAGER
(MIMER/DB)

Reference Manual

Version 3.2
January 1984

CHAPTER 1

MIMER/DB GENERAL DESCRIPTION

1.1 Introduction

MIMER/DB is a multi-user relational data base management system with an active data dictionary to control data access, usage, and security. MIMER/DB was developed at Uppsala University Data Centre (UDAC) in Sweden and it is now the nucleus of a family of products which as application programs, access the data base. These products include:

A query language

-MIMER/QL, an English type query language suitable for end users which has simple commands for accessing the data quickly and also for interactively building and extending the data base records.

A prototyping language and program generator

-MIMER/PG, which eliminates the need for coding in primitive languages such as COBOL. MIMER/PG includes a report generator for rapidly preparing reports to any specification.

A screen handler system

-MIMER/SH, for quick and easy formatting of data entry and query screen menus.

An information retrieval system

-MIMER/IR, for large data and text search applications.

Utility programs

-used to execute frequent operations on the data base, such as back-up and recovery operations, and to carry out statistical analysis of disk usage, analysis of buffer pool usage etc.

1.2 Physical Storage Structure

1.2.1 Databanks

. MIMER/DB works with storage units called databanks. A MIMER data base may contain an arbitrary number of databanks. A databank corresponds to an ordinary direct access file as far as the operating system is concerned. It consists of commonly sized pages addressable by a relative page number. The size of the pages used in MIMER/DB is installation dependent. A large block size is desirable for sequential access but a small size is preferable for multi-index searching. Note that if you do change the block size for an installation, it is necessary to reload **all** the applications. This is a lot of work!

A databank contains an arbitrary number of tables. All the contents in a table are stored in one databank, whereas one databank may contain several tables. The structure of a databank is shown in Fig. 1.

In MIMER/DB data transfer between primary and secondary memory is done on a page by page basis. Every databank has a bitmap which is used to indicate which pages are used, and which pages are free. Thus the bitmap is a directory of space utilization. If MIMER/DB needs a new page in a databank, this page is marked "used" in the bitmap. If a page is detached, e.g. when deleting data, it is marked "free" in the bitmap. This means that the databank grows and shrinks, depending upon user needs, without the user having to state a maximum size for each table to be stored in the databank.

When creating a databank the first page will become the bit-map and the second will be the "root" page. The root page is effectively a master directory of all the tables in the databank. If the databank is very large, additional bitmap pages maybe used and these will be connected to the first one. In a similar way, if the root page does not have enough entries, addtional root pages are created, connected to the the first. This implies that there is no limit to the number of tables within a databank. All other pages in a databank are either free or used for indexes or data.

1.2.2 Tables

As mentioned earlier, a databank contains a number of tables. According to the terminology of relational data bases, every table is a "n-ary relation", i.e. it contains a number of "tuples" (rows or records), each one consisting of n "attributes" (columns of fields).

The disk storage structure of a table is shown in Fig. 2 and is represented by a so called B*-tree.

In MIMER/DB, the B*-tree consists of an index section and a data section. The rows of a table are stored in the data section, i.e. in the leaf nodes of the tree. The index section of the tree, i.e. where the non-leaf nodes of the tree are stored, is a roadmap to enable rapid location of the required node on the next level. Inside all nodes, data are stored sorted on the key values. This makes it possible to use a fast binary search algorithm to find a certain row, or to find the place where a row should be inserted. When inserting new rows, the tree grows and when deleting rows, the tree gets smaller quite automatically. The MIMER/DB algorithm for maintaining the B*-tree structure will give an average of 83% (5/6) used space in each node. This implies that continuous reorganization is done automatically by MIMER/DB, and periodical reorganizations are obsolete.

A table is entered via the root page of the databank containing the table. The root page is the directory of the tables in the databank and contains page number references to the roots of the B*-tree structures.

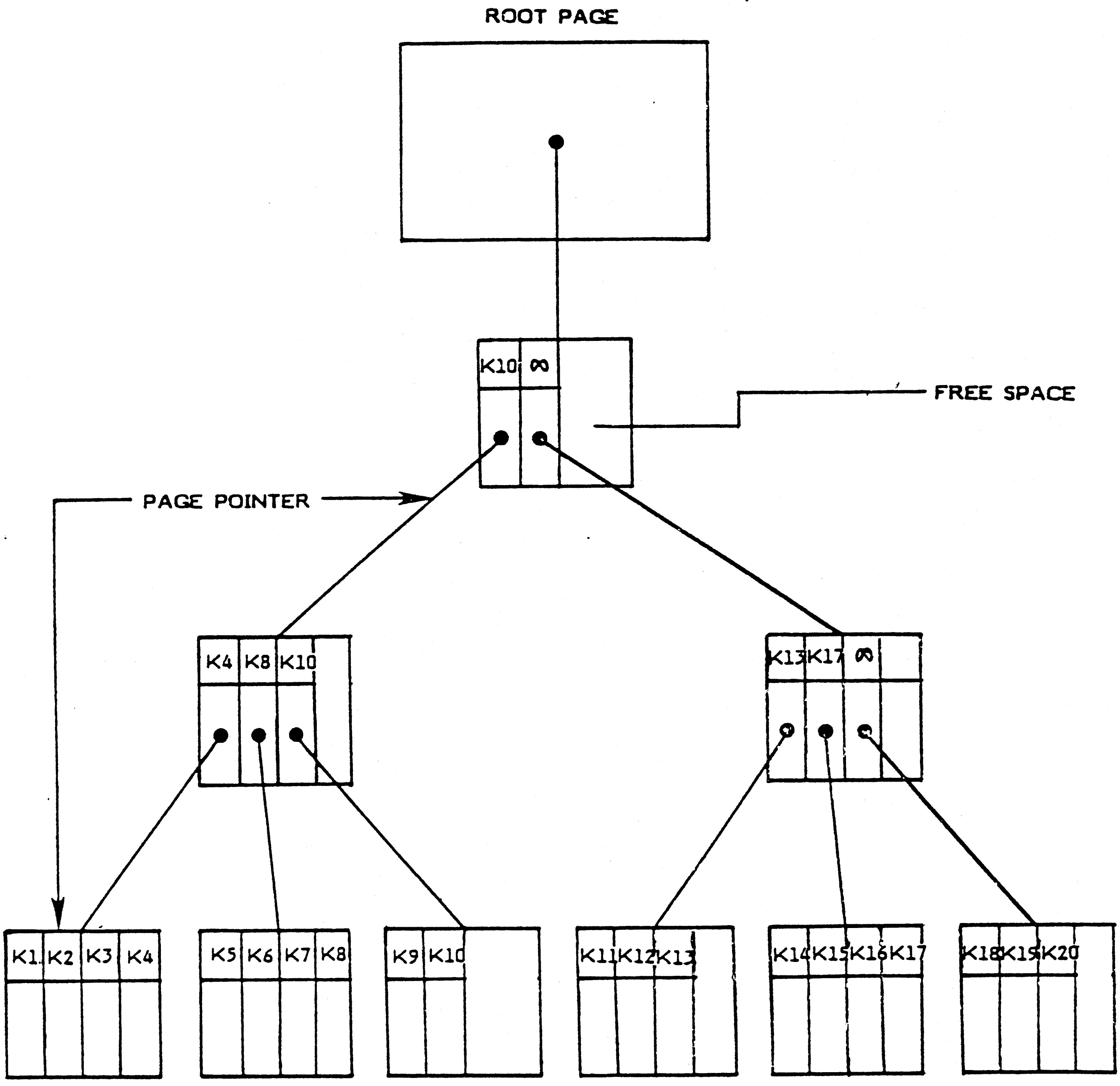

Fig. 2 Table Structure (B*-tree)

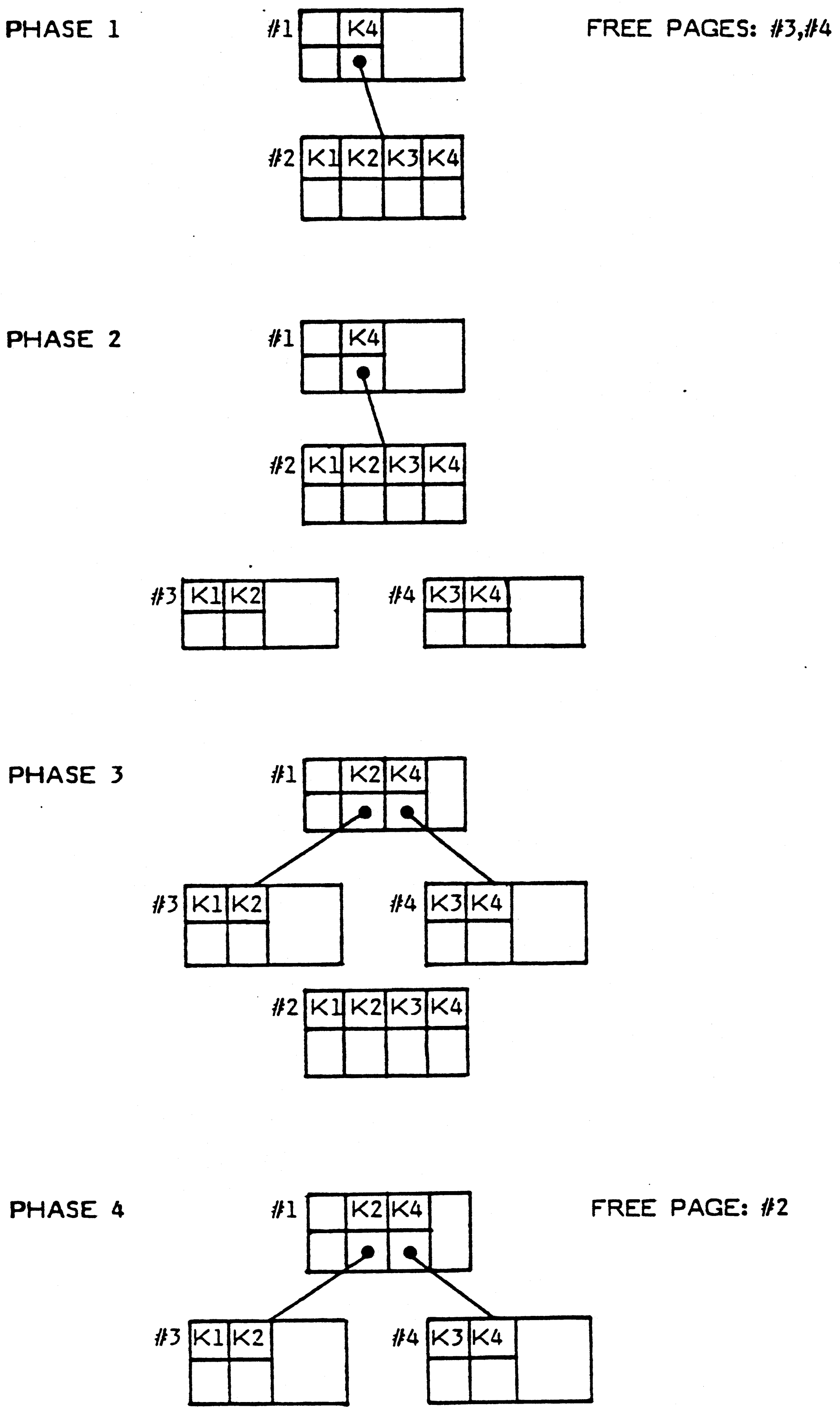

Fig. 3 B*-tree Transformation (Split)

Valid numeric constants:

```
12
 1.E-5
-33.523
 6E22
```

Invalid:

2.5E	E must be followed by an integer value
E33	E must be preceded by a decimal value
125A3	Letter among the digits
2,4	Use decimal, point comma not valid

Character format

Characters are stored one to a byte in the same way as the internal machine representation (ASCII, EBCDIC etc). This means that the logical storage order is in accordance with the representation, and implies that for ASCII the digits are sorted before the letters, whereas in EBCDIC it is the other way round. A string of null-bytes is regarded as undefined.

Integer format

Integers are stored in an internal binary format not identical to the machine representation. The internal binary format is necessary to improve performance in bit by bit searching within compare operations. Integers can be stored in an arbitrary length up to the machine dependent length IBYTE. The shorter lengths can be used to save space, but the valid range is reduced.

One value is reserved. INULL, is regarded as an undefined value. It appears for instance on input (character to integer conversion) if a blank string or if an illegal numeric constant occurs. If overflow occurs at input, a maximum value is set (sign*IMAX if stored in length IBYTE). Observe that if shorter storage length is used, a smaller maximum value is also used. On output (conversion to character) an undefined value is translated to a string of null bytes.

Floating point format

Floating point numbers are stored in an internal binary format not identical to the machine representation. They can be stored in an arbitrary length up to the machine dependent length FBYTE. (On some machines DOUBLE PRECISION is not implemented). Shorter lengths can also be used, to save space, but the precision is then decreased. To store any meaningful information at least two or three bytes are required.

Conversion between floating point numbers and characters can possibly give a slight decrease in significance.

1.3 Access module structure

The MIMER/DB software is structured in a number of modules, which in turn are grouped into well defined layers.

The user interface is the highest layer (2), and it manages the data transfer from the internal control area to the application program (or vice versa). In connection with this transfer, a format conversion of the involved data may be performed.

The row manager is the next layer (1), which among other things transfers the required rows between a page in the buffer-pool and the user interface control area. Facilities for transaction management and sort/merge are also included.

The page manager is the lowest layer (0), which controls the buffer pool in a similar way to virtual memory management. When a certain page is required, which is not in the buffer-pool (page fault), the least recently used (LRU) page makes room for the new page. If the LRU-page is marked updated, it is written back to disk, before the new page is read into the buffer. The actual direct access I/O operations are performed via the I/O interface, which is the connection between MIMER/DB and the operating system.

The single-user system structure is shown in Fig. 4.

In the multi-user system, which is shown in Fig. 5, the user interface accesses the row manager through a traffic controller connected to the MIMER/DB nucleus. There may be an arbitrary number of nucleus processes working against the shared memory buffer-pool. This reveals that MIMER/DB supports multi-threaded access.

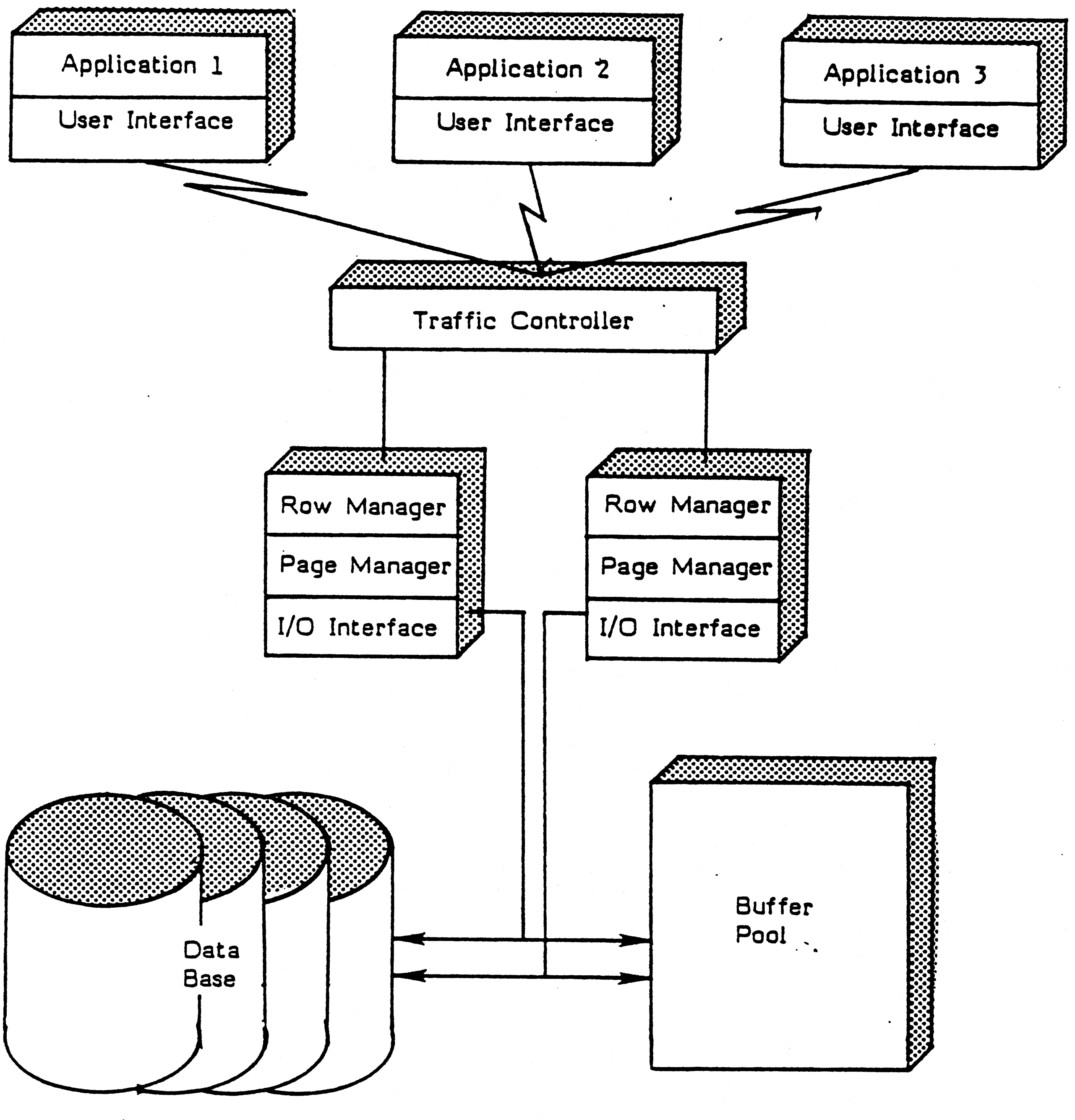

Fig. 5 MIMER/DB Multi-User System

1.4.2 Specification of Databank SYSDB

Table	Column			Description		
*DBDEF	DATABANK	* C	8	Logical name of databank		
	DBNO	C	4	Databank identity		
	ACCESS	C	1	Type of databank	General access	
				X - User databank	exclusive	(all operations)
				S - User databank	shared	(read and write operations except load, drop)
				R - User databank	read	(read only)
				P - User databank	private	(refer to table *ACCESS)
				B - Databank backup	private	(only COPYD2 and RESTD2 can access B databanks)
	PHYSFILE	C	xx	Physical name of databank		
*USERDEF	USER	* C	8	User identification or username		
	USERNO	C	4	User identity (used internally)		
	AUTH	C	1	User authority		
				X - data base administrator (DBA)		
				S - ordinary user		
	(PASSWORD)	C	8	Coded password	Non-accessible	
*ACCESS	USER	* C	8	User name		
	DATABANK	* C	8	Databank name		
	ACCESS	C	1	Type of access allowed		
				X - exclusive	(all operations)	
				S - shared	(read and write operations except load, drop)	
				R - read	(read only)	
				P - private	(no access allowed)	

1.5 Data base security

1.5.1 Access to MIMER/DB

Any program that is to communicate with MIMER/DB must first provide a user identification (userid) and a password by entering BEGIN2.

Authorized userids and passwords for accessing MIMER/DB are stored in the *USERDEF system table. Initial entries in this system table are made when the data base is generated. A user who has Data Base Administration authority can define a new user by entering DEFUS2.

A user may have one of following authorities:

'S' - Standard	(Ordinary user)
'X' - Exclusive	(Data Base Administrator)

1.6 Transaction management

In order to preserve the integrity during concurrent access to a shared database, operations must be grouped in units called transactions.

When using the MIMER/DB transaction facilities, a databank named TRANSDB must be defined. TRANSDB acts as a secondary memory storage for MIMER/DB. If the logging facility is required, the databank LOGDB has to be defined. When LOGDB is existence, MIMER/DB will automatically log every commited update-transaction.

The beginning of a transaction is indicated by entering BEGTR2, and the end by ENDTR2 requesting either commit or abort.

The transaction manager in MIMER/DB does not use any locking protocol in order to preserve integrity. Therefore, dead-lock detection or prevention is not applicable. Instead, an "optimistic" concurrency protocol is used, which relies for efficiency on the expectation that conflicts between transactions will not occur.

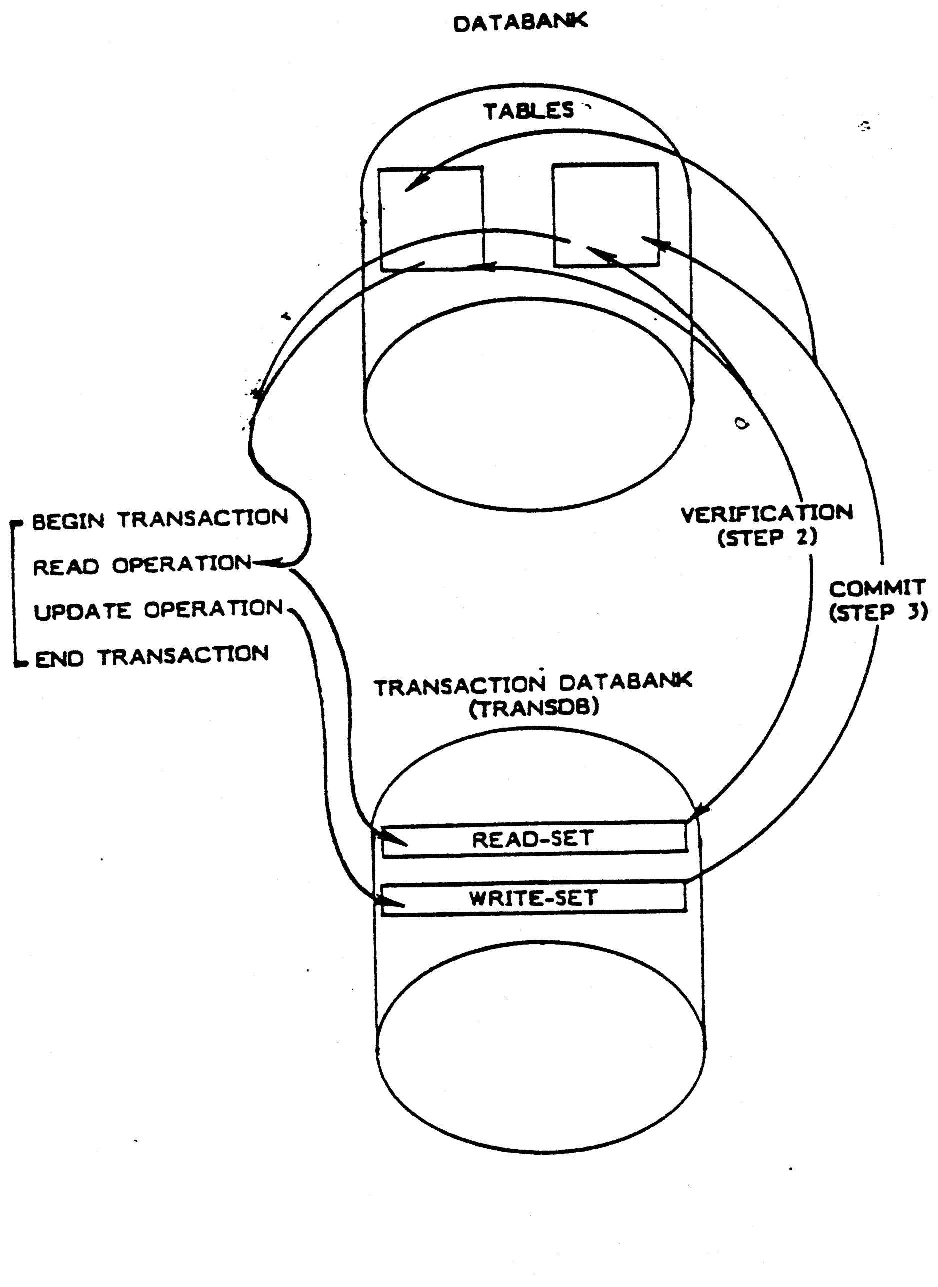

Fig. 7 Optimistic Concurrency Control

1.8 How to use MIMER/DB Routines

In this section, we have used several examples, trying to illustrate how the different routines in MIMER/DB are connected to each other. The examples are written in an ADA-like pseudo-language. Parameters in capital letters are used, when they denote a character string containing the same information. A detailed description of each MIMER/DB routine will be found in Chapter 2.

1.8.1 How to start/end processing

First of all, **BEGIN2** must be entered with userid and password, in order to determine if the user is allowed to use the data base (LOGON).

Each databank that the user wishes to access, must then be opened by entering **OPEND2**, in order to determine the users access privilege.

Later, when doing operations on a databank, that should be treated as one transaction, they must be enclosed by entering **BEGTR2** and **ENDTR2**. Applications using transactions are shown in Examples 3 and 4.

When all operations on a databank have been finished, the user may enter **CLOSD2**, indicating that the databank is not in use.

Finally, before ending the application program, **END2** must be entered, in order to release MIMER/DB control (LOGOFF).

Example 1

For each manufacturer, display all drugs and their classification.

```
+-----------------------------------------------------+
|                                                     |
|    BEGIN2(rcode, userid, password);                 |
|                                                     |
|    OPEND2(did,DRUGDB,R);                            |
|                                                     |
|    OPENT2(cursor1,did, MANUFACT,R);                 |
|    PROJE2(cursor1,F,MANID,RO,base,pmanid,C, 3);     |
|    PROJE2(cursor1,A,MNAME,RO,base,pmname,C,50);     |
|                                                     |
|    OPENT2(cursor2,did,DRUGS,R);                     |
|    PROJE2(cursor2,F,DNAME,RO,base,pdname,C,15);     |
|    PROJE2(cursor2,A,CLASS,RO,base,pclass,C,40);     |
|    SELEC2(cursor2,F,MANID,EQ,base,pmanid,C, 3);     |
|                                                     |
|    SET2  (cursor1,base);                            |
|    loop                                             |
|        GET2  (cursor1,base);                        |
|        exit when cursor1(1)> 0;                     |
|        ---------------------- display 'pmname'      |
|        SET2  (cursor2,base);                        |
|        loop                                         |
|            GET2  (cursor2,base);                    |
|            exit when cursor2(1)> 0;                 |
|            --------------- display 'pdname,pclass'  |
|        end loop;                                    |
|    end loop;                                        |
|                                                     |
|    END2;                                            |
|                                                     |
+-----------------------------------------------------+
```

When using MIMER/DB, the solution to this problem is to use **PUSH2** to save the current cursor position, when going down to next level, and **POP2** to restore the cursor, when going back up, as shown in Example 2.

Example 2

Parts explosion.

```
BEGIN2(rcode,userid,password);

OPEND2(did,databank,R);

OPENT2(cursor,did,USAGE,R);
PROJE2(cursor,F,MINOR,RO,base,pminor,C,6);
PROJE2(cursor,A,QTY  ,RO,base,pqty  ,C,3);
SELEC2(cursor,F,MAJOR,EQ,base,pminor,C,6);
pminor:="Car   ";

SET2  (cursor,base);
loop
    GET2  (cursor,base);
    if cursor(1) = 0
    then
        -------------- display 'pminor,pqty'
        PUSH2 (cursor);
        SET2  (cursor,base);
    else
        POP2  (cursor);
        exit when cursor(1)>0;
    end if;
end loop;
END2;
```

Example 3

Reduce the 'Tartrazin'-contents of all drugs by 50%, and replace it by the new and harmless colouring matter 'Falu-red'.

```
+--------------------------------------------------------+
|                                                        |
|    BEGIN2(rcode,userid,password);                      |
|                                                        |
|    OPEND2(did,DRUGDB,S);                               |
|                                                        |
|    OPENT2(cursor1,did,CONTENT,S);                      |
|    PROJE2(cursor1,F,DRUGID   ,RO,base,pdrugid   ,C,3); |
|    PROJE2(cursor1,A,PWEIGHT  ,RW,base,ppweight  ,I,2); |
|    SELEC2(cursor1,F,COMPOSIT,EQ,base,scomposit,C,9);   |
|    scomposit:="Tartrazin";                             |
|                                                        |
|    OPENT2(cursor2,did,CONTENT,S);                      |
|    PROJE2(cursor2,F,DRUGID   ,WO,base,pdrugid   ,C,3); |
|    PROJE2(cursor2,A,COMPOSIT,WO,base,pcomposit,C,8);   |
|    PROJE2(cursor2,A,TYPE     ,WO,base,ptype     ,C,1); |
|    PROJE2(cursor2,A,PWEIGHT  ,WO,base,ppweight  ,I,2); |
|    pcomposit:="Falu-red";                              |
|    ptype    :="C";                                     |
|    loop                                                |
|        BEGTR2(rcode);                                  |
|        SET2  (cursor1,base);                           |
|        loop                                            |
|            GET2  (cursor1,base);                       |
|            exit when cursor1(1)> 0;                    |
|            ppweight:=ppweight/2;                       |
|            DELET2(cursor1,base);                       |
|            INSER2(cursor2,base);                       |
|        end loop;                                       |
|                                                        |
|        ENDTR2(rcode);                                  |
|        exit when rcode = 0;                            |
|    end loop;                                           |
|                                                        |
|    END2;                                               |
|                                                        |
+--------------------------------------------------------+
```

1.8.5 How to load/drop data in a table

Before doing a load or drop operation on a table, a cursor must be defined by entering **OPENT2**. The access option given must be 'X' (exclusive).

All rows in a table may then be dropped by entering **DROP2**.

The load operation works on a row-by-row basis, and must be fed with data from the application program, and therefore **PROJE2** must be entered in order to connect columns with program variables.

By repeatedly entering **LOAD2**, all rows will be dispatched to an internal sort/merge routine, which implies that an indication must be given when all rows have been loaded.

The end-of-load signal is given by entering **SET2**, which means that the sort/merge will terminate and the sorted rows will be loaded into the table.

An application dropping all rows, and then re-loading the table is shown in Example 5.

1.8.6 How to define/remove a table

Tables are defined on a column-by-column basis by subsequently entering **DEFCO2**, as shown in Example 6.

Columns may be removed from a table by entering **REMCO2**. When the last column is removed, the table ceases to exist.

1.8.7 How to define/remove an index

A secondary index will be created for a column by entering **DEFIX2**, as shown in Example 6.

When no longer needed, a secondary index can be dropped by entering REMIX2.

Example 6

Define the table DRUGS, and create a secondary index for the column DNAME.

```
BEGIN2(rcode,userid,password);

OPEND2(did,DRUGDB,X);

DEFCO2(did,DRUGS,DRUGID   ,"*",C, 3);
DEFCO2(did,DRUGS,DNAME    ," ",C,15);
DEFCO2(did,DRUGS,FORM     ," ",C, 8);
DEFCO2(did,DRUGS,STRENGTH ," ",C,10);
DEFCO2(did,DRUGS,MANID    ," ",C, 3);
DEFCO2(did,DRUGS,CLASS    ," ",C,40);

  DEFIX2(did,DRUGS,DNAME);

END2;
```

1.8.11 How to copy/restore a databank

A backup copy of a databank will be created by entering COPYD2, as shown in Example 8.

Example 8

Create backup copies of DRUGDB and ADVDB. The backup databanks are named BDRUGDB and BADVDB respectively.

```
+--------------------------------------------------+
|                                                  |
|   BEGIN2(rcode,userid,password);                 |
|                                                  |
|   COPYD2(rcode,BDRUGDB,DRUGDB);                  |
|   COPYD2(rcode,BADVDB,ADVDB);                    |
|                                                  |
|   END2;                                          |
|                                                  |
+--------------------------------------------------+
```

When a disk-crash has occurred, a databank can be restored by entering **RESTD2**, as shown in Example 9.

Example 9

Restore the databanks DRUGDB and ADVDB from the backup databanks BDRUGDB and BADVDB.

```
+--------------------------------------------------+
|                                                  |
|   BEGIN2(rcode,userid,password);                 |
|                                                  |
|   RESTD2(rcode,DRUGDB,BDRUGDB);                  |
|   RESTD2(rcode,ADVDB,BADVDB);                    |
|                                                  |
|   END2;                                          |
|                                                  |
+--------------------------------------------------+
```

1.9 Summary of MIMER/DB Routines

Database Control

BEGIN2	- Begin	MIMER/Db Session
OPEND2	- Open	Databank
BEGTR2	- Begin	Transaction
ENDTR2	- End	Transation
CLOSD2	- Close	Databank
END2	- End	MIMER/DB Session

Data Control

OPENT2	- Open	Table Cursor
PROJE2	- Project Column	
SELEC2	- Select Rows	
SET2	- Set	Table Cursor
PUSH2	- Push	Table Cursor
POP2	- Pop	Table Cursor
DEQUE2	- Dequeue	Table Cursor
CLOST2	- Close	Table Cursor

Data Retrieval/Manipulation

	GET2	- Get (Next) Row
	INSER2	- Insert New Row
	UPDAT2	- Update Current Row
	DELET2	- Delete Current Row
*	LOAD2	- Load New Rows
*	DROP2	- Drop All Rows

Data Definition

*	DEFCO2	- Define column
*	REMCO2	- Remove Column(s)
*	DEFIX2	- Define Index
*	REMIX2	- Remove Index(es)

Data Base Definition

*	DEFAC2	- Define Access
*	REMAC2	- Remove Access
**	DEFDB2	- Define Databank
**	REMDB2	- Remove Databank
**	DEFUS2	- Define User
**	REMUS2	- Remove User

Data Base Service

**	COPYD2	- Copy Databank
**	RESTD2	- Restore Databank
**	GENDB2	- Generate System Databank (SYSDB)

---

* Only for user with 'X'-privilege on the databank
** Only for user with 'X'-authority (DBA)

CHAPTER 2

MIMER/DB ROUTINE DESCRIPTIONS

The MIMER/DB routines are grouped as follows:

```
+-----------------------------------------------------+
|                                                     |
|     Data Base Control                               |
|                                                     |
|     Data Control                                    |
|                                                     |
|     Data Retrieval/Manipulation                     |
|                                                     |
|     Data Definition                                 |
|                                                     |
|     Data Base Definition                            |
|                                                     |
|     Data Base Service                               |
|                                                     |
+-----------------------------------------------------+
```

2.1.1 BEGIN2 - Begin MIMER/DB Session

Purpose:

To initiate MIMER/DB for processing.

Format:

```
+---------------------------------------------------+
|                                                   |
|  BEGIN2 (RCODE, UNAME, PASSW)                     |
|                                                   |
+---------------------------------------------------+
```

Arguments:

RCODE	Iw out	-	return code: < 0 - error = 0 - ok > 0 - backout
UNAME	C8 in	-	user name
PASSW	C8 inout	-	password

Description:

The user, specified by the argument UNAME, is searched for in the system table *USERDEF, and verified by the password, specified by the argument PASSW. The backout return code will be set when the user name is not found, or when the password is incorrect. If the verification is successful, internal control-areas will be initiated, and MIMER/DB is ready for processing.

Note:

The contents of the password argument is always destroyed.

2.1.3 BEGTR2 - Begin Transaction

Purpose:

To indicate the beginning of a transaction.

Format:

```
+-----------------------------------------------------+
|                                                     |
| BEGTR2 (RCODE)                                      |
|                                                     |
+-----------------------------------------------------+
```

Arguments:

```
RCODE        Iw out     -  return code:

                                  < 0 - error
                                  = 0 - ok
```

Description:

The beginning of a transaction is indicated, i.e. all update-operations (INSER2,UPDAT2,DELET2), performed before end-of-transaction (ENDTR2), should be put on an intention-list, and executed at end-of-transaction during the commit-phase.

Note:

The databank TRANSDB must be in existence when the transaction management facilities are used.

Tables opened with access option 'X' will not be handled by transaction management.

2.1.5 CLOSD2 - Close Databank

Purpose:

To close a databank.

Format:

```
+----------------------------------------------------+
|                                                    |
|  CLOSD2 (DID)                                      |
|                                                    |
+----------------------------------------------------+
```

Arguments:

DID	4Iw inout	-	databank identifier DID(1) return code: < 0 - error = 0 - ok

Description:

The databank associated with the identifier, specified by the argument DID, is closed and the connected control space is released for re-use. Subsequently all table cursors connected to the databank are automatically closed.

2.2 Data Control

The following routines are available:

OPENT2	-	Open Table Cursor
PROJE2	-	Project Column
SELEC2	-	Select Rows
SET2	-	Set Table Cursor
PUSH2	-	Push Table Cursor
POP2	-	Pop Table Cursor
DEQUE2	-	Deque Table Cursor
CLOST2	-	Close Table Cursor

Note: Base Address

The declarative routines PROJE2 and SELEC2 use an argument called "base address". The variables used as I/O-areas and for restriction values in the program will have their addresses converted and stored relative to this base address.

When the executive routines SET2, GET2, INSER2, UPDAT2 and LOAD2, are performed, the argument base address are used in order to calculate the absolute addresses of the I/O-areas and the restriction values.

The reason for storing relative addresses instead of absolute addresses is that some host languages may use an address space for those variables which is not fixed from time to time. This means, when calling e.g. PROJE2 the base has a certain address, and when calling e.g. GET2, the base may have another address.

Note that the I/O-areas and restriction values must be located in the space covered by the base address, because a change of the base address implies the same change for all other addresses as well.

2.2.2 PROJE2 - Project column

Purpose:

To project a column onto a program I/O-area, i.e. to specify a binding condition between a column and a variable in the program.

Format:

```
+--------------------------------------------------------------+
|                                                              |
|PROJE2 (TID,LOP,CNAME,TOP,BASE,IOAREA,IOTYPE,IOLEN)           |
|                                                              |
+--------------------------------------------------------------+
```

Arguments:

```
TID         4Iw inout   -  table cursor
                           TID(1) return code:
                                < 0 - error
                                = 0 - ok

LOP         C1 in       -  logical operator:
                                'F' - first
                                'A' - and
                                'E' - erase

CNAME       C8 in       -  column name

TOP         C2 in       -  transfer operator:
                                'RO' - read only
                                'WO' - write only
                                'RW' - read/write
                                'WR' - write/read

BASE        ref         -  base address

IOAREA      ref         -  I/O-area location

IOTYPE      C1 in       -  I/O-area type:
                                'C'   - character
                                'I'   - integer
                                'F'   - floating point

IOLEN       Iw in       -  I/O-area length (in bytes)
```

Description:

A projection is connected to the table cursor, specified by the argument TID, by using a logical operator, specified by the argument LOP. The projection consists of three parts, first column name, specified by the argument CNAME, second a transfer operator, specified by the argument TOP, and third the base address, location, type, and length of an I/O-area, specified by the arguments BASE, IOAREA, IOTYPE and IOLEN respectively.

Description:

A select-condition is connected to the table cursor, specified by the argument TID, by using a logical operator, specified by the argument LOP. The condition consists of three parts, first column name, specified by the argument CNAME, second a relational operator, specified by the argument ROP, and third the base address, location, type, and length of the restriction value, specified by the arguments BASE, RVALUE, RVTYPE and RVLEN respectively.

Notes:

The default option for the table cursor, when no selections are specified, is no restriction, i.e. all rows are selected.

Subsequent conditions are connected to each other by means of the logical operator, thus making a conjunctive normal form expression. E.g. the conditions "First A, Or B, And C, Or D" will be interpreted as: (A or B) and (C or D), i.e. 'or' has precedence to 'and'.

When a SET2 operation has been executed on the table cursor, the logical operator OR cannot be used in the first additional entry to SELEC2.

2.2.5 PUSH2 - Push Table Cursor

Purpose:

To save the table cursor status.

Format:

```
+--------------------------------------------------------+
|                                                        |
|  PUSH2 (TID)                                           |
|                                                        |
+--------------------------------------------------------+
```

Arguments:

TID	4Iw inout	-	table cursor TID(1) return code: < 0 - error = 0 - ok

Description:

The current status of the table cursor, specified by the argument TID, is pushed onto a stack and saved, thus making it possible to use the table cursor on a new set, defined by the same selection conditions, but with new restriction values.

2.2.7 DEQUE2 - Dequeue Table Cursor

Purpose:

To restore the first saved table cursor status.

Format:

```
+------------------------------------------------------+
|                                                      |
|  DEQUE2 (TID)                                        |
|                                                      |
+------------------------------------------------------+
```

Arguments:

```
TID          4Iw inout   -   table cursor

                             TID(1) return code:

                                  < 0 - error
                                  = 0 - ok
                                  > 0 - stack empty
```

Description:

The table cursor, specified by the argument TID, is restored to the previous status, saved by the first PUSH2.

2.3 Data Retrieval/Manipulation

The following routines are available:

GET2	- Get (Next) Row
INSER2	- Insert New Row
UPDATE2	- Update Current Row
DELET2	- Delete Current Row
LOAD2	- Load New Rows
DROP2	- Drop All Rows

Note: Base Address

The declarative routines PROJE2 and SELEC2 use an argument called "base address". The variables used as I/O-areas and for restriction values in the program will have their addresses converted and stored relative to this base address.

When the executive routines SET2, GET2, INSER2, UPDAT2 and LOAD2, are performed, the argument base address are used in order to calculate the absolute addresses of the I/O-areas and the restriction values.

The reason for storing relative addresses instead of absolute addresses is that some host languages may use an address space for those variables which is not fixed from time to time. This means, when calling e.g. PROJE2 the base has a certain address, and when calling e.g. GET2, the base may have another address.

Note that the I/O-areas and restriction values must be located in the space covered by the base address, because a change of the base address implies the same change for all other addresses as well.

2.3.2 INSER2 - Insert New Row

Purpose:

To insert a row into a table.

Format:

```
+------------------------------------------------------+
|                                                      |
|  INSER2 (TID,BASE)                                   |
|                                                      |
+------------------------------------------------------+
```

Arguments:

```
TID          4Iw inout   -  table cursor

                            TID(1) return code:

                                < 0 - error
                                = 0 - ok
                                > 0 - row already exists

BASE         ref         -  base address
```

Description:

A row is created with all columns having null-values initially. Then the row is updated, by making a data transfer from the write-projected I/O-areas, implicitly referenced by the argument BASE. If the row does not exist, it is inserted into the table.

Notes:

Within a transaction, the insert-operation is not actually performed; instead a request is put on an intention-list and executed at end-of-transaction during the commit-phase.

2.3.4 DELET2 - Delete Current Row

Purpose:

To delete current row.

Format:

```
+-------------------------------------------------------+
|                                                       |
|  DELET2 (TID)                                         |
|                                                       |
+-------------------------------------------------------+
```

Arguments:

```
TID          4Iw inout   -  table cursor

                            TID(1) return code:

                                < 0 - error
                                = 0 - ok
                                > 0 - row does not exist
```

Description:

The current row, referenced by the table cursor, specified by the argument TID, is deleted.

Note:

Within a transaction, the delete-operation is not actually performed; instead a request is put on an intension-list and executed at end-of-transaction during the commit-phase.

DELET2 may only be entered after a GET2 or an INSER2 operation.

2.3.6 DROP2 - Drop All Rows

Purpose:

To drop all rows in a table.

Format:

```
+----------------------------------------------------+
|                                                    |
|  DROP2 (TID)                                       |
|                                                    |
+----------------------------------------------------+
```

Arguments:

TID	4Iw inout	-	table cursor
			TID(1) return code:
			< 0 - error = 0 - ok

Description:

All rows in the table, referenced by the cursor, specified by the argument TID, are dropped.

Note:

The table cursor must be open with access mode 'X' (exclusive).

2.4.1 DEFCO2 - Define Column

Purpose:

To define a new column in a table.

Format:

```
+------------------------------------------------------+
|                                                      |
| DEFCO2 (DID,TNAME,CNAME,CFLAG,CTYPE,CLEN)            |
|                                                      |
+------------------------------------------------------+
```

Arguments:

DID	4Iw inout	- databank identifier DID(1) return code: < 0 - error = 0 - ok
TNAME	C8 in	- table name
CNAME	C8 in	- column name
CFLAG	C1 in	- column flag: '*' - primary key ' ' - other
CTYPE	C1 in	- column type: 'C' - character 'I' - integer 'F' - floating point
CLEN	Iw in	- column length (in bytes)

Description:

A new column is created with name, key, type and length, as specified by the arguments CNAME, CFLAG, CTYPE and CLEN respectively. The column is qualified by the databank identifier, and the table name, specified by the argument DID and TNAME respectively. If the table does not exist, a new table containing one column is created.

2.4.2 REMCO2 - Remove Column(s)

Purpose:

To remove a column in a table.

Format:

```
+----------------------------------------------+
|                                              |
|  REMCO2 (DID,TNAME,CNAME)                    |
|                                              |
+----------------------------------------------+
```

Arguments:

DID	4Iw inout	-	databank identifier DID(1) return code: < 0 - error = 0 - ok
TNAME	C8 in	-	table name
CNAME	C8 in	-	column name

Description:

The definition of the column, specified by the argument CNAME, is removed. The column is qualified by the databank identifier, and the table name, specified by the argument DID, and TNAME respectively.

Notes:

This operation may only be performed by a user who has X-privilege on the corresponding databank.

No cursors may be opened on the table involved during this operation.

At present, columns can only be removed when the table is empty, i.e. contains no data.

All other columns must be removed before removing last primary key column.

It is not possible to remove any column from a **system** table.

The possibility of removing a whole table is provided, by setting CNAME as '*'.

2.4.4 REMIX2 - Remove Index(es)

Purpose:

To remove a secondary index on a column.

Format:

```
+----------------------------------------------------+
|                                                    |
|  REMIX2 (DID,TNAME,CNAME)                          |
|                                                    |
+----------------------------------------------------+
```

Arguments:

DID	4Iw inout	-	databank identifier DID(1) return code: < 0 - error = 0 - ok
TNAME	C8 in	-	table name
CNAME	C8 in	-	column name

Description:

The secondary index on the column, specified by the argument CNAME, is removed. The column is qualified by the databank identifier, and the table name, specified by the arguments DID, and TNAME respectively.

Notes:

This operation may only be performed by a user who has X-privilege on the corresponding databank.

No cursors may be opened on the table involved during this operation.

The possibility of removing all secondary indexes in a table is provided, by setting CNAME as '*'.

2.5.1 DEFAC2 - Define Access

Purpose:

To define (or redefine) an access privilege specification.

Format:

```
+-----------------------------------------------+
|                                               |
|    DEFAC2  (RCODE,UNAME,DNAME,AOP)            |
|                                               |
+-----------------------------------------------+
```

Arguments:

RCODE	Iw out	- return code: < 0 - error = 0 - ok > 0 - ok (replaced)
UNAME	C8 in	- user name
DNAME	C8 in	- databank name
AOP	C1 in	- access option: 'P' - private 'R' - read only 'S' - shared 'X' - exclusive

Description:

An access privilege specification, which contains user name, databank name, and access, specified by the arguments UNAME, DNAME, and AOP respectively, will be inserted into the system table *ACCESS. If the specification already exists, the access privilege is replaced.

Notes:

This operation may only be performed by a user who has X-privilege on the corresponding databank.

Access privileges may not be defined for the databanks TRANSDB and LOGDB.

2.5.3 DEFDB2 - Define Databank

Purpose:

To define (or redefine) a databank specification.

Format:

```
+----------------------------------------------------+
|                                                    |
|  DEFDB2 (RCODE,DNAME,AOP,FNAME,SIZE)               |
|                                                    |
+----------------------------------------------------+
```

Arguments:

RCODE	Iw out	- return code: < 0 - error = 0 - ok > 0 - ok (replaced)
DNAME	C8 in	- databank name
AOP	C1 in	- access option: 'B' - backup (private) 'P' - private 'R' - read only 'S' - shared 'X' - exclusive
FNAME	Cx in	- physical file name
SIZE	Iw in	- databank size (number of pages)

Description:

A databank specification, which contains databank name, general access option, and physical file name, specified by the arguments DNAME, AOP, and FNAME respecitively, will be inserted into the system table *DBDEF. If the specification already exists, the access option and the physical file name values are replaced. Additionally, if the databank size, specified by the argument SIZE, is greater than zero, the physical file is formatted as a new MIMER databank, initially consisting of the number of pages stated. On the other hand, if the size value equals zero, the physical file is only certified to be an existing MIMER databank.

Notes:

This operation may only be performed by a user who has X-authority.

The databank involved in this operation may not be open at the same time.

When defining the databanks TRANSDB and LOGDB, the access option must be 'P' (private).

2.5.5 DEFUS2 - Define User

Purpose:

To define (or redefine) a user specification.

Format:

```
+------------------------------------------------------+
|                                                      |
|    DEFUS2 (RCODE,UNAME,AUTH,PASSW)                   |
|                                                      |
+------------------------------------------------------+
```

Arguments:

RCODE	Iw out	- return code: < 0 - error = 0 - ok > 0 - ok (replaced)
UNAME	C8 in	- user name
AUTH	C1 in	- authority: 'S' - standard 'X' - exclusive
PASSW	C8 inout	- password

Description:

A user specification, which contains user name, authority, and password, specified by the arguments UNAME, AUTH, and PASSW respectively, will be inserted into the system table *USERDEF. If the specification already exists, the authority and the password values are replaced.

Notes:

This operation may only be performed by a user who has X-authority. There is one exception. Any user may use DEFUS2 in order to change his own password.

The contents of the password argument is always destroyed.

2.6 Data Base Service

The following routines are available:

COPYD2	-	Copy Databank
RESTD2	-	Restore Databank
GENDB2	-	Generate System Databank

2.6.2 RESTD2 - Restore Databank

Purpose:

To restore a databank, using a backup-copy together with an optional log-file.

Format:

```
+-----------------------------------------------------+
|                                                     |
|    RESTD2 (RCODE,DNAME,BNAME)                       |
|                                                     |
+-----------------------------------------------------+
```

Arguments:

RCODE	Iw out	- return code: < 0 - error = 0 - ok
DNAME	C8 in	- databank name
BNAME	C8 in	- databank name (backup-copy)

Description:

The backup databank, specified by the argument BNAME, is copied to the databank, specified by the argument DNAME. Subsequently, if a log-file exists for the databank, all transactions recorded are applied.

Notes:

This operation may only be performed by a user who has X-authority.

The backup databank must have the general access 'B' (backup).

Appendix A

MIMER/DB ERROR CODES

When an error is detected by a MIMER/DB routine, the call-error-exit routine EXITC1 is entered with a negative four-digit error-code as argument value. The actions taken in EXITC1 are either installation or application-dependent, and are further discussed in Appendix B.

The first two digits of the error-code describes in which routine the error was detected, and the last two digits describes the error itself.

The last two digits of the error-code are specified as follows:

xx11	-	First argument value incorrect
xx12	-	Second argument value incorrect
xx13	-	Third argument value incorrect
xx14	-	Fourth argument value incorrect
xx15	-	Fifth argument value incorrect
xx16	-	Sixth argument value incorrect
xx17	-	Seventh argument value incorrect
xx18	-	Eighth argument value incorrect
xx21	-	Requested operation not allowed (illegal sequence)
xx22	-	Tried to add OR-condition after SET2
xx23	-	Tried to grant privilege on TRANSDB or LOGDB
xx24	-	Databank has already been opened
xx25	-	Table has already been opened (X-access involved)
xx26	-	Tried to remove last primary key incorrectly
xx27	-	Tried to exceed maximum number of columns
xx28	-	Tried to change a non-empty table
xx29	-	Tried to exceed maximum row-length
xx31	-	System control-block-area exhausted
xx32	-	No access rights for requested operation
xx33	-	Tried to use transaction handling without TRANSDB
xx34	-	Transaction management required
xx35	-	Operation not allowed, SYSDB opened for read only
xx41	-	Databank open error (installation dependent)
xx..	-	....
xx49	-	Databank open error (installation dependent)
xx51	-	System databank-identifier-area exhausted
xx52	-	Tried to open a non-MIMER databank
xx53	-	Tried to open a non-restarted databank
xx54	-	Tried to open a write-protected databank
xx61	-	Databank disk-space exhausted
xx62	-	LOGDB disk-space exhausted
xx63	-	TRANSDB disk-space exhausted
xx71	-	System locked (communication-area exhausted)
xx72	-	Databank locked by other user(s)
xx73	-	Table locked by other user(s)
xx81	-	SYSDB corrupted (should not occur)
xx82	-	Databank corrupted (should not occur)
xx83	-	System stack exhausted (should not occur)
xx84	-	Internal error, please submit error report

Appendix B

MIMER/DB EXIT ROUTINE DESCRIPTIONS

MIMER/DB has currently two application exit routines:

```
+-------------------------------------------------+
|                                                 |
|    EXITA1 - Allocation-exit                     |
|                                                 |
|    EXITC1 - Call-error-exit                     |
|                                                 |
+-------------------------------------------------+
```

Now, let us consider an example in single-user mode:

Suppose that EXITA1 is installed with a default size of the system contol-block-area set to 2048 words, the buffer-pool size set to 10240 words, the maximum number of concurrently opened databanks set to 16, the default access mode for SYSDB set to update (1), and default physical file-name of SYSDB set to "SYSDB":

```
+----------------------------------------------------+
|                                                    |
|        SUBROUTINE          EXITA1                  |
|        INTEGER             W,V                     |
|        COMMON /STORE2/     W(2048)                 |
|        COMMON /STORE1/     V(10240)                |
|        W(1)=2048                                   |
|        V(1)=10240                                  |
|        V(2)=16                                     |
|        V(3)=1                                      |
|        CALL MDRMVB(V(4),0,8HSYSDB    ,0,8)         |
|        RETURN                                      |
|        END                                         |
|                                                    |
+----------------------------------------------------+
```

If an application program requires the following environment: a buffer-pool size of 40960 words, the access mode for SYSDB set to read only (0), and the physical file-name of SYSDB to be "COMDB"; the user may link the following private EXITA1-routine:

```
+----------------------------------------------------+
|        SUBROUTINE          EXITA1                  |
|        INTEGER             W,V                     |
|        COMMON /STORE2/     W(2048)                 |
|        COMMON /STORE1/     V(40960)                |
|        W(1)=2048                                   |
|        V(1)=40960                                  |
|        V(2)=16                                     |
|        V(3)=0                                      |
|        CALL MDRMVB(V(4),0,8HCOMDB    ,0,8)         |
|        RETURN                                      |
|        END                                         |
|                                                    |
+----------------------------------------------------+
```

Appendix C

MACHINE DEPENDENT ROUTINE DESCRIPTIONS

The machine dependent routines are grouped as follows:

- Direct Access I/O Routines
- Sequential Access I/O Routines
- Character String Routines
- Data Conversion Routines
- Data Move Routines
- Special Routines

The I/O routines and some of the special routines are the interface between MIMER and the operating system. The other routines are used as an extension of the FORTRAN-66 language. The machine dependent routines may very well be used by other programs outside the MIMER family, provided that the programming language is compatable with code generated by the FORTRAN compiler.

MDRDAO - Direct Access Open

Purpose:

To open a direct access file.

Format:

```
+------------------------------------------------+
|                                                |
|  MDRDAO (FCB,NAME)                             |
|                                                |
+------------------------------------------------+
```

Arguments:

FCB	3Iw	inout	file control block
		in	FCB(1) access option: < 0 - create = 0 - update > 0 - read
		in	FCB(3) number of blocks to allocate
		out	FCB(1) return code: < 0 - error = 0 - ok
		out	FCB(2) file identifier
NAME	Cx	in	file name

Description:

The direct access file associated with the name, specified by the argument NAME, is opened with the access option given in the file control block, specified by the argument FCB.

Note:

When create option is given, FCB(3) denotes the number of blocks to be physically allocated on disk.

MDRDAW - Direct Access Write

Purpose:

To write a block to a direct access file.

Format:

```
+--------------------------------------------------+
|                                                  |
|  MDRDAW (FCB,IOAREA)                             |
|                                                  |
+--------------------------------------------------+
```

Arguments:

FCB	3Iw	inout	file control block
		in	FCB(2) file indentifier
		in	FCB(3) relative block number
		out	FCB(1) return code: < 0 - error = 0 - ok
IOAREA		ref	I/O-area

Description:

The block located in the I/O-area, specified by the argument IOAREA, is written to the direct access file, specified by the argument FCB.

MDRSIO - Start asynchronous I/O

Purpose:

To start a direct access I/O-operation.

Format:

```
+-----------------------------------------------------+
|                                                     |
|  MDRSIO (FCB,IOAREA)                                |
|                                                     |
+-----------------------------------------------------+
```

Arguments:

FCB	3Iw	inout	file control block
		in	FCB(1) operation code: < 0 - read = 0 - read > 0 - write
		in	FCB(2) file identifier
		in	FCB(3) relative block number
			FCB(1) access identifier (>0)
IOAREA		ref	I/O-area

Description:

A direct access I/O-operation is started by using the options given in the file control block, specified by the argument FCB, and as cache memory uses the I/O-area, specified by the argument IOAREA.

Note:

On computers where asynchronous I/O is not applicable, this routine will act as either MDRDAR or MDRDAW.

SEQUENTIAL ACCESS I/O ROUTINES

The following routines are available:

MDRSAO	- Sequential Access Open
MDRSAR	- Sequential Access Read
MDRSAW	- Sequential Access Write
MDRSAC	- Sequential Access Close

MDRSAC - Sequential Access Close

Purpose:

To close a sequential access file.

Format:

```
+--------------------------------------------------+
|                                                  |
|  MDRSAC (FCB)                                    |
|                                                  |
+--------------------------------------------------+
```

Arguments:

FCB	3Iw	inout	file control block
		in	FCB(1)=xy: x : print option = 0 - no = 1 - yes y : file disposition = 0 - keep = 1 - delete
		in	FCB(2) file identifier
		out	FCB(1) return code: < 0 - error = 0 - ok > 0 - ok, but only keep file

Description:

The sequential access file, specified by the argument FCB, is closed.

Note:

On some implementations the possibility to print and/or delete the file might not be supported.

MDRMVC - Move Characters

Purpose:

To move character string.

Format:

```
+------------------------------------------------+
|                                                |
|  MDRMVC (STR1,OFF1,LEN1,STR2,OFF2,LEN2,FILL)   |
|                                                |
+------------------------------------------------+
```

Arguments:

STR1		ref	first string address
OFF1	Iw	in	first string offset
LEN1	Iw	in	first string length
STR2		ref	second string address
OFF2	Iw	in	second string offset
LEN2	Iw	in	second string length
FILL	ref		fill character

Description:

The first string, specified by the arguments STR1, OFF1 and LEN1, is replaced by the second string, specified by the arguments STR2, OFF2 and LEN2. If the first string is longer than the second string, the highest addressed bytes of the first string are replaced by the fill character, specified by the argument FILL. If the first string is shorter than the second string, the highest addressed bytes of the second string are not moved.

Note:

If the strings are overlapping, and the first string has a higher starting address than the second string, the result is unpredictable.

MDRMVF - Move Character Fill

Purpose:

To fill a block of memory with a fill character.

Format:

```
+-----------------------------------------------------+
|                                                     |
|  MDRMVF (STRING,OFFSET,LENGTH,FILL)                 |
|                                                     |
+-----------------------------------------------------+
```

Arguments:

STRING		ref	character string address
OFFSET	Iw	in	character string offset
LENGTH	Iw	in	character string length
FILL		ref	fill character

Description:

The bytes in the string, specified by the arguments STRING, OFFSET and LENGTH are replaced by the fill character, specified by the argument FILL.

Note:

MDRMVF is a special case of MDRMVC with LEN2 = 0.

MDRCLB - Compare Logical Character Blocks

Purpose:

To compare one block of memory with another.

Format:

```
+-----------------------------------------------------+
|                                                     |
|  MDRCLB (STR1,OFF1,STR2,OFF2,LENGTH,COND)           |
|                                                     |
+-----------------------------------------------------+
```

Arguments:

STR1		ref	first string address
OFF1	Iw	in	first string offset
STR2		ref	second string address
OFF2	Iw	in	second string offset
LENGTH	Iw	in	length of first and second string
COND	Iw	out	return code: < 0 - first string low = 0 - strings are equal > 0 - first string high

Description:

The bytes of the first string, specified by the arguments STR1, OFF1 and LENGTH, are compared with the bytes of the second string, specified by the arguments STR2, OFF2 and LENGTH. Comparison proceeds until inequality is detected, or all the bytes of the strings have been examined.

Note:

MDRCLB is a special case of MDRCLC with LEN1 = LEN2, except that the absolute value of COND does not reflect any position.

MDRSKC - Skip Character

Purpose:

To skip character in character string.

Format:

```
+--------------------------------------------------+
|                                                  |
|   MDRSKC (STRING,OFFSET,LENGTH,CHAR,COND)        |
|                                                  |
+--------------------------------------------------+
```

Arguments:

STRING		ref	character string address
OFFSET	Iw	in	character string offset
LENGTH	Iw	in	character string length
CHAR		ref	character
COND	Iw	out	return code: = 0 - not found > 0 - found

Description:

The character, specified by the argument CHAR, is compared with the bytes of the string, specified by the arguments STRING, OFFSET and LENGTH. Comparison continues until inequality is detected, or until all bytes of the string have been compared. In case of inequality, COND is set to the position of the last processed byte in the string.

Note:

If LENGTH is negative, the absolute value of LENGTH is used as string length, and the bytes of the string are processed backwards, from the highest addressed byte to the lowest.

MDRSPC - Span Characters

Purpose:

To skip a set of characters in character string.

Format:

```
+---------------------------------------------------------+
|                                                         |
|  MDRSPC (STRING,OFFSET,LENGTH,TABLE,MASK,COND)          |
|                                                         |
+---------------------------------------------------------+
```

Arguments:

STRING		ref	character string address
OFFSET	Iw	in	character string offset
LENGTH	Iw	in	character string length
TABLE		ref	table
MASK		ref	mask byte
COND	Iw	out	return code: = 0 - not found > 0 - found

Description:

The bytes of the string, specified by the arguments STRING, OFFSET, and LENGTH, are successively used to index into a byte table, whose zero entry address is specified by the argument TABLE. The byte selected from the table is ANDed with the mask byte, specified by the argument MASK. The operation continues until the result of the AND is zero, or until the bytes of the string are exhausted. If the operation is terminated with a zero AND result, COND is set to the position of the last processed byte in the string.

MDRTRC - Translate Characters

Purpose:

To translate character string.

Format:

```
+-----------------------------------------------------+
|                                                     |
|  MDRTRC (STRING,OFFSET,LENGTH,TABLE)                |
|                                                     |
+-----------------------------------------------------+
```

Arguments:

STRING		ref	character string address
OFFSET	Iw	in	character string offset
LENGTH	Iw	in	character string length
TABLE		ref	table

Description:

The bytes of the string, specified by the arguments STRING, OFFSET and LENGTH, are successively used to index into a byte table, whose zero entry address is specified by the argument TABLE. The byte selected from the table replaces the byte in the string. The operation continues until the bytes of the string are exhausted.

Note:

If the string and the table are overlapping, the result is unpredictable.

MDRCIC - Convert to Integer from Character

Purpose:

To convert data, from numeric character string representation, to integer number representation.

Format:

```
+-----------------------------------------------------+
|                                                     |
|  MDRCIC (INTEGV,STRING,OFFSET,LENGTH,COND)          |
|                                                     |
+-----------------------------------------------------+
```

Arguments:

INTEGV	Ix	out	integer number
STRING		ref	numeric character string address
OFFSET	Iw	in	numeric character string offset
LENGTH	Iw	in	numeric character string length
COND	Iw	out	return code: < 0 - undefined value = 0 - ok > 0 - overflow value

Description:

The numeric character string, specified by the arguments STRING, OFFSET and LENGTH, is converted to a rounded integer number, specified by the argument INTEGV. If overflow occurs, the integer number is set to the maximum value (IMAX), and COND > 0 is returned. If the character string is not numeric, or LENGTH < 1, the integer number is set to the undefined value (INULL), and COND > 0 is returned.

MDRCFC - Convert to Floating from Character

Purpose:

To convert data, from numeric character string representation, to floating point number representation.

Format:

```
+------------------------------------------------------+
|                                                      |
|     MDRCFC (FLOATV,STRING,OFFSET,LENGTH,COND)        |
|                                                      |
+------------------------------------------------------+
```

Arguments:

FLOATV	Fx	out	floating point number
STRING		ref	numeric character string address
OFFSET	Iw	in	numeric character string offset
LENGTH	Iw	in	numeric character string length
COND	Iw	out	return code : < 0 - undefined value = 0 - ok > 0 - overflow value

Description:

The numeric character string, specified by the arguments STRING, OFFSET and LENGTH, is converted to a floating point number, specified by the argument FLOATV. I overflow occurs, the floating point number is set to the signed maximum value (FMAX), and COND > 0 is returned. If the character string is not numeric, or LENGTH < 1, the floating point number is set to the undefined value (FNULL), and COND < 0 is returned.

MDRCIF - Convert to Integer from Floating

Purpose:

To convert data, from floating point number representation, to integer number representation.

Format:

```
+---------------------------------------------------+
|                                                   |
|    MDRCIF (INTEGV,FLOATV,COND)                    |
|                                                   |
+---------------------------------------------------+
```

Arguments:

INTEGV	Ix	out	integer number
FLOATV	Fx	in	floating point number
COND	Iw	out	return code: < 0 - undefined value = 0 - ok > 0 - overflow value

Description:

The floating point number, specified by the argument FLOATV, is converted to a rounded integer number, specified by the argument INTEGV. If overflow occurs, the integer number is set to the signed maximum value (IMAX), and COND > 0 is returned. If the floating point number is undefined (FNULL), the integer number is set to the undefined value (INULL), and COND < 0 is returned.

DATA MOVE ROUTINES

The following routines are available:

MDRMIA	- Move to Integer from Attribute
MDRMAI	- Move to Attribute from Integer
MDRMFA	- Move to Floating from Attribute
MDRMAF	- Move to Attribute from Floating

Notes:

An attribute byte string is a string which contains an integer number or a floating point number, not necessary on word boundary.

An attribute byte string is defined by using three arguments: ADDRESS, OFFSET and LENGTH. The lowest addressed byte of the string is determined by adding the value of the argument OFFSET to the address of the argument ADDRESS. Note that the byte offset value is valid even if it is negative. The number of bytes in the string is given by the value of the argument LENGTH.

The special-format is an internal representation of integer and floating point numbers, used in MIMER/DB, which makes it possible to use logical compare operations, not only for character strings, but also for integer and floating point numbers.

MDRMAI - Move to Attribute from Integer

Purpose:

To move an integer number, from an integer word, to an attribute byte string, optionally converted to special-format.

Format:

```
+------------------------------------------------+
|                                                |
|    MDRMAI (STRING,OFFSET,LENGTH,INTEGV,FLAG)   |
|                                                |
+------------------------------------------------+
```

Arguments:

STRING		ref	attribute byte string address
OFFSET	Iw	in	attribute byte string offset
LENGTH	Iw	in	attribute byte string length
INTEGV	Ix	in	integer number
FLAG	Iw	in	flag: = 0 - no conversion = 1 - convert to special-format

Description:

The integer number, specified by the argumet INTEGV, is optionally converted, depending on the value of the argument FLAG, and possibly compressed to the attribute byte string length. The resulting value is stored in the string, specified by the arguments STRING, OFFSET and LENGTH.

MDRMAF - Move to Attribute from Floating

Purpose:

To move a floating point nymber, from a floating point word,to an attribute byte string, optionally converted to special-format.

Format

```
+------------------------------------------------------+
|                                                      |
|    MDRMAF (STRING,OFFSET,LENGTH,FLOATV,FLAG)         |
|                                                      |
+------------------------------------------------------+
```

Arguments:

STRING		ref	attribute byte string address
OFFSET	Iw	in	attribute byte string offset
LENGTH	Iw	in	attribute byte string length
FLOATV	Fx	in	floating paint number
FLAG	Iw	in	flag : = 0 - no conversion = 1 - convert to special-format

Description:

The floating point number, specified by the argument FLOATV, is optionally converted, depending on the value of the argument FLAG, and possibly compressed to the attribute byte string length. The resulting value is stored in the string, specified by the arguments STRING, OFFSET and LENGTH.

MDRPDT - Provide Date and Time

Purpose:

To provide date and time from the system-clock.

Format

```
+------------------------------------------------------+
|                                                      |
|    MDRPDT (STRING)                                   |
|                                                      |
+------------------------------------------------------+
```

Arguments:

STRING	C16	out	-	numeric character string.

Description:

The current date and time is retrieved from the system-clock, and edited into the string, specified by the argument STRING, with the format 'YYYYMMDDHHMMSSth'.

MDRCRA - Compute Relative Address

Purpose:

To compute the byte address of a variable relative to another variable.

Format

```
+--------------------------------------------------------+
|                                                        |
|     MDRCRA (RELADR,ADDR1,ADDR2)                        |
|                                                        |
+--------------------------------------------------------+
```

Arguments:

RELADR	Iw	out	relative byte address (see note)
ADDR1		ref	first variable
ADDR2		ref	second variable

Description:

The address of the first variable, specified by the argument ADDR1, is subtracted from the address of the second variable, specified by the argument ADDR2, with the result converted to reflect the difference in bytes. The result is returned in the argument RELADR.

Note:

On 16-bit computers, the argument RELADR is the low-order part of the double word, where the relative byte address is returned.

MDRALI - Add Logical Interlocked

Purpose:

To perform a logical add operation in interlocked mode.

Format

```
+--------------------------------------------------+
|                                                  |
|   MDRALI (WORD,VALUE)                            |
|                                                  |
+--------------------------------------------------+
```

Arguments:

WORD	Iw	Inout	word (can be located in shared memory)
VALUE	Iw	in	value

Description:

The word, specified by the argument WORD, is updated by making a logical add with the value, specified by the argument VALUE. A possible carry is ignored. The operation is executed in interlocked mode, thus making it possible to update counters in shared memory.

MDRDEQ -Dequeue Semaphore (Signal)

Purpose:

To perform a signal-operation on a semaphore.

Format

```
+--------------------------------------------------+
|                                                  |
|    MDRDEQ (SEM)                                  |
|                                                  |
+--------------------------------------------------+
```

Arguments:

SEM	2Iw	inout	semaphore (located in shared memory)

Description:

The semaphore, specified by the argument SEM, will have its value increased by one. If the previous value was zero and a semaphore-queue exists, the waiting processes are made runnable and the queue is removed.

MDRALC - Alter case

Purpose:

To change case of alphabetical characters in a character string.

Format:

```
----------------------------------------------------

    MDRALC(STRING,OFFSET,LENGTH,CASE)

----------------------------------------------------
```

Arguments:

```
STRING     ref  - character string address
OFFSET Iw  in   - character string offset
LENGTH Iw  in   - character string length ,

CASE   Iw  in   - case code
                  = 1 - alter to lower case
                  = 2 - alter to upper case
```

Description:

The alphabetical characters of the string, specified by the arguments STRING, OFFSET and LENGTH, are converted to lower or upper case depending on the value of the argument CASE. This conversion is performed in place i.e directly in the input string.

Alphabetical characters are by default characters in the intervals A-Z and a-z. This definition can be changed by calls to MDRSCT.

In translation to upper case, upper case characters in the input string are normally unchanged, and the same holds for lower case input when lower case translation.

Note:

If LENGTH <= 0 or if CASE < 1 or CASE > 2 return only.

Note that the MDRSCT may define translation for any byte, 8-bit codes also.

MDROTF - Open terminal file

Purpose:

To open the terminal file

Format:

```
-------------------------------------------------------

    MDROTF (CODE)

-------------------------------------------------------
```

Arguments:

```
CODE   Iw out - return code

                   = -1 - error
                   else OK
```

Description:

Open terminal for read/write

Note:

MDRSCT - Set case translation

Purpose:

To change the definition of the alphabetical character set used in the MDRALC routine.

Format:

```
------------------------------------------------------------

   MDRSCT(CHAR,TRCHAR,CASE)

------------------------------------------------------------
```

Arguments:

```
   CHAR   C1  in   - character for which translation is to be changed
   TRCHAR C1  in   - character to which CHAR will be translated in
                     the new translation definition

   CASE   Iw  in   - case code:
                     = 1 - alter to lower case
                     = 2 - alter to upper case
```

Description:

This routine affects the translations which are performed by the MDRALC routine. MDRALC contains the definition of the alphabetical set of characters and their translations for each CASE. A call to MDRSCT changes that definition.

This routine will change the definition of translation character for CHAR to TRCHAR. The CASE parameter specifies if it is the translation to upper or to lower case that is to be affected.

Note:

If CASE < 1 or CASE > 2 return only.

Note that the MDRSCT may define any byte as character, 8-bit codes also.
The most common use of this routine is to define case translations for national characters.