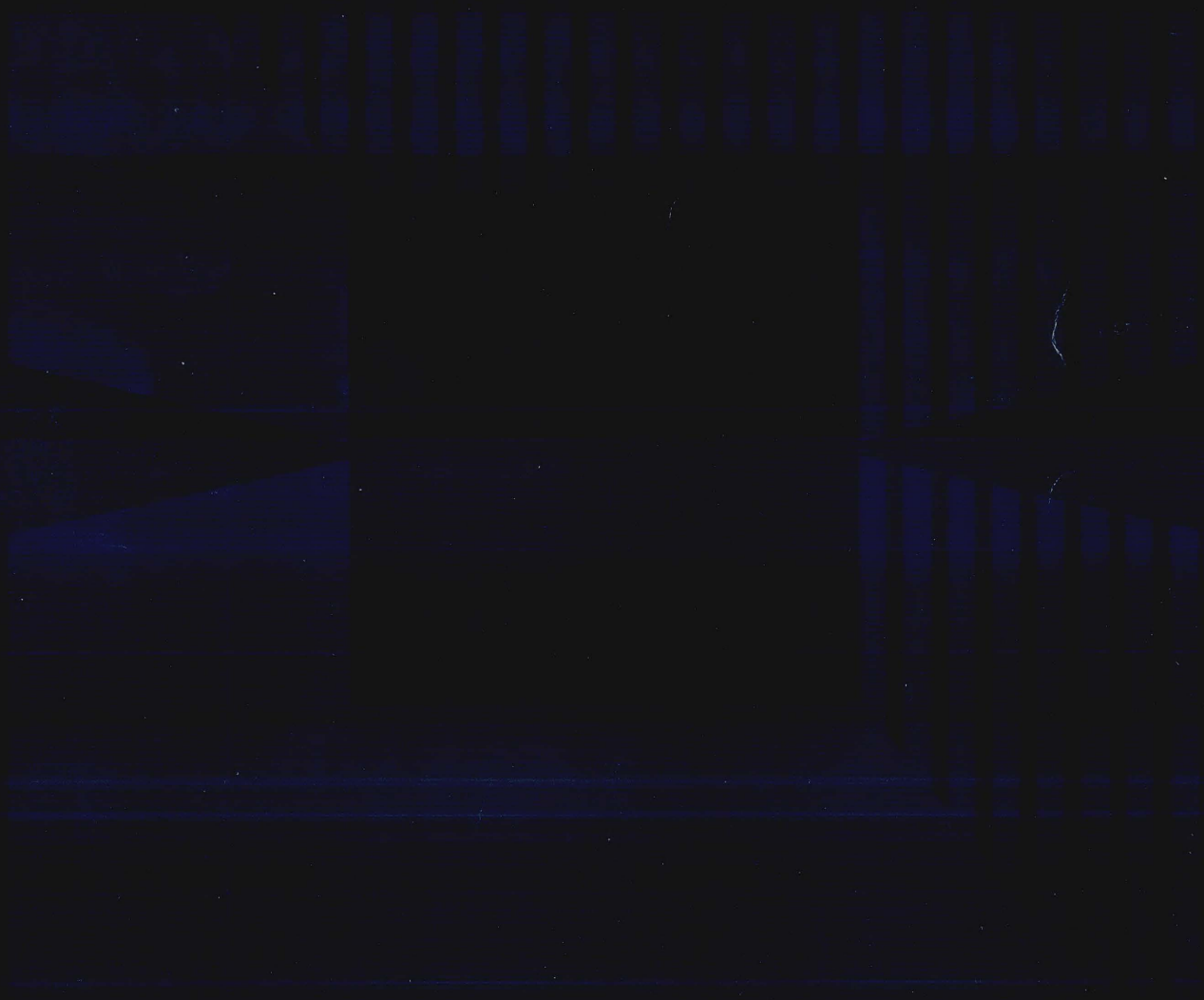
Britton Lee, Inc.

IDM 500

Intelligent Database Machine

Product Description



CONTENTS

1. INTRODUCTION TO THE IDM

The Intelligent Database Machine (IDM) is a tool for providing sophisticated end user database systems. It is an integrated hardware/software database computing system. The system is housed in a single rack-mountable 16 by 24 by 12 inch electronic chassis that is connected between user-supplied front-end systems and user-supplied disks. Front-end systems can be intelligent (OEM programmable) terminals; micro, mini, and/or mainframe computer systems. The IDM performs the following functions:

IDM Database Management Functions

- transaction management
  - guarantees database consistency
- optional logging of database changes
  - for audit trail
- indexing of data
  - for fast access and update
- crash recovery facilities
- large RAM cache
  - to hold indices and frequently used disk blocks
- concurrency control
  - multiple users can access the same database
- protection features
  - unauthorized users may not access the database
- data definition facilities
  - dynamic data definitions
- data manipulation facilities
  - high level, non-procedural functions
- data independence

The IDM is a back-end system and performs data management functions at speeds up to 10 times faster than conventional systems. The speed is obtained by the use of special-purpose hardware. The IDM includes:

Hardware

- an optional special-purpose 10 mip database processor
- a general-purpose processor
- disk controllers
- ram memory
- I/O processors
- a high-speed bus
- rack-mountable chassis

Software

- complete relational database management system
- random access file system

OEM Support

- the complete definition for a general-purpose query language
- extensive documentation
- training sessions

Both the hardware and the software systems are designed and manufactured by Britton-Lee. This document contains an overview of the IDM, description of and configuration guides for the hardware system, and specification of the software system, including the interface to the database management system.

Overview

The IDM is a database computing system that can be used as a back-end machine or, together with intelligent terminals, as a stand-alone data management system. Figures 1.1 and 1.2 show the IDM in both configurations.

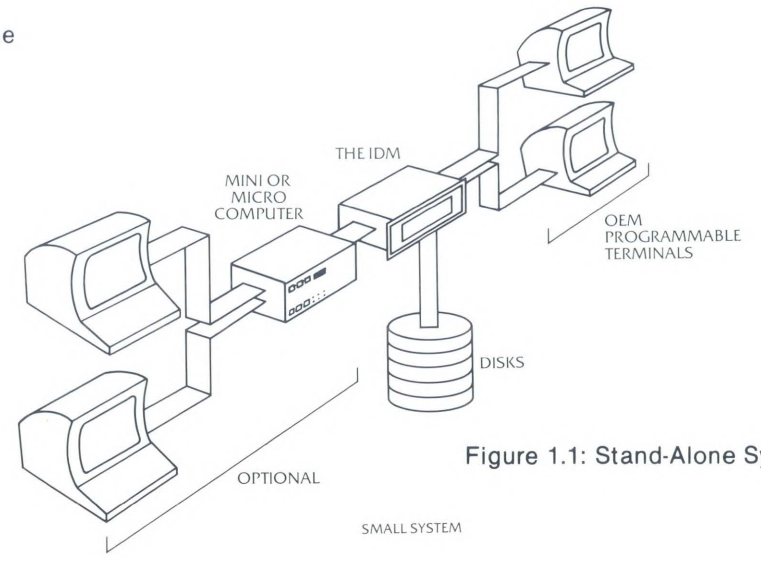


Figure 1.1: Stand-Alone System

The IDM requires pre-processing of user queries. This is a task which can easily be performed in an intelligent terminal.

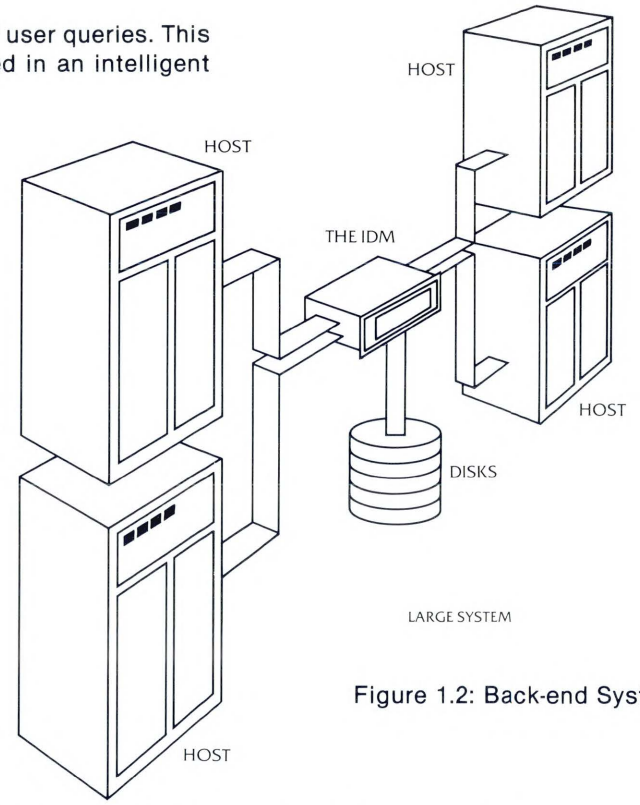


Figure 1.2: Back-end System

When used as a back-end database machine the IDM provides a powerful central data management facility. Several computers can share access to data stored on an IDM because concurrency control, scheduling, and protection functions are implemented within the IDM itself.

Whether the user is communicating with the IDM through an intelligent terminal or through a large mainframe, the user never has to program the IDM itself. The IDM is a complete database software/hardware package that is fully programmed. Database commands are preprocessed in the front-end system (the terminal or mainframe) and sent to the IDM. After the commands are processed, the IDM buffers the results until the front-end can receive them, then sends the results. The user does not intervene in the processing of a database command from the moment it is received by the IDM until the results are sent back. This processing is extraordinarily fast because the hardware and software are specifically designed for data management.

Software

The IDM contains a complete relational database management system. Relational data management systems were developed at research institutions and have been found to provide the best performance and data organization for back-end database machines. A relational database management system organizes data into "relations". A relation is a table: it has rows and columns. A row is technically termed a "tuple", and a column an "attribute". Relations can also be thought of as "files", tuples as "records" and attributes as "fields". Figure 1.3 shows the logical organization of two databases.

DATABASE: PERSONNEL

EMPLOYEES

NAME	NUMBER	ADDRESS	SALARY	DEPT.
JONES	2501	210 12th St.	50000	25
SMITH	8912	404 4th St.	12000	12
VITALE	1573	809 24th St.	7000	7

DEPARTMENTS

NAME	NUMBER	MANAGER
SHOE	25	2715
HAT	12	8912
BOOK	7	1531

RELATION OR TABLE

COLUMN, OR DOMAIN

Figure 1.3

Figure 1.3 (continued)

DATA BASE: INVENTORY

PARTS

NAME	NUMBER	IN. STOCK	PRICE
HAT	250931B	12	35
HAT	250970B	4	70
SHOE	370210B	14	25

ORDERED

VENDOR	ORDER DATE	QUANTITY	PRICE	NUMBER
HATS, INC.	80/06/06	25	20	250931B
B.JONES, INC.	80/09/05	30	10	71025A
LEATHER, INC.	80/07/02	12	15	370210B

TUPLE, OR ROW

The IDM will manage up to 50 databases. Each database can have up to 32,000 relations. There may be up to 2 billion tuples per relation. Each tuple can contain up to 2000 bytes.

The IDM database management system provides data independence, so the user programs do not need to be changed when the database is changed. It is a non-procedural system: that is, database commands specify only what is required, and are not concerned with the actual structure of the data.

A complete description of the IDM software is contained in Section 4. The IDM database management system includes the following basic database commands:

- create, destroy database
- create, destroy relation
- retrieve data
- change data
- delete a tuple (record)
- add a tuple (record)
- create, destroy index

In order for a data management system to be useful, more than the above commands must be provided. The IDM, since it is a complete date management system, also includes the following features:

- **transaction management**
  - The user defines the beginning and end of transactions; the entire transaction is either completely finished, or not started.
- **transaction logging**
  - for audit trails and crash recovery
- **relation (table) load and dump facilities**
  - for data backup and fast loading of the database.
- **protection of data from unwarranted access**
- **partial view of the data**
  - selected users can only see part of the data
- **concurrency control**
  - changes to the data are coordinated so more than one user at a time can safely interact with the same database.

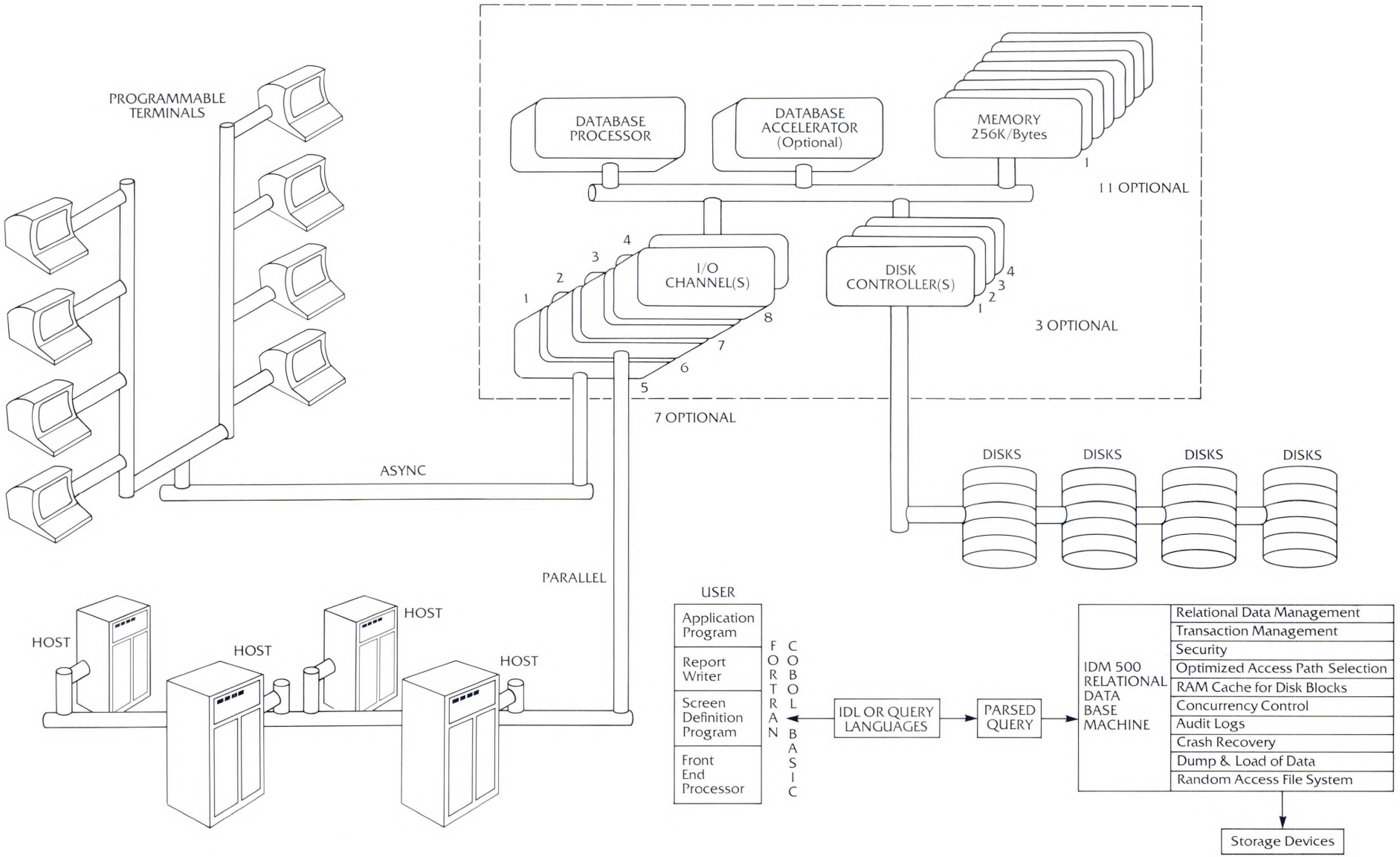
A unique feature of the IDM data management system is the stored command. A stored command is one that has been previously defined by the user, and is stored in a partially processed form in the IDM. From then on, the command is referred to by the user only as a short name or number. Since the command is stored in the IDM, the transmission time from the front-end to the IDM is minimized; since it is already partially processed, execution time is also minimized.

Because one configuration of the IDM database computing system is a stand-alone system, the IDM software also includes a general purpose file system. The file system is also described in Section 4.

OEM Support

The IDM accepts commands that have been preprocessed by the front-end system. An OEM must therefore write the programs that translate a user query from the form the user types in at a terminal to the IDM-internal form expected by the IDM. This translation consists of parsing the user query: that is, putting it into a standard form understood by the IDM. To facilitate that task, Britton-Lee will provide the complete specifications for a general-purpose query language, IDL. IDL (Intelligent Database Language) translates easily into the IDM-internal form. In Section 3 several examples of user interfaces to the IDM are given. It is not necessary to use IDL; one of the examples in Section 3 does not. It is simply provided as a tool and an example for the OEM.

The Britton-Lee Intelligent Database Machine



Hardware

Figure 1.4 is a block diagram of the IDM's hardware architecture.

Section 2 discusses, in detail, the IDM architecture. The following is an overview. The IDM can be configured with up to 16 boards. the configuration guide is in Section 7.

In figure 1.4 the front-end system is shown communicating with the IDM through the I/O processors. There are two types of I/O processors: serial (RS232C) and parallel (GPIB, IEEE-488). Up to 8 asynchronous lines can be connected to each serial I/O processor. Since the GPIB is a multiplexed interface, several front-end systems can be connected through one parallel I/O processor. The IDM can be configured with up to eight I/O processors (4 serial and 4 parallel).

The Database Accelerator is designed to perform time-critical data management functions, while the Database Processor performs the less time-critical functions. The Database Accelerator is an optional feature of the system; in its absence the Database Processor accommodates its functions.

The RAM cache is provided in units of 256K bytes each; up to twelve units can be connected to one system (subject to the limitation that the sum of all units may not be more than 16). The RAM cache is used by the IDM to cache pointers, store frequently used disk pages, and to store data associated with user commands.

The IDM controller is designed to operate with the Database Accelerator to process data at the speed that it is read from the disk. Up to 4 controller boards may be on the IDM. Each controller board will support up to 4 SMD disks.

Disk storage is supplied by the OEM. Section 7 (Product Support) discusses integrating the IDM with the users' disk storage.

Summary

The hardware and software of the IDM are designed to function jointly. They are not available separately. The IDM is up to 10 times faster than conventional software-only general purpose data management systems because it is an integrated hardware/software data management system.

2. THE IDM SYSTEM ARCHITECTURE

The IDM is an integration of hardware and software designed for the specific task of managing a relational database. This section provides a basic overview of the architecture.

Hardware Architecture

The IDM is organized around a central, very high speed bus. The design is broken up into independent, functional modules each of which performs a specialized task. A block diagram of the system is shown below.

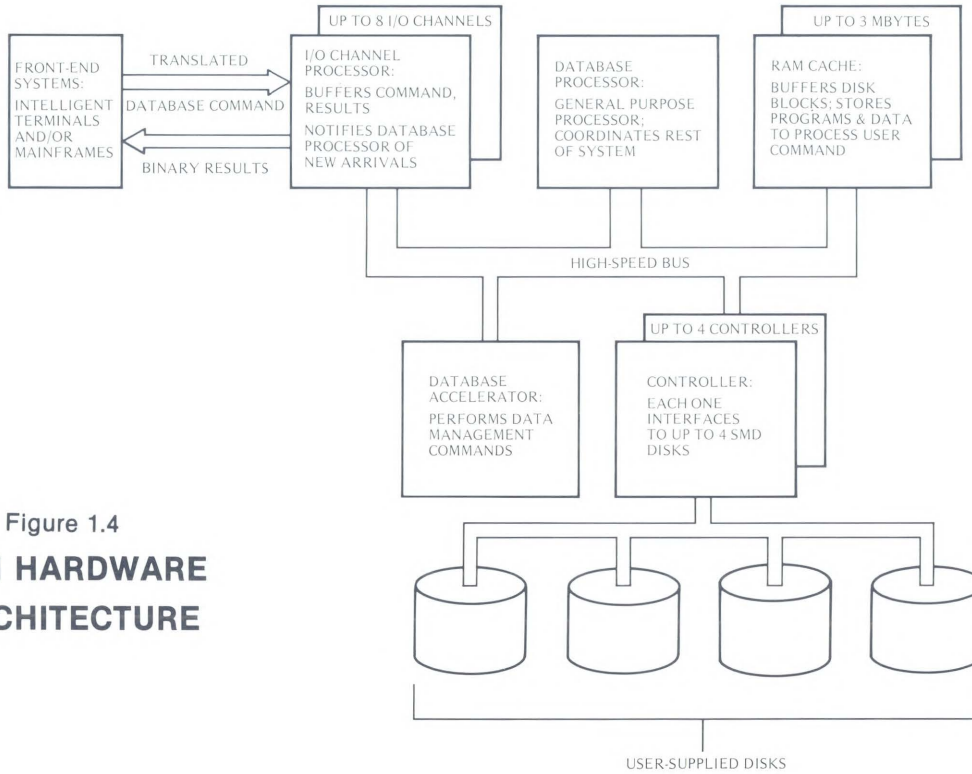


Figure 1.4

IDM HARDWARE ARCHITECTURE

Each component is a single board which plugs into the IDM bus. The components are:

1. *Database Processor (DBP)*

   The Database Processor manages all system resources and executes most of the software in the system.

2. *I/O Processor (SIO and PIO)*

   The I/O processor is a channel that connects a front-end device to the IDM. The I/O processor accepts commands from a host and returns results to the host. It performs all host dependent hand-shaking including checking that all data sent to or from the host has been correctly transmitted and received. In the event of an error during data transmission, the channel processor coordinates the retransmission of the data. The channel optionally converts various host data types to a standard, IDM format. For example, some hosts will send a two byte integer as the least significant byte followed by the most significant bytes. The I/O processor will swap bytes. When a command has been completely received or reaches a predetermined length, the Database Processor is notified and the command is transferred at high speed into main memory for processing.

   There are two types of I/O channels processors: parallel (PIO) and serial (SIO). The parallel channel provides byte parallel transmissions between host and channel at up to 250,000 bytes/second. The serial channel allows up to 8 separate bit serial lines to simultaneously communicate at individual speeds up to 19,200 bits/second.

3. *Disk Controller (CON)*

   The disk controller moves data between external disks and the IDM main memory. It performs burst error detection and correction. The IDM uses a 2K byte disk page size. The controller disk interface is SMD CDC 9760 compatible. This allows disk drives made by many different vendors to be directly connected to the IDM.

   One controller manages from 1 to 4 disk drives. Each drive can be a different capacity. New drives can be added at any time. Drives dedicated to the IDM can only be accessed through the IDM.

4. *Database Accelerator (DAC)*

   The Database Accelerator is a very high speed processor with an instruction set specifically designed to support a relational database. Its instruction time is 100ns. The Database Accelerator performs specific functions under the direction of the Database Processor. It can initiate disk activity and can search disk pages while they are being transferred into memory. In most cases the Database Accelerator can complete processing of a page by the time the page has been completely read in. The Database Accelerator and the IDM system software are structured so that most of the time consuming processing is performed by the DAC.

5. *Memory Timing and Control (MTC)*

   The Memory Timing and Control manages the main memory subsystem. It provides single bit error correction and double bit error detection. The MTC controls one or more memory storage boards. The board is pipelined to supply data at the processing speed of the Database Accelerator.

6. *Memory Storage (MEM)*

   An IDM can have from one to twelve semiconductor memory storage boards. These boards are managed by the Memory Timing and Control.

BASIC SYSTEM

A basic IDM configuration consists of:

- Database Processor
- Memory Timing and Control
- 256K bytes of Memory
- One Disk Controller
- One I/O Processor

The architecture of the IDM allows it to operate with or without the Database Accelerator. The presence of the Database Accelerator provides a dramatic increase in system performance. For applications which require only a modest transaction rate, the Database Accelerator can be omitted, providing a cost savings.

More disk controllers can be added to the system up to a total of 16 drives. The maximum storage capacity of all disks is limited to 32 gigabytes.

Additional I/O channels can be added to increase the number of hosts and terminals directly connected to the IDM. Finally, additional main memory can be added up to a limit of 3 megabytes. Section 6 discusses how to configure an IDM.

Software Architecture

The software features of particular importance to the OEM are:

- logical data organization
- integrated data dictionary
- physical data organization
- user process organization
- crash recovery

Logical Data Organization

The user of the IDM sees data organized into one or more independent databases. Each database is a collection of relations or files. A user on a host computer opens a database and then issues commands to manipulate the data in the database.

A "relation" is an object similar to a "file" in other systems. As an example, there could be a database designated "personnel" and inside that database there could be many relations, one of which would hold information about employees:

employee relation

name	number	deptno	salary
Baker, R.	A8080	17	25000.00
Booth, J.W.	B3083081	3	27500.00

This example shows a relation called "employee" with column names (also called "attributes" or "domains") name, number, deptno, and salary. Each "row" in the relation (also called "record" or "tuple") represents specific data about one employee.

There are very few limitations to the sizes of relations in the IDM. Each database can have up to 32,000 separate relations. Each relation has from 1 to 250 columns and up to 2 billion rows. A row is limited to 2000 bytes. Section 4 explains how relations are created and accessed.

Integrated Data Dictionary

Both the IDM and the user need to know what information is kept in a database. This information is commonly called a "data dictionary". The IDM maintains this information in relations which are available to the user. When a database is created by the user, the IDM will create a set of standard relations which comprise the data dictionary. The information in the data dictionary includes:

- Each relation, its name and an optional description.
- Each column name and its type (character, integer, etc.)
- The indices on each relation
- A list of database users and their access privileges.
- An audit trail of changes made to the database.
- Other information.

Users do not need special commands to access the data dictionary. The integrated data dictionary allows the OEM user to integrate sophisticated data descriptions with the IDM defined relations.

All IDMs contain one special database called the "system" database. This database is created when the IDM is first installed and holds information about the IDM system and about the databases on the IDM. In addition to the data dictionary found in user databases, the "system" database contains:

- list of all databases
- list of all disks and their specifications.
- physical allocation of disks to databases and relations.
- IDM binary software.
- log of system failures.

The system database is accessed exactly like any other database and is of particular interest to personnel maintaining the IDM.

Physical Organization

Physically, the databases are stored on disk drives. Each disk drive is given a name supplied by the user together with the necessary drive information including drive capacity, number of heads, and bits per track. The IDM uses the information to format the drive initially and for allocating space on the drive. The IDM subdivides each disk drive into "zones". A zone is always a whole number of disk cylinders. When a database is created, the user can specify the number of zones to give to the database. As the database grows, more zones can be added. Each zone is further subdivided into 2048 byte "pages". A page is a basic unit of allocation. As relations grow they are allocated one or more pages at a time. The number of pages per zone depends on the particular capacity and organization of the disk. The user of the IDM may control how much space a database or relation can occupy on each disk.

User Process Organization

The IDM supports a large number of users simultaneously accessing the same or different relations. It contains all the logic needed to control the processing order of different requests and to prevent the different requests from interfering with each other. One user can have many commands running in parallel on the IDM.

To access a database, a user sends an "open database" command to the IDM. If the database exists then a process is created on the IDM which manages commands for that particular user on the specified database.

The IDM process will service the user's requests until a "close database" command is received. The IDM allows an arbitrary number of users to simultaneously access the same or different databases. Main memory is used both to store the user command and as a private workspace for processing. The IDM has been designed to use a minimum amount of main memory per user process. If the IDM does not currently have a command to process for a particular user, that user's process requires no main memory. As the command is processed it can use up to 18K bytes of memory. The typical command will use about 8K bytes. The actual amount depends on the complexity of the command.

When the IDM has many commands all in various states of execution, it will schedule among them giving each a priority which depends on various parameters. If there is insufficient main memory to hold all active user processes, they will be moved to temporary storage on disk.

Crash Recovery

The IDM is a "transaction" oriented system which provides complete database consistency. The user may group one or more commands into a "transaction". The IDM will guarantee that either the entire transaction is performed or none of it. This is crucial both for consistency during multiple user interactions and in case of a system wide failure. When the IDM detects a failure condition, (for example, loss of power, host processor failure, or IDM hardware failure), it is capable of maintaining database consistency. This is achieved by keeping redundant information at certain critical points in the processing of a transaction. As is true of all disk based systems, certain failures such as a head crash may not be recoverable. In such a case, the IDM provides the ability to reload the database from a previous checkpoint or dump.

The IDM System

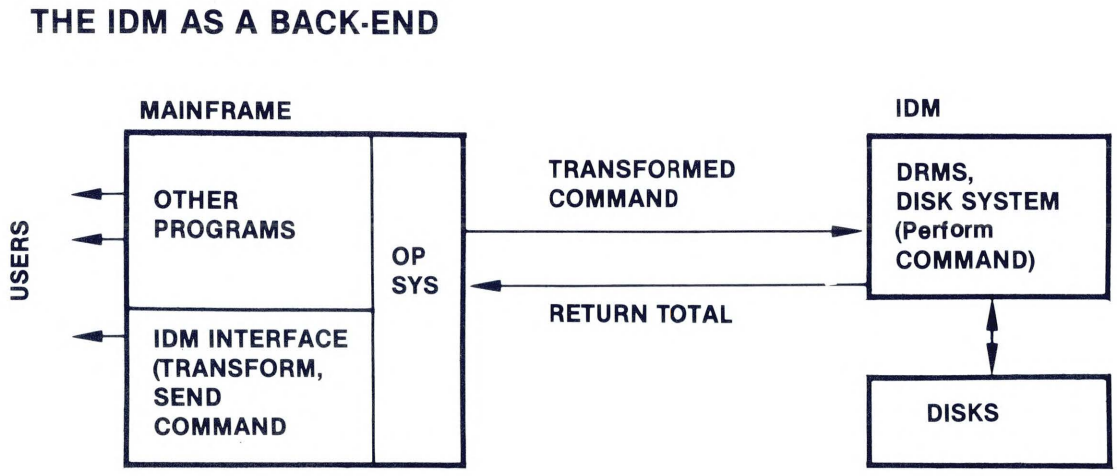
All the components of the IDM are integrated to support a common task of efficiently and reliably supporting a database. The next section overviews how to program the IDM user interface.

3. PROGRAMMING THE USER INTERFACE

The IDM is a complete database computing system that depends on its front-end to provide the direct interface to the end-users. There are two types of "users" of the IDM: those who buy multiple IDMs and write the programs to interface the IDM to their systems: throughout this document we refer to this user as "the OEM". The second type of user is the person for whom the OEMs are developing their products. This person sees only the combined OEM/IDM system and is referred to as the "end user".

The end-user, at a terminal or through a batch system, executes a database command. Then the OEM-written software translates the command to a compact form understood by the IDM. This compact form is the "IDM-internal form". The translated command is subsequently sent to the IDM to be executed; the results are returned, and the OEM-written software formats the results to be displayed to the user. This interaction is shown in the following figure.

Figure 3.1



The figure 3.1 shows the interface in a large mainframe hose. The host operating system would also have to be changed to manage the communication with the IDM. In any system that front-ends the IDM, the following functions must be performed:

Front-end Functions

- Communicate with the end-users.
- Translate user commands to IDM-internal form.

- Send commands to the IDM.
- Receive results from the IDM.
- Format the results and display to the user.

The extent to which the OEM goes beyond the above depends on the specific application. There are many different types of user interfaces possible. We will show three examples, ranging from a minimal system to a more complicated one.

Subroutine Calls

The first example is a hypothetical company, the Parts Control Company (Parts), which sells a parts inventory control system to small manufacturers. In order to upgrade their system, Parts is going to an IDM-based system. To protect their software investment, and to keep the customer re-training costs down, Parts has decided to simply connect the IDM to the mini-computer system they now have. This connection is physically made by connecting the IDM via a GPIB bus interface. The IDM can take the place of some of the disk controllers that previously were on the system.

Parts will implement a subroutine library which contains subroutines that call the IDM to perform certain, limited database functions. The current programs are modified to call subroutines in the library to use the IDM to perform all database-related functions. The screen editing, data checking, forms production and report formatting functions that Parts spent many years developing are untouched and remain in the mini-computer. For example, the monthly report of the outstanding orders used to require reading a large file, where the comparison was made on each element in the file to determine if the order was over a month late. The previous read loop was this:

```
10 read (filel) ordernum, date, vendor_name,
   part_num
   if (end of file) go to 12
   if (date. < T. late)
      print ordernum, date, vendor_name,
      part_num
   go to 10
12 continue
```

Using the IDM, the loop is replaced with subroutine calls:

IDM Example 1: Subroutine Calls

```
      call write_dm(db_num, "lateout", late)

10 call read_idm(results, max_char)
   if (end of data) go to 12
   call print_data("outstand", results)
   go to 10
```

The subroutines "write_idm" and "read_idm" are the IDM interface routines. "write_idm" calls the IDM with the information that the stored command "lateout" is being executed on database db_num, with the date to be compared against stored in "late". Any record (tuple) in the relation referred to in the stored query "lateout" for which the date is less than "late" is passed back to the minicomputer. The command "read_idm" reads the records (to a maximum of max_char characters) that are passed to the minicomputer, and stores them in "results". Then "print_data" is called, which, using the formatting information provided with the returned data prints the results.

Note that the work that the IDM performs for this query is to read the records from the disk system and send to the front-end only those records for which the date is less than "late". The IDM is faster at this function than the mainframe because it uses its own special-purpose controller and database accelerator to read the records and perform the comparison.

The subroutines "read_idm" and "write_idm" are operating system routines. The interface from the operating system to the IDM is designed to be logically like writing to and receiving results from a disk. The OEM (Parts in our hypothetical example) will have the following work to do to implement an application similar to the one just described:

- modify operating system to interface with the IDM
- modify programs to call the IDM for data management
- set up the database and define the stored commands

To set up the database and define the stored commands requires the same sort of interaction as described above. Subroutines are called that send specific information to the IDM: in the case of a "define command" a binary representation of the command is sent to the IDM. In the case of a database load, a "load" command is sent to the IDM, and so on. The Parts Control Company, because it uses only a few well defined database interactions, does not need a general purpose query language, and can use this subroutine-only application of the IDM. That is not the case with the following example.

General Purpose Query Language

In this hypothetical example, the Accounting Services Co. (Account) has decided to offer an accounting package that includes a management information package. This will be an intelligent terminal based system, with each OEM-programmable terminal directly communicating with the IDM over asynchronous lines. There is no host system involved.

Account's customers have expressed the need to analyze the data on income, taxes, accounts payable, and specific parts of their operations that might be losing (or making) money. Account cannot write a well-defined set of subroutines to execute any possible user query since the queries are not known ahead of time. Therefore they decide to implement a general purpose query language.

The Intelligent Database Language (IDL) is a general-purpose query language. It is similar to QUEL which was developed at the University of California, Berkeley. IDL translates easily into IDM-internal form. Let us say that Account decided to use IDL as its general purpose query language. Then the work that must be done in the intelligent terminal is shown in Figure 3.2:

SIMPLE INTERFACE

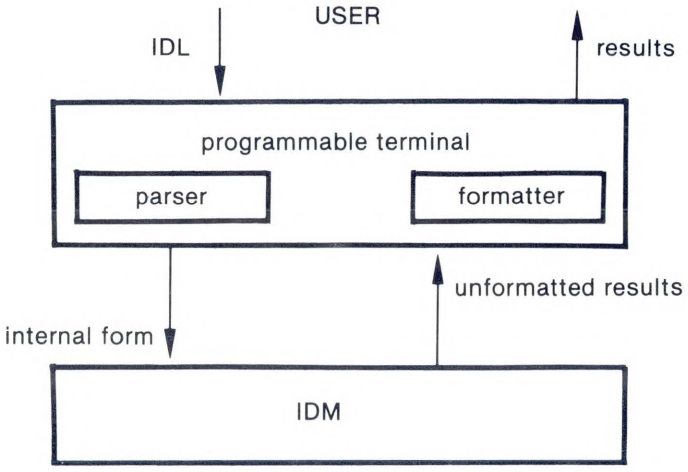


Figure 3.2

The intelligent terminal translates the user command to the IDM-internal form by using a parser. The full definition for the parsing of IDL commands is furnished by Britton-Lee. The parsed command is sent to the IDM in exactly the same way that the command was sent in the Parts example. Since the interface is asynchronous the terminal must accept the results as they come from the IDM rather than loading DMA registers.

An example of the use of such a system is the following. Let us say that a customer of Account's types in the query:

IDM Example 2: Intelligent Database Language

```
retrieve (total = sum (sales.amt
    where sales.month = "July"))
```

This is the IDL query for "get the July sales total". The intelligent terminal parses the query, putting it into the standard form. Then it sends it to the IDM. The result, when it is sent back from the IDM, is in binary form. This is formatted by the terminal and displayed to the user:

$10,159.46

Note that all of the work to perform this query is done in the IDM. Those records for the month of July are found, and the summation performed within the IDM. Only the total is sent to the front-end system.

The programs that any OEM must write to implement an intelligent-terminal based general purpose query system are:

- a parser for the query language
  This is facilitated by the use of IDL and associated parse tables.
- routines to transfer data to and from the IDM
- routines to format data received from the IDM

Embedded Query Language

The third example is the Combined Service Bureau (CSB) which sells software that runs on large mainframes. The IDM is CSB's chance to enter the data management market; since the IDM provides data management without impacting the host system, CSB can sell its new data management system to its current users without having to also sell them new mainframes. CSB wants the flexibility of the general purpose query language, but needs to be able to issue database commands from application programs as well as directly from terminals. CSB has therefore decided to use an embedded query language.

An embedded query language is a database command language that is written as a part of a program in another language. The following is a program fragment which is written in Fortran with IDL embedded:

IDM Example 3: Embedded Query Language

```
C***  get transaction code***
200   print 10
 10   format ("enter transaction code")

      read 15, code
 15   format (15)

      if (code .NE. 10) go to 100
C***  10 - accumulate interest***
      print 20
 20   format ("enter posting date")
      read 25 post_date
 25   format (16)
M     replace receive (due = receive.due*
      receive.rate)
M        where receive.date_due > post_date
      go to 200
```

The database commands are denoted by an "M" in the first column. This program fragment passes the database command "replace" to the IDM when transaction 10 is entered. This will update all acounts which are past due.

To implement this system, CSB must write a pre-processor program. The pre-processor reads a Fortran program, finds all the IDL statements, parses them, and sends them to the IDM as stored commands. It then replaces the lines with calls to the run-time subroutines that interface to the IDM. The program now contains only Fortran statements and can be compiled. At run-time the subroutine calls result in the execution of the IDM commands, exactly as in the Parts example.

The work that the OEM must do in order to implement an embedded general purpose query language in a large mainframe is:

- write a pre-processor
- write a parser
- modify operating system to interface with the IDM

The following section assumes that there are one or more such interfaces to the IDM, and shows the full functionality of the IDM itself.

4. IDM COMMAND SET

Introduction

The full set of IDM commands is given in this section. Examples are given for each of the commands. These examples are in IDL (Intelligent Database Language). IDL translates easily into the IDM-internal form that is actually sent to the IDM. If an OEM wishes to implement IDL on a front-end system, Britton-Lee will furnish the complete language definition. IDL is used for examples in this section for clarity; it is important to note, however, that the interface to the IDM may be via subroutine calls and involve no IDL commands at all. It is also possible that the OEM will want to develop a different general-purpose query language.

The rest of this section is divided into two parts. The first part is a general overview of IDM database management. It contains a discussion of features that are common to several commands. The second part contains the command set.

IDM Database Management

The IDM database management system is relational. Figure 4.1 shows a sample database, the "personnel" database, with two relations.

The employee relation has at least 7 tuples; each tuple has 5 attributes. Three of the tuples in the relation "departments" are shown. The relation "departments" has three attributes.

The relations are set up (created) by the end-user. The data types for each of the attributes are specified when the relations are created. The data types supported by the IDM area are:

	IDM Data Types
BCD	binary coded decimal 1 - 31 decimal digits
int1, int2, int4	1, 2 and 4 byte integer
f4, f8	4 and 8 byte floating point (limited support)
char	1 to 255 ascii characters
bin	binary string

When a relation is created the maximum length of the BCD and character attributes is specified. The IDM will automatically compress the data (deleting trailing blanks in character attributes, leading zeros in BCD attributes) to save storage space.

Floating point numbers are only stored or retrieved. No floating point arithmetic is included in the IDM, except floating point comparison.

The commands sent to the IDM are at a high level. For instance, to obtain the address of the employee "Smith" the *retrieve* command is submitted by the end-user:

```
retrieve (employee.address)
      where employee.name = "Smith"
```

This command is translated to IDM-internal form by the front-end system, then sent to the IDM. The IDM performs the command, and returns the answer:

```
404 4th St.
```

to the user.

DATABASE: PERSONNEL

EMPLOYEES

NAME	NUMBER	ADDRESS	SALARY	DEPT
JONES	2501	210 12th St.	50000	25
SMITH	8912	404 4th St.	12000	12
VITALE	1573	809 24th St.	7000	7
BROWN	8122	802 Ave. J	18000	17
KELLEY	1503	25 Ave. N	24000	25
JONES	1201	820 12th St.	15000	25
BLACK	1402	400 Ave. N	48000	12

TUPLE, OR ROW

DEPARTMENTS

NAME	NUMBER	MANAGER
SHOE	25	2715
HAT	12	8912
BOOK	7	1531
AUTO	17	2510

RELATION OR TABLE

COLUMN, OR DOMAIN

Figure 4.1

Tuples are accessed by values not by position. Therefore the structure of a relation can change. Attributes can be added, the relation can be re-organized, with minimal impact on end-user programs. The values are specified in the "qualification". The qualification in the above query is "where employee.name = Smith". Qualifications are always specified with the "where clause", and can include multiple relations and functions.

Commands can access data in several relations at once. The command to make a new relation from data in existing relations is *retrieve into:*

```
retrieve into sdept (employee = employees.name,
            number = employees.number)
        where enployees.dept = departments.number
        and departments.name = "shoe"
```

The above command creates a new relation, the "sdept" relation, which contains data about only those employees who work in the shoe department. The "retrieve into" command returns no data to the front-end except the count of the number of tuples in the created relation. All the work is done in the IDM, except translating the query.

In the above commands, the list of attributes between the parentheses is called the "target list". The target list can be any attribute name, aggregate or arithmetic function. For example, the command to change data in the database is *replace.*

```
replace employees (salary = 2*salary)
        where employees.name = "Jones"
```

The above command will give a 100% salary increase to both employees named "Jones". The target list contains the arithmetic function "*", which is multiply.

Arithmetic functions in the IDM

+	*addition*
–	*subtraction*
/	*division*
*	*multiplication*
mod	*modulo: gives remainder*
abs	*absolute value*

The arithmetic functions can be applied to any BCD, int1, int2 or int4 attributes.

An aggregate function is one that can take on several values. For example, the command:

```
retrieve (departments.name,
    total = sum(employees.salary by
            departments.name
            where employees.dept =
            departments.number))
```

will produce the following results:

name	**total**
shoe	44000
hat	60000
book	7000
auto	18000

The following aggregate functions are performed within the IDM:

IDM Aggregate Functions

count
count_unique *counts only the unique values*
average
average_unique
maximum
minimum
sum
sum_unique
any *shows existence of a value*

There are two special functions: The "time" function supplies the time of day in a 4-byte integer. The "date" function provides the date in a 4-byte integer.

Data may be converted from one type to another in both the target and qualification lists. The following example uses the command to remove a tuple, *delete.*

```
delete employees where employees.salary =
        bcd(employees.year)
```

The above command removed from the employees relation all employees whose salary is the year they were born

Type conversion is supported from binary coded decimal (bcd), binary (bin) ascii characters (char), 1, 2, or 4 byte integers to each of the other types.

COMMAND LIST

Setting up a database

create database

The command to set up a database is

```
create database DBNAME
with size = NUM on DISK1,
     size = NUM2 on DISK2
```

where DBNAME is a database name of the user's choosing, and NUM is the size of the database on the disk named DISK1, NUM2 is the size on DISK2. The IDM then sets up a database with the name DBNAME. If the disks are not specified (the "with clause" is not in the command) the IDM will use the first available space.

The user can control who has access to this database through the protection mechanisms provided. The database is initially empty except

for system relations that the IDM maintains for each database. These system relations provide a data dictionary and aid in managing the data.

Entering the database

open database

The database, once created, must be opened by each user who will be issuing commands to that database. The IDL command to open the database is

```
open DBNAME
```

The result of an "open" is that a process is set up in the IDM for the user who executed the open. A unique number is returned to the user (the "dbid") which then must be supplied with each subsequent command to this open database.

Putting data into relations

Getting data into a relation is a two-part process.

create relation

First the relation is defined:

```
create employee(name = c10, number = i4,
   deptno = 12, address = c20, salary = i4)
   with size = NUM on DISK1, size = NUM1
   on DISK2,
   quota = NUM2, logging
```

The employee relation is defined, with the attributes of name (maximum of 10 characters, variable length field), number and salary (4-byte integer fields), department number (a 2-byte integer field), and address (maximum of 20 characters, variable length field).

It is allocated an initial size of NUM on logical device DISK1 and NUM1 on DISK2. It will not be allowed to grow beyond NUM2. There are default values for the size specification; quota is assumed to be the size of the database if the quota is not specified. The quota can, at any time, be increased by the person who created the relation.

The audit facility consists of logging all changes to any relation. If it is needed for a given relation, the relation is created with the specification "logging".

The entire "with clause" is optional. Default values are assigned.

append

Data is put into a relation with the "append" command.

```
append to employees(name = "Jones"
   deptno = 27,number = 1234)
```

The tuple is stored in the relation "employee"; the name and number fields have the values specified. Since the salary and address fields are not specified, the IDM assigns to them the default values: zero for salary, since it is a numeric field; blank for address since it is a character field.

There is also a bulk loading capability for filling a relation with large amounts of data already stored by the host computer. It is discussed in the section "load".

retrieve into

Data can be put into a new relation as a result of a "retrieve into":

```
retrieve into namsal (employee.name,
employee.salary)
   where employee.dept =
   department.number
   and department.name = "shoe"
```

In the above command, the relation namsal is created by the IDM and contains only the names and salaries of those people who work in the shoe department.

Accessing data in the database

range

For ease in abbreviating the relation names, and for multi-relation commands, a "range" statement is required to associate a variable name with a relation name:

```
range of e is employee
```

In this example, "e" is a variable that stands for the "employee" relation.

retrieve

Data is displayed through the command "retrieve".
The command:

```
range of e is employee
retrieve (e.name, e.salary)
        where e.salary > 1000
   order by e.name
```

will display the employee names and salaries of all employees who have a salary greater than 1000. The "order by" clause causes the results to be returned sorted by name.

The items between the parentheses comprise the "target list", which are the items to be displayed. The qualification (the "where clause") defines which of the items in the

target list are affected by the command. If, for instance, the user wished to see the names and salaries of all the employees, the command would be:

```
retrieve (e.name, e.salary)
```

Both the qualifications and the target list can refer to several relations. The below query lists all those who work in the manufacturing department:

```
range of e is employee
range of d is department
retrieve (e.name) where e.deptno = d.deptno
   and d.deptname = "manufacturing"
```

The processing of multi-relation queries is done completely in the IDM. Advanced techniques are used to efficiently execute multi-relation queries.

create index

Data in relations can be accessed quickly if pointers to the data exist; an index is a pointer to the data. There are two kind of indices: clustering and non-clustering. A clustering index is one that points to an attribute or group of attributes that the relation is ordered by. The command:

```
create clustering index on
   employees (number)
```

causes the IDM to sort the relation "employees" on "number", store it in that order, and to make a B-tree index that contains page identifiers for values of the "number" attributes of the employees relation. When the employees relation is accessed with the number specified in the qualification, the index is searched first and the access is made directly to the data needed.

There can be only one clustering index per relation. When the relation will be accessed by other attributes, non-clustering indices can be created.

```
create non clustering index on
   employee (name)
```

The above command results in the creation of a directory that has one entry for each tuple in the employees relation. Each entry contains the value of "name" for the employee, and the pointer to the page that contains the tuple for that employee.

View Support

create view

A view is a relation which doesn't really exist. It is a "virtual relation" that is composed of parts of one or more relations. One use of views is to create a virtual relation that is missing sensitive data present in the real relation.

```
create view emp (employees.name,
      employees.address,
      employees.dept, employees.number)
```

The view "emp" created above will contain the information in the employees relation without the sensitive "salary" attribute. The view "emp" can be retrieved from and protected in the same way as a real relation.

The advantage of using a view is that all changes to the real relation "employees" are automatically reflected in the view "emp". This is a particularly useful feature when there is data that comes from several relations and is needed in the form of one relation:

```
create view ndept (employee = employees.name,
      dept = departments.name)
   where employees.dept = departments.number
```

Changing Data

replace

The values in the database can be changed through the "replace" command. An example is:

```
Replace e (salary = 5000)
where e.name = "Jones"
   and e.deptno = d.number
   and d.name = "shoe"
```

This changes the salary to 5000 for all people named Jones who work in the Shoe department.

delete

The "delete" command can be used to delete tuples from a relation.

```
delete e where e.name = "Jones"
      and e.deptno = d.number and d.name =
"Shoe."
```

This command removes all employees named Jones from the employee relation if they are also employed in the Shoe department.

destroy

To remove an entire relation, the command is:

```
destroy RELNAME
```

This command completely removes the relation RELNAME from the database.

To remove an index, the command is:

```
destroy clustering index on employees
```

or

```
destroy non-clustering index
on employees (number)
```

destroy database

To remove an entire database, the command is:

```
destroy database dbname
```

The IDM will check user access privileges before allowing any command to take place. Only the user who created the database can destroy it.

Stored Commands

define

An efficient way to execute database commands is to store them in the IDM; the IDM does all the necessary pre-processing when the command is stored. Then when the command is run it is specified by name or number.

To store a command in the IDM:

```
range of e is employee
define getname retrieve e.name
where e.sal > $min
```

Parameters are specified with a $.

execute

To execute the command, the statement is:

```
execute getname with min = 1000
```

The stored query feature is particularly important for front-end programs. It allows the internal form of the query to be stored in the IDM. The front-end program sends the query name and appropriate parameters. This reduces the amount of information that must be transmitted to the IDM to run a query, and it reduces the size of the front-end program. The end result is more efficient, smaller programs.

Protection

deny

Users are denied access to a relation based upon the user identifer provided by the front-end system. The command to deny access to a relation is:

```
deny write of employee(salary) to USER
```

The user or group of users identified by USER is denied permission to change the salary field in the relation "employee".

permit

Conversely, to permit access to a relation to a user is:

```
permit read of department to USER
```

In this example, the entire relation "department" can be read by the user or group identified by USER. For both "deny" and "permit" an entire relation or parts of a relation can be specified.

The command:

```
deny execute of getname to USER
```

similarly protects the stored query getname from execution by USER

Transaction Management

begin transaction

Transactions are commands which, as a group, must either be completely executed, or not executed at all. An example is a money exchange:

```
begin transaction

replace customer(bal = customer.bal
    - new.sales)
    where new.cust_number =
    customer.number

replace sales(total = sales.total +
    new.sales)
    where new.product = sales.product

end transaction
```

These two commands - decrementing the customer's balance and incrementing the sales totals - must take place as a pair, or the books will not balance. The IDM data management system processes the commands between "begin" and "end" transaction as a group - if the command is aborted for any reason before the end transaction is reached, the data is restored to its pre-transaction state.

abort transaction

The abort transaction command:

```
abort transaction
```

stops the end-user's current transaction, and restores the database to its state prior to the beginning of transaction.

Auditing Support

create audit

The transaction log is written to whenever a relation is changed, unless logging was declared "off" when the relation was created. The transaction log can be periodically dumped to other files for back-up, as discussed in "dump" below. In order to create a relation from the log file that auditors can use the create audit command is used:

```
create audit ANAME from RELNAME
where time > T1 and time < T2
```

This command creates an audit relation ANAME from the dumped transaction log name RELNAME. The audit relation is processed from the time specified by T1 until the time specified by T2.

Crash Protection

In order to protect the integrity of databases in case of equipment malfunction or other data-damaging catastrophes there are three types of dumps available: physical, logical and incremental.

dump disk
load disk

A physical disk is written to another device with the dump disk command. In case of catastrophe, the disk can be re-written with the load disk command.

```
dump disk DISK1 to DISK2
```

DISK1 and DISK2 are the logical names of disks on the IDM; if the disk is to be dumped to a device connected to the front-end system,

```
dump disk DISK1
```

will write the contents of the disk to the front-end system.

Similarly,

```
load disk DISK1 from DISK2
     and
load disk DISK1
```

re-write the contents of the disk DISK1. The disk that is loaded must correspond exactly to the disk that was dumped - it must have the same number of blocks, cylinders, etc.

dump database
load database

A logical dump of a database means writing the data out a relation at a time, keeping the logical structure of the database. When a logical dump is re-loaded, it need not be loaded to a similar device. In fact, the load database command can be used to bulk load data into the IDM from external sources.

```
dump database DNAME to DISK1
```

The above dump will put the entire database on the IDM disk DISK1. Then to restore the existing database, the command is

```
load database DNAME from DISK1
```

To dump the database to the front-end system the command is

```
dump database DNAME
```

To dump or load a relation at a time, the command is:

```
dump database DBNAME (RELNAME1),
   RELNAME2,...)
     or
load database DBNAME (RELNAME1,
   RELNAME2,...)
```

A database or relation can also be dumped and loaded to IDM files.

dump transactions

The transaction log is updated whenever an audited relation is changed. A strategy for handling the possibility of catastrophic loss of data is to

1) dump the entire database occasionally
2) dump the transaction log frequently
3) in case of loss of data, load the last database dump and apply the transaction log.

The success of the above strategy depends on the fact that logging is on for relations important to the application.

To dump the transaction log:

```
dump transaction into RELNAME
```

rollforward

To apply the transaction log:

```
rollforward RELNAME to DATE, TIME
```

If DATE is not specified, the databases are restored to the last date on the RELNAME transaction log.

File System

The IDM random access file system is provided particularly to support users who need to use the IDM disks for uses other than database management.

create file

The command:

create file FILENAME with alloc = SIZE1 on DISK1, quota = SIZE2

creates FILENAME of size SIZE1 on DISK1. It will not be allowed to grow beyond SIZE2. The file size is limited to 2**31 bytes. Only the creator of a file many destroy it. Protection (read and write capabilities) for files are expressed exactly as for relations.

destroy file

The command:

destroy FILENAME

will remove the file from the system.

open file

To access the data in a file, or to change it, the file must first be opened. A file number is returned to the used as the result of a successful open.

filenum = open file FILENAME

The file is then referred to by filenumber.

read
write

To read or write the file, the commands are:

read (filenum, count, offset)
write (filenum, count, offset)

The count is given in bytes; the offset is the byte number within the file where reading or writing is to begin. Sequential access is therefore implemented by the user program keeping track of the position within the file for the next access.

If the file size is less than 580K bytes, a random access is 2 disk references. If it is less than 168M bytes, it is three references. There are a maximum of 4 disk references to randomly access a 2G byte file. Since the IDM utilizes a buffer cache, after the first access the actual disk references may be to the cache, not to the disk.

close file

When the user finishes with a file, it is closed:

close file filenum

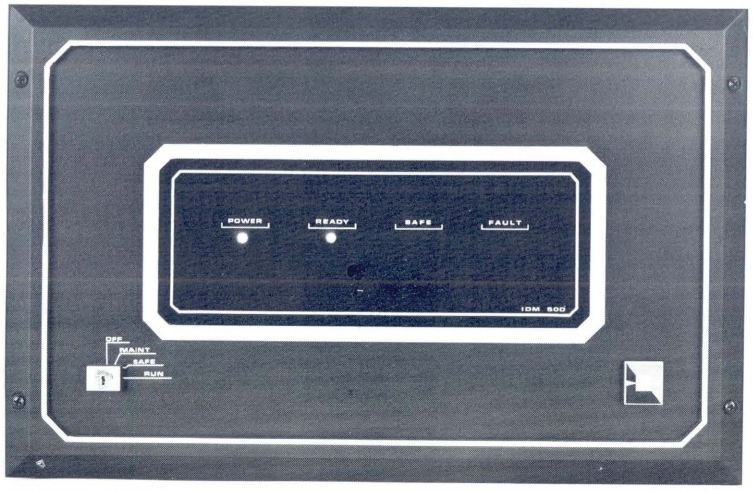
5. HARDWARE RELIABILITY AND MAINTENANCE

The IDM tolerates a variety of common hardware error conditions including uncorrectable memory errors, host-IDM communication errors, disk read/write errors, intermittent IDM bus errors, and power failure. Each board except the memory storage boards contains built-in diagnostics and self-test capability. The memory storage boards are tested from the Database Processor. Every IDM has two special RS232 ports labeled "console" and "maintenance" which can be used to monitor system performance and for remote diagnostics.

The IDM classifies hardware errors into three categories: fully recoverable means that no user is affected by the problem; for example, a correctable main memory error or a disk read error on a disk block which is subsequently read correctly. Recoverable means that one user may have to be aborted or restarted. This would happen, for example, if an uncorrectable memory error occured in the user's workspace. Unrecoverable means the IDM cannot reliably determine its current state. In such a case, the IDM restarts itself, cancelling all current user operations. The IDM then comes up in recovery mode and diagnoses the hardware failure. If possible it will then come up and bring the database to a consistent state. Finally it will become available for use. As long as the error causing the crash did not damage the disk drives, the consistency of the database is guaranteed.

Front Panel Controls

The IDM is controlled using the front panel key switch and a terminal plugged into the "console port" on the back panel.



The IDM function key may be positioned in the following locations:

1) Off

DC power is turned off. All IDM boards are powered off; however, AC power is still supplied to the IDM power supply

2) Safe

The database is brought to a consistent state. Current transactions are allowed to complete. Any new transactions are rejected. The "safe" light will come on once the last transaction has completed and all changes have been reflected to the database.

3) Run

The IDM is in its normal operation. The "ready" light will be on.

4) Maintenance

The IDM will accept commands only from the console port. The "ready" light will blink. The IDM will ignore any communications from the host computers.

Under normal usage, the IDM can be taken off-line by putting it in safe mode. This allows current transactions to finish and notifies the host processors that the IDM is going off line. If necessary, the IDM can be brought into maintenance mode which will cause all currently running transactions to be aborted.

When the IDM is powered up and put in safe mode, it will check the consistency of the databases, will complete any committed transactions and will backout any uncommitted transactions. When this is done the "safe" light will come on. If the IDM is powered up and immediately put in "run" mode, it will check the consistency of the databases and come on line once the databases are ready.

It should be noted that changing the IDM's running mode will not compromise database consistency. Its main usage is for gracefully bringing the system up and down and for diagnosing system problems.

Diagnostics

The IDM reports errors by maintaining error logs in the "system" database, through use of the FAULT light on the front panel, and through a written log on the console (if one is connected). If an unrecoverable hardware error occurs, the front panel FAULT light will go on. If the IDM can diagnose the problem and work around it, the IDM will turn off the FAULT light, print a diagnostic message, and come back up.

If the FAULT light remains on, there is a system error which prevents the database processor from running. This would happen if the system bus were inoperable or the database processor was not functioning.

In most cases the IDM can self-diagnose problems down to the board level and report which board needs replacing. The system is also designed to run diagnostics with various boards missing. The Database Processor is capable of running by itself in the IDM chassis.

Fault Tolerance

The specific way the IDM handles each type of hardware fault will now be discussed.

Main Memory Errors. Main memory can detect and correct single bit errors and can detect double bit errors. Both conditions are logged by the IDM in the "system" database. At any time, a log of hardware problems can be examined. Correctable, single bit failures are expected from time to time and are not logged unless they exceed a predetermined frequency. If that happens, they are analyzed to determine if a memory device had completely failed. If so the failure is reported and the IDM avoids using that section of memory. Periodic replacement of failed memory devices is necessary to guarantee that uncorrectable errors will not occur. Uncorrectable errors occur when two bits are in error. In such a case, a variety of attempts are made to keep the IDM running. In many cases, the bad memory can be mapped out of the system and a complete recovery is possible. Whenever possible a second attempt is made to read memory in case the double bit failure was intermittent.

When the IDM is powered up, all of memory is checked and bad sectors of memory are reported. If a sufficient amount of good memory remains (at least 256K bytes), the IDM will come up; otherwise, the memory must be repaired before the IDM can function.

IDM bus errors. The IDM internal bus operates at extremely high speed. To insure integrity of bus operations, parity is used throughout the bus. If an error is detected, the bus operation will be retried if at all possible. Retries are counted and an abnormally high incidence will generate a diagnostic message. The use of bus parity and bus operation retry provides very stable and reliable interboard communications.

Disk Errors. The IDM disk controllers have built-in burst error correction and automatically retry and correct disk pages. This virtually eliminates the possiblilty of a disk read error causing the user to see incorrect data. When a disk page has been correctly read, the disk controller reports if an error procedure had to be used to read the disk page. If it is warranted, an attempt is immediately made to rewrite the disk page and if necessary reassign the page to a new location before the data becomes unreadable. This elaborate fault tolerance is designed to detect gradual deterioration of the disk media and move valuable data before it is lost.

When a disk is first formatted by the IDM, a surface analysis is automatically performed and any defective areas are reported and tagged to prevent the IDM from us-

ing them. The IDM can avoid using individual sectors, or whole tracks. The IDM maintains 8 bits of information about every sector on every disk. This information is available in the "system" database.

Host-IDM communications. Communications between a host and the IDM is done over either a serial line or IEEE 488 bus. Both types of lines use a block mode protocol with error detection. The error detection is programmable and can be either none, a checksum, or CRC characters. Every message passed between a host and the IDM requires a positive acknowledgement. A negative acknowledgement causes a retransmission of the message. If an individual line fails or a complete I/O Channel fails, the IDM will abort any commands associated with the corresponding hosts and continue to run.

Periodically the channel will poll each host to make sure the host is still operational. If a response is not received within a specified period of time, the host is presumed down. When the host comes back up, it can reestablish communications with the channel.

Power Fail. The IDM will shut down upon internally detecting AC power low. When power is restored, the IDM will check the consistency of the databases, backout or complete any transactions in progress at the time of the power fail, and then come up. Just before writing to a disk the disk controller checks the status of AC power. If it is normal, it commits the write. If the power fails during the write, the IDM power supply will maintain DC power through the completion of the write. A similar power sensing capability is required from the disk drive itself.

6. CONFIGURATION AND SELECTION GUIDE

This section discusses the physical limits of the IDM. The IDM can be configured to expand the amount of main memory, expand the number of host communication channels, expand the number of disks, or add a Database Accelerator.

Basic IDM System Components

The basic IDM comes in a rack mountable chassis with self-contained power supply and room for a maximum of 16 IDM boards. The minimum IDM requires:

- Database processor Board
- Memory Control & Timing board
- One 256K byte Memory Storage board
- One Disk Controller board
- One I/O Processor

for a total of 5 boards. This leaves ample room for up to 11 additional boards. The various configurations will be described as follows:

The IDM Chassis Bottom Plane

The IDM chassis has room for up to 16 boards. The IDM bottom plane does not support an "expansion chassis", therefore, no IDM can have more than 16 boards. The bottom plane is designed electrically so that any board can be plugged into any slot. IDM boards have no user settable straps or options. The IDM internally determines the system configuration when the unit is powered up. Parameters to specific boards are entered through software control and are stored in the database. Certain boards have external cables leading from the boards to the back panel of the IDM. For ease of installation it is recommended that IDM boards be inserted in the following order (starting from the power supply):

1. Database Processor
2. Disk Controller(s)
3. Host I/O Processors(s)
4. Memory Timing & Control, Memory Storage, and Database Accelerator

Parallel I/O Channel Processors

One I/O parallel processor board controls one IEEE 488 GPIB bus. The standard back panel on the IDM has room for 4 independent GPIB buses. A GPIB bus can connect several hosts to the IDM or each host (up to a maximum of four) can connect to the IDM via a dedicated GPIB parallel channel. In very high transaction rate environments, and in some other special applications, it may be beneficial to use multiple GPIB buses to multiple hosts. The primary advantage is increased parallelism - two hosts can communicate with the IDM simultaneously. The maximum data rate of the IDM GPIB is 250K bytes/second.

The parallel and serial I/O Channel Processors do all host dependent data type conversions (swapping bytes on integers, one's to two's complement conversions, etc.) The channel must be told what type of host it is talking to and also the unique host ID. This information can be supplied automatically from the host, or it can be kept in the system database in the IDM.

Serial I/O Processors

One serial I/O Processor controls eight independent RS232 compatible lines. The serial channels are programmable for baud rates from 110 baud through 19,200 baud. Modem control is standard and can be disabled if not needed.

A Serial I/O Channel processor is connected to the back panel of the IDM through one fifty pin connector. The connector services 8 serial lines. The standard IDM back panel has room for up to 8 connectors. This provides 64 serial communication lines controlled by eight serial I/O processors. Back panels can be tailored for an appropriate mix of serial and parallel ports limited only by the physical dimensions. Britton-Lee distribution panels (DTP) are available which connect the fifty pin connector from a serial I/O channel to eight EIA standard connectors. The distribution panel is completely passive and can be manufactured by the OEM to meet specific requirements.

Memory

Additional Memory Storage boards can be added up to the maximum number of slots available on the bottom plane. The IDM requires a minimum of 256K bytes in order to run at all. Additional main memory improves performance in two ways. First it reduces the amount of disk activity by keeping a cache of disk pages, and second it can increase the number of users which can simultaneously have commands in progress. The IDM dynamically balances the amount of memory available for various resources. The performance improvement achieved by increasing the amount of main memory greatly depends on the specific application. The basic IDM code requires 224K bytes. The remaining memory is divided into 2K byte pages which can be used either for caching disk activity or storing user commands in execution. Storage boards are available in 256K byte sizes. Only one Memory Timing and Control is required.

Disk Controller

Each IDM disk controller is compatible with the CDC 9760 SMD interface. The disk controller has one "A" cable and four "B" cables on the back panel of the IDM allowing four disk drives to be directly connected to the disk controller. The IDM allows up to four disk controllers per system providing a maximum of 16 disks. The total capacity of all disk drives cannot exceed 32 gigabyte (2^{35} bytes). Certain disk drives which claim to be SMD compatible are in fact, not strictly compatible. Britton-Lee will provide a list of compatible drives. A drive can be connected to a disk controller without any changes to the controller board. All parameters about the disk, such as number of bytes per track, number of heads, number of cylinders, are supplied through software and once given, are kept on the actual disk. The disk may be cabled to any connector of any disk controller. The IDM reads each disk to determine its actual logical device name and does not depend on any specific cabling.

Database Accelerator

The IDM will run with or without the Database Accelerator. When the IDM is powered up, it determines whether the Database Accelerator is present. If not, it automatically emulates the accelerator.

7. PRODUCT SUPPORT

Installation The IDM 500 is a self contained hardware device that can be installed by the OEM user. It is preferable, however, to allow Britton-Lee, Inc. to install the first system with the OEM disk drive being used. The disk drive can be drop shipped to Britton-Lee, Inc., or the Service Department will arrange installation on site. The I/O connection to the host processor can be accomplished by simply plugging into the back panel serial or parallel port. Field service fees are included in the IDM price schedule.

Training Four day training courses will be offered at the factory to cover IDM software integration and use. A contiguous course of one day will cover maintenance and installation of the IDM. Upon the signing of the OEM agreement the buyer has the right to send two people to these courses free of charge. Accommodations are the responsibility of the OEM. Fees for these courses are included in the IDM price schedule.

Application Notes A Britton-Lee application note publication is available on a subscription basis to all users of the IDM. This application note will profile typical applications of users, and document future products and improvements to the IDM 500. Contact the Marketing Department for subscription information.

Depot Maintenance Depot maintenance at the PCB level will be handled at the California factory. Maintenance will be handled on a spares replacement basis. The OEM customer will purchase a complement of spares to serve his user needs. When the OEM has replaced his part with a spare, the problem part should be sent to the Service Department with the service authorization number and a description of the fault. The Service Department will repair or replace that part within one week of receipt and return the part to the OEM. Depot maintenance fees are included in the OEM schedule.

Board Exchange Policy

The OEM, upon agreement may enter into a PCB exchange policy whereby the OEM may return a faulty PCB and receive a one day turnaround at the factory. This process will be billed on a flat per board fee basis. See the IDM price schedule. Contact Field Service for further information on the Board Exchange Policy.

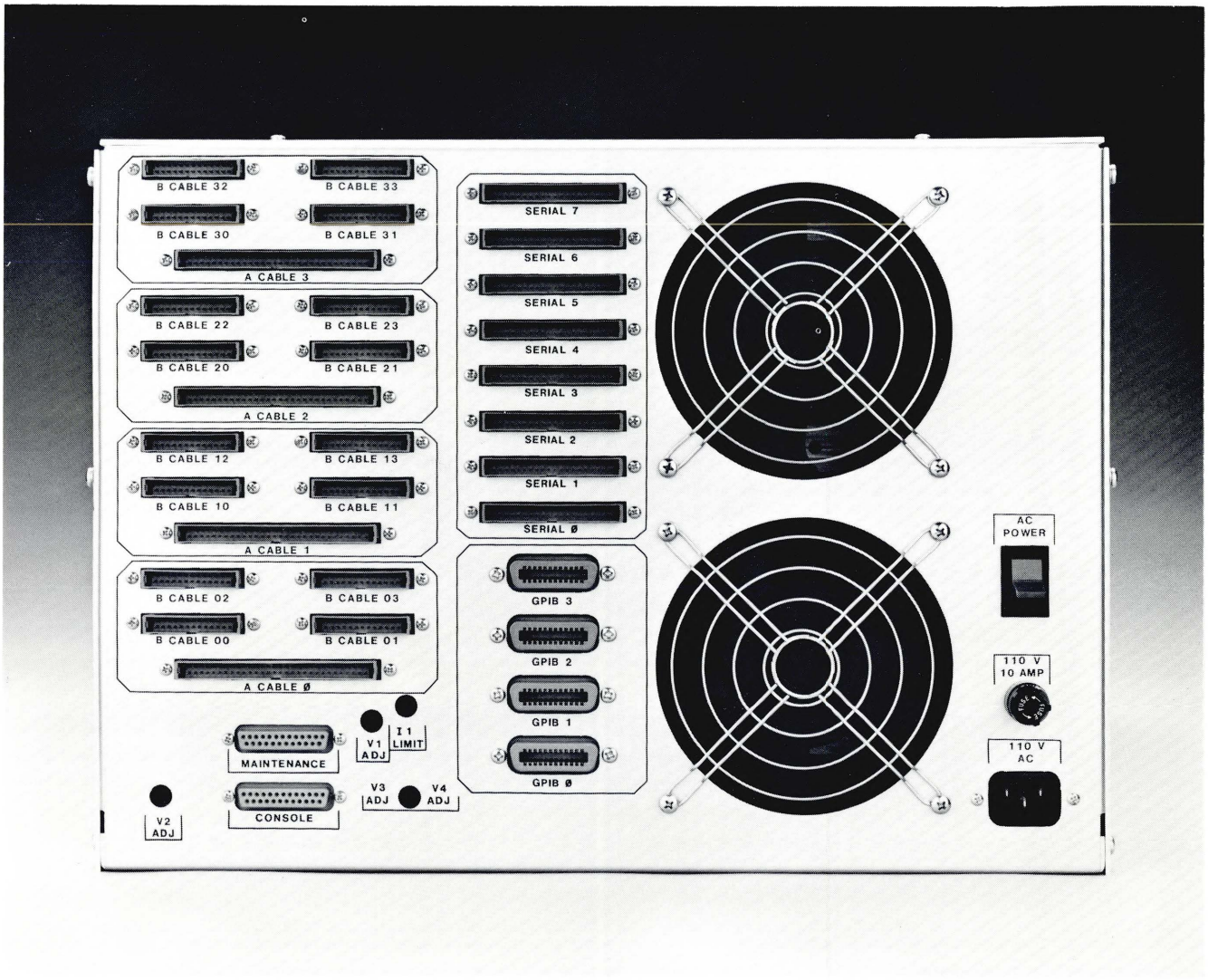
Ordering Procedure

General. A purchase order may be used to order small quantities of the IDM. After credit verification has been accomplished an acknowledgement of the order will be sent confirming price, delivery, terms and conditions. Orders may be placed by following the configuration guide and the IDM price list.

Example

Product Number	*Description*
IDM 500	IDM 500 System Includes: power supply, bottom plane, 1-256K memory, 1-database processor, 1-serial I/O processor, 1-controller chassis with 12 spare slots.

Warranty A limited warranty of 90 days is provided with the purchase of each IDM. The warranty is restricted to repair and/or replacement of any IDM part at the Britton-Lee, Inc., factory in California.



Britton Lee, Inc.

ALBRIGHT WAY
LOS GATOS, CALIFORNIA 95030
(408) 378-7000