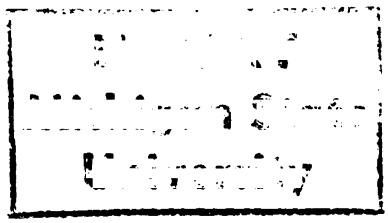


This is to certify that the

dissertation entitled

AUTOMATED ANALYSIS OF
TRIPLE QUADRUPOLE MS/MS DATA

presented by

Hugh Ralph Gregg

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Chemistry

C. G. Enke
Major professor

Date Dec. 17, 1986

AUTOMATED ANALYSIS OF TRIPLE QUADRUPOLE MS/MS DATA

By

Hugh Ralph Gregg

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Chemistry

1986

ABSTRACT

AUTOMATED ANALYSIS OF TRIPLE QUADRUPOLE MS/MS DATA

By

Hugh Ralph Gregg

A data explosion has accompanied the advent of microcomputer controlled instrumentation. Many of these automated instruments are capable of generating more data than the operator is capable, willing, or able to analyze. What the analyst wants is not the data, but the chemical information contained in the data. In this dissertation, several software tools developed for collecting, storing and retrieving data as well as methods for extracting the information contained in mass spectrometry/mass spectrometry (MS/MS) data are presented.

Triple quadrupole mass spectrometry (TQMS) is a multivariate technique with unique data storage and retrieval requirements. Traditional MS or GC/MS data base systems are unable to handle the many dimensions of data that make MS/MS a powerful tool for structure determination and mixture analysis. A multi-dimensional data base was developed to quickly and efficiently store all data and instrumental parameters produced by TQMS. A program is

presented which is able to extract any arbitrary two-dimensional plane of data for display and interpretation.

The final tools presented in this dissertation are aids in the interpretation of MS/MS data. One of the techniques used in analyzing a mass spectrum of an unknown is to identify characteristic neutral losses from the ions present in the spectrum. MS/MS is unique in that if the collision conditions demonstrate first order fragmentation of the parent ion, all the daughter ions formed are direct, single event fragmentation products of the parent. A program was developed which compares neutral losses (the difference between parent and daughter masses) with a table of common losses and presents a list of possible fragments and/or the substructures giving rise to those fragments.

Mass spectra matching routines have traditionally been used to identify unknown spectra. The simplicity and wide variations in intensities in daughter spectra make normal EI matching techniques unsuitable. The matching algorithm presented in this dissertation was designed to group daughter spectra.

Each of the tools presented in this thesis addresses different aspects of the data/information handling problem and demonstrates the need for powerful software tools to complement today's complex instrumentation.

ACKNOWLEDGEMENTS

I'd like to thank everyone that has helped make my stay in East Lansing both rewarding and pleasurable, with an extra thanks to Chris Enke for his guidance and friendship. Tom Atkinson also deserves special recognition, for as busy as he was, he always had time to talk about chemistry, computers or home repair.

Chris Enke's research group provided an ideal atmosphere to both learn and make friends - thanks to all past and present group members. I'd also like to acknowledge the following people and organizations: the Physiology Department, Jack Hoffert, Paul Sorrenson (and Art's bar), the Milton Hilton (and all former tenants), the Winded Spartans, Brenda Spiewak, Mike McPherson, the Beak & Dome Bar & Grill and of course, Milton Webber.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Introduction

In the fall of 1978 a new instrument was just coming online and being tested in Dr. Enke's research laboratory. The triple quadrupole mass spectrometer (TQMS), designed by Yost and Enke (1-3), was completed and proved capable of generating daughter spectra (mass spectra of selected parent ions). The technique of mass spectrometry/mass spectrometry (MS/MS) was not new (4-6), but this new instrument was capable of unit mass resolution in both mass analyzers, and boasted a highly efficient collision chamber. It was a new instrument - a promising technique and the basis for many years of fascinating research.

In tandem quadrupole mass spectrometry (7), ions created in the source are selected by the first mass filter (quadrupole 1) and undergo a fragmenting collision with neutral molecules in the collision cell (quad 2, RF only: not mass filtering). The ionic products of this collision are then mass analyzed (by quad 3) and detected. By this process, a fragmentation spectrum from each ion in the normal mass spectrum can be obtained.

The TQMS instrument can be operated in several ways by using the mass filtering quadrupoles in one of three modes: 1) selecting one mass for transmission, 2) scanning a series of masses, or 3) allowing all masses to pass through the quadrupole (RF only mode). Table 1.1 shows a summary of the operating modes and the resulting scans.

The information contained in MS/MS data can be used in a variety of ways. Daughter spectra are useful in mixture analysis and screening techniques (2,5,8-11) as well as structure determination problems (2,5,12-14). I will present, in this dissertation, tools and techniques necessary to extract the information present in this type of multi-dimensional data. To help the reader to gain an appreciation for how these pieces fit into the overall goal, I will present the structure determination scheme developed in our laboratory. This method, and associated tools, have been developed and refined over the years by several people (15-16).

Structure determination by MS/MS

The basic premise behind our structure elucidation scheme is that if many small, "simple" parts or substructures of a sample are known, its structure can be determined. Each ion in the source results from some

Table 1.1

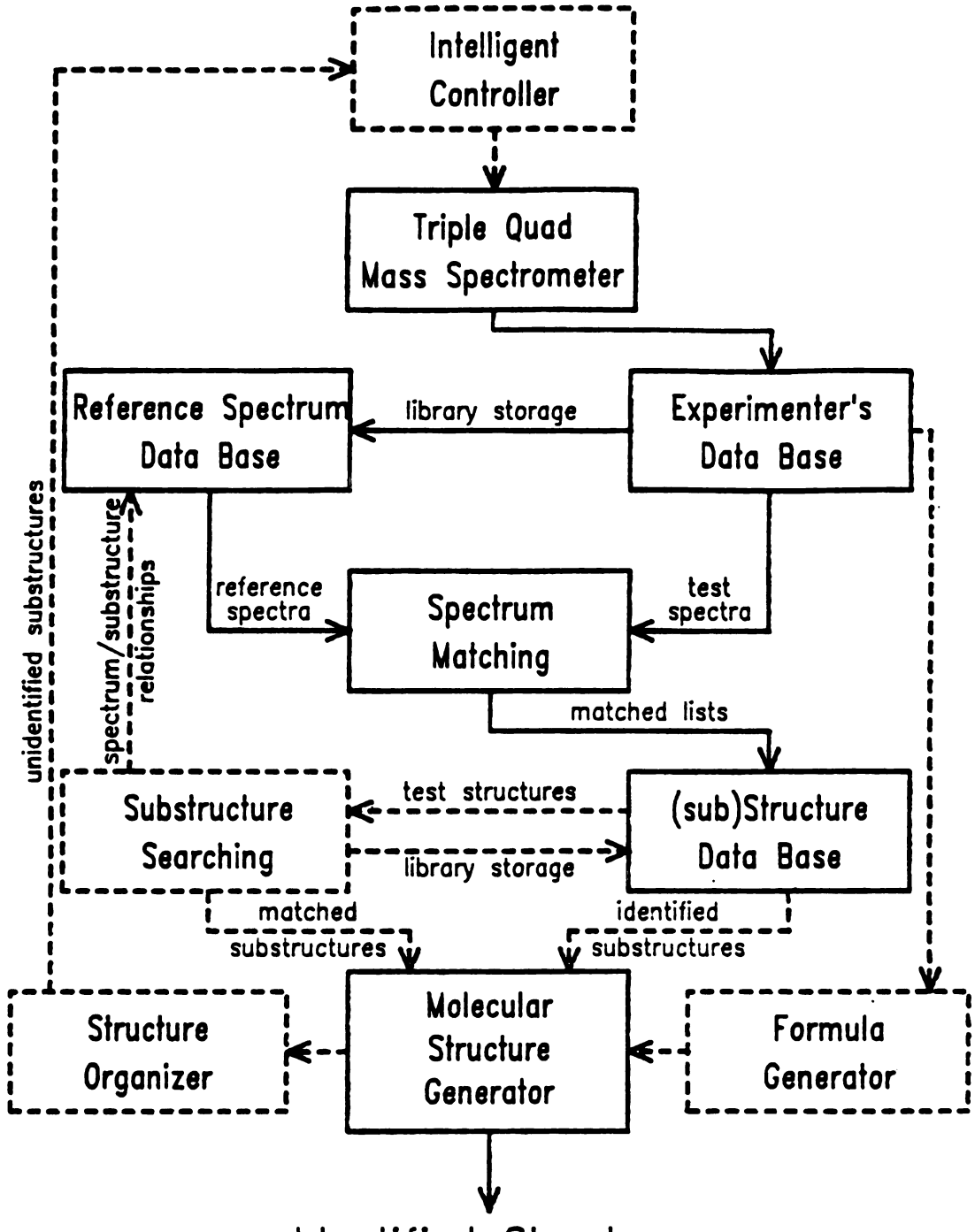
TQMS operational modes

Quad 1	Quad 2	Quad 3	Description
scan	RF only* no gas	RF only*	Normal mass spectrum
RF only*	RF only* no gas	scan	Normal mass spectrum
fixed	RF only* gas on	scan	Daughter scan: spectrum of all daughter ions from a selected parent
scan	RF only* gas on	fixed	Parent scan: spectrum of all parents that fragment to form a specific daughter ion
scan	RF only* gas on	scan at a fixed offset from Quad 1	Neutral loss/gain scan: spectrum of those parents that lose/gain a given mass during a collision
fixed	RF only* gas on	fixed	Single or multiple reaction monitoring

*RF only quadrupoles pass all masses (not mass filtering)

substructural feature of the original molecule, and the fragmentation (daughter) spectra of these ions are often indicitive of these substructures. Other MS/MS information (parent scans, neutral loss/gain scans, neutral spectra, etc.) can also be used to identify structural fragments. The general approach is to make correlations between a known substructure and some subset of the complete MS/MS fragmentation map. With a library of these correlations, an unknown MS/MS fragmentation map can be quickly analyzed for the indicated structural characteristics. In the first implementation of this scheme, we will compare an unknown daughter spectrum against a library of daughter spectra. The closely matching daughter spectra are assumed to be related to structural similarities, specifically common substructures. The structures of the reference compounds will be compared to determine the substructures they have in common. Lists of substructure/spectrum correlations will be made in this way. This process would be impractical without a group of synergistic software tools to help automate each step. Figure 1.1 shows in block form each of the major tools needed for a project of this magnitude.

An unknown sample enters the scheme diagramed in Figure 1.1 as experimental data which is stored in the multi-dimensional data base (see Chapter 3). These unknown



Software tools for structure determination

Figure 1.1

daughter spectra are compared to spectra in the reference data base (17-18) by matching algorithms (19-20, also see Chapter 5). The substructures of the best matched daughter spectra are obtained (currently these substructures are manually entered, but algorithms to extract these substructural features from structures of the parent molecule are being developed). All resulting substructures (and information from other sources) will be used as input to GENOA (21-23) to determine all chemically possible structures that contain the identified substructures. The final step involves analyzing GENOA's output structures for the major features and determining what experiments will further reduce the number of candidate structures.

Thesis outline

This thesis describes my work toward automating the data collection and analysis of a complex, multi-variate instrument. That this work involves three vastly different areas of computer science (real-time processing for control systems, data base management and expert systems for data analysis) underscores the diversity of techniques needed to take full advantage of today's automated instruments. This dissertation consists of five chapters, including this introduction. Each chapter is independent of the others (although Chapter 4 refers back to Chapter 3 for details of

the data base) and includes introductory and concluding remarks.

Control system hardware and software are discussed in Chapter 2. This chapter describes the first and second phases of the control system and its operation. The design considerations and tradeoffs that went into each phase are discussed. Although these phases of the control system have been subsequently superseded, they provided an excellent foundation on which more sophisticated systems were implemented.

Chapter 3 describes a data base for multi-dimensional data. The first version of this data base software was developed jointly with researchers at Lawrence Livermore National Laboratory (24-25), but was modified and extended locally. This data base software is currently running on three TQMS instruments, and has proven to be extremely reliable and invaluable for the storage of MS/MS data.

A special data retrieval program is discussed in Chapter 4. Described is a program that is able to extract any two dimensional plane of data from the multi-dimensional data stored in the data base. This is a convenient and powerful tool for trend analysis and allows the user to look at a matrix of data from several orthogonal axes.

The final chapter describes some of the chemical information available in MS/MS data, and presents several ways of extracting some of this information from the data. The concept of a neutral spectrum is introduced, it's utility is explored, and a simple expert system for the analysis of these spectra is described. Also presented is a grouping algorithm designed to cluster daughter and neutral spectra so that the common structural features of their parent molecules can be obtained. The substructural information gained by using these techniques can be used to help determine the overall structure of unknown compounds.

Conclusions

In this dissertation, the process of automating the analysis of TQMS data is described, from the analog to digital converter that samples the ion intensity, through an expert system which samples the information present in MS/MS data collected. Throughout this work, I have tried to show that comprehensive software tools can aid the chemist in such complex tasks as structure determination. Someday these tools will be integrated with the addition of an intelligent controller as shown in Figure 1.1. This controller will be able to direct the experiments performed by the instrument. The work that others and I have done in Dr. Enke's lab (solid lines and boxes in Figure 1.1) sets an excellent foundation for these higher level systems.

Chapter 2

Instrument Control

Introduction

In the fall of 1978, the triple quadrupole mass spectrometer designed by Drs. Richard Yost and Christie Enke (2) produced it's first daughter spectra. This instrument, while designed to be controlled by computer, was initially operated manually. The early TQMS instrument used a ramp generator to sweep the mass selected by one of the quadrupoles and had a strip chart recorder for data output. To collect daughter spectra, the first quadrupole was manually set (with a potentiometer) to the mass of a parent ion (identified by an oscilloscope), the sweep generator was switched to the third quadrupole, and a scan was taken. The strip chart recording was then measured and a mass scale added. As this process indicates, the collection and analysis of each spectrum was a time consuming task.

The design of a general purpose, single or multiple microprocessor system for instrument control had been initiated in the fall of 1978 (26). One of the goals of this research was complete control of all instrumental parameters of the TQMS. We realized that this would not be

accomplished overnight, and decided to implement computer control in three phases. The first phase would be a simple digital strip chart recorder, the second phase would be the implementation of the new microprocessor system hardware, and the final phase would be complete computer control of the instrument with a multiple microcomputer system.

Phase 1: Digital strip chart recorder

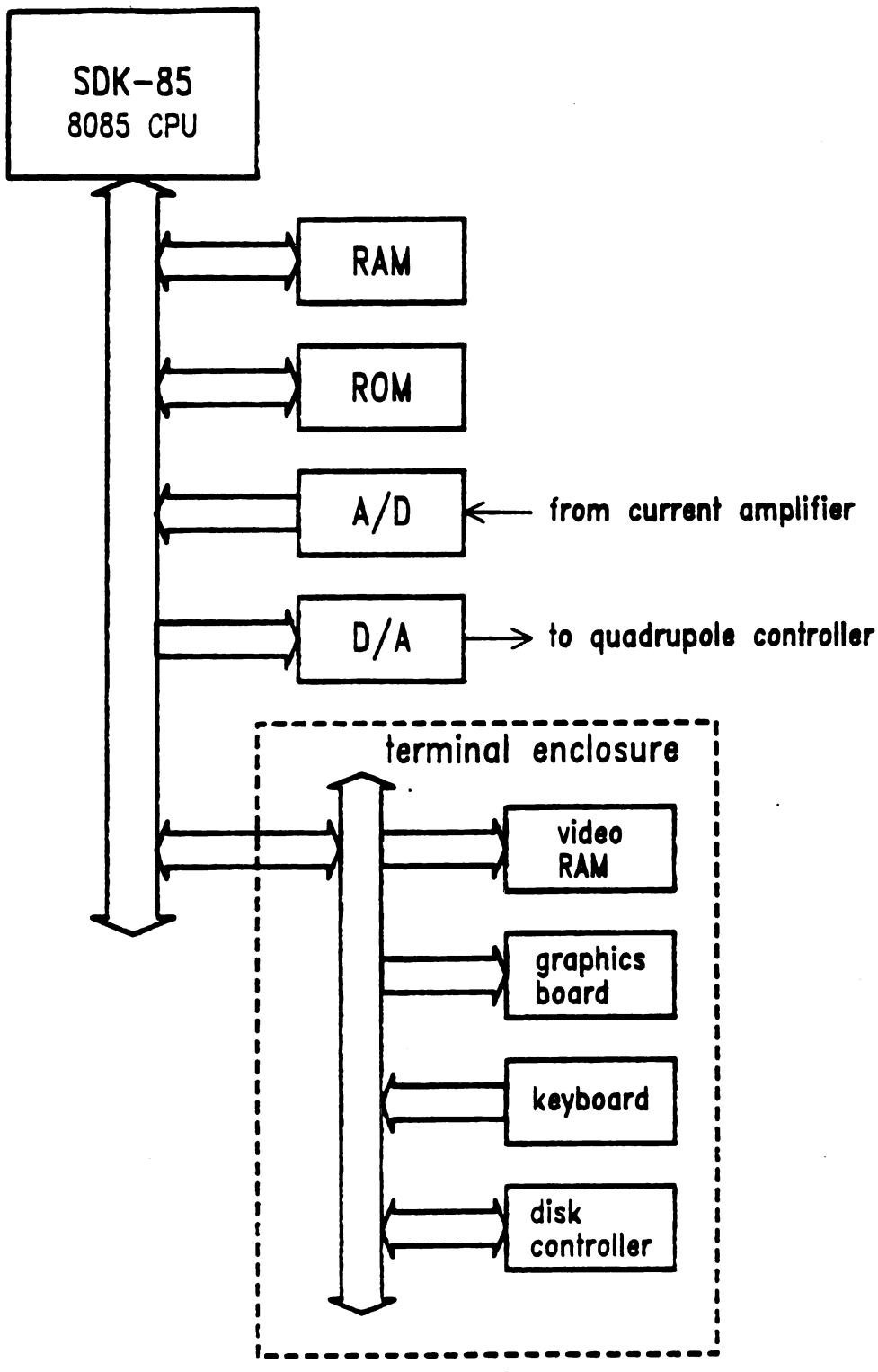
The first phase of the TQMS automation was the implementation of a digital strip chart recorder. This allowed a computer to generate the sweep signal for mass selection and to collect the ion intensities. A computer can then assign the mass values and display the data in several different formats.

Digital strip chart recorder: Hardware

The INTEL 8085 microprocessor chip was chosen to be the heart of the general purpose control computer. An 8085 microcomputer evaluation board, the SDK-85, was convenient for implementing the first phase of the TQMS automation. The SDK-85 is a microcomputer with a monitor (in read only memory, ROM), a limited amount of program memory (random access memory, RAM), a keypad, an eight character display and an area for custom hardware. In the extra space provided on the SDK-85, I wire-wapped interfaces to the

instrument, a floppy disk drive and a keyboard/display unit along with some extra RAM and ROM (see Figure 2.1). The terminal I constructed for this system included two video memories; a graphics memory which provided a low resolution display (256 pixels horizontal by 240 pixels vertical) and a text display which provided 16 lines of 64 characters. Both of these memories, a 9 inch monitor, a parallel keyboard and a disk drive with an intelligent controller were mounted in a terminal enclosure. This terminal was connected to a parallel port on the SDK-85 with a ribbon cable. The instrument was interfaced with a single digital-to-analog converter (DAC) to control the mass selected by one of the quadrupoles, and an analog-to-digital converter (ADC) to measure the ion intensity. The ion current was converted to a voltage and amplified by a Keithley model 18000 programmable current amplifier.

The disk drive (model 270) and intelligent controller (model 1070) were manufactured by PERSCI. The disk controller had an INTEL 8080 microprocessor which handled all the disk functions (seek, read, write, etc) and maintained a file structure. The use of this controller saved much development time, since all the disk related functions were already done. I wrote a program for the lab PDP-11 minicomputer (PIPERSCI) to read and write PERSCI formatted floppy disk. With PIPERSCI, programs are created on the PDP-11 (with good editors and cross-compilers) and



First TQMS control system

Figure 2.1

carried to the microcomputer for use. Data stored on a floppy by the microcomputer are transferred to the PDP-11 for post-processing and display.

Digital strip chart recorder: Software

Software for this control system was written entirely in assembly language. The initial versions of the code were assembled and typed in by hand; later versions were cross-compiled on the PDP-11 and transferred to the microcomputer on floppy disk. A bootstrap for the microcomputer disk was burned into a ROM (after interfacing a PROM programmer to the lab PDP-11 and writing a program to burn the PROMs).

The final version of the control software for this phase consisted of only a few commands (summarized in Table 2.1). These commands consist of only the basics: load in a new program, set the number of data points to average, set the threshold level, scan a quadrupole, store collected data to disk, and display data. This system was not meant to be the final word in mass spectrometry control systems, but was designed to replace the strip chart recording method with a digital method and gain the experience and tools required to design and implement a more complete system. With this system, scans could be initiated and data could be collected and stored on disk.

Table 2.1

Summary of commands for the digital strip chart software

CHANGE	query the user for the following: low mass high mass mass increment number of points to average threshold for saving data
FSCAN	quickly scan from low mass to high mass to observe the peak on an oscilloscope
SCAN	scan from low to high, collecting data
DISP	display the data on the graphics display
WRITE	write the data to the disk

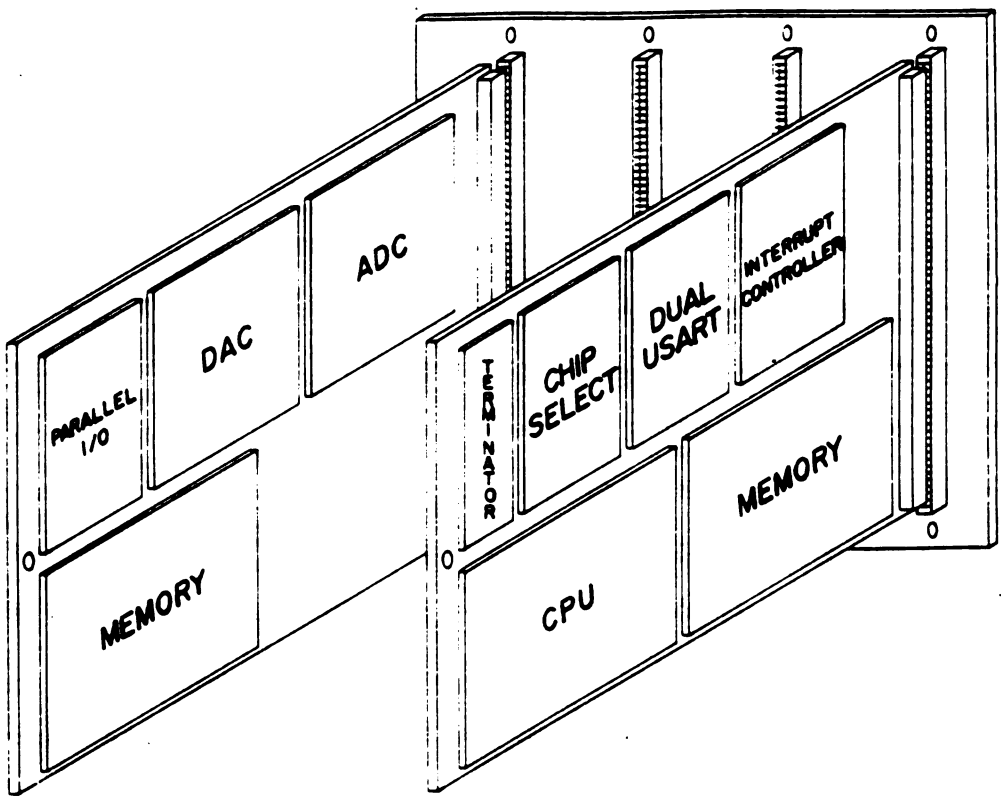
Post processing and display of processed data were accomplished on a PDP-11 computer using programs written by Phil Hoffman (18,27).

Phase 2: Single micro control system

The second phase of instrument control proceeded on two fronts, the development of the control hardware and the software systems. The design and development of microprocessor hardware modules was done primarily by Bruce Newcome (26,28). My role in the development of the hardware was as the software consultant. Together, we would determine whether a hardware design 'feature' would make programming the hardware easier, or conversely make the software much more difficult to code. By designing both the hardware and software together, we were able to trade off functions between hardware and software for the most efficient operation of the resulting system. The hardware will be briefly described here to enable the reader to gain appreciation for the control software.

Single micro control system: Hardware

As can be seen in Figure 2.2, the Newcome microprocessor has bus connections on two levels; each module is attached to a 'mother-board', which is in turn plugged into a backplane. Each 'Bruce-bus' microprocessor module



A typical single microprocessor system

Figure 2.2

performs one specific function. A set of these modules are joined to comprise a microcomputer system. The 'standard' microcomputer board consists of an 8085 microprocessor board, a memory board capable of holding either RAM or ROM, two serial ports and an interrupt controller. This hardware modularity allows a great deal of flexibility in the design of a control microcomputer system.

Specific modules for the triple quadrupole mass spectrometer control consisted of two digital-to-analog converters (DACs), one analog-to-digital converter (ADC), a controller for a Keithley programmable current amplifier, two video displays and a keyboard. Each of the DACs controlled one quadrupole power supply, allowing the software to select masses in both quadrupoles. With computer control of the amplification range of the Keithley amplifier, the microcomputer was able to sample the input signal and adjust the gain for a 10^5 dynamic range. The graphics and alphanumeric video memories were split to separate display units, allowing simultaneous display of data and of the collection parameters. The terminal constructed for the first phase was dismantled and the disk drive was built into the TQMS console.

Single micro control system: Software

The development of microcomputer systems from the chip level necessitated the development of software for testing both hardware and software. At that time, there were no affordable, commercially available software packages that could be adapted to our needs. The first versions of the home-built microcomputers were debugged with highly specific software; the first operating processor/memory combination did nothing but flash a light on the CPU board. As the hardware matured, so did the software. A software monitor (called "Dr. Memory"), able to operate on any standard CPU board, was developed and refined, and used as the basis of more complex software systems. This monitor is the equivalent of a simple operating system, complete with device drivers for the commonly used peripherals (i.e. the terminal and disk). This frees the user to program the higher level functions without concern for the details of the hardware registers. Although this monitor eased the hardware dependencies of the applications software, a more complete programming system and command interpreter was needed.

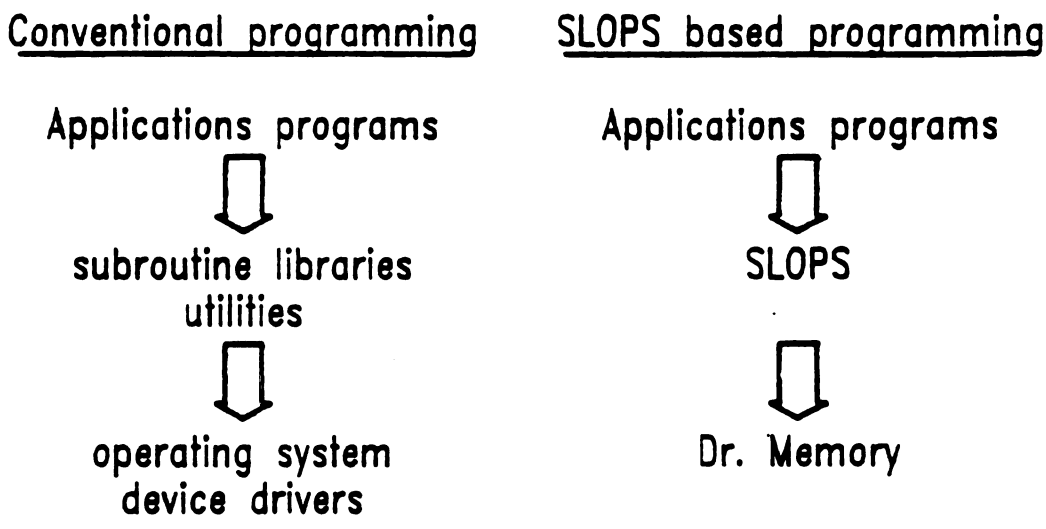
A library of commands, and a programming system was developed (the structured library oriented programming system, SLOPS) as a basis for specific control applications. Finally, control software for the triple

quadrupole mass spectrometer was implemented using the tools provided by both Dr. Memory and SLOPS. In practice, all three levels of programming (monitor, library and control software) were developed concurrently, but they will be described separately below. Today, one would choose from a variety of convenient and inexpensive hardware and software modules that are commercially available, but at this relatively early time in microprocessor applications, we were on our own. Figure 2.3 shows the relationship between currently available software modules and Dr. Memory, SLOPS and the applications programs.

Dr. Memory's monitor

A monitor for the home-built microcomputer systems needs the following characteristics: handle all terminal I/O, allow display/change of memory locations, execute code starting at any location, stop and restart execution of programs and load software. Dr. Memory was designed with all these features in mind, and the code fit in only 2 Kbytes of ROM, complete with help screens, communication software and general purpose I/O subroutines.

Dr. Memory is a general purpose monitor used for both hardware and software debugging. The majority of the available commands (listed briefly in Table 2.2 and more fully in Appendix 1) are for examining and changing memory



Relationship between conventional and SLOPS programming

Figure 2.3

Table 2.2

Dr. Memory's Monitor. (V2.6 9/3/80)

Commands	Description
Break	Breakpoint - stops program execution
<ESC>	Cold restart - used to clean up user stack
O	Octal entry and display format
H	Hexadecimal entry and display
?	Prints a summary of this text
a/	Open location a for modification
a\	Opens two bytes for modification
n<CR>	Modify open location, close it
n<LF>	Modify, close, open next location
n^	Modify, close, open previous location
$r	Open register <abcdefhilABDHSP>
aG	Start user program at location a
P	Procede (at saved PC)
S	Start the second EPROM (SLOPS)
T	Talk to the PDP-11
L	Download from the PDP 11/40

Note: a's and n's are optional, defaulting to the last value.

Restarts	Hex	Description
RST 0	C7	Cold restart - sets default stack, etc.
RST 1	CF	A \
RST 2	D7	BC \ Diagnostics - prints
RST 3	DF	DE) the contents of the
RST 4	E7	HL / indicated registers
RST 5	EF	PC /
RST 6	F7	SP, flags printed
RST 7	FF	Breakpoint (warm restart)

Useful subroutines

Crlf	Outputs a <CR><LF> combination
Downld	Downloads data from the PDP 11
Efclr	Clears an event flag
Getnum	Gets a number input
Gettt	Gets a character from the USARTs
Lights	Sends a predefined light pattern out
Lite	Sends a specified light pattern out
Nulljb	Boredom routine
Print	Prints an ASCII string
Putnum	Outputs a number
Puttt	Writes a character to a USART
Rhlr	Rotates HL right

locations, and starting programs. New pieces of hardware are easily checked out with this tool. The new interface hardware is connected into the system, the power applied, and the operator can use Dr. Memory to access the registers of the new device and manually exercise and test it.

Software (8085 assembly language) entered into the lab PDP-11 computer is cross-compiled and downloaded to the microcomputer. Dr. Memory has two commands that facilitate this process: TALK and DOWNLOAD. TALK is a routine to connect the user's terminal to the PDP-11 through a second serial port. The DOWNLOAD command instructs the microcomputer to accept a program from the PDP-11, and load it directly into memory. Dr. Memory is used to start execution of these new routines, and stop execution at any time to examine memory or registers.

This monitor includes a series of software debugging tools to aid the assembly language programmer. The 8085 restart commands (RST 0 to RST 7) are software interrupts, used in the monitor as debugging breakpoints and display routines. Restarts 1-6 print the contents of different registers and restart 0 initiates a cold restart (as if the power were just applied). The breakpoint restart (RST 7) is the most powerful restart for the programmer, returning control to Dr. Memory. The complete context of the user's program is saved, and the programmer is able to examine and

change registers and memory locations. When control is returned to the program under test, the program's context is restored and program execution continues as if the breakpoint never occurred.

Dr. Memory's monitor internal structure

Internally, Dr. Memory consists of data tables, interrupt handlers, subroutines and a command processor. All the terminal I/O's are interrupt driven, so commands can be entered and buffered while the microcomputer is performing another task. These commands are executed when the current command finishes. A break character from the console terminal acts as a special command, caught by the interrupt handler, which causes the microcomputer to stop executing the current program, saves the program context and returns control to Dr. Memory. The command processor is a simple program that accepts input from the terminal and checks it against the list of available commands. Valid commands are executed while undefined input returns a "Say what?" response.

A special subroutine, NULLJB, is called whenever the microcomputer is not executing a command or when terminal input is pending. This subroutine continually checks for an 'event' to occur (a character input from either serial port, a timer to go off, or any other interrupt process),

at which time control is returned to the calling program if the program was waiting for that event. While NULLJB is waiting for an event to occur, it flashes the lights on the processor board in a characteristic pattern. By watching the microcomputer, one can instantly tell if a program is running or waiting for input.

Structured Library Oriented Programming System

The structured library oriented programming system (SLOPS) defines a structure for both commands and subroutines (which are identical in SLOPS' view of the world). The base SLOPS system fits into 2 Kbytes ROM, and consists of a command processor (more elaborate than Dr. Memory's command processor) and a series of subroutines and/or commands. The basic premise for SLOPS is to create a library of entries (either subroutines or commands) which can be built upon to create new, more elaborate entries. These entries are chained together to form a linked list which can be quickly scanned.

SLOPS is conceptually similar to the FORTH programming language (29), in that they are both threaded, interpreting compilers. When either programming system is started, it is in interactive mode. In this mode, commands entered are immediately executed. A new command definition, when entered, is "compiled", stored in memory, and ready for

execution. This compilation phase searches the library (or dictionary, as it is called in FORTH) for each word in the definition and replaces it with the address of that subroutine, or assembles the instruction if necessary. This process of combining previously defined modules to form more complex functions is the basis of threaded programming.

The major difference between SLOPS, FORTH and CONVERS (a FORTH style language developed by Bonner Denton, et. al. at the University of Arizona) (30-31) is the way arguments are handled. Both FORTH and CONVERS are post-fix languages (similar to Hewlett-Packard calculators), while SLOPS uses pre-fix notation. In FORTH, arguments for the subroutine or function are assumed to be pushed onto an internal stack followed by the function (i.e. 2 2 +), while SLOPS functions assume arguments will follow the function name (i.e. + 2 2). Both of these notations have merits, but pre-fix notation seemed clearer, especially for single argument functions (i.e. MASS 41).

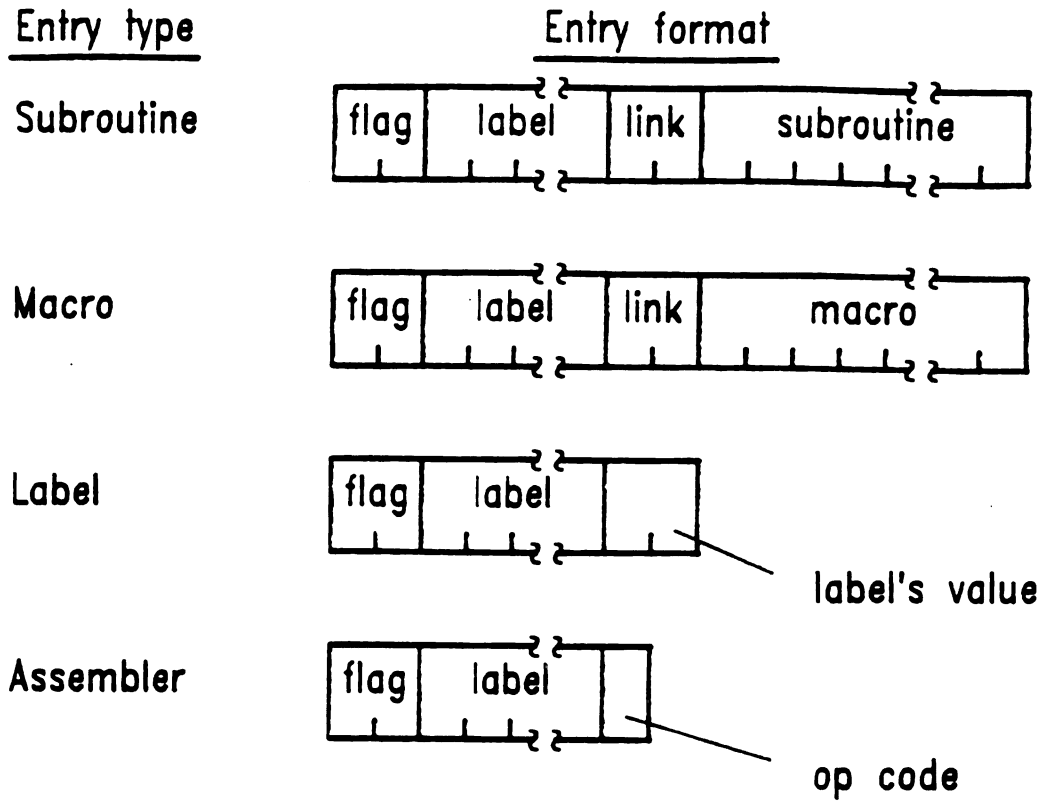
At the time we needed to implement the control software for the TQMS instrument, FORTH was only available for PDP-11 minicomputers. CONVERS was developed at the University of Arizona as a FORTH style language for the INTEL 8080 series microcomputers, and was implemented in our lab by Eric Carlson (32) for an early version of the

multi-microcomputer system used to control a stopped-flow spectrophotometer. Instead of porting CONVERS to the new microcomputer hardware, we decided to use the experience and concepts gained from the development of CONVERS to implement a new, hopefully better, programming system.

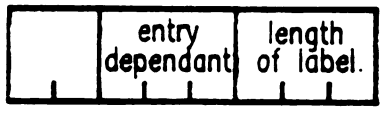
SLOPS internal structure

Each library entry consists of a header flag, the name of the entry, a link to the next entry, and the entry itself. The header flag byte contains the length of the entry's name and a two bit flag describing the type of entry. Only three types of entries are currently used. Figure 2.4 shows each of the entry formats. Two of the formats have variable length entries; hence they have link fields pointing to the next entry in the library. The other two types of entries have fixed lengths and do not require link fields, saving a few bytes of RAM per entry.

The subroutine or command library entry is the most common type of entry. The code for the subroutine starts immediately after the link field and can be as long as the available RAM. The macro library entry is presently unused. The label and assembler library entries are used for the built in assembler/linker. The data field of the label entry is two bytes long, and can be either a value or an address (absolute or relative). The data field for the



flag byte format:



0 0 Subroutine entry
0 1 Label entry
1 0 Macro entry
1 1 Assembler entry

SLOPS library entry format

Figure 2.4

assembler entry is one byte long, and is the microprocessor operational code for this mnemonic.

The command processing routine of SLOPS prompts the user for one or more lines of input, and breaks the input stream down into "words". These "words" are defined as alphanumeric strings delimited by non-alpha characters (spaces, commas, quotes, etc.) The input words are kept in a forward-chaining stack, and the first word in a line is assumed to be a command. The command processor searches the library for a match, and if a subroutine/command is found, it is executed. If the executing procedure needs input parameters, it gets them from the word stack; if no words are available, SLOPS prompts the user for the required input. Any executing task is able to call any subroutine in the system, recursively if necessary. When any procedure is finished, it simply RETURNs to the program that called it. The top level command, when done, returns to SLOPS. If more words are in the word stack, they are then executed in turn until the stack is empty, and SLOPS prompts the user for more commands.

The base SLOPS system contains a number of useful subroutines as well as the command kernel. Table 2.3 contains a list commands and brief descriptions of each subroutine or command, and Appendix 1 is an abbreviated form of the user's manuals for both Dr. Memory and SLOPS.

Table 2.3

Basic SLOPS subroutines and commands

Subroutines

Addr	Returns code address of library entry
Ascii	Converts binary to and from ASCII
Blank	Clears the screen
Brkdwn	Breaks input line into words
Check	Compares word with library entry
Cvtext	Converts a number to ASCII string
Cvtint	Converts ASCII string to binary number
Dcmp	Double compare
Delay	Software time delay
Link	Links to next library entry
Number	Get number from word stack
Search	Search library for a match
Ttyin	Gets a line of input
Word	Get word from word stack

(also see subroutines listed under Dr. Memory)

Arithmetic subroutines

Ddiv	Double divide
Div	Divide
Dmult	Double multiply
Dsub	Double subtract
Mult	Multiply

Commands

Convert	Converts a number to any base
Downld	Loads from the PDP 11 (See Dr. Memory)
Drmem	Puts Dr. Memory in control
Talk	Talk to the PDP 11 (See Dr. Memory)

As can be seen from this list, about half of the subroutines are used by SLOPS to keep track of library entries, etc. The other half are general purpose subroutines (double multiply, convert binary to/from ASCII, etc.) An extended SLOPS system includes an 8085 assembler and many more general purpose subroutines.

The extended SLOPS system is a powerful tool for software development. The user is able to create and test small subroutines and build more complex systems using these building blocks. Execution speed of the resulting commands is excellent due to the extensive use of assembly language and the low overhead of the command processor. When a library of low level commands is built, this system behaves like a higher level programming language, allowing the programmer to use structured programming techniques.

Mass spectrometer control software

SLOPS provided a good foundation for control software for the triple quadrupole mass spectrometer. The new control hardware featured two video displays, one for graphics and another for status display and user interaction. The graphics screen could be logically divided into two halves termed the upper and lower displays. Each of these displays were equivalent, but could be programmed independently. The status display also

had two 'halves' to control each half of the graphics display. The status display, shown in Table 2.4, displayed the current status of the instrument, the mass range for each quadrupole to scan and the threshold for gathering and storing intensity data.

Table 2.4

TQMS status display

	UPPER	LOWER
Mass: Quad 1	xxx-xxx	xxx-xxx
Quad 3	xxx-xxx	xxx-xxx
# pts to avg	xxxxx	xxxxx
threshold	xxx.x:-x	xxx.x:-x
min/max range	-x/-x	-x/-x

Quad 1	Quad 3	Intensity
DC:xxx.x	RF:xxx.x	xxx.x:-x

The two halves of each screen effectively allow two separate experiments to be run simultaneously. This feature, as well as much improved graphics and the ability to change the gain on the amplifier during data collection, made this control software much more convenient and versatile than the 'digital strip chart' of phase 1.

The commands available to the operator of this system are listed in Table 2.5. The control software allows four sets of parameters to be stored (with the SAVE command) and retrieved (with the GET command). This allows the operator to set up several different experiments and quickly switch

Table 2.5

TQMS control system commands

INI Initializes the system, clearing the displays, setting default mass values, etc.

GET Gets a set of parameters from one of four stored setting areas.

SAVE Saves the current parameter table in a bank of parameter tables, for future recall.

PARAM Sets general parameters. It acts in a 'single character input' mode for the parameter to change, i.e. 'Q' not 'QUAD' for changing a Quad's mass range. The parameters for either graph, the UPPER or the LOWER, must be changed individually, and the commands U and L identify which graph is being changed.

The following commands are currently supported:

Q n low-high	Changes Quad n's mass range
T int range	Changes the threshold
R min-max	Changes the min and max range
A n	Changes the # of points to average
G low,n	Changes the low mass on the graph
F	Changes flags, each flag queried
M text	Puts message on the status display
H text	Puts a header (title) on the graph
?	Prints a summary of the options
^Z	exits (control Z)

MANUAL Implements 'manual' control. Using the keypad of the terminal, the keys 1, 2 and 3 are for quad 1; 4, 5 and 6 are for quad 3; and keys 7, 8 and 9 are for the range. Keys 1, 4 and 7 decrease the current value by 1, keys 2, 5 and 8 set the current value into the parameter table, and keys 3, 6 and 9 increase the current value by 1.

FSCAN Increments the mass, checks for a typed character, and if none entered, loops about.

SCAN Scans the set mass ranges, collects data, updates the screens, records the data and loops until a character is typed.

WDATA Writes the data from RAM to a disk file.

between them. Setting parameters is accomplished by using the PARAM command. The parameter changes are immediately displayed on the status and/or graphics displays. The MANUAL command allows the operator to step the quadrupole masses by single DAC units to accurately find the maximum of a peak. For example, when set up for a daughter scan, the MANUAL command allows the operator to tune the first quadrupole mass to the maximum intensity (which might not be an integral mass due to calibration inaccuracies).

Two commands, FSCAN and SCAN sweep one or both (as defined on the status display) quadrupole masses from the preset minima to the maxima. The FSCAN command sweeps quickly, allowing the spectra to be displayed on an oscilloscope. The SCAN command is necessarily slower since data are being collected and averaged. Each of these commands sweeps the entire specified range of one or both displays, and restarts the sweep unless told to stop by any keystroke on the keyboard. One last command, WDATA, writes the data collected by a SCAN command to the disk for post processing by the lab PDP-11 computer.

This control software is written as a series of modular subroutines allowing for independent testing of each module. Appendix 1 briefly describes the library of subroutines created for and used by this system. Enhancements to the control software, such as special

purpose commands, are easily constructed using these routines as a foundation.

Phase 3: Current mass spectrometer control system

The single microcomputer system described above was a temporary solution to the control problem. When that system was implemented, we were aware that the system could not be operated at the high speeds we desired. The third and current phase of instrument control was accomplished with a multiple microprocessor system, designed and implemented to allow several tasks to run concurrently. A full description of the control hardware can be found elsewhere (26,33).

Commercial software tools and systems were being developed and becoming available during the implementation of the SLOPS based system. These tools were now sufficiently powerful, flexible and affordable that we decided to implement the third phase of software control using these tools. The control software for the multiple microprocessor system was written in the FORTH programming language, now available for microprocessors. FORTH is a high level programming language commercially available that is well suited to control systems. The experience gained from the SLOPS based control system was invaluable in the design and implementation of the current FORTH based

software. Complete descriptions of the current control software can be found in Carl Myerholtz's thesis with modifications by Adam Shubert and Mike Kristo (11,34-39).

Conclusions

The triple quadrupole mass spectrometer is a complex and experimental instrument, and automation could not occur in a single step. The three-phase process described in this chapter worked very well, allowing us to become familiar with the advantages and disadvantages of various techniques. The transition from a completely manually operated instrument to a completely computer controlled instrument was arduous both in implementing the harware/software as well as in convincing the operators that a computer can reliably operate a complex instrument.

Chapter 3

Multi-dimensional Instrument Data Base

Introduction

The evolution of GC/MS brought with it the development of computer data systems designed to handle the additional dimension of time as well as the mass and intensity axes. However, even three dimensions are inadequate to cope with the measurement capabilities of MS/MS instruments and many other modern computer-controlled systems. In the case of a totally computer-controlled MS/MS, a large number of variables can be scanned, either singly or jointly. Each such scan will produce one plane of information in a multi-dimensional data base. As an example, the axial energy can be scanned at fixed masses or the mass (in either quad 1 or quad 3) can be scanned with incremental changes in axial energy. Other variables (dimensions) include direct inlet probe temperature, collision gas pressure, ionization voltage, chemical ionization gas pressure, lens voltages (both in the source and between quadrupoles), collision gas type and others. Thus, a new, more versatile data base management system was required.

Fortunately, the storage of all these data has become affordable using computers with large capacity disks and magnetic tapes. Now the analyst can store all data as they are acquired on disk, and subsequently put them on magnetic tape for archival storage. Once these data are stored, the problem becomes one of rapidly and flexibly accessing the data. An additional problem in multi-dimensional instrumentation is the need to extract any plane of information in the data base, even if that plane does not correspond to the types of scans that produced the data base. The concerns of data retrieval are explored in the next chapter.

The chemical literature contains many examples of systems for creating and searching libraries (40-49). There are also suggestions for using pattern recognition (50-55) and artificial intelligence or heuristic (21,56-59) means of interpreting data. Little is said, though, about rapid, versatile and efficient initial storage of raw data in real time. We have developed a system that provides these storage characteristics, provides for storage of multi-dimensional data, and incorporates mechanisms for rapidly accessing the data.

Scientific versus business data bases

In many ways, both business and scientific data bases are similar, but there are significant differences (60-74). Business data bases must be able to handle text ("warehouse X"), integers (52 widgets left in stock), and floating point numbers ($42,000) (60-72). In addition to these requirements, scientific data bases may also have to accommodate bit strings, vectors, arrays, graphs and multi-dimensional data (63-74). Both types of systems must have query languages, or methods to retrieve the data stored in the data base. Business systems usually require a small set of specific answers (all employees with salaries greater than $30,000). Scientific data base retrieval languages or programs must be able to handle the uncertainty (error) present in many scientific measurements (68). Queries must be made in the form "retrieve all energies between -20 volts and -18 volts". Both data base query systems must be able to join several queries ("names of employees older than 62" AND "been with the company longer than 15 years"), and display the results as lists, tables or graphs.

The data formats of business data bases must be defined before use by the data base administrator using a data definition language, and the data are often entered by keypunch operators. The structure of the data base must be

carefully considered and analyzed, for the resulting data base may be in use for many years. This data base definition process can take weeks or months, and has spawned an industry that provides data base definition templates for the popular data base programs. Scientific data bases, especially those created by during an experiment, must be created quickly (possibly automatically) and be able to accept data from the ongoing experiment in real time. A scientist does not have the time to spend weeks organizing a data base for one experiment, and then redefine the data base for the next! A scientific data base must be easily adapted to new and rapidly changing problems.

Survey of data base structures

There are essentially 4 major structures for data bases: flat file, hierarchial, network and relational (60-61,72). Of these, the flat file is the simplest, and is used in the majority of the "spread sheet" programs for micro- and mini-computers (75-76). These flat files are two-dimensional tables consisting of columns for each variable or attribute, and rows for each observation or instance. If the data to be recorded are amenable to this form with minimum duplication of values, this format is efficient and can represent a natural working order of data

from right to left and top to bottom. Example 1 in Figure 3.1 illustrates a simple flat file data base.

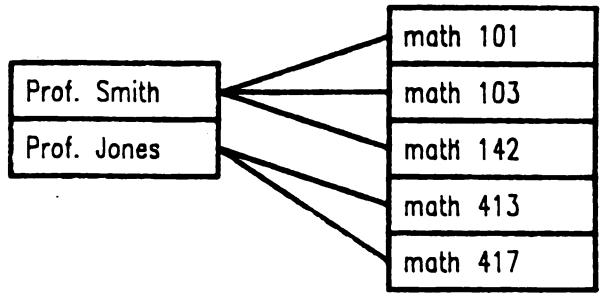
Hierarchial data bases provide 1 to N links between records, such that one root record may have one or more "child" records, each of which may have one or more "grandchild" records, and so forth. An illustration of an hierarchical data base can be seen in example 2 in Figure 3.1, where Prof. Smith teaches 3 classes and Prof. Jones teaches two others. By providing the links between the records, redundant information (Prof. Smith teaches math 101; Prof. Smith teaches math 103, etc.) is eliminated.

Network model data bases provide for N to M linkages between records, and have all the features and advantages of hierarchial data bases. In addition, they provide more flexible ways to interconnect records, allowing almost all redundant information to be eliminated. Example 3 in Figure 3.1 shows a simple network data base. As is expected, this increased flexibility comes at a price; network model data bases are more complex than hierarchial models, and often require special training for data base administrators.

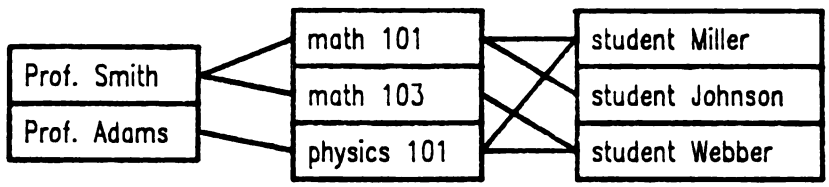
Relational data bases make a radical departure from the linkage schemes used in both hierarchial and network

Prof. Smith	math 101	room 123
Prof. Jones	math 413	room 42

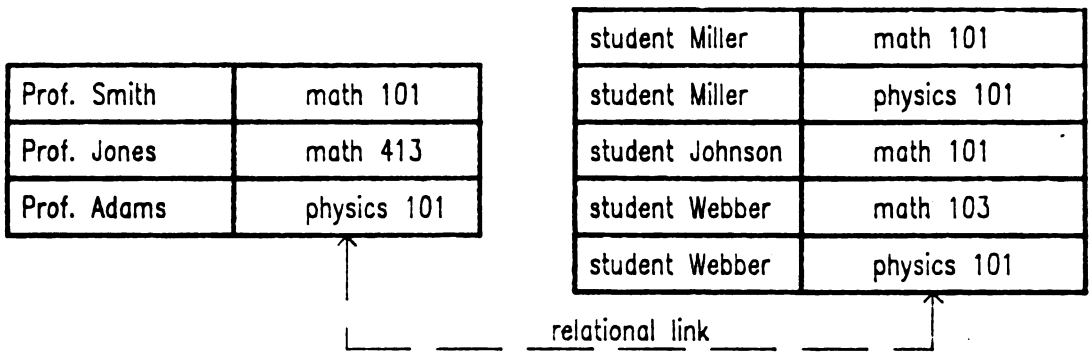
Example 1: flat file



Example 2: hierarchial model



Example 3: network model



Example 4: relational model

Data base structures

Figure 3.1

models. In the relational model, all data are stored in flat file format, and "relations" are established between various tables. In this manner, implicit links are made between relations as opposed to the explicit links needed in hierarchial and network models. The data base system creates these links as needed by "joining" tables together on the fly. Example 4 in Figure 3.1 shows an example of a simple relational data base. This type of data base is easy to maintain (the flat files are conceptually simpler), but require more computational power to join relations for all but the simplest queries. For example, from Figure 3.1, a query "list all students of Prof. Jones" requires the course number from the two tables to be joined, and only students in one of Prof. Jones' classes to be listed.

Data base capabilities

We have included in our system several capabilities that have been either overlooked or not necessary in past data base management implementations for scientific instrumentation (see Table 3.1).

These extra capabilities of the multi-dimensional data base system are essential for use with our triple quadrupole mass spectrometers. There is a large degree of interdependence among the TQMS parameters, and the effects

Table 3.1

Capabilities of the multi-dimensional data base

1) The ability to store all instrument parameters
2) The ability to store a variety of changing parameters
3) The ability to store any X/Y data pairs (such as mass/intensity and energy/intensity pairs)
4) The ability to add comments before, during or after the experiment.
5) One compound or experiment for each dataset.

of these dependencies are not yet completely understood. Storage of all instrumental parameters along with the data is required if we are to fully understand these parameters. To study these interdependencies effectively, the operator may vary several parameters incrementally to obtain a dataset with multiple dimensions of data. These data can later be searched in a variety of ways to study the effects of individual instrumental parameters.

For many research applications, complete mass spectra or daughter spectra are not required. For an energy dependence study, the operator may want to vary the axial energy for a certain parent/daughter fragmentation. This data base is able to handle these data (intensity vs. axial energy) as well as the more usual mass spectral data (intensity vs. mass). The additional capability of allowing the operator to enter comments at any time during the experiment allow the dataset to be used as a notebook,

storing information about that experiment with the experimental data.

An example of multi-dimensional data

The control systems for the triple quadrupole mass spectrometers at MSU have the ability to devise "methods" which allow the rapid collection of multiple dimensions of data. This is useful for characterizing both the instrument as well as a chemical sample. A method to collect data for a study of axial energy and collision gas pressure would follow the outline presented in Table 3.2.

Table 3.2

Method to collect 5 dimensions of data

```
1) LOOP: collision gas pressure:  from low to high.
2)   LOOP: axial energy: from low to high.
3)     LOOP: quad 1 mass: set to parent in the EI spectrum
4)       scan quad 3.
5)     continue to the next parent mass
6)   continue to the next energy
7) continue to the next pressure
```

This method creates a five dimensional dataset: collision pressure, axial energy, quad 1 mass, quad 3 mass and intensity. This dataset contains a wealth of information about the compound, and the data are collected automatically (except for setting the collision gas pressure). The operator is then able to search through

this dataset and extract a plane of information, even if that plane wasn't specifically scanned. For example, with this dataset, we can extract data to produce a plot of the intensity vs. axial energy for a certain parent/daughter pair at a specific collision pressure. Details of data retrieval and extraction from a dataset are presented in the next chapter (24-25,77).

A programmer's view of the data base

The data base definition is complete when the variables to be studied are defined. Since many experimental parameters will not change, an hierarchial model was chosen for this application. At the top level or root of the hierarchy are the parameters that do not change during the course of the experiment. A second level was implemented in order to store a traditional mass spectrum as one record in the data base. This structure was choosen to reduce redundancy in the data base, and hence conserve disk space. The details of the data base structure are presented later in this chapter.

The portion of this system used to write and read data to and from the data base consists of a series of FORTRAN subroutines. These subroutines are easy to incorporate into any computer-controlled instrument. Only a few lines of code are needed to implement this data base system into

the control software (see Figure 3.2). All the work of managing the data in a dataset is accomplished by a set of subroutines which frees the programmer to concentrate on other aspects of the project. Appendix 2 provides more information regarding the subroutines used for data storage and retrieval.

Because the subroutines are short and data storage is efficient, this system could be added to many existing dedicated instrument control computers. It could also be put on a separate, time-shared computer system, with a high speed data link. Both of the triple quadrupole mass spectrometers at MSU are controlled by one or several microprocessors linked to a minicomputer which handles the data storage and retrieval. This allows for the efficient separation of tasks, the data acquisition and the data storage and retrieval. By using separate computers, the instrument can be collecting data while other operators analyze their previously collected data.

Data base file formats

Computer files are generally organized in one of three ways: sequential, indexed or direct (60-62,71,78). A sequential file is the simplest, and as the name implies, is a sequentially ordered collection of information, placed in the file as received and packed in an unstructured way.

```
      PROGRAM SAMPLE
        .
        .
        .
      INCLUDE 'MDDB.CMN'          ! include the commons
      REAL*4 X(500),Y(500)        ! arrays for data
        .
        .
        .
      LOONTI = 5                  ! define the terminal LUN
      CALL MSINIT                 ! initialize things
      IF (IERR .NE. 0) THEN ...   ! check for errors
      IFILE = 1                   ! define the dataset number
      CALL MSOPEN(IFILE,'New dataset name? ','NEW',2,3,4)
                                  ! open new dataset, using
C                                 !   LUNs 2,3,4.
C     Now define the variables
C
      NUMSTC(IFILE) = 3           ! three static variables
      ISTATC(1,IFILE) = 23        !  the first is code #23
      RSTATC(1,IFILE) = 2.0       !  and it's value is 2.
        .
        .
      NUMVAR(IFILE) = 6           ! six variable parameters
      IVAR(1,IFILE) = 42          !  first variable is #42
        .
        .
      CALL PUTPRM(IFILE)          ! and write to the dataset
        .
        .
10    (fill up the X, Y arrays with data)
      RVAR(n,IFILE) = value       ! record the variables
        .
        .
      NUMDAT(IFILE) = xx          ! number of data pairs
      CALL PUTDAT(IFILE,X,Y)      ! and store the data
      IF (more) GOTO 10
        .
        .
      CLOSE (UNIT=2)              ! close all the files
        .
        .
      END                         ! and all done
```

Writing to the multi-dimensional data set

Figure 3.2

If these data are written in such a way that they can be printed or displayed directly without any processing, the file is called a sequential, formatted ASCII file. This type of file is almost never used for large amounts of data, as it requires a great deal of space on the disk and is cumbersome to access.

A variation, of an unformatted sequential file, is again sequential, but the data are written in internal (unformatted or non-printing) format. This is a compact form of storing data. It has been used in some instrument data systems, but has several drawbacks, the most severe being access speed. As with any sequential file, the only way to find any particular piece of data is to start at the beginning of the file and read all entries until the particular datum is encountered. For large datasets, this is a very slow process.

A second type of file format is an indexed file. In this type of file, a "key field" is associated with some part of the data record. the keys from all the records are collected in one or more structured indicies, and access to the records are provided through these indicies. This is a powerful file format, but unfortunately is not part of the FORTRAN-77 standard. It was not used in order to remain within the standard. Another drawback is the time required to write a record; after the data are written, the file

system must update the indicies which could slow the data collection process down.

A third type of file is a direct access file. The file is broken into fixed length records, and any one of these records may be accessed almost immediately. An elementary direct access file, consisting of one record per scan, could be used by an instrument that always puts out a fixed amount of data. For mass spectrometry, this format would waste a great deal of space, as the fixed record length would have to be long enough to hold the maximum number of mass/intensity pairs that might be recorded in any one scan. Table 3.3 shows the advantages and disadvantages of sequential, indexed and direct access files.

Table 3.3

Comparison of sequential and direct access files

	Sequential	Indexed	Direct
Record length	variable	fixed	fixed
Time required to write record	fast	medium	fast
Access speed to specific datum	slow	fast	fast
Memory/storage space	very compact	wastes space	wastes sapace

In our system we have combined elements of several file types to create a fast, efficient system. An unformatted, sequential file, the header file, is used to store a variety of information that describes the instrument, the experimental conditions, and the variables to be recorded. This file also contains the instrumental parameters that did not change during the given experiment, and these variables become the root of the data base hierarchy. A second file, the pointer file, is written with short, direct access records. The major function of this file is to record the starting record number (pointer or index) of the variables and data pairs located in the third file. This file also contains some redundant data for fast data retrieval. The third file, the data file, is also direct access and contains the values for variable parameters and the X-Y data pairs acquired. Another file, the instrument description or dictionary file, is also direct access and each record contains a definition of a variable or parameter. Each file is described in more detail below.

Dictionary file

The instrument description file, or dictionary, contains complete descriptions of the instrumental parameters that may be varied or recorded. To reduce computer storage, all variables and parameters are assigned a code number. These code numbers are the record numbers in

a direct access file which point to the descriptions of each code number. Two descriptions are stored for each code number, a short one (less than 20 characters) and a long one (up to 57 characters), and one flag integer (see Figure 3.3). The short descriptions are used for speed when the operator is entering them, or when display space is at a premium. The long descriptions are more suitable for tables and graphs. The general purpose nature of the data base is thus maintained by conferring the instrument-specific parameter lists to this file, which is, in effect, a kind of conversion table or template. Each different instrument can have it's own dictionary file.

The flag associated with each description in the dictionary tells how that description is used. If the flag is greater than zero for a particular code (say the code for "collision gas type"), the value associated with the variable also needs to be looked up in the dictionary. For example, if we were looking at code 81, "collision gas type", and had a value of 1, we would display "Argon", not the numeric value 1 (see Figure 3.4, example 1). In this case, the flag is a pointer into the dictionary, and the value is an offset from this pointer. However, if the flag were zero, we would use the definition itself; in this case, there either isn't a value (as in the "Argon" example above), or the value has no physical meaning. In the final case, when the flag is negative (or more precisely, a

Record Number	Record Contents (80 byte direct access records)

1

A20	A58	I2
QSDICT	QLDICT	IPTDIC

QSDICT Short description of instrument parameter
QLDICT Long description of instrument parameter
IPTDIC Flag: −1 ==> The value is the answer
(i.e. "70.0" eV)
0 ==> No values
(i.e. "Argon")
>0 ==> Must look up the value
(i.e. gas #1 = "Argon")

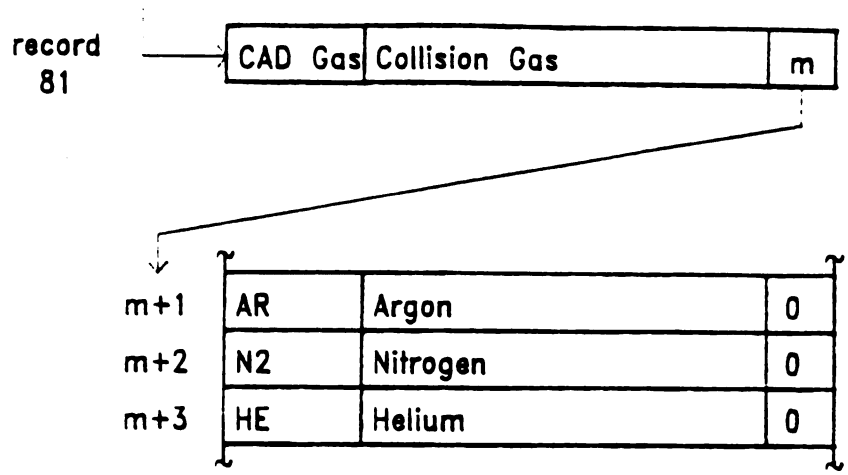
2
⋮
More definitions
END

Instrument description file format

Figure 3.3

Example 1:

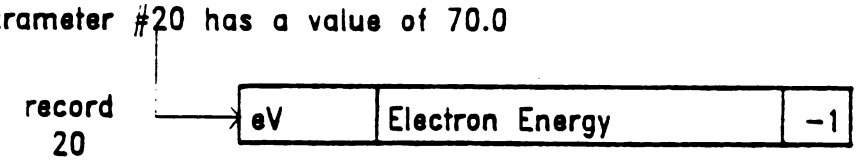
Parameter #81 has a value of 3



Result: Collision Gas: Helium

Example 2:

Parameter #20 has a value of 70.0



Result: Electron Energy: 70.0

Use of the dictionary file

Figure 3.4

"-1"), the value itself has physical significance, such as 70.0 eV electrons (see Figure 3.4, example 2).

The instrument description file is a powerful feature of this system. By changing this dictionary, the data base management software can be used by almost any analytical instrument. Table 3.4 shows a portion of the dictionary for our triple quadrupole mass spectrometer. Modifications to an instrument often add new variables, and many data base systems would need to be fundamentally modified to account for the new parameter. This software only requires the addition of a new definition in the dictionary.

Header file

The header file serves primarily as a notebook, storing a variety of numerical and textual descriptions of the analysis. Included in this file are the values of all instrument parameters that will not change in the experiment. Examples of these static parameters are the operator's name, positive ion mode, EI spectra, etc. (see Table 3.5, a sample dump of a header file). Other parameters that will change in the course of the experiment are called variables, and their codes are listed in this file. The contents and order of data in the header file are diagramed in Figure 3.5 and summarized in Table 3.6.

Table 3.4

Definitions in the TQMS dictionary

1)	Operator	Operator
	1) John	John Q. Public
	2) Milton	Milton Webber
2)	Date	Date of Experiment
4)	Scan type	Scan type
	1) 1scan	Quad 1 scan
	2) 3scan	Quad 3 scan
	3) Dscan	Daughter Ion Scan
	4) Pscan	Parent Ion Scan
	5) Nscan	Neutral Loss (gain) Scan
	6) Sweep	Potential Sweep
	7) Stable ion	Stable ion Scan
5)	Neutral Loss Mass	Neutral Loss (gain) Mass (amu)
9)	Parent mass	Parent mass
10)	Daughter mass	Daughter mass
13)	Source	Source Type
	1) CI	Chemical Ionization
	2) EI	Electron Impact
14)	Ions	Ion Type
	1) Pos	Positive Ions
	2) Neg	Negative Ions
15)	SP	Source Pressure (Torr)
17)	CI Gas	Chemical Ionization Gas
	1) CH4	Methane
	2) H2	Hydrogen
	3) CH4+N2O	Methane + Dinitrogen Oxide
18)	FC	Filament Current (Amperes)
19)	EC	Emission Current (Milliamperes)
20)	eV	Electron Energy (Volts)
21)	Ion Volume	Ion Volume (Volts)
22)	Repeller	Repeller Potential (Volts)
23)	CI Drawout	CI Drawout Potential (Volts)
24)	EIV	EI Ion Volume (Volts)
43)	Q1 Lens 1	Pre-Quad 1 Lens 1 Potential (V)
44)	Q1 Lens 2	Pre-Quad 1 Lens 2 Potential (V)
45)	Q1 Lens 3	Pre-Quad 1 Lens 3 Potential (V)
46)	Q1 Offset	Quad 1 Offset Potential (Volts)
47)	Q1 Mode	Quad 1 Mode
	1) RF	RF Only (No mass filtering)
	2) DC	DC (Mass Filtering)
	3) Scan	Scan
48)	Q1 Mass	Quad 1 Mass (amu)
51)	Q1 Delta M	Quad 1 Delta M
52)	Q1 Res.	Quad 1 Resolution
73)	Q2 Lens 1	Pre-Quad 2 Lens 1 Potential (V)
74)	Q2 Lens 2	Pre-Quad 2 Lens 2 Potential (V)
75)	Q2 Lens 3	Pre-Quad 2 Lens 3 Potential (V)
76)	Q2 Offset	Quad 2 Offset Potential (Volts)
80)	Q2 Pressure	Quad 2 Pressure (Torr)

Table 3.4 (cont'd.)

81)	CAD Gas	Collsion Gas
	1) AR	Argon
	2) N2	Nitrogen
	3) HE	Helium
	4) SF6	Sulfur Hexaflouride
	5) CO2	Carbon Dioxide
	6) CH4	Methane
	7) H2	Hydrogen
	8) CH4+N2O	Methane + Dinitrogen Oxide
104)	Q3 Lens 3	Pre-Quad 3 Lens 1 Potential (V)
105)	Q3 Lens 2	Pre-Quad 3 Lens 2 Potential (V)
106)	Q3 Lens 3	Pre-Quad 3 Lens 3 Potential (V)
107)	Q3 Offset	Quad 3 Offset Potential (Volts)
108)	Q3 Mode	Quad 3 Mode
	1) RF	RF Only (No mass filtering)
	2) DC	DC (Mass Filtering)
	3) Scan	Scan
109)	Q3 Mass	Quad 3 Mass (amu)
112)	Q3 Delta M	Quad 3 Delta M
113)	Q3 Res.	Quad 3 Resolution
114)	EM Lens 1	Pre-Electron Multiplier Lens 1
135)	Conversion Dynode	Conversion Dynode Potential (V)
136)	EM Voltage	Electron Multiplier Potential
140)	Peak Finding Thres	Peak Finding Threshold
141)	Min Peak Width	Minimum Peak width
142)	Max Peak Width	Maximum Peak width
143)	Scan Rate	Scan Rate

Table 3.5

Dump of the parameters in the Header file

11 variable parameters:

1	Modification History
2	Registry Identification Number
3	Quad 1 Mass (amu)
4	Quad 3 Mass (amu)
5	Scan type
6	Quad 1 Mode
7	Quad 3 Mode
8	Neutral Loss (gain) Mass (amu)
9	Source Type
10	Ion Type
11	Quad 2 Pressure (Torr)

24 static parameters:

1	70.0	Electron Energy (Volts)
2	14.9	Repeller Potential (Volts)
3	13.5	CI Drawout Potential (Volts)
4	10.1	EI Ion Volume (Volts)
5	-19.1	External Potential (Volts)
6	21.6	Pre-Quad 1 Lens 1 Potential (Volts)
7	0.000	Pre-Quad 1 Lens 2 Potential (Volts)
8	-53.8	Pre-Quad 1 Lens 3 Potential (Volts)
9	4.20	Quad 1 Offset Potential (Volts)
10	-11.6	Pre-Quad 2 Lens 1 Potential (Volts)
11	-4.90	Quad 2 Offset Potential (Volts)
12	-25.1	Pre-Quad 3 Lens 1 Potential (Volts)
13	-5.60	Quad 3 Offset Potential (Volts)
14	0.144E+04	Electron Multiplier Potential (Volts)
15	-12.5	Quad 1 Delta M
16	-14.0	Quad 3 Delta M
17	0.000	Quad 1 Resolution
18	0.000	Quad 3 Resolution
19	0.000	Quad 2 Pressure (Torr)
20	0.150E+04	Peak Finding Threshold
21	4.00	Minimum Peak width
22	25.0	Maximum Peak width
23	4.00	Scan Rate
24	0.129E+05	Date of Experiment

Comments: TRIAL RUN.

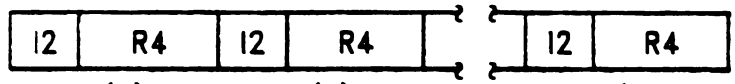
Record Number	Record Contents (unformatted binary variable length)

1

NUMSTC QDATE QTIME
NUMVAR

NUMSTC Number of static (fixed) parameters
NUMVAR Number of variable parameters
QDATE Date the dataset was created (DD-MMM-YY)
QTIME Time the dataset was created (HH:MM:SS)

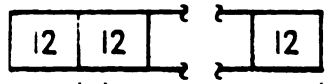
2



ISTATC(1) ISTATC(2) ISTATC(NUMSTC)
RSTATC(1) RSTATC(2) RSTATC(NUMSTC)

ISTATC(i) Static parameter code number
RSTATC(i) Static parameter value

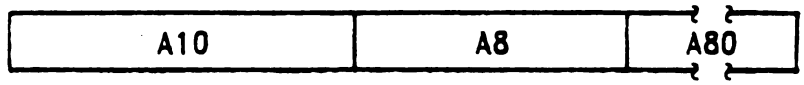
3



IVAR(1) IVAR(NUMVAR)
IVAR(2)

IVAR(i) Code number for variable i

4



QDATE QTIME QCOMNT

QDATE Date this comment was entered (DD-MMM-YY)
QTIME Time this comment was entered (HH:MM:SS)
QCOMNT ASCII comment

5
⋮
More comments

END

Header file format

Figure 3.5

Table 3.6

Summary of Header file format

1) The number of static or fixed parameters (all computer readable instrument settings that will not be changed in this experiment)
2) The number of variables (those parameters which are likely to be changed during the analysis)
3) Date and Time
4) The code numbers and values of static parameters
5) The code numbers of variable parameters
6) Comments as needed, with time of comment entry in front of each.

The header file is constructed so the comments are stored at the end. This allows comments to be entered before, during and after an experiment, thus providing exceptional archival value. The computer clock time is prefixed to each comment so that the coincidence of comments and particular sections of the collected data can be established during post-collection analysis.

Pointer file

There exists one record in the pointer file for each scan performed on the instrument, and each record has as it's first entry a pointer into the data file. This allows one to access any data from an experiment rapidly. The other elements in the record are either for housekeeping, or to enable faster display of the data. The last entries, the fast access variables, are especially important for multi-dimensional work. These are copies of variables

stored in the data file. They are redundant, but this redundancy gains speed when searching the dataset for specific results (see the next chapter for more details).

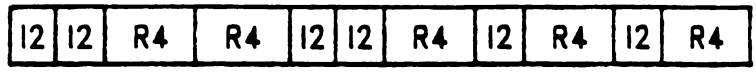
Selected variables, kept in the pointer file, save significant amounts of time by not accessing the data file. For example, if we want to examine the behavior of a daughter ion from a specific parent, and one of the fast access variables is quad 1 mass, we can quickly determine if we must retrieve the data for this scan. If quad 1 mass were equal to the parent of interest, we would get the data for this scan. However, if quad 1 mass didn't equal the parent of interest, we don't have to access the data file, and we've saved the operator some time. The time savings can become significant (from seconds to minutes) for medium to large datasets.

The contents and order of data in the pointer file are shown in Figure 3.6. Each scan produces one record in the pointer file. Each record contains the elements listed in Table 3.7.

The pointer file was designed as a separate file for several reasons. First, since it is a small, direct access file, it can be quickly accessed. Each read of this file immediately brings the important aspects of this scan to light, i.e. the dependent variable, number of data pairs

Record Number	Record Contents (32 byte direct access records)

1



IPTDAT RTIME IVARD RVARF(1) RVARF(2) RVARF(3)
NUMDAT RSUMY IVARF(1) IVARF(2) IVARF(3)

IPTDAT	Index (pointer) into the data file
NUMDAT	Number of X,Y pairs in this scan
RTIME	Time this scan was taken (seconds since start)
RSUMY	Sum of Y values for this scan
IVARD	Code for the dependant variable
IVARF(i)	Code for a selected variable
RVARF(i)	Value for a selected variable

2

⋮

Identical to above

END

Pointer file format

Figure 3.6

Table 3.7

Summary of one record in the Pointer file

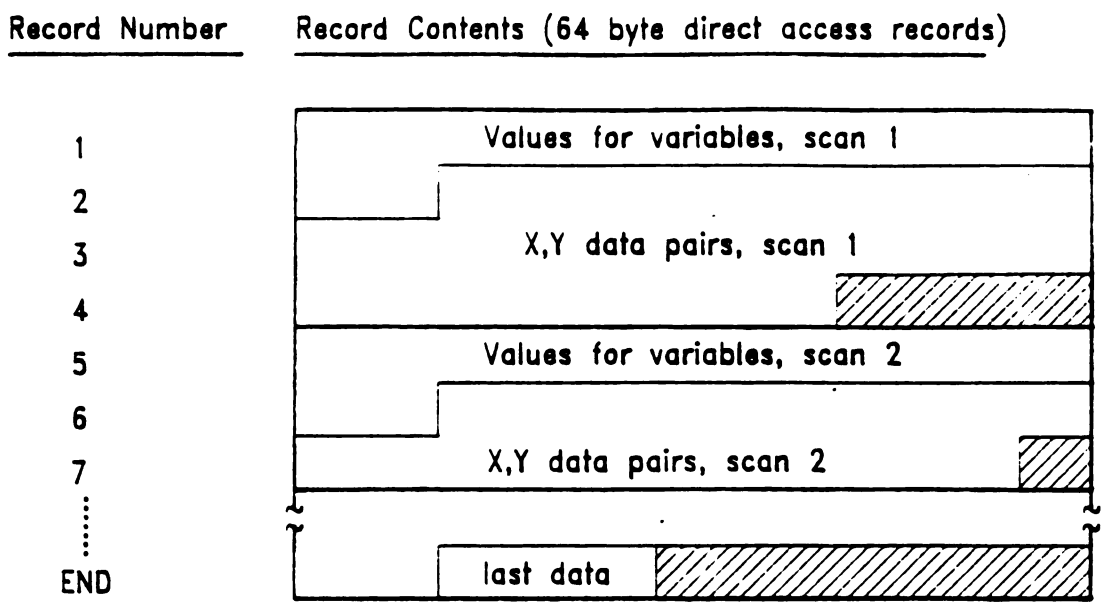
1) The record number (pointer) where the data for each scan begins in the Data file
2) The number of data pairs (for example, mass/intensity or energy/intensity) in the scan
3) Elapsed time (in seconds) from experiment start to scan start
4) The sum of the Y values (for MS, the total ion current) for the scan
5) A code number for the dependent variable in the scan
6) Copies of three selected variables

and most importantly, three selected variables and their values. The second reason the pointers were placed in a separate file was for extensibility. More scans can easily be added without wasting any space caused by preallocation of space within the data file.

Data file

The data file contains one or more records per scan. The contents and order of data in this file are shown in Figure 3.7. Each record or set of records associated with a particular scan contains 1) the values of variable parameters, such as probe temperature, lens voltage, etc. and 2) the X-Y data pairs (intensity vs. dependent variable).

The logical records in the data file are variable length; one scan may have 10 X,Y pairs and the next may



Data file format

Figure 3.7

have 1000 pairs. In this scheme, variable length logical records have been imposed onto one or more direct access physical records. The direct access records allow us to jump quickly to a specific scan. One disadvantage of imposing variable length logical records onto fixed length physical records is that some space following the end of the logical record is wasted. With small physical records, this wasted space is small, and is outweighed by the convenience of direct access to the data. Table 3.8 is a display of one logical record in the data file.

The physical record number that is the beginning of each logical record, or scan, is stored in the pointer file. Each logical record consists of the values of the variables (in the order specified in the header file), and the X,Y data pairs. Each of these values is stored unabridged as real numbers. Since the data are stored in a separate file, adding new scans of data is simply a matter of extending the file.

Creation of a data set

The multi-dimensional data base described follows the hierarchial model. Those variables, instrument parameters and miscellaneous data that do not change during the experiment, form the foot segment of a 1:N tree and are stored in the header file. As data are acquired by the

Table 3.8

Dump of one logical record in the Data file

Scan number 1
Time (seconds) 0
Sum of the Y's 0.600E+05

Variable parameters:

1	0.000	Modification History
2	1.00	Registry Identification Number
3	-1.00	Quad 1 Mass (amu)
4	0.000	Quad 3 Mass (amu)
5	lscan	Scan type
6	Scan	Q1 Mode
7	RF	Q3 Mode
8	0.000	Neutral Loss (gain) Mass (amu)
9	EI	Source
10	Pos	Ions
11	0.000	Quad 2 Pressure (Torr)

Q1 Mass	intensity	Q1 Mass	intensity
18.100	4433.000	32.000	6607.000
28.000	19263.000		

instrument control computers, they are stored in scan records consisting of one record in the pointer or index file, and one or more records in the data file. Again, the hierarchial model is followed and those variables not scanned are stored together with a link to the scanned data (actually, the variables and data are stored together in the data file as described earlier, and the link is implicit).

Business style data bases are designed with "data definition languages" (60-62) which define the records and fields available. A multi-dimensional dataset is defined by creating the header file, specifying the static and variable parameters. This process is usually done by the control computers and the user is not required to learn a "data definition language" or become a data base administrator. This ability to create a dataset tailored to an individual experiment allows for very fast data storage and retrieval. when an instrument is modified and a new parameter added, the dictionary file is simply updated, and new datasets may be created containing the new variable.

Conclusions

The data base described here has several unique features: 1) It has the ability to store multi-

dimensional data in real time; 2) Datasets can be automatically created and tailored to individual experiments; and 3) The data base can be extended with a simple addition to the dictionary file. These features set this data base system apart from those used in business (no data definition language, no complicated structures or programming required). Since each dataset holds information about only one compound or experiment, the overhead and complexity of a complete laboratory information management package (66,67,70,73-74) are eliminated.

The subroutines that comprise this data base are all capable of handling multiple datasets. This is necessary for matching spectra and doing various "massaging" functions on the data, such as averaging spectra. Various programs for matching and data manipulation have been written by Kevin Cross, of MSU. Examples of the use of this system, and a program for extracting orthogonal planes of data are presented in the following chapter.

The instrument data base system described here is an efficient, extensible and modular set of routines to store multi-dimensional data (24-25) in a hierarchial data base. The dictionary file is an extremely powerful mechanism for adapting this system to ever changing instrumentation. This system has been in routine use since 1982 on three

triple quadrupole mass spectrometers, each with a different set of ion optics and features. The three file formats for the data allow a dataset to expand almost without limit. The provisions for the rapid retrieval of data make the system easy to use. The ease of programming and modularity of the subroutines has been proven by the variety of applications for which it has been adapted.

Chapter 4

Retrieval and Display of Multi-dimensional Data

Introduction

The ability to store and retrieve data is essential to its utility. New instrumentation, capable of creating and collecting more data than can be practically analyzed or far more than is ultimately needed is constantly being used to collect unwieldy amounts of difficult-to-access data. Any new computer-controlled instrument can be told to blindly collect and store data, and a disk quickly becomes full of data, much of which will never encounter human observation. However, with the right tools, one might be able to automate a search through this sea of data to extract trends, obtain minimum or maximum values for parameters, or otherwise gain some appreciation for the collected data. With the ability to automatically collect grand amounts of data comes the need to automatically sort and analyze them to extract the reduced set of trends or conclusions we seek (79-80).

In triple quadrupole mass spectrometry, large amounts of multi-dimensional data can be collected by the control computers. This instrument is capable of collecting much

more data in one hour than the operator could possibly analyze in a month. This is not to say that large quantities of data should not be taken, just that tools to look at these data must be utilized. If such tools are not available, and cannot be developed, a more selective approach to data collection is required to simplify the analysis. The latter requires the ability to anticipate over which range of experimental variables the needed information will be found. Since the triple quadrupole MS/MS technique is relatively new, the knowledge needed to anticipate the role of the various parameters is not yet available. In fact, copious amounts of data must be collected in order to assess the effects of these parameters. Since we must take large quantities of data, and since suitable software tools for data analysis were not available, we developed our own tools, notably a program called EXTRACT.

The previous chapter described a multi-dimensional data base suitable for storage of data from MS/MS experiments. Fast storage of data into this data base is essential while the instrument is operational; fast retrieval of data from the data base is also a requirement. In the first case, fast storage is required since the sample may have a limited life. In the second case, fast retrieval of data from the data base is required due to an operator's limited patience. The program we have developed, EXTRACT, provides

for the quick presentation of results from the dataset, as well as several other 'user-friendly' features. In this chapter, I will discuss the considerations that went into the design of EXTRACT; how it works, the user interface, examples of use, internal configuration of the program and possible future directions for this and other data retrieval programs. Appendix 3 is a copy of the EXTRACT User's Guide for details on the operation of the program.

Data retrieval

The data retrieval methods for business and scientific data bases have the same goal: to extract a user defined subset of the dataset for display or a report. The query languages for business systems often involve data manipulation languages, query languages and report generators (60-62). While these full report generation systems are extremely powerful and flexible, programmers or data base administrators are often needed to create the templates required for even simple reports. Since the data base needs of the business community are relatively static (i.e. same inventory form used every week), the weeks or months required to tailor a report generation template are justified.

Full laboratory information management systems (LIMS) often have full reporting capabilities (73-74), but our

needs dictated a much more simple interface. LIMS packages are more akin to business systems and generate standard sample analysis reports. A scientific data base is needed for raw experimental data, tools are needed to view these data from a variety of viewpoints and scientists often need to interactively search the data base for the desired information. The retrieval program must include provisions for: 1) the uncertainty (error) present in our measurements; 2) ever changing instrumentation; and 3) simplicity of use. Simplicity of use is a key point; scientists generally don't want to learn a full query language or design report templates to see the results of their experiments.

EXTRACT - the program

EXTRACT is a general program for the retrieval of data from a multi-dimensional dataset. This program is completely generalized, and can be used with any instrument that uses the multi-dimensional data base described in Chapter 3 (25). The only element that links the data in a multi-dimensional dataset to a specific instrument is the correlation between a parameter code and a physical or logical parameter of an instrument, as defined in the instrument description (or dictionary) file. EXTRACT is instrument independent; it makes extensive use of the dictionary, and all displays are derived from this file.

In this way, the report generation displays are automatically created from the contents of the dataset and definitions from the dictionary.

How EXTRACT works

Data, as stored in the multi-dimensional data base, are stored in scans (intensity vs. something) as recorded by the instrument. If the operator wishes to see these data in this format (i.e. one of the scans performed by the instrument), it is a simple matter to retrieve that scan. However, if the operator wishes data presented along an axis that wasn't scanned, EXTRACT must search through the dataset extracting the data that the user wishes to see. To do this, the operator sets the limits of the search and instructs the program to extract data matching these criteria.

The base rule of EXTRACT is to exclude as much of the data in a dataset as possible, and then to present the user with all data that remains. In this way, we are assured that no datum will go unnoticed unless we specifically reject if from further consideration. Specifically, EXTRACT follows the steps outlined in Table 4.1.

Table 4.1

Outline of how EXTRACT functions

1) EXTRACT loops through each scan in the dataset.
2) The variables stored with this scan are examined; if any variable is outside of the user set limits, this scan is rejected.
3) The data in this scan are examined; if the user requested values that don't exist, this scan is rejected.
4) The values the user wished displayed are extracted and saved.
5) The next scan is examined.

By following these steps, we are certain that all the data that fall within the user-specified limits are extracted and saved for later display. When a datum is retrieved from a scan (step 4 above), the minimum and maximum values of the variables throughout the extraction process are saved. These values are then presented to the user after the entire dataset has been searched, and serve to inform the user of the status of the extraction just performed. Say that the user extracted some data, but did not set any limits for the CAD gas pressure. If the minimum and maximum extracted values indicate a wide pressure range, the operator may wish to set limits on the pressure and extract the data again.

The user sets the acceptable limits for variables, extracts the data, and the program displays the actual minimum and maximum values for all the variables. In this

way, the user may interactively modify the extraction limits while keeping an eye on the resulting data.

EXTRACT internal structure

EXTRACT consists of three main subroutines, a user interface (EXEDIT), a graphics interface (MSPLOT) and the extraction subroutine (XTRACT).

User interface

Various types of user interfaces were considered for this program. There are four main types of user/machine interfaces (see Table 4.2).

Table 4.2

User/machine interface types

1) question/answer or prompting
2) command or switch oriented
3) menu driven
4) icon driven

The first type of interface, prompting, is the simplest to code, but the worst to use. Prompting, or question and answer, is also the easiest for a novice computer user to understand and use; the computer asks questions and the user answers them. However, this type of interface is the

most difficult for an experienced user of the software, for he or she too, must answer each question in turn, and this is a time consuming process. On the other hand, a command oriented interface doesn't prompt the user at all. With this style interface, the user is assumed to be an expert, and has all the commands or switches memorized, and they are entered as needed. For a novice, or any user that doesn't use the software routinely, this type of interface requires the user to have a copy of the user's guide beside the terminal.

The fourth interface, icon driven, is a special graphical form of a menu-driven system, and requires a graphics terminal to operate the software. The menu style interface, not requiring a graphics terminal, was selected as appropriate for EXTRACT. As described in the theory section above, the user must set the limits of variables, and EXTRACT displays the actual limits found. To be a useful, interactive program, all these limits must be presented to the user, the user must be allowed to change the extract limits, and see the results of these changes. A menu format, on a video terminal, fulfills these requirements.

The format selected for the menu is shown in Figure 4.1. This display consists of four sections: two lines for the header or titles, up to 20 lines for the

Variable	Extract Limits min	max	Ext. Results min	max	Variable	Extract Limits min	max	Ext. Results min	max
XScan	(no limit)		(unknown)		Time	(no limit)		(unknown)	
Sum of Y	(no limit)		(unknown)		YY value	(no limit)		(unknown)	
Mod Hist	(no limit)		(unknown)		RID #	(no limit)		(unknown)	
Q1 Mass	(no limit)		(unknown)		Q3 Mass	(no limit)		(unknown)	
Scan type	(no limit)		(unknown)		Q1 Mode	(no limit)		(unknown)	
Q3 Mode	(no limit)		(unknown)		Neutral Lo	(no limit)		(unknown)	
Source	(no limit)		(unknown)		Ions	(no limit)		(unknown)	
Repeller	(no limit)		(unknown)		CI Drawout	(no limit)		(unknown)	
EIV	(no limit)		(unknown)		External	(no limit)		(unknown)	
Q1 Lens 1	(no limit)		(unknown)		Q1 Lens 3	(no limit)		(unknown)	
Q2 Lens 1	(no limit)		(unknown)		Q2 Offset	(no limit)		(unknown)	
Q3 Lens 3	(no limit)		(unknown)		Q3 Offset	(no limit)		(unknown)	
EM Voltage	(no limit)		(unknown)		Peak Findi	(no limit)		(unknown)	

Scan -0.10000E+37

Menu format for EXTRACT

Figure 4.1

display of the variables, a line for the commands and a last line for expanded descriptions. Each of the variable displays consists of 40 character positions, including 10 characters for the variable description, and 4 numeric fields of 6 characters each. This crowded display is necessary to display all the variables, the user set limits and the resulting extraction limits.

These extraction displays are generated from parameters stored in the dataset. The parameter code numbers (form the header file) are looked up in the dictionary file to produce text for the display. All text displays (except the two header lines) are generated this way, making EXTRACT a fully generalized program able to retrieve data from any dataset as described in chapter 3. This automatic creation of customized report generation screens makes EXTRACT easy to use.

Subroutine EXEDIT

EXEDIT is a screen oriented editor, allowing the user to enter and change the limits of the extraction. EXEDIT is a driver that keeps track of the values and their locations on the screen, and calls subroutines to update the screen and parse command input. The command input subroutine is special because it accepts single character input from the terminal, decides if the user is starting to

type a command or a number, and prompts the user accordingly. By doing this, the advanced user may reduce the number of keystrokes required to enter extraction limits. This input subroutine also allows the user to enter both the minimum and maximum limits separately, or enter them at the same time.

Upon entering the EXTRACT program, none of the variables have extraction limits set (as in Figure 4.1); the user must set these limits as desired. To change a variables limit, the user moves the highlighted or active area using the cursor keys on the terminal. Once the variable to change has been selected ("Scan" in Figure 4.1), the user simply types in a new value, and that value is inserted into the display. Since each variable position on the screen is only six characters long, some significant figures may not be displayed (see Table 4.3).

Table 4.3

Numeric representation for EXTRACT

	Full number	Truncated number
integers and floating point numbers less than six characters long	42	42
	123.4567	123.45
	-63.7	-63.7
floating point numbers greater than six digits	1.23E+07	.123+8
	-2.34E-17	-.2-16

The worst case truncation of the numbers is for small negative numbers, where the number is truncated to one digit plus the exponent. Note that EXTRACT displays the full number on the last line of the display.

To alleviate this problem, the last line on the screen is an enhanced description of the active area, including the full dictionary definition, and the full value, or the value's description, if there is one.

EXTRACT is based on the principle of retrieving a plane of data from a multi-dimensional data base. The retrieved plane of data can be visualized as a plot of Y vs. X. These X and Y variables are chosen by positioning the active area to a variable, and typing either X or Y (or Z for three dimensional plots, see description below). The selected X and Y variables become the axes titles for the resultant plot.

Presentation of EXTRACT results

Once the user has entered all the limiting values for the extraction and specified the X and Y axes, the program is told to extract all data that are not outside of the given limits. This extraction process can take from micro-seconds to several minutes, depending on the size of the data base and the type of extraction desired. When the extraction is complete, the screen is updated, showing the user-requested limits and the actual extraction limits found. The actual limits will always fall within the given limits, but may indicate a larger range than acceptable. For example, if no limits were set for a variable, say

quad 2 pressure, an extraction performed, and the resulting display showed that the actual limits of quad 2 pressure ranged from zero to 1.0×10^{-4} torr, this would indicate that several points were extracted at various pressures, and the resulting plot would be meaningless. In this case, the user should limit the pressure range, and extract the data again.

When an extraction has yielded limits within an acceptable range, the data may be filed and plotted (using the graphics program MULPLT). The link to the graphics program has been automated (subroutine MSPLOT) and can be called up in two keystrokes. This automated link sets up a variety of scaling factors, axes limits and other parameters for the user, in one of four formats: point plot, line plot, bar graph and spectrum plot. The first three of these are options of MULPLT, but the fourth, a spectrum plot, is a specialized bar graph that normalizes the data to the base peak. By calling MULPLT (81) directly from EXTRACT, either simple or publication quality graphics, on a variety of graphics devices, can be obtained. (MULPLT was originally written by Dr. Tom Atkinson, but I have extensively modified it in recent years, adding such features as color, new commands, new device support, shading of bar graphs, speed enhancements, a post processor for raster printers and multiple plots, etc.)

EXTRACT is capable of generating pseudo three dimensional plots by extracting a series of two dimensional planes of data, each offset by a specific 'Z' variable increment. These planes of data are extracted and filed, and a post processing program (PLOT3D) is used to generate a series of MULPLT commands to produce pseudo three dimensional plots. These plots are not true three dimensional plots; they are just two dimensional plots offset with X and Y increments. True 3D plots are projection plots, and usually have the 'hidden' lines removed. They are "prettier", but the hidden data are inaccessible, a feature considered undesirable for most of our applications.

Subroutine XTRACT

The subroutine XTRACT was written with speed in mind. Since one complete pass through the entire dataset is required for a thorough extraction of data, hooks are built in to extract more than one plane of data in one pass through the dataset. While the ability to extract several planes of information in a single pass would dramatically speed up three dimensional extractions, it would have no effect on standard two dimensional extractions. Currently, three dimensional plots are semiautomatically made by EXTRACT calling the subroutine XTRACT many times. Each time XTRACT is called, one more two dimensional plane of

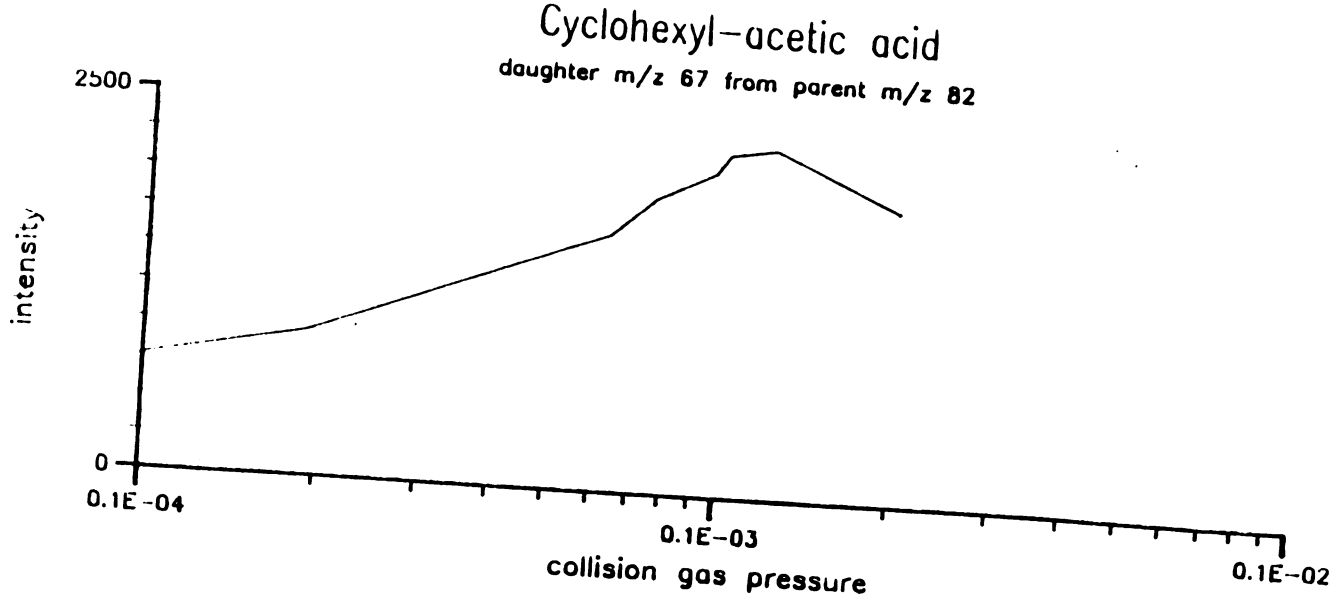
data is added to a three dimensional file for future plotting. Although the multiple plane extraction feature has not yet been utilized, there is no reason to believe that it will not work. To implement this feature would require modifications to the EXTRACT main program and possibly to the EXEDIT subroutine.

Examples of EXTRACT use

EXTRACT has proven to be a very useful program for looking at data collected by a triple quadrupole mass spectrometer. By using this tool, we are able to collect large amounts of data from one sample, and present them in a variety of ways. This allows us to easily pick out trends in the data, and identify the effects of instrument parameters on the collected data. By using EXTRACT, we have been able to quickly gain new insights into the processes going on in the TQMS instrument.

A five dimensional matrix (intensity, quad 1 mass, quad 3 mass, collision energy and collision gas pressure) of data is often obtained for a compound on the TQMS. An example of such a dataset is used for Figures 4.2 and 4.3, of the compound cyclohexyl-acetic acid. The upper portion of Figures 4.2 and 4.3 shows the extraction limits used to produce the plots shown in the lower portion of the figure. In the case of Figure 4.2, a pressure plot for a specific

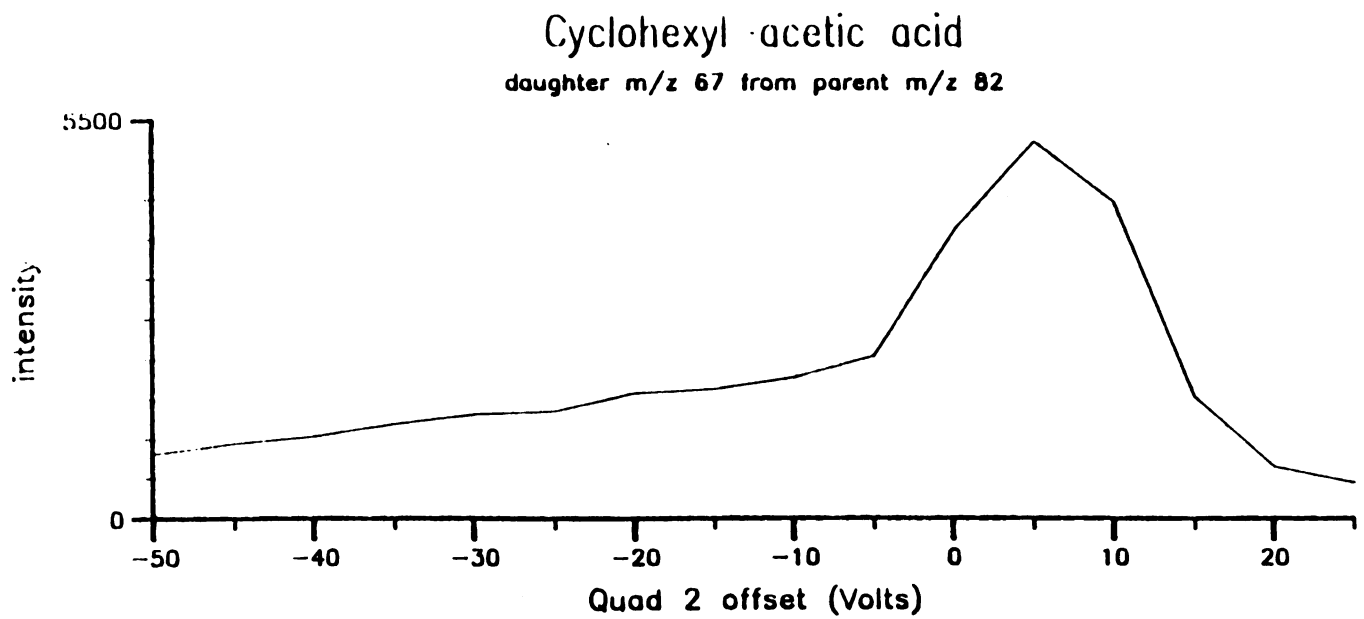
Variable	Extract Limits min	max	Ext. Results min	max	Variable	Extract Limits min	max	Ext. Results min	max
Scan	40	200	49	175	Time	(no limit)		79231	80083
Sum of Y	(no limit)		7109	30277	YY value	(no limit)		756	2300
Mod Hist	(no limit)		0	0	RID #	(no limit)		49	175
Q1 Mass	81.5	82.5	82.1	82.1	Q3 Mass	66.5	67.5	67.2	67.2
Scan type	(no limit)		3	3	Q1 Mode	(no limit)		2	2
Q3 Mode	(no limit)		3	3	XQ2 Pressur	(no limit)		.100-4	.200-3
Source	(no limit)		2	2	Ions	(no limit)		1	1
Q2 Offset	-5	-5	-5	-5					



Example plot of intensity vs. pressure extraction

Figure 4.2

Variable	Extract Limits min	max	Ext. Results min	max	Variable	Extract Limits min	max	Ext. Results min	max
Scan	40	200	130	145	Time	(no limit)		79821	79887
Sum of Y	(no limit)		970	26554	YY value	(no limit)		492	5211
Mod Hist	(no limit)		0	0	RID #	(no limit)		130	145
Q1 Mass	81.5	82.5	82.1	82.1	Q3 Mass	66.5	67.5	67	67.2
Scan type	(no limit)		3	3	Q1 Mode	(no limit)		2	2
Q3 Mode	(no limit)		3	3	Q2 Pressur	.990-4	.110-3	.100-3	.100-3
Source	(no limit)		2	2	Ions	(no limit)		1	1
XQ2 Offset	(no limit)		-50	25					



Example plot of intensity vs. energy extraction

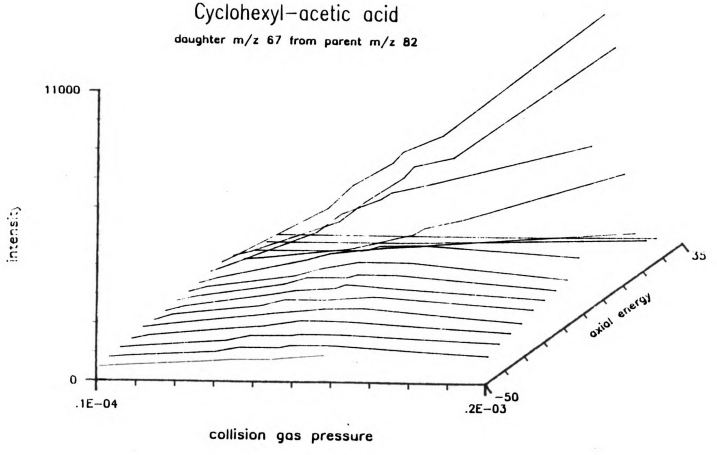
Figure 4.3

parent/daughter combination at a specific collision energy has been extracted. Figure 4.3 is from the same dataset, showing an energy plot for the same parent/daughter at a specific pressure.

EXTRACT is capable of generating pseudo three dimensional plots, enabling the user to see trends in several dimensions at once. Figure 4.4 is an extraction from the same dataset used in the above examples, but now three dimensions of data are displayed. In this example, if we were looking for the energy/pressure that resulted in the largest daughter intensity, we would see that quad 2 offset needs to be about 5 volts, and the collision pressure about 2×10^{-4} torr.

Conclusions

EXTRACT has proven to be a valuable tool in the analysis of multi-dimensional data. The ability to extract data in a plane other than that scanned gives the operator more flexibility designing an experiment. For many samples, collecting a full three dimensional map (intensity, quad 1 mass, quad 3 mass) at a variety of collision pressures and energies allows us to determine the optimum collision parameters.



Example plot of three dimensional data extraction

Figure 4.4

EXTRACT is an instrument-independent program; it is capable of working with any multi-dimensional data base (as described in the last chapter). This is due the use of an instrument description (or dictionary) file. EXTRACT has no instrument specific parameters coded in; instead the dictionary file provides translation for the variables and other parameters associated with the multi-dimensional data base.

The generality of EXTRACT and the multi-dimensional data base was proven by their use with simulated electrochemical data. Simulated data was generated and stored in the data base. EXTRACT was used to retrieve and display these data in ways that would have been difficult or impossible to simulate directly.

The use of the dictionary file for EXTRACT display has the added advantage of easing report generation. With business data bases, the field names must be known and entered with each query ("list all employees with salary GT 35000"). The menu format for EXTRACT displays all the parameters in a dataset, and the cursor keys provide an easy way to select the variables.

EXTRACT provides for range limits and feedback for each variable to minimize the impact of data uncertainty. Being able to select the extraction limits for a variable and see

the actual extracted range for all variables make EXTRACT a powerful tool for retrieving scientific data.

Chapter 5

Extracting the Information Contained in MS/MS Data

Introduction

The triple quadrupole mass spectrometer is capable of generating large amounts of data. The analysis of these data in a reasonable time would be impossible without the use of the software tools presented in the last several chapters. These and other tools allow the operator to examine the collected data and display them in a variety of formats. These displays present the operator with the raw data as they were collected (or possibly normalized), and allow the operator to interpret the results. The next logical step in computer-assisted problem solving is to automate the extraction of the information present in the data.

There are two predominant ways that a computer can help interpret data from a mass spectrometer: expert systems and pattern recognition. Artificial intelligence (AI) is a field of computer science dealing, generally speaking, with making computers "think". A subfield of AI is the study of expert systems. An expert system (82-84) is a computer program that mimics a human expert in a specific field of expertise. This is currently possible only for small, well

defined problems for which there exists a well-defined "knowledge base" of information on which the program bases its decisions. Essentially, these programs apply a set of "rules" to the data, presumably the same rules that human experts use, to extract the information present in the data.

Several researchers have applied expert systems (85-93) to mass spectrometry. The DENDRAL project, from Stanford University (21,85-93) has shown that expert systems could be used to help interpret mass spectra. The original DENDRAL algorithm, developed by J. Lederberg, was able to identify all possible acyclic molecular structures given a set of constituent atoms. Heuristic DENDRAL achieved the same objective in less time by using mass spectrometric data and rules to infer constraints on the structure generation. Meta-DENDRAL was then developed to automatically generate the rules for Heuristic DENDRAL. CONGEN was later developed to replace the older DENDRAL algorithm, and was able to generate cyclic structures. DENDRAL has three functional units, Plan, Generate and Test. The planning phase uses rules to constrain the generate phase (using CONGEN). The test phase then uses another set of rules to "fragment" the generated structure and compare the resulting mass spectrum with the unknown. In this way, the generated structures could be ranked, and top ranked structures are often indicative of the unknown's structure.

One program from the DENDRAL project, GENOA (22-23), is a program to generate all chemically possible structures from the molecular formula, constrained by any additional chemical information presented to it. We are currently using this program in our laboratory as one part of our structure elucidation scheme (see Figure 1.1). Another expert system applied to mass spectrometry is the system developed at Lawrence Livermore National Laboratory (95-98). Currently this system is capable of tuning a triple quadrupole mass spectrometer based on both signal intensity and peak shape.

Pattern recognition is a field of study based on the assumption that data can be clustered or grouped into distinct sets (50-54,85-87). Each of these sets is presumed to have one or more characteristics that distinguish it from other sets. A subset of this field is routinely used in mass spectrometry: spectrum matching (85-87). Any MS data system capable of searching a library for reference spectra that "match" an unknown is performing a simple pattern recognition task. The criteria for separating normal EI mass spectra into groups, with top ranked matches, have been studied extensively (40-49).

In this chapter, I will present two tools that help a chemist extract a small portion of the information present in MS/MS data. MS/MS data are different (i.e. more

specific) than "normal" MS data, and new rules for information extraction are required. The first tool presented is a knowledge-based program to aid in the analysis of neutral losses from parent ions. The second tool is an example of a pattern recognition technique, and is used to match daughter (or neutral) spectra.

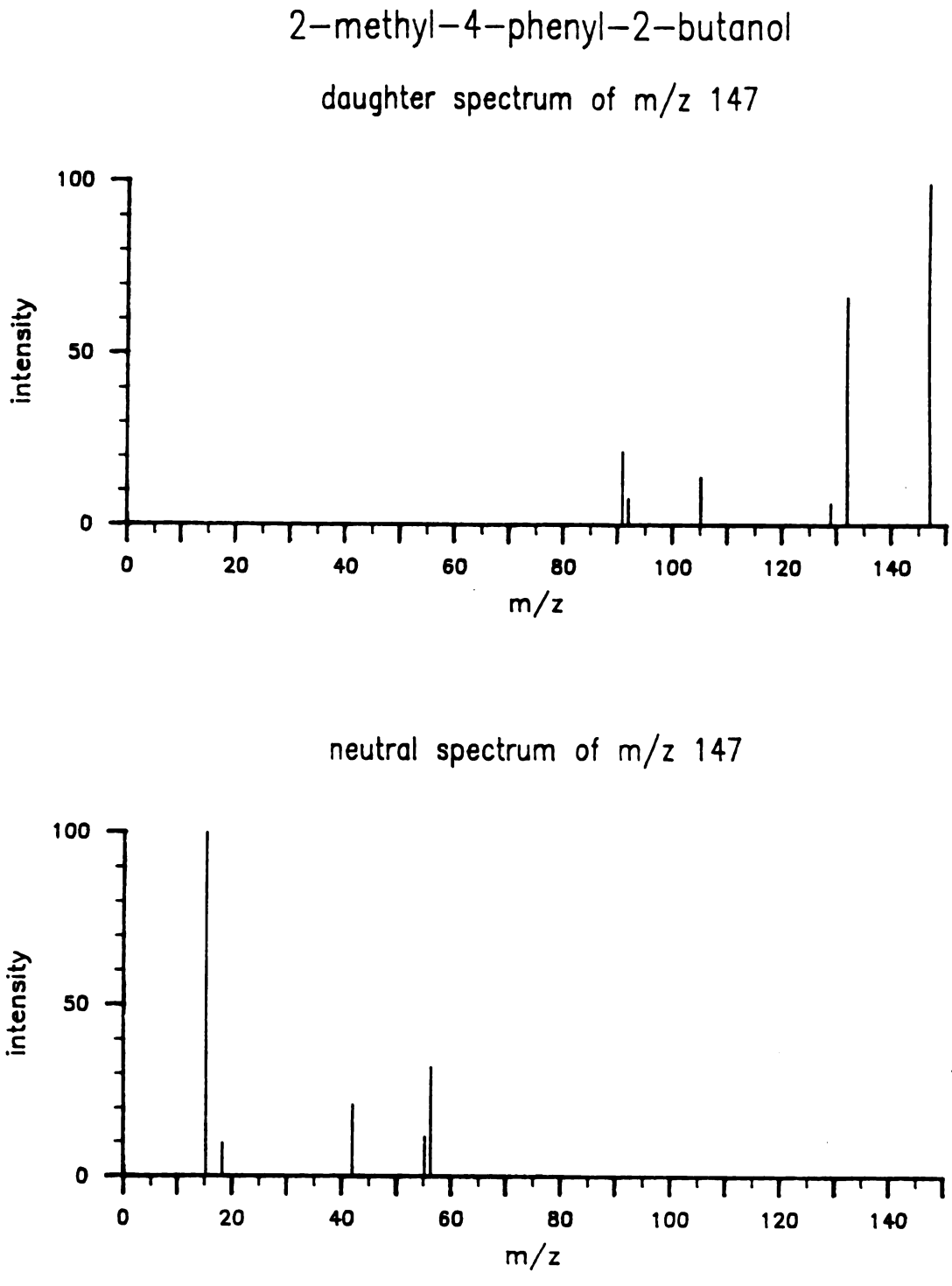
Information contained in MS/MS data

Mass spectrometers have traditionally been used to aid chemists in determining the structure of unknowns. The information contained in a typical mass spectrum comes from several basic sources, including 1) the absolute mass to charge ratio of an ion; 2) the relative m/z value relative to another ion in the spectrum; 3) the intensity of an ion peak relative to the base peak intensity, total intensity or another ion's intensity; and 4) the absence of ions at certain m/z values. The absolute mass of a peak shows the presence of a relatively stable charged species, giving an indication of it's chemical makeup. The relative masses of two or more peaks gives the mass(es) of non-charged species that may have been lost from the higher mass ion. The relative intensities of the ion peaks in a spectrum yield information about which fragmentation pathways are most predominant. The data present in a mass spectrum are interpreted by analyzing these absolute and relative masses and intensities according to well-defined rules (99).

A set of all daughter spectra from a sample contains all the information present in an EI spectrum from the same sample. Each daughter spectrum has the advantage of being simpler than the EI spectrum of the molecule, but the concatenation, or overlapping, of all the daughter spectra produces a spectrum similar to the EI spectrum. The relative simplicity of a daughter spectrum is due to the selective nature of the first mass filter and the less severe nature of the second collision. Each daughter ion contributes evidence of the composition of its parent ion, and can be used to help interpret the structure of the parent. Due to these differences, new information extraction rules or algorithms are needed to analyze MS/MS data.

Neutral spectrum

A neutral spectrum is similar to and derived from a daughter spectrum. A neutral spectrum presents the relative amounts and masses of neutral fragments lost in the fragmentation of a parent. These scans are easily derived from a normal daughter spectrum by simply subtracting the daughter masses from the parent mass. The resultant spectrum contains the same information present in a daughter spectrum (less the parent ion, now at mass 0), but is presented in a different form. Figure 5.1 shows



Comparison of daughter and neutral spectra

Figure 5.1

both daughter and neutral spectra for parent m/z 147 from 2-methyl-4-phenyl-2-butanol.

One of the techniques for identifying an unknown from its EI mass spectrum is to characterize the neutral losses from ions in the spectrum. The characterization of these losses is complicated by the richness of an EI spectrum, in that it is difficult to tell exactly which ions are formed from neutral losses from the molecular or any other ions. Losses are most easily observed from the molecular ion, looking backwards down the spectrum. If the next ion present is 28 daltons less, the parent lost either CO or C_2H_4. However, as we proceed down the spectrum, it becomes unclear where a specific loss occurred from. This limits the usefulness of this technique to confirming postulated structures.

In tandem mass spectrometry, we can determine exactly which daughter ions are formed from which parent ions. If we set up the collision conditions (CAD gas pressure, axial energy) so as to ensure only a single collision for the parent ion and little or no chance that the resultant daughters will further fragment, we know that all the daughter ions formed are direct, single-event fragmentation products of the parent. If the collision gas pressure were too high, some of the daughter ions formed would again collide with the target gas. These second and higher order

fragmentation processes give the same types of uncertainty in a daughter spectrum as in an EI spectrum. Therefore, if we keep the collision conditions conservative (CAD gas pressure low, medium energy), we produce spectra of first-order collision products. These spectra are not as rich as spectra produced at higher pressures since only first-order products are generated. However, first-order daughter and neutral spectra give direct, definitive information about the composition of parent ions.

Daughter spectra and neutral spectra present essentially the same information, but this information can be used in a complementary fashion. Daughter spectra present those species that retain the charge during fragmentation, while neutral spectra present the uncharged fragments. Deriving neutral spectra gives spectra of a form amenable to the same matching techniques used for ion spectra. If the spectra were left in daughter ion form, different pattern recognition techniques (such as sliding correlations) would have to be used to group spectra by neutral loss information. By analyzing both of these spectra, we are able to deduce some information about the parent ion's composition and structure.

A simple expert system for neutral loss analysis

The low mass neutral fragments are simple structures, generally corresponding to simple, common neutral losses. I have written a program (ANEUT) to aid in the determination of these losses. "Expert knowledge" may be encoded in a knowledge-based system, such as an expert system, in two primary ways: as rules and as base of knowledge upon which rules can draw information (85-87,90-93). The knowledge base for this program was derived from a table of possible losses from appendix A.5 in McLafferty's Interpretation of Mass Spectra, and was enhanced as we gained experience with the program. The conclusions drawn from this deductive system are simply a list of possible neutral losses - it is the chemist's responsibility to utilize the resulting information and update the knowledge base as required.

ANEUT knowledge base and rules

The knowledge base is a text file which allows for the easy extension of the knowledge available to the program. This file consists of one entry for each possible loss, or series of losses. Figure 5.2 shows several entries from the current file, and the format of each entry. For series of losses, such as C_nH_{2n+1}, an alkyl loss, limits are placed on the values of n (for this case, n is greater than

range for n	formula	Comments
1, 3	Cn On	(CO)n Aromatic oxygen compounds, cyclic ketones, R--C=-O+
1,20	Cn H(2n+1)	Alkyl loss
0, 3	Cn H(2n+1) C O	R'CO--R (stable R+ only)
0, 0	N H3	Amines
0, 0	H2 O	Alcohols

Entries from the knowledge base for ANEUT

Figure 5.2

0). These limits, the formula, and a comment comprise one record. The comment describes the conditions or conformation that lead to a specified loss.

There are currently only three "rules" for the program ANEUT. One rule is used to match the actual losses (found in the spectrum) with the possible losses (generated from the knowledge base). Another rule is used to allow higher mass neutrals to be a sum of lower neutrals, and the final rule adds a "quality" to a possible loss. The primary rule is simple: if the mass of an actual loss is within $\pm$0.3 daltons of the mass calculated from the knowledge base, that loss/knowledge base entry correlation is retained. The mass deviation of $\pm$0.3 allows for instrumental effects (the instrument may be slightly off calibration, etc). This rule searches the entire knowledge base generating a list of correlations. It should be noted that this rule does not guarantee a complete and exhaustive check of all possible neutral losses, instead it relies on the knowledge base to contain a fairly complete list of the common neutral losses.

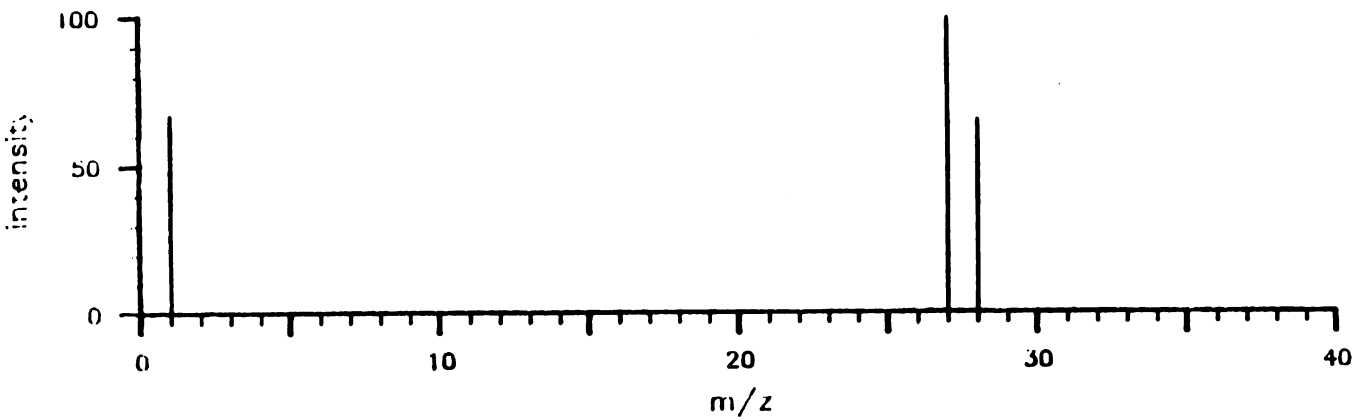
The second rule allows several of the neutral losses to occur, and effectively adds their masses together. The third rule adds a "quality" factor to certain of the correlations made by the first rule. If the unknown daughter or neutral spectrum contains a series of losses

(such as 15, 29, etc. from C_nH_{2n+1}), each of these entries is flagged as being the likely loss. If just one of the elements in a series is present, it is not flagged. This is based on the assumption that if a series of losses is possible, the fragmentation process will generate the entire series.

Example of ANEUT use

Examples of ANEUT's use are seen in Figures 5.3 and 5.4. Figure 5.3 shows the neutral spectrum from parent m/z 108 (the molecular ion) of 1,4-benzenediamine and the output from ANEUT. As can be seen from this list of possible neutral losses, the peaks at 1 and 27 daltons are well accounted for (H and HCN), while the peak at 28 daltons is the loss of both H and HCN. Figure 5.4 shows the neutral spectrum from parent m/z 147 (probably $HOOC(CH_2)_4COOH_2^+$) from bis-2-ethylhexyl adipate and the results from ANEUT. After ignoring the Cl, N and S containing structures, we are left with fragments indicative of acids and alcohols.

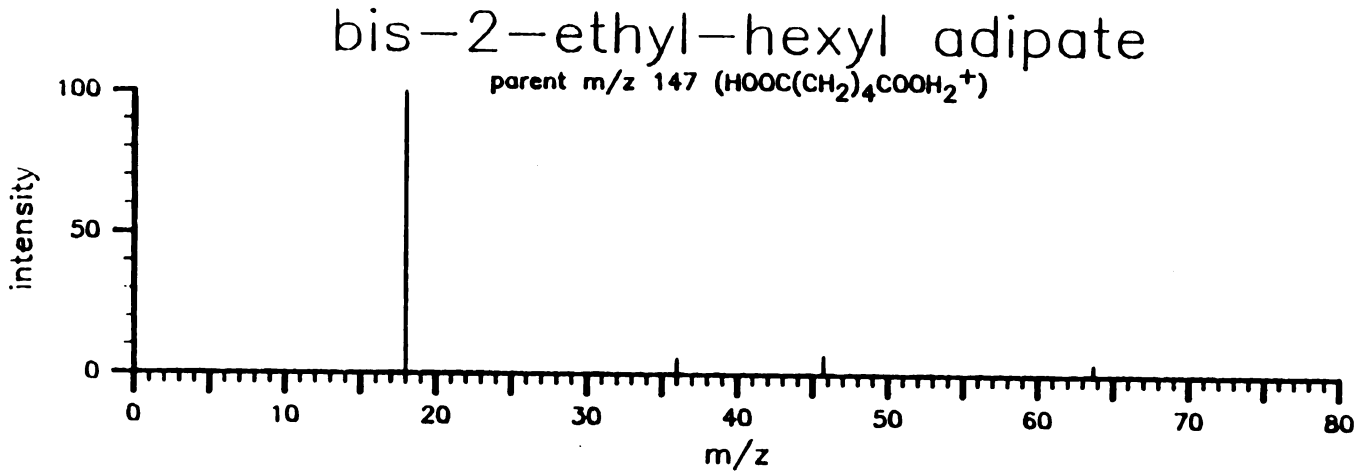
The comments used for ANEUT's output are directly from McLafferty's table which was derived from normal EI spectra. These comments represent the possible conformation of the original molecule and take into account the extensive rearrangements possible in the source. As a



Nom	Mass	%TIC	Formula	Comments
1	1.0	29.0	H	labile H; alkyl cyanides, lower flourides and aldehydes
27	27.0	42.9	H C N	Nitrogen heterocyclic compounds, cyanides
27	27.0	42.9	C2 H3	RCOOR' specific rearrangement loss of (R-2H) or (R'-2H)+(R-H)
28	28.0	28.2	N2	Aryl-N=N-Aryl, >C=N2, cyclic -N=N-
28	28.0	28.2	C O	Aromatic oxygen compounds, cyclic ketones, R—C=-O+
28	28.0	28.2		combination of masses 1 + 27
28	28.0	28.2	C2 H4	RCH2COCH2R' specific rearrangement loss of (R-H) or (R-H)2

Neutral spectrum and analysis of 1,4-benzenediamine

Figure 5.3



Nom	Mass	%TIC	Formula	Comments
18	18.0	85.3	H2 O	Alcohols
18	18.0	85.3	H O H	loss of ROH from R'CH2OR; from R'COOR with labile H
36	36.0+	5.1	H O H H O H	Dialcohols, methoxyalcohols
36	36.0	5.1	H Cl	R==HCL
36	36.0	5.1		combination of masses 18 + 18
46	45.6	5.9	C O H O H	loss of CO + HOR from R'COOR with labile H; cyclic -OCHRO-
46	45.6	5.9	N O2	Nitroaromatics, nitroesters
46	45.6	5.9	C H2 S	Methyl aryl sulfides, some cyclothioalkanes
46	45.6	5.9	C H3 O H C H2	loss of ROH with further loss of CnH2n
46	45.6	5.9	C2 H5 O H	loss of ROH from R'CH2OR; from R'COOR with labile H
64	63.7+	3.7	C H3 O H C H3 O H	Dialcohols, methoxyalcohols
64	63.7	3.7	S O2	RSO2R, Aryl-SO2OR

Neutral spectrum and analysis of bis-2-ethyl-hexyl adipate

Figure 5.4

result, these comments may only be useful for analyzing neutral species lost during the fragmentation of the molecular ion.

Future extensions to this system include constraining the matching process to include or exclude certain atoms. For example, if we know that the compound doesn't include nitrogen, several of correlations will not be displayed, shrinking the list of possible losses. Alternatively, we may know that the unknown contains only carbon, hydrogen and oxygen, effectively eliminating all other elements.

This same program, used with a different knowledge base, could be used to aid in the identification of peaks in a daughter spectrum. This program or similar expert systems can be (and will be) used in an iterative manner with others, including GENOA. The more information that an expert system is given about an unknown, the more certain its analysis of the unknown. For example, output from ANEUT can be used as constraint input into GENOA, limiting the number of structures generated.

Data groupings

The data content of a daughter spectrum is not as great as that in an EI spectrum; however, the information content may be greater if the conclusions are more certain. The

sum of the information contained in all the daughter spectra for one compound could then greatly exceed the information available from the EI spectrum. Many information extracting techniques have been applied to EI spectra to deconvolute them into simple patterns (52-54). Some of these pattern recognition techniques have not fared well, being overwhelmed by the overlapping data present. These same techniques applied to the daughter spectra from a compound don't reveal any new information, because the daughter spectra are already well grouped subsets of the EI spectrum.

A daughter spectrum of the molecular ion bears a great similarity to the EI spectrum, however the daughter spectra from other than the molecular ion are relatively simple. The normal EI spectrum contains fragments from all substructures in the compound, while a daughter spectrum contains fragments from one (or a small number) of substructures of the molecule. If we are able to group sets of these daughter spectra from different compounds, we expect, and find, that the closest matching spectra are derived from compounds with similar substructural features.

MS versus MS/MS spectra matching

The traditional spectrum matching programs designed for matching EI mass spectra take one of several general

approaches to this problem of grouping spectra. Each of these methods relies on a library of spectra which may be abridged in a variety of ways. Biemann searching techniques abridge the spectra to the two most abundant peaks in a 14 dalton window (42). Other techniques reduce the intensity to a binary value, i.e. either there is a peak at a mass or not (40-41). Kevin Cross of MSU has implemented a matching system for EI spectra that is based on the unabridged spectra (20,47). This matching program, patterned after a variety of other matching techniques, assigns weights to the masses and the intensities of each peak, and produces good results for EI spectra.

The simplicity and wide variations in intensities in daughter spectra make normal EI matching techniques less suitable for daughter spectra. Better daughter spectra groupings (and hence substructural feature groups) can be achieved by doing minimal intensity screening for peaks at similar m/z values. I have studied a variety of intensity matching algorithms to determine the importance of the relative intensities of daughters at the same mass. The problem becomes more complicated when you must match a daughter spectrum from a weak parent ion to a spectrum from an intense parent ion. Simply normalizing the spectrum to the parent peak or to the total ion current may result in the noise peaks becoming too prominent. The threshold level for recording a peak on the spectrometer may have

been set too high, and low intensity daughter ions may not be seen. These and other problems require special attention when matching daughter spectra.

Intensity matching for daughter spectra

The approach I took to determining the importance of daughter ion intensity was to try several intensity compression algorithms on daughter spectra from the same parent structure taken under different conditions. Also included in the reference library were vastly different compounds with the same parent ion masses. The first attempt ignored intensities altogether, i.e. a binary compression. If any peak existed at the same mass in both the unknown and reference spectra, it scored one point. On this basis, the top scoring matches proved to be an unselective sample of the data base. This was primarily due to scoring 'matches' of intense daughter peaks with very small peaks, and neglecting to account for missing peaks. This indicated the need for some intensity weighing, and several methods were tried.

Reducing the intensity range from six orders of magnitude available from the instrument to a number representative of the magnitude (i.e. LOG_{10}) gave a total dynamic range of zero to six. Even with this dramatic reduction of scale, and counting intensities that matched

within ±1 unit, the resultant groupings of spectra were not as good as desired. The intensity differences, from similar spectra, were just too great for this 6 level approach.

The current grouping algorithm uses only three levels of intensity information: strong, medium and weak. First, any peaks that are "weak" (less than an arbitrary 800 counts) are marked as being weak. These small peaks must be classified as weak before normalizing the spectrum to identify them as peaks not much greater than the instrument background noise. Next, the daughter spectrum is normalized to the parent ion, and those peaks that are less than one percent are marked weak, between 1 and 10 percent are medium, and greater than 10 percent are strong. Two peaks are considered to have matched intensities if they are in the same or adjacent groups. This algorithm produces good results, allowing for wide intensity variations present in daughter spectra.

Ranking and sorting the data groups

The last step in a grouping program is to have the computer rank the results (Figure 5.5). This involves assigning quality factors to matching peaks, and deducting points for mis-matched spectra. Peaks in an unknown that intensity match peaks in the reference are given full

Intensities reduced to: Weak (< 800 counts or < 1%)
Medium (1% ≤ intensity ≤ 10%)
Strong (10% ≤ intensity ≤ 100%)

mass comparison	intensity comparison	weighting factor
ref = unknown (a mass in both)	intensity matches (within ± one group)	+ 1.0 U
ref = unknown (a mass in both)	intensity doesn't match	+ 0.5 U
mass in unknown not in reference	weak intensity	± 0.0 (ignored)
mass in unknown not in reference	medium, strong intensity	- 0.33 U
mass in reference not in unknown	weak intensity	- 0.05 R
mass in reference not in unknown	medium, strong intensity	- 0.1 R

The weighting factor is based on the number of peaks in either the reference or unknown spectra. R and U each represent the percent of the total spectrum (based on the number of peaks, not their intensities) in the reference and unknown spectra.

Characteristics used to match daughter spectra

Figure 5.5

credit, while those peaks at the same mass but different intensities gain only one half credit. Each "credit" is the percentage of the number of peaks in the unknown. For example, if an unknown has 5 peaks, and 4 peaks match in mass and intensity and one matches mass but not intensity, the rating would be 90% (4*100/5 + 1*50/5). Medium and strong peaks in the unknown that are not in the reference spectrum are negative 1/3 credits. Weak peaks from the unknown, not in the reference, are given a zero credit (i.e. they are ignored). The last of the ranking credits are based on reverse searching (comparing the reference to the unknown). These negative points are based on the number of peaks in the reference spectrum (here one credit is a percentage of the number of peaks in the reference spectrum). Unmatched strong and medium peaks in the reference are a -10% credit, while weak peaks not in the unknown spectrum deduct 5% of a credit.

Examples of daughter spectra grouping

An example of this grouping algorithm is shown for a group of 21 compounds, each containing a parent at m/z 149. In both Figures 5.6 and 5.7, the phthalate ion ($C_6H_4(CO)_2OH^+$) of di-ethyl-phthalate is matched against all the daughter spectra from these 21 compounds (which include other phthalates and a variety of other compounds). The grouping technique correctly clusters the spectra from the

Matching scan 6 (D149.0) from dataset R15444 (diethyl phthalate)

Group factor	Scan	Parent	Dataset
100	6	(D149.0)	R15444 (diethyl phthalate)
97	10	(D149.0)	R27399 (dioctyl phthalate)
87	10	(D149.1)	R20688 (dibutyl phthalate)
66	13	(D120.8)	R27399 (dioctyl phthalate)
64	10	(D149.0)	R22855 (dipentyl phthalate)
63	7	(D121.0)	R8248 (p-t-butylbenzyl alcohol)
58	4	(D149.0)	R8239 (2-t-butyl-6-methyl phenol)
58	5	(D149.0)	R12776 (10-undecenoic acid, methyl ester)
55	7	(D121.1)	R15444 (diethyl phthalate)
54	13	(D121.0)	R20688 (dibutyl phthalate)
53	13	(D120.6)	R22855 (dipentyl phthalate)
50	2	(D164.0)	R8248 (t-butylbenzyl alcohol)
49	5	(D177.1)	R15444 (diethyl phthalate)
35	6	(D 93.0)	R8416 (1,3 benzenedicarboxylic acid)
34	2	(D164.0)	R8239 (2-t-butyl-6-methyl phenol)

Example of the daughter spectrum matching algorithm

Figure 5.6

Matching scan 6 (D149.0) from dataset R15444 (diethyl phthalate)

PT	PC	NC	NS	NR	IS	IR	Scan	Parent	Dataset	
100	100	4	0	0	0	0	6	(D149.0)	R15444	(diethyl phthalate)
68	55	3	1	0	61	0	7	(D121.1)	R15444	(diethyl phthalate)
67	51	2	2	0	25	0	3	(D149.0)	R8416	*1
66	59	3	1	1	61	2	13	(D121.0)	R20688	(dibutyl phthalate)
59	58	3	1	2	61	9	13	(D120.6)	R22855	(dipentyl phthalate)
58	60	3	1	2	61	16	13	(D120.8)	R27399	(dioctyl phthalate)
57	55	3	1	2	61	6	4	(D121.0)	R8416	*1
56	61	3	1	3	8	7	7	(D149.0)	R13923	*2
53	52	2	2	2	25	20	2	(D164.0)	R8239	*3
50	55	4	0	1	0	0	10	(D149.1)	R20688	(dibutyl phthalate)
49	40	2	2	1	70	16	10	(D121.0)	R13923	
49	32	2	2	0	75	0	6	(D 93.0)	R8416	*1
46	59	3	1	3	8	35	5	(D177.1)	R15444	(diethyl phthalate)
45	33	2	2	1	75	6	16	(D 92.8)	R22855	(dipentyl phthalate)
44	34	2	2	1	75	14	9	(D 93.2)	R15444	(diethyl phthalate)

*1 1,3 benzenedicarboxylic acid
*2 tetra ethyl silicic acid
*3 2-t-butyl-6-methyl phenol

PT overall match factor
PC pattern correspondence - intensity based fit
NC number of peaks in common
NS number of peaks in the sample (unknown) not matched to ref.
NR number of peaks in the reference not in the unknown
IS percent total ion current of unmatched sample (unknown)
IR percent total ion current of unmatched reference

Example of the EI spectrum matching algorithm

Figure 5.7

phthalates, and other samples are distributed lower on the scale (Figure 5.6). Figure 5.7 is a comparison of the match factors produced from the EI matching technique. As can be seen from these figures, the EI matching algorithm did not successfully group the phthalates together, while an algorithm designed for matching daughter spectra performed well.

Conclusions

These two techniques extract only a small fragment of the information present in MS/MS data. The neutral spectral studies have shown that the first-order losses from parent ions give direct, definitive information about the composition of the parent ion. The expert system presented in this chapter is a simple program that shows the promise of this technique. The grouping method presented here demonstrates the differences between daughter/neutral spectra and EI spectra. In order to effectively group daughter or neutral spectra, we need to use algorithms with minimal intensity screening.

The technique of MS/MS is a powerful tool for structure determination. The tools presented in this thesis are an introduction to the kind of tools, rules and algorithms needed to elucidate the structure of unknowns. The combined research efforts of many members of the Enke group

have and will continue to lead toward the development and implementation of a variety of tools and techniques useful in determining the structure of unknown samples.

BIBLIOGRAPHY

BIBLIOGRAPHY

1 Yost, R.A., Enke, C.G., J. Am. Chem. Soc., 100, 2274 (1978).

2 Yost, R.A., Enke, C.G., Anal. Chem., 51, 1251A (1979).

3 Yost, R.A., Ph.D. Dissertation, Michigan State University, E. Lansing, MI (1979).

4 McLafferty, F.W. in "Tandem Mass Spectrometry", F.W. McLafferty, Ed., John Wiley & Sons, New York, NY, 1983, Chapter 1.

5 Levsen, K., Beckey, H.D., Org. Mass Spectrom., 9, 570 (1974).

6 Bozorgzadeh, M.H., Morgan, R.P., Benyon, J.H., Analyst, 103, 613 (1978).

7 Yost, R.A., Enke, C.G., in "Tandem Mass Spectrometry", F.W. McLafferty, Ed., John Wiley & Sons, New York, NY, 1983, Chapter 8.

8 Yost, R.A., Fetterolf, D.D., Hass, J.R., Harvan, D.J., Weston, A.F., Skotnick, P.A., Simon, N.M., Anal. Chem., 56, 2223 (1984).

9 Maquestian, A., et. al., in "Tandem Mass Spectrometry", F.W. McLafferty, Ed., John Wiley & Sons, New York, NY, 1983, Chapters 21-26.

10 Bauer, M.R., Masters Thesis, Michigan State University, E. Lansing, MI (1983).

11 Myerholtz, C.A., Ph.D. Dissertation, Michigan State University, E. Lansing, MI (1983).

12 Yost, R.A., Enke, C.G., Org. Mass Spectrom., 16, 171 (1981).

13 Yost, R.A., Enke, C.G., American Lab, June 1981.

14 Levsen, K. in "Tandem Mass Spectrometry", F.W. McLafferty, Ed., John Wiley & Sons, New York, NY, 1983, Chapter 3.

15 Giordani, A.B., Gregg, H.R., Hoffman, P.A., Cross, K.P., Beckner, C.F., Enke, C.G., presented at the 32nd Annual Conference on Mass Spectrometry and Allied Topics; San Antonio, TX, May 27-June 1, 1984.

16 Cross, K.P., Palmer, P.T., Giordani, A.B., Beckner, C.F., Hoffman, P.A., Gregg, H.R., Enke, C.G., ACS Symposium Series, in press.

17 Hoffman, P.A., Enke, C.G., presented at the 31st Annual Conference on Mass Spectrometry and Allied Topics; Boston, MA, May 8-13, 1983.

18 Hoffman, P.A., Ph.D. Dissertation, Michigan State University, E. Lansing, MI, (in preperation).

19 Cross, K.P., Enke, C.G., presented at the 32nd Annual Conference on Mass Spectrometry and Allied Topics; San Antonio, TX, May 27-June 1, 1984.

20 Cross, K.P., Ph.D. Dissertation, Michigan State University, E. Lansing, MI (1985).

21 Lindsay, R.K., et. al., "Applications of Artificial Intelligence to Organic Chemistry: The Dendral Project", McGraw Hill, New York, NY (1980).

22 Carhart, R.E., Smith, D.H., Gray, N.A.B, Nourse, J.G., Djerassi, C., J. Org. Chem., 46, 1708 (1981).

23 Carhart, R.E., Varkony, T.H., Smith, D.H., ACS Symposium Series 54, 126 (1977).

24 Crawford, R.W., Brand, H.R., Wong, C.W., Gregg, H.R., Hoffman, P.A., Enke, C.G., presented at the 30th Annual Conference on Mass Spectrometry and Allied Topics; Honolulu, HI, June 6-11, 1982.

25 Crawford, R.W., Brand, H.R., Wong, C.W., Gregg, H.R., Hoffman, P.A., Enke, C.G., Anal. Chem., 56, 1121 (1984).

26 Newcome, B.H., Ph.D. Dissertation, Michigan State University, E. Lansing, MI, (1984).

27 Hoffman, P.A., Private Communication.

28 Newcome, B.H., Enke, C.G., Rev. Sci. Instr., 55, 2017 (1984).

29 Brodie, L., "Starting FORTH, An Introduction to the FORTH Language and Operating System for Beginners and Professionals", Prentice-Hall, Englewood Cliffs, NJ (1981).

30 Denton, M.B., Private Communication.

31 Tilden, S.B., Denton, M.B., Technical Report #18, for ONR contract #N00014-75-C-0513 (1979).

32 Carlson, E., Ph.D. Dissertation, Michigan State University, E. Lansing, MI (1978).

33 Gregg, H.R., Chai, J.W., Chakel, J.A., Hoffman, P.A., Latven, R.K., Matthews, R.S., Myerholtz, C.A., Newcome, B.H., Enke, C.G., presented at the 29th Annual Conference on Mass Spectrometry and Allied Topics; Minneapolis, MN, May 24-29, 1981.

34 Schubert, A.J., Ph.D. Dissertation, Michigan State University, E. Lansing, MI (in preperation).

35 Myerholtz, C.A., Newcome, B.H., Enke, C.G., presented at the 31st Annual Conference on Mass Spectrometry and Allied Topics; Boston, MA, May 8-13, 1983.

36 Kristo, M.J., Enke, C.G., presented at Rochester FORTH Conference, June 1985.

37 Kristo, M.J., Myerholtz, C.A., Schubert, A.J., Enke, C.G., presented at Rochester FORTH Conference, June 1985.

38 Myerholtz, C.A., Schubert, A.J., Kristo, M.J., Enke, C.G., accepted for publication in Instruments and Chemistry.

39 Myerholtz, C.A., Schubert, A.J., Kristo, M.J., Enke, C.G., accepted for publication in Instruments and Chemistry.

40 Grotch, S.L., Anal. Chem., 42, 1214 (1970)

41 Wangen, L.E., Woodward, W.S., Isenhour, T.L., Anal. Chem., 43, 1605 (1971).

42 Hertz, H.S., Hites, R.A., Biemann, K., Anal. Chem., 43, 681 (1971).

43 Naegli, P.R., Clerc, J.T., Anal. Chem., 46, 739A (1974).

44 Gronneberg, T.O., Gray, N.A.B., Eglinton, G., Anal. Chem., 47, 415 (1975).

45 Pesyna, G.M., Venkataraghavan, R., Dayringer, H.E., McLafferty, F.W., Anal. Chem., 48, 1362 (1976).

46 Blaisdel, B.E., Anal. Chem., 49, 180 (1977).

47 Damen, H., Henneberg, D., Wiemann, B., Anal. Chim. Acta, 103, 289 (1978).

48 Van Marlen, G., Van Den Hende, J.H., Anal. Chim. Acta, 112, 143 (1979).

49 Lebedev, K.S., Tormyshev, V.M., Derendyaev, B.G., Koptyug, V.A., Anal. Chim. Acta, 133, 517 (1981).

50 Jurs, P.C., Kowalski, B.R., Isenhour, T.L., Anal. Chem., 41, 21 (1969).

51 Jurs, P.C., Kowalski, B.R., Isenhour, T.L., Rielley, C.N., Anal. Chem., 41, 1949 (1969).

52 Bender, C.F., Shepherd, H.D., Kowalski, B.R., Anal. Chem., 45, 617 (1973).

53 Lam, T.F., Wilkins, C.L., Brunner, T.R., Saltberg, L.J., Kaberline, S.L., Anal. Chem., 48, 1768 (1976).

54 Ritter, G.L., Isenhour, T.L., Computers Chem., 1, 243 (1977).

55 Lowry, S.R., Isenhour, T.L., Justice, J.B., McLafferty, F.W., Dayringer, H.E., Venkataraghavan, R., Anal. Chem., 49, 1720 (1977).

56 Buchs, A., Duffield, A.M., Schroll, G., Djerassi, C., Delfino, A.B., Buchanan, B.G., Sutherland, G.L., Feigenbaum, E.A., Lederberg, J., J. Am. Chem. Soc., 92, 6831 (1970).

57 Delfino, A.B., Buchs, A.B., Helv. Chim. Acta, 55, 2017 (1972).

58 Buchanan, B.G., Smith, D.H., White, W.C., Gritter, R.J., Feigenbaum, E.A., Lederberg, J, Djerassi, C., J. Am. Chem. Soc., 98, 6168 (1976).

59 McLafferty, F.W., Anal. Chem., 49, 1441 (1977).

60 Cardenas, A.F., Data Base Management Systems, 2nd ed., Allyn and Bacon, Inc., Boston, MA (1985).

61 Gillenson, M.L. Database, Wiley-Interscience, New York, NY (1985).

62 Wiederhold, G., Database Design, McGraw-Hill, New York, NY (1977).

63 Borman, S.A. Anal. Chem, 57, 983A (1985).

64 Hampel, V.E. in Data for Science and Technology, Glaesen, P.S. (ed.), Pergamon Press, Oxford, (1981).

65 Chen, C.C. and Hernon, P. (ed.) Numeric Databases, Ablex Publishing, Norwood, NJ (1984).

66 Glaeser, P.S. (ed.) Data for Science and Technology, Elsevier Science Publishers, Amsterdam, The Netherlands (1984).

67 Glaeser, P.S. (ed.) Data for Science and Technology, North-Holland Publishing, Amsterdam, The Netherlands (1983).

68 Shoshani, A., Olken, F. and Wong, H., pg. 349 in The Role of Data in Scientific Programs, Glaeser, P.S. (ed.) Amsterdam, The Netherlands (1985).

69 Sobel, y., Dagane, I., Carabedian, M., and Dubois, J., pg. 395 in The Role of Data in Scientific Programs, Glaeser, P.S. (ed.) Amsterdam, The Netherlands (1985).

70 Smith, F.J. and Hughes, J.G., pg. 435 in The Role of Data in Scientific Programs, Glaeser, P.S. (ed.) Amsterdam, The Netherlands (1985).

71 Rumlble, J.R. Jr. and Hampel, V.E. (eds.), Database Management in Science and Technology, Elsevier Science Publishers Amsterdam, The Netherlands (1984).

72 Rumlble, J.R. Jr. in Database Management in Science and Technology, Rumlble, J.R. Jr. and Hampel, V.E. (eds.), Elsevier Science Publishers Amsterdam, The Netherlands (1984).

73 Kipiniak, W. and Finnerty, W. pg 17 in Computers in the Laboratory, Liscouuski, J.G. (ed.), ACS Symp. Series 265 (1984).

74 Baumann, F., Lewis, K.A., and Brown, A.C. III, pg 23 in Computers in the Laboratory, Liscouuski, J.G. (ed.), ACS Symp. Series 265 (1984).

75 RS-1, BBN Software Products.

76 Lotus 1-2-3, Lotus Development Corp.

77 Gregg, H.R., Enke, C.G., in preperation for submission to Computers and Chemistry.

78 VAX FORTRAN, Digital Equipment Corp.

79 Enke, C.G., Science, 215, 785 (1982).

80 Perone, S.P., ACS Symp. Series, 265, 99 (1984).

81 Atkinson, T.V., Gregg, H.R., "MULPLT: A Multiple Dataset, File Based Plotting Program", submitted to the DECUS software library, Marlboro, MA (1984).

82 Hayes-Roth, F., Waterman, D.A., Lenat, D.B., "Building Expert Systems", Addison-Wesley, Reading, MA (1983).

83 Hayes-Roth, F., Computer, 17, 263 (1984)

84 Barr, A., Feigenbaum, E.A., "Handbood of Artificial Intelligence", William Kaufman, Los Altos, CA, Vol. 1 (1981), Vol. 2 (1982).

85 Jurs, P.C., Computer Software Applications in Chemistry, Wiley-Interscience, New York, NY (1986).

86 Rames, L.S., et. al., Anal. Chem., 58, 295R (1986).

87 Martinsen, D.P. and Song, B.H., Mass Spectrom. Rev., 4, 461 (1985).

88 Barr, A. and Feigenbaum, E.A., The Handbook of Artificial Intelligence Vol. 2, William Kaufmann, Los Altos, CA (1982).

89 Buchanan, B.G. and Feigenbaum, E.A., Artificial Intelligence, 11, 5 (1978).

90 Dessay, R., Anal. Chem., 56, 1200A (1984).

91 Dessay, R., Anal. Chem., 56, 1312A (1984).

92 Addis, T.R., Designing Knowledge-Based Systems, Kogan Page Ltd, London (1985).

93 Pierce, T.H. and Hohne, B.A. (eds.), Artificial Intelligence in Chemistry, ACS Symp. Series, 306 (1986).

94 Mclafferty, F.W. and Stauffer, D.B., J. Chem. Inf. Comp. Sci., 25, 245 (1985).

95 Wong, C.M., Crawford, R.W., Kunz, J.C., Kehler, T.P., IEEE Transactions on Nuclear Sciences, NS-31, 804 (1984).

96 Wong, C.M., Crawford, R.W., Lanning, S.M., Brand, H.R., presented at the 32nd Annual Conference on Mass Spectrometry and Allied Topics; San Antonio, TX, May 27-June 1, 1984.

97 Wong, C.M., Lanning, S.M., Crawford, R.W., Brand, H.R., presented at the 32nd Annual Conference on Mass Spectrometry and Allied Topics; San Antonio, TX, May 27-June 1, 1984.

98 Brand, H.R., Lanning, S.M., Wong, C.M., submitted to Int. Joint Conf. on Artificial Intelligence, August 18-24, 1985.

99 McLafferty, F.W., "Interpretation of Mass Spectra, 3rd Ed.", University Science Books, Mill Valley, CA (1980).

APPENDICES

Appendix 1

Dr. Memory, SLOPS and control software

This appendix contains brief descriptions of each of the commands and subroutines used in the TQMS control software. For the purpose of brevity, the complete user's manuals for each of the three software packages (Dr. Memory, SLOPS, and the control software) will not be reproduced. Each command or subroutine in each user manual consisted two or more pages. A sample of the format used is presented (for the subroutine that checked with the user, then set/reset specified bits, BITCHK), followed by only the description section of the documentation for the rest of the subroutines.

Appendix 1,	part 1:	Bit check example
	part 2:	Dr. Memory
	part 3:	SLOPS
	part 4:	TQMS control software

Appendix 1, Part 1

BITCHK

Description

This routine, bit check, sends out a message, gets a response, and clears, sets or leaves alone the indicated bit in the indicated byte. A YES response sets the bit, NO clears is, and nothing (<CR>) does not change it.

Version

hg 1.0

Required

B = bit position (mask)
DE = pointer to the byte to change
HL = pointer to query text

Returns

none

Modified

none

Pushes/Pops

0/0 - register A modified

Calls

PRINT
TTYIN

Bugs

```
        SUBR    BITCHK
BITCHK: CALL    PRINT           ; ask the question
        CALL    TTYIN           ; get the answer
        LDA     CHRBUF          ; get the first character
        CPI     'Y              ; yes
        JZ      BITCH2          ;
        CPI     'y              ;
        JZ      BITCH2          ;
        CPI     'N              ; or no
        JZ      BITCH1          ;
        CPI     'n              ;
        RNZ                     ; ignore ...
BITCH1: MOV     A,B             ; get
        CMA                     ; and reverse the mask
        MOV     B,A             ;
        LDAX    D               ; clear the bit
        ANA     B               ;
        STAX    D               ;
        RET                     ; and head home
BITCH2: LDAX    D               ; set the given bit
        ORA     B               ;
        STAX    D               ;
        RET                     ; and head home
```

Appendix 1, Part 2

Dr. Memory summary

Globals	Definitions, memory locations, etc
Lowcore	Restart and trap jumps
Interupts	Interupt handling routines
RST0	Cold start, sets defaults
RST1	A \
RST2	BC \ Diagnostics - prints
RST3	DE \ the contents of the
RST4	HL / indicated registers
RST5	PC /
RST6	SP+flags/
RST7	Software breakpoint
Kernal	Dispatcher for the monitor
Commands	Processes several commands
Go	Go and Procede commands
Quest	Prints a summary of the commands
Regstr	Open a register for modification
Talk	Connect terminal to PDP 11

Subroutines

Chrchk	Checks and converts ASCII to binary*
Close	Closes an open location*
Crlf	Outputs a <CR><LF> combination
Downld	Downloads data from the PDP 11
Efclr	Clears an event flag
Getnum	Gets a number input
Gettt	Gets a character from the USARTs
Lights	Sends a predefined light pattern out
Lite	Sends a specified light pattern out
Modify	Modify a location in memory*
Nulljb	Boredom routine
Open	Opens a memory locations*
Print	Prints an ASCII string
Putnum	Outputs a number
Puttt	Writes a character to a USART
Rhlr	Rotates HL right
Stkbit	Stacks 3/4 bits of HL on stack*

* these routines are useful only to Dr. Memory, and probably will not help the casual programmer.

GLOBALS This is a list of mnemonics that assign values to ASCII characters, memory locations, etc.

LOWCORE These are the hardware vector jumps. This routine just transfer control to the processing routines. The restart interrupts are transferred to RAM where RST 7's await (unless user modified).

INTERRUPTS These routines process interrupts from devices. The USART interrupt routines fetch and store one character. The break interrupt routine does a warm restart.

RST0 This cold start routine initializes several parameters needed for the monitor to operate. Breakpoint instructions are placed in RAM where unused interrupts transfer control, jump tables set up, flags set to default values, default user stack set up as well as a default program counter.

RST1 Debugging aid: prints the contents of register A

RST2 Debugging aid: prints contents of register pair BC

RST3 Debugging aid: prints contents of register pair DE

RST4 Debugging aid: prints contents of register pair HL

RST5 Debugging aid: prints the user's program counter

RST6 Debugging aid: prints the value of the stack pointer and flags

RST7 Warm start or breakpoint entry. This routine saves all of the user registers and status. Control is returned to either Dr. Memory or Slops, depending on who is in control.

KERNAL This is the heart of Dr. Memory. It accepts numbers (addresses and values) and commands and dispatches to the appropriate routine.

COMMANDS These are routines that process Dr. Memory's commands. Those processed in this module are slash, backslash, carriage return, line feed, up arrow, Slops, Hex, and Octal.

GO Two of Dr. Memory's commands live here: Go and Procede. Procede is useful - it restores all user status and registers, including program counter. This gets us back to 'user mode'.

QUEST This is the ? command of Dr. Memory. It prints a summary help message.

REGSTR This is another of Dr. Memory's commands, the $. It is used to translate a register mnemonic to an address, and open one or two bytes for modification.

TALK This is a routine that 'connects' the two USARTs together. Used to communicate to the PDP 11 thru the micro. Note that there is no exit to this routine. As such, it is both a command and a subroutine. A break is needed to exit.

CHRCHK Character check. Numbers (actually digits, in register A) are converted to binary (numbers either octal or hex) and the result is left in register A. Non-numbers cause the carry bit to be set.

CLOSE Sets the open/closed flag to closed, and write out a carriage

CRLF This routine sends a carriage return/line feed combination to the terminal.

DOWNLD This subroutine talks to the PDP 11, and expects a defined protocol for loading the micro's memory. Binary records are loaded directly into memory, while ASCII records are sent to the terminal.

EFCLR This routine clears the event flag specified in register A.

GETNUM This routine gets characters from the keyboard, echoes them, checks them, and if numeric, builds a number and loops for more. The number, if entered, is built in HL and the 'entered number' flag is set. This routine returns when a non-numeric is entered.

GETTT These are the get character routines. GETTT0 gets a character from USART 0, while GETTT1 gets a character from USART 1. Each routine waits until a character is entered. The character is returned in register A.

LIGHTS This routine displays a characteristic light pattern each time called. The pattern displayed is stored at location LITES.

LITE This routine turns lights on/off as defined by a mask in register A. Each of the low 4 bits specify a light as follows:

0 ==> bottom PS light
1 ==> middle PS light
2 ==> top PS light
3 ==> SOD light

MODIFY This routine will modify a memory location if there is an open location and a number has been entered.

NULLJB This routine waits for the occurence of specified event flags. Register A contains a mask of the flags to wait for. While waiting, this routine plays with the lights.

OPEN These routines set the open location flag and writes the contents of the open byte(s) to the terminal. OPEN opens a one byte location, OPEN2 opens a two byte location.

PRINT This subroutine prints a string of ASCII characters which are ended by a null (0). HL point to the start of the string.

PUTNUM This routine writes out a number, one or two bytes long, octal or hex, to the terminal. HL contain the number to output, and B contains a flag byte: 1 for byte output, 0 for word output.

PUTTT These routines output a byte to a USART. PUTTT0 sends a byte to USART 0, while PUTTT1 send it to USART 1. The character to send is in register A. These routines wait until the USART is ready to send.

RHLR This routine rotates register pair HL right one bit. The most significant bit is always set to 0, and the least significant bit is returned in the carry flag.

STKBIT This routine takes the least significant 3 or 4 bits, depending on the octal/hex flag, of HL and pushes them on the stack under the return address.

Appendix 1, Part 3

SLOPS summary

Globals

Globals	Mnemonics, address, etc for Dr. Memory
Sglobals	Globals for Slops
Init	Initialization routine
Kernal	Driver routine and dispatcher

Subroutines

Addr	Returns code address of library entry
Ascii	Converts binary to and from ASCII
Blank	Clears the screen
Brkdwn	Breaks input line into words
Check	Compares word with library entry
Cvtext	Converts a number to ASCII string
Cvtint	Converts ASCII string to binary number
Dcmp	Double compare
Delay	Software time delay
Link	Links to next library entry
Number	Get number from word stack
Search	Search library for a match
Ttyin	Gets a line of input
Word	Get word from word stack

(also see subroutines listed under Dr. Memory)

Arithmetic subroutines

Ddiv	Double divide
Div	Divide
Dmult	Double multiply
Dsub	Double subtract
Mult	Multiply

Commands

Convert	Converts a number to any base
Downld	Loads from the PDP 11 (See Dr. Memory)
Drmem	Puts Dr. Memory in control
Talk	Talk to the PDP 11

GLOBALS This is a list of mnemonics, address assignments, etc. for Dr. Memory. Needed for proper assembly.

SGLOBALS Slops globals. Mnemonics for Slops use, definition of memory locations and subroutine addresses.

INIT This routine initializes a few parameters, and sets Dr. Memory to look directly at Slops, not here.

KERNAL This is the heart of Slops. It gets a command line, searches the library, and if the command if found, starts the named routine.

ADDR This routine returns a pointer to the code of a library subroutine entry, given a pointer to the entry. e.g. the pointer is incremented past the flag byte, the name and the link field, if present.

ASCII These are two routines, TOASCI and UNASCI. TOASCI adds either 48 or 55 to the number in register A to make it either numeric or alpha, as needed. UNASCI checks the character in register A to see if it is a legal digit, with radix given in register C. If legal, the number is converted to binary. Carry flag is set otherwise.

BLANK Blanks the screen and resets the cursor to the 'home' position.

BRKDWN This utility routine breaks a line of input (assumed to be in the terminal buffer) down into 'words'. A word is defined as alphanumeric characters delimited by non-alpha characters. These non-alpha characters are also words of length 1. Spaces are defined as non-alpha delimiters, but do not count as words. Text after a semicolon is treated as a comment, and not broken down into words.

CHECK Compares the current word (eg an entered 'command') with an entry in the library. Sign flag set if the words don't match.

CVTEXT Converts a number (DE) to an ASCII string (radix (C)). The resultant string is 'pushed' to an area pointed to by HL. The end of the string is flagged by a null (0). Note that the string built is built backwards, so on entry, HL must point to the TOP of a buffer. On return, carry set implies a bad radix was given.

CVTINT This routine converts an ASCII string (radix (C)) to a number (DE). The carry flag will be set on return if a bad character was discovered. On return, carry set implies a conversion error (bad character in string).

DCMP Double compare, patterned after the 8085 CMP instructions. Here, flags (S, CY, Z) set according to DE - HL.

DDIV Double divide. The following takes place:
BC = HLBC / DE, remainder in HL

DELAY Software timing loop generate approximate time delays. HL contain the count, each count is about 10 microseconds. Note that HL=1 is the shortest time, and HL=0 is the longest.

DIV Divide routine. This acts as follows:
DE = DE / A, remainder in A

DMULT Double multiply. This routine acts as follows:
DE = DE * A

DRMEM This routine sets Dr. Memory as the kernel in charge, and returns to it.

DSUB Double subtract. This routine acts as follows:
HL = HL - DE
This is the counterpart to the 8085 instr DAD D

LINK This routine links from one library entry to the next. e.g. BC pointing to a library entry is redirected to point to the next entry. The sign flag is set if no more entries exist.

MULT Multiplication. This routine acts as follows:
DE = D * E

NUMBER This routine tries to find a number as the next element on the word stack. If found, conversion to binary is attempted. The conversion radix may be specified with the number or defaulted. Radix qualifiers follow the number, as follows:

```
'   ==>  octal
.   ==>  decimal
"   ==>  hexadecimal
:n  ==>  radix n, where n is any valid
         number.
```

The carry flag tells of errors, which might include any of the following: no next word, next word not a number, or the radix qualifier (number after :) is not a number.

SEARCH This routine searches through the library for a match with the current word. If a match is found, the entry is passed back along with its flag byte. The sign flag is used as an error indicator for no match.

TTYIN Keyboard driver. Accepts a line of input, ended by a control character (ASCII value < 32), stores the line in the input buffer and echoes each character as typed. This routine also processes the following special characters:

delete	deletes the last character
escape	blanks the screen
cntl/U	deletes the current line of input

WORD Pops pointer to and length of next word off the word stack. If no more entries exist, the sign flag is set.

CONVERT This command converts a number from one base to another. To invoke this routine, type:

```
CONVERT number base
```

where 'number' is the number to be converted to the base 'base'. For a description of valid numbers, see NUMBER.

Appendix 1, Part 4

Control system summary

Commands

INI	Initialize the system
GET	This routine gets a set of stored parameters
SAVE	This routine save the current parameter table
PARAM	This is the general parameter setting routine
MANUAL	Routine for 'manual' control
FSCAN	Fast scan - see the spectra on the oscilloscope
SCAN	Scan, collect data and display it
WDATA	This routine writes the data from RAM to a disk file

Graphics subroutines

CHRLIB	Character library
CLEAR	This subroutine/command clears the graphics screen
DRAW	Etch-a-sketch command
GRFAXS	This is the axis drawing routine
GRFCHR	This subroutine draws a character on the screen
GRFLAB	This routine draws a string on the graphics screen
GRFVEC	Draws a vector on the graphics screen

Display update routines

UPDATE	This routine updates the status and graphic displays
UPSTAT	This routine updates the status display
UPSNUM	This routine updates numbers on the status display
UPINT	Format and display the intensity on status display
UPMASS	Format and display the mass on the status display
UPGRAF	This routine updates the graphic display
UPAXES	This routine draws the axes
DLAST	Draws the last collected intensity

Other subroutines

BITCHK	Sets/clears bits in a flag based on user response
FMASS	Formats the mass as follows: aa:xxx.x
FORMAT	Format a number for output
GETGRF	This routine returns a flag byte with graph info.
GETMAS	This routine gets a mass from the word stack
GETPM	Get X coordinate for current mass for display
GETPRM	Get a new set of parameters
INCMAS	Increment the quads to the next mass
INTENS	This routine collects the data
INT16	This routine returns the sum of 16 intensities
MSADC	Gets one ADC value
MSDATA	Store current intensity in memory
MSINIT	Initialize the control hardware
MSMASS	Send the quad controller a mass via DACs
PVRAM	Put text to the status display video RAM
SMASS	Set current mass
SMGL	Send message to user, get line of response
SRANGE	Set range of Kiethley amplifier
STARTP	This is a file of the startup (default) parameters

Commands

INI This is a routine to clean things (mainly the display) up. The data RAM is zeroed, graphics and status screens are cleared and updated, range and mass reset.

GET This routine gets a set of parameters from a stored bank. The number of the bank where the parameters are stored is requested, and the entire bank is transferred into the current parameter table.

SAVE This routine save the current parameter table in a bank of parameter tables, for future recall. Use is as follows:

SAVE n

where n is a number between 1 and 4 inclusive.

PARAM This is the general parameter setting routine. It acts in a 'single character input' mode for the parameter to change, i.e. 'Q' not 'QUAD' for changing a Quad's mass range. At this time, little checking is done of the parameters, and each parameter must be set individually, i.e. changing the mass range does not change the graph parameters.

The parameters for either graph, the UPPER or the LOWER, must be changed individually, and the commands U and L identify which graph is being changed.

The following commands are currently supported:

Q n low-high	Change Quad n's mass scan
T int range	Change the threshold
R min-max	Change the min and max range
A n	Change the # points to average
G low,n	Change low mass on the graph
F	Change flags, each queried
M text	Puts message on status display
H text	Puts a header on graph
?	Prints a summary of the options
^Z	to exit (control Z)

MANUAL Routine for 'manual' control. Using the keypad of the terminal, the keys 1, 2 and 3 are for quad 1; 4, 5 and 6 are for quad 3; and keys 7, 8 and 9 are for the range. Keys 1, 4 and 7 decrease the current value by 1, keys 2, 5 and 8 set the current value into the parameter table, and keys 3, 6 and 9 increase the current value by 1.

FSCAN This routine increments the mass, checks for a typed character, and if none entered, loops about.

SCAN This routine scans the set mass ranges, collects data, updates the screens, records the data and loops until a character is typed.

WDATA This routine writes the data from RAM to a disk file.

Graphics subroutines

CHRLIB This is not a subroutine, but a library of vector moves that draws the ASCII character set on the MATROX graphics board. A list of pointers, in ASCII order, points to the entries for each character.

CLEAR This subroutine/command clears the graphics screen. The actual clearing takes a maximum of 34 msec, but this routine does not wait.

DRAW This command allows the user to draw, dot by dot, on the MATROX graphics board. The numeric keypad is used to determine the direction, 5 ==> don't move, 9==> northwest, 4 ==> east, etc. The 0 key tells the routine to clear dots in the drawn path, the period (.) tells the routine to draw points. To exit, type a slash (/).

GRFAXS This is the axis drawing routine. It will draw an axis either vertical or horizontal. Large and small tick marks can also be drawn, and large tick marks can be labeled. This routine works from a table of data the defines the axis to be drawn. The table (an axis description block) is defined below.

GRFCHR This subroutine draws a character on the graphics screen. The ASCII code for a character is passed in register A, the graphics board must be aware of the address to plot at (lower left corner of the character space), and this routine will draw the character from the library (CHRLIB).

GRFLAB This routine draws a string of characters on the graphics board. The supported characters are all the printing ASCII characters. The string can be left, center or right justified. Registers D and E contain the coordinates of the lower left (center or right) for the string to be drawn. HL points to the string, terminated by a null.

GRFVEC Draws a vector, of length (C), in the direction (B) [Note that the direction mnemonics (north, south, east, west, ne, se, nw, sw) should be used], starting at physical coordinates X and Y (D,E).

Display update routines

UPDATE This routine updates the status and graphic displays.

UPSTAT This routine updates the status display. Individual parts of the display can be updated, or the entire screen can be blanked and redrawn.

UPSNUM This routine updates the numbers on the status display for either the upper or the lower scan. Not updated are the descriptions, the current info, or the messages; just the numbers under the "UPPER" or "LOWER" headers.

UPINT This routine scales the given intensity and range to the format xx.x -xx and sends it to the status display, at location pointed to by HL.

UPMASS This routine updates a mass range on the status display, in the following format: xxx-xxx

UPGRAF This routine updates the graphic display. The entire graph may be updated, or only the current information can be plotted.

UPAXES This routine draws the axes, taking into account single or double plot, linear or log scale, low mass for display, and # pts/AMU.

DLAST This routine updates the last mass, and the current intensity. This is done to allow time for the quads to settle, ions to be selected, etc. before the next intensity is collected. While waiting, graphing the last displayed point seemed like the thing to do.

Other subroutines

BITCHK This routine sends out a message, gets a response, and clears, sets or leaves alone the indicated bit in the indicated byte. A YES response sets the bit, NO clears is, and nothing (<CR>) does not change it.

FMASS This routine formats the given mass as follows: aa:xxx.x, where aa is either RF or DC, and xxx.x is the mass.

FORMAT This routine converts a number from registers DE into ASCII in a predefined format, into a buffer pointed to by HL. This routine converts numbers to decimal, adds the required number of trailing zeros, inserts a decimal point, and pads the from of the string with spaces. The user than calls PRINT to output the formatted string.

GETGRF This routine returns a flag byte with graph information.

GETMAS This routine attempts to get a valid mass from the word stack. If a number was entered, it is scaled for the DAC's in use (MASS = MASS * 8.). If no number given, the carry flag is set.

GETPM This routine returns the X-coordinate on the graphics display corresponding to the current mass. Also returned are some of the graph parameters.

GETPRM This routine changes the entire parameter table for Mass Spec control. A table, set up exactly like the parameter table, is pointed to by HL, and its contents are moved to the current or active parameter table.

INCMAS This routine checks the parameter table, finds which quads want their mass incremented, and increments them. Checks are made if any of the masses increment past the end of their allowed scan, and if so, the masses are reset.

INTENS This routine collects the data. A small sample of intensities (16) are collected, and this set is used to determine if autoranging is required. If so, the Keithley is ranged, and time is allowed for it to settle. This is repeated until the signal is in range, or no more auto-ranging is possible. The signal is then sampled and averaged the required number of times.

INT16 This routine returns the sum of 16 intensities. It is intended to give an idea of the current ion intensity.

MSADC This is the lowest level data acquisition routine. It requests and receives a datum from the analog to digital converter.

MSDATA This routine stores the current intensity in the data RAM, at a location corresponding to the DAC value (mass).

MSINIT This routine sets up the parallel ports used for the DACs, ADC. It also gets the default parameters, sets masses to reasonable values, and cleans up the displays.

MSMASS This is the lowest level routine to send a mass to the DACs controlling the quad's mass.

PVRAM This routine prints the text pointed to by BC to the VRAM pointed to by HL, offset by DE. The display is written into during vertical or horizontal flyback, minimizing flicker.

SMASS This routine gets the "current masses" from the parameter table, and sends them to the DACs.

SMGL This routine sends a message (pointed to by HL), and gets the response from the user. The message is broken down into words, and an attempt to get a number is done.

SRANGE This routine sends a new range to the Keithley, and delays a bit to allow the amplifier to settle and become reasonably stable.

STARTP This is a file of the startup (default) parameters. On startup, this file is copied into the parameter table.

Appendix 2

Multi-dimensional data base subroutines

This appendix contains descriptions of the subroutines used to interface with the multi-dimensional data base format.

Subroutines to initialize the datasets

```
      SUBROUTINE MSINIT
C
C This subroutine (along with the PARAMETER statements in
C MDDB.CMN) set up the characteristics of this system.
C All possible datasets are initialized, and system-wide
C globals are set up, and the dictionary file is opened.
C
C calls     CONCAT (STRING)
C           SCOPY  (STRING)
C           DSINIT (MDDB)
C           EXIT   (RSX)
C
C                                          h gregg jul-82
C
C-------------------------------------------------------------
C

      SUBROUTINE MSOPEN(IFILE,QPRMPT,QTYPE,LOON1,LOON2,LOON3)
      BYTE        QPRMPT(1),QTYPE(1)
      INTEGER*2   IFILE,LOON1,LOON2,LOON3
C
C This subroutine open (close) the files associated with a
C dataset.  New or Old files may be opened, and filenames
C can be prompted or defaulted.
C
C where     IFILE  the number of the data set to open
C                  negative means close that dataset.
C           QPRMPT prompt for asking for a dataset name
C                  if LENGTH(QPRMPT) = 0, use QDSFIL as
C                  the dataset name (in common).
C           QTYPE  is either 'NEW', 'OLD' or 'READONLY'
C           LOONn  are the logical units to use for the
C                  dataset
C
C returns   (if 'OLD', see GETPRM,GETCOM)
C
C calls     CONCAT (STRING)
C           LENGTH (STRING)
C           GETCOM (MDDB)
C           GETPRM (MDDB)
C
C                                          h gregg jul-82
C
C-------------------------------------------------------------
C
```

Subroutines to write into a dataset

```
      SUBROUTINE PUTPRM(IFILE)
      INTEGER*2  IFILE
C
C This subroutine writes parameters to the header file.
C with the current time and date.
C
C where     IFILE  is the dataset number
C
C uses      NUMSTC the number of static variables
C           ISTATC code for the static parameters
C           RSTATC value of corresponding parameter x
C           NUMVAR the number of static variables
C           IVAR   code for the variable parameters
C
C calls     DATE   (RSX)
C           TIME   (RSX)
C
C                                        h gregg jul-82
C
C-------------------------------------------------------
C

      SUBROUTINE PUTCOM(IFILE)
      INTEGER*2  IFILE
C
C This subroutine writes a comment (title, words of wisdom,
C etc.) to the header file, including the current date and
C time, if needed.
C
C
C where     IFILE  is the dataset number
C
C uses      QDATE  is the date to write out.
C                  If LENGTH(QDATE) = 0, get current date.
C           QTIME  is the time to write out.
C                  If LENGTH(QTIME) = 0, get current time.
C           QCOMNT an array holding the comment
C                  The comment is assumed to end with a null
C                  and be 80 bytes or less long.
C
C calls     DATE   (RSX)
C           TIME   (RSX)
C           LENGTH (STRING)
C
C                                        h gregg jul-82
C
C-------------------------------------------------------
C
```

```
      SUBROUTINE PUTDAT(IFILE,X,Y)
      INTEGER*2  IFILE
      REAL*4     X(1),Y(1)
C
C This subroutine puts a scan of data into the pointer and
C data files.  All pointers are kept internally, and some
C values are calculated.
C
C where    IFILE  is the number of the dataset to use
C                 X,Y are the X,Y data pairs
C
C uses     NUMDAT number of data points
C          IVARD  code for X of X,Y pair
C          RTIME  time the scan was taken
C          RVAR   values of the variables
C          IVARF  codes for fast variables
C          RVARF* values of fast variables, from RVAR.
C          RSUMY* sum of Y values, done here
C          ISCAN* + 1 is where this scan will go
C
C * ==> calculated or kept internally, do not change!
C
C calls    PTDATM (MDDB)
C          PUTPTR (MDDB)
C
C                                        h gregg jul-82
C
C-------------------------------------------------------------
C
```

Subroutines to read from a dataset

```
      SUBROUTINE GETCOM(IFILE)
      INTEGER*2  IFILE
C
C This subroutine gets the next comment in the header file,
C along with its time and date of entry.
C
C where    IFILE  is the dataset number
C
C returns  QDATE  the date of the comment
C          QTIME  the time of the comment
C          QCOMNT the comment (or title) itself
C
C calls    none
C
C                                        h gregg jul-82
C
C-------------------------------------------------------------
C

      SUBROUTINE GETDAT(IFILE,JSCAN,X,Y)
      INTEGER*2  IFILE,JSCAN
      REAL*4     X(1),Y(1)
C
C This subroutine gets all information associated with
C the current scan.  All error information is from the
C subroutines.
C
C where    IFILE  is the dataset number
C          JSCAN  is the scan number to retrieve
C          X,Y    are arrays for the X,Y data pairs
C
C returns  (see GETPTR, GETVAR, GETXY)
C
C calls    GETPTR (MDDB)
C          GETVAR (MDDB)
C          GETXY  (MDDB)
C
C                                        h gregg jul-82
C
C-------------------------------------------------------------
C
```

```
      SUBROUTINE GETPTR(IFILE,JSCAN)
      INTEGER*2  IFILE,JSCAN
C
C This subroutine gets JSCAN's pointer variables out
C of the pointer file.
C
C where    IFILE  is the dataset number
C          JSCAN  is the number of the pointer to get
C
C returns  IPTDAT pointer into the data file
C          NUMDAT number of data points
C          RTIME  time this scan was taken
C          RSUMY  sum of the Y values
C          IVARD  code for the dependent variable
C          IVARF  codes for the fast variables
C          RVARF  values of the fast variables
C
C calls    none
C
C                                        h gregg jul-82
C
C-------------------------------------------------------------
C

      SUBROUTINE GETVAR(IFILE)
      INTEGER*2  IFILE
C
C This subroutine get the variable parameters for the
C current scan (i.e. GETPTR must have been called).
C
C where    IFILE  is the dataset number
C
C returns  RVAR   variable values for the current scan
C
C calls    none
C
C                                        h gregg jul-82
C
C-------------------------------------------------------------
C
```

```
      SUBROUTINE GETXY(IFILE,X,Y)
      INTEGER*2  IFILE
      REAL*4     X(1),Y(1)
C
C This subroutine gets the X,Y pairs for the current scan
C (i.e. GETPTR and GETVAR must have been called).
C
C where    IFILE  is the dataset number
C          X,Y    are arrays for the X,Y pairs
C
C calls    none
C
C                                        h gregg jul-82
C
C-------------------------------------------------------------
C

      SUBROUTINE GETEND(IFILE)
      INTEGER*2  IFILE
C
C This subroutine gets quickly to the end of the dataset,
C and sets up the commons for the next write.
C
C where    IFILE  is the dataset number
C
C returns  all variables necessary for the next write
C
C calls    GETPRM (MDDB)
C          GETVAR (MDDB)
C
C                                        h gregg may-85
C
C-------------------------------------------------------------
C
```

User interface subroutines

```
      SUBROUTINE EXEDIT(INSTR,TRACE,IXY)
      INTEGER*2 INSTR,IXY(6)
      REAL*4    TRACE(4,1)
C
C This subroutine edits the 'trace' matrix used by the
C extract subroutine.  This trace matrix is the mask for
C the extraction program, and this subroutine allows
C manipulation of the input parameters (1&2).
C
C where    INSTR is a return variable for the main program:
C                < 0 ==> 3D plot: -INSTR is the parameter
C                                  for 3rd dimension
C                = 0 ==> exit
C                = 1 ==> start an extraction
C                = 2 ==> file the data
C                = 3 ==> call MULPLT
C                = 4 ==> extract and file the data
C                = 5 ==> file and plot the data
C                = 6 ==> extract, file and plot the data
C           TRACE  is the matrix of min/max values
C            (1,n)&(2,n) are the min/max limits allowed -
C                 edited here
C            (3,n)&(4,n) are the min/max found in the
C                 data base - from extract
C           IXY(1) is the code for the independent variable
C           IXY(2) is the code for the dependent variable
C           IXY(3) is the number of pairs extracted
C           IXY(4-6)aren't used here
C
C The numbers on the screen have the following format (see
C NUMDIS):
C
C           integer - if it fits
C           real    - if its fits
C           exponential: .xx-ee, .xxx-e, .xx+ee or .xxx+e
C                 if negative, one decimal place is lost
C
C The screen format is as follows:
C           lines 1&2 are for titles
C           lines 3-22 are for the numbers:
C                 3 variable 1      . . . .      variable 2
C                 4 variable 3      . . . .      variable 4
C                 etc
C                 22 variable 39    . . . .     variable 40
C           line 23 is for input
C           line 24 is for descriptive information
C
```

```
C Each variable location consists of a 40 character area as
C follows:
C          bytes 3-12 are a description of the variable
C          bytes 13,20,27,34 are spaces
C          bytes 14-19,21-26,28-33,35-40 are the numeric
C                fields 1-4
C
C calls:   GETPOS (MDDB)
C          VTxxxx (string)
C          EDISPL (MDDB)
C          NUMDIS (MDDB)
C          INPUT  (MDDB)
C
C                                        h gregg 7/84,8/84
C
C-------------------------------------------------------------
C

  SUBROUTINE INPUT(ICMD,VALUE)
  INTEGER*2 ICMD
  REAL*4    VALUE
C
C This subroutine works for EXEDIT, getting the user input,
C correcting mistakes and translating the input into an
C index for the main program.
C
C calls    VTxxxx (string)
C
C returns: ICMD   VALUE what the user typed
C                 1  -- Extract,  <PF1>Extract
C                 2  -- File,  <PF1>File
C                 3  -- Plot,  <PF1>Plot
C                 4  -- EF,  <PF1>EF
C                 5  -- FP,  <PF1>FP
C                 6  -- Go,  <PF1>Go
C                 7  -- Quit, ^Z, <PF1>Quit
C                 8  -- Z,  <PF1>Z
C                 9  -- Help, <PF2>, <PF1>Help
C                 10 -- <up-arrow key>
C                 11 -- <down-arrow key>
C                 12 -- <right-arrow key>
C                 13 -- <left-arrow key>
C                 14 -- <carriage-return>
C                 15 value value,  <PF3>value
C                 16 -- X,  <PF1>X
C                 17 -- Y,  <PF1>Y
C
C                                     h gregg may-85
C
C-------------------------------------------------------------
C
```

Main extraction subroutine

```
      SUBROUTINE XTRACT(IFILE,NPLANE,IXY,TRACE,ARRAY,MAXARY)
      INTEGER*2  IFILE,NPLANE,IXY(6,1),MAXARY
      REAL*4     TRACE(1),ARRAY(1)
C
C This subroutine extracts data from the multi-dimensional
C data base.  Several 'planes' of data may be extracted at
C once, dependent on the size of the data ARRAY.  As many
C planes of data as can fit into the ARRAY will be
C extracted.  The extraction parameters are passed in two
C arrays - IXY and TRACE.  IXY contains the codes for the X
C and Y values to be retrieved, while TRACE contains the
C constraints on the constants (min and max values forming
C range).  On return, TRACE will also contain the range of
C values selected for every variable; this helps determine
C if the extraction parameters were specified in enough
C detail.
C
C where     IFILE  is the dataset number
C           NPLANE is the number of X,Y planes to extract
C           IXY    are codes and pointers for each plane
C                  (1,m) pointer to code for X
C                  (2,m) pointer to code for Y
C                  (3,m) number of X,Y pairs extracted
C                  (4,m) pointer to first X value in ARRAY
C                  (5,m) pointer to first Y value in ARARY
C                  (6,m) max number of X,Y pairs, no shuffle
C            Note that an additional plane is used intern'ly
C            and space for it must be allocated by caller.
C           TRACE  extraction parameters
C            [equivalent to TRACE(4,NUMVAR(IFILE)+4,NPLANE)]
C                  (1,n,m) minimum allowed value
C                  (2,n,m) maximum allowed value
C                  (3,n,m) minimum found value
C                  (4,n,m) maximum found value
C                  (i,1,m) scan number (ISCAN)
C                  (i,2,m) time scan was taken (RTIME)
C                  (i,3,m) sum of Y values (RSUMY)
C                  (i,4,m) the Y values
C                  (i,k,m) the variables [IVAR(ifile,j)]
C                          where k=j+4
C                  m is the extraction plane
C           ARRAY  is the data array
C           MAXARY is the maximum size of ARRAY
C
```

```
C
C calls     ASHUFL (MDDB)
C           SHUFFL (MDDB)
C           GETPTR (MDDB)
C           GETVAR (MDDB)
C           GETXY  (MDDB)
C           STORXY (MDDB)
C           TRACER (MDDB)
C           IT     (MDDB)
C           ITRACE (MDDB)
C
C                                          h gregg aug-82
C
C-------------------------------------------------------------
C
```

Common area - variable definitions

```
C
C MDDB.CMN - common areas for the MDDB data base routines
C
  PARAMETER MXFIL=1      ! maximum number of open datasets
  PARAMETER MXSTC=40     ! maximum number of static params
  PARAMETER MXVAR=40     ! maximum number of variable params
C
C Variables common to all datasets:
C
C for the dictionary:
C
  BYTE      QSDICT (20) ! short dictionary description
  BYTE      QLDICT (58) ! long dictionary description
  INTEGER*2 LUNDIC      ! logical unit for the dictionary
  INTEGER*2 IPTDIC      ! dictionary pointer: no/val/point
C
C misc. variables:
C
  BYTE      QSYSTM (20) ! device and UIC for common files
  BYTE      QDATE  (10) ! the date \  either current
  BYTE      QTIME  ( 8) ! the time /  or from the dataset
  BYTE      QCOMNT (82) ! comment lines from the header
  INTEGER*2 LOONTI      ! the logical unit for TI: (errors)
  INTEGER*2 IERR        ! error return flag
  INTEGER*2 MAXFIL      ! == MXFIL
  INTEGER*2 MAXSTC      ! == MXSTC
  INTEGER*2 MAXVAR      ! == MXVAR
C
C variables for each data set:
C
  BYTE      QDSFIL (   30,MXFIL) ! dataset file name
  INTEGER*2 LUNHDR (      MXFIL) ! LUN for the header file
  INTEGER*2 LUNPTR (      MXFIL) ! LUN for the pointer file
  INTEGER*2 LUNDAT (      MXFIL) ! LUN for the data file
C
  INTEGER*2 ISCAN  (      MXFIL) ! current scan number
  INTEGER*2 IPTDAT (      MXFIL) ! pointer to current scan
  INTEGER*2 JPTDAT (      MXFIL) ! pointer to memory record
  INTEGER*2 IDAT   (      MXFIL) ! pointer into RDAT
  REAL*4    RDAT   (   16,MXFIL) ! one data record buffer
  REAL*4    RTIME  (      MXFIL) ! time this scan was taken
C
```

```
C
  INTEGER*2 NUMSTC (      MXFIL) ! number of static params
  INTEGER*2 ISTATC (MXSTC,MXFIL) ! code for static params
  REAL*4    RSTATC (MXSTC,MXFIL) ! value for static param
C
  INTEGER*2 NUMVAR (      MXFIL) ! number of var. params
  INTEGER*2 IVAR   (MXVAR,MXFIL) ! code for variable params
  REAL*4    RVAR   (MXVAR,MXFIL) ! value for variable param
  INTEGER*2 IVARF  (    3,MXFIL) ! code for the fast vars
  REAL*4    RVARF  (    3,MXFIL) ! values of fast vars
C
  INTEGER*2 MAXDAT (      MXFIL) ! max number of data pairs
  INTEGER*2 NUMDAT (      MXFIL) ! number pairs this scan
  INTEGER*2 IVARD  (      MXFIL) ! code for dependent var
  INTEGER*2 IPTX   (      MXFIL) ! pointer for X, this scan
  INTEGER*2 IPTY   (      MXFIL) ! pointer for Y, this scan
  REAL*4    RSUMY  (      MXFIL) ! sum of Y, this scan
C
  COMMON    /MSFILE/
 1 QSDICT, QLDICT, QSYSTM,  QDATE,  QTIME, QCOMNT,
 2 LUNDIC, IPTDIC, MAXFIL, MAXSTC, MAXVAR,   IERR,
 3 QDSFIL, LUNHDR, LUNPTR, LUNDAT, LOONTI,
 4 NUMSTC, ISTATC, NUMVAR,   IVAR,  ISCAN, IPTDAT, JPTDAT,
 5 NUMDAT,  IVARD,  IVARF, MAXDAT,   IPTX,   IPTY,   IDAT,
 6 RSTATC,   RVAR,   RDAT,  RTIME,  RSUMY,  RVARF
C
C-------------------------------------------------------------
C
```

Appendix 3

EXTRACT User's Guide

Hugh Gregg
June 24, 1985

EXTRACT is a program for interrogating a multi-dimensional dataset and extracting any two dimensional plane of data from it. The resulting data can then be formatted and presented to MULPLT for immediate graphics presentation, or the user may choose different extraction parameters and extract another plane of data.

EXTRACT, the program

The program EXTRACT was created to extract any two dimensional plane of data from a multi-dimensional dataset. This is accomplished by specifying a set of extraction limits, and letting the program search through the dataset extracting all the data that matches the specified constraints. The actual limits of the extraction are then presented to the user. If these limits are acceptable, the user may plot (using MULPLT) the extracted data.

Multi-Dimensional Dataset

Datasets created by multi-dimensional instruments, such as Triple Quadrupole Mass Spectrometers (TQMS), may contain more than two dimensions of data (i.e. more than mass vs. intensity pairs). A TQMS has many instrumental parameters, each of which, when varied, creates an orthogonal dimension of data. Examples of these dimensions include quad 1 mass, quad 3 mass, the axial energy, collision pressure, and ion intensity. Varying more than two variables results in a dataset with multiple dimensions of data. A multi-dimensional data base system (MDDB) was created to handle these data.

An MDDB dataset is a set of three files (*.HDR, *.PTR, *.DAT) that contain all the instrumental parameters as well as the data. The data is stored in this dataset in a series of "SCANS". Each scan is the result of one instrumental operation (i.e. an intensity vs. quad 1 mass scan, or an intensity vs. axial energy scan at specific parent/daughter masses). The values of all variables at the time each scan was recorded are stored with the scan. In this way, each scan is a two dimensional slice into the multi-dimensional dataset with all other variables held constant.

How extraction works

EXTRACT extracts a plane of data based on the constraints specified by the user. The user is presented with a list of all the variables in the dataset and must choose what data is to be presented. The abscissa and ordinate (X and Y) of the resulting plane are selected. If the user wishes to limit the extraction process, the limits of the variables must be set. When EXTRACT is told to do it's stuff, it extracts all data that satisfy the constraints. When this is accomplished, the user is presented with the actual limits found during the extraction. If these limits are acceptable, the data may be formatted for MULPLT and displayed. However, if the actual limits of the extracted data indicate a variance larger than acceptable, the limits of the offending variable may be narrowed and the extraction tried again.

Using EXTRACT

To use EXTRACT, you must first convince RSX to let you use it. This is done by typing "EXTRACT" or "EXTRACT datasetname". If you don't specify the dataset name on the command line, EXTRACT asks you for the name. If the dataset exists, EXTRACT displays all the variables and waits for the user to type something. One section of the screen is highlighted - this is the "active" area. The user may move to a new active area by using the cursor control (arrow) keys on the terminal. To change a specific value, move the active area to the position of the value to change, then just type in the new value (followed by a <CR>). Commands can be entered at any time, and take immediate action. For more information about the commands and how to enter values, see the sections on COMMANDS and VALUES.

Screen Display Format

The screen format of EXTRACT displays all the user variables stored in the dataset. Each variable has four values associated with it: the minimum and maximum extraction limits (set by the user) and the minimum and maximum limits found by EXTRACT in the dataset.

Each value is displayed in only six character positions and may appear truncated. If the value conveniently fits within the six character window, it is displayed in full. A problem arises when displaying large or small numbers (i.e. 2.3456E7 counts). These numbers are displayed in exponent format with implicit "times 10 raised to" (i.e. .234+8). In the worst case, only one digit of the number plus the exponent is visible (-.3-16).

The bottom line of the screen displays an enhanced version of the active area. A number truncated in the main portion of the screen is displayed in full along with the full name of the variable. This status line is also used to display definitions of certain values. For example, a value of "2" for the variable "CAD gas type" produces the description "CAD gas type: Nitrogen".

Values

Values may be entered in a variety of ways, the simplest of which is to type in the new value. While this method will always work, there are a few short cuts available. When the active area is at either the minimum limit or the maximum limit, the following may be used:

Minimum limits

<CR>	A single carriage return, when the minimum limit is the active area, resets the minimum and maximum limits to "no limit". This allows EXTRACT to use any values for this variable.
N	Entering a number N results in that value being inserted into both the minimum and the maximum value slots. Note that the input of exponents must be done in computer notation (i.e. 1.23E7), and that a carriage return must be typed after the number.
N_1, N_2	This format enters the number N_1 into the minimum value slot, and the number N_2 into the maximum value slot.
N,+i	This format enters the number N into the minimum value slot, and the number N+i into the maximum value slot. This format allows easy entry of a range of limits; you specify the minimum value and the range or increment.
N,+	This format enters the number N into the minimum value slot, and the number N+1.0 into the maximum value slot. This format is identical to the above format, but defaults to a range of 1.0 (useful for mass selection).
N,.	This format enters the number N into the minimum value slot, and sets the maximum value to infinity (1.0E36).

Maximum limits

<CR>	A carriage return at the maximum limit sets the maximum equal to the minimum limit.
N	Entering a number N results in that value being inserted into the maximum value slot.

Command Summary

This is a list of the available commands and a brief description of their use. All commands may be abbreviated to their first letter, and are explaned in greater detail below.

Single keystroke commands:

arrow keys	move the active area
<PF1>	get the "Command?" prompt
<PF2>	Help: display the known commands
<PF3>	get the "Value:" prompt
<CR>	at first limit: sets "no limits"
	at second limit: set second value to first
^Z	Exits EXTRACT
anything else	starts either a command or value

Other Commands (in response to the "Command?" prompt):

X	set the abscissa
Y	set the ordinate
Z	three-dimensional extraction and file
Extract	Extract a plane of data
File	File the extracted data
Plot	call MULPLT
EF	Extract and File a plane
FP	File and Plot extracted data
Go	Extract, File and Plot extracted data
Save	Save the current extract limits
Read	Read in saved extract limits
Help	Display the command summary
Quit	Exits EXEDIT

Description of Commands

X or Y — This command sets the active area to be either the abscissa or ordinate (X or Y) for the resulting extract plane. A highlighted letter (X or Y) is placed in front of the variable name to indicate the current X and Y variables.

Z — This command executes a three-dimensional extract. Position the active area to the variable that will be the third axis (note that this variable's limits must be set). Enter the Z command, and you will be asked for the incremental value for the Z variable. This procedure does a series of two dimensional extracts, starting with Z equal to the minimum limit, and subsequent extraction with the Z value incremented until the maximum limit is reached. Data

from each two dimensional extract are filed. An auxiliary program (PLOT3D) is used to read this file and produce MULPLT files for pseudo three-dimensional plots.

Extract — This command extracts a plane of data from the dataset and displays the resulting extraction limits. The data extracted are not automatically filed or plotted.

File — This command files previously extracted data. The filename is derived from the dataset name (*.BIN), and each File command in one extraction run uses the same file (the MULPLT tag for each file command starts at "AA" and is incremented for subsequent file commands: "AB", "AC", etc). A MULPLT command file is also created (*.PDL). Before the first file command is executed, the user is asked whether the plot will be a point plot, line plot, bar graph or spectra plot. Point plots and bar graphs are just that; the data for line plots will be sorted; spectra plots are normalized to the base peak, and the maximum intensity is put into a MULPLT special features file (*.SPF).

Plot — This command go directly to MULPLT, passing it the name of the command file generated by the File command. The user is left in MULPLT, ready for a GO command, or whatever. To exit MULPLT, type Halt or ^Z, and you will be returned to EXTRACT.

EF — Extract and File command for those times that you know the extraction will work, but you don't want the data plotted yet.

FP — File and Plot command. Used after doing an extract command, and you wish to plot the data.

Go — Extract, File and Plot extracted data, this command "Goes for it", and does it all.

Save — This command saves the current extraction limits in a file (the default file name is derived from the dataset name, *.EXT). This is a normal file that may be edited, displayed, etc. For information on the file format, see the Read command below.

Read	This command reads in a file of saved extraction limits (the default file name is derived from the dataset name, *.EXT). The file contains lines with the following form: "variable = N_1 to N_2" for each of the variables with limits set. The variable name must exactly match the short dictionary definition, and the two keywords "=" and " to " are not optional.
Help	This command produces a short list of available commands.
Quit	This command exits EXTRACT.

The mechanics of EXTRACT

Internally, EXTRACT consists of three major phases: extraction parameter editing (EXEDIT), extraction (XTRACT) and formatting for plotting (MSPLOT). Both the editing and plotting phases are described in the COMMANDS section. The extraction routine itself basically operates as two units. The first unit processes the simplest of the extractions; i.e. when the extraction plane is a scan. In this case, the requested plane of information is stored in the format that the user wants, and all that is required is to retrieve it.

The retrieval of a plane of data that wasn't scanned is more complicated. A large loop is initiated where each cycle of the loop looks at one scan in the dataset. The limits specified by the user are used to try to reject a scan from further consideration. If the scan cannot be rejected, the X and Y variables that the user specified are studied. If there are multiple X values, all X values are retrieved along with their corresponding Y values and are stored in the extraction plane. If there are multiple Y values and only a single X, the Y values are averaged and the resulting X, Y pair is placed in the extract plane. Finally, if no X or Y values match their extraction limits, no data from this scan is saved. Once this scan is finished, EXTRACT loops back and gets the next scan until the entire dataset has bee examined.

The dimensions of the internal arrays in EXTRACT are not fixed as in conventional programs. This allows quite a bit of flexibility in the size of the scans and planes extracted. Space for the data in each scan (from the dataset) and the extract plane (under construction) is dynamically allocated from a pool of 16000 bytes (room for 2000 X,Y pairs). Therefore, as the extract plane is built (more data pairs added to it), the room available to hold the next scan diminishes. In practice, this space is more than enough. If we assume a fixed length scan of 1000 X,Y

pairs, we still have room for an extraction plane of 1000 X,Y pairs.

GLOSSARY of terms used

Active area	describes the area currently highlighted on the screen.
Extract	the process of searching the dataset for a specific two dimensional plane of data.
Plane	a two dimensional grouping of data, usually extracted from a multi-dimensional dataset.
Scan	one instrument scan; i.e. one grouping of data consisting of X,Y data pairs and the corresponding values for the variables.
Value slot	one of the four minimum or maximum values for a variable. A value slot may become the active area.
<CR>	the carriage return key.
<PF1>	the PF1 key on VT100's, the "gold" key. Used to start a command.
<PF2>	the PF2 key on VT100's, next to "gold" key. Used to obtain Help.
<PF3>	the PF3 key on VT100's. Used to start a value.
^Z	control Z (hold the control key down and press the Z key). Used to exit EXTRACT.