Other Investigations

LITHCHEM: AN INTEGRATED GEOLOGICAL DATABASE FOR MICROCOMPUTERS

By J. C. Harrop and A. J. Sinclair
The University of British Columbia
Department of Geological Sciences

INTRODUCTION

Lithchem is a dedicated geologic database system with graphical and statistical support integrated into the basic design. The system was developed with a specific application in mind, storage and manipulation of chemical analyses of igneous rocks. However, because other sets of geologic data have essentially the same types of fields within each record, lithchem can be modified for other applications. Lithchem is currently 2500 lines of source code, which compiles to about 47 kilobytes (Kb) of object code.

Lithchem was written in TURBO Pascal[1] on an IBM XT microcomputer with 576 Kb internal memory and a 10 megabyte (Mb) hard disk. The 8088 microprocessor was augmented with a 8087 numerical co-processer greatly enhancing execution speed of *real* arithmetic and *transcendental* functions, as well as providing extended precision reals and integers. TURBO-87 takes advantage only of the extended precision reals. To use the extended integers of *transcendental* functions one must develop the appropriate assembly language subroutines or await advanced versions of TURBO-87.

DEVELOPMENT CRITERIA

Of the criteria controlling the choice of language and algorithms in this project, time was probably the ultimate one. The end result has numerous areas for improvement and expansion. With time of development being necessarily short, methods and algorithms were the simplest that would do the job. The most elusive bugs were found to be features of the compiler and alternate code was written to avoid these problems. It was anticipated that the system would be used by non-experts with regard to programming and thus prompting for input with menus was used as much as possible. Input that requires data is always assigned a default value if no value is supplied. Range checking and checking membership in a known set are also required to reduce typographic errors that would confuse searches and be difficult to find later. To ensure ease of use for non-expert programmers, a system that would internally pass data between searching, graphical, and statistical procedures was conceived.

The following considerations were given to developing such a system in the microcomputer environment. Choice of the programming language had to meet the following requirements:

(1) The system will require many steps so a compiler, rather than an interpreter, is needed.
(2) The system will become complex and should allow possibilities for expansion, therefore the language should be structured.
(3) The language will be need at least graphics primitives.
(4) The language should utilize dynamic variables to allow the full power of the memory to be used.
(5) The language should have adequate mathematical support and not limit the statistical applications.
(6) The language should be commonly available and able to run on other microcomputers, portability is important.

Given the requirements, any language will fall short somewhere so compromise is inevitable. FORTRAN is reasonably portable and certainly easily available, but it lacks easily accessible dynamic variables and the version available to the writers also lacks graphics. Consequently, FORTRAN and for similar reasons BASIC, were ruled out. The remaining possibilities included C and Pascal with more points in favour of Pascal. While C fits the preceeding description well and is probably the most portable, Pascal is available for most microcomputers in Borland's economical Turbo version. Pascal only falls short in the mathematical support but this could be overcome by using the 8087's powers. Certain features of Turbo Pascal have been of use in the development of this project and they will be mentioned in the appropriate section.

DATA STRUCTURES

Each sample has one record associated with it that contains:

(1) Sample number.
(2) Map area (NTS map sheet, for example, 93G/11).
(3) Assemblage.
(4) Rock unit (general and specific).
(5) Rock type (author's and revised).
(6) Source (number reference to a bibliography).
(7) Twelve main oxides, LOI, CO_2, and S.
(8) Six trace oxides.

The data are found in the form of strings of unknown length, integers, and some fixed point numbers or reals since Pascal does not contain a fixed point type. In some languages namely FORTRAN and BASIC we probably would have to define arrays, of fixed size, for each of these fields of data. With a total of 28 fields we would be lucky to be able to work with one or two thousand samples at one time. With sets of data exceeding the memory's capacity, pages of data, perhaps in smaller amounts to lessen each loading time, would have to be moved on and off the disk. On the IBM XT it took between 2 and 3 minutes to load 1 000 samples this way from the hard disk. We did not use the primitive read procedures and consequently could be made faster; nonetheless it is comparable or better than FORTRAN or BASIC which are notoriously slow with their I/O routines. For a search to go through 3 000 samples, 12 to 18 minutes would be spent on I/O processing alone! A different approach to variable declaration must be taken. Arrays of the type mentioned above have to be defined at the beginning of the system and are of fixed size. Because of the internal structure of the compiler these arrays are usually limited to around 64 kilobytes of memory for all the arrays. If one relieves the compiler of the job of keeping track of the location in memory of the variable (in this case an array), there is no longer the 64-kilobyte limit to the data storage capacity and in this application approximately 5 000 samples can be dealt with at one time! This should remove the need to flip pages on and off the disk for most data sets, and search times will consist only of the time the system takes to work its way through memory.

The next major decision lies in how the data should be stored in each record. Pascal allows the programmer to define a *record* type variable that consists of several fields of similar or different variable types. For instance we could define a record with a field for each of

1. TURBO-87 © 1984 by Borland International, in TURBO PASCAL Version 2.0

the previously mentioned variables of the appropriate type for the data. Closer inspection shows that substantial savings can be made by compressing the data. The real numbers reported for whole rock analyses are never less than 0.01 per cent and never (we hope) more than 99.99 per cent for a single oxide. Real numbers in Turbo require 6 bytes whereas integers only require 2 bytes. The range of a *two's* complement binary integer is as $-(2^{n-1})$ to $2^{n-1}-1$ where n is the number of binary digits. In this case a 16-digit integer will give the range -32 768 to 32 767. Since the per cent oxides are reported with four digits of accuracy an integer of four and one-half digits can contain the data as long as we keep track of the position of the decimal place. This can be done by storing the data as one hundred times the reported values, truncated. Oxide data forms a large part of the record and this method gives us two or three times as much room as would be available using all reals.

The rest of the data is mainly strings, which in Turbo are stored as one byte per character and one preceding byte to specify the length of the string. The source can be recorded as a number reference to a bibliography kept in a separate text file. This text file does not require anything more than the editor to maintain so further discussion of this part of the system is not needed. Sample numbers currently are used to sort out which of the sources samples is being referred to, so an integer will probably suffice in this application. We still have the 'map area', 'assemblage', 'rock unit', and 'rock type' to represent. Turbo requires that we specify the length of the string at the start of the system, so a maximum length that will not cause problems will have to be set. We could use four-letter mnemonics and keep the strings short, but this will result in the system being harder to use by persons not very familiar with it as well as increasing the chance of typographic errors during input. Almost all names we need here can be put into a twelve-character variable, and these names will be repeated many times throughout the samples. We have assumed during development that each of these variables will have less than fifty different names. Some such as 'assemblages' may have fewer but others such as 'map area' may need more than fifty. As the system is used these details can be clarified. Now, if each name is kept in an array of 12-character strings, one array for each variable, then the individual records will only need to keep the integer that represents the position in the array of the appropriate name. The names will be stored in the long, rather than mnemonic form and for each time the name is repeated a saving of ten bytes is made over storing the full name. Ease of use is maintained and in finding the position of each name entered in the array any typographic errors can be queried and the correction made at the time of entry. In the case of rock type and rock unit, two fields have been merged into one by using the eight most significant bits to represent an 8-bit number, and the eight least significant bit *numbers* as a second 8-bit number. These can be separated easily by binary masking using an arithmetic *and* operation. If the upper eight bits are being recovered an 8-bit right shift is also used (equivalent to an integer divide by 128) to recover the data. Note that this has left room in the '*map area*' and '*assemblage*' variables for later addition of new variables.

Thus, we see that all data entries can be reduced to integers and stored in a compressed form. There are four more variables attached to each record that also require note here. These are *next*, *last*, *nexta*, and *nextb*. Dynamic variables are not defined at a particular location by the compiler, so it is the job of the system to keep track of where the record currently being operated on is kept. This record will also need to contain the memory address of the next record, thus building a *linked list* of records. When these records are being examined the user may wish to back up one or more records so a second link to the last record is also maintained resulting in a *doubly linked list*. These addresses are generated and stored in *next* and *last* during the initial setup when the records are recovered from the disk. The two other variables *nexta* and *nextb* are set aside for sorting linked lists by some parameter (for example, by SiO_2 content) and have not yet been used. One last variable used in the record are integers used as flags in the search routines. These integers are also available to the plotting and statistical routines and may be saved on the disk if desired. This variable will be discussed in detail in the next section. The remaining field is not used as a variable but is assigned when a sample is entered and is a unique identifier that remains set and cannot be edited or refused. *Refnum* is also used to identify samples for editing and other such operations of the system.

This results in the following list of variables in each record, using a total of sixty-four bytes per record.

(1) next, last (pointer variables) : linked list.
(2) main[1.28] (array of 28 integers) : compressed data.
(3) samflag (integer) : set membership flags.
(4) nexta, nextb (pointer variables) : for future sorting routines.
(5) refnum (integer) : unique identifier.

MAIN ALGORITHMS

Here we will move through the system in about the same order in which the system would run. The majority of the system has been designed to be *menu driven* with the goal of always prompting the user for instructions. The user is kept aware of what commands are currently valid. A few situations, such as when plotting a composition diagram, are not favourable for showing menus. In these cases the menu is either very simple, or a space or return keystroke will get the program to the next frame or menu. These areas should not cause problems after only limited use of the system.

The records are loaded from and written to the disk using the lowest level procedures available in Pascal, namely *blockread* and *blockwrite*. The advantages of these are speed, smaller disk files, and the fact that the procedures are standard Pascal. They do, however, require that data be transferred in 128-byte blocks and a double sample buffer is used to do this. Notice that one record requires 64 bytes, thus two will fit neatly into 128 with no wasted bytes. When the next record pointer is nil, or in other words there are no more records in the list, the procedure continues on to the next stage.

Samples are entered and edited one at a time in a form mode. Using the IBM special characters for the text screen, a form for the sample entry and editing routines has been designed. The form allows the user to fill in the blanks and, if no data are available, a default is provided. This helps to prevent random values from entering sample data. The form also enhances readability of the data. Editing provides the current value for the default and, if no re-entry is required, then a return keystroke will pass on to the next line. The routines that read in the data are set to allow only valid characters. This helps reduce typographic errors and ensures that data can be compressed correctly (*see* preceding data structures). Individual forms can be printed on the dot matrix printer with the print screen key. While forms are a convenient way to view small amounts of data on the screen, larger sets often need a tabular format. This can be done to the printer by specifying which fields to print per line. The whole sample is considerably larger than a single line.

Searching is a major part of this system and the routine has been kept simple to ensure ease of use. This area of computing has many refinements to offer which have not been used due to lack of time. Every sample has an integer flag associated with it. This is then used as fifteen *set flags* to show which sets, if any, a given sample belongs to. When searching a source and target, flag must be specified. The source, which could also be the whole file, tells the system which of the samples to make the search comparison with. The target, which cannot be the whole file, is a flag that the search will use to mark which samples meet the search criteria. This target can then be used for plotting, listing, or further searching. When a search is made the samples meeting the criteria can be added to the target, removed from the target, or removed from the target if they do not fit the criteria. These operations can only be done one at a time, but they can be sequenced to find the desired set of data. There is also an operation to join two sets. When saving the samples the option is given to save the flags. This allows one to continue with the same sets the next session since otherwise the sets would be lost.

GRAPHICS

The graphics support given by Turbo goes no further than moving or drawing from point to point with the coordinates given in pixels. In the case of the IBM XT the vertical range is 320 and the horizontal 600 pixels. Graphics routines are provided for nine plots which one may select at random. A tenth option sets the symbols to be used and with which flags the symbols are to be associated. This remains set so that several different diagrams may be displayed. Up to six different characters may be assigned to any or all of the fifteen sets defined by the search routines. This was conceived to be useful in the comparison of potentially different sets and in combining sets at the plotting stage for comparison rather than returning to the search facility. Copies of the screen output can be made on the dot matrix printer with the print screen key. Some distortion with respect to the length of the axes will occur when this is done, but this should be of little concern in this situation since the axes are of arbitrary length to begin with.

STATISTICS

This section, while planned for in future expansion, has not been included in the initial version of the system. This is only for time constraint reasons and because the application of the system currently does not require statistical procedures to be useful.

PORTABILITY

Some of the problems of portability have been dealt with in the choice of language. Turbo Pascal is easily available and very reasonably priced so any microcomputer supported by this language will be able to accept a source code version of Lithchem. Any IBM PC or XT should be able to run Lithchem immediately and if they do not have an 8087 numerical processor then a compiled version with the regular Turbo would be directly transferable. Most IBM compatibles will be able to accept the system with very few modifications. The areas of code that are machine dependent have been isolated, whevever possible, to short subroutines, many of which are found in the first few hundred lines of source code. The adjustments needed would amount to one or two days of work.

Should the system be moved to a microcomputer using another version of Pascal the problems will be greater. Care has been taken to keep the code as close to standard Pascal as possible. Again, the majority of changes would be in the sub-routines found at the beginning of the code. If the microcomputer is not IBM compatible but uses the 8088 and 8087 processors a compiled version may be portable.

Perhaps the most important question would be whether the machine has enough memory for the proposed application of the data base and whether a hard disk is essential or not. In its current configuration the size of the Lithchem system's database is limited by the amount of random access memory (RAM) in the host. No capacity for *paging* data on and off the disk has been included. As well as having simplified the development, this strategy has kept the operational speed of the system up.

EXPANSION AND FURTHER DEVELOPMENT

As mentioned previously, one section that can be added with relatively small effort is that for statistical procedures. These would interleave with the graphics since some of the statistical methods would require graphical output. This is the only other set of procedures that should be added internally to the system. To add more would result in the complexity and size of the system becoming unmanageable for future programmers to work with. Thus a method has been used to allow system additions to be made without having to enter the main body of the program. This is the ability in Turbo to *chain* systems so that a second system can be executed with the data left by the first system still accessable in memory, from the first. As long as subsequent programmers know the format of the data further sections can be added as particular needs arise. The main system can be returned to by *chaining* back to a version identical in everything except that it does not initialize variables or load the data from the disk.

Of the many further developments possible one in particular would be most useful. That is a routine to enter a file of data from either MTS (the operating system on the mainframe at the University of British Columbia) or another external source, an analytical laboratory, for instance. Batch entry would remove the most time consuming part of the current system.

The modular form of the Lithchem system and data structure can be modified to work for similar applications. Some of these would only require a revised record structure, while others would need new routines added. Applications of such an integrated system could range from soil geochemical interpretation to working with isotopic data.

ACKNOWLEDGMENTS

This work has been supported by the British Columbia Science Council in cooperation with the Geological Branch, British Columbia Ministry of Energy, Mines and Petroleum Resources.

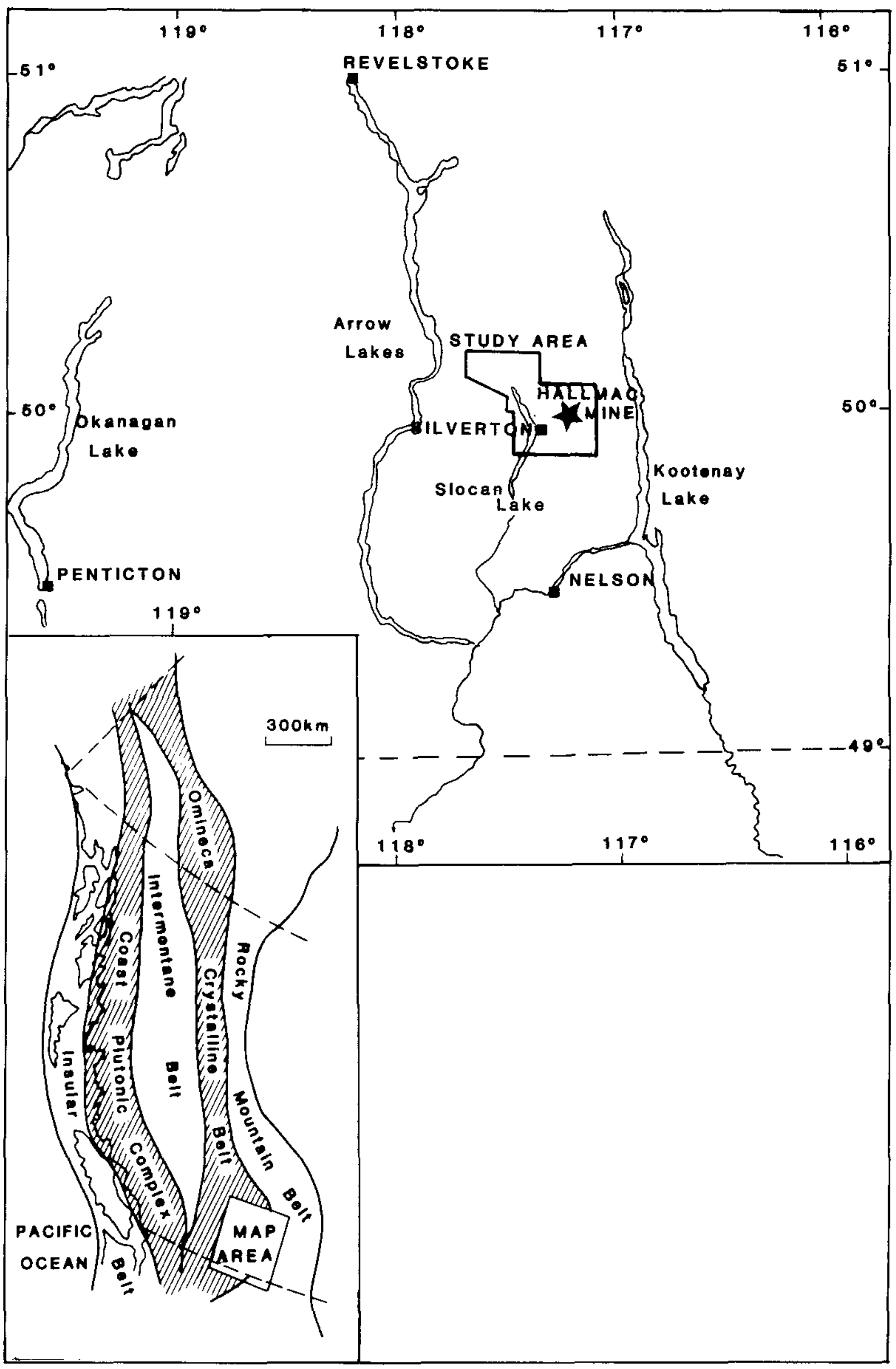

Figure 45-1. Location map of Hallmac mine. Inset map shows location with respect to major physiographic subdivisions of the Canadian Cordillera.