APPLIED MATH *HAS A BABY*

BRAIN AT DAL

By Allan Jost

Dalhousie University has expanded into the field of electronic brains. For the past year the University has been contemplating getting a computer, and on December 1, this idea became a reality when the newly formed Applied Mathematics Department received a $100,000 IBM machine.

The new department, formed this year as a division of the Department of Mathematics, is headed by Dr. A. D. MacDonald, who did his undergraduate work in Mathematics here at Dalhousie, and now has a Ph.D., in Physics from Massachusetts Institute of Technology. The other faculty member in the department is Dr. J. R. Baines. The department also employs a full-time programmer, Miss Judith Hunter, a graduate of McGill University and native of Alberta.

The department is presently offering only two courses, but it plans to expand in the future and this year's freshman should be able to obtain a degree with Honours in Applied Mathematics. One of the undergraduate courses now being offered includes a certain amount of work with the computer, but the machine is intended mainly for graduate research.

The computer, an IBM 1620 Data Processing System, is rented on a long-term basis from IBM who are responsible for its maintenance. They still own the machine. The only cost to Dalhousie is the rental fee and the cost of electricity, and the electric power consumption is very low, since the machine is completely trassistorized. Is spite of this, the computer is quite heavy, and proved to be too awkward to handle on stairs, so that a crane had to be used to get it to its present quarters in the penthouse on the roof of the Sir James Dunn Science Building.

A computer is, in simple terms, a fantastically accurate moron. It can add, subtract, multiply, divide, take square and cube roots, calculate the values of determinants, integrals and Taylor Series', and work with matrices, but it cannot think. It will do exactly what it is told to do, and not a thing more. For example, when instructing it to add two numbers, it must be told explicitly where those numbers are, and where to put the results. When solving equations, the problem must first be carefully broken down into single arithmetic operations, before the machine can tackle it. It can handle only two numbers at a time, and complicated expressions can take 30 steps or more to evaluate (I know of one problem which required over 1000 steps.)

As an example of the useless things computers can be made to do, they can be and have been programmed to play (and usually win) such games as poker, blackjack, tic-tac-toe, checkers, and chess. A British computer has calculated the value of "pie" to more than 5,000 places. On one occasion a particularly large machine, when it developed a fault in its circuitry, actually found a way to get along without the affected section, and the defect wasn't found for quite some time, because the computer had, in effect, repaired itself! Even in this case we could not say that the machine was actually thinking for itself, but it was probably dangerously close.

Individual steps in a problem take the form of numerically coded instructions. Each arithmetic operation has a 2-digit code number which the machine can understand. Similarly, "branch" instruction i. e. instructions involving simple decisions which may affect the flow of operations, have 2-digit codes. Each complete instruction given to the computer consists of a 12-digit number, containing, along with other information, the 2-digit operation code—the "other information" usually pertains to where to find the numbers to be operated on, and where to put the result. A collection of these numerical instructions, put together in a way that enables the computer to solve a problem, is known as a program, and a program written in the above mentioned manner, i. e., written in "machine language", is known as an "object program".

Writing object programs involves a lot of tedious work. Many operation codes must be memorized, and the programmer must keep track of the available space in the memory. To make the job of programming easier, IBM and other computer companies have developed simplified programming systems. One of these, "symbolic programming", substitutes groups of letters, usually of high mnemonic (look it up) value, for the numerical codes. Thus "A" can be used instead of "21" (the code number for "add"). Similarly, a particular number may be called "X" or "BIGA" instead of "14575" (this number is an "address" specifying a particular location in the memory). These systems simplify programming a lot, but it is still possible to go a step further.

Representative of "the step further" is the main programming system in use at Dalhousie. Instead of using artificial numerical language to tell the machine to "add A to B and store the result in memory location C", in this system you simply write "C=A+B". Since this resembles ordinary mathematical language, the system is known as FORmula TRANslation, or simply FORTRAN. As a further illustration of the language used, in Fortran,

$$X = \frac{R(A+B)}{AB}$$

is written "X=R*(A+B)/(A*B)", where the asterisk denotes multiplicition. A program written in this language, or in any other symbolic language, is known as a "source program".

To enable the computer to understand programs written in this language, a machine language program had to be written by IBM to translate programs written in Fortan to make object programe out of them. In effect, the machine was programmed to write its own programs, although strictly speaking, it is only a translation process. It reads the Fortran program, interprets it, and spews out, on punched cards, a machine language version of the original source program. The "translator" program consists of a deck of punched cards, obtainable from IBM. This deck, when fed into the computer, programs it to solve the "problem" of translating. The machine language program to accomplish this task is punched on the cards in numerial form. Having this system, programs can be written in the relatively simple language of Fortran, and the really tedious work can be done by the computer.

First, the entire memory of the machine is "erased", to remove unwanted material (and to make sure the machine doesn't get any wrong ideas). Then the Fortran "translation" program is read in via punched cards. This puts the program in the memory, where it can be used. (Note that this is why the machine is called a "stored-program" computer—the program is stored in numerical form in the memory.) This done, the source program, which must be punched by hand into IBM cards, is placed in the read section of the Read-Punch unit (this machine, actually a part of the computer system, is used whenever the computer is using cards, whether as an input medium or an output one). When the start key is depressed, control of the computer is handed over to the translation program, which reads in the source program, translates it, and again using the Card-Punch unit, punches out the object program, the machine language version of the Fortran program, on IBM cards.

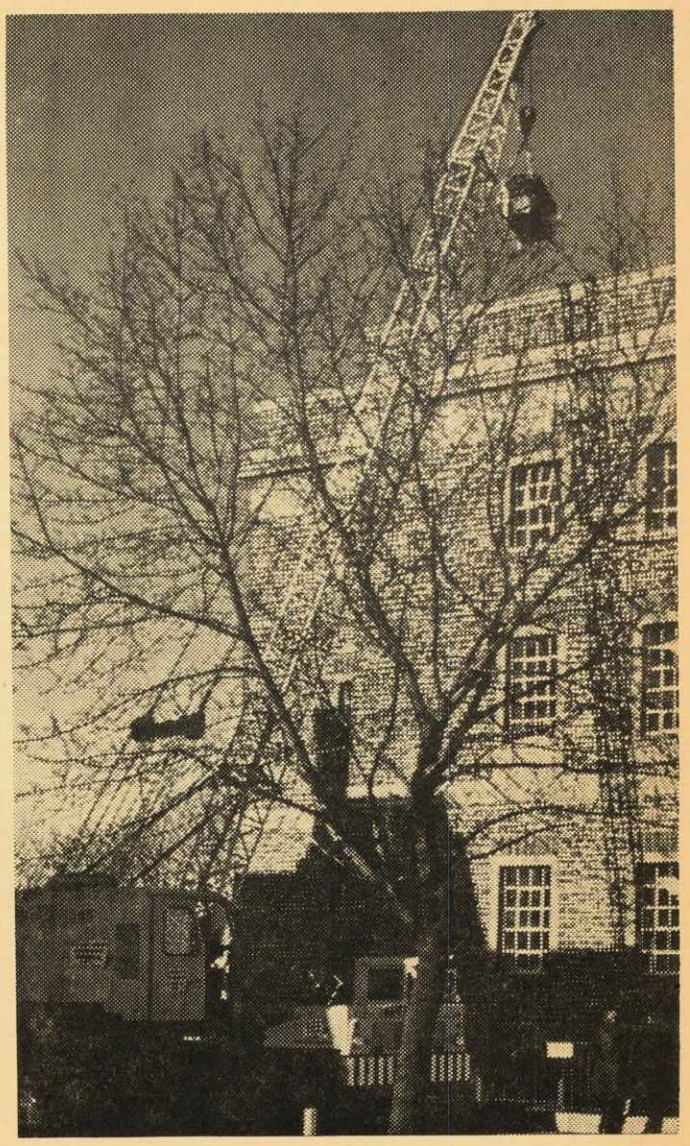
Now the memory is again cleared, and the object program card deck is placed in the read section of the Read-Punch unit. The computer reads this program into its memory and transfers control to the program. The entire system is now under "program control", and the program can read in the data to be processed via the Read-Punch unit. This data must be supplied on cards by the programmer. Then it performs the indicated operations on the data, and types out the answer(s) on the console typewriter.

The above procedure is obviously a bit too involved and time-consuming for simple problems which may only have to be done once or twice (note that the object program turned out by the Fortran system can be used to solve any number of similar problems, and that even when it is erased from memory, the deck of cards containing the object program is still intact and may be used again without the necessity of retranslation).

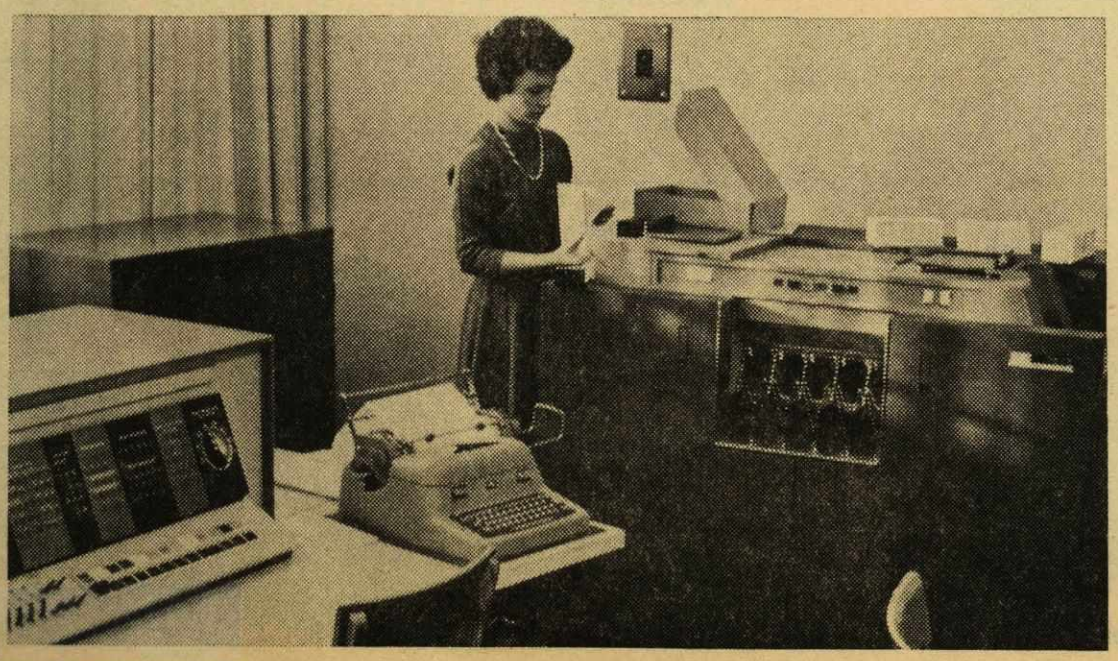
For one-shot programs, a system called GOTTAN has been evolved. Handling is similar to Fortran, except that instead of punching out an object program, the computer stores the final program in a designated section of its memory, where it is ready for immediate use. In this case, the data is fed to the machine right after the Gotran source program, and the answers are typed out as before. One obvious disadvantige of this system is its one-shot nature. When the memory is erased, the program is lost. Another disadvantage, which hasn't been mentioned before, is that in Gotran, there can be no more than one arithmetic operation per statement. Thus "C=A+B" is acceptable in Gotran, but "A=(B*C)/D" is not. The latter statement is quite valid in Fortran, which is not subject to the "one arithmetic operation" rule, but in Gotran, it would have to be solved in two steps: "X=B+C" and "A=X/D". This is cumbersome with complicated expressions and as a result major problems are relegated to Fortran.

In case the artsmen have not had enough, or the engineers and mathematicians want more, I'll give a few of the technical details of the system. The machine is a stored-program, high-speed electronic computer with variable word length (which means that it can handle numbers up to several thousand digits long). Using ten-digit numbers, it will perform addition and subtraction in less than two milliseconds, multiplication in 18 milliseconds, division in 60, and data transfer in about ½ millisecond. It has a magnetic-core mem-

(Continued on page eight)



UP AND OVER — Dalhousie's new electronic brain is hoisted into the Dunn Building penthouse. It is an IBM 1620 data processing system.



MISS JUDITH HUNTER operates the IBM 1622 read-punch unit. Computer with typewriter is at left.