THE DUECE MATRIX INTERPRETIVE SCHEME

by

R.G. Smart
School of Electrical Engineering
N.S.W. University of Technology

INTRODUCTION

This paper describes the General Interpretive Programming Scheme, "G.I.P.", which is available for use on the English Electric DEUCE type computer. The scheme was devised at the National Physical Laboratories at Teddington, England and has been added to by the various DEUCE users. Other similar schemes have been devised for this machine but these will not be described here. The UTECOM computer at the N.S.W. University of Technology is a DEUCE machine and considerable use has been made of this scheme in Sydney. The UTECOM staff has made only a few contributions to the scheme and this paper is certainly not intended to be an original account but rather a description, from a users point of view of an interpretive scheme which has been in use for almost two years and has proved very effective for certain classes of calculations.

INTERPRETIVE SCHEMES

Interpretive schemes have been devised to reduce to a minimum the effort and time required in preparing a programme for a new calculation. A library of subroutines for types of calculations frequently carried out is accumulated for most digital machines which have been in operation for some time. Although these library routines effect an enormous saving in programming time, the use of an interpretive scheme (such as G.I.P.), offers the additional advantage that entire programmes can be written in a special code each instruction of which replaces perhaps hundreds of ordinary instructions. The use of interpretive schemes thus reduces programming to a very simple form so that ideally programmer training time is a minimum, programme preparation time is a minimum and the number of programming errors is minimum. From another point of view, many more programmes are worth writing and many more programmers are worth having.

DIFFICULTIES OF INTERPRETIVE SCHEMES

Increased computing time and reduction of available storage capacity are the prices paid for the use of interpretive schemes, although the extent of these disadvantages depends on the scheme and the purpose for which it is being used. In some cases these difficulties are insignificant and in many more cases they are far outweighed by the advantage of the scheme. The more general an interpretive scheme is, the more space will be occupied by instructions which are redundant in a particular application of the scheme. Similarly, as more facilities are provided to simplify the use of the scheme, for example automatic checks on accuracy and inconsistancies and automatic control of the position of the binary point, so the computing time is increased over the time required to compute the same job by a specially written programme.

THE RELEVANT FEATURES OF DEUCE

The DEUCE type computer has 402 words of Mercury Delay line storage and 8192 words of magnetic drum storage, distributed over 256 circumferential tracks holding 32 words each. The drum operates as an auxiliary store and all transfers to or from the drum are made via one of the 32 word delay lines. Information is transferred to or from the drum in blocks of 32 words each. Each block transfer requires a minimum of 12 milliseconds and a maximum of about 50 milliseconds, but of course the remainder of the computer (i.e. all but one delay line) may be used while drum transfers are in progress.

THE G.I.P. SCHEME

The General Interpretive Programming scheme is based on a programme (referred to as G.I.P.) capable of performing the following operations :-

1. Read in and store on the drum a number (up to 31 in a batch) of separate programmes referred to as "bricks".

2. Read in and store groups of 32 instruction codes.

3. Select these instruction codes in order, interpret them and see that the indicated operations are carried out, using the programme bricks stored on the drum.

The assembly of instruction codes constitutes the programme which must be prepared for the particular calculation on hand. Each code specifies three addresses and the operation which is to be carried out. Each different kind of operation requires a different programme brick in general (there are a few exceptions) but of course only one copy of a brick is stored on the drum even though that brick may be used many times in the same calculation.

Once the G.I.P. routine and the necessary bricks are stored on the drum, the computer is converted into a three address machine with a very elaborate "arithmetic unit", such that one G.I.P. instruction codeword takes the place of up to several hundred ordinary instructions. Very complicated calculations can thus be programmed with relatively few instructions, each of which has a simple three address form. The G.I.P. scheme includes elaborate facilities for counting iterations and modifying instruction codes, and if necessary, ordinary DEUCE instructions can be included in with the G.I.P. instruction codes giving very considerable flexibility to the scheme.

ORGANISATION OF A G.I.P. CALCULATION

Before a calculation is commenced the G.I.P. routine and the necessary programme bricks are stored on the drum and the programme codes are read in. A copy of the G.I.P. routine transfers from the drum to the delay line store and selects the first codeword. The three addresses from the codeword are planted ready for use by the brick called for by the function part of the codeword. A copy of this brick is then transferred from the drum into the delay line store and carries out this step in the calculation, on the data referred to by the three addresses. When this operation is completed, control is restored to the G.I.P. routine which again enters the delay line store displacing the brick. The second codeword is then selected and the process repeated with another brick for the second step in the calculation, and so on.

Codewords which are associated with conditional transfers, instruction modification etc., are obeyed by the G.I.P. routine itself but all operations involving data are carried out by one of the bricks.

APPLICATION TO MATRIX ALGEBRA

The G.I.P. scheme provides a very effective method of performing calculations which can be expressed in terms of matrix algebra. There is no significant limit to the number of instructions each brick contains, making it very convenient to have one brick for each matrix operation; multiplication, addition, etc.. If necessary the brick can have several sections which are brought down from the drum in sequence. The bricks are programmed to include checks against programmer, operator and machine mistakes so that there is little possibility of errors going undetected. Given the calculation as a sequence of matrix operations, the programming can be done very quickly indeed. Iterations and "loops" in the flow diagram for the calculation offer no difficulty.

BLOCK STORAGE OF DATA

One simplifying feature of the scheme is the facility of manipulating the data in large blocks. With each block of data are four parameters :-

1. the number of rows, and

2. the number of columns into which the data is considered to be arranged,

3. the number of binary places to which all the data is given (i.e. a constant multiplier),

4. a check sum of all the elements in the block and including the three above parameters.

Each block of data can thus be treated as an array of rows and columns constituting a matrix. One codeword is sufficient for example to read in one block, regardless of its size and configuration, similarly for punch out and of course the matrix operations. This manipulation of data in blocks is a feature of the individual bricks which have been written.

STORAGE ADDRESSES

All data is stored on the drum and only brought into the delay lines during an operation by a brick. About 175 tracks each of 32 words capacity are available for data, the exact figure depending on the amount of programme stored. Successive elements of a matrix are stored on the drum filling track after track if necessary. The elements are preceded by the four parameters for the matrix and the address is the first track of storage. Thus, a matrix transferred to address 100, would occupy track 100, 101, and so on as required by the number of elements it contained.

The codewords refer to the matrices by their addresses and make no reference to the size of the matrix since this information is carried with the matrix itself on the drum.

THE CODEWORD

The G.I.P. Codeword has four parts, three addresses referred to as "a", "b", and "c" and a reference number "r" to the required operation. It would be quite impossible and unnecessary to store all the library bricks on the drum at once. Only the bricks required for the particular calculation to be programmed, need be read in. These are simply numbered in the order in which they are supplied to the reader and stored away, and this number used as a reference to the brick and the corresponding operation.

In some operations, for example read in and store a matrix, all three addresses a, b and c are not required. The read in brick would thus ignore "a" and "b" and store the matrix at address "c" on the drum. For matrix multiplication, addition and other operations of this kind, "a" and "b" specify the addresses of the operands and "c" the address of the result.

If more than three addresses are required, as for example in specifying the extraction of a sub-matrix, three more auxiliary addresses can be provided by the previous codeword.

EXAMPLE OF A G.I.P. PROGRAMME

The following part of a calculation illustrates the method of using this scheme :-

Given two row vectors "L" and "C" each of 100 elements and a scalar number "w" find and store the row vector :-

$$x = (wL - \frac{1}{wC}),$$

There are seven steps in the calculation requiring seven codewords. Note that the codewords have four parts, three addresses "a", "b" and "c" and a brick reference number "r". The usual convention with the addresses is that "a" and if necessary, also "b" refer to the operands and "c" is the address of the result of the operation.

There are five bricks needed for this calculation and these are numbered in the order in which they are read in and stored on the drum. This reference number is used for "r" in the codewords.

1. Read and store a matrix at address "c".

2. Multiply the matrix at address "a" by the number in the special scalar store and place at "c".

3. Reciprocal of each element of the matrix at "a" and place the resultant matrix at "c".

4. Read in a number to the special scalar store.

5. Subtract the matrix at "b" from the matrix at "a" and place the result at "c".

All unused addresses can be made zero. The codewords are now given with an explanation of each.

Codewords				Description
a	b	c	r	
0	0	1	1	Read in "L" and store at 1
0	0	5	1	Read in "C" and store at 5
0	0	0	4	Read in "w" to the special scalar store
1	0	9	2	Scalar multiply "L" by "w" and place at 9
5	0	13	2	Scalar multiply "C" by "w" and place at 13
13	0	17	3	Reciprocal each element of "wC" and place at 17
9	17	21	5	Subtract "$\frac{1}{wC}$" from "wL" to give "X" at 21.

This simple programme involves no transfers of control or counters. When these facilities are required, they are selected by special values of "r".

EXTENSIONS TO THE SCHEME

It is a straight-forward process to extend the facilities available in the scheme by writing new programme bricks to carry out the new operations. The scheme is designed in such a way that each brick may use practically all the delay line store and can be written as an ordinary programme except for a few instructions at the end of the brick to re-enter the G.I.P. routine.

The scheme is not in any way limited to calculations which can be expressed in terms of matrix algebra, although its widest application has been to work of this kind. The scheme is best suited for calculation which can be performed as a series of fairly lengthy steps on blocks of data because it is designed to allow several hundred instructions per brick and in addition, data is transferred to and from the drum at each step in blocks of 32 numbers. A considerable number of drum transfers are required for each codeword and the total time occupied by these is of the order of $1^1/3$ seconds per codeword.

This means that when the scheme is used on work where less than a few seconds calculation occurs at each step, the "red tape" time of the scheme becomes an appreciable part of the total computing time. For calculations involving more lengthy steps (as in large matrix operations) howevar, the scheme is highly efficient.

AVAILABLE FACILITIES

The following are some of the operations for which bricks are available at present. With few exceptions, these operations require only one codeword each.

Read in and store a matrix.
Punch out a matrix.
Matrix Addition and Subtraction.
Matrix Multiplication.
Multiply a matrix by a scalar.
Form scalar product of two vectors.
Extract a sub-matrix (2 codewords).
Extract the major diagonal of a matrix.
Compound two matrices by rows or columns.
Pre-multiply or post-multiply a matrix by a diagonal matrix
(stored in compact form).

Transpose a matrix.
Square root each element of a matrix.
Reciprocal each element of a matrix.
Sine or Cosine each element of a matrix.
Log of each element of a matrix.
Extract an element of a matrix for use as a scalar.
Read in a number for use as a scalar (there is a special scalar store).
Invert a matrix (3 codewords).

The remaining bricks include a group for determining latent roots and vectors.

SUMMARY

The DEUCE General Interpretive Scheme provides a very simple method of programming matrix and similar calculations by a three address code where each code carried out a complete matrix operation. The facilities are easily extended and the scheme has been used on a large number of calculations, not normally considered to be matrix calculations. The scheme is highly efficient for operations on large matrices, but for many other purposes its simplicity of use justifies any extra computing time. The scheme has proved very successful and has been in use for nearly two years.

DISCUSSION

Dr. D. Fenna, W.R.E.

What would you suggest is an efficient lower limit for the size of matrices, so that it is worth using an interpretive system?

Mr. R.G. Smart (In Reply)

The upper limit of a matrix that can be handled with a 5000 word store is about 40 x 40, assuming that several matrices are in use at the same time. For a lower limit I would suggest a matrix of 100 terms. Although this is not very efficient, you save a lot of programming effort.

Dr. D. Fenna, W.R.E.

Do you check, before you bring a subroutine brick down, that it is not the one you are already using? If it is, there is no need to bring it down again.

Mr. R.G. Smart (In Reply)

No. It is a waste of time, but as the system is at present there is nothing that can be done about it.

Dr. T. Pearcy, C.S.I.R.O.

Is the structure of an interpretive order the same as an ordinary machine order?

Mr. R.G. Smart (In Reply)

No, it is entirely different.

Mr. B.A. Chartres, University of Sydney.

Are there any fault-finding techniques incorporated in the programme?

Mr. R.G. Smart (In Reply)

The scheme is so easy to use that we seldom have mistakes, and if we do they are easy to find. There are a number of checks built in - sum checks on rows and columns, checks on compatibility of matrices for multiplication and addition etc.

Mr. R.H. Merson, R.A.E.

It seems to be this programme still makes you do too much work. You still have to check how many numbers you are operating on, and make sure one matrix does not overwrite another. Surely the machine could do some of this sorting work, for example by keeping track of where the last matrix finished, and writing the next one after it.

Mr. R.G. Smart (In Reply)

You normally want to overwrite a matrix after you have finished using it. You would need some way of specifying when you had finished with each matrix.

Dr. D. Fenna, W.R.E.

What is the efficiency of putting numbers into the tracks consecutively? Would you not lose efficiency on smaller matrices by having one row or column stored in consecutive tracks, when it might have fitted in one track?

Mr. R.G. Smart (In Reply)

That is something that might be done.