INTERPRETIVE PROGRAMMING

by

I.M. Bassett

University of Melbourne

I am going to give what is essentially a very elementary account of interpretive methods, by describing two interpretive schemes which have been used on C.S.I.R.A.C. They are suitable for floating point, double precision or complex arithmetic. Some general features, some virtues and vices, of interpretive methods, will thereby be illustrated.

Pearcey and Hill worked out the first interpretive method for C.S.I.R.A.C., and I will describe this first. As an introduction to this description, I will remind you of some of C.S.I.R.A.C.'s features. It is a binary machine, and the word-length is 20 digits. It is provided with 16 registers, named D_0, D_1, ...D_{15} into which addition, subtraction and transfer can be carried out. Further, it has 3 registers, A, B, and C, which are concerned with multiplication. Registers A and C can also be used for addition and subtraction and various other operations. Numbers are represented in sign and complement form, the first digit giving the sign. The built-in operations of addition, subtraction, and multiplication are designed to give the right answer, roughly speaking, if the patterns of 20 digits are regarded either as fractions with the binary point on the left, immediately following the sign digit, or as integers, with the binary point on the extreme right. Unless special provision is made in the programme, therefore, only fractions of absolute value less than one, or integers, can be represented. Now any number (except zero) can be represented uniquely in the form $x\,2^p$ where $\frac{1}{2} \leq |x| < 1$ and p is an integer. It may be convenient to represent numbers in this form - that is in floating point form - by two 20-digit words, in adjacent positions in the store, the first containing the fraction x and the second the integer p. In this case, such processes as transfer of a number from one location to another, addition, or multiplication of two numbers, can no longer be performed by single machine orders. On the contrary, transfers and arithmetical operation require a block of orders. The standard way of introducing such a block of orders into a programme is to have the block arranged as a sub-routine. A sub-routine for the addition of two numbers in floating-point form would produce the sum of the contents of definite pairs of registers and put the answer in a definite pair of registers - say it forms the sum of the numbers in A, B and D_0, D_1 and puts the answer in A, B. The subroutine is terminated by an order which directs control to whatever number is in a particular D register, D_{15} say. In order to add together the numbers in store locations (0,1) and (2,3) and put the answer in (0, 1) the following orders in the main programme would be necessary :

contents of store location	0	transfer to	D_0
"	1	"	D_1
"	2	"	A
"	3	"	B

transfer current control number to D_{15}

transfer control to first order of subroutine

On completion of its task, the subroutine adds one to D_{15}, and then transfers control to the number in D_{15}, i.e. transfers control back to the appropriate position in the main programme. The remaining orders required are then

contents of	A	transfer to store location	0
contents of	B	transfer to store location	1

Thus the simple process of adding one number to another requires eight orders in the main programme. It is convenient for the programmer to be able to regard such processes as single steps, to get rid of the undergrowth and get a good view, at least, of the trees. To do this, a special code is devised, in which such processes correspond to single orders. Corresponding to this code, a decoding routine is devised. After the decoding of an order in the special code - such an order will be called a hyper-order - the transfers required for the preparation for the desired arithmetic operation are performed, and then the arithmetic operation itself.

In the interpretive code of Pearcey and Hill, the first ten of the 20 binary digits of a hyper-order are an address, the next five a function number and the last five a label to distinguish orders which are to be performed interpretively from orders which are to be performed as ordinary machine orders. This label is simply a pattern of digits in the last five positions which does not occur in any ordinary machine order. It is a one-address code. Let us consider for example how the addition of the numbers in cells (0,1) and (2,3) would be coded and carried out. First, an order bringing about the transfer of the contents of (0,1) to a particular location, say A,B.
This transfer order would be coded

0 t Z

Z stands for the label. t stands for the code number for transfer to AB. 0 is the address of the store location (actually the first member of the pair of store locations) from which transfer takes place. Then an order

2 a Z

where a is the code-number for addition into AB. Finally an order

0 t' Z

where t' is the code-number for transfer from AB to store. The cycle of processes by which such a hyper-order is carried out may be illustrated by the second one. The store location of the hyper-order which is currently being performed is kept in a particular D register, D_{14} say; first, one digit is added to D_{14}, and the hyper-order now to be performed extracted from the store location indicated by the number in D_{14}. The order is examined for the Z-label. As it has a Z-label, it is then treated interpretively. The address part is examined and then the contents of the appropriate store locations, namely 2 and 3, are placed in D_0, D_1. Then the function code-number is examined, and control directed to the appropriate sub-routine - namely that which forms the sum of D_0, D_1 and A,B in A,B. Finally, control is directed back to the beginning of the cycle, the point at which one is added to D_{14}. Besides addition, and transfer into and out of AB, the code provides for subtraction, division, multiplication and square root.

Besides these hyper-orders for transfer and arithmetic operations, there are certain ones which have to do with bringing about a change in the normal sequence of performance of hyper-orders. As we have seen, after an arithmetic or transfer hyper-order, the next hyper-order done is the one following in the store. This is the normal sequence. A hyper-order is provided which has the effect that the next hyper-order is taken from store-location n, where n can be chosen at will. Other hyper-orders are provided which are concerned with counting, and with switching on completion of a count. There is a hyper-order for varying the effective address of the following hyper-order. There is provision for linking to and from subroutine in a hyper-programme. One last facility provided in this interpretive scheme must be mentioned. In the course of a programme written interpretively, one wants as a rule some ordinary machine orders. It is for this reason that the Z-labelling is used. An order which is not labelled Z is put into a certain memory location, m say; real control is transferred to m, with the result that the order is carried out as an ordinary machine order.

I shall now describe another interpretive method which has been used on C.S.I.R.A.C. and which differs from the one I have just described in some respects. It dispenses with the labelling of hyper-orders. The alternative to the labelling of hyper-orders and the Z-testing of all orders is to place in the programme as needed orders which indicate a "change of language". If one wants some hyper-orders at the end of a batch of real orders then the last two real orders, in locations m, m + 1 say, should be

store number m in D_{14}

transfer control to first order of interpretive block

Subsequent orders are performed interpretively - are regarded as being in "hyper-language". At the end of every operation directed by a hyper-order, control is directed back to the first order of the interpretive block. This cycle is broken only by the special hyper-order.

change to real language

which results in transfer of control to the store location indicated by the number in D_{14}.

This feature of the second interpretive scheme, the replacement of the label by a "change of language", is clearly a disadvantage in one way, in that orders must be put into the main programme to bring about such changes of language. However, at least in many programmes, few such changes of language are required. Its advantages are (i) that ordinary machine operations are performed about 5 times faster, not requiring to be tested for the label, (ii) that it appears to be practicable to carry out changes of control, counts and switches in real language, thus saving time, and space in the interpretive routine, (iii) it multiplies the number of possible hyper-orders which can be written by a factor of 32.

There is another respect in which the second interpretive method increases the amount of useful information which can be put into one 20-digit hyper-order. It is possible to have, in effect, an n-digit address in a space of less than n digits. In this interpretive code, for example, two addresses each ranging from 0 to 1023 are effectively represented in a space of only 10 binary digits i.e. in a space which is, strictly speaking, only sufficient to represent one address ranging from 0 to 1023. This can be done because it appears that, in practical programming, the addresses of the source and destination vary only slowly from order to

order. The orders contain only the least significant half of their addresses, the most significant half being given by a prior "setting" order. Because the addresses vary only slowly, one such setting order will do as a rule for a whole block of consecutive orders. The setting order can also be used to make the following order or orders into orders of variable address. Let us consider an example. To calculate the distance between two points whose Cartesian coordinates are in locations which we may represent as 100, 101 and 102, and 250, 251 and 252, the following orders will do: (the notation is transparent)

setting order			(250, 100)
0	$\xrightarrow{-}$	0	Δx
1	$\xrightarrow{-}$	1	Δy
2	$\xrightarrow{-}$	2	Δz
setting order			(100, 100)
0	$\xrightarrow{x}$	0	Δx^2
1	$\xrightarrow{x}$	1	Δy^2
2	$\xrightarrow{x}$	2	Δz^2
1	$\xrightarrow{+}$	0	
2	$\xrightarrow{+}$	0	
$\sqrt{0}$	$\longrightarrow$	0	$\sqrt{(\Delta x^2 + \Delta y^2 + \Delta z^2)}$

The principle of economy used here is the same as that used by the Melbourne newspapers in the publication of the examination numbers of passes - only the less significant figures are given, except when the more significant ones change.

Any given interpretive programme could be replaced by a direct programme by more complex design of the machine. For example there would normally be no need to use double precision interpretive programming on C.S.I.R.A.C. if the word length were 40 instead 20 digits. Nevertheless, most machines seem to find use for floating-point routines. And of course, for any given machine, more elaborate ways of operating can be thought of, so that interpretive methods are required to make them effectively operate in this way - for example that matrix operations are accomplished by single orders.

Interpretive methods waste machine time - by a factor which varies, but about 5 is suggested for the sort of interpretive programming considered in this paper. On the other hand, interpretive methods may save an enormous amount of time in programming, and in some cases save store space.

DISCUSSION

Dr. J.M. Bennett, University of Sydney.

Do you write interpretive orders in the same form as machine orders, so that they can be read in by the initial orders?

Mr. I. Bassett (In Reply)

It is quite different from machine code.

Dr. J.M. Bennett, University of Sydney.

Does that mean you have to have a special input routine?

Mr. I. Bassett (In Reply)

No, you have to keep a cool head when you are punching it.

Dr. J.M. Bennett, University of Sydney.

Does that mean you punch direct in binary?

Mr. I. Bassett (In Reply)

Yes.

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

We use 12 hole binary tape on C.S.I.R.A.C. That simplifies input.

Mr. B.Z. de Ferranti, I.B.M.

Do these interpretive routines help in teaching. This is a question to both speakers.

Mr. R.G. Smart, University of Technology, Sydney.

The students find the interpretive scheme is very much easier to learn. You have less to remember, all your data is in one place, and you achieve much more with each instruction.

Mr. I. Bassett (In Reply)

I agree with that, - it is very much easier to use.

Professor T.M. Cherry, University of Melbourne.

With regard to the question of efficiency, the loss of efficiency depends on the kind of work you are doing. For example in the matrix case, provided you do the organisation in machine code, the amount of time lost is not great. In the algebraic case it would be larger.

Dr. J.M. Bennett, University of Sydney.

I endorse those remarks.You have to be very careful in using an interpretive scheme to make sure you are not invoking the whole machinery of interpretation to add one to a counter. In a programme involving a lot of bookkeeping, that bookkeeping must be done in machine code.