Some Techniques for dealing with Two-Level Storage

by R. A. Brooker

Summary: This paper describes some devices used in connection with Mercury Autocode to overcome the difficulties of two-level storage, particularly for manipulating matrices stored on the drum.

INTRODUCTION

Almost all digital computers employ multi-level storage in some form as an economical means of realizing large storage capacities. In most cases there are just two levels: an immediate-access store (the *fast* store) supplemented by a much larger *auxiliary* store, which is slower of access but very much less costly (word for word) than the fast store. In Mercury the immediate-access store is a 1,024-word (40 bits) ferrite-core matrix with an access time of 10 μsec. The auxiliary store is provided by a 16,384-word magnetic drum which rotates at 3,475 r.p.m., corresponding to a revolution time of $17\frac{1}{4}$ msec. The storage locations on the drum are arranged in 256 *tracks* each of 64 words, and each track consists of two half tracks, or *sectors*, of 32 words. These are numbered 0, 1, 2, . . ., 511, all the even-numbered sectors (and likewise all the odd) occupying the same azimuthal position on the drum. The fast store is partitioned into 32 *pages* each of 32 words, and provision is made to transfer the contents of any sector to any page, and vice versa. Transfers from the drum to the fast store are known as *reading* transfers, and those in the reverse direction as *writing* transfers. In either case the operation lasts for half a revolution of the drum from the time at which the transfer proper begins. Since the average waiting time is also half a revolution, the total transfer time for randomly selected sectors is a complete revolution of the drum. There is no delay, however, when a transfer is followed immediately by another transfer of opposite parity, i.e. an odd sector followed by an even sector, or vice versa. Thus if a group of consecutively numbered sectors are transferred in rapid succession, then all except the first will take half a revolution each. If all 32 words of a sector can be used, then the access time to the drum is 0·54 msec (1/32 of the revolution time) per word for individual transfers, and correspondingly less if more than one sector is involved. This should be compared with the times of other operations, e.g.

transfers to and from the accumulator	120 μsec
addition and subtraction	180 μsec
multiplication	300 μsec
red-tape instructions (counting, etc.)	60 μsec

It can be seen that if transfers are confined to suitably large sets of words (for example, routines, vectors, etc.), and some at least of these are used repeatedly, then from the time-economy point of view these arrangements may compare favourably with a single fast store of indefinite size. The capital saving can be appreciated when one compares the cost of the two forms of storage: between £2 and £3 per word for cores, and approximately 2s. 6d. per word for the drum. (These figures are slowly changing with time but appear to maintain the same ratio.) It should be emphasized, however (and this is often overlooked by machine designers), that while it may be theoretically possible, it is not always convenient to arrange one's program to take advantage of two-level storage, and this may prove a severe handicap to an inexperienced programmer. Indeed it has been argued that the resulting increase in programming costs more than offsets the cost of a large one-level store!

An example of the difficulties which may arise is provided by the serial Jacobi process (Goldstine, Murray and Von Neumann, 1959) for the determination of the latent roots and vectors of a symmetric matrix which is too large to be contained in the fast store. In its simplest form, this demands rapid access to both rows *and* columns of a matrix, which is not possible with conventional methods of recording matrices on a drum, that is by rows *or* columns.

In the rest of this article we shall describe some of the strategies adopted in connection with the Mercury Autocode system (Brooker, 1958, 1959) for alleviating the difficulties inherent in the form of two-level storage just described.

THE GENERALIZED TRANSFER INSTRUCTIONS

The first problem was to provide the Autocode user with some convenient means of access to data* stored on the drum. Clearly the basic tansfer instructions involving blocks of 32 words would be out of place in an automatic-programming system. Instead, Autocode treats the drum (or that portion of it occupied by numerical data) as a consecutive series of locations numbered 0–10,751 (sector n corresponding to the addresses $32n$ to $32n + 31$).

Pseudo-transfer instructions have been adopted which can transfer any consecutive set of words on the drum to any consecutive set of locations in the fast store, and vice versa. To do this the user specifies the first location on the drum, the first location in the fast store, and the number of words to be transferred. Thus, for example,

$$\phi_6\ (1{,}000 + 20i - 20)\ a_0,\ 20$$

transfers the ith row of a 20 $\times$ 20 matrix stored by rows (see next section), starting at location 1,000, to the fast

* The question of instructions does not arise since this is taken care of by the chapter-changing scheme.

Downloaded from https://academic.oup.com/comjnl/article/2/4/189/470600 by guest on 19 April 2024

store locations $a_0, a_1, \ldots, a_{19}$. The corresponding writing operation is ϕ_7.

These pseudo-transfer operations are effected by means of a small routine (36 instructions) kept permanently available in the working store. A further page of the working store is used as a buffer store in connection with these operations. The routine first determines S and T from the relation $a = 32S + T$, where a is the specified drum address, S the sector on which it lies, and T is the address within the sector $(0 \leqslant T \leqslant 31)$. (This is done by shifting and collating.)

In a reading operation the relevant sectors S, $S + 1$, etc., are transferred one at a time to the buffer page, from which the words are copied into their final destination in the fast store. The total time for a transfer by this scheme is $(17{\cdot}25p + 0{\cdot}36n)$ msec, where p is the number of sectors involved, and n is the number of words transferred: p is either $\left[\frac{n-1}{32}\right] + 1$ or $\left[\frac{n-1}{32}\right] + 2$ depending on the positioning of the words relative to the sector divisions.

In a writing operation the words are copied from their original locations in the fast store into the buffer page, and from there transferred to the drum. It is not quite as simple as the reading operation, because of the need to preserve the "flanking" material on the first and last sectors of the group of sectors involved. These sectors are first read down to the buffer page *before* the material to be placed on them is transferred there. There is no need for this with the intermediate sectors because all the material on these sectors will be replaced. The time for a writing transfer is thus $(34{\cdot}5 + 0{\cdot}36n)$ msec, if only one sector is involved, and $[17{\cdot}25(p + 2) + 0{\cdot}36n]$ msec, if more than one sector is involved.

THE MATRIX OPERATIONS

To make the drum still more useful to the Autocode user, the pseudo-transfer operations are supplemented by a set of instructions (ϕ_8 to ϕ_{28}) for manipulating matrices recorded on the drum, and for the input and output of material direct to the drum. These assume the matrices to be stored by rows, that is the elements in any row stand in consecutive locations, with the first element of each row following the last element of the previous row. Thus the element $a_{i,j}$ stands in the location $a' + n(i - 1) + j - 1$, where $i = 1(1)m$ and $j = 1(1)n$. The first element of the first row stands in a' which will be referred to as the *location* of the matrix. An example of a matrix operation is the multiplication instruction

$$a' = \phi_{26}(b', c', u, v, w) \text{ corresponding to } A(u \times v) = B(u \times w)C(w \times v).$$

Here a', b', c' are the locations of A, B, C, and u, v, w their dimensions. A complete list of the matrix instructions can be found in the Autocode Manual (Brooker, Richards, Berg and Kerr, 1959). In the remaining section we describe some of the methods used in the actual routines which perform these operations.

THE "WINDOW" DEVICE

The generalized transfer operations provided for the Autocode user are unsuitable for use behind the scenes where more efficient techniques are called for. Instead, five special transfer operations were developed whereby the programmer can look at successive rows of a matrix recorded on the drum, through a "window" consisting of two or more pages of the fast store. A somewhat similar scheme has also been suggested by Robertson (1957). These pages correspond to the sectors over which the row extends. The standard size of a window is five pages (although this could be altered by means of a preset parameter in the routine), and is associated with a directive of the form

$$a \rightarrow 159$$

which allocates the 160 variables $a_0, a_1, \ldots, a_{159}$ to five pages of the fast store. Also associated with each window is a 40-digit code word consisting of four 10-bit parameters describing the relationship of the material in the window to that on the corresponding sectors of the drum. The five transfer operations associated with a window are as follows.

1. *Opening the window.*

This is denoted by $\phi_1(a', a, p)$,
where a' is an arbitrary drum address,
a is the variable to be employed as code word,
p is the first page of the window.

As in the case of ϕ_6 and ϕ_7 the routine first finds S and T from the relation

$$a' = 32S + T,$$

S being the sector on which a' lies. This is transferred to page p and the code word set up as follows:

p	S
p	T

2. *READ*, denoted by $\phi_2(a, i, n)$.

This is for use after the window has been opened (or following another ϕ_2 instruction). The effect is to transfer the next n words from the drum, i.e. those standing in $a' + 1, a' + 2, \ldots, a' + n$, to n consecutive locations in the window. The index i is adjusted so that if the associated directive is $x \rightarrow 159$ these locations can be referred to as x_i, $x_{(i+1)}$, etc.

The mechanism of the operation is as follows. Assume the window has just been opened. Then unless $T = 31$

Downloaded from https://academic.oup.com/comjnl/article/2/4/189/470600 by guest on 19 April 2024

some of the next n words (and possibly all of them) lie on the sector brought down to page p. The ϕ_2 routine then brings down the necessary number of sectors $S+1, S+2, \ldots, S+k$ to pages $p+1, p+2, \ldots, p+k$, to complete the transfer. The code word is adjusted thus:

$p+k$	$S+k$
p	$T+n$

$$k = \left[\frac{T+n}{32}\right]$$

and the index i is set to the value $T+1$, the address of the first word required. Here $S+k$ is the last sector transferred and $p+k$ the corresponding page: $T+n$ is the location of the last element required. At the start of any subsequent operation these quantities will be referred to as S, P, and T respectively.

Subsequent applications of this operation cause the code word to be adjusted in a similar fashion until a case arises where $T+n \geqslant 160$ (the limit for a 5-page window). In this case the last sector (S) transferred on the previous occasion is brought down again to the first page p, and the subsequent sectors to pages $p+1$, $p+2$, etc. (unless $T = 31 \bmod 32$, in which case sectors $S+1, S+2$, etc., are transferred to $p, p+1$, onwards).

This operation will be referred to as *resetting* the window. The code word and index are adjusted as follows:

$T \neq 31 \bmod 32$

$p+k$	$S+k$
p	$T \bmod 32 + n$

$$k = \left[\frac{T \bmod 32 + n}{32}\right]$$

$$i = T \bmod 32 + 1$$

$T = 31 \bmod 32$

$p+k$	$S+k+1$
p	$n-1$

$$k = \left[\frac{n-1}{32}\right]$$

$$i = 0.$$

It is clear that the window technique will be particularly useful when transferring successive rows of a matrix, because there is no waste of flanking material, the elements following the end of one row being part of the next, and so on. The window must, of course, be large enough to accommodate a row of the matrix under all circumstances. The standard 5-page window permits rows of up to 129 elements, which is sufficient for normal matrix work. For $n \geqslant 80$ the resetting operation ($T+n \geqslant 160$) will take place at every stage, and correspondingly less frequently for smaller rows.

3. *READ/WRITE*, denoted by $\phi_3\ (a, i, n)$.

This is used when it is required to alter a row on the drum *in situ*, that is to read it, alter it, and write it back *in the same position*. It is very similar to ϕ_2 and differs only in the case where resetting takes place. In this case the "active" pages $p, p+1, \ldots, P$ are written back to their corresponding sectors before transferring the last sector S to page p (or sector $S+1$ in the special case $T = 31 \bmod 32$). The parameters a, i, n play the same roles as in ϕ_2.

As it turned out, only one of the matrix operations required this type of transfer, namely, the reduction process as applied to the right-hand sides in matrix division, and even here it can be dispensed with by using separate READ and WRITE windows (see next section). It is the WRITE operation which is more frequently used owing to the fact that the matrix Autocode instructions are of the 3-address code variety, namely $A\theta B \rightarrow C$, where C is usually distinct from either A or B.

4. *WRITE*, denoted by $\phi_4\ (a, i, n)$.

Like the previous operations this is intended to be used following a ϕ_1 or another ϕ_4 operation. The effect is to prepare to write the next n words on to the drum. The mechanism is as follows. Immediately after a ϕ_1 operation the code word and index are adjusted as follows:

p	S
p	$T+n$

$$i = T+1$$

(except in the special case $T = 31$ and $n = 129$, which is dealt with below). Material destined for the next n locations on the drum can then be placed in the corresponding locations of the window, x_i, $x_{(i+1)}$, etc.

Subsequently applications cause the code word to be adjusted in a similar fashion until the case $T+n \geqslant 160$ occurs. Two possibilities then arise, depending on whether or not T is the last location of a page, i.e. $T = 31 \bmod 32$. If not the active pages $p, p+1$, etc., are written on to sectors $S, S+1$, etc., stopping short of the page containing the location T, outstanding elements being copied across to the corresponding locations on the first page. The code word and index in this case become

p	$S + \left[\frac{T}{32}\right]$
p	$(T \bmod 32) + n$

$$i = (T \bmod 32) + 1.$$

If $T = 31 \bmod 32$, then all the pages $p, p+1, \ldots, p + \left[\frac{T}{32}\right]$ are transferred to their corresponding sectors.

Downloaded from https://academic.oup.com/comjnl/article/2/4/189/470600 by guest on 19 April 2024

In this case the code word and index become

p	$S + \left[\frac{T}{32}\right] + 1$
p	$n - 1$

$i = 0.$

The operation can be used to write successive rows of a matrix on to the drum.

5. *Closing the window*, $\phi_5\,(a)$.

The purpose of this operation is to restore all outstanding material to the drum. Two cases arise according to whether the window has been used for READ/WRITE transfers or purely WRITE transfers. These can be distinguished by comparing the two left-hand quantities in the code word. If these differ then the window has been used for READ/WRITE transfers; if not it can be treated as if it had been used for WRITE transfers.

Following a succession of READ/WRITE transfers all that is necessary is to transfer the "active" pages of the window to their respective sectors on the drum, i.e. pages $p, p + 1, \ldots, p + k$ to sectors $S, S + 1, \ldots, S + k$.

In the case of WRITE transfers the close operation is done in two steps. First the WRITE operation $\phi_4(a, i, 129)$ is employed to restore all the complete pages to the drum, the parameter $n = 129$ being sufficient to ensure that this will take place for all $T \geqslant 31$. The next step is to deal with outstanding elements which have been copied into page p. The sector corresponding to this page is transferred to page $p + 1$ and the flanking material copied across to page p, which can then be transferred to the drum. The second step can be omitted if $T = 31 \bmod 32$ since there are no outstanding elements to deal with.

THE CENTRAL ROUTINE

The transfer operations as described can be realized by a multi-entry subroutine of less than 100 instructions. The routine currently employed dispenses with the READ/WRITE transfer and extends to only 61 instructions; but this is partly due to another difference: the treatment of the flanking elements in the WRITE operation. The sector $S + \left[\frac{T}{32}\right]$ is transferred to page p before the outstanding elements are copied there. The closure operation can then be completed by simply transferring this page to its appropriate sector on the drum. This is a clumsy device, however, and introduces an extra transfer into the WRITE operation.

The call sequences for the five operations each extend to 7 or 8 instructions.

TIME OF OPERATION

In addition to the time of the magnetic transfers proper, approximately 1·8 msec is spent within the routine and the call sequences, thereby adding an amount (1·8/n) msec to the access time for each element of the row. Thus the scheme is less efficient (but still very useful) for small values of n, in particular for $n = 1$, e.g. column vectors. (This important special case is considered later on.) However, it is not to be expected that a general-purpose routine will be the most efficient routine in any particular case.

The main contribution to the access time, that from the magnetic transfers, is between 0·54 msec and 0·28 msec per element, depending on what proportion of the sectors were transferred individually. Again, for small values of n (less than 32), all the transfers will be individual.

MATRIX MULTIPLICATION

To illustrate the use of the row transfer operations, we give below the outline of a routine to form the matrix product $A(r \times s) = B(r \times t)C(t \times s)$. The matrices are stored by rows, starting at locations a', b', c'. Their dimensions are r, s, t. (The routine also illustrates some of the conventions suggested by the British Computer Society Research Committee on Scientific Programming Notation. See *The Computer Bulletin*, Vol. 3, No. 3.)

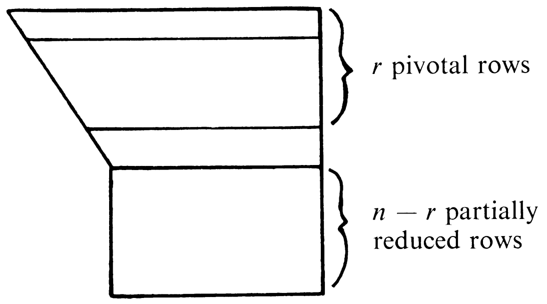
```
$a \to 159$   (pages 16–20) }
$b \to 159$   (pages 21–25) } directives describing window space
$c \to 159$   (pages 26–30) }

$\phi_1(a' - 1, a, 16)$        open "$A$" window
$\phi_1(b' - 1, b, 21)$        open "$B$" window
$l = 1(1)r$
    $\phi_4(a, i, s)$          prepare to write next row of $A$
    $\phi_2(b, j, t)$          read next row of $B$
    $\phi_1(c' - 1, c, 26)$    (re)open "$C$" window
    $n = 1(1)s$                prepare to accumulate
        $a_{(i+n-1)} = 0$      elements of next row of $A$
    $m = 1(1)t$
        $\phi_2(c, k, s)$      read next row of $C$
        $n = 1(1)s$
            $a_{(i+n-1)} = b_{(j+m-1)}c_{(k+n-1)} + a_{(i+n-1)}$
$\phi_5(a)$                    close the "$A$" window
```

A refinement incorporated in the actual program, but not shown here, is a test for the special case where C has less than 160 elements. In this case the entire matrix is transferred to the "C" window by means of a ϕ_6 instruction, namely $\phi_6(c')c_0, q$, where $q = st$, and remains there for the entire process.

Downloaded from https://academic.oup.com/comjnl/article/2/4/189/470600 by guest on 19 April 2024

REDUCTION TO TRIANGULAR FORM

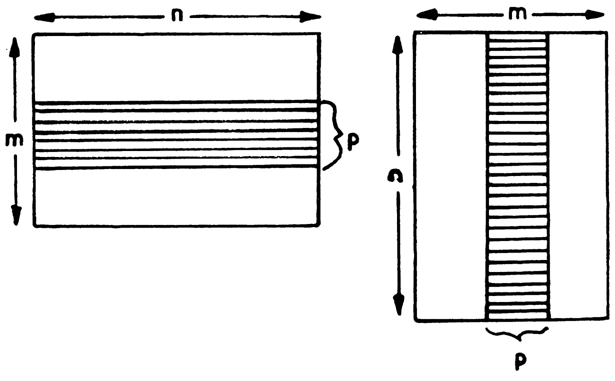
This is the means of evaluating determinants and is part of the process of matrix division. The actual method used is that described by Wilkinson (1953) which employs row interchanges. The storage layout is of interest here. After r reductions there are r pivotal equations and a square array of order $n - r$, thus



The rows are packed end to end on the drum so as to take advantage of the row transfer operations. The number of sectors scanned in each reduction gets progressively less. The ϕ_6 and ϕ_7 instructions are used to effect the row interchanges, which, because they lie outside the inner loop, need not be treated as efficiently as the rest of the calculation. This principle was used extensively in coding the matrix routines.

TRANSPOSITION

As a further illustration of storage strategy we describe very briefly the method used to transpose a rectangular $(m \times n)$ matrix stored by rows in the usual way. As many rows as possible are transferred to the fast store, at the same time leaving space for a single row of the corresponding partition of the transpose. The number of rows is the maximum value of p such that $pn + p \leqslant 480$ (the size of the fast data store).



The window transfers cannot be used here, and the transposed row segments are built up individually in the p (consecutive) spare locations and transferred to the drum by ϕ_7 operations. The method relies on being able to transfer a sufficient number of rows so that the matrix is completed in as few sections as possible. Thus for a 30×40 matrix there are three sections 11, 11, 8. Except for small matrices (more precisely those for which $m(n + 1) \leqslant 480$) the operation does not permit the user to transpose a matrix on top of itself.

SINGLE PAGE WINDOWS

When dealing with one-dimensional arrays of any kind, it is more convenient to call for the elements singly rather than n at a time. It has already been mentioned that the general window routine is less efficient for small values of n, but in the particular case $n = 1$ the window can be confined to a single page, and the central routine becomes very much simpler. As a result the access time due to red-tape operations can be reduced to 180 μsec per element. The transfer operations take precisely the same form as before, and the following example illustrates their use in this case.

Form the expression

$$f = \sum_{t=1}^{N-s} X_t X_{(t+s)}$$

from X's recorded on the drum in locations $x' + 1$, $x' + 2, \ldots, x' + N$ (N being too large for the series to be contained in the fast store).

$a \to 31$	X_t window
$b \to 31$	$X_{(t+s)}$ window
$f = 0$	
$\phi_1(x', a, 16)$	opens X_t window
$\phi_1(x' + s, b, 17)$	opens $X_{(t+s)}$ window
$t = 1(1)N-s$	
$\phi_2(a, i, 1)$	calls for next X_t
$\phi_2(b, j, 1)$	calls for next $X_{(t+s)}$
$f = a_i b_j + f$	

CONCLUSIONS

Operational experience with the ϕ_6 and ϕ_7 instructions has proved them to be very suitable forms of drum transfer instructions for the ordinary Autocode user: they are easily understood and easy to use. The row transfer operations have so far only been used behind the scenes in writing the routines for matrix arithmetic. However, this is only because it is necessary to write out by hand the call sequences for $\phi_1 - \phi_5$ in terms of machine instructions. If it is decided to make these operations generally available then this would be done automatically by including the appropriate "generators" in the program.

The row transfer operations could possibly be used in other problems involving a two-dimensional array, e.g. in the solution of partial differential equations where the finite difference mesh is too large to be contained in the fast store.

Downloaded from https://academic.oup.com/comjnl/article/2/4/189/470600 by guest on 19 April 2024

REFERENCES

BROOKER, R. A. (1958). "The Autocode Programs developed for the Manchester University Computers," *The Computer Journal*, Vol. 1, p. 15.

BROOKER, R. A. (1958). "Further Autocode Facilities for the Manchester (Mercury) Computer," *The Computer Journal*, Vol. 1, p. 124.

BROOKER, R. A. (1959). "Mercury Autocode: Additional Notes," *The Computer Journal*, Vol. 2, No. 1, April 1959, p. xi.

BROOKER, R. A. RICHARDS, B., BERG, E., and KERR, R. (1959). "The Manchester Mercury Autocode System," The Computing Machine Laboratory, University of Manchester, May 1959.

ROBERTSON, H. (1957). "Cyclic Use of the High-Speed Store," *C.E.R.N. Report* (European Organization for Nuclear Research, Geneva).

GOLDSTINE, H. H., MURRAY, F. J., and VON NEUMANN (1959). "The Jacobi Method for Real Symmetric Matrices," *J. Assoc. Comp. Mach.*, Vol. 6, p. 59.

WILKINSON, J. H. (1953). "Linear Algebra on the Pilot A.C.E.," *Proceedings* of a Symposium held at the National Physical Laboratory, March 1953. London, H.M.S.O., 1954.

Book Review

Proceedings of the Canadian Conference for Computing and Data Processing. 383 pages. (Toronto: University Press, $5.00; London: Oxford University Press, 40s. 0d.)

The first Canadian Computing Conference, held at the University of Toronto in June 1958, was organized largely on the initiative of the Toronto University Computation Centre. Its object was, apparently, to review developments in the Canadian computing field, since the Centre was established a decade ago, and to give some picture of the current situation. The degree to which the Conference succeeded may be judged from these the published Proceedings.

There are several informative and interesting papers on data processing in banking, life insurance, the National and C.P. Railways, the Ontario Highways Department, the Government Service and the aircraft and oil industries. There is a paper on the application of computers to Canadian business forecasting, which summarizes "improvements in statistical theory that have stemmed from computer application," for example in recognizing the inadequacy of the twelve-month moving average; and there is an after-dinner address by the Deputy Minister of Economics of the Ontario Government, who expressed the hope that "by taking more of the guess work out of business economic dislocations may be discerned sooner and corrective action taken in time."

It is, however, a little difficult to understand why it was necessary to include a number of contributions, which, while competent in a condensed way, are essentially introductory in character (e.g. "Fundamentals of Computers," "Elements of Programming," "Character Representation and Storage," etc.), alongside others of greater interest and of a much more restricted and technical character, dealing with systems optimization, multiple and orthogonal regression, self-consistent field theory, crystal structure, and large-scale matrix calculations. Especially incongruous is a blatant sales-talk type of paper on the Burroughs E 101, which contrasts strongly with a useful survey of the problems of high-speed printing and two informative papers on automatic programming, one from Remington Rand, the other from I.B.M.

Probably the "registrants" included both those with considerable computer experience and others attending to find out "what it's all about." This is always a problem confronting the organizers of general conferences on computing, although it may be partially overcome by running parallel sessions. The solution is, maybe, to limit conferences to relatively restricted topics and to run information lecture courses for potential users and others wanting a less specialized approach.

There is a further disadvantage of such "all-in" conferences: often even the most informative and more original contributions are forced to be somewhat sketchy and less detailed than one could wish them to be. It is very aggravating to be told that such and such a procedure was successfully programmed but to be given little or nothing of the problems and difficulties encountered or of the program itself. And, of course, there is the inevitable danger of repetition. Although it is difficult completely to avoid this in any actual conference, there is much to be said for some judicious sub-editing, on the one hand, and expansion on the other, in the published reports.

One feature of the Toronto Conference must be stressed: the very clear fact that the University there is well aware of its responsibility not merely to train analysts, programmers and computer engineers, but also to "study the theoretical ideas that are relevant to a broad understanding of the whole business" (W. H. Watson, "On learning to do better.") In this connection, the paper on Computer Education in Canadian Universities is of considerable interest. Here an attempt is made to assess Canada's computer manpower requirements for 1960 and the probable student output from which they will be, partially, met—"our present curricula and enrolments are extremely inadequate and will require drastic improvements." It would be of interest to know whether a serious and comprehensive survey of the position in the U.K. has ever been attempted or even contemplated. It would also be interesting to ascertain to what extent the development of numerical automation is influencing the curricula of our own universities and major technical colleges.

Throughout the Proceedings the influence of the U.S.A. is, as one might expect, strongly evident. It is, however, pertinent to point out that the British contribution to computer science is by no means negligible. Indeed, one has the impression, from time to time reading this volume, that it would be helpful all round if more systematic information about British work and achievements in all fields of the science were made available to our Canadian colleagues. We on our side are grateful for this volume both as a useful survey of the Canadian computing scene and as a non-academic introduction to computer science and some of its applications.

R. GOODMAN.

Downloaded from https://academic.oup.com/comjnl/article/2/4/189/470600 by guest on 19 April 2024