Modelling of indexed sequential files: Monitoring disc transfers

Eva Huzan

Slough College of Higher Education, Wellington Street, Slough, Berks.

This paper describes monitoring experiments that were carried out to obtain a detailed understanding of a particular disc hardware/software system to enable a disc file simulation (DFS) model to be constructed and validated. The validated DFS model provides a means of testing the validity of estimation formulae for the performance evaluation of indexed sequential files.

(Received November 1977)

There are many combinations of file parameters and design alternatives which affect the record access time for disc files. This has been clearly demonstrated in the reports by Senko *et al.* (1967, 1968, 1969, 1971), Lum *et al.* (1970, 1974) and Owens (1970), which describe the file organisation models (FOREM I and FOREM II) developed by IBM to aid the system designer in exploring alternative parameter combinations and in selecting the final design.

The work described in this paper is part of a larger project concerned with an investigation and modelling of disc files. A detailed disc file simulation (DFS) model has been designed to be used as a research tool to test the validity of existing estimation formulae for the performance evaluation of indexed sequential files, as given for example in Senko *et al.* (1967), Ling *et al.* (1970), Clifton (1972), London (1974), Waters (1975) and Kollias (1978), and to enable new estimation methods to be devised for different parameter combinations.

In order to construct the DFS model and validate it against actual files it was necessary to obtain a detailed knowledge of the hardware and particularly of the software to be modelled. This paper describes how a software monitor was used to investigate the way data is accessed and transferred by the particular software being modelled. These details were obtained by setting up actual disc files of specified organisations and updating these in a controlled way to force certain data accesses to occur for the different situations that can exist.

It is intended that the results of this project will also be of use to the commercial user, and it was therefore decided that one of the most commonly used disc file organisations (indexed sequential) should be investigated in depth, that the applications simulated should be typical commercial ones (updating and information retrieval) and that the computer with its associated software should be one that is widely used (ICL 1900, with Direct Access Housekeeping Mark III).

The results given in this paper are for selective (skip) sequential processing, and these show how the Direct Access Housekeeping package accesses and transfers data for different buffer combinations. Further tests were carried out for sequential and random processing. To model the full capabilities of the software, tests would need to be carried out for the other options available, for example the enhanced selective/sequential processing facility starts the search from the current position of the index and allows the user to allocate a two-cell area instead of an index buffer at any level of index.

The ICL 1900 Direct Access Housekeeping (DAH) package

This consists of a number of subroutines which enable the programmer to access data in direct access files at the record level. The basic high level commands, known as Storage Device Macros, direct housekeeping to open, close or extend a file, to search index tables to determine the location of a record in a file and to read, write, delete, insert or update a single record. The necessary low level operations, such as issuing Executive level commands, allocation of buffers and scheduling of transfers between these and the disc, searching of index tables, and handling of overflow, are carried out by housekeeping routines (ICL, 1974).

Note:
The unit of data transfer for ICL 1900 disc files is a bucket, which may be 1, 2, 4 or 8 blocks of 128 24-bit words in length.

Buffers

Direct Access Housekeeping uses a number of buffers for transferring data when processing at the logical level. Housekeeping buffers are allocated by the compiler as a result of parameters in the SDDEF macro in the PLAN update program. One or two home buffers may be specified to hold home buckets and second level overflow buckets. An overflow buffer may be specified to hold first and second level overflow buckets, otherwise the home buffer(s) is used for holding both types of overflow bucket. At least two buffers must be available if DAH is to deal with second level overflow insertions and updating. Record buffers are used to hold a single record and are allocated by the user.

Index buffers are used by the SDIND macro to access the indexes in a sequential file. Up to three levels of index may be specified by the user. The level 1 index is at the beginning of the file, the level 2 index is at the beginning of each file area and the level 3 index is at the beginning of each seek area. The index buckets contain cells of information consisting of the logical bucket number of the start of the next level of index, and the highest key in the next level. The cells of the lowest level index being used contain the logical bucket number containing the required record key, followed by the highest key in that bucket. The user may allocate buffers for none, some or all of the indexes, by parameters associated with the SDIND macro.

File allocation and organisation

The ICL 1900 File Allocator routine #XPJC (ICL, 1975) was used to allocate a small indexed sequential file of seven seek areas (cylinders), two of which were designated for the second level overflow area. Each cylinder contained 16 1-block buckets, three of which in each cylinder (other than those in the second level overflow area) were designated as first level overflow buckets as a result of specifying a cylinder packing density of 85%. The first cylinder contained the level 1 index for the file in bucket number 1 and the level 3 index for that cylinder in bucket number 2. The remaining four cylinders contained the level 3 index in the first bucket of the cylinder. Thus the first cylinder had 11 home buckets and the other four cylinders had 12 home buckets as shown in **Fig. 1.**

This file was loaded with data initially using the ICL File Organisation Software. This consists of two parts: first a PLAN skeleton program is generated which is run with parameters cylinder — logical bucket numbers — first level overflow area

cylinder																	
1	1 L1	2 L3	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
2	L3																
3	L3																
4	L3																
5	L3																
6																	*
7																112	

L1 = level 1 index
L3 = level 3 index
* second level overflow area

Fig. 1 File map

to give a specified file organisation. A bucket packing density of 75% was input as a parameter. This is the percentage of space that may be used in the home buckets for data on loading.

The indexed sequential file was loaded with data from a serial direct access file. The latter was created by a COBOL program which had as input the following parameters: the number of records (150) to be output to the disc file, the record length (30 words), the first key (10), the gap between successive keys (50), and the file generation number.

The contents of the buckets on the indexed sequential files were printed by means of an ICL routine #XPJL, and showed that three records had been loaded into each bucket. The space left in the buckets was 36 words (128 − 2 word header − (3 × 30)).

Updating the file

The program developed to update the indexed sequential file was written in PLAN and incorporated a segment which recorded the disc transfers in a stack. The latter was output at the end of each run in the form of a core printout which showed the unit number, mode, logical bucket number, transfer size and buffer address, of each disc transfer in the order in which the transfers occurred (see **Fig. 2**).

In addition, the DAH Performance Information routine was incorporated. This outputs the mill time used in housekeeping routines and the number of home, first and second level overflow and index buckets read and written (HOME R, HOME W, 1OF R, 1OF W, 2OF R, 2OF W, IND R). The disc transfers as recorded by this routine were compared with the core printouts. The latter were used to determine the action of DAH, so that this could be incorporated in the DFS model. These core printouts are also used to validate the action of the DFS model.

Core printout of stack

Fig. 2 shows the format of the core printout (in octal) and its interpretation. A core printout of all the disc transfers that occurred was obtained after the same data file had been updated with the same insertions for each of the sixteen possible buffer combinations. Each transfer was interpreted and this information was used to construct **Table 1** and to explain the differences in the number of disc transfers according to the buffer combination used and the distribution of the insertions over a cylinder. The effect of the latter is illustrated by comparing C2, C3 and C4 for each specified combination.

Distributions of insertions

Five records were inserted in each of three cylinders (C2, C3 and C4). For C2, the insertions were made into separate buckets; as there were 36 words left in each bucket after loading, no overflow occurred in C2. For C3, the distribution of record keys for the insertions was such that one bucket had two insertions, and another bucket had three insertions. For C4, all five insertions were made into one bucket.

Explanation of variations in number of disc transfers

1. 1 *HOME*, 1 *OVERFLOW BUFFER*

1.1 *Buffers for level* 1 (*L*1) *and level* 3 (*L*3) *indexes*

L1 is read into its own separate buffer at the start of the processing and remains there throughout. L3 is read once into its own separate buffer for each cylinder.

For C2, there is one home bucket read into the home buffer for every insertion (into separate home buckets) and corresponding home writes to write away the updated bucket each time.

For C3, only two home buckets are accessed (two home reads and writes). One bucket had two insertions; however there is room for only one, therefore the second insertion has to be written into a first level overflow bucket. Two 1OF reads are required. The first reads the first bucket in the cylinder into

Core printout	*Interpretations*
*00200044	unit 0 (main file), mode 2 = read, Home BN 36
*00000200	transfer size = 128 words (1 block bucket)
*00014662	address of home buffer (main file)
*00200041	read 1st bucket in cylinder (to establish overflow BN)
*00000200	
*00015074	address of overflow buffer (main file)
*00200060	
*00000200	
*00015074	
*01200003	unit 1 (transaction file) read BN 3
*00000200	
*00017002	address of home buffer (transaction file)
*00300044	main file, mode 3 = write, Home BN 36
*00000200	
*00014662	

Fig. 2 Core printout of part of stack for 1 home, 1 overflow buffer combination

Table 1 Disc transfers for different buffer combinations for the same data files

Buffer combination			*Number of disc transfers*					
			HOME R	1OF R	IND R	HOME W	1OF W	Cylinder TOTAL
1 HOME, 1 OVERFLOW	L1, L3	C2	5	0	1	5	0	11
		C3	2	2	1	2	1	8
		C4	1	4	1	1	3	10
	L1 only	C2	5	0	5	5	0	15
		C3	5	2	5	5	1	18
		C4	5	4	5	5	3	22
	L3 only	C2	5	0	6	5	0	16
		C3	5	2	6	5	1	19
		C4	5	4	6	5	3	23
	NO INDEX BUFF	C2	5	0	10	5	0	20
		C3	5	2	10	5	1	23
		C4	5	4	10	5	3	27
1 HOME, NO OVERFLOW	L1, L3	C2	5	0	1	5	0	11
		C3	5	4	1	5	3	18
		C4	6	8	1	6	6	27
	L1 only	C2	5	0	5	5	0	15
		C3	8	4	5	5	3	25
		C4	10	8	5	6	6	35
	L3 only	C2	5	0	6	5	0	16
		C3	8	4	6	5	3	26
		C4	10	8	6	6	6	36
	NO INDEX BUFF	C2	5	0	10	5	0	20
		C3	8	4	10	5	3	30
		C4	10	8	10	6	6	40
2 HOME, NO OVERFLOW	L1, L3	C2	5	0	1	5	0	11
		C3	2	3	1	2	2	10
		C4	2	4	1	2	3	12
	L1 only	C2	5	0	5	5	0	15
		C3	2	3	5	2	3	15
		C4	2	6	5	2	5	20
	L3 only	C2	5	0	6	5	0	16
		C3	2	4	6	2	3	17
		C4	2	7	6	2	5	22
	NO INDEX BUFF	C2	5	0	10	5	0	20
		C3	2	3	10	2	3	20
		C4	2	6	10	2	5	25
2 HOME, 1 OVERFLOW	L1, L3	C2	5	0	1	5	0	11
		C3	2	2	1	2	1	8
		C4	1	4	1	1	3	10
	L1 only	C2	5	0	5	5	0	15
		C3	2	1	3	2	1	9
		C4	1	2	2	1	3	9
	L3 only	C2	5	0	6	5	0	16
		C3	2	2	3	2	1	10
		C4	1	4	3	1	3	12
	NO INDEX BUFF	C2	5	0	10	5	0	20
		C3	2	1	10	2	1	16
		C4	1	2	10	1	3	17

the overflow buffer to establish the logical bucket number of the bucket currently accepting overflow records. The overflow bucket write occurs when the overflow buffer is required by C4.

For C4, there are five insertions into one bucket, therefore there is one home read into the home buffer and one home write. The home bucket has room for one record to be inserted and there are six words left (126 − (4 × 30)). For the second and third insertions, tags have to be written in the home bucket to show in which first level overflow bucket the records have been written. The length of a tag is equal to the key size rounded up to an integral number of words plus one word (the tag for a 7-character key = 3 words). The tag for the fourth insertion cannot be written until a record has been displaced from the home bucket to make room for it.

The left-most record of size ⩾ 2 tags is written in the first level overflow area and 2 tags (for the insertion and the displaced record) are written in the home bucket. Thus five records need to be written in the first level overflow area requiring two overflow buckets to be accessed (each overflow bucket can hold four 30-word records only). The first overflow bucket read establishes the logical bucket number of the required overflow bucket for this cylinder as for C3. The second and third overflow bucket reads access the required two overflow buckets. The fourth overflow bucket read accesses the first bucket in the cylinder to record the number of the overflow bucket now accepting overflow records (as this is a different bucket to that originally used). An overflow bucket write is required to write this away plus two more for the two overflow buckets used. The write occurs just before an overflow bucket needs to be read into the overflow buffer in each case.

1.2 *Level* 1 (*L*1) *index buffer only*

If no separate buffer has been allocated for L3, then it will be read into a home buffer when required. This means that for C2, the number of disc transfers are the same as before, since each insertion is into a separate home bucket so that a different home bucket is read in for each insertion.

However, for C3 and C4, the contents of the home buffer will be written away before each L3 is read after the first one. This means that the same home bucket has to be read into the home buffer again, whereas in 1.1 the required home bucket remained in the home buffer and was used again. L3 needs to be searched before each insertion can take place so that for C2, C3 and C4 there are five home reads and five home writes.

1.3 *Level* 3 (*L*3) *index buffer only*

L1 has to be searched before each insertion can be made, so that L1 will be read five times into the home buffer for C2, C3 and C4 overwriting the contents as for 1.2. In addition, L3 will be read into its own buffer once for each cylinder, thus giving six index reads for each cylinder.

Note:

Extra time will be spent in making a seek to the first cylinder in the file to access L1, and for the return seek to the required cylinder, in addition to any extra time due to disc transfers.

1.4 *No index buffers*

In this case, both L1 and L3 need to be read into a home buffer so that they can be searched before each insertion giving 2 × 5 = 10 index reads in each case.

2. 1 HOME, NO OVERFLOW BUFFER

2.1 *L*1 *and L*3 *buffers*

Because there is no overflow buffer, any overflow processing requires the use of the home buffer, whose contents must be written away first if it has been updated. There is no overflow processing for C2, so the number of disc transfers are as for 1.1.

C3 requires one record and two records, respectively, to be written in an overflow bucket for two home buckets. The disc transfers are as follows:

(*a*) read first home bucket to insert first record (for which there is room), and to try to insert second record (for which there is no room)

(*b*) write home bucket away, since it has been updated, prior to reading in the first bucket in the cylinder to establish the logical bucket number of the overflow bucket currently accepting overflow records

(*c*) read first bucket in cylinder into home buffer (this counts as an overflow read)

(*d*) read required overflow bucket into home buffer and insert record

(*e*) write overflow bucket away, prior to reading in home bucket so that tag can be written for record just written in the first level overflow bucket

(*f*) read home bucket and write tag

(*g*) write home bucket prior to reading in next home bucket

(*h*) read next home bucket into which the next insertion is to be made and insert one record; there is no room for the second record

(*i*) write home bucket

(*j*) read overflow bucket into home buffer and insert second record

(*k*) write overflow bucket

(*l*) read home bucket and write tag for second record

(*m*) write home bucket

(*n*) read overflow bucket and insert third record

(*o*) write overflow bucket

(*p*) read home bucket and insert tag for third record

(*q*) write home bucket prior to reading in next required home bucket in C4.

The processing for C4 corresponds to that for C3. The overflow bucket reads are for establishing the required overflow bucket for the cylinder initially and recording the new value (since two overflow buckets are used), plus four reads for one bucket to insert four records, plus one read to find there is no room left in the current overflow bucket to insert another record and to establish the next overflow bucket number, plus one read for that bucket = 8 overflow bucket reads, and correspondingly 6 overflow bucket writes.

2.2 *L*1 *buffer only*

The extra home bucket reads are due to the contents of the home buffer having to be written away before reading in the L3 index for each insertion. Thus, whereas in 2.1 for C3 it was possible to establish that there was no room for the second insertion into the same bucket, this time for C3 an extra home bucket read has to be made to determine this. One bucket will require one extra read for its second insertion, and the other bucket will require two extra reads for its second and third insertions. For C4, four extra home bucket reads are required for the four insertions for which there is no room in one bucket.

2.3 *L*3 *buffer only*

This case is similar to 2.2 but in addition there is one extra index read for reading L3 into its separate buffer once for each cylinder.

2.4 *No index buffers*

L1 and L3 have to be read into the home buffer each time as for 1.4.

3. 2 HOME, NO OVERFLOW BUFFER

3.1 *L1 and L3 buffers*

In this case, two home buffers are available for all the home bucket and overflow bucket transfers. Comparing the number of disc transfers for C3 with 1.1, the extra overflow bucket read and write is due to the overflow bucket which has been read into the second home buffer being overwritten by the second home bucket read, and this has to be read again into the first home buffer for the second home bucket overflow processing. In C4, the home bucket was overwritten by the second overflow bucket accessed thus causing one extra home bucket read and write.

3.2 *L1 buffer only*

The two home buffers are now shared by the home buckets, overflow buckets and L3s. L3 has to be read into a home buffer before each insertion so that it can be searched to find the next required home bucket. Compared with 2.2, for C3 one overflow bucket read to establish the number of the bucket currently accepting overflow records apparently was 'saved' because the L3 index containing the first bucket in the cylinder was already in a buffer. A similar situation occurred for C4.

3.3 *L3 buffer only*

This time L3 was in its own buffer, an extra overflow bucket read had to take place so that the situation was comparable to 2.3 for C3 overflow processing. However, only two home bucket reads and writes were required for C3.

3.4 *No index buffers*

Overflow processing was as for 3.2, but extra index reads occurred for the same reasons as those stated in 2.4.

4. 2 HOME, 1 OVERFLOW BUFFER

4.1 *L1 and L3 buffers*

No savings in disc transfers could be made under these circumstances compared with 1.1.

4.2 *L1 buffer only*

Savings in disc transfers were made for two reasons:

(*a*) The alternate home buffer was used for reading in L3, thus avoiding rereading the required home buckets as was the case for 1.2, 1.3 and 1.4.

(*b*) Extra overflow bucket reads to establish and record the bucket number currently accepting overflow records were avoided as in 2.3 and 2.4, because L3 (containing the first bucket in the cylinder) was already in a buffer which had not been allocated as an index buffer.

4.3 *L3 buffer only*

There were fewer home bucket reads and writes and L1 reads for C3 and C4 compared with 1.3 because both home buffers were used in the following way. If one buffer contained an updated bucket then the second buffer was used for the L1 read. If both buffers contained an updated bucket then the buffer containing the bucket *not* last updated was used; thus the bucket last updated was still available in a buffer.

4.4 *No index buffers*

There were fewer overflow bucket reads for the reason explained in 3.2. However, because there was no separate L3 buffer, L1 was overwritten for each insertion giving the same number of index bucket reads as for 1.4.

Second level overflow processing

Second level overflow will occur for a sequential file if any of the following circumstances arises:

(*a*) a particular home bucket is full of tags. In this case, no further tags can be written since there are no records left to displace

(*b*) there is no room for a tag, and no record in the home bucket has a length greater than or equal to two tags

(*c*) all the overflow buckets in the cylinder have become full.

A number of cylinders may be allocated as a second level overflow area at the end of the file by specifying these as a parameter to #XPJC. This was done for the test file used (see Fig. 1). Buckets in use in this area are known as extension buckets because they are chained to one particular home bucket by means of pointers. Records that cannot be stored in their appropriate home buckets, or in the associated first level overflow area for the reasons given above, are written in their logical key sequence in the chain.

The records are in physical key sequence within the extension buckets. The extension buckets are in logical (but not necessarily physical) sequence by means of pointers, so that the whole of the chain including the home bucket is in key sequence. Records and tags are moved out of home and extension buckets and written elsewhere in the chain to maintain logical key sequence.

The first level overflow areas are not reused after records have been deleted from them unless this facility has been specified by setting bit 0 of word 9 of the File Definition Area. If this is set, the search for space in an overflow bucket will start at the beginning of the overflow area for that cylinder, i.e. the last bucket in the cylinder, working backwards towards the last home bucket in the cylinder. Use of this facility is likely to decrease the number of times the file has to be reorganised because of excessive second level overflow, but may increase the number of disc transfers required and should be used with caution. When the file is reorganised all the records in the overflow areas are returned to their appropriate home buckets.

Writing records in or retrieving records from an extension bucket always involves seeks to the appropriate cylinders as well as extra read and write operations. **Table 2** shows the number of disc transfers for a point overflow situation giving rise to second level overflow processing. The file previously defined was first updated to a state where one cylinder had all first level overflow buckets full and three insertions had been made in a second level overflow extension bucket for one home bucket. The example illustrates the effect of deleting two records from a first level overflow bucket (the first level overflow area is not reused) and inserting eight records into the same home bucket. This caused eight records to be written into extension buckets and involved moving a record and some tags out of the home bucket and altering the contents of extension buckets, and resetting pointers so that the resultant chain was in logical key sequence. In each case, the home and overflow buffers were used for second level overflow processing.

Conclusion

A method of monitoring the action of a particular piece of commercially available software has been described. The information obtained from the monitoring experiments has been used to construct and validate the DFS model. This simulates the action of ICL 1900 Direct Access Housekeeping accurately for the processing of indexed sequential files, and provides a means of testing the validity of various formulae and hypotheses concerned with the performance of these types of files. Because of the differences in hardware and software, some of the tests will not apply to other systems. However, the model can be adapted to simulate other hardware and software environments, provided their action is known to the level of detail indicated in this paper, and validating facilities are available.

Table 2 Disc transfers for a point overflow situation

		Number of disc transfers							
Buffer combination		HOME R	1OF R	2OF R	IND R	HOME W	1OF W	2OF W	TOTAL
1 HOME, 1 OVERFLOW	L1, L3	6	1	16	2	1	1	8	35
	L1 only	10	1	16	11	4	1	8	51
	L3 only	10	1	16	11	4	1	8	51
	NO INDEX BUFF	10	1	16	20	4	1	8	60
2 HOME, NO OVERFLOW	L1, L3	6	1	21	2	1	1	13	45
	L1 only	9	1	22	11	3	1	14	61
	L3 only	9	1	22	11	3	1	14	61
	NO INDEX BUFF	9	1	22	20	3	1	14	70
2 HOME, 1 OVERFLOW	L1, L3	6	1	13	2	1	1	8	32
	L1 only	8	1	13	9	3	1	8	43
	L3 only	8	1	13	9	3	1	8	43
	NO INDEX BUFF	9	1	13	19	3	1	8	54

Acknowledgements

The author wishes to thank Mr F. F. Land and Dr S. J. Waters, of the London School of Economics and Political Science where this reasearch project is being carried out, for helpful discussions, her colleagues in the Computer Unit at Slough College of Higher Education for help in setting up and processing the test files, and International Computers Limited for their co-operation and assistance.

References

Clifton, H. D. (1972). *Systems analysis for business data processing*, Business Books.

ICL. (1974). *Direct Access*, Technical Publication 4385.

ICL. (1975). *Unified Direct Access Standards Utilities*, Technical Publication 4405.

Kollias, J. G. (1978). An estimate of seek time for batched searching of random or index sequential structured files, *The Computer Journal*, Vol. 21, No. 2, pp. 132-133.

Ling, H., Lum, V. Y., and Senko, M. E. (1970). Calibration of the File Organisation Evaluation Model (FOREM I), Formatted File Organisation Techniques, Final Report, IBM Research, Contract AF30602-69-C-0097.

London, K. R. (1974). *Techniques for direct access*, Petrocelli Books.

Lum, V. Y., Ling, H. and Senko, M. E. (1970). Analysis of a complex data management access method by simulation 'modelling, *AFIPS FJCC*, Vol. 37, pp. 211-222.

Lum, V. Y., Senko, M. E., Ling, H. and Barlow, J. H. (1974). Quantitative timing analysis and verification for file organisation modelling, *Information Systems*, Coins IV, Plenum, New York, pp. 377-387.

Owens, P. J. (1970). Phase II, A data management system model, Formatted File Organisation Techniques, Final Report, IBM Research, Contract AF30602-69-C-0097.

Senko, M. E. *et al.* (1967). Formatted File Organisation Techniques, Final Report, IBM Research, Contract AF30(602)-4088.

Senko, M. E., Lum, V. Y. and Owens, P. J. (1968). A File Organisation Evaluation Model (FOREM), *IFIP Congress*, pp. C19-C23.

Senko, M. E. *et al.* (1969). File Design Handbook, Final Report, IBM Research, Contract AF30602-69-C-0100.

Senko, M. E. *et al.* (1971). Semi-operational evaluation of file modelling techniques, Final Report Volume I, IBM Research, Contract AF30602-70-C-0114.

Waters, S. J. (1975). Estimating magnetic disc seeks, *The Computer Journal*, Vol. 18, No. 1, pp. 12-17.

Book reviews

The BASIC Idea: An Introduction to Computer Programming, by R. Forsyth, 1978; 154 pages. (*Chapman and Hall*, £1·95)

This book is one of a number of recent introductions to programming and the use of BASIC. These introductions range in style from the most pedantic programming manuals to flashy, journalistic approaches that are breaking new ground for serious computing texts. The present book falls roughly in the middle of that range: the style is a reasonable compromise between the rival approaches and the reader is presented with an illuminating (and sometimes entertaining) range of problems and sample programs.

The whole presentation is in terms of the BASIC implementation for the DEC System-10; while compatability problems with other versions of the language are mentioned this is done in a somewhat haphazard way. The immediate execution facilities of BASIC are not mentioned—these can be helpful during program testing. With these reservations I liked the book and would certainly recommend it to those learning programming by using DEC System-10 BASIC.

Peter Wallis (Bath)

Large-Scale Evaluation Study of On-line and Batch Computer Information Services, by A. Vickery and A. Batten, 1978; 176 pages.

This is a very carefully compiled report on the investigations into the need and suitability of a computerised library information retrieval service for the University of London. The points for and against are well presented, even if the enthusiasm of the authors is apparent. There is an irritating large scale use of abbreviations and, although they are defined, it is difficult to carry them all in one's mind when reading the report. The repetition of facts in the various sections is much more acceptable however, since without this the completeness of the presentation would suffer.

Inevitably aid is needed in information services since 'as the volume and variety of information expands the proportion immediately available in one institution contracts'. This rather specialised report will be of value to librarians, researchers, providers of systems and equipment, and providers of information services.

A. J. Thomas (Sunbury on Thames)