Banking operations often require complex facilities for their data processing. This application required a multiprocessor configuration controlled by a single job step running continuously for many hours a day. Discussed are the special access methods and recovery procedures designed for this environment. The paper also describes a particularly efficient sorting technique evolved for handling large volumes of paper documents.

Design features of a real-time check-clearing system

by J. A. Banham and P. McClelland

The transactions involved in the operations of the banking industry often require the use of data processing facilities in a manner that is different from that of other industries. A large volume of documents must be processed. Many data entries are made, and the data must be processed quickly to provide a short turnaround time. The check-clearing operation is typical of the industry.

The passage taken by a check through a banking system is similar in all countries; the difference in England lies solely in the large size of the centralized clearing operation, which is facilitated by the small number of banks (seven) who operate all the current (checking) accounts in England and Wales or act as clearing agents for the others. Among them they have some 10,000 branches of which four banks have over 2,000 each and of which another has fewer than 10. The seven banks are all members of the Committee of London Clearing Bankers that lays down all the standards and procedures involved in the exchange of checks among the banks.[1] These banks clear an average of about 4,000,000 checks per day, the number rising to twice that on a peak day.

In this paper, we describe the techniques used in the clearing system of the National Westminster Bank. This large, real-time, data-entry, document-handling system receives up to 2,200,000 checks daily, sorts them uniquely for return

to the bank branches, reconciles the total value with the presenting bank, and provides output to debit drawers' accounts. The data processing facilities of the system consist of an IBM System/360 Model 65 coupled to three IBM System/360 Model 40s.

Problems encountered in designing the system required certain facilities for their solution. In the paper, the system is described as a whole indicating why it became necessary to provide these facilities. Among the problems was the need for the following: (1) a powerful restart and recovery technique, (2) unique data management, (3) an efficient sorting technique, (4) a long job step, (5) computer-controlled task scheduling, and (6) an easily varied configuration. Because they are believed to be novel, the solutions to the first three problems are discussed in detail.

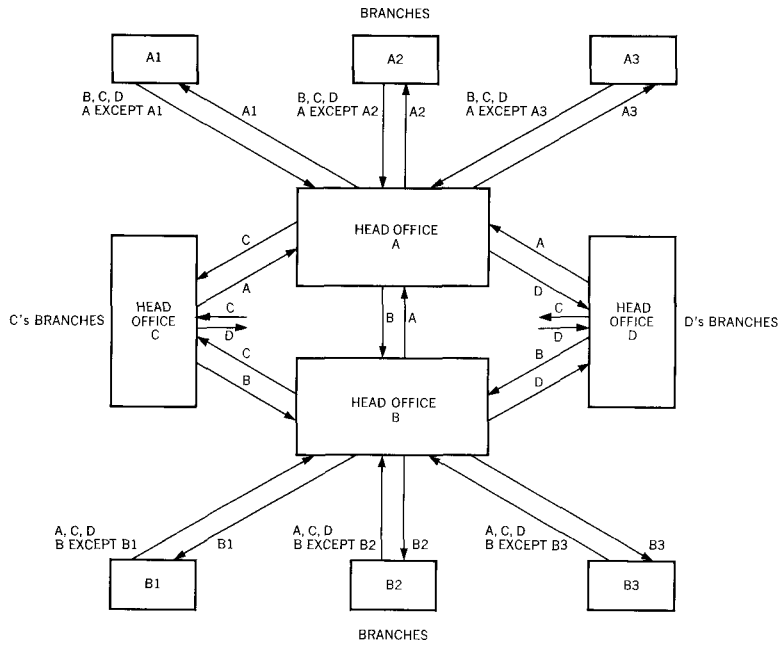
Destruction of the input file by the sorting operation, such that there can never be any backtracking to a checkpoint, is the reason for the restart and recovery problem. New data management facilities were required because 4,000 data groups that were entered at up to 100 entry points had to be indexed, accessed, concatenated, merged, sorted, split, etc. rapidly. The sort technique designed for the system sorts in six passes what would require 20 on a conventional digit sort. It does this by building a sort table based on the document numbers that actually happen to be present on a particular day and the sequence in which they are expected to appear on each pass.

We first describe the system, including the hardware configuration and program architecture. We then discuss the work flow through the system and certain special features. Finally, the special access method and the sorting techniques are examined in detail.

System description

After the close of business each day, the staff in each branch separate all the checks that have been presented to them that day, and that are not drawn on accounts held at their branch, into seven bundles, one for each of the clearing banks. The checks are then encoded with their amounts in magnetic ink characters. A control voucher, known as a DCV, is encoded with the total amount of the bundle. A control list of the amounts in each bundle is produced as a by-product of the encoding operation. All the bundles are then mailed together to the Head Office of the bank in London, where the following morning all the bundles from every branch of that bank are sorted and delivered to the Head Office of the bank on which the checks were drawn (see Figure 1).

Figure 1 Clearing operation



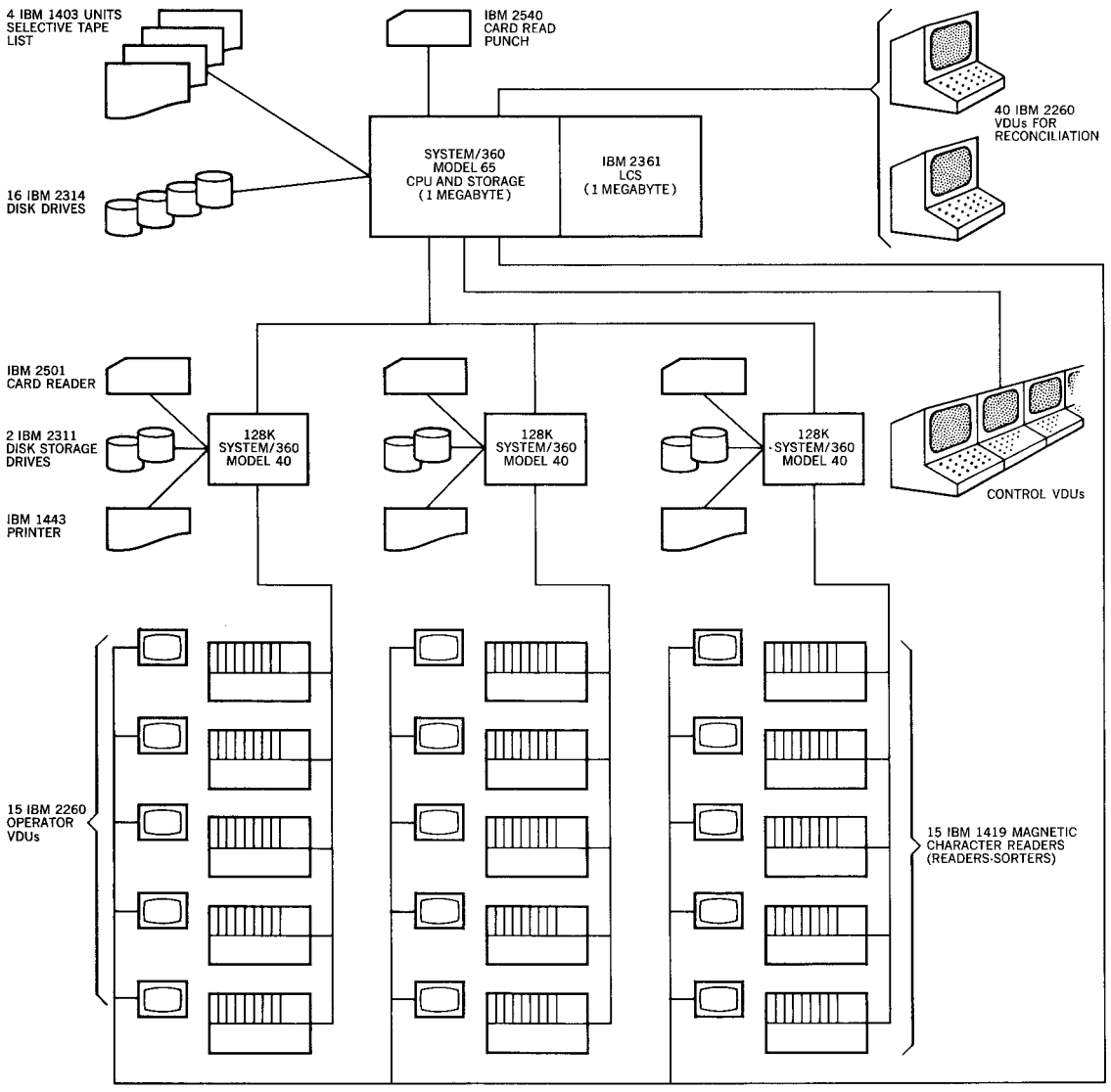
REFERENCE LETTERS ON ARROWS INDICATE DIRECTION OF FLOW OF CHECKS DRAWN ON THE BANK OR BRANCH REFERENCED

Each major bank has a computerized clearing system to handle the incoming checks. Any automated clearing system must provide sorting capacity to sort all the checks to branch order, at least. The complete magnetic ink code line must be captured from as many of the checks as possible. This data is used to create magnetic tapes for use on the bank's bookkeeping system. It is also used to provide audit trails both to control payment for the checks to other banks and also to charge bank branches for the checks they receive. The movement of checks and control documents is illustrated in Figure 1. Checks not successfully read by the reader-sorter machines must be manually posted to the customer's account by the branch.

Additional requirements for this system included:

- Improving the service to the branches by sorting the checks into serial number within account number order for each branch
- Providing an automated trial balance, so that the value of all checks received, including those rejected by the reader-sorters, can be reconciled with the other banks more easily
- Providing a wide range of statistics, financial summaries, and a microfilm record of the code line of each check read to enable subsequent differences to be traced more easily

Figure 2 Hardware configuration



The design limit of approximately 3,000,000 checks a day gives a minimum of over 15,000,000 document passes daily. The prime requirement is, therefore, to maintain throughput. This leads to a need for very flexible control so that reader-sorters can be operating independently on different passes at any time, and also so that work can easily be transferred from one machine to another without affecting the whole of the installation. Hardware switching and rapid restart are also of prime importance should a major system failure occur.

hardware configuration

The preliminary design study showed that 20 reader-sorters would eventually be required to handle the predicted peak vol-

ume of checks. In addition, some 60 to 80 visual display terminals would be required to handle the checks rejected from the reader-sorters. All the data captured on all of these devices has to be merged together to produce the required output. The time scale for doing this, particularly for producing the trial balance, is such that a multiprocessor system is essential. Figure 2 shows an outline of the configuration.

The reader-sorter device used, the IBM 1419 Magnetic Character Reader, generates an external interruption as soon as it has read each check. The CPU must then calculate and issue the stacker select command within 15 milliseconds, or the check is rejected.[1] A simulation study of the stacker-select interruption handling time and the document processing time showed that one Model 40 could support up to six reader-sorters and still maintain full throughput without rejecting many documents. The configuration that we evolved presently requires three Model 40s. On an average day, six reader-sorters are attached to each of two Model 40s. On a busy day, additional reader-sorters are attached to the third Model 40. Eventually the configuration will grow to four Model 40s, controlling 20 reader-sorters. The Model 65 collates all the data sent to it from the reader-sorters by the Model 40s. Two groups of visual display terminals (VDUs) are attached to the Model 65. On the reconciliation group, data is keyed in from the checks that were rejected by the reader-sorters. On the operators' group, the VDUs receive and display messages for controlling the operation of the system. For backup, a second Model 65 is available should the first fail.

program architecture

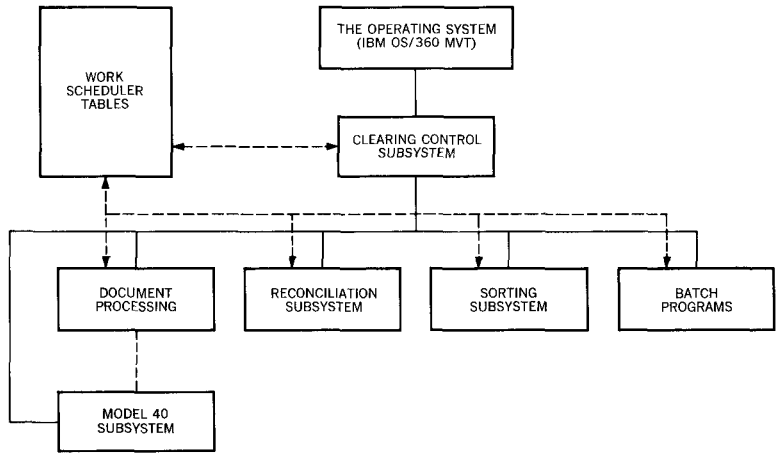
The Model 65 is controlled by Operating System/360 (OS/360) with MVT (multiprogramming with a variable number of tasks), whereas the Model 40s run under the Disk Operating System (DOS/360). OS/360 was chosen for the Model 65 because of the need for a variable number of tasks and flexibility of operation. The main interface with the operating system is the user control subsystem, which in turn controls the following major application subsystems:

- Document processing
- Reconciliation
- Sorting
- Batch process

Each application subsystem consists of one or more tasks attached by the control subsystem. To ease control and restart problems, the application subsystems are not permitted to attach further tasks.

The work scheduler program forms the heart of the control subsystem. It drives and is driven by a set of tables that show the

Figure 3 System structure



order in which tasks are to be performed. As each task is completed, the appropriate table entry is updated; thus the day's progress is continuously monitored and controlled.

In the Model 40, the need for very efficient execution and the limited main storage available led to the choice of DOS/360, using a single partition and user-written, multiple-wait routine. Each Model 40 is controlled by the control subsystem in the Model 65. The system structure is illustrated in Figure 3.

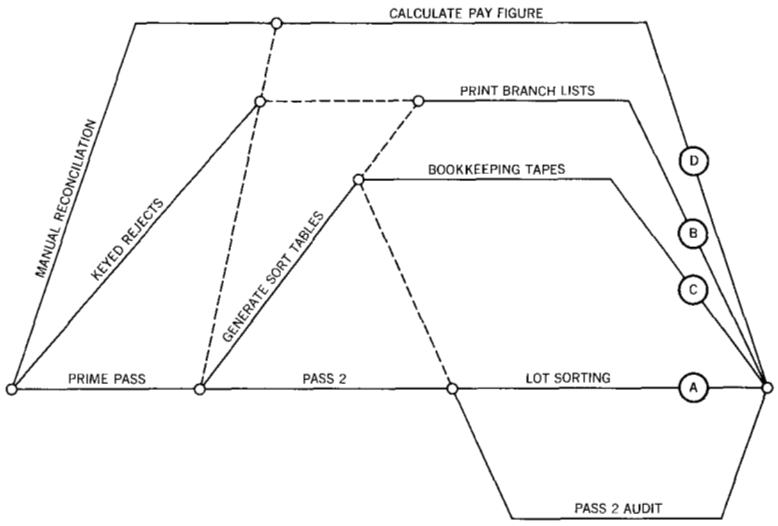
System work flow

Checks are received from presenting banks in bundles, one for each branch. Each bundle is accompanied by a Docket Control Voucher (DCV) which was encoded with the bundle total in the branch. The checks arrive in trays of 2,000–3,000 checks. A set of Charge Identifiers is placed in each tray, which is then known as a Charge.

Figure 4 shows the sequence of operations in the system. Prime pass, keyed rejects, and reconciliation are all taking place at the same time, but any one charge must go through the three processes in that sequence. As shown in the figure, the output from the system consists of:

A – The sorted checks
B – A list of the checks to be returned to the drawee branches
C – Input to the bank's bookkeeping system with which to debit the drawers' accounts
D – The pay figure, i.e., the amount to be paid to the presenting bank

Figure 4 Sequence of operations



We now describe these operations.

prime pass

When a Charge is read on the prime pass, the Charge Identifiers and DCVs are sorted to the reject pocket of the reader-sorter, together with any checks that could not be read. Data read from the checks is recorded twice, as follows: (1) in input order, indexed by Charge number on the reconciliation file, and (2) in output order, with each pocket of each reader-sorter used on the first pass as a member of the pocket file. So that they can be retrieved again easily, these data sets are written on disk using the special access method described later in this paper.

reconciliation

For each Charge, the total value of the checks, as read by the reader-sorters and accumulated by the computer, is almost always different from the totals of the DCVs in the Charge. The reconciliation operation is carried out to resolve this difference.

In most cases, the addition of the value of checks rejected by the reader-sorter to the value of checks accepted will resolve the difference on a DCV. In this system, the value of each check rejected is entered at a VDU, enabling the addition to be carried out by the computer. An exception list of remaining out-of-balance DCVs is then printed for each Charge. Further investigation is a clerical operation.

When all Charges have been reconciled, overall agreement has been reached, and the individual presenting banks can be paid.

further processing

At the end of the first pass, the pocket file is sorted on disk into the final output sequence of the checks themselves. The sorted

file is used to create magnetic tapes for updating the customers' accounts held on the bank's bookkeeping systems, to create the sorting pattern for later passes, and to produce a control list to accompany the checks to each branch.

further sorting

The fine-sorting technique described later in the section on document sorting brings very considerable savings in the number of document passes per day. On the first two passes, the checks are sorted into 144 "lots" of roughly equal size, each lot containing all the checks for one or more bank branches. From pass 3 onwards each lot is completely and independently sorted on one reader-sorter, no further interchange of checks between reader-sorters being necessary. One of two alternative sorting techniques can be chosen for each lot: (1) Coarse sorting, using a conventional programmed sorting technique, takes two further passes to sort to branch order, (2) Fine sorting, using the new sorting technique described later, takes four or five further passes, depending on volume, to sort to serial number within account number within branch order.

Special features

The system contains many features that we believe would interest those who are using character recognition, data entry terminals, multiprocessing, or even a large multitasking system. We cannot describe all the features in a paper of this nature, so we have selected just three for further expansion—the restart and recovery facilities, the disk-access method, and the document sorting technique. They were selected in the hope that they will be of interest to most readers.

The restart and recovery facilities are of paramount importance on any complex system. The way in which the work scheduler tables act as an interface between the control program, the application program, and the restart routines is of particular interest in this system. The disk-access method is probably of most interest in systems with a large number of data entry points, and the document-sorting technique is probably of most use in financial applications.

restart, recovery, and standby

In a system of this complexity with a hundred or more application subtasks working concurrently, any failure must be localized and recovery made as easy as possible.

Extensive use is made of the work scheduler tables to achieve the necessary flexibility and speed of recovery. In normal operation, the work scheduler program uses the tables as a reference to attach application subtasks in the correct sequence throughout the day. The tables are held in main storage and checkpointed by the control program each time it updates an entry.

The work carried out by an application subtask is known as a *mission.* A work scheduler table of Mission Status Blocks (MSBs) is used to record progress through each mission. The MSB is the main interface between the control program and the application subtask. Standard fields within the MSB are set up by the control program prior to the attachment of the subtask. These indicate the starting point for the mission and the resource allocated. During the progress of the mission, the application subtask updates other fields of the MSB (and checkpoints the table) at intermediate points that are significant to the particular mission. For example, each prime-pass subtask records the latest Charge number in the MSB on completing a Charge.

The early part of each day is a prime data capture operation. This is the critical period because it must always be possible to identify exactly which checks have been processed. The processing destroys the input order; so it is not possible to refeed checks through a reader-sorter should a failure occur. All data captured is logged in duplicate during this period.

Later in the day, logging is discontinued when prime data capture has been completed. Any mission from then on can be rerun if necessary because the input data remains available until the end of the day.

The loss of throughput in deactivating the system to take a system-wide checkpoint precluded this approach. However, the combination of data logging and the "minicheckpoints" taken by each subtask ensure that a rapid, reliable restart can be undertaken at any time.

data logging

The object of writing the log tapes is to record all the checks read by the system so that if the system fails, it will be possible to restart the system by reading the magnetic tape instead of refeeding the checks.

The Model 65 writes duplicate log tapes of all the data as soon as it is received from the Model 40s. Each block is preceded by a time stamp and sequence number. If a complete system restart is required, the two tapes are played back in parallel (in case one has an I/O error). The end of a particular day's data is recognized by a discontinuity in the sequence field or by a tape mark if the tapes were closed correctly.

The restart mechanism replays the log tape and feeds the data to the applications programs exactly as it was originally entered through the reader-sorters or visual display terminals. At the end of the tape, the mechanism switches over to reading the data from the sorters and terminals directly, without the programs being aware of the change. In a cold restart, the application pro-

grams reprocess the data and rewrite the disk files exactly as in following a normal start. If the failure did not cause loss of data held on disk, a faster procedure, known as a warm restart, is used.

warm restart

When a warm restart is required, the work scheduler reloads the checkpointed tables as they were at the time of failure. Although a complete system checkpoint can never be taken, each task can checkpoint relevant parts of the tables whenever necessary. This means that the checkpointed tables always contain the latest configuration and a record of all completed tasks and those that were attached at the time of failure, along with their MSBs. The work scheduler then merely reattaches the tasks, and they automatically pick up their MSBs.

If the failure occurred during prime data capture, the work scheduler reattaches only those tasks that were originally attached at the start of the day and that were not complete at the time of failure. As the log tape is replayed, a request from it for a task to be attached is serviced if the task was still active at the time of failure.

During execution, the prime data collection tasks NOTE (as in OS/360 macroinstructions[3]) the starting addresses in the pocket files of the current Charge and record them in their MSBs. Each task knows when it is doing a restart and ignores all data from the tape until it reaches the Charge number recorded in its MSB. Then it POINTS to the pocket file disk addresses and continues writing as though nothing had happened. In this way, the restart runs almost at tape speed until the last few blocks are reached.

Of course, if a warm restart is required after the end of prime data collection, the tasks merely start again or continue from whence they left off, without there being any need to replay the log tape.

Special partitioned access method

At an early stage in the system design, it was realized that a special access method was required to handle each day's data. The peak volume to be handled would fill 16 IBM 2314 disk packs. For efficient processing, this volume had to be recorded as 10,000 or more data set members. An option was also needed to allow selected records within a member to be retrieved without searching sequentially through the member, i.e., it was necessary to NOTE the record address when it was written so that it was possible to POINT directly to the address later. The access method had to provide macroinstructions at the GET and PUT level and deal with space allocation, buffer control, and checkpointing.

The design of a suitable access method required a megabyte of large core storage (LCS) on the Model 65. This was used to hold control tables, thus making the access method very efficient. Because the 10,000 or more members are opened and closed many times a day, the OS/360 routines would be unnecessarily powerful and unnecessarily slow. The open and close routines in the new access methods are fast because they are dealing with preformatted packs and the allocation is controlled by the tables stored in LCS.

Head contention problems would normally arise in attempting to create over 200 members concurrently using a limited number of disk drives, as is necessary during prime pass. Further contention could arise later, if, say, a Charge prime pass member and a pocket file member were stored on the same drive and each were retrieved at the same time. The second problem is overcome by allocating files to one of several groups at initialization time. The total space available is subdivided into groups, the minimum allocation being one IBM 2314 disk pack. Thus members allocated to different groups cannot be stored on the same pack. The first problem is overcome by allocating a track at a time from each pack within the group to members of the group as they request space. Thus, during the writing of many members concurrently, head movement between members is minimized. The members are written to a series of tracks, usually not contiguous, and checkpointed tables are maintained by the access method in LCS to enable the data to be retrieved.

options

At initialization time, a series of user-supplied card images are read. Each card defines the following parameters: filename, maximum number of members, record length, and group allocation. A further card image defines the number of drives to be allocated for each group.

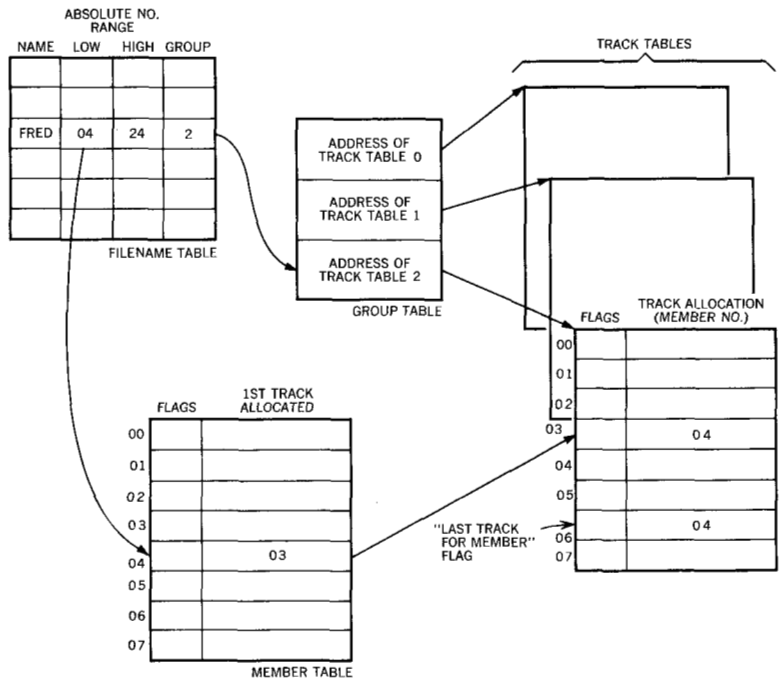
After initialization, special macroinstructions are available to the application programmer, providing functions equivalent to the OS/360 macroinstructions OPEN, CLOSE, GET, PUT, NOTE, and POINT.[3] In general, OS/360 conventions are observed. For instance, a return code is provided so that successful execution of the macroinstruction can be checked. To prevent data corruption, only one task can write to a member at any one time, but several can read a member concurrently.

internal structure

A series of tables are set up in LCS from the data supplied at initialization time to control and record the location of data. The most important of these tables are:

Filename table – One entry per filename card supplied
Group table – One entry per group defined
Track table(s) – One table is created for each group defined

Figure 5 Partitioned access method tables



– One entry in each table for each track available to the group

Member table – One entry per member defined

The relationship between these tables is illustrated in Figure 5.

filename table

The filename table consists of a series of entries in alphanumeric sequence. The group number and lowest and highest absolute member number are recorded with each entry, for example, as shown in Table 1. The filename table is used to convert the filename and member number specified by the programmer to the absolute member number used internally.

group table

The group table contains one entry for each group with the main storage address of the corresponding track table stored in each entry. The group table acts as a link between the filename table and the appropriate track table.

track table

The track table for each group contains one entry per track available within the group, in allocation sequence, i.e., drive number within track number. The track table descriptor block at the head of the track table contains various control fields, including the address of the first free entry in the table and the address of the end of the table.

Table 1 Filename table

Filename	Absolute member number Lowest	Highest	Group number
ALPHA	0	19	2
BRAVO	20	24	1
CHARLIE	25	34	2
DELTA	35	49	1

member table

The member table consists of a series of entries in ascending member number order. The entry for a given member either contains the address of the first entry on the appropriate track for that member or zero.

When a member is opened for output, the following amendments are made to the track table and member table:

- The first free entry in the track table is allocated and the member number stored in it. The entry is also flagged temporarily as the last entry allocated to the member.
- The first free entry field in the track table descriptor block is updated.
- The address of the first track table entry for the member is stored in the member table.

If a further track is required for a member, no amendments are needed to the member table. The following amendments are made to the track table:

- The first free entry in the track table is allocated and the member number stored in it. The entry is flagged as the last entry allocated, and this flag removed from the previous entry for the member.
- The first free entry field in the track table descriptor block is updated.

use of access method

A specific example may clarify the space allocation technique used by the access method. Suppose five members, say member numbers 101 through 105 of a particular group, are opened for creation concurrently, and drives 1, 2, and 3 are allocated to the group. Thus we have the following setup:

	Drive 1	Drive 2	Drive 3
Track 1	101	102	103
Track 2	104	105	Free

Suppose now that member number 102 has sufficient data for two more tracks, member 104 has sufficient data for one more

track, and no further space is required for the others. Then the final track allocation at this stage may be according to this setup:

	Drive 1	Drive 2	Drive 3
Track 1	101	102	103
Track 2	104	105	102
Track 3	102	104	Free

To retrieve, say, member 104, the entry 104 in the member table is used to find the first track allocated (track 2, drive 1). After retrieving this track, the track table is searched until the next member number 104 is found (track 3, drive 2). In this case, the entry is found to be flagged as the last for that member, and the search is terminated.

Document sorting

The bank wished to return the checks to the branches already fine-sorted into a serial number (six digits) within an account number (eight digits) within a branch code (six digits) sequence. This sorting would require 20 passes of a conventional digit sort. However, the design limit of the system is three million checks a day, and it is theoretically possible to sort nearly that number of documents uniquely in six passes on a 12-stacker machine ($12^6 = 2.99$ million). To do this, a sort pattern must be calculated that is based only on those documents that happen to be present on a particular day.

During prime pass, the data is captured and recorded on the pocket files so that the fine-sort tables for passes 3 and higher in sequential order can be calculated during pass 2. The first two passes are fixed, table look-up sorts to block sort the checks into 144 lots, each containing branches with the same dispatch deadline.

sort pattern theory

It can be shown that the maximum number of unique destinations M into which any number of items can be sorted in p passes on a machine with S stackers is given by

$$M = S^p \tag{1}$$

or more generally by

$$M = S_1 \times S_2 \times S_3 \times \ldots \times S_n$$

where S_n is the number of stackers used on the particular pass 1 to n. Since we wish to sort all the documents into unique ascending sequence, the number of destinations equals the number of documents.

Table 2 Sorting pattern for three stackers

Output sequence	*Document identity*	*Fine-sort pocket number*		
		Pass 3	*Pass 2*	*Pass 1*
1	A	0	0	0
2	C	0	0	1
3	E	0	0	2
4	G	0	1	0
5	H	0	1	1
6	J	0	1	2
7	K	0	2	0
8	M	0	2	1
9	P	0	2	2
10	R	1	0	0
11	S	1	0	1
12	T	1	0	2
13	U	1	1	0
14	W	1	1	1
15	Y	1	1	2

Consider an example in which we have to sort 15 documents, each identified by a single letter, into alphabetic sequence on a three-stacker machine:

Input sequence	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Identifier	S J A H R Y G K T U C P W E M

If the identifier on each document is known before the first fine-sort pass, an optimized sorting pattern for the particular identifiers present can be developed.

Rearranging Equation 1

$$p = \frac{\log M}{\log S}$$

in this case

$$\frac{\log 15}{\log 3} = 2.5$$

Thus the optimum number of fine-sort passes is 2.5. For simplicity, the sorting pattern is generated for the next higher whole number of passes, in this case, three. If it is assumed that the stackers are numbered 0, 1, 2, the sorting pattern needed is shown in Table 2. The technique for allocating pocket numbers for each document on each fine-sort pass can be inferred from the table.

The document images are sorted into the desired output sequence; then stacker allocations are appended to each image in sequence. For an S stacker-sorter, with a "lot" of checks requiring p passes, a number consisting of p digits is appended. The radix in each column equals the number of stackers available in

Table 3 Sorting pattern in input sequence for first fine-sort pass

Final pass output sequence	*Document identity*	*Pocket number*		
		Pass 3	*Pass 2*	*Pass 1*
11	S	1	0	1
6	J	0	1	2
1	A	0	0	0
5	H	0	1	1
10	R	1	0	0
15	Y	1	1	2
4	G	0	1	0
7	K	0	2	0
12	T	1	0	2
13	U	1	1	0
2	C	0	0	1
9	P	0	2	2
14	W	1	1	1
3	E	0	0	2
8	M	0	2	1

that pass. Thus, in our example, a three-digit number to the base three is required for each image. The values are in the range 000_3 to 112_3.

After stacker allocation, the images are sorted back to the expected input order for the first fine-sort pass. These images then form the Predicted Input Flow List (PIFL) for this pass. Table 3 shows the sequence.

The stacker select routine then merely selects the incoming documents to the given stacker numbers going sequentially down the right-hand column. (How the images are matched and the inevitable errors are handled is described later near the end of the paper.)

The input sequence for the next pass will be as in Table 4. The PIFL for this pass is created in very much the same way as the checks are sorted. The PIFL for the preceding pass is read and the check records with their pocket numbers are written to the pocket files using the special access method. These pocket files are concatenated to form the PIFL for the next pass which is used for sorting and for creating the PIFL for the final pass shown in Table 5. After this pass, the documents will be in the required alphabetic sequence.

practical considerations

We have described the theory on which the sorting pattern is based. We next describe solutions to the practical problems of gathering the data from which to create the tables, of generating the predicted input flow lists (PIFL) for each pass, and of associating the stacker select command inside the computer with each physical check outside of it.

Table 4 Sorting pattern input sequence for second fine-sort pass

Final pass output sequence	*Document identity*	*Pocket Number* *Pass 3*	*Pass 2*	
1	A	0	0	
10	R	1	0	From
4	G	0	1	pocket 0
7	K	0	2	on pass 1
13	U	1	1	
11	S	1	0	
5	H	0	1	From
2	C	0	0	pocket 1
14	W	1	1	on pass 1
8	M	0	2	
6	J	0	1	
15	Y	1	1	From
12	T	1	0	pocket 2
9	P	0	2	on pass 1
3	E	0	0	

Table 5 Sorting pattern in final pass input sequence

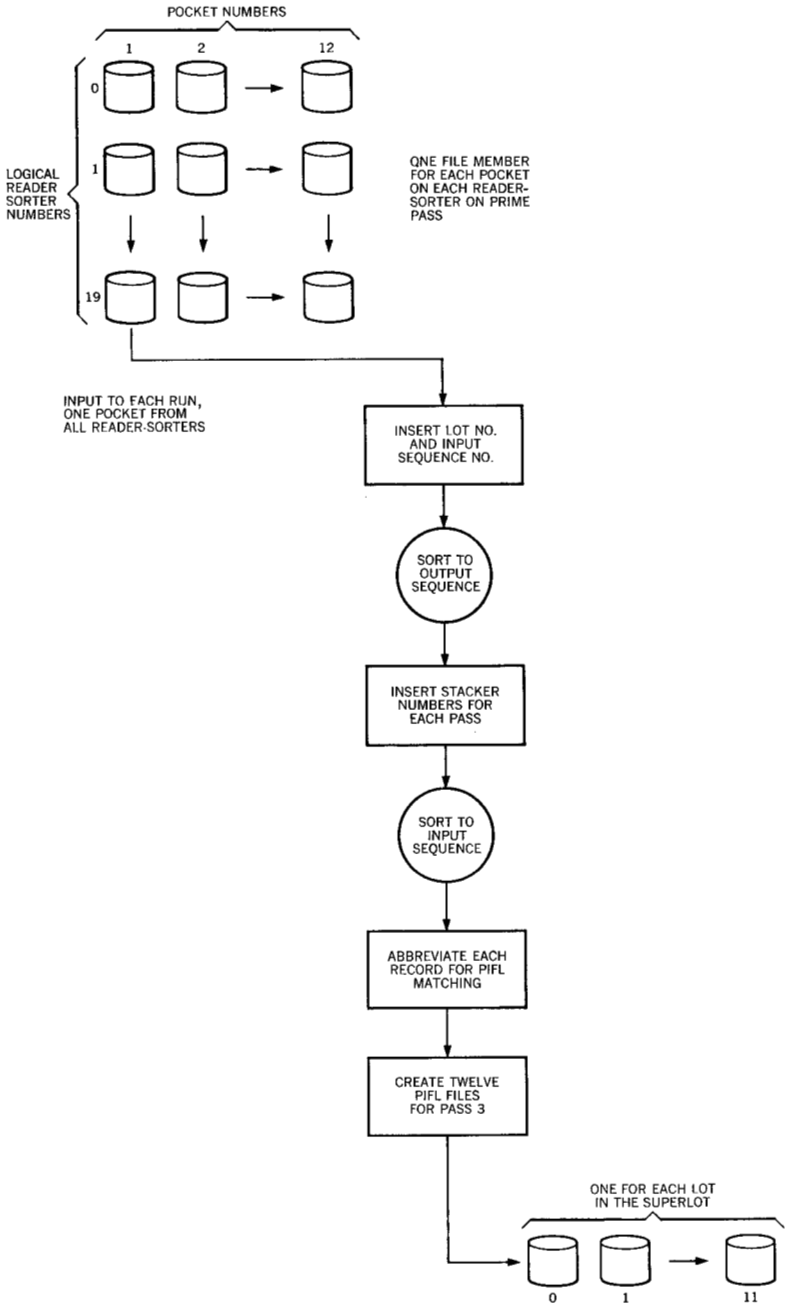
Output sequence	*Document identity*	*Pocket Number* *Pass 3*	
1	A	0	
10	R	1	From
11	S	1	pocket 0
2	C	0	on pass 2
12	T	1	
3	E	0	
4	G	0	
13	U	1	From
5	H	0	pocket 1
14	W	1	on pass 2
6	J	0	
15	Y	1	
7	K	0	From
8	M	0	pocket 2
9	P	0	on pass 2

generating the sort tables

The sequence of operations for generating the fine-sort tables is shown in Figure 6 as:

1. Insert the lot and input sequence number into each check record.
2. Sort the records to the required output sequence.
3. Insert the stacker numbers for each pass into each check record.
4. Re-sort all the records back to the input sequence for pass 3.
5. Shorten the records and write the pass 3 PIFL files.

Figure 6 Fine-sort table generation



The records on the pocket files written by the prime pass contain the complete check image, the input sequence number, and the number of the pocket to which the check was sorted on prime pass. To these must be added the pass 2 pocket number since the pass 1 and pass 2 pocket numbers together make up the lot number.

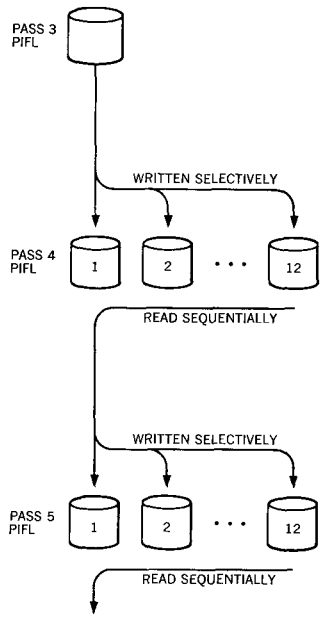
All the records are then sorted into the required output sequence. During the input phase, the records in each lot are counted so that the sort pattern can be calculated in the exit from the output phase. The stacker numbers for up to five fine-sort passes are inserted in each record according to the algorithm already described. The records are finally re-sorted back to the sequence in which they are presented to pass 3.

PIFL files

The records are then abbreviated to contain only that data actually required for the subsequent sorting. The check data is reduced to the low order half of each serial number, account number, and branch number and packed to reduce the length of the subsequent compare instruction. The sequence fields for re-sort to input are also dropped. This represents the PIFL (predicted input flow list) for pass 3.

The pass 4 PIFL is created by reading the pass 3 file sequentially and copying the records to one of 12 pocket files according to the stacker for which the check will be selected on pass 3. The pass 5 PIFL is similarly created by concatenating the pass 3 pocket files and splitting them according to the pass 4 stacker selections. (See Figure 7.) The compacted check images on the PIFL should be in exactly the same sequence as the check file input to each pass, but inevitably there are discrepancies.

Figure 7 Splitting to the PIFL for each pass



CHECK IMAGES ARE WRITTEN TO 12 MEMBERS CORRESPONDING TO THE POCKETS INTO WHICH THEY WILL BE SORTED ON PASS 3. THE 12 MEMBERS ARE THEN CONCATENATED TO FORM THE PIFL FOR PASS 4. THIS IS THEN SPLIT ACROSS A FURTHER 12 MEMBERS TO FORM THE PIFL FOR THE NEXT PASS, AND SO ON.

The Model 40 receives an MSB from the work scheduler specifying the lot number to be processed (and in the case of a restart, the starting check sequence number within that lot). The Model 40 then retrieves the PIFL data for the particular lot and stores it on an IBM 2311 disk before engaging the reader-sorter allocated. Sufficient PIFL data is held in main storage to overlap reader-sorter operation and retrieval of PIFL data from the disk.

To allow for documents rejected on previous passes, the image comparison routine includes a forward search of the PIFL file of up to five images, should the document not match the first image. One feature of the system is that the documents are distributed approximately equally between the pockets. This means that if a block of 24 documents were rejected on one pass, only two, on average, will be missing from each pocket, and a forward search of five positions usually enables the correct comparison to be made.

If a forward search of five positions on the PIFL is still unsuccessful, the document is rejected, and the search for the next image is resumed at the original point of failure. If five successive documents are rejected for this reason, all subsequent documents are rejected until the next control voucher is read and the PIFL can be realigned. Major mishandling of complete pockets would be detected as a control voucher sequence error, causing the reader sorter to be disengaged immediately.

Obviously a comparison of only a part of the check code line with the compacted PIFL image could result in incorrect sorting, but the probability of that happening is less than 1 in 10^{10}.

operational problems

The document-handling operation in such a system is particularly important as the predicted input sequence must be preserved. To reduce the possibility of operator errors (which can occur as early as stacking the output from pass 1), a system of sequence control vouchers is used.

If a control voucher sequence error is detected, the Model 40 disengages the reader-sorter, and the operator's visual display terminal indicates the number that was expected. The operator can then correct the input stream, but the reader-sorter can only be restarted by a command keyed in on the supervisor's terminal.

In spite of these difficulties, experience gained so far shows that, if operator training is good, this sorting technique is viable, and the reject rate per pass is at least as low as on more conventional systems. The total volume of rejects is less due to the much smaller number of passes.

Summary comment

The main objectives of this banking operation—to provide a full check-sorting service to the bank branches and to maintain tight control of the operation—led to the system and, in particular, to the special features just described.

The sorting technique provides an improved service to many branches, without the need for time-consuming, off-line sorting. The special access method gives a means of real-time collation of several thousand data set members, enabling the financial accounting operations to be very closely controlled. The restart and recovery methods enable the operation to meet delivery deadlines consistently, in spite of the occasional equipment malfunction which is inevitable on a system of this size.

ACKNOWLEDGMENTS

The authors wish to thank Mr. B. J. Keyte, Head of Data Processing, National Westminster Bank, for his personal help in reviewing this article. Our thanks are also due to many of his staff for the opportunity to discuss the presentation of some of the more detailed sections.

REFERENCES

1. *Requirement for Automatic Cheque Processing*, Committee of London Bankers Publication ESC/7 (August 1962). (Due for revision; new publication expected by the end of 1972.)

2. *1219 Reader Sorter—1419 Magnetic Character Reader*, No. GA24-1499, IBM Corporation, Data Processing Division, White Plains, New York. Describes the timing and programming techniques required for the IBM 1419 check reader-sorter.
3. *OS/360 Supervisor and Data Management Services*, No. C28-6646, IBM Corporation, Data Processing Division, White Plains, New York.
4. *Integrated Check-Processing System at the Federal Reserve Bank of Philadelphia*, No. K20-0523, IBM Corporation, Data Processing Division, White Plains, New York. Describes another use of image matching.