SLAC-PUB-614
(MISC)

Computer Organization with Streaming Memory*

John V. Levy

Computer Science Department and Stanford Linear Accelerator Center
Stanford University, Stanford, California

March 1969

Submitted to the Navy Postgraduate School Symposium on Parallel Processor Systems, Technology, and Applications, Monterey, California, June 25-27, 1969.

*Work supported by U. S. Atomic Energy Commission.

Abstract

Disks and drums are not really random-access devices. A new device is proposed, combining each disk or drum track with integrated circuit or core memory buffers, which yields a sector-wide random-access "window" into the track contents. The sequential nature of the device is acknowledged in providing for a forced swapping algorithm. Three potential applications are described.

In a small multiprogramming system, the buffers are addressed directly by the processor. Size-limited programs are stacked in vertical slices around the drum or disk. In a larger multiprogramming system, address mapping is included, bringing the system closer to conventional page-swapping systems.

Various retrieval modes in large data base systems are described. A system is proposed having numerous local processors, each handling retrieval from a group of disk tracks. The application of this system to the various modes of retrieval in large data files is discussed.

Introduction

Computer systems both large and small suffer chronically from a Parkinson's Law[1] of the core memory: programs and data sets expand to fill the space allotted to store them. This phenomenon has led to provision of multi-level storage, where one can have orders of magnitude larger storage capacity at a cost of increased access time. Such storage is usually on rotating magnetic recording devices which cost orders of magnitude less than core memory, and will probably continue to do so for the foreseeable future. Along with this cure, however, come the difficult problems of allocation, paging, segmentation, swapping algorithms, memory fragmentation, and scheduling.

Second-level storage devices such as disks and drums are not strictly random-access media, despite nomenclature to the contrary. They yield sequential strings of characters or bits of which only one (or a few) are accessible at a time, and only at fixed periodic intervals. Most systems using disks and drums attempt to ignore this fact, regarding the devices instead as "random-access", with the access time being a stochastic variable having a predictable mean value.

Proposed below is a drum-like device which encourages the natural sequential use of its contents by presenting a wide random-access "window" into its bit string. The window is stepped along the string at periodic intervals, with no delay between "views".

Three computer organizations are described which have memory based on this device, but used in different ways when applied to small and large multiprogramming systems, and to accessing a large data base.

A Rotating Memory with Random-Access Windows

Consider one track of a disk memory system, as shown schematically in Fig. 1. This track is composed of n sectors (shown in Fig. 1 with $n = 6$). [Note that sector is used here to denote a part of an annulus, not a true "sector" as used in geometry.] With a fixed-head system, each track has a read/write head permanently positioned over it. Unfortunately, this restricts the access to the recording medium (the track surface) to one character (or one bit) at a time, and the characters or words may be transferred in only one direction (either reading or writing) in a given sequence. It is these constraints we wish to overcome.

Let us generalize the track shown above to a ring of k segments, which we shall still call "sectors". Thinking of a sector not as a recording surface, but as a string of stored characters or bits, suppose we decide that random-access to a sector is a desirable quality. Implementations are given below for a system with the following characteristics:

1. The ring is composed of k equal size sectors.
2. The period of rotation is $k\tau$.
3. Read-write random access is provided to an entire sector for a fixed duration, τ sec. At the end of this time, the entire next sector (in rotational sequence) immediately becomes accessible in place of the former. At the end of another period of τ sec, the next successive sector becomes accessible, and so on.

Implementation 1.

Let us suppose we have a disk unit with k sectors per track and m characters per sector. We shall use two buffers of m words each, which may be two integrated-circuit memories, or contained in a single core memory.

Separate read and write heads (fixed) are required in this implementation, with the heads spaced two sectors apart as shown schematically in Fig. 2. At the instant shown in Fig. 2, the contents of sector 6 are in buffer β and are accessible to the "user" or processor. Buffer α contains part of sector 5 contents, which are being written back onto sector 5, while concurrently it is being filled with the contents of sector 1. (Spacing between the two heads should be slightly more than two sectors, to allow time to write-out slightly ahead of the read-in. Or this could be handled by a few words of extra buffering in the read or write data paths.) At the end of the next sector rotation time, access is switched over to sector 1 contents in buffer α, and the read-write operation is performed on buffer β, to sectors 2 and 6, respectively.

This implementation has an advantage over some others, in that the old contents of a sector are preserved on the disk during the time of random access to its copy in a buffer. Thus, one could decide not to rewrite the buffer copy if, for example, a failure or exception occurred during processing with that sector. Furthermore, the sector contents are rewritten onto the same physical sector so that hard failures in the recording medium will tend to cause consistent errors in the data.

Implementation 2.

Suppose we are given a conventional fixed-head disk. How can the device above be simulated?

First, it is clear that the read-write operation must be split into two sequential operations. Thus we envision a basic access interval to be twice the sector rotation interval.

If equal and consistent time intervals are to be preserved, then the track must have an odd number of sectors. On a track with $2n+1$ sectors, a ring of $2n-1$ sectors can be implemented. Consider a track with 7 sectors and one read/write head. Five logical sectors, A through E, are recorded on it, as shown in Fig. 3.

The operations are performed as described in Table I.

This procedure does cause the contents of the logical sectors to precess around on the physical sectors. This makes error recovery difficult and also predisposes the system to time and location interdependencies for errors on the physical track.

One track and its associated buffers may be represented schematically as shown in Fig. 4. The random-access window is a bi-directional data path which switches from one buffer to the other at the end of each time interval, trading buffers with the read-write transfer paths. Thus the pair of buffers appears to the user (processor or data channel) as if it were a single buffer whose contents change instantaneously at the end of the time interval.

Application to a "small job" multiprogramming system.

Now suppose we have a number of tracks on a disk or drum with read-write heads and buffers as described above. The "windows" may be connected to form a contiguous address space for a processor, as diagrammed in Fig. 5. The buffers may be implemented as part of a single core memory, or as physicallyremote integrated circuit or core memories. Transfer bandwidth requirements will probably require one of the latter.

In Fig. 5 is represented a system with 16,384 words of memory of which half are in a conventional "static" core memory, and the other half are composed of 128 buffer windows of 64 words each. The processor operates

TABLE I

time	operation	after the operation α contains	after the operation β contains	give access to
0			E	β
1	read A(1) into α	A	E	α
2	write E onto 2	A	E	α
3	Read B(3) into β	A	B	β
4	write A onto 4	A	B	β
5	read C(5) into α	C	B	α
6	write B onto 6	C	B	α
7	read D(7) into β	C	D	β
8	write C onto 1	C	D	β
9	read E(2) into α	E	D	α
10	write D onto 3	E	D	α
11	read A(4) into β	E	A	β
12	write E onto 5	E	A	β
13	read B(6) into α	B	A	α
14	write A onto 1	B	A	α

normally out of the static part of its memory; this part will contain the monitor programs, compilers, interruption and input/output control. The other half of memory will contain user programs or text which have been previously placed on the disk or drum tracks.

One user job is allocated a "vertical" slice through the disk or drum -- that is, a group of sectors which arrive in the windows at the same time, up to as many as one sector per track. Jobs are stacked in this way around the disk or drum, as diagrammed in Fig. 6.

As a job appears in the upper half of the processor's address space, it picks up the status of the job (register contents, etc), and begins processing it. Just before the end of the sector-swap time interval, an interruption to the processor causes it to store the status, preparatory to resuming work on the next one.

There are several crucial questions relevant to the efficiency of this system organization:

1. How does one avoid inordinate waste of processor time when premature termination of the job occurs to await I/O?

Several factors mitigate this problem. First, one might do considerable buffering of I/O into the static part of memory, allowing asynchronous data channels to do the peripheral communication later. Second, by combining track transfer paths (and adding more read and write heads), one can effectively speed the transfer rate to the windows, shortening the window "view" interval. The price of doing this relates only to the cost of the disk track hardware, since the window addressing would remain the same. Thus the view interval is adjustable over a wide range.

Furthermore, since the swapping of views involves no time overhead to the processor, one would tend to have a very short view interval. It is

only necessary that the context-switching (register loading) time of the processor be small compared to the view interval. Thus, if a context switch could be accomplished in 5 microseconds, one might have a view interval as short as several hundred microseconds. With such a short interval, the probability of performing a significant proportion of useful job processing in each interval becomes quite high.

2. What happens when a job will not fit in a vertical "slice"?

The system described here will require that a job be bounded in size by the window view size. A number of successful small-scale time-sharing systems (e.g., SDS 940, PDP-1 and PDP-6) require a similar restriction on user programs. There are some interesting algorithms for shuffling sectors on the disk or drum, which will not be explored here. The section below on an application to large-program systems develops algorithms which are closer to conventional paging techniques.

3. When the disk or drum is not fully loaded with jobs, won't a great amount of the round-robin cycle time be wasted on empty views?

By a simple modification of the control of the window buffers, one could choose at any interval end not to swap buffers, thus causing the old window view to be retained. In this way, a job may be given a number of successive view intervals of processing, while waiting for another active job to come into p ition. (During this time, jobs and vacant vertical slices on the drum or disk would be physically advanced by one slice, due to the short-circuiting of one buffer load in the "slice-stream".) In the case of non-self-modifying programs, one might be able to retain multiple copies in various positions on the disk or drum..

Application to a "large job" multiprogramming system

To expand the organization above to make it useful for large program processing, two modifications are imposed. An address-mapping mechanism is provided between the windows and the main processor, and read-access into the alternate (unseen-half) buffer of each window is provided so that a processor may monitor the contents of the buffers in read-write transition. In Fig. 7, this is shown as a separate processor connected by a bus to all of the alternate buffers. These buffers are still instantaneously interchangeable with their counterparts which are connected to the main processor addressing bus, but it is now proposed that the page control processor have control over the interchange, separately for each window.

Several modes of operation are possible with this organization. When jobs will fit into one vertical slice (a single set of windows), the page control processor can be used to set up, in the alternate buffers, the next job, no matter where on the disk or drum the pages were located. If certain restrictions on arrangement were observed, a job could always be set up in one rotation.

Another mode would allow the main processor to recognize need for and to request specific pages from the disk or drum. Using the alternate buffers, the page control processor would locate the page, interchange it with a buffer on the "active" side, and set up an appropriate address mapping. This could go on while the main processor is executing the job.

This arrangement has little advantage over conventional page-swapping multiprogramming systems. It could perhaps be simplified by eliminating the address mapping, allowing instead the page control processor to transfer pages among the alternate-side buffers. In this way, one would have an effective, although primitive, swapping arrangement. The power of this

system lies in its use along with a further extension described below, in which pages are requested by content rather than location.

Application to retrieval from a large data base

Information retrieval and processing in large data bases is becoming more and more widespread, yet computer organization has not changed radically to reflect the different nature of the problem. In a large storage system, one must resort to large quantities of second-level media, such as disks, drums, photo-digital stores, data cells, magnetic tapes. To provide large amounts of on-line read-write storage, disks are primarily used.

Large data file systems, such as bank files, airline reservations, library information retrieval, on-line experimental scientific data, and inventory systems operate in several different modes. Some of these are:

1. Serial processing of large files, performing updates on the entire file in one pass.
2. Individual random retrieval of unrelated items from the file, with or without updating the items.
3. Extraction of a large number of items which have a common attribute (sometimes creating a subfile of these items).
4. Linking-search, in which the key to the next item sought is not known until the first item is located, and so on.
5. Sort-merge processing on files and subfiles.

Conventional computer systems are notably inefficient in modes 2 and 4, because of random-access demands on the file system, and mode 3 is usually equivalent to mode 1. Fig. 8 shows a diagram of a computer organization in which access to large disk-based files is under the control of a number of independent local processors, each one associated with a group of disk

tracks and windows. This system is capable of highly parallel search activity and is therefore particularly well suited to retrieval modes 2 and 4. It also permits unconventional file structuring on the disks.

Let us postulate that information is stored in units of "records" which are larger than one word, but smaller than one window width. Let us also assume that the system is serving a community of users who are located at remote terminals, and that their requests for retrieval are being multi-programmed in some manner. Such a user community is most likely to be interacting in modes 2, 3, and 4.

A simple request for retrieval of a single item (mode 2) will be broadcast from the main processor on the request bus and added to the queue of requests at each local processor. (If the physical location is known, the request can be directed to the single appropriate processor.) This queue is a random-access list which is maintained individually by each local processor, probably in its core memory. The local processor then can scan the current queue of requests, compare with the window contents (or previously known physical addresses), and return on the result bus the item requested, when it appears. If necessary during result transfer, the window "view" may be frozen, as described in the previous section. When a request has been satisfied, it is deleted from the local queue, and if it had been broadcast to more than one processor, the one which satisfied the request will broadcast a deletion signal to the others on the request bus. For data transfer from the main processor to a specified location on a disk, the process is similar, but the item must either be buffered at the local processor, or else a pre-emptive transfer on the result bus will be required by the processor which finds the entry location.

With such an organization as that shown here, however, it is feasible to leave file structuring to the local processor. For an example, one could

allow completely unstructured files in a system where all items are effectively independent of each other. Thus in retrieval mode 3, for example, in retrieving subject-related references in a library index, all local processors would concurrently search all tracks for records containing the appropriate keywords. Here we have introduced a new level of responsibility for the local processor -- that of selecting the records by their content.

The level of sophistication in the local processor can be carried further. In retrieval mode 4, such as for tree searching, one can imagine having the local processor not only select a record on the basis of its contents, but also perform an algorithm to determine which, if any, successors to the record are required to be retrieved. This could be done without sending the intermediate results to the main processor. The successive requests could be sent to other processors on the request bus, if necessary.

Several general advantages obtain with the organization proposed here. First, the ability to maintain (physically) relatively unstructured files allows simple insertion and deletion of items in the files. One can of course take advantage of the serial nature of the disks by arranging related sequences in physical proximity; this would probably be worthwhile for access modes 1 and 5. Since the file maintenance is performed by the local processor, one could allow different structuring in different disk groups. Second, placing the local processor and memory at the disk helps to eliminate the inordinate volume of data transfer into main memory which is characteristic of present file-handling systems. Third, this effective isolation of the data selection and retrieval from the main processor allows convenient connection of different memory media at each local processor, providing a standard interface to the main processor. This idea

is not new[2] but the organization shown here is probably the first to be oriented toward the modes of access described above. Fourth, one can see that by adding direct addressability from the main processor to the windows, the large-program multiprogramming capability could be implemented within this organization.

Some questions of implementation are still unresolved. For example, the format of the retrieval request must take on a fair complexity in the case when the local processor is expected both to find and to update a record. What is the format of the request? How and when is the update information passed to the local processor? How is acknowledgement of success or failure handled at the main processor? Also, how are multiple responses to a single retrieval request to be handled?

Summary

Disks and drums as second-level storage media seem likely to be with us for a long time. It seems economically feasible to append sufficient random-access memory (integrated-circuit or core) to implement the "window" device described here.

The purpose of the device is to force on the system designer and user an awareness of the sequential nature of disks and drums and at the same time to take advantage of that nature. Since memory systems are clearly to be the bottlenecks in future system organizations, it is worthwhile having device limitations brought out clearly. Thus in designing a multiprogramming system organization, we have reverted to an old concept of drum-oriented program layout; yet the result may be economical.

For large data base information retrieval, it is clear that systems must be organized with the particular retrieval algorithms in mind. We have tried to show how several processors operating in parallel on a system of window-disk files may be an effective organization for some retrieval requirements.

Acknowledgement

The author wishes to acknowledge a series of fruitful discussions on large data base problems with Professor W. F. Miller, Victor Lesser, William Riddle, and Dr. Harold Stone.

References

1. C. Northcote Parkinson, Parkinson's Law, (Boston: Houghton-Mifflin Co.), 1957.

2. Control Data Corporation, "CDC 7600 Preliminary Reference Manual", (St. Paul, Minn.), 1968, section 5.

Bibliography

1. Catt, I., E. C. Garth, and D. E. Murray, "A High-Speed Integrated Circuit Scratchpad Memory," AFIPS Conference Proceedings, vol 29 (1966), pp. 315-331 (Fall Joint Computer Conference).

2. Control Data Corporation, "CDC 7600 Preliminary Reference Manual," (St. Paul, Minn.) 1968.

3. Denning, P. J., "The Working Set Model for Program Behavior," Comm.ACM vol. 11, no. 5 (May 1968), pp. 323-333.

4. Evans, D. C., and J. Y. Leclerc, "Address Mapping and the control of access in an interactive computer," AFIPS Conference Proceedings, vol. 30, (1967), pp 23-30 (Spring Joint Computer Conference).

5. Levy, J. V., "A Rotating Memory with Random-Access Ports," Stanford Linear Accelerator Center, Computation Group internal memo, Nov. 15, 1968.

6. McFarland, K., and M. Hashiguchi, "Laser recording unit for high density permanent digital data storage," AFIPS Conference Proceedings, vol. 33, part 2, (1968), pp. 1369-1380 (Fall Joint Computer Conference).

7. Pirtle, M., "Intercommunication of Processors and Memory," AFIPS Conference Proceedings (Washington: Thompson Book Co.), vol. 31 (1967), pp. 621-633 (Fall Joint Computer Conference).

8. Randell, B., and C. J. Kuehner, "Dynamic Storage Allocation Systems", Comm. ACM. vol. 11, no. 5 (May 1968), pp. 297-306.

9. _____, "Demand Paging in Perspective," AFIPS Conference Proceedings, vol. 33, part 2 (1968), pp. 1011-1018 (Fall Joint Computer Conference).

10. Scientific Data Systems, "SDS 940 Computer Reference Manual," (Santa Monica, California) 1966.

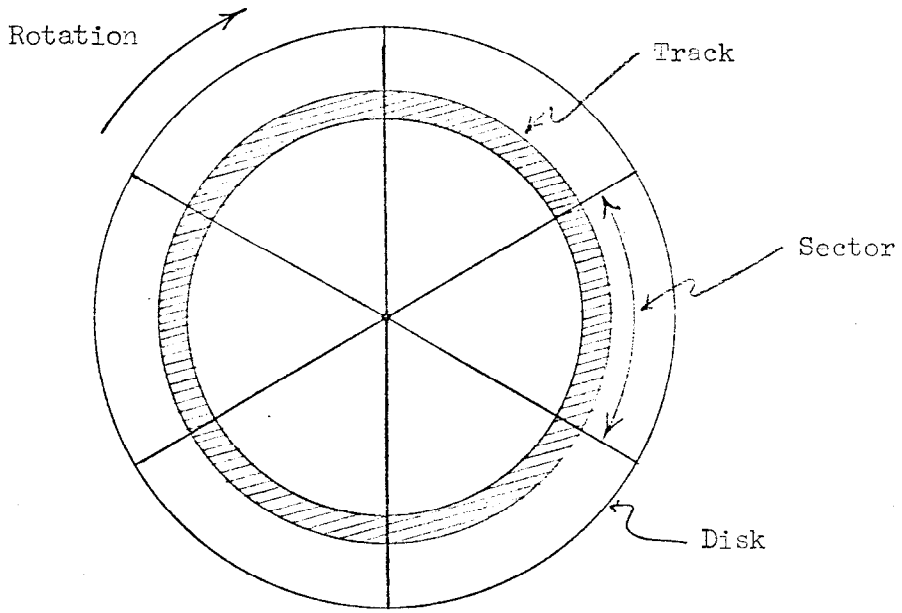

Figure 1 Schematic Layout of Disk Storage

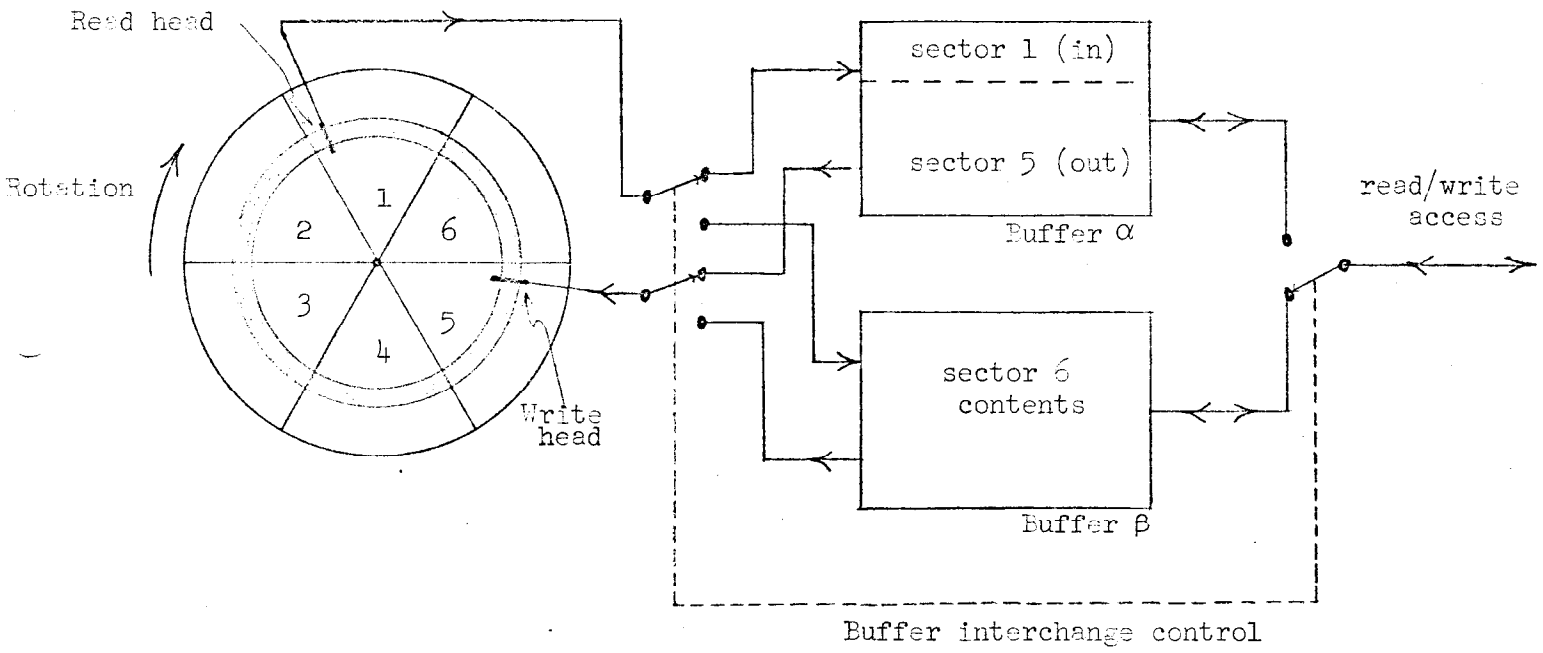

Figure 2 Implementation Using Two Heads

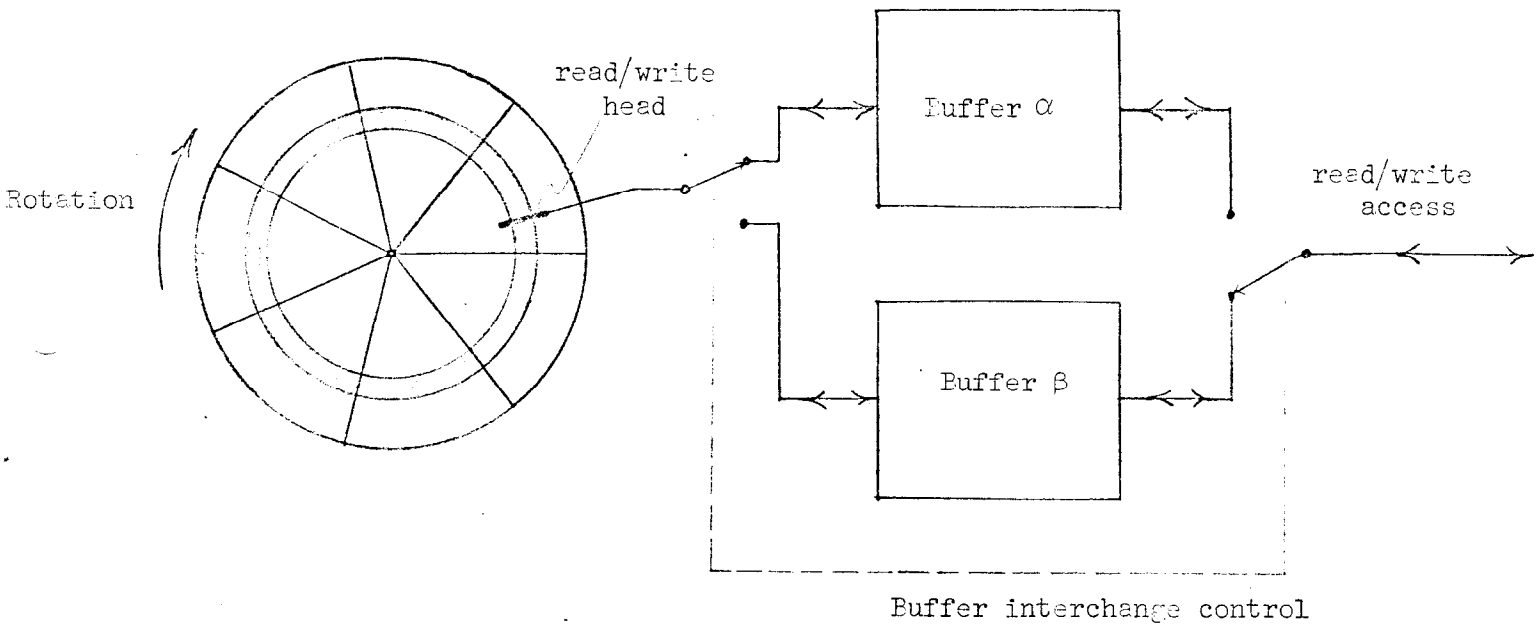

Figure 3 Implementation Using One Head

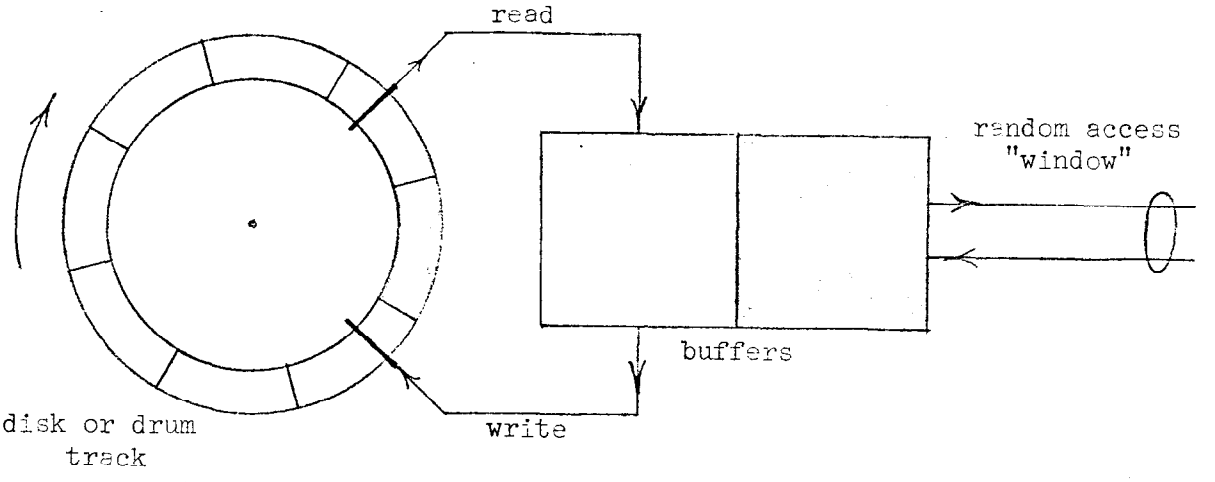

Figure 4 Schematic "window" Access Path

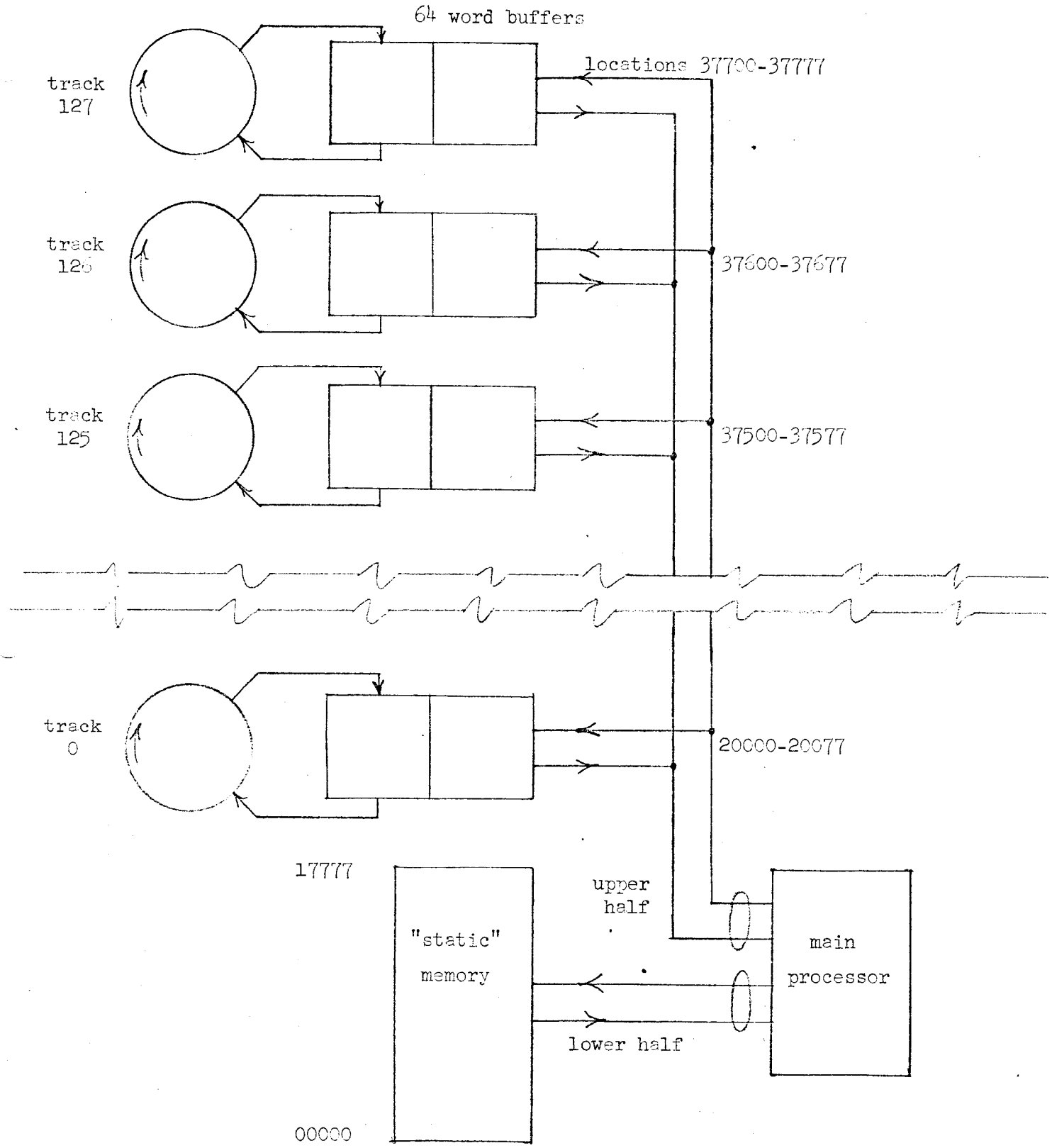

Figure 5 Computer Organization for a Small-job Multiprogramming System

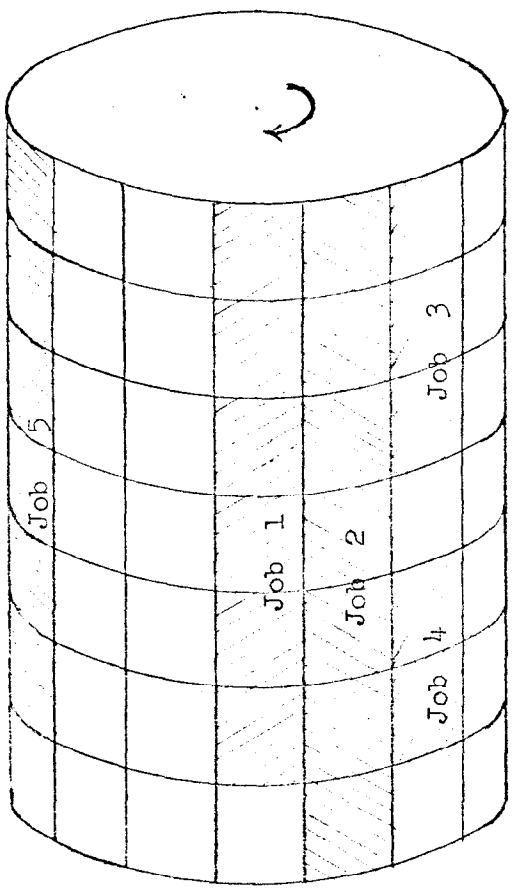

Figure 6 Stacking of Jobs in a Small-job Multiprogramming System

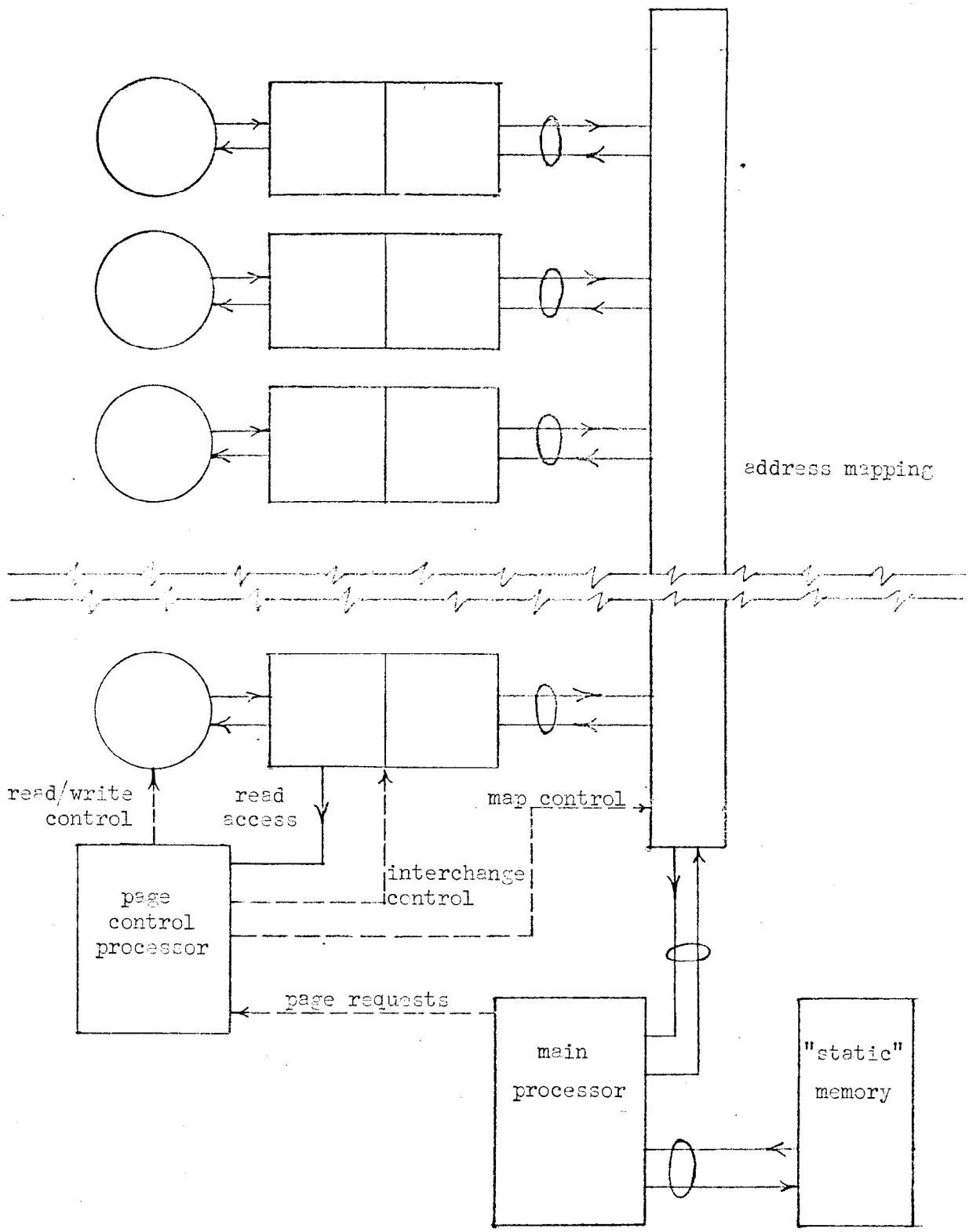

Figure 7 Computer Organization for a Large-job Multiprogramming System

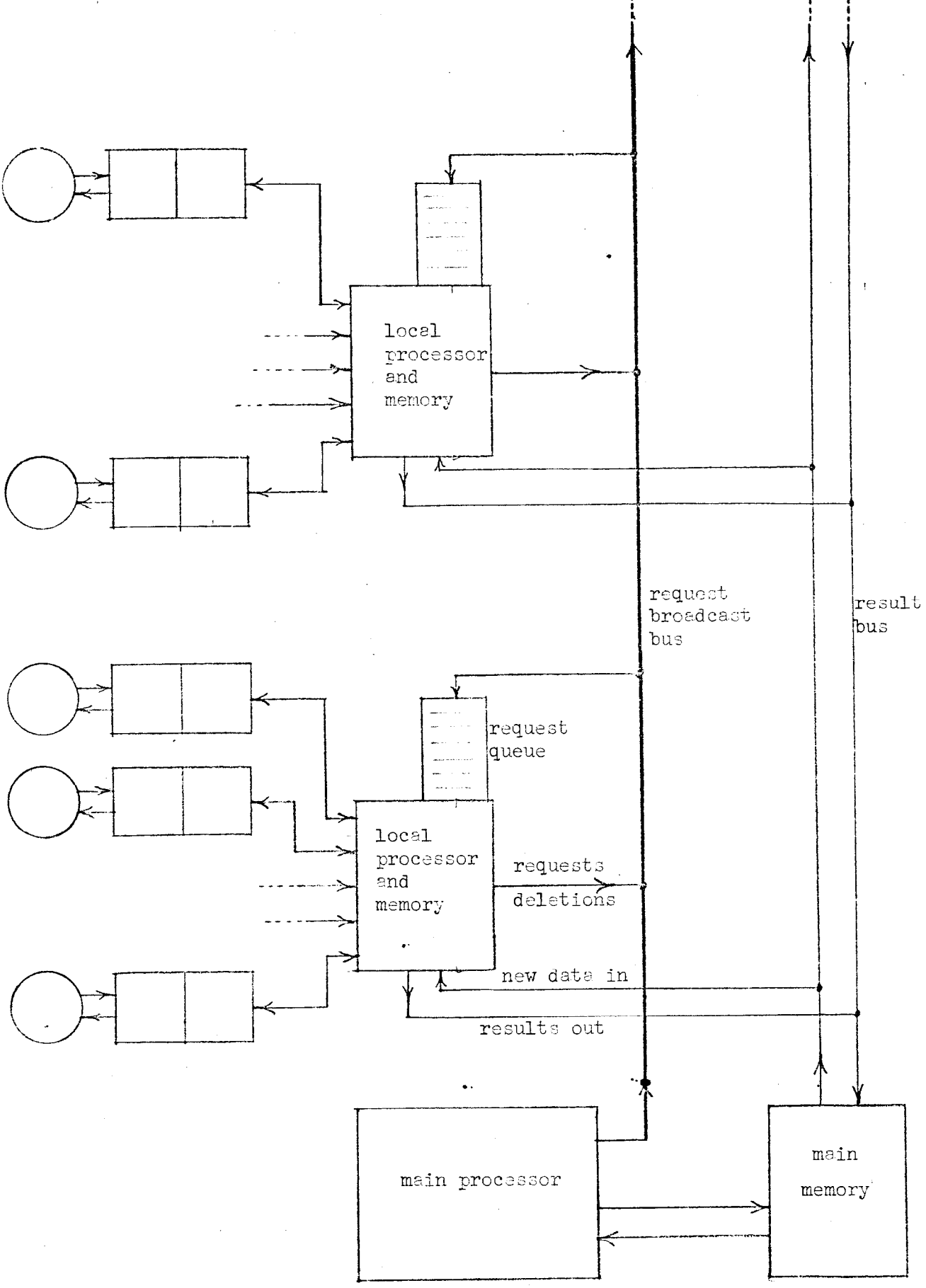

Figure 8 Computer Organisation for Retrieval from Large Data Files