

# One-Level Storage System<sup>1</sup>

*T. Kilburn / D. B. G. Edwards / M. J. Lanigan / F. H. Sumner*

**Summary** After a brief survey of the basic Atlas machine, the paper describes an automatic system which in principle can be applied to any combination of two storage systems so that the combination can be regarded by the machine user as a single level. The actual system described relates to a fast core store-drum combination. The effect of the system on instruction times is illustrated, and the tape transfer system is also introduced since it fits basically in through the same hardware. The scheme incorporates a "learning" program, a technique which can be of greater importance in future computers.

## 1. Introduction

In a universal high-speed digital computer it is necessary to have a large-capacity fast-access main store. While more efficient operation of the computer can be achieved by making this store all of one type, this step is scarcely practical for the storage capacities now being considered. For example, on Atlas it is possible to address  $10^6$  words in the main store. In practice on the first installation at Manchester University a total of  $10^5$  words are provided, but though it is just technically feasible to make this in one level it is much more economical to provide a core store (16,000 words) and drum (96,000 words) combination.

Atlas is a machine which operates its peripheral equipment on a time division basis, the equipment "interrupting" the normal main program when it requires attention. Organization of the peripheral equipment is also done by program so that many programs can be contained in the store of the machine at the same time. This technique can also be extended to include several main programs as well as the smaller subroutines used for controlling peripherals. For these reasons as well as the fact that some orders take a variable time depending on the exact numbers involved, it is not really feasible to "optimum" program transfers of information between the two levels of store, i.e., core store and drum, in order to eliminate the long drum access time of 6 msec. Hence a system has been devised to make the core drum store combination appear to the programmer as a single level of storage, the requisite transfers of information taking place automatically. There are a number of additional benefits derived from the scheme adopted, which include relative addressing so that routines can operate anywhere in the store, and a "lock out" facility to prevent interference between different programs simultaneously held in the store.

## 2. The Basic Machine

The arrangement of the basic machine is shown in Fig. 1. The available storage space is split into three sections; the private store which is used solely for internal machine organization, the central store which includes both core and drum store, in which all words are addressed and is the store available to the normal user, and finally the tape store, which is the conventional backing-up large capacity store of the machine. Both the private store and the main core store are linked with the main accumulator, the B-store, and the B-arithmetic unit. However the drum and tape stores only have access to these latter sections of the machine via the main core store.

The machine order code is of the single address type, and a comprehensive range of basic functions are provided by normal engineering methods. Also available to the programmer are a number of extra functions termed "extracodes" which give automatic access to and subsequent return from a large number of built-in subroutines. These routines provide

<sup>1</sup> A number of orders which would be expensive to provide in the machine both in terms

of equipment and also time because of the extra loading on certain circuits. An example of this is the order:

Shift accumulator contents  $\pm n$  places where  $a$  is an integer.

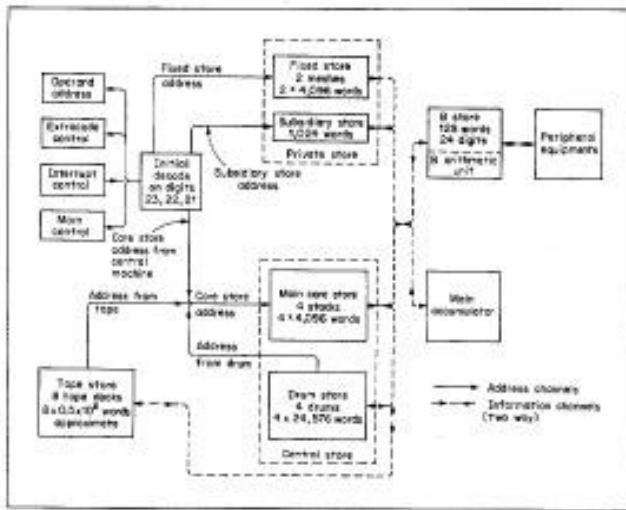


Fig. 1. Layout of basic machine.

1

- 2 The more complex mathematical operations, e.g.,  $\sin x$ ,  $\log x$ , etc.
- 3 Control orders for peripheral equipments, card readers, parallel printers, etc.
- 4 Input-output conversion routines.
- 5 Special programs concerned with storage allocation to different programs being run simultaneously, monitoring routines for fault finding and costing purposes, and the detailed organization of drum and tape transfers.

All this information is permanently required and hence is kept in part of the private store termed the "fixed store" [Kilburn and Grimsdale, 1960] which operates on a read only" basis. This store consists of a woven wire mesh into which a pattern of small "linear" ferrite slugs are inserted to represent digital information. The information content can only be changed manually and will tend to differ only in detail between the different versions of the Atlas computer. In Muse this store is arranged in two units each of 4096 words, a unit consisting of 16 columns of 256 words, each word being 50 bits. The access time to a word in any one column is about  $0.4 \mu$  sec. If a change of column address is required, this figure increases by about  $1 \mu$  sec due to switching transients in the read amplifiers. Subsequent accesses in the new column revert to  $0.9 \mu$  sec. The store operates in conjunction with a subsidiary core store of 1024 words which provides working space for the fixed store programs, and has a cycle time of about  $1.8 \mu$  sec. There are certain safeguards against a normal machine user gaining access to addresses in either part of the private store, though in effect he makes use of this store through the extracode facility.

The central store of the machine consists of a drum and core store combination, which has a maximum addressable capacity of about  $10^6$  words. In Muse the central store capacity is about 96,000 words contained on 4 drums. Any part of this store can be transferred in blocks of 512 words to/from the main core store, Which consists of four separate stacks, each stack having a capacity of 4096 words.

The tape system provides a very large capacity backing store for the machine. The user can effect transfers of variable amounts of information between this store and the central store. In actual fact such transfers are organized by a fixed store program which initiates automatic transfers of blocks of 512 words between the tape store and the main core store. The system can handle eight tape decks running simultaneously, each producing or demanding a word on average every 88  $\mu$  sec.

The main core store address can thus be provided from either the central machine, the drum, or the tape system. Since there is no synchronization between these addresses, there has to be a priority system to allocate addresses to the core store. The drum has top priority since it delivers a word every 4  $\mu$  sec, the tape next priority since words can arise every 11  $\mu$  sec from 8 decks and the machine uses the core store for the rest of the available time. A priority system necessarily takes time to establish its priority, and so it has been arranged that it comes into effect only at each drum or tape request. Thus the machine is not slowed down in any way when no drum or tape transfers take place. The effect of drum and tape transfers on machine speed is given in Appendix 1.

To simplify the control commands given to the drum, tape, and peripheral equipment in the machine, the orders all take the form  $b \rightarrow S$  or  $s \rightarrow B$  and the identification of the required command register is provided by the address  $S$ . This type of storage is clearly widely scattered in the machine but is termed collectively the  $V$ -store.

In the central machine the main accumulator contains a fast adder [Kilburn, et al., 1960b] and has built-in multiplication and division facilities. It can deal with fixed or floating point numbers and its operation is completely independent of the  $B$ -store and  $B$ -arithmetic unit. The  $B$ -store is a fast core store (cycle time 0.7  $\mu$  sec) of 120 twenty-four bit words operating in a word selected partial flux switching mode [Edwards et al., 1960]. Eight "fast"  $B$  lines are also provided in the form of flip-flop registers. Of these, three are used as control lines, termed main, extracode, and interrupt controls respectively. The arrangement has the advantage that the control numbers can be manipulated by the normal  $B$ -type orders, and the existence of three controls permits the machine to switch rapidly from one to another without having to transfer control numbers to the core store. Main control is used when the central machine is obeying the current program, while the extracode control is concerned with the fixed store subroutines. The interrupt control provides the means for handling numerous peripheral equipments which "interrupt" the machine when they either require or are providing information. The remaining "fast"  $B$  lines are mainly used for organizational procedures, though  $B124$  is the floating point accumulator exponent.

The operating speed of the machine is of the order of  $0.5 \times 10^6$  instructions per second. This is achieved by the use of fast transistor logic circuitry, rapid access to storage locations, and an extensive overlapping technique. The latter procedure is made possible by the provision of a number of intermediate buffer storage registers, separate access mechanisms to the individual units of core store and parallel operation of the main accumulator and  $B$ -arithmetic units. The word length throughout the machine is 48 bits which may be considered as two half-words of 24 bits each. All store transfers between the central machine, the drum and tape stores are parity checked, there being a parity digit associated with each half-word. In the case of transfers within the central store (*i. e.*, between main core store and drum) the parity digits associated with a given word are retained throughout the system. Tape transfers are parity checked when information is transferred to and from the main core store, and on the tape itself a check sum technique involving the use of two closely spaced heads is used.

The form of the instruction, which allows for two  $B$ - modifications, and the allocation of the address digits is shown in Fig. 2a. Half of the addressable store locations are allocated to the central store which is identified by a zero in the most significant digit of the address. (See Fig. 2b.) This address can be further subdivided into block address and line address in a block of 512 words. The least significant

digits, 0 and 1, make it possible to address 6 bit characters in a half word and digit 2 specifies the half word.

The function number is split into several sections, each section relating to a particular set of operations, and these are listed in Fig. 2c. The machine orders fall into two broad classes, and these are:

1 *B codes*: These involve operations between a *B* line specified by the  $B_A$  digits in the instruction and a core store line whose address can be modified by the contents of a *B* line determined by the  $B_m$  digits. There are a total of 128 *B* lines, one of which,  $B_0$ , always contains zero. Of the other lines 90 are available to the machine user, 7 are special registers previously mentioned, and a further 30 are used by extracode orders.

2 *A Codes*: These involve operations between the Accumulator and a core store line whose address can now be doubly modified first by contents of  $B_m$  and then by the contents of  $B_A$ . Both fixed and floating point orders are provided, and in the latter case numbers take the form of  $X8^Y$ , the digit allocation of X and Y being shown in Fig. 2d. When fixed point working occurs, use is made only of the X digits.

### 3. One-Level Store Concept

The choice of system for the fast access store in a large scale computer is governed by a number of conflicting factors which include speed and size requirements, economic and technical difficulties. Previously the problem has been resolved in two extreme cases either by the provision of a very large core store, e.g., the 2.5 megabit [Papian, 1957] store at M.I.T., or by the use of a small core store (40,000 bits) expanded to 640,000 bits by a drum store as in the Ferranti Mercury [Lonsdale and Warburton, 1956; Kilburn et al., 1956] computer. Each of these methods has its disadvantages, in the first case, that of expense, and in the second case, that of inconvenience to the user, who is obliged 'to program transfers of information between the two types of store and this can be time consuming. In some instances it is possible for an expert machine user to arrange his program so that the amount of time lost by the transfers in the two-level storage

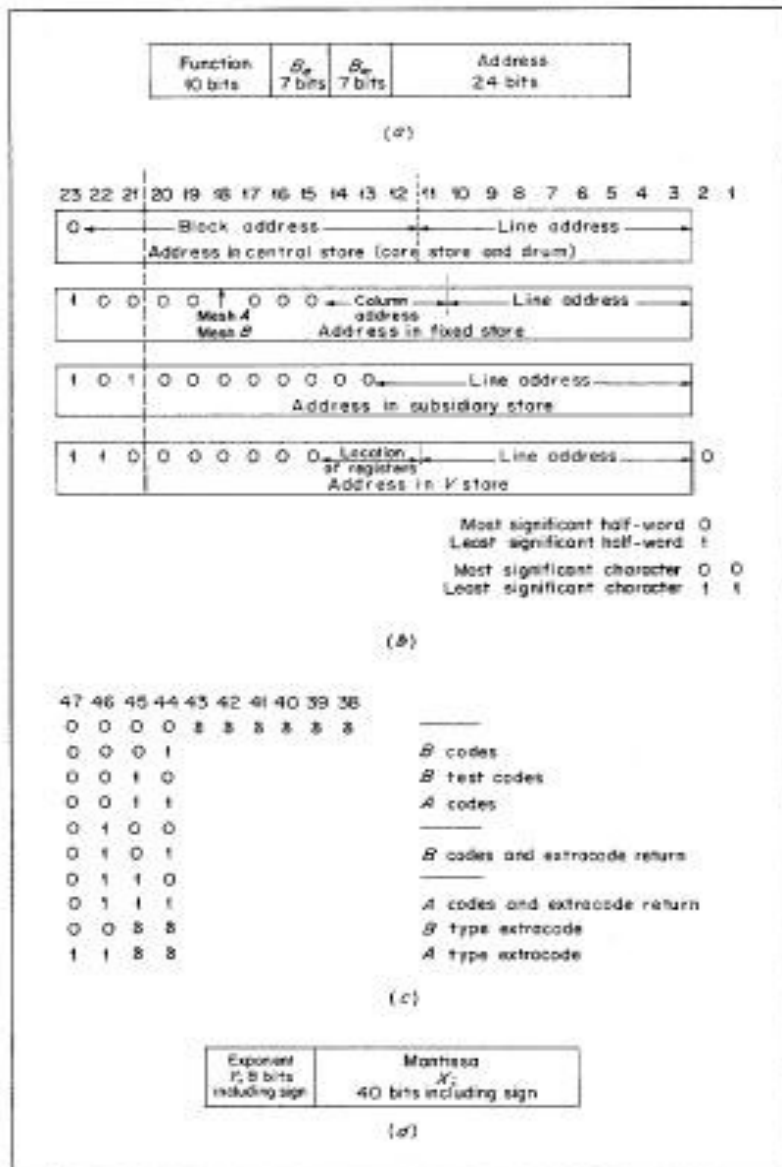


Fig. 2. Interpretation of a word. (a) Form of instruction. (b) Allocation of address digits. (c) Function of decoding. (d) Floating-point number  $XB^r$ .

arrangement is not significant, but this sort of "optimum" programming is not very desirable. Suitable interpretative coding [Brooker, 1960] can permit the two-level system to appear as one level. The effect is, however, accompanied by an effective loss of machine speed which, in some programs and depending on details of machine design, can be quite severe, varying typically, for example, between one and three.

The two-level storage scheme has obvious economic advantages, and inconvenience to the machine user can be eliminated by making the transfer arrangements completely automatic. In Atlas a completely automatic system has been provided with techniques for minimizing the transfer times. In this way the core and drum are merged into an apparent single level of storage with good performance and at moderate cost. Some details of this arrangement on the Muse are now provided.

The central store is subdivided into blocks of 512 words as shown by the address arrangements in Fig. 2b. The main core store is also partitioned into blocks of this size which for identification purposes are

called pages. Associated with each of these core store page positions is a "page address register" (P. A. R.) which contains the address of the block of information at present occupying that page position. When access to any word in the central store is required, the digits of the demanded block address are compared with the contents of all the page address registers. If an "equivalence" indication is obtained, then access to that particular page position is permitted. Since a block can occupy any one of the 32 page positions in the core store, **it** is necessary to modify some digits of the demanded block address to conform with the page positions in which an equivalence was obtained.

These processes are necessarily time consuming but by providing a by-pass of this procedure for instruction accesses (since, in general, instruction loops are all contained in the same block) then most of this time can be overlapped with a useful portion of the machine or core store rhythm. In this way information in the core store is available to the machine at the full speed of the core store and only rarely is the over-all machine speed affected by delays in the equivalence circuitry.

If a "not equivalence" indication is obtained when the demanded block address is compared with the contents of the PAR's, then that address, which may have been B-modified, is first stored in a register which can be accessed as a line of the V-store. This permits the central machine easy access to this address. An "interrupt" also occurs which switches operation of the machine over to the interrupt control, which first determines the cause of the interrupt and then, in this instance, enters a fixed store routine to organize the necessary transfers of information between drum and core store.

### ***A. Drum Transfers***

On each drum, one track is used to identify absolute block positions around the drum periphery. The records on these tracks are read into the *O* registers which can be accessed as lines of the V-store and this permits the present angular drum position to be determined, though only in units of one block. In this way the time needed to transfer any block while reading from the drums can be assessed. This time varies between 2 and 14 msec since the drum revolution time is 12 msec and the actual transfer time 2 msec.

The time of a writing transfer to the drums has been reduced by writing the block of information to the first available empty block position on any drum. Thus the access time of the drum can be eliminated provided there are a reasonable number of empty blocks on the drum. This means, however, that transfers to/from the drum have to be carried out by reference to a directory and this is stored in the subsidiary store and up-dated whenever a transfer occurs.

When the drum transfer routine is entered the first action is to determine the absolute position on a drum of the required block. The order is then given to carry out the transfer to an empty page position in the core store. The transfer occurs automatically as soon as the drum reaches the correct angular position. The page address register in the vacant position in the core store is set to a specific block number for drum transfers. This technique simplifies the engineering with regard to the provision of this number from the drum and also provides a safeguard against transferring to the wrong block.

As soon as the order asking for a read transfer from the drum has been given, the machine continues with the drum transfer program. It is now concerned with determining a block to be transferred back from the core store to the drum. This is necessary to ensure an empty core store page position when the next read transfer is required. The block in the core store to be transferred has to be carefully chosen to minimize the number of transfers in the program and this optimization process is carried out by a learning program, details of which are given in Sec. 5. The operation of this program is assisted by the provision of the "use" digits which are associated with each page position of the core store.

To interchange information between the core store and drums, two transfers, a read from and a write to the drum, are necessary. These have to be done sequentially but could occur in either order. The

technique of having a vacant page position in the core store permits a read transfer to occur first and thus allows the time for the learning program to be overlapped either into the waiting period for the read transfer or into the transfer time itself. In the time remaining after completion of the learning program an entry is made into the over-all supervisor program for the machine, and a decision is taken concerning what the machine is to do until the drum transfer is completed. This might involve a change to a different main program.

A program could ask for access to information in a page position while a drum or tape transfer is taking place to that page. This is prevented in Atlas by the use of a "lock out" (L. 0.) digit which is provided with each Page Address Register. When a lock out digit is set at 1, access to that page is permitted only when the address has been provided either by the drum system, the tape system, or the interrupt control. The last case permits all transfers from paper tape, punched card, and other peripheral equipments, to be handled without interference from the main program. When the transfer of a block has been completed, the organizing program resets the LO. digit to zero and access to that page position can then be made from the central machine. It is clear that the LO. digit can also be used to prevent interference between programs when several different ones are being held in the machine at the same time.

In Sec. 3 it was stated that addresses demanding access to the core store could arise from three distinct sources, the central machine, the drum, and the tape. These accesses are complicated because of (1) the equivalence technique, and (2) the lock out digit. The various cases and the action that takes place are summarized in Table 1.

The provision of the Page Address Registers, the equivalence circuitry, and the learning program have permitted the core store and drum to be regarded by the ordinary machine user as a one-level store, and the system has the additional feature of "floating address" operation, i.e., any block of information can be stored in any absolute position in either core or drum store. The minimum access time to information in this store is obviously limited by the core store and its arrangement, and this is now discussed.

### ***B. Core Store Arrangement***

The core store is split into four stacks, each with individual address decoding and read and write mechanisms. The stacks are then combined in such a way that common channels into the machine for the address, read and write digits, are time shared between the various stacks. Sequential address positions occur in two stacks alternately and a page position which contains a block of 512 sequential addresses is thus arranged across two stacks. In this way it is possible to read a pair of instructions from consecutive addresses in parallel by increasing the size of the read channel. This permits two instructions to be completely obeyed in three store "accesses." The choice of this particular storage arrangement is discussed in Appendix 2.

The coordination of these four stacks is done by the "core stack coordinator" and some features of this are now discussed, starting with the operation of a single stack.

### ***C. Operation of a Single Stack of Core Store***

The storage system employed is a coincident current M.I.T. system arranged to give parallel read out of 50 digits. The reading operation is destructive and each read phase of the stack cycle is followed by a write phase during which the information read out may be rewritten. This is achieved by a set of digit staticizers which are loaded during the read phase and are used to control the inhibit current drivers during the write phase. When new information is to be written into the store, a similar sequence is followed, except that the digit staticizers are loaded with the new information during the read phase. A diagram indicating the different types of stack cycle is shown in Fig. 3.

There is a small delay  $W_D$  ( $\sim 100$  nsec) between the "stack request" signal,  $SR$ , and the start of the read phase to allow for setting of the address state and the address decoding. The output information from the store appears in the read strobe period, which is towards the end of the read phase. In general, the write phase starts as soon as the read phase ends. However, the start of the write phase may be held up until the new information is available from the central machine. This delay is shown as  $W_w$  in Fig. 3c. The interval  $T_A$  between the stack request and the read strobe is termed the stack access time, and in practice this is approximately one-third of the cycle time  $T_C$ . Both  $T_A$  and  $T_C$  are functions of the storage system and assuming that  $W_w$  is zero have typical values of  $0.7 \mu$  sec and  $1.9 \mu$  sec respectively. A holdup gate in the request channel prevents the next stack request occurring before the end of the preceding write phase.

#### **D. Operation of the Main Core Store with the Central Machine**

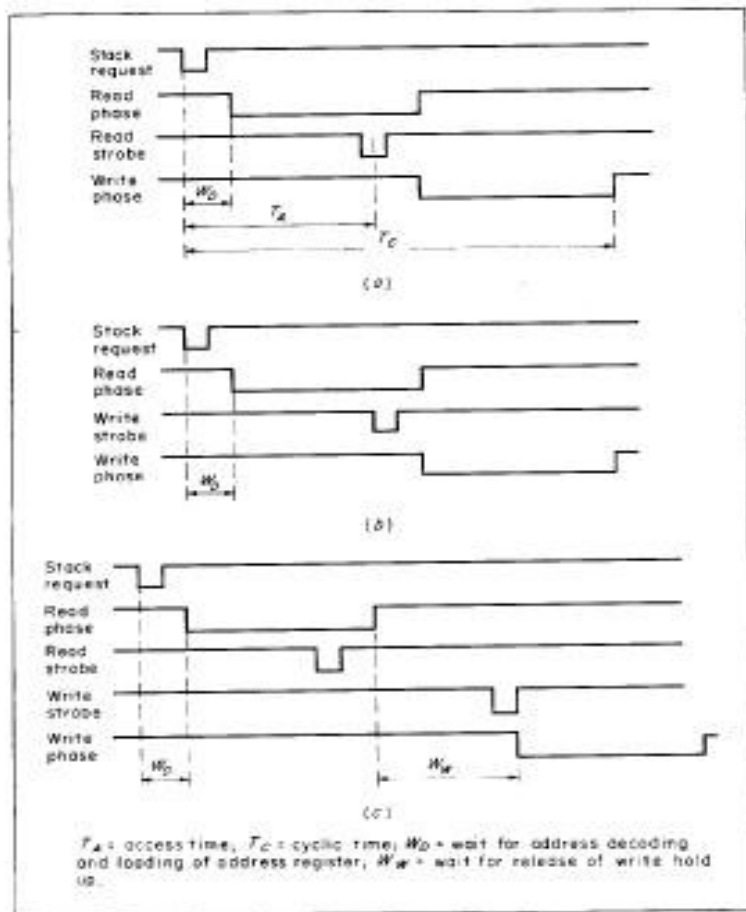
A schematic diagram of the essentials of the main core store control system is shown in Fig. 4. The control signals  $SA_1$  and  $SA_2$  indicate whether the address presented is that of a single word or a pair of sequentially addressed instructions. Assuming that the flip-flop  $F$  is in the reset condition, either of these signals results in the loading of the buffer address register (BAR.). This loading is done by the signal B. A.B.A. which also indicates that the buffer register in the central machine has become free.

In dealing with the first request the block address digits in the B. A. R. are compared with the contents of all the page address registers. Then one of the indications summarized in Table 1 and

**Table 1 Comparison of Demanded Block Address with Contents of the P.A.R.'s Resultant State of Equivalence and Lock Out Circuits**

<i>Source of address</i>	<i>{Equivalence Lock out = 0} [E.Q.]</i>	<i>Not equivalence [N.E.Q.]</i>	<i>{Equivalence Lock out = 1} [EQ. &amp; LO.]</i>
1. Central Machine	Access to required page position	Enter drum transfer routine	Not available to this program
2. Drum System	Access to required page position	Fault condition indicated	Fault condition indicated
3. Tape System	Access to required page position	Fault condition indicated	Fault condition





**Fig. 3. Basic types of stack cycle (a) Read order (s→A). (b) Write order (a→s). (c) Read-write order (b + s→S).**

indicated in Fig. 4 is obtained. Assuming access to the required store stack is permitted, then a set C.S.F. signal is given which resets the flip-flop  $F$ . If this occurs before the next access request arises, then the speed of the system is not store-limited. In most cases SET CSF is generated when the equivalence operation on the demanded block address is complete, and the read phase of the appropriate stack (or stacks) has started. Until this time the information held in the B.A.R. must not be allowed to change. In Fig. 5 a flow diagram is shown for the various cases which can arise in practice.

When a single address request is accepted, **it** is necessary to obtain an "equivalence" indication and form the page location digits before the stack request can be generated. The SET CSF signal then occurs as soon as the read phase starts. If a "not equivalent" or "equivalent and locked out" indication is obtained, a stack request is not generated and the contents of the B.A.R. are copied in to a line of the V-store before SET CSF is generated.

When access to a pair of addresses is requested (i.e., an instruction pair), the stack requests are generated on the assumption that these instructions are located in the same page position as the last pair requested, i.e., the page position digits are taken from the page digit register. (See Fig. 4.) In this way the time required to obtain the equivalent indication and form the page location digits is not included in the over-all access time of the

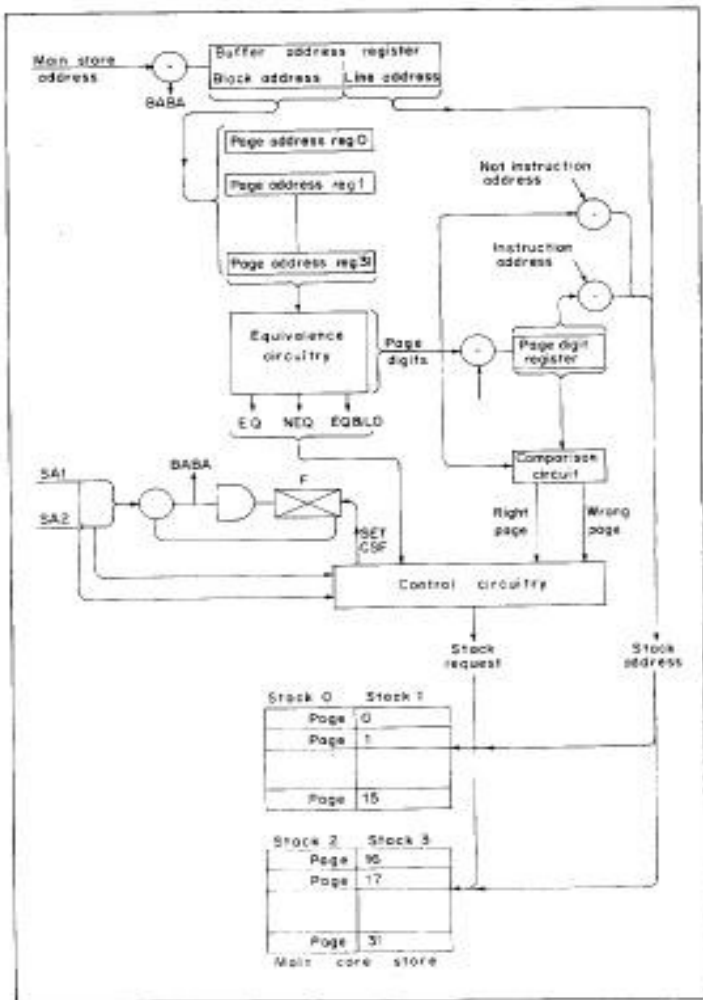


Fig. 4. Main core store control.

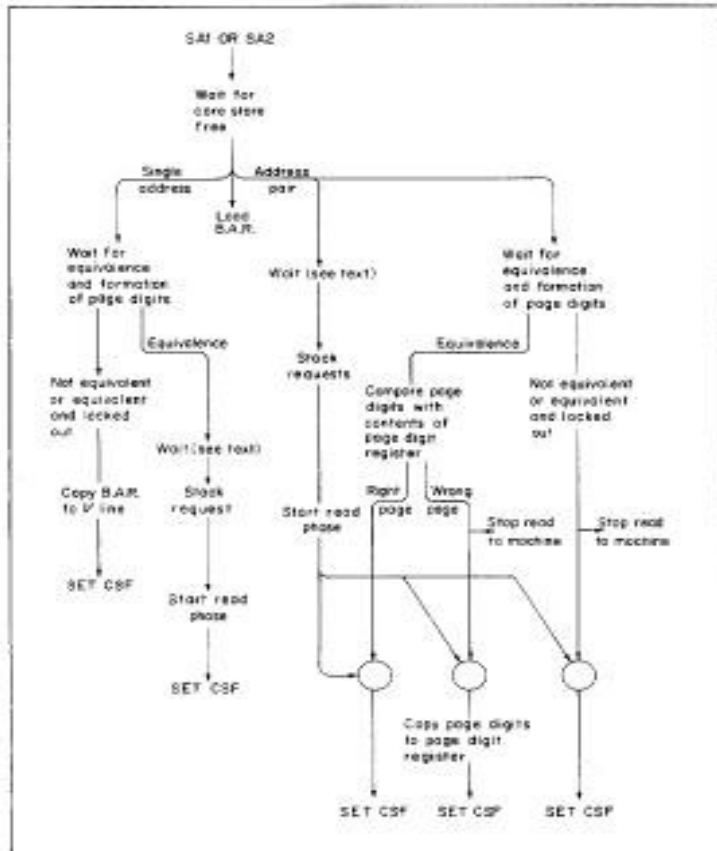


Fig. 5. Flow diagram of main core store control.

system. The assumption will normally be true, except when crossing block boundaries. The latter cases are detected and corrected by comparing the true position page digits obtained as a result of the equivalence operation with the contents of the page digit register, and a "right page" or "wrong page" indication is obtained. (See Fig. 4.) If a wrong page is accessed this is indicated to the central machine and the read out is inhibited. The true page location digits are copied into the page digit register, so that the required instruction pair will be obtained when next requested. The read out to the central machine is also inhibited for "not equivalent" or "equivalent and locked out" indications.

In Fig. 5 the waiting time indicated immediately before the stack request is generated can arise for a number of reasons:

- 1 The preceding write phase of that stack has not yet finished.
- 2 The central machine is not yet ready either to accept information from the store or to supply information to it.
- 3 It is necessary to ensure a certain minimum time between successive read strobes from the core stacks to allow satisfactory operation of the parity circuits, which take about 0.4  $\mu$  sec to check the information. This time could be reduced, but as it is only possible to get such a condition for a small part of the normal instruction timing cycle it was not thought to be an economical proposition.

The basic machine timing is now discussed.

#### 4. Instruction Times

In high-speed computers, one of the main factors limiting speed of operation is the store cycle time. Here a number of techniques, e.g., splitting the core store into four separate stacks and extracting two instructions in a single cycle, have been adopted despite a fast basic cycle time of 2  $\mu$  sec in order to alleviate this situation. The time taken to complete an instruction is dependent upon

- 1 The type of instruction (which is defined by the function digits)
- 2 The exact location of the instruction and operand in the core or fixed store since this can affect the access time
- 3 Whether or not the operand address is to be modified
- 4 In the case of floating point accumulator orders, the actual numbers themselves
- 5 Whether drum and/or tape transfers are taking place

The approximate times for various instructions are given in Table 2. These figures relate to the times between completing instructions when a long sequence of the same type of instruction is obeyed. While this method is not ideal, it is necessary because in practice obeying one instruction is overlapped in time with some part of three other instructions. This makes the detailed timing complicated, and so the timing sequence is developed slowly by first considering instructions obeyed one after another. It is convenient to make these instructions a sequence of floating point additions with both instruction and operand in the core store and with the operand address single B-modified.

To obey this instruction the central machine makes two requests to the core store, one for the instruction and the second for the operand. After the instruction is received in the machine the function part has to be decoded and the operand address modified by the contents of one of the *B* registers before the operand request can be made. Finally, after the operand has been obtained the actual accumulator addition takes place to complete

**Table 2 Approximate Instruction Times**

<i>Type of instruction</i>	<i>Number of modifications of address</i>	<i>Instruction in core store. Operands in core store. Time(<math>\mu</math> sec)</i>	<i>Instructions in fixed store. Operands in core store. Time(<math>\mu</math> sec)</i>	<i>Instructions in fixed store. Operands in fixed store. Time (<math>\mu</math> sec)</i>
Floating Point Addition	0	1.4	1.65	1.2
	1	1.6	1.65	1.2
	2	2.03	1.9	1.9
Floating Point Multiplication	0, 1 or 2	4.7	4.7	4.7

Floating Point Division	0,1 or 2	13.6	13.6	13.6
Add Store Line to an Index Register	0	1.53	1.65	1.15
	1	1.85	1.85	1.85
Add Index Register to Store Line and Rewrite to	0	1.63	1.65	
Store Line	1	1.8	1.7	

the instruction. The time from beginning to end of one instruction is 6.05  $\mu$  sec and an approximate timing schedule is as follows in Table 3.

If no other action is permitted in the time required to complete the instruction (steps 1 to 8 in Table 3), then the different sections of the machine are being used very inefficiently, e.g., the accumulator adder is used only for less than 1.1  $\mu$  sec. However, the organization of the computer is such that the different sections,

**Table 3\* Timing Sequence for Floating Point Addition (Instructions and Operands in the Core Store)**

<i>Sequence</i>	<i>Time interval between steps (<math>\mu</math> sec)</i>	<i>Total time (<math>\mu</math> sec)</i>
1. Add 1 to Main Control		0
(Addition time)	0.3	
2. Make Instruction Request		0.3
(Transfer times, equivalence time and stack access time)	1.75	
3. Receive Instruction in Central Machine		2.05
(Load register and decode)	0.2	

4. Function decoding complete		2.25
(Single address modification)	0.85	
5. Request Operand		3.10
(Transfer times, equivalence time and stack access time)	1.75	
6. Receive Operand in Central Machine		4.85
(Load register)	0.1	
7. Start Addition in Accumulator		4.95
(Average floating point addition, including shift round and standardise)	1.1	
8. Instruction complete		6.05

\*In step 4, time is for single address modification. Times for no modification and two modifications are 0.25 $\mu$  sec and 1.55  $\mu$  sec respectively.

such as store stacks, accumulator and B-arithmetic unit, can operate at the same time. In this way several instructions can be started before the first has finished, and then the effective instruction time is considerably reduced. There have, of course, to be certain safeguards when, for example, an instruction is dependent in any way on the completion of a preceding instruction.

In the time sequence previously tabulated, by far the longest time was that between a request in the central machine for the core store and the receipt in the central machine of the information from that store. This effective access time of 1.75  $\mu$  sec is made up as shown in Table 4. It has been reduced in practice by the provision of two buffer registers, one in the central machine and the other in the core stack coordinator. These allow the equivalence and transfer times to be overlapped with the organization of requests in the central machine.

In this way, provided the machine can arrange to make requests fast enough, then the effective access time is reduced to 0.8  $\mu$  sec. Further, since three accesses are needed to complete two instructions (one for an instruction pair and one for each of the two operands) the theoretical minimum time of an instruction is 1.2  $\mu$  sec  $3 \times 0.8/2$  and it then becomes store limited. Reference to Table 3 shows that the arithmetic operation takes 1.2  $\mu$  sec to

**Table 4 Effective Store Access Time**

<i>Sequence</i>	<i>Total time (<math>\mu</math> sec)</i>
1. Request in Central Machine	0

2. Request in Core Stack Coordinator	0.25
3. Equivalence complete and request made to selected stack	0.95
4. Information in Core Stack Coordinator	1.65
5. Information in Central Machine	1.75

complete so that, on the average, the capabilities of the store and the accumulator are well matched.

Another technique for reducing store access time for instructions has also been adopted. This permits the read cycles of the two stacks to start assuming that the same page will be referred to as in the previous instruction pair. This, of course, will normally be true and there is sufficient time to take corrective procedures should the page have been changed. The limit of 1.2  $\mu$  sec per instruction is not reduced by this technique, but the possibility of reaching this limit under other conditions is enhanced.

A schematic diagram of the practical timing of a sequence of floating point addition orders is shown in Fig. 6. The overlapping is not perfect and in the time between successive instruction pairs the computer is obeying four instructions for 25 per cent of the time, three for 56 per cent and two for 19 per cent. It is therefore to be expected that the practical time for the complete order is greater than the theoretical minimum time; it is in fact approximately 1.6  $\mu$  sec.

For certain types of functions the reading of the next pair of instructions before completing both instructions of the first pair would be incorrect, e.g., functions causing transfer of control. Such situations are recognized during the function decoding, and the request for the next instruction pair is held up until a suitable time.

In a sequence of floating point addition orders with the operand addresses unmodified the limit is again 1.2  $\mu$  sec while the time obtained is 1.4  $\mu$  sec. For accumulator orders in which the actual accumulator operation imposes a limit in excess of 2  $\mu$  sec then the actual time is equal to this limit.

Perhaps a more realistic way of defining the speed of the computer is to give the time for a typical inner loop of instructions. A frequently occurring operation in matrix work in the

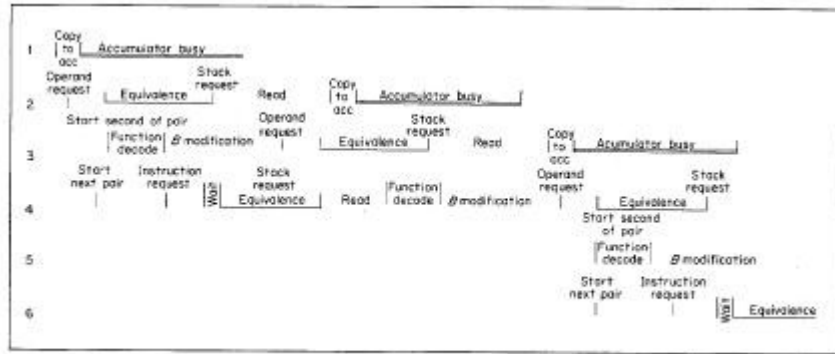


Fig. 6. Timing diagram for a sequence of floating point addition orders. (Single-address modification.)

formation of the scalar product of two vectors, this requires a loop of five instructions:

- 1 Element of first vector into accumulator. (Operand B-modified.)
- 2 Multiply accumulator by element of second vector. (Operand B-modified.)
- 3 Add partial product to accumulator.
- 4 Copy accumulator to store line containing partial product.
- 5 Alter count to select next elements and repeat.

The time for this loop with instructions and operands on the core store is 12.2  $\mu$  sec. The value of the overlapping technique is shown by the fact that the time from starting the first instruction to finishing the second is approximately 10  $\mu$  sec.

When the drum or tape systems are transferring information to or from the core store, then the rate of obeying instructions which also use the core store will be affected. The affect is discussed in more detail in Appendix 1. The degree of slowing down is dependent upon the time at which a drum or tape request occurs relative to machine requests. It also depends on the stacks used by the drum or tape and those being used by the central machine. The approximate slowing down is by a factor of 25 per cent during a drum transfer and by 2 per cent for each active tape channel. (See Appendix 1.)

## 5. The Drum Transfer Learning Program

The organization of drum transfers has been described in Sec. 2A. After the transfer of the required block from the drum to the core store has been initiated, the organizing program examines the state of the core store, and if empty pages still exist, no further action is taken. However, if the core store is full, it is necessary to arrange for an empty page to be made available for use at the next nonequivalence. The selection of the page to be transferred could be made at random; this could easily result in many additional transfers occurring, as the page selected could be one of those in current use or one required in the near future. The ideal selection, which would minimize the total number of transfers, could only be made by the programmer. To make this ideal selection the programmer would have to know (1) precisely how his program operated, which is not always the case, and (2) the precise amount of core store available to his program at any instant. This latter information is not generally available as the core store could be shared by other central machine programs, and almost certainly by some fixed store program organizing the input and output of information from slow peripheral equipments. The amount of core store required by this fixed store program is continuously varying [Kilburn et al., 1961]. The only way the ideal pattern of transfers can be approached is for the transfer program to monitor the behavior of the main program and in so doing attempt to select the correct pages to be transferred to the drum. The techniques used for monitoring are subject to the condition that they must not slow down the operation of the program to such an extent that they offset any reduction in the number of transfers required. The method described occupies less than 1 percent of the operating time, and the reduction in



the number of transfers is more than sufficient to cover this.

That part of the transfer program which organizes the selection of the page to be transferred has been called the "learning" program. In order for this program to have some data on which to operate, the machine has been designed to supply information about the use made of the different pages of the core store by the program being monitored.

With each page of the core store there is associated a "use" digit which is set to "1" whenever any line in that page is accessed. The 32 "use" digits exist in two lines of the V-store and can be read by the learning program, the reading automatically resetting them to zero. The frequency with which these digits are read is governed by a clock which measures not real time but the number of instructions obeyed in the operation of the main program. This clock causes the learning program to copy the "use" digits to a list in the subsidiary store every 1024 instructions. The use of an instruction counter rather than a normal clock to measure "time" for the learning program is due to the fact that the operations of the main program may be interrupted at random for random lengths of time by the operation of peripheral equipments. With an instruction counter the temporal pattern of the blocks used will be the same on successive runs through the same part of the program. This is essential if the learning program is to make use of this pattern to minimize the number of transfers.

When a nonequivalence occurs and after the transfer of the required block has been arranged, the learning program again adds the current values of the "use" digits to the list and then uses this list to bring up to date two sets of times also kept in the subsidiary store. These sets consist of 32 values of  $t$  and  $T$ , one of each for each page of the core store. The value of  $t$  is the length of time since the block in that page has been used. The value of  $T$  is the length of the last period of inactivity of this block. The accuracy of the values of  $t$  and  $T$  is governed by the frequency with which the "use" digits are inspected.

The page to be written to the drum is selected by the application in turn of three simple tests to the values of  $t$  and  $T$ :

- 1 Any page for which  $t > T + 1$ , or
- 2 That page with  $t \neq 0$  and  $(T - t)$  max, or
- 3 That page with  $T_{max}$  (all  $t = 0$ ).

The first rule selects any page which has been currently out of use for longer than its last period of inactivity. Such a page has probably ceased to be used by the program and is therefore an ideal one to be transferred to the drum. The second rule ignores all pages with  $t = 0$  as they are in current use, and then selects the one which, if the pattern of use is maintained, will not be required by the program for the longest time. If the first two rules fail to select a page, the third ensures that if the page finally selected is wrong, in that it is immediately required again; then, as in this case,  $T$  will become zero and the same mistake will not be repeated.

For all the blocks on the drum a list of values of  $\tau$  is kept. The values of  $\tau$  are set when the block is transferred to the drum:

$$\tau = \text{time of transfer} - \text{value of } t \text{ for transferred page}$$

When a block is transferred to the core store, the value of  $\tau$  is used to set the value of  $T$ .

$$T = \text{time of transfer} - \text{value of } \tau \text{ for this block}$$

$$= \text{length of last period of inactivity}$$

for the block transferred from the drum  $t$  is set to 0.

In order to make its decision the learning program has only to update two short lists and apply at the most three simple rules; this can easily be done during the 2 msec transfer time of the block required as a result of the nonequivalence. As the learning program uses only fixed and subsidiary store addresses, it is not slowed down during the period of the drum transfer.

The over-all efficiency of the learning program cannot be known until the complete Atlas system is working. However, the value of the method used has been investigated by simulating the behavior of the one-level store and learning program on the Mercury computer at Manchester University. This has been done for several problems using varying amounts of store in excess of the core store available. One of these was the problem of forming the product  $A$  of two 80th order matrices  $B$  and  $C$ . The three matrices were stored row by row, each one extending over 14 blocks; only 14 pages of core store were assumed to be available. The method of multiplication was

$$b_{11} \times \text{1st row of } C = \text{partial answer to 1st row of } A$$

$$b_{12} \times \text{2nd row of } C + \text{partial answer} = \text{second partial answer, etc.}$$

Thus matrix  $B$  was scanned once, matrix  $C$  80 times and each row of matrix  $A$  80 times.

Several machine users were asked to spend a short time writing a program to organize the transfers for a general matrix multiplication problem. In no case when the method was applied to the above problem were fewer than 357 transfers required. A program written specifically for this problem which paid great attention to the distribution of the rows of the matrices relative to block divisions required 234 transfers. The learning program required 274 transfers; the gain over the human programmer was chiefly due to the fact that the learning program could take full advantage of the occasions when the rows of  $A$  existed entirely within one block.

Many other problems involving cyclic running of single or multiple sets of data were simulated, and in no case did the learning program require more transfers than an experienced human programmer.

### ***A. Prediction of Drum Transfers***

Although the learning program tends to reduce the number of transfers required to a minimum, the transfers which do occur still interrupt the operation of the program for from 2 to 14 msec as they are initiated by nonequivalence interrupts. Some or all of this time loss could be avoided by organizing the transfers in advance. A very experienced programmer having sole use of the core store could arrange his own transfers in such a way that no unnecessary ones ever occurred and no time was ever wasted waiting for transfers to be completed. This would require a great deal of effort and would only be worthwhile for a program that was going to occupy the machine for a long time. By using the data accumulated by the learning program it is possible to recognize simple patterns in the use made by a program of the various blocks of the one-level store. In this way a prediction program could forecast the blocks required in the near future and organize the transfers. By recording the success or failure of these forecasts the program could be made self-improving. For the matrix multiplication problem discussed above the pattern of use of the blocks containing matrix  $C$  is repeated 80 times, and a considerable degree of success could be obtained with a simple prediction program.

## **6. Conclusions**

A specific system for making a core-drum store combination appear as a single level store has been described. While this is the actual system being built for the Atlas machine the principles involved are applicable to combinations of other types of store, for example, a tunnel diode-fast core store combination for an even faster machine. An alternative which was considered for Atlas, but which was not as attractive economically, was a fast core-slow core store combination. The system too can be

extended to three levels of storage, and indeed if  $10^6$  words of total storage had to be provided then it would be most economical to provide it on a third level of store such as a file drum.

The automatic system does require additional equipment and introduces some complexity, since it is necessary to overlap the time taken for address comparison into the store and machine operating time if it is not to introduce any extra time delays. Simulated tests have shown that the organization of drum transfers are reasonably efficient and other advantages which accrue, such as efficient allocation of core storage between different programs and store lock out facilities, are also invaluable. No matter how intelligent a programmer may be, he can never know how many programs or peripheral equipments are in operation when his program is running. The advantage of the automatic system is that it takes into account the state of the machine as it exists at any particular time. Furthermore if as in normal use there is some sort of regular machine rhythm even through several programs, there is the possibility of making some sort of prediction with regard to the transfers necessary. This involves no more hardware and will be done by program. However, this stage will probably be left until results on the actual system are obtained.

It can be seen that the system is both useful and flexible in that it can be modified or extended in the manner previously indicated. Thus despite the increase in equipment, the advantages which are derived completely justify the building of this automatic system.

## **APPENDIX 1 ORGANIZATION OF THE ACCESS REQUESTS TO THE CORE STORE**

There are three sources of access requests to the core store, namely the central machine, the drum, and the tape systems. In deciding how the sequence of requests from all three sources are to be serialized and placed in some sort of order, a number of facts have to be considered. These are

1 All three sources are asynchronous in nature.

2 The drum and tape systems can make requests at a fairly high rate compared with the store cycle time of approximately  $2 \mu$  sec. For example, the drum provides a request every  $4 \mu$  sec and the tape system every  $11 \mu$  sec when all 8 channels are operative.

3 The drum and tape systems can be stopped only in multiples of a block length, i.e., 512 words. This means that any system devised for accessing the core store must deal with both the average rates of drum and tape requests specified in 2. Only the central machine can tolerate requests being stopped at any time and for any length of time. From these facts a request priority can be stated which is

a Drum request.

b Tape request.

c Central machine request.

4 A machine request can be accepted by the core store, but because there is no place available to accept the core store information, its cycle is inhibited and further requests held up. In the case of successive division orders this time can be as long as  $20 \mu$  sec, in which case 5 drum requests could be made. To avoid having an excessive amount of buffer storage for the drum two techniques are possible:

a When drums or tapes are operative do not permit machine requests to be accepted until there is a place available to put the information.

b Store the machine request and then permit a drum or tape request.

The latter scheme has been adopted because it can be accommodated more conveniently and it saves a small amount of time.

5 If the central machine is using the private store then it is desirable for drum and tape transfers to the core store not to interfere with or slow down the central machine in any way.

6 When the central machine, drum and tape are sharing the core store, then the loss of central machine speed should be roughly proportional to the activity of the drum or tape systems. This means that drum or tape requests must "break" into the normal machine request channel as and when required.

The system which accommodates all these points is now discussed. Whenever a drum or tape request occurs, inhibit signals are applied to request channel into the core stack coordinator. This results in a "freezing" of the state of flip-flop *F* (Fig. 5) and this state is then inspected (Fig. 7, point X). If the state is "busy" this means that a machine order has been stopped somewhere between the loading of the buffer address register (B.A.R.) and the stack request. Normally this time interval can vary from about 0.5  $\mu$  sec if there are no stack request holdups to 20  $\mu$  sec in the case of certain accumulator holdups. In either case

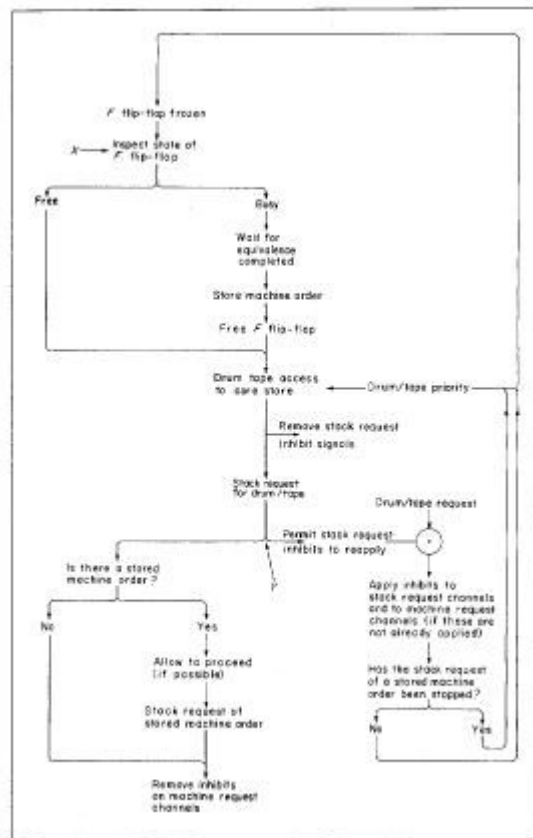


Fig. 7. Drum and tape break in systems.

sufficient time is allowed after the inspection to ensure that the equivalence operation has been completed. If an equivalence indication is obtained, all the information relevant to this machine order (i.e., the line address, page digits, stack(s) required and type of stack order) are stored for future reference. Use is made here of the page digit register provided to allow the by-pass on the equivalence circuitry for instruction accesses. The core store is then made free for access by the drum or the tape. If the core store is found to be free on inspection, the above procedure is omitted.

A drum or tape access (as decided by the priority circuit) to the core store then occurs, which removes the inhibits on the stack request channels. When the stack request for the drum or tape cycle is initiated, these inhibits are allowed to reapply. At this stage (Fig. 7, point Y), if there is a stored machine order it is allowed to proceed if possible. The inhibits on the machine request channels are removed when the stack request for the stored machine order occurs. If there is no stored machine order, this is done immediately, and the central machine is again allowed access to the core store. However, another drum or tape request can arise before the stack request of the stored machine order occurs, in particular because this latter order may still be held up by the central machine. If this is the case the drum or tape is allowed immediate access and a further attempt is made to complete the stored machine order when this drum or tape stack request occurs.

If the stored machine order is for an operand, the content of the page digit register will correspond to the location of this operand. The next machine request for an instruction pair will then almost certainly result in a "wrong page" indication. This is prevented by arranging that the next instruction pair access does not by-pass the equivalence circuitry.

The effect on the machine speed when the drum or tapes are transferring information to or from the core store is dependent upon two factors. First, upon the proportion of time during which the buffer register in the core coordinator is busy dealing with machine requests, and second, upon the particular stacks being used by the central machine and the drum or tape. If the computer is obeying a program with instructions and operands on the fixed or subsidiary store, then the rate of obeying instructions is unaffected by drum or tape transfers. A drum or tape interrupt occurring when the B.A.R. is free prevents any machine address being accepted onto this buffer for 1.0  $\mu$  sec. However, if the B. A. R. is busy then the next machine request to the core store is delayed until 1.8  $\mu$  sec after the interrupt if different stacks are being used, or until 3.4  $\mu$  sec after the interrupt if the stacks are the same.

When the machine is obeying a program with instructions and operands on the core store, the slowing down during drum transfers can be by a factor of two if instructions, operands, and drum requests use the same stacks. It is also possible for the machine to be unaffected. The effect on a particular sequence of orders can be seen by considering the one discussed in Sec. 4 and illustrated in Fig. 6. In this sequence the instructions are on stacks 0 and 1 while the operands are on stacks 2 and 3. If the drum or tape is transferring alternately to stacks 0 and 1 then the effect of any interrupt within the 3.2  $\mu$  sec of an instruction pair is to increase this time by between 0.5 and 3.4  $\mu$  sec depending upon where the interrupt occurred. The average increase is 1.8  $\mu$  sec and for a tape transfer with interrupts every 88  $\mu$  sec the computer can obey instructions at 98 percent of the normal rate. During drum transfers the interrupts occur every 4  $\mu$  sec, which would suggest a slowing down to 60 per cent of normal. However, for any regular sequence of orders the requests to the core store by the machine and by the drum rapidly become synchronized with the result in this particular case that the machine can still operate at 80 percent of its normal speed.

## **APPENDIX 2 METHODS OF DIVISION OF THE MAIN CORE STORE**

The maximum frequency with which requests can be dealt by a single stack core store is governed by the cycle time of the store. If the store is divided into several stacks which can be cycled independently, then the limit imposed on the speed of the machine by the core store is reduced. The degree of division which is chosen is dependent upon the ratio of core store cycle time to other machine operations and also upon the cost of the multiple selection mechanisms required.

Considering a sequence of orders in which both the instruction and operand are in the core store, then for a single stack store the limit imposed on the operating speed by the store is two cycle times per order, i.e.,  $4 \mu \text{ sec}$  is Atlas. This is significantly larger than the limits imposed by other sections of the computer (Sec. 4). If the store is divided into two stacks and instructions and operands are separated, then the limit is reduced to  $2 \mu \text{ sec}$  which is still rather high. The provision of two stacks permits the addressing of the store to be arranged so that successive addresses are in alternate stacks. It is therefore possible by making requests to both stacks at the same time to read two instructions together, so reducing the number of access times to three per instruction pair. Unfortunately such an arrangement of the store means that operands are always on the same stacks as instruction pairs, and the limit imposed by the cycle time is still  $2 \mu \text{ sec}$  per order even if the two operand requests in the instruction pair are to different stacks and occur at the same time.

Division into any number of stacks with the addressing system working through each stack in turn cannot reduce the limit below  $2 \mu \text{ sec}$  since successive instructions normally occur in successive addresses and are therefore in the same stack. However, four stacks arranged in two pairs reduces the limit to  $1 \mu \text{ sec}$  as the operands can always be arranged to be on different stacks from the instruction pairs. In order to reduce the limit to  $0.5 \mu \text{ sec}$  it is necessary to have eight stacks arranged in two sets of four and to read four instructions at once, which would increase the complexity of the central machine.

The limit of  $1 \mu \text{ sec}$  is quite sufficient and further division with the stacks arranged in pairs only enables the limit to be more easily obtained by suitable location of the instructions and operands.

The location of instructions and operands within the core store is under the control of the drum transfer

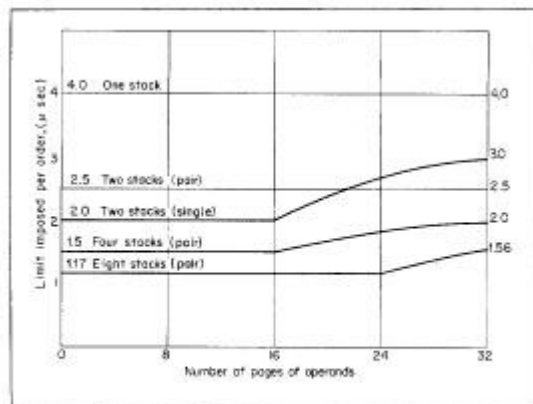


Fig. 8. Limit imposed by cycle time on operating speed for different divisions of the core store.

program, thus when

there are several stacks instructions and operands are separated wherever possible. Under these conditions it is possible to calculate the limit imposed on the operating speed by the cycle time for different divisions of the core store. The results are shown in Fig. 8; for stacks arranged in pairs instructions are read in pairs and in all cases both instructions and operands are assumed to be on the core store. Operands are assumed to be selected at random from the operand space; for instance, in the case of two stacks arranged as a pair, successive operand requests have equal probability of belonging to the same stack or to alternate stacks.

The limit imposed by a four stack store is never severe compared with other limitations; for example, the sequence of floating point addition orders discussed in Sec. 4 required  $1.6 \mu \text{ sec}$  per order with ideal distribution of instructions and operands. Division into eight stacks, although it reduces the limit, will not have an equivalent effect on the over-all operating speed, and such a division was not considered to be justified.

## References

Kilburn, Edwards, Lanigan, and Sumner [1962]; Brooker [1960]; Edwards, Lanigan, and Kilburn [1960]; Kilburn, Edwards, and Thomas [1956]; Kilburn, Edwards, and Aspinall [1960]; Kilburn and Grimsdale [1960]; Kilburn, Howarth, Payne, and Sumner [1961]; Lonsdale and Warburton [1956]; Papian [1957]; Fotheringham [1961]; Hartley [1968]; Howarth [1963]; Howarth, Jones, and Wyld [1962]; Howarth, Payne, and Sumner [1961]; Morris, Sumner, and Wyld [1967]; Sumner, Haley, and Chen [1962].

.