View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by NASA Technical Reports Server

N 70 41192

CR 113908

Technical Report 70-126 August 1970

NGR 21-002-206

CDL Description of a Memory Buffer Organization

by

Yaohan Chu

CASE FILE COPY

UNIVERSITY OF MARYLAND

COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND

Technical Report 70-126 August 1970
NGR 21-002-206

CDL Description of a Memory Buffer Organization

by

Yaohan Chu

This research was supported in part by Grant NGR 21-002-206 from the National Aeronautics and Space Administration to the Computer Science Center of the University of Maryland.

Abstract

In recent years, an important development in the organization of computer memories has been the use of a storage hierarchy on the nanosecond/microsecond level and more specifically of a very high-speed semiconductor memory as a buffer to the main memory of the computer system. This report describes a memory-buffer organization and its operation similar to that implemented in the IBM System/360 model 85. The Computer Design Language is employed to describe the details of the functional organization and of the sequential operation of the buffer in a concise and precise manner.

Table of Contents

CDL Description of a Memory Buffer Organization

by Yaohan Chu*

Storage hierarchy in the form of a relatively fast but relatively small main memory such as a magnetic-core memory and a relatively slow but relatively large mass storage such as a magnetic drum or disk storage has been employed ever since the large-scale digital computer system was first built. The basic idea behind such a storage hierarchy is to have the mass storage provide the necessary storage capacity and the memory give the desired processing speed. Such a storage hierarchy is at a microsecond/millisecond level.

An important development in memory organization during the last several years is to extend the above idea of storage hierarchy to a nanosecond/microsecond level. This idea in the embryonic form was implemented in a number of computers [1], [3], [4] by using registers or even a very small-capacity memory. It was proposed by Bloom, etc. [2] in 1962 and Lee [5] in 1963 as a "look-aside memory" and by Wilkes [7] in 1965 as a slave memory. It was first implemented as a memory buffer in the IBM System/360 model 85 computer [11] where the buffer is called the "cache" and is transparent to the programmer.

This report describes, by the Computer Design Language or CDL [14], a memory buffer organization and operation similar to that implemented in the IBM System/360 model 85.

---

*Computer Science Center, University of Maryland, College Park, Md. 20742.

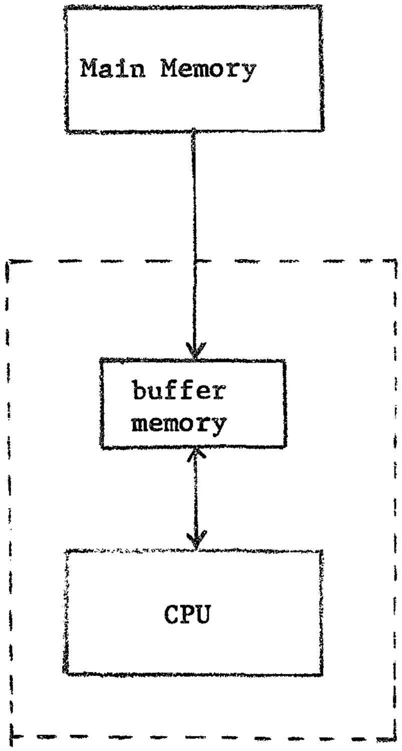


Fig. 1 Memory Buffering

page address	block address	word address
0 ... 9	10 ... 13	14 ... 15

(a) main memory address format

page address	block address	word address
0 ... 3	4 ... 7	8 ... 9

(b) buffer memory address format

Fig. 2 Memory address formats

1. Memory buffering

Conventionally, the main memory of a computer system is referenced by the CPU, one memory word at a time; the processing in the CPU is limited by the speed of the main memory. This limitation has become more critical as the capacity of the main memory becomes larger and larger and the speed faster and faster.

If a small-capacity memory which is one order of magnitude faster than the main memory is used as a buffer, as shown in Figure 1, the processing in the CPU could be greatly speeded up because the number of main memory references can be sharply reduced due to the following reasons:

(a) The transfer from the main memory to the buffer memory can be made a block (i.e., several words) at a time. If the main memory has a multiple-way interleaving, the block of words can be transferred in one main-memory cycle time.

(b) The block transfer may prefetch the desired words into the buffer memory and make them available to the CPU because there is a great probability that the other words of a referenced block would be soon needed.

(c) The words in the buffer memory may be used several times due to iterative loops and subroutines in a program, thus greatly reducing the need for memory references from the main memory.

2. Organization

2.1 Memories

In this description the main memory and the buffer memory are chosen with the characteristics shown in Table 1. The main memory has a cycle time of one microsecond, a data transfer width (i.e., word length) of 128 bits, and a capacity of 64K (where K represents a multiplier of 1,024) 128-bit words; it is four-way interleaved. The buffer memory has a cycle time of 80 nanoseconds, a data transfer width of 128 bits, and a capacity of 1,024 words. Both memories are divided into four-word blocks; thus, there are 16K blocks in the main memory and 256 blocks in the buffer memory. Every 16 contiguous blocks form a page; thus, there are 1K pages in the main memory and 16 pages in the buffer memory. Data transfer between the main memory and the buffer memory is one block at a time; data transfer between the buffer memory and the CPU is one word at a time. The main memory requires a 16-bit address, while the buffer memory a 10-bit address; their formats are shown in Figure 2. The main memory address consists of a 10-bit page address, a 4-bit block address, and a 2-bit word address. The buffer memory address format is identical except that the page address is 4-bit.

For the organization to be described here, it is assumed that the first page of the main memory does not exist; thus, page address 0 of the main memory should not occur.

2.2 Registers P, Q, and V

As mentioned, both the main memory and the buffer memory are divided into pages. During operation, 16 of the 1,023 pages of the main memory are stored in the 16 pages of the buffer memory. These 16 pages are tagged by their main-memory page addresses in an array of 16 page-address registers. This arrangement of page mapping and page-address tagging is illustrated in Figure 3.

Table 1 Characteristis of the main memory and the buffer memory

Characteristics	main memory	buffer memory
memory cycle time	1 microsecond	0.08 microseconds*
data transfer width	128 bits or 1 word	128 bits or 1 word
data units	(a) 128 bits per word (b) 4 words per block (c) 16 blocks per page	(a) 128 bits per word (b) 4 words per block (c) 16 blocks per page
memory capacity**	(a) 64K words (b) 16K blocks, or (c) 1K pages	(a) 1,024 words, (b) 256 blocks, or (c) 16 pages
interleaving	4-way	none
address register	16 bits	10 bits

*CPU cycle time is also 0.08 microsecond

**K represents a multiple of 1,024

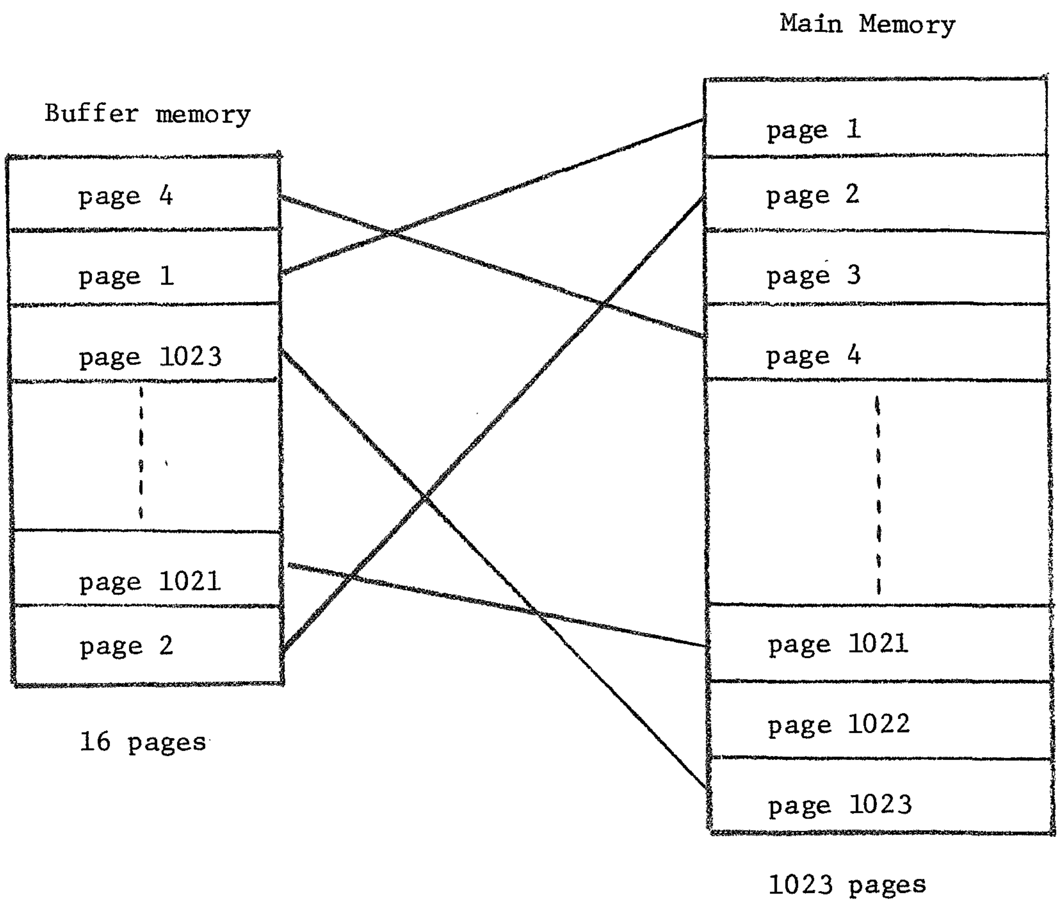


Fig. 3 Mapping between the pages in the main memory and those in the buffer memory

As also mentioned, each page in the main memory and buffer memory is divided into 16 blocks. The 16 blocks in a page of the buffer memory are illustrated in Figure 4 where each block is further divided into four words (not shown). Associated with each page of the buffer memory are, as also shown in Figure 4, a register which holds a 10-bit page address of the main memory, a 16-bit block validity register whose 16 bits store the status (1 means valid) of the 16 blocks of the page, a register which holds a 4-bit page address of the buffer memory. Since there are 16 pages in the buffer memory, there is an array of 16 page-address registers P, an array of 16 validity registers V, and an array of 16 page-address registers Q. Thus, one validity register, one main-memory page address register, and one buffer-memory page address register are associated with one page of the buffer memory.

As mentioned above, associated with each page of the buffer memory is a pair of registers P and Q. The P register stores the main-memory page address of the page in the buffer memory; the Q register stores the buffer-memory page address where this page in the buffer memory is stored. This is illustrated in the diagram in Figure 5. Note that the numbers shown in the buffer memory are main-memory page addresses; they should be the pages themselves addressed by these page addresses. Furthermore, these pages in the buffer memory may have partially been stored in the buffer memory as will be further described.

2.3 Activity list

The array of page-address registers P are made to perform three functions. The first function, as mentioned, is to store the page addresses of those 16 pages in the main memory that are (partially or completely) in

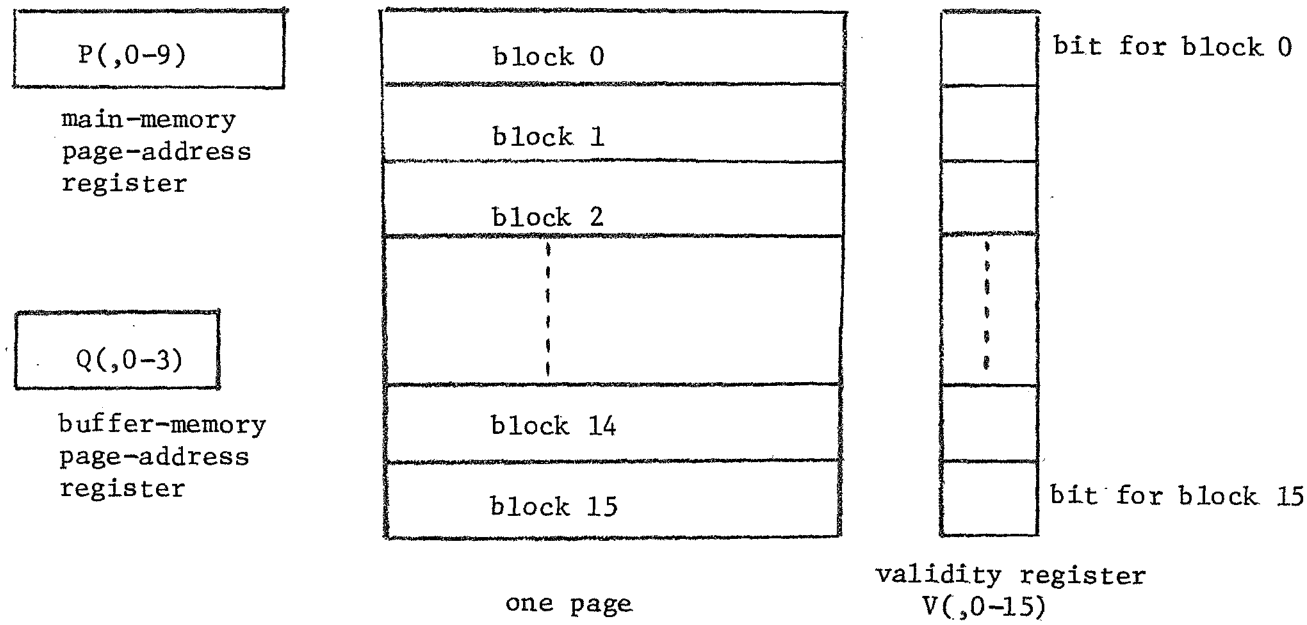


Fig. 4 Blocks in a page and the associated P, Q and V registers
(16 blocks in a page and 4 words in a block)

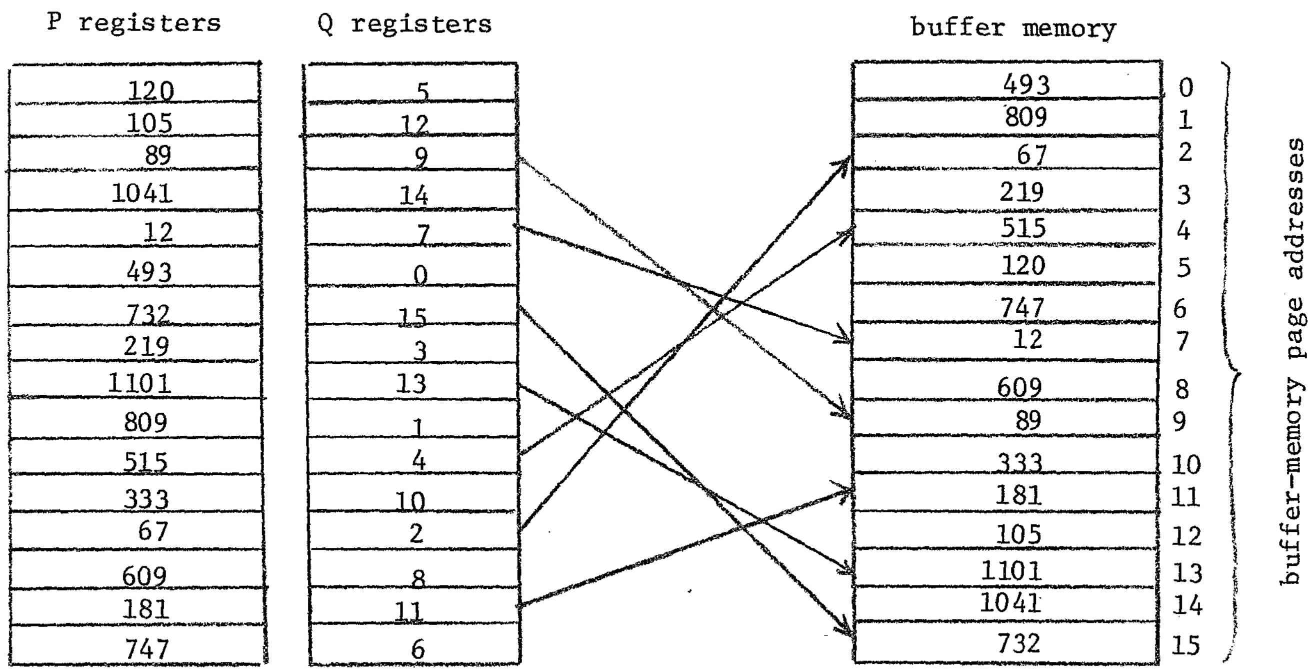


Fig. 5 Translation between main-memory page addresses and buffer-memory page addresses by array-registers P and Q.

the buffer memory. The second function is to make array P work as an associative memory so that, given a page address, simultaneous comparisons with the addresses in array P are made; those matched are indicated in the associated match register M. The third function is to store an activity list; the page address which is the most recent referenced by the CPU is placed at the top of the list, while the page address whose page in the buffer memory is next to be replaced is stored at the bottom of the list.

2.4 Configuration description

The above-described configuration is shown in the block diagram in Figure 6. Main memory MM is associated with address register MAR, buffer register MBR, and read and write control registers READ and WRITE. Buffer memory BM is associated with address register BAR, buffer register BBR, and read and write control register RB and WB. The effective address, the data word, and the read-write command all from the CPU are stored in registers S, DATA and RW, respectively. In addition, register C serves as a counter, and register B is used to control the buffer access sequence as will be further described. This configuration is now described by the following CDL statements.

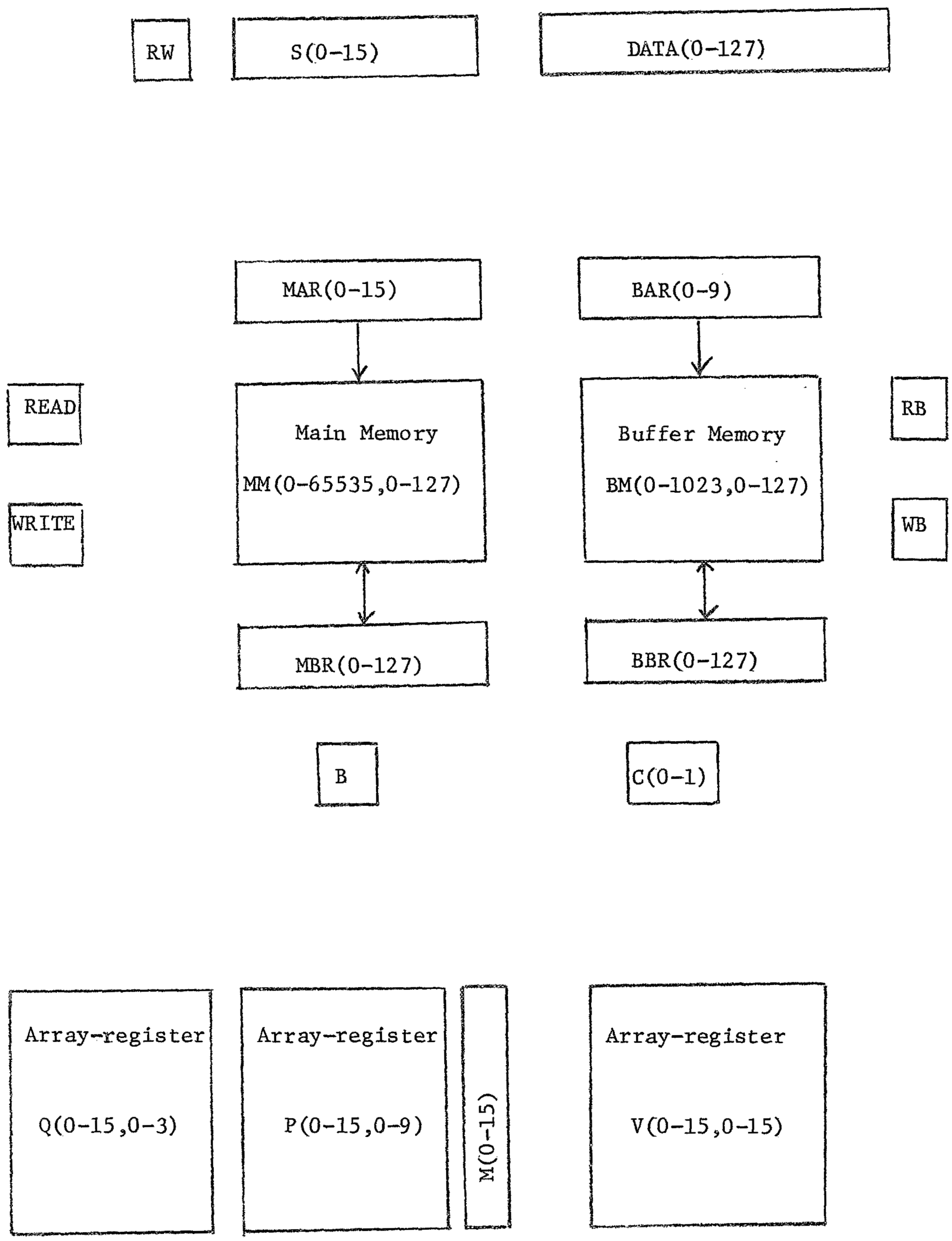


Fig. 6 A configuration of memory buffering

```
comment, configuration of buffer-memory access sequence          (1)

comment, buffer control registers

Array-register, P(0-15,0-9)     $main-memory page-address array-register

                Q(0-15,0-3),    $buffer-memory page-address array-register

                V(0-15,0-15),   $Block validity array-register

Register,       M(0-15),        $match register for P array-register

                C(0-1),         $counter

                B,              $buffer-access control register

Encoder,        N(0-3)=M,       $encode match register

comment, CPU registers

Register,       S(0-15),        $CPU effective address register

                DATA(0-127),    $CPU data register

                RW,             $CPU read-write command register

subregister,    S(PA, BA,WA)=S(0-9,10-13,14-15),

comment, main and buffer memory and their associated registers

Register,       MAR(0-15),      $main-memory address register

                MBR(0-127),     $main-memory buffer register

                READ,           $main-memory read command

                WRITE,          $main-memory write command
```

```
          BAR(0-9),        $buffer-memory address register

          BBR(0-127),      $buffer-memory buffer register

          RB,              $buffer-memory read command

          WB,              $buffer-memory write command

Memory,   MM(MAR)=MM(1-65535,0-127),

          BM(BAR)=BM(0-1023,0127),

Block, UPDATE(IF (M(1)=1) THEN (P(0-1,)<--cir P(0-1,),Q(0-1,)<--cir Q(0-1,),

                               V(0-1,)<--cir V(0-1,)),

             IF (M(2)=1) THEN (P(0-2,)<--cir P(0-2,),Q(0-2,)<--cir Q(0-2,),

                               V(0-2,)<--cir V(0-2,)),

             .................................................................

             IF (M(15)=1) THEN (P(0-15,)<--cir P(0-15,),Q(0-15,)<--cir Q(0-15,),

                               V(0-15,)<--cir V(0-15,)),

Operator,  J(0-15)<--K(0-15,) match L

Register,       L(0-9),

Array-register, K(0-15,0-9),

/begin/         J(0)<--(K(0,0)ΘL(0))*(K(0,1)ΘL(1))*.........*(K(0,9)ΘL(9)),

                J(1)<--(K(1,0)ΘL(0))*(K(1,1)ΘL(1))*.........*(K(1,9)ΘL(9)),

                J(15)<--(K(15,0)OL(0)*(K(15,1)ΘL(1))*.......*(K(15,9)ΘL(9)),

                end of operator
```

The above encoder encodes the contents of match register M into a buffer-memory page address. The above UPDATE micro-operations update the activity list as will be further described. The above operator match is defined in order to perform the match between the given main-memory page address and those 16 addresses in the P registers.

3. Buffer Access

3.1 Access sequence

The sequence for accessing a word from the buffer memory is described in the flow chart of Figure 7. When the CPU requests a memory reference, the effective address is transferred to the main memory. The array of registers P is then searched for the effective page address.

For a read operation, if the page is active, the page address is put on the top of the activity list; if the page is not active, remove the page address at the bottom of the activity list, put the new page address at the top, and reset the associated validity register to 0 to indicate that none of the 16 blocks of the page in the buffer memory has been loaded from the main memory. In either case, the validity bit associated with the effective block address is tested. If the validity bit is 1, the block is in the buffer memory; and the word is next read out of the buffer memory. If the validity bit is 0, the block (not the page) is next loaded from the main memory into the buffer memory and during the loading the first word from the main memory is also transferred to the CPU.

For a write operation, the word is always written into the main memory; this is known as "storage through". If the page is active, the word is also written into the buffer memory and the activity list is updated. The purpose of writing into both memories is due to the fact that the input-output channels also communicate with the main memory.

3.2 Sequence chart

The sequential operations in accessing the buffer memory are organized as a sequence, called the buffer-access sequence, which is controlled by register B. The buffer-access sequence is shown in the sequence chart of

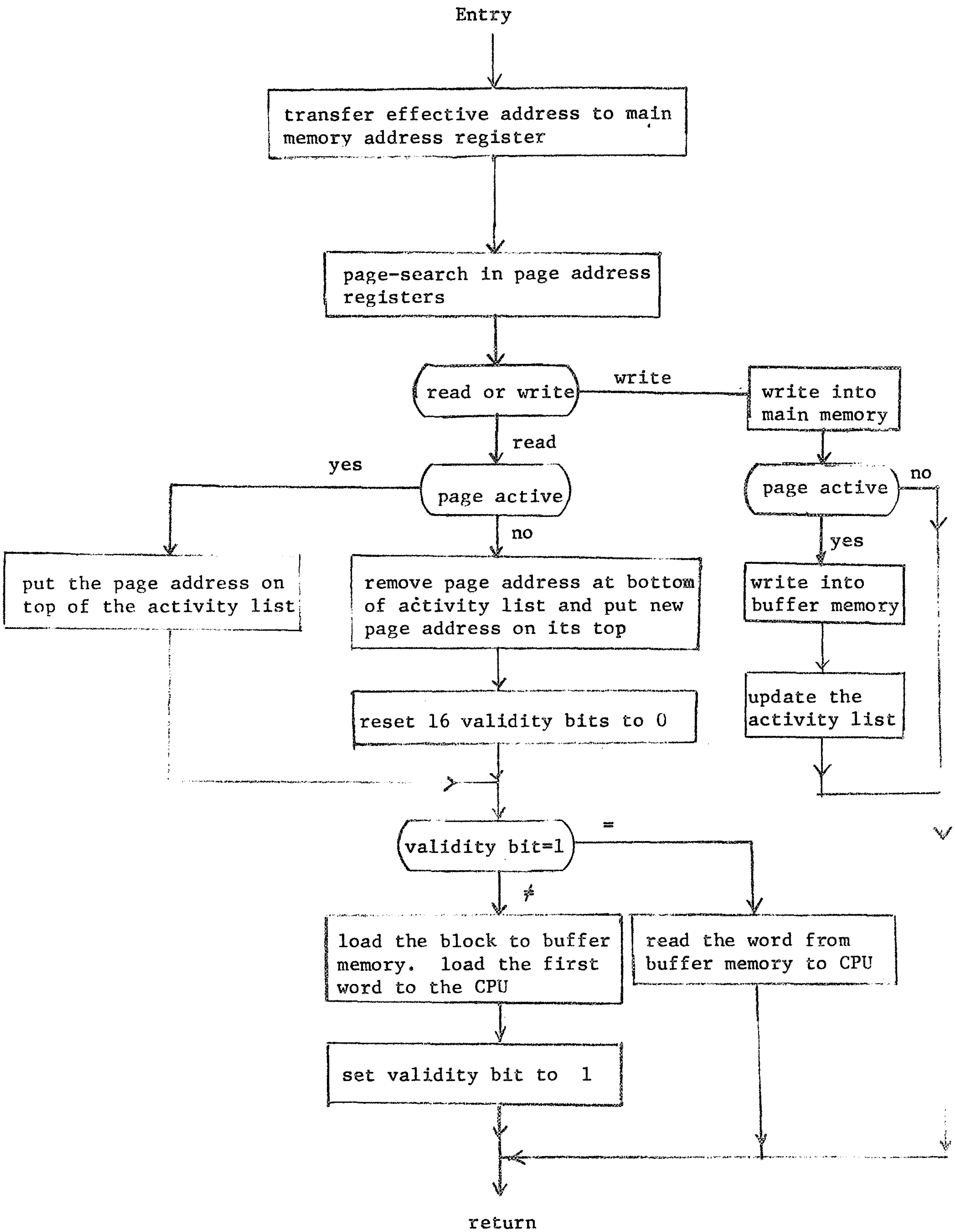


Fig. 7 Flow chart of read and write operations with memory buffering

Figure 8. It is assumed that the effective address, the data word if there is one, and the read-write command are initially placed by the CPU in registers S, DATA and RW, respectively. It has been assumed that main-memory page address 0 does not occur.

As shown in Figure 8 when register B is set to 1, the buffer-access sequence is activated. The effective address in register S is transferred to the main-memory address register MAR. The P registers are now searched for the effective page address in subregister S(PA); those matched are marked in match register M. At this point further operations depend on whether a read or a write operation is requested by the CPU.

If it is a write operation, the data word in the DATA register is stored into the main memory and, if match register M does not contain 0, the activity list is updated by placing the matched page address on the top of the list as will be further described, and the data word is also stored into the buffer memory. Registers B and M are next reset to 0. The sequence is now completed.

If it is a read operation, match register M is tested to determine whether the page is active. If the page is not active, both arrays of registers P and V are right-shifted and the array of registers Q is circularly right-shifted. The manner in which the P registers are shifted puts the effective page address on the top of the activity list and removes the page address at the bottom of the list; this is illustrated in the diagram of Figure 9(a). The manner in which the Q registers are shifted makes the address of the newly available buffer memory page attached to the effective page address now at the top of the activity list. The manner in which the V registers are shifted rests to 0 those block validity bits associated with

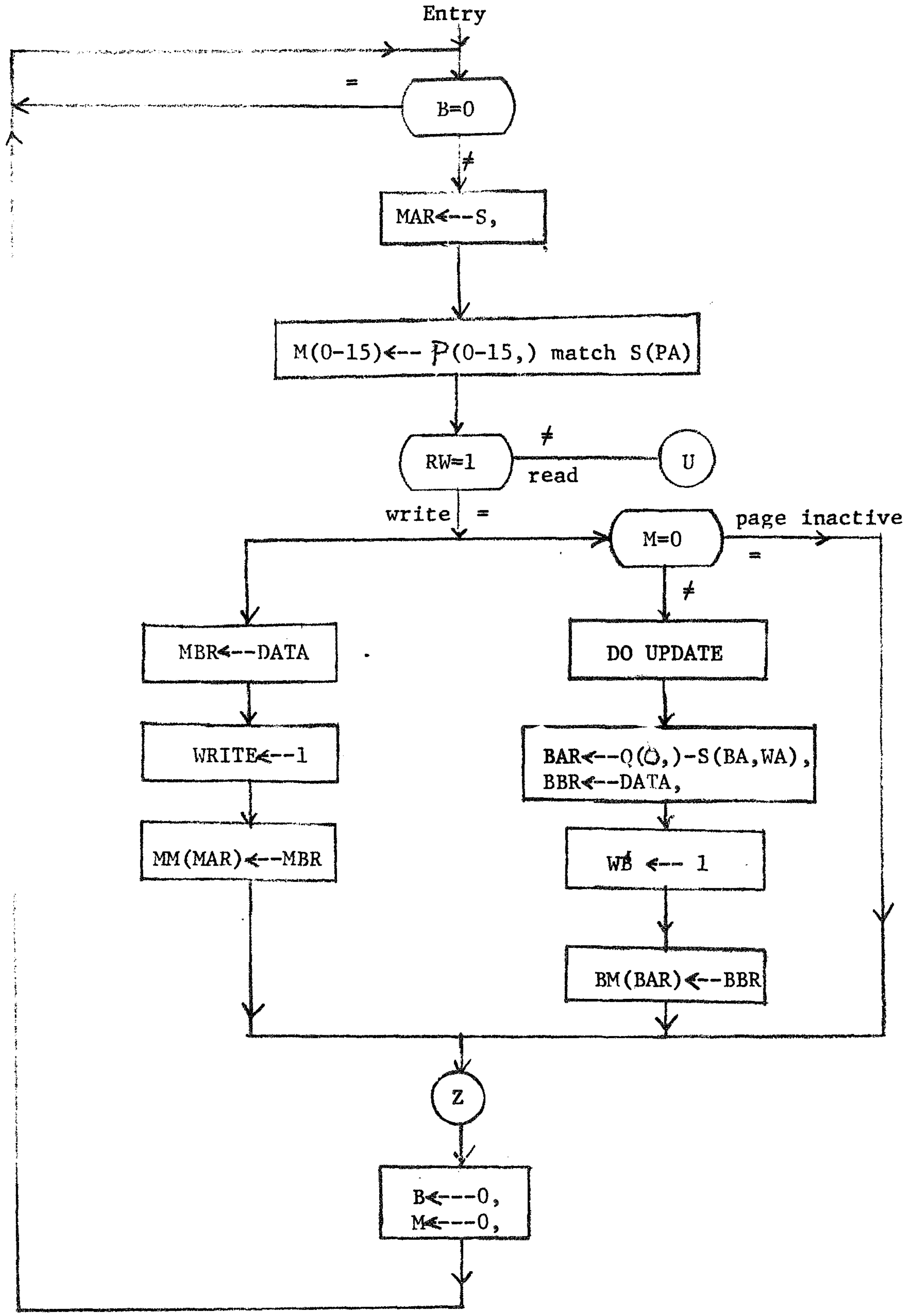


Fig. 8 Sequence chart for buffer-memory access sequence

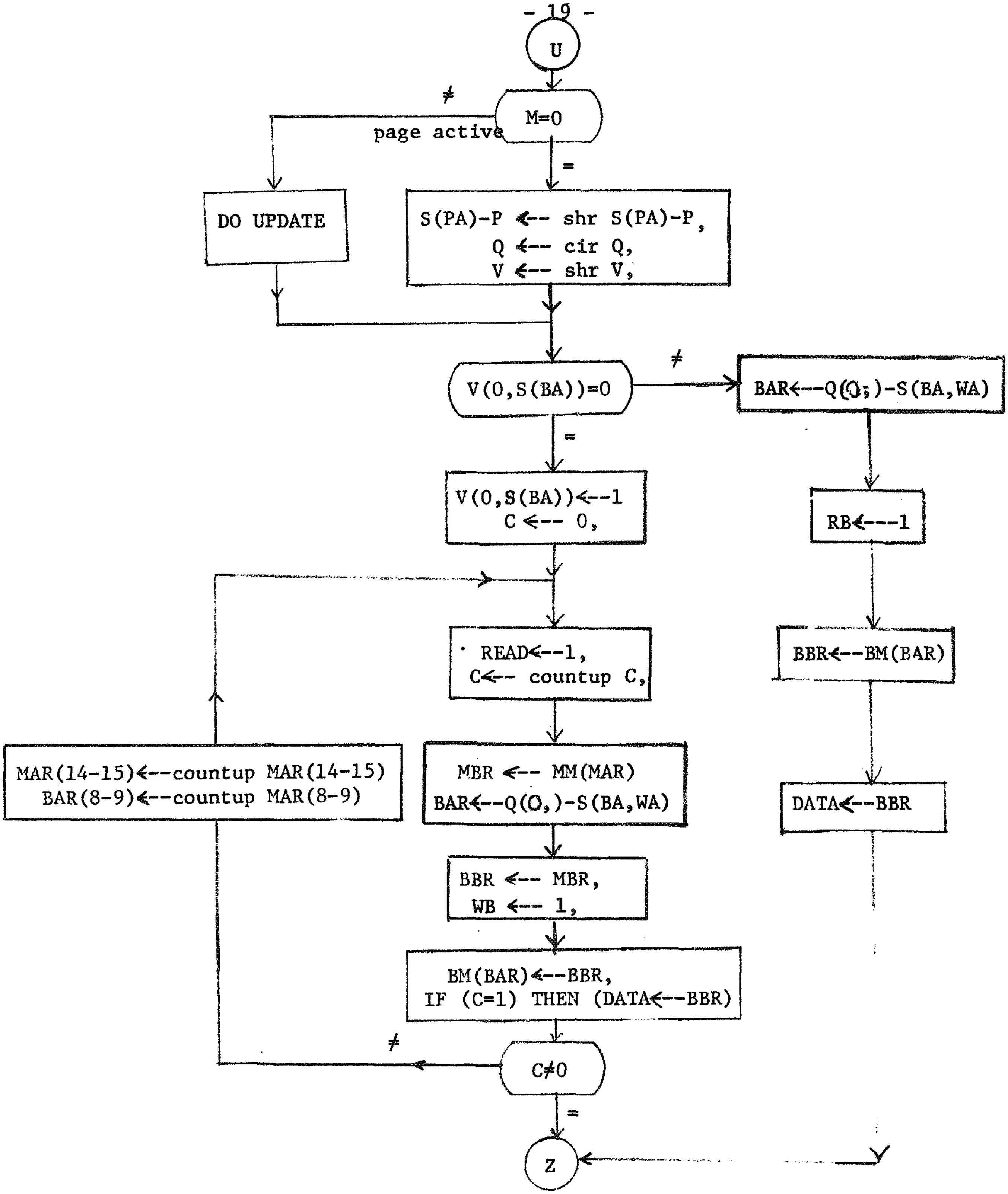


Fig. 8 (continued)

the effective page address. If the page is active, the UPDATE micro-operations as defined by the block statement in statements (1) are carried out. These micro-operations move the matched page address to the top of the activity list and move the intervening page addresses down one position; this is illustrated in the diagram of Figure 9(b). (These are also the micro-operations that are required to update the activity list during a write operation if register M does not contain 0.) While the page addresses in the P registers are being moved, the validity bits in registers V and the buffer-memory page addresses in registers Q are similarly moved. The manner of handling the activity list as illustrated in the diagrams of Figure 9 makes the least active page address drift down to the bottom of the list and eventually be displaced if that page address has been longest without being referenced.

After the activity list is updated as a result of the request being a read operation, the validity bit specified by the block address in subregister S(BA) is tested. Since the exact validity bit depends on the particular block address, this validity bit is addressed by the following symbolic subscript,

V(0,S(BA)).

The above manner of addressing this particular validity bit is equivalent to the description by the following 16 conditional micro-statements,

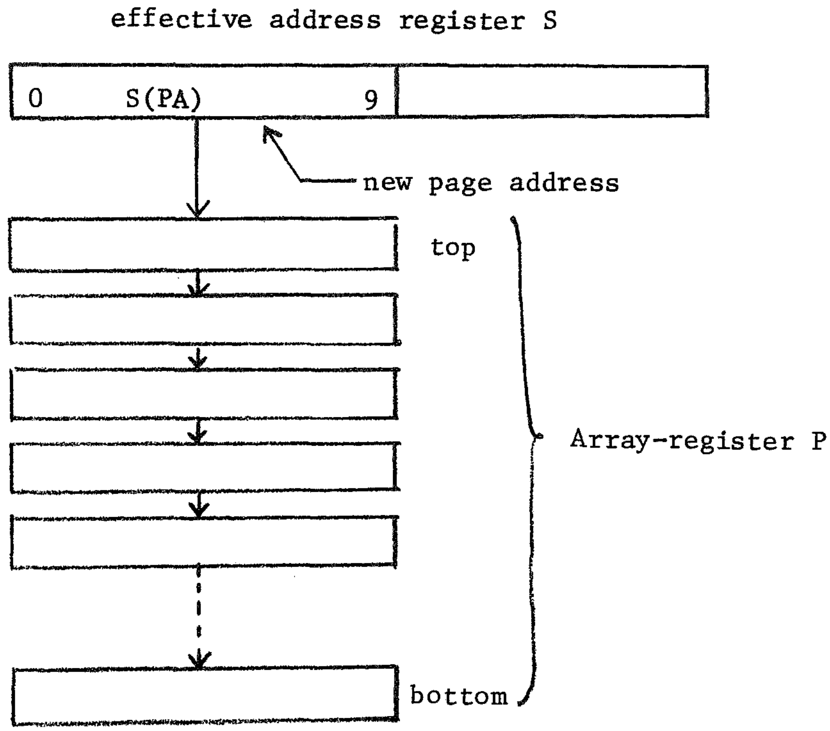
IF (S(BA)=0) THEN (V(0,0)................),

IF (S(BA)=1) THEN (V(0,1)................),

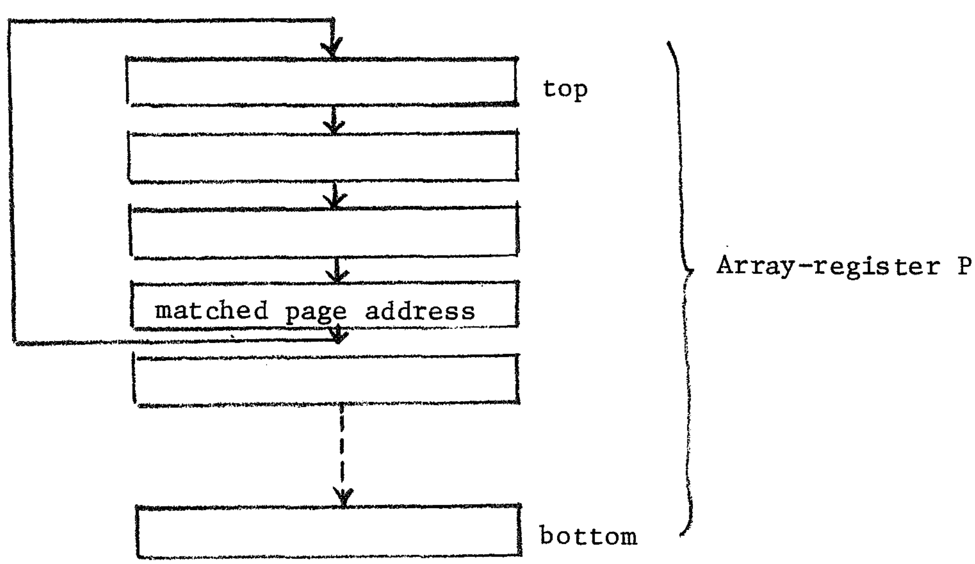
....

IF (S(BA)=15) THEN (V(0,15))............,

which are too lengthy to be desired. If the validity bit is 1, the block of



(a) put the new page address at the top of the activity list



(b) put the matched page address at the top of the activity list

Fig. 9 Two ways of updating the activity list words is in the buffer memory and the particular word in the buffer memory is read out into the DATA register. The buffer memory address is formed by using the contents of register Q(0) as the page address and the contents subregister S(BA,WA) as the block address and the word address. If the validity bit is 0, the block of words is not in the buffer memory; this block of words is now loaded into the memory and the validity bit is set to 1. During the loading counter C is used to count the number of words, and the first word from the main memory is also transferred to the CPU in order to reduce the access time. Registers B and M are next reset to 0. The sequence is now completed.

3.3 Sequence Description

The buffer-access sequence in Fig. 8 is now described by the following procedural statements.

```
comment,  buffer-memory access sequence begins here          (2)

/W/       IF (B=0) THEN (GOTO W);

comment,  transfer effective address to address registers

          MAR<--S;

comment,  page search in array-register P

          M(0-15)<--P(0-15,) match S(PA);

comment,  determine read or write

          IF (RW=1) THEN (GOTO Y);

comment,  micro-operations for a read operation

/U/       IF (M=0) THEN (S(PA)-P<--shr S(PA)-P, Q<--cir Q, V<--shr V)

                    ELSE (DO UPDATE);

comment,  test block validity bit

          IF (V(0,S(BA)=1) THEN (GOTO X)  ELSE (V(0,S(BA))<--1, C<--0);

comment,  load the block to the buffer memory

/R/       READ<--1, C<--countup C;
```

```
            MBR<--MM(MAR), BAR<--Q(O,)-S(BA,WA);

            BBR<--MBR, WB<--1;

            BM(BAR)<--BBR , IF (C=1) THEN (DATA<--BBR);

            IF (C=O) THEN (BAR(8-9)<--countup BAR(8-9),

                           MAR(14-15)<--countup MAR(14-15), GOTO R)

                    ELSE(GOTO Z);

comment, read the word form the buffer memory

/X/         BAR<--Q(O,)-S(BA,WA);

            RB<--1;

            BBR<--BM(BAR);

            DATA<--BBR, GOTO Z;

comment, micro-operations for a write operation

/Y/         IF (M≠O) THEN (DO UPDATE);

            MBR<--DATA, IF (M≠O) THEN (BBR<--DATA);

            WRITE<--1, IF (M≠O) THEN (WB<--1);

            MM(MAR)<--MBR, IF (M≠O) THEN (BM(BAR)<--BBR);

/Z/         B<--O, M<--O, GOTO W;

            END
```

4. References

1. Eckert, J. P., J. C. Chu, A. B. Tonik and W. F. Schmit, "Design of UNIVAC-LARC system:I", Proceedings of Eastern Joint Computer Conference, 1959, pp. 59-65.

2. Bloom, L., M. Cohen, and S. Porter, "Considerations in the design of a computer with high logic-to-memory speed ratio", Proceedings of Gigacycle Computing Systems, AIEE Special Publication S-136, 1962, pp. 53-63.

3. Takahashi, S., H. Nishino, K. Yoshihiro, and K. Fuchi, "System Design of the ETL Mk-6 computers", Information Processing 1962, Amsterdam, the Netherlands, North Holland Publishing Co., 1963, p. 690.

4. __________, "Ferranti Computing Systems, Atlas 2", Ferranti Ltd., London, 1963.

5. Lee, F. F., "Look-Aside Memory Implementation", Memorandum MAC-M-99, Massachusetts Institute of Technology, Cambridge, Mass., August 19, 1963.

6. Lee, F. F., "Look-Aside Memory Simulation", Memorandum MAC-M-131, Massachusetts Institute of Technology, Cambridge, Mass., January 2, 1964.

7. Wilkes, M. V., "Slave Memories and Dynamic Storage Allocation", IEEE Transactions on Electronic Computers, April 1965, pp. 270-271.

8. Scarrott, G. G., "The Efficient Use of Multilevel Storage", Proceedings of IFIPS Congress 1965, Spartan Books, 1965.

9. Belady, L. A., "A Study of Replacement Algorithms for a Virtual Storage Computer", IBM Systems Journal, Vol. 5, No. 2, 1966.

10. Gibson, D. H., "Considerations in Block-Oriented Systems Design", Proceedings of the Spring Joint Computer Conference, 1967, pp. 75-80.

11. Lipstay, J. S., "Structural Aspects of the System/360 Model 85, the Cache", IBM Systems Journal, Vol. 7, No. 1, 1968, pp. 15-21.

12. Sisson, S. S., M. J. Flynn, "Addressing patterns and memory handling algorithms", Proceedings of the Fall Joint Computer Conference, 1968, Thompson Book Company, pp. 957-967.

13. Lee, F. F., "Study of 'Look-Aside' Memory", IEEE Transactions on Computers, November, 1969, pp. 1062-1064.

14. Chu, Y., "Introduction to Computer Organization", Prentice-Hall, Inc. 1970.