Virtual Memory Applications Of The MU9C1480A LANCAM®

The availability of large content-addressable memories, with their unique associative search properties brings the possibility of improved search speeds in virtual memory applications. This application note outlines methods to improve database server and network response times using MUSIC Semiconductors' MU9C1480A 1K x 64 LANCAM as an example.

INTRODUCTION

Acceleration of database file retrieval is extremely important in today's world of heavily networked computer systems. Accelerated disk file retrieval or manipulation makes the file server more efficient, which improves network throughput, and the use of associative or content-addressable memory (CAM) can provide this improvement.

Improvements in disk access times can contribute greater network performance than increases in network transmission rate, frequently at lower cost. Since a large percentage of computer system usage is database oriented, network traffic is weighted toward communications between remotes and servers. Most of these communications will be remote requests for database records and related data in a relational database, rather than the transmission of complete files. In these cases, the disk file access time becomes a sizable proportion of the overall network response time as seen by the remotes. For example, a full screen of text data could be transmitted in about one millisecond over a 16 MHz network (ignoring overhead), while the time required for the disk access to retrieve the data is measured in tens of milliseconds. These factors make disk latency time a major component of network file retrieval request time. Disk latency includes the time to read the File Allocation table and then the time to retrieve the data, both of which are affected by disk access time. Additionally, if the file is fragmented, or if a record is to be found by "key field" searches, more disk accesses are required.

Disk access times can be improved by using external memory. One approach to improving access times has been to use large RAM caches (RAMDISK) to store frequently used disk files to eliminate disk accesses for files that have been stored in the cache. If enough virtual memory is installed, many database files with their additionally related files can be stored and accessed at RAM speeds providing significant system performance enhancements. An additional way to improve access time by milliseconds is to store the File Allocation table (FAT) in CAM, saving the initial disk access time. The availability of large CAMs from MUSIC Semiconductors makes rapid FAT caching a possibility by providing a large tag buffer space and nearly instantaneous translation of virtual addresses to actual addresses. Previous approaches to look up table tag buffers suffer from resorting time delays whenever files are replaced, which can be quite often, and access time delays due to the need for shadow memory areas to perform address translations. The primary file handles or codes can be stored in the MU9C1480A, and the disk executive control software can fetch likely related files to a CAM-based tag buffer as well. If the FAT were augmented or duplicated in CAM, whenever a file was requested, the CAM-based FAT would be examined first to determine if the file were resident in virtual memory, saving an unnecessary disk access. This process consumes only hundreds of nanoseconds rather than tens of milliseconds. If the file was not found in virtual memory, the starting physical sector of the file could be obtained directly from the associated CAM data, with the disk accessed directly after the CAM operation. This method is faster than search tree schemes due to the fully associative nature of a CAM, saving critical milliseconds and improving server performance.

The MUSIC Semiconductors' MU9C1480A is an ideal Fully Associative Tag Register for virtual memory mapping, particularly for mapping disk files to main memory. Its size, 1024 entries x 64 bits, allows the creation of large buffer areas for many files of arbitrary size and arbitrary location in main memory. Although the access time of the MU9C1480A is slower than a fast SRAM, for fast instruction or data caching of main memory the access time is very fast compared to disk directory or file search times. However, its large size and dual associativity makes the MU9C1480A effective in memory-to-memory cache, since it provides space for 1024 addresses of blocks of arbitrary size in memory for virtual storage, and facilitates replacement algorithms through its versatile instruction set.

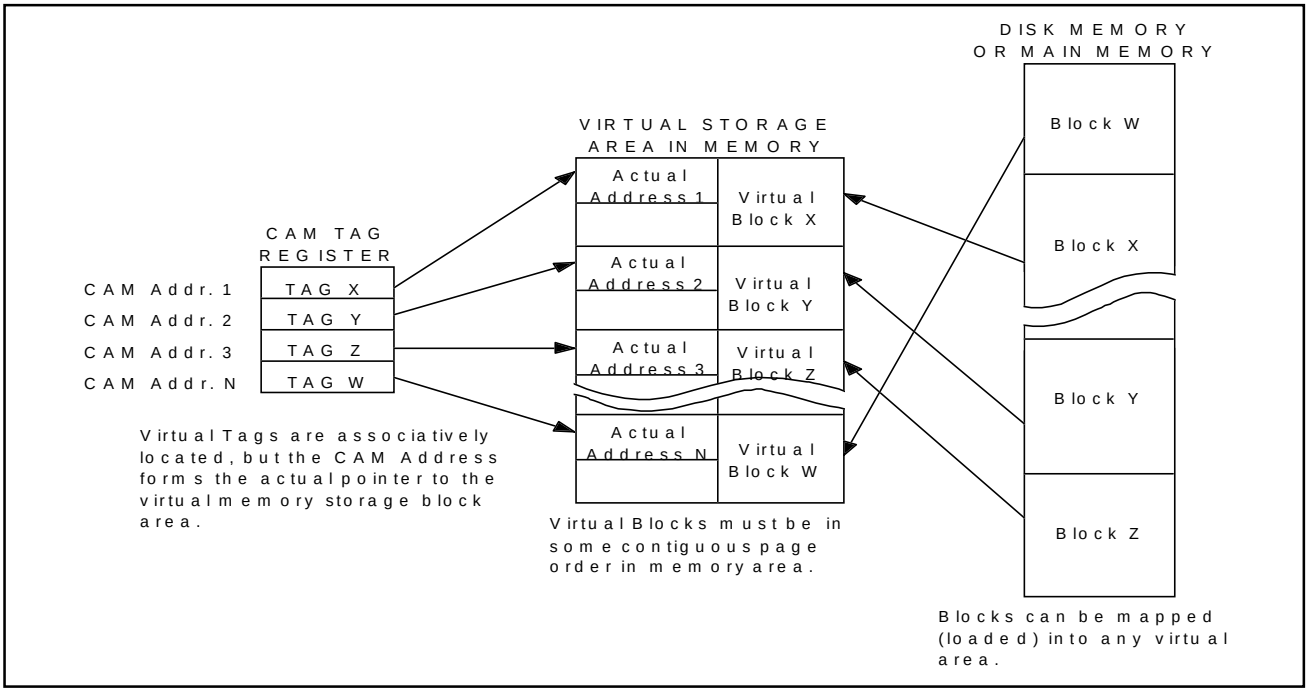

Figure 1: Virtual Memory Mapping, Type 1

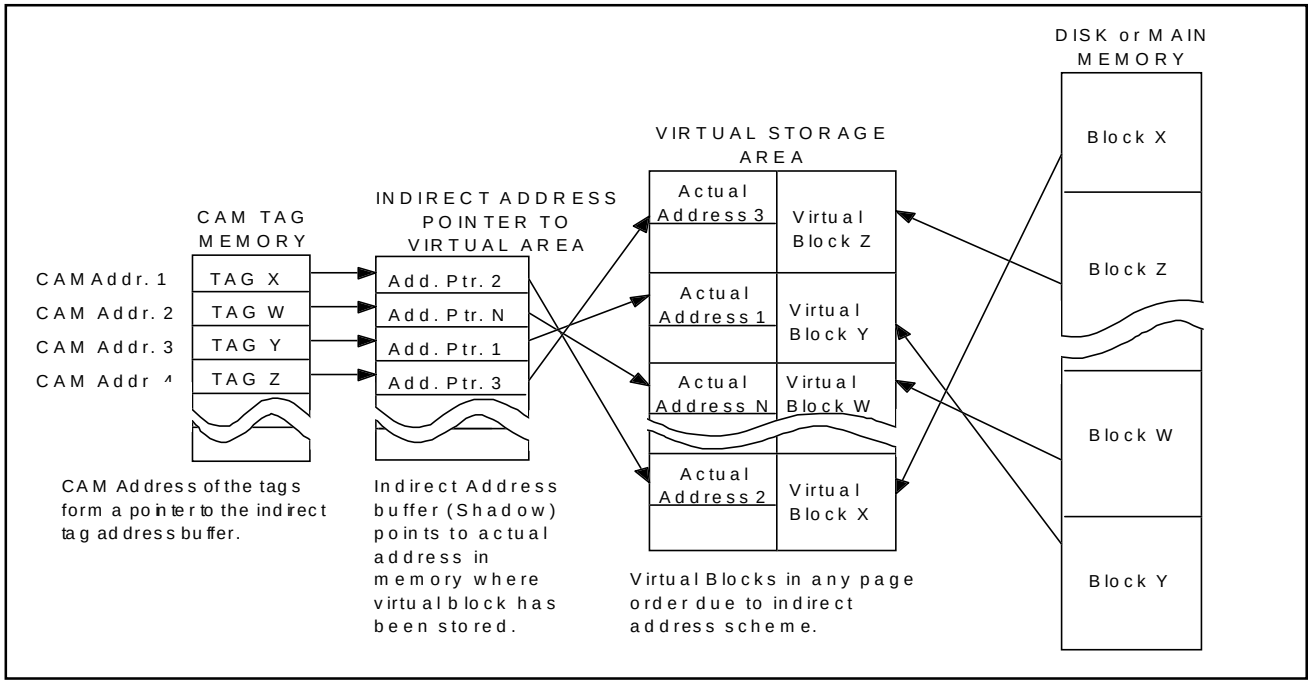

Figure 2: Virtual Memory Mapping, Type 2 (Indirect Pointer)

The MU9C1480A has a unique architectural feature particularly useful in virtual memory tag register applications in its adjustable CAM/RAM word partitioning, which creates a dual associativity. The actual address pointer to the cache or virtual memory area is stored in a RAM partition in the same memory location as the virtual tag, which is stored in the CAM partition. The actual address in the RAM partition can be accessed associatively after the virtual tag in the CAM partition is associatively accessed. The actual address pointer can then be output from the CAM in the succeeding CAM cycle and presented to the system address bus. This is a significant improvement over previous methods of pointing to the actual address, where the virtual tag must reside in a CAM location whose address is equal to the actual starting address of the virtual block in the cache area, or the method where the CAM address is used as an indirect pointer to a shadow RAM location which finally contains the actual start address of the virtual block.

VIRTUAL MEMORY MAPPING APPROACHES

Figure 1 depicts the traditional (Type 1) approach to CAM tag buffering. The blocks of memory from disk (for simplicity, we will use the term “disk” to mean mass storage or slow memory as opposed to virtual or cache memory) can be

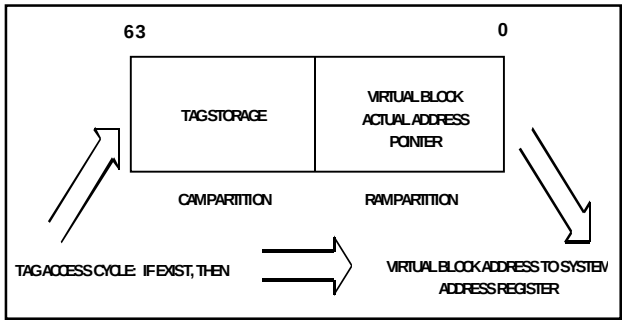

Figure 4: Contents of each Word in the

written to the virtual memory area in any order, but must normally be mapped into areas of known address boundaries to simplify access. Each block mapped into the virtual memory area is assigned an actual address and that address must be of a value that can be directly equated to an address in the tag buffer CAM by some offset, page or block value. The content of that address in the CAM is the tag or address of the disk file to be associatively located by the CAM. When the system calls for the file or address of a disk block, the address goes first to the CAM and if present, the system reads the address of the matched location in the CAM and uses it as an offset to an address of a memory block set aside for virtual or cache storage. That offset plus the base address becomes the actual

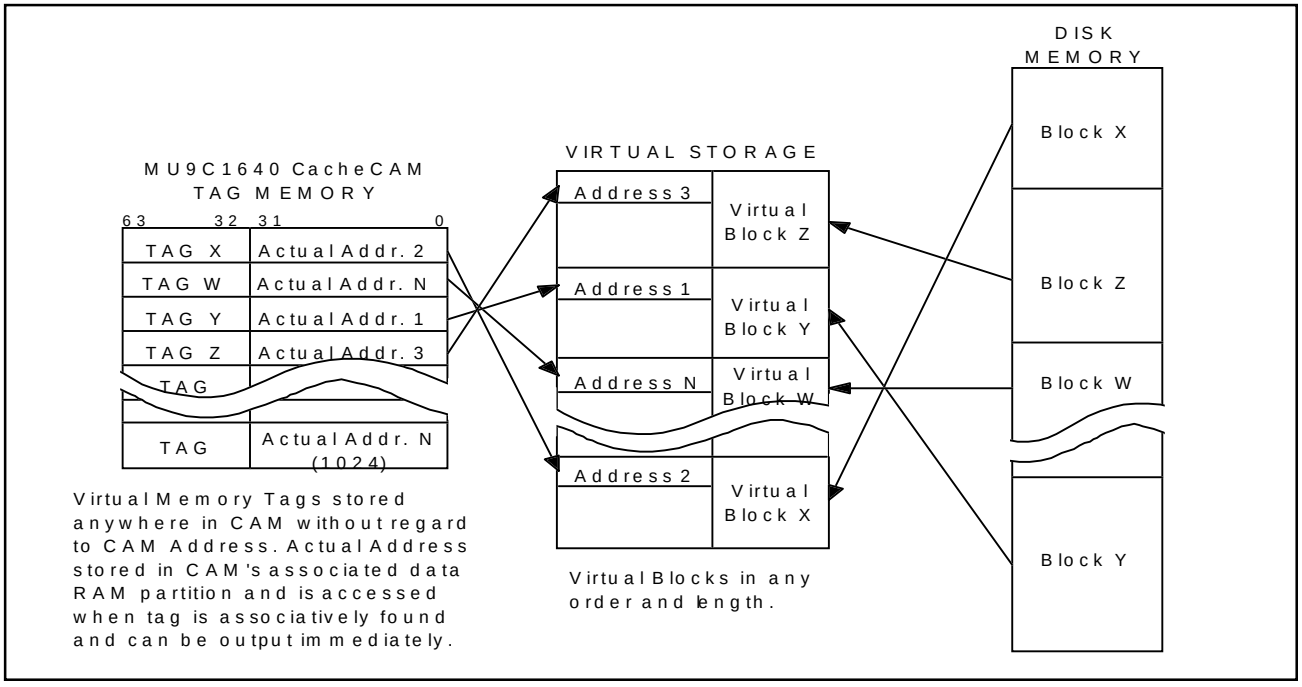

Figure 3: Virtual Memory Mapping with MU9C1480A Partitioning

address of the virtual file. In short, the CAM address becomes the pointer to the virtual file actual address. This approach is functional but suffers from the necessity of requiring the operating system virtual memory control firmware or hardware to always keep track of the addresses of contiguous blocks in the virtual memory area so that they can be related to the CAM tag word addresses. Virtual memory area storage blocks must be of regular size, since the CAM address is used as an offset to point to the starting locations. This approach makes replacement schemes convoluted, since a regular block size must be removed when it becomes dormant, and its replacement written in that exact location to maintain the relationship to the CAM address as its pointer. These direct relationship requirements slow down the access time of the virtual memory system, since there is considerable latency in all these memory accesses and translations, and if replacements occur often, the overall performance suffers.

An improved design is shown in Figure 2, the Type 2 Indirect Virtual-to-Actual mapping technique. The general approach is similar to that of Figure 1, but the indirect technique provides more flexibility in mapping virtual addresses to actual addresses. The difference can readily be seen in Figure 2 by the appearance of another storage area, the indirect pointer memory, between the CAM tag buffer and

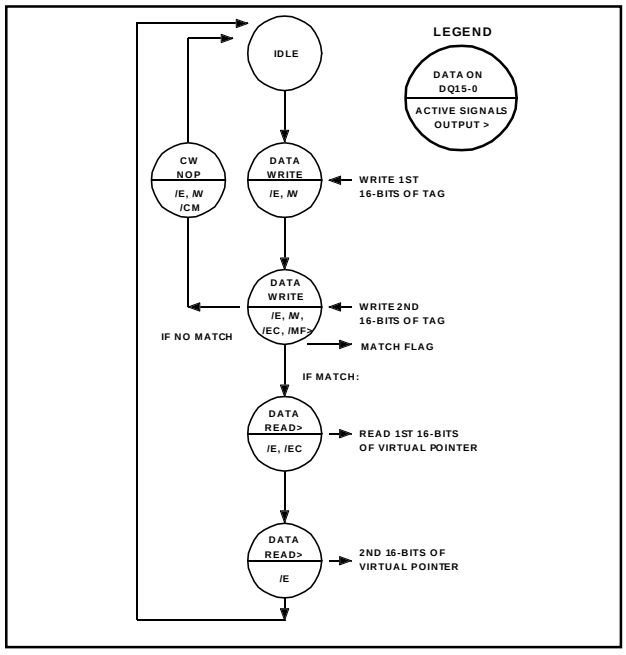

Figure 5: State Sequence for 32-bit Search for Tag and Fetch of Pointer

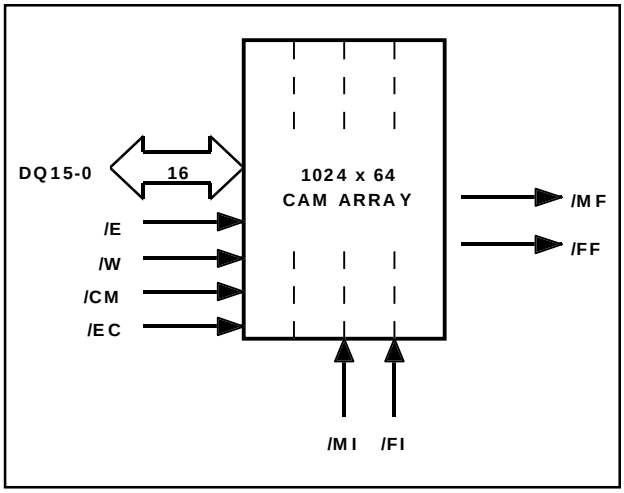

Figure 6: MU9C1480A Logic Symbol

the actual storage area of the virtual blocks. In this approach, the addresses of the virtual tags in the CAM are used as pointers to an indirect address storage area in which the final pointers are stored to the virtual blocks in the memory. With this method, virtual blocks can be stored in any order and length in the cache area, since the indirect pointers can be made to point to effectively anywhere in memory. Contiguous organization of virtual blocks is unnecessary, and if the indirect pointers are wide enough, the length of each virtual block can be arbitrary. Although this method is significantly more flexible than the Type 1 approach, it is still less than optimal due to the complex housekeeping of all the different memory areas involved.

Figure 3 illustrates the streamlined approach with the MUSIC Semiconductors' MU9C1480A LANCAM as the Tag Buffer and Indirect Pointer memory all in one IC. The operation is identical to that of Figure 2, but significantly enhanced by the incorporation of indirect address pointers within the CAM device itself, which are directly accessible when the virtual address is associatively located. Consequently, translation tables between CAM locations and indirect address pointers need not be maintained in any external memory, and no secondary memory access cycles are required to fetch pointers. Since the pointer to the actual memory address of the virtual block is contained in a RAM partition of the CAM word, it is immaterial where in the CAM any tag is stored, since the location of the tag in CAM is of no consequence to the operation. Storage of tags in the CAM is thus simplified, which also simplifies memory management schemes by reducing the complexity

of the hardware and/or software. The MU9C1480A LANCAM provides a hardware "Match Flag" pin to signal if a match is found, giving an immediate indicator whether a virtual file is present in the cache or not. A hardware "Full Flag" is also provided to facilitate scheduling of replacement algorithms. An internal Status register replicates these flags for reading by software routines for off-line or background maintenance.

The system designer has the choice of using the tag location address in the CAM to expand the relational storage of additional attributes, if needed by virtual memory applications or database management. Given the fully associative characteristics, performance benefits, and capacity of a virtual memory design using the MU9C1480A LANCAM, the system designer now has the opportunity for a more competitive and higher performance solution for virtual memory management and mass storage caching.

ASSOCIATIVITY FEATURES OF THE MU9C1480A

A more in-depth review of the secondary associativity feature of the MU9C1480A LANCAM and its implementation will be beneficial at this point to clarify just how the above storage and associativity of the tag and related indirect pointer are realized. The 64-bit word width in the MU9C1480A LANCAM can be partitioned into CAM/RAM on 16-bit boundaries. The CAM partition is accessed associatively with the input, as one would expect with a fully associative memory, and the RAM partition forms the secondary associated or related data field, which is then available for output on the next immediate device cycle. In the following example, the MU9C1480A LANCAM is globally partitioned as 32 bits of CAM with 32 bits of associated RAM. The virtual tags are stored in the CAM partition, and the actual address pointers to main memory of the virtual blocks (files) are stored in the RAM partition. The availability of 32 bits of address field for both tag and pointer fields provides another system enhancement by eliminating the need for hardware or software to do the address offset calculation. Figure 4 is a graphic representation of the contents of each word in the MU9C1480A LANCAM array when so partitioned.

The access and control state sequence for virtual tag field identification and fetch of the actual address pointer of the virtual storage block in main memory is illustrated by Figure 5. The operation is as follows:

1. The first 16 bits of the tag are written via the data bus, DQ (15-0), to the MU9C1480A's Comparand register, which is the default destination for Data Write cycles.

2. The second 16 bits are written to the Comparand register, using the internal Segment Control counter to guide the data to the CAM partition and begin the automatic Compare operation. If the matching 32-bit Tag is present, the hardware Match Flag pin goes LOW, and the Status register is set with the match condition and location.
3. If a successful match is indicated, a Data Read cycle can be performed to fetch the first 16 bits of the actual address pointer from the Highest-priority match location and output them to the DQ (15-0) data bus, again using the internal Segment Control counter to obtain the data from the RAM partition.

4. Another Data Read Cycle is performed to fetch the second 16-bit segment of the actual address pointer, completing the procedure.

Device Cycle Control Definitions

A Data Write Cycle is defined by taking the /W (write) pin LOW at the beginning of a cycle coincident with driving the /E (enable or clock) pin LOW. A Read cycle is performed by leaving the /W pin HIGH during /E.

To issue Command instructions to the device for initialization or to modify its operation, the /CM (command) pin is brought LOW coincident with /E, and an instruction selected from the extensive instruction set is written to the Instruction register via the DQ (15-0) bus.

Figure 6 shows the logic symbol for the MU9C1480A, demonstrating the simplicity of mapping the device into a system architecture with its 16-bit data bus, four-wire control bus with straightforward timing requirements, match flag output, full flag output, and match-in and full-in pins for cascading.

REPLACEMENT ALGORITHM CONSIDERATIONS

Most virtual memory caching architectures must take into consideration a means of replacing tags when they have become dormant, or when the CAM is full. There are several approaches to these replacement algorithms, the most common of which are summarized below. All these methods have their advantages and disadvantages, but they have different effects on the efficiency, or "hit rate," of the caching scheme depending upon the size and flexibility of the CAM tag memory. These considerations are significantly different for the MU9C1480A LANCAM versus previously available tag buffer memories.

1. Least Recently Used (LRU)

In an LRU scheme, each virtual tag is flagged with some means of identifying the chronological order in which the tags have been accessed. When the CAM is full, or after some arbitrary interval, the least recently used tag(s) are purged. One effective method is to create a linked list where each tag has some means to point to the address of the tag which was previously addressed, creating an ordered chain of next-most-recently-accessed tags.

At first glance, the LRU linked list scheme appears most efficient, since only the least recently used tag will be purged and replaced, because it is at the end of the address chain. However, it requires additional control and temporary storage external to the tag buffer to implement, since each address must be read as the tag contents are read, and then be appended to the next accessed tag in order to form the linked list. The MU9C1480A LANCAM facilitates this approach, since the match address is readily available to the controlling system via the device Status register, and the Associated Data RAM partition can be used to store the previously accessed tag address. There are two additional considerations for this method. First, external intervention is required to read the match address and store it in readiness for the next tag match, taking extra foreground processing time. Secondly, it uses up ten of the Associated Data field bits to store the next address, which are shared with the virtual block actual address pointers. The CAM approach is an improvement, however, over other tag buffer designs, which typically have no associated data field in them and require an even more elaborate indirect pointer system to associated data memory.

Another LRU approach is to merely flag each location as it is accessed with an ever-increasing (or decreasing) number, indicating the order of access. When the CAM is full, the lowest (or highest) numbered locations are purged. Assigning an incrementing number value to each tag access operation is only a little more efficient, since it still requires another operation after each tag access. Also, there is a limit to the maximum number that can be assigned, since a limited number of bits are available to assign to the LRU flag. One approach to this problem is to clear all the LRU value flags and start numbering over again. Another is to reduce each assigned number by a fixed offset after purging an identical amount of the oldest locations. Both of these procedures require repetitive operations to examine each tag location for its counter value and reset it, which reduce the efficiency of these approaches.

2. Least Frequently Used (LFU)

Another replacement approach is the LFU method, where each tag is flagged with a time interval value which increments periodically. The tags then have "time stamps" related to when they were last accessed. Several tags may have been accessed during the same interval, of course. Purging takes place on the oldest time stamps, based upon the fact that those tags least frequently accessed will have the oldest time stamps and are therefore statistically safe to purge and replace. The associative nature of the CAM allows a one cycle purging of all entries with the same time stamp.

The LFU approach purges not just the least recently used tag as in an LRU approach, but rather all the oldest tags within some time period. Previous caching schemes with small tag buffers preferred the LRU approach because of that distinction, but with today's larger tag buffers and the degree of multi-tasking, file access, and subroutine execution, LFU techniques have become more commonplace.

3. Random Replacement

The third major approach is a random replacement scheme where tags are arbitrarily purged when the CAM is full, using some pseudo random number generation algorithm to target randomly selected CAM addresses for replacement.

The random replacement method is the simplest to implement, and causes very little overhead in hardware and software, since no markers are used at all. Tags and

blocks are randomly purged on the assumption that once purged, they can be reinstalled if called again. The more random the purge and the larger the tag buffer, the lower the probability of deleting a frequently used tag. Obviously, a larger tag buffer minimizes the "miss" probability, so that system performance is maximized. "Hit" or "miss" performance is not deterministic, however. Having no additional foreground hardware or software overhead to conduct a more complex purge compensates somewhat for the difference. Random replacement may often be the fastest overall purging method.

SUMMARY

In light of the above considerations, an LFU replacement scheme might be most effective overall. Because the MU9C1480A has 1024 entries, even after many tags have been stored, a considerable length of time is likely to have passed, and the probability is quite high that the Least Frequently Used tag is also the Least Recently Used tag. Also, all the necessary LFU markers can be easily contained within the CAM itself with no appreciable performance degradation. The functionality of the CAM lends itself to rapid purging of LFU tags in the background, and also to expedient flagging of the locations as they are "hit," or accessed, with the least execution time penalty. Only a few Associated Data partition bits need be used, since a fairly long time interval can be used and one to eight bits of time stamp should suffice. Using eight bits of the RAM partition for LFU time stamping would still leave 24 bits for the actual address pointer field of the virtual blocks, which allows 16 million words of virtual storage.

TIME STAMPING AND PURGING

The methodology of time stamping takes advantage of a few features of the device itself. Due to the flexibility of the Segment Control register within the MU9C1480A, the upper 32 bits of the 64-bit Comparand register can be written with the desired search tag, leaving the lower 32 bits untouched as an additional storage resource. Periodically, when the processor-maintained time base changes, the new value of the time stamp can be written into the lower portion of the Comparand register, remaining there until overwritten by another time stamp value.

If the search for the desired tag is successful, the state diagram of Figure 5 on page 4 is modified by adding a MOV HM,CR[MR1] instruction to the state loop between the second Data Write cycle and the first Data Read cycle. In this instruction, Mask Register 1 masks all bits except the time stamp bits, so the previously stored tag and its associated virtual address pointer are not altered by the move operation. Because the time stamp modification is infrequent, it can be considered a background operation and of little consequence to system latency time. This LFU approach satisfies the need to minimize access cycles during foreground virtual access operations, since it takes only one CAM cycle to update the time stamp, with no external memory references or specialized hardware, and utilizes the CAM functionality to locate and purge the oldest entries. When purging is necessary, an explicit compare operation on only the time stamp bits is executed using the same mask, so that only the locations with the oldest time stamp match and can be purged with a VBC ALM,E instruction, which changes the validity bits for all matching locations to "Empty."

RE-ALLOCATION OF MEMORY

An additional replacement issue involves reallocating the actual memory space in the virtual memory allocation map to permit the installation of new tags after purging the old. Obviously, when a tag is purged from the CAM, the actual memory space occupied by the virtual routine is now available, and can be reallocated by the virtual memory driver, which needs to know which locations have become free. There are several ways in which this can be done, depending upon system constraints and requirements. It is assumed that the virtual memory driver has kept track of just where in main memory each virtual routine had been stored when it was added to the virtual area, and that each file stored has a length marker, offset, or end address contained within itself. The operating system would store within its allocation table the length or stop address of each virtual file when it is first tagged and stored in main memory. This information can be used during the purge routine to reallocate, or flag as "available," the main memory space occupied by a virtual file whose tag has been purged.

One re-allocation method is to modify the above purge routine so that when the explicit compare operation is done to locate virtual tags with the oldest LFU tag, the associated data RAM field (the actual memory address of the virtual routine) of the Highest-priority match is read from the MU9C1480A LANCAM, and that actual address is directly passed to the operating system memory allocation driver and used to flag the starting location of the memory space to be reallocated. The allocation map should already know the size of the virtual file, so the correct amount of memory can be reallocated directly. The purge procedure then includes a VBC HM,E instruction to empty the first Highest-priority matching tag location after reading its associated data field for the actual address pointer, followed by a CMP V instruction which will find the next matching tag location, and so on. The purge time can be improved if the virtual addresses to be purged and reallocated are read from the CAM into a file by the system, and then acted upon for re-allocation after the CAM purge is completed. With this approach, the time that the purge routine accesses the CAM is minimized, and the actual main memory re-allocation can be done as a background operation by the system. A second approach for determining which areas in virtual memory are to be reallocated requires fewer foreground CAM cycles, but requires more system background execution time. If the CAM purge routine follows the steps as given above, and just purges all the oldest entries at once using a VBC ALM,E instruction, the number of CAM access cycles is minimized. In order to determine which tags were purged and which actual memory locations can be reallocated, the system returns to the CAM in a background operation and performs a read of all valid locations to learn which virtual tags and their actual addresses still exist in the CAM. This information can be read into a system file and compared off-line with the previous memory allocation map maintained by the system. The virtual addresses which exist in the system allocation map but not in the CAM array, have been purged, and can therefore have their memory space reassigned. This approach is the reverse of the previous approach, and has the constraint that the system needs to return to the CAM to read the existing tags and reallocate space before the available virtual storage area of main memory is full.

LEARNING

To facilitate learning and storage of new tags, the MU9C1480A has a powerful "Move Comparand to the Next Free Address, set Valid" (MOV NF,CR,V) instruction. When a file handle or subroutine address is presented to the Comparand register and a "no match" occurs, the system memory management supervisor can issue a MOV NF,CR,V instruction to the CAM, and the non-matching tag in the Comparand register will be copied to the Next Free location in the MU9C1480A, which is the lowest numbered empty location. An internal priority encoder automatically keeps track of the Next Free location, so the addition of new tags is straightforward and requires very little overhead. Before issuing the MOV NF,CR,V command to the device, the system must determine where it wants to put the virtual block in actual memory, so that the actual address pointer can be moved to the lower portion of the Next Free location along with the tag in the upper portion. The internal Segment Control register can be used to load the actual address pointer into the lower portion of the Comparand register, while the tag occupies the upper portion.

DISK CACHING USING CAMs

Based on the foregoing discussion, it is necessary to consider some additional circumstances unique to disk cache virtual memory designs, particularly if full file names are used as the tag pointing to the virtual file area. Whereas main memory location tags may find 32 bits of tag sufficient, caching based upon disk file names may require a significantly wider tag word, because file names may be 11 or more characters long, each character comprising seven or eight bits. One approach is to use the power of the CAM associative look up to locate the long word by using multiple CAM locations to store the full file name, and execute more than one compare cycle to find each part of the file name which is stored in more than one location using additional bits to identify partial words. The basic function is the same as described earlier, but with the added complexity of storing sections of each file name in separate CAM locations along with appended flag bits to identify matching partial words.

Another approach uses two CAMs in parallel, each with a part of the file name stored in it, and searches in parallel to find the virtual file tag. In this approach, the file name sections are stored in identical locations in each CAM, so that similar file names having a partial match on a portion of the name do not result in erroneous match detection. The address of the matching location in each CAM is read to assure detection of the correct complete file name. If a match occurs at different locations in each CAM, it is the same as a "no match" condition; the file is not in virtual memory, because the match was only partial. This condition could occur if different file names are stored with identical upper or lower sections to the desired one, but not both upper and lower. If the desired file name also existed in the CAMs, multiple matches might occur in one or both CAMs. In that case, only the matching locations which have identical addresses in both CAMs are the correct match. The MU9C1480A provides a convenient method to process multiple matches for location of the correct match.

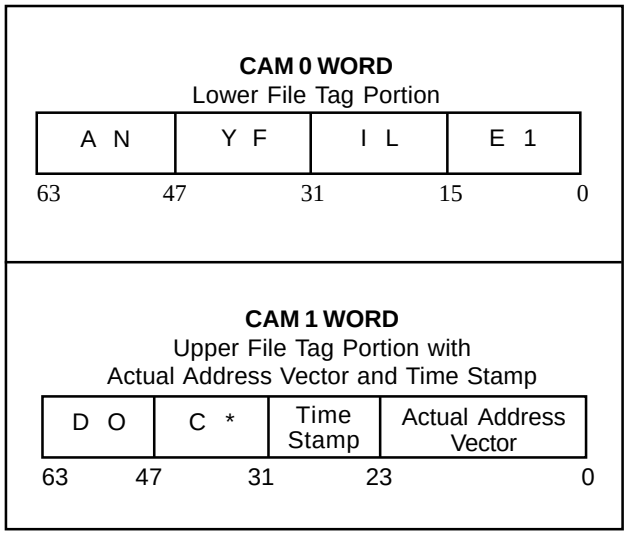

Figure 7: File Name Example

PARALLEL CAM EXAMPLE

The operational sequence would be as follows, assuming an 11 character file name. Divide the file names, as they are loaded into the CAMs, into upper and lower sections. In the example shown in Figure 7, an eight character lower section is loaded into CAM 0, and a four character upper section is loaded into CAM 1. Segments 0 and 1 of CAM 1 will be loaded with the actual address pointer to the virtual file location in main memory. 32 bits are set as an associated data RAM partition, with 24 bits for the actual address value and the eight remaining bits reserved for time stamps as described earlier. Names shorter than 12 characters can be padded out with some illegal value.

Although the method uses a parallel arrangement of CAMs, they can be loaded and read sequentially, depending upon the system bus width. An external means of control must be provided to synchronize the data bus inputs to the CAMs via separate control of their /E input enable pins, to sequence the upper and lower portions of the file name and the vector in the associated data partition of CAM 1. The external control must also read the match flag from each CAM and then examine the Status registers of each CAM in turn to determine if the match addresses are equal. The external control logic will also need a 10-bit comparator to detect match address equivalence between the two CAMs. If the match addresses are equal, the file resides in virtual memory, and the associated data is read from segments 0 and 1 of CAM 1 to fetch the actual starting address of the virtual file.

If the match addresses are unequal, it is a "no-match," and the tag is not in virtual memory. If multiple matches occur (detected by reading the MM bit in the CAMs Status registers), the multiple matches are interrogated sequentially until a match location equal in both CAMs is found. This might occur if file names such as "ANYFILE1.DOC," and "ANYFILE1.WK1" are stored, as in the example of Figure 7. For simplicity, the upper portion of a file name is shown stored in CAM 1, but this is likely to result in numerous "alias" multiple matches. "ANYFILE1.DOC" and "ANYFILE1.WK1" will have a multiple match in CAM 0 but only one match in CAM 1. The Status register of CAM 1 is read to fetch the match address and retained externally to the CAM. The Status register of CAM 0 is then read to determine the first matching address of the multiple match. If equal to the address in CAM 1, the match is valid, and the actual address vector can be read from CAM 1 and the file executed. If not equal, the match location in CAM 0 is set to "skip" by a VBC HM,S instruction, and a CMP V instruction is executed in CAM 0. This compare will find the next matching location. The Status register is again read, and if equal to that in CAM 1, the match is valid. After reading the associated data to fetch the actual vector, the skipped location(s) must be set back to "valid" to enable future comparisons. To do

that, a CMP S instruction is now executed to compare only those locations set to "skip" with the contents of the Comparand register. Next a VBC ALM,V instruction is executed to set all the matching skipped locations back to "valid."

ADDITIONAL CONSIDERATIONS

Multiple matches due to common file name extensions can be minimized if short file names are right justified rather than left justified as shown in Figure 7. The padding would be in the high order CAM 0 bits in that case. The associated data vector could be in CAM 0, so more primary file name characters are in the same CAM as the extension, minimizing multiple matches. The best method to minimize the probability of multiple matches on "alias" names is to alternate characters between the two CAMs, rather than merely splitting the file name at some point in length.

The primary caution for this methodology is that both CAMs must be initially set to the same number of empty and RAM-only locations, if used, to maintain synchronicity between the CAMs, so that when new file name tags are added, they will load to the same location in each CAM when the "Move to Next Free Address" instruction is used for learning a new tag.

The procedure for finding the aged time stamp will be the same as the given above for CAM 1; but once CAM 1 is purged, the empty locations must be found and the equivalent addresses in CAM 0 set to empty to maintain the identical available locations. This is accomplished by executing a CMP E instruction in CAM 1 after the purge cycle, and locating the empty addresses by reading the Status register after the match, then setting the first empty location found to "skip" as in the previous example of multiple match processing, and repeating the CMP E, Status Read, and VBC HM,S procedure, until all matching empty locations are found. The CMP E instruction compares only the empty validity bits, regardless of memory content, so no masking need be done to find empty locations.

CONCLUSION

The major features of the MU9C1480A LANCAM in virtual memory tag buffer are its word width, memory depth, associated data partitioning, and internal instruction set. The width provides ample addressing capability without offset calculations, and the partition capability allows the powerful function of intrinsic virtual block addressing. Its capacity and dual associativity with partitioning combination, and the ease of Next Free Address learning enables efficient replacement, further reducing system latency time. Integration of the MU9C1480A virtual memory scheme into a memory management system can be achieved without requiring modification to the operating system or BIOS by the use of a virtual memory driver program to intercept disk file calls. The combination of the CAM tag buffer, control logic, and virtual driver can be transparently placed between the operating system and the disk controller.

These combinations of features, coupled with the ability to locate a large amount of virtual storage anywhere in memory at essentially random addresses, gives the CAM solution an advantage over fast SRAM or small cache tag buffers. The ability of the MU9C1480A LANCAM to easily point to any arbitrary memory area as a virtual block for storage of any size file should give significant acceleration to virtual or mass memory storage systems.

ACKNOWLEDGMENTS:

Teuvo Kohonen, "Content-Addressable Memories" 2nd Edition 1987 Springer-Verlag

NOTES

NOTES

MUSIC Semiconductors Agent or Distributor:

MUSIC SEMICONDUCTORS

http://www.music-ic.com
email: info@music-ic.com

USA Headquarters
MUSIC Semiconductors
254 B Mountain Avenue
Hackettstown, New Jersey 07840
USA
Tel: 908/979-1010
Fax: 908/979-1035
USA Only: 800/933-1550 Tech. Support
888/226-6874 Product Info.

Asian Headquarters
MUSIC Semiconductors
Special Export Processing Zone 1
Carmelray Industrial Park
Canlubang, Calamba, Laguna
Philippines
Tel: +63 49 549 1480
Fax: +63 49 549 1023
Sales Tel/Fax: +632 723 62 15

European Headquarters
MUSIC Semiconductors
Torenstraat 28
6471 JX Eygelshoven
Netherlands
Tel: +31 45 5462177
Fax: +31 45 5463663