HONEYWELL INTEROFFICE CORRESPONDENCE

DATE: APRIL 11, 1973

TO: J. M. RISAN

FROM: C. T. CLINGEN CTC

DIVISION: CISL/PCO

SUBJECT: CACHE MEMORY FOR 6180 MULTICS

CC: JF Couleur
JN Dahl
RF Montee ←
RL Ruth

RECEIVED
[illegible] 12 1973
R F MONTEE

Attached is Steve Webber's proposal for adding cache memory to the 6180 and Multics. In my opinion the strategy employed, which takes advantage of segmentation in the hardware and software, provides a very clean solution to an intrinsically difficult problem. Seems like it is worth patenting.

As you know, Jim Dahl and Steve Webber have been discussing many of the issues described in the proposal; I assume this should continue in order to achieve a final design. Let me know if I can be of any help.

Attachment: Webber's Proposal (Original and copies)

S.H. Webber
April 11, 1973

HARDWARE MODIFICATIONS TO ALLOW CACHE STORAGE IN THE MULTICS HARDWARE

This document is being prepared because of an increasing pressure to find out what types of problems arise when a cache memory is added to the Multics hardware configuration. There are many problems created and these notes are meant to describe some of the more obvious ones with possible solutions. The real purpose of this document, however, is to describe the basic problems so that all problems can be recognized and possibly solved.

The basic cache that is being proposed exists within the CPU's of the system. Each CPU has its own cache which is organized as described later. The basic problem with a cache system organized in this manner is that shared data must be very carefully handled in that it is possible to have multiple copies of data in the multiple caches of a multiple CPU configuration.

General Problems with Caches

A cache, as mentioned here, is a small, fast memory managed by the CPU. It is meant to contain recently referenced words of main (core) memory and be able to retrieve these words when needed without going to core memory for the data. (The cache is of course much faster than core memory and tests made to date show references to the cache add no time to the execution of instructions on the 6080.) The cache data consists of blocks of 4-36 bit words which are addressed (pseudo-associatively) by 24 bit absolute address.

The cache is small - 2048 36 bit words total - and therefore only a very small number of words can reside in the cache at one time. This means that a removal algorithm is needed to purge words from the cache when more cache storage is needed for a recent memory reference. This removal algorithm, although of general interest, is independent of any of the problems we are worrying about in this document.

GENERAL (excerpts from TDM-CPA-18)

The Cache Store is a "Look-Aside Memory" or high speed buffer storage located in the Central Processor. This buffer provides a fast access to blocks of data previously fetched from the Main Storage (memory). The effective access time is approximately 10ns as compared to 650ns for a main memory fetch. This effective access time is obtained by operating the Cache in parallel to existing Processor functions.

Successful usage of a Cache Store requires that a high ratio of storage fetches be made from the Cache. Cache usage must be high with a small time spent in loading the Cache.

Extensive investigation and simulation of Cache storage problems has been done by Systems Engineering. They show a probability of finding the needed fetch in Cache on average programs is between 80% to 95% (hit probability) for a GCOS environment.

BASIC STRUCTURE

The Cache System can be broken into three main areas, - the buffer storage, the associative memory or Directory, and the control area. These areas have been merged into the existing processor design without changing basic processor timing or philosophy. The Cache is not an external "Black-Box".

The Cache is divided into 512 blocks of 4-36 bit words for a total size of 2048. A block (4 words) will be loaded into Cache whenever the program first requires any word of the block. Subsequent references from this block will be made from the Cache until the block is replaced.

CACHE STORE

The Cache storage is 2048 (36 bit words). A single parity bit is carried with each word and is checked the same as on fetches from main memory.

CACHE DIRECTORY

A Directory is used to identify all blocks in Cache storage. 512 "TAG" words are used in the directory to reflect the Absolute Address of each data block. A 24 bit Absolute Address is used such that this Cache will work on a processor with the Extended Memory Addressing option.

Considerable work was spent during simulation on the problem of Directory Organization.

Directory structure and its replacement algorithm have considerable effect on hit probability, number of chips required to implement, time required for search, and control sequence complexity. The hit probability was compromised some in favor of a simpler design using fewer chips to fit the directory and controls on one board.

The mapping strategy used is called 4 level set associative. The directory is divided into 128 columns of 4 levels each. The Main Memory is then divided into "N" number of sections of 128-4 word blocks (512 words). Each block maps directly into a corresponding column of the directory. Each column then can contain addresses of 4 blocks, each from different sections. The replacement algorithm for loading new blocks into a column which is full is on a first in, first out basis and is called Round Robin Organization (RRO).

The directory is built as a small memory with 128 locations. Address Bits 9-15 of the effective address are used to access one of the 128 locations. Each of these locations contain 4 address tag words. Each tag word is the rest of the absolute address, bits 0-8 and E0-E5. Since bits 9-15 of the effective address are the same in the Absolute Address, but are available sooner (only for GCOS?) they are used for directory access. During the same time that directory access is made, the base addition will be completed. The Absolute Address and tag words should be available at the same time for a comparison to be made.

The Cache storage address is developed in the following manner: Effective address bits 16 and 17 will be used to tell which word of the block is needed. Bits 9-15 will be used directly. Two additional bits will be developed from the 1 of 4 compare made. An 11 bit address is therefore available for access of the 2048 word Cache storage.

The Cache directory has a Full/Empty status bit associated with each tag word to indicate that the block is full and the data is valid. The Cache storage can be cleared by resetting all F/E bits. The Cache will be cleared whenever the CPU executes a gating instruction in Master Mode. The Full/Empty bit will be set on when a block load is initiated.

Each of the 128 columns of the directory has a two-bit RRO counter associated with it. Whenever the Cache is cleared, all counters will be reset to 00. This counter will be used to indicate which level or tag is to be loaded next and generates two bits of the Cache storage address on block loads. When a new block is loaded into storage, the absolute address bits 0-8 and E0-E5 will be stored into the directory location accessed by bits 9-15 and then the RRO counter will be advanced.

CACHE CONTROL

Four types of cycles can be performed with the Cache. When a compare is made on a data fetch, the function will be a Cache Read. Data fetches with no compare will cause a block load. Store-ops with a directory compare will cause a Cache Write cycle along with the Port Store cycle. Store-ops with no compare will use Cache for timing purposes along with the Port Store, but will not cause a Cache Write cycle. SCU Controller cycles and Fault and Interrupt cycles will not affect the Cache and will operate normally (as will PTW and SDW fetches and stores).

To increase efficiency of Store-ops, a type of "store-aside" policy was implemented. All Store-ops go to backing store. All Store-ops will also cause a Cache cycle, but only those with a compare will cause a Cache Store to update Cache Storage. (End of excerpts from TDM-CPA-18)

PROBLEMS

The problem which must be solved with any such cache organization is that shared code or data may reside in multiple caches of a multiple CPU configuration. If this data is modified by one CPU the modification does not get reflected to the other CPU's even though the core storage (common to all CPU's) has been updated.

Several solutions to this problem have emerged from the industry. However, all such solutions known to the author require extensive inter-processor communication in the form of explicit control lines and connections between the actual CPU's. In addition to having the transfer of information from 1 CPU directly to another CPU there are also timing and controls required to synchronize acknowledgements of received signals. This type of system organization does not easily fit into the current Honeywell 6000 line architecture and therefore the technique described below is proposed.

Before progressing to the proposed solution, it would be better to point out some assumptions and system constraints posed by the 6180 (Multics) hardware.

(1) The 6180 makes extensive use of the key-lock strategy within the system controllers. Currently the SCU is locked by all read-alter-rewrite instructions in such a way that no other such instructions are allowed access to an SCU while one of them is in progress. This

feature must be preserved with a cache system and although there seems to be no problem here it should be pointed out and not forgotten. The necessary solution is to:

a. fetch from main core on the read cycle of a read-alter-rewrite

b. set the lock (as usual)

c. store the possibly modified data back to main core.

In other words ignore the cache for such instructions.

(2) The 6180 paging mechanism will force the clearing of the cache whenever a new page of data is brought into core.

(3) It must be possible to look at words in <u>segments</u> and be sure the cache is <u>not</u> searched for the given data. This applies to PTW's, bulk store status words, etc.

(4) It must be possible for one processor to signal the other processors to clear their caches (and it must be possible for a CPU to clear its own cache at will).

At present it is being proposed that all PTW's and SDW's as fetched and stored by the appending unit ignore the cache (they are not brought up to the cache). This is quite reasonable in that the associative memories are sufficient to eliminate most core references for these. This also eliminates the obvious problems that would result when one CPU wanted to turn on the referenced bit or modified bit. Such an action would have to be recognized by the other CPU's.

However, there is a problem when the <u>software</u> wants to look at one of the PTW's for example. The software, by its nature, must reference the PTW as a normal data fetch through the appending unit -- i.e., as a word of a segment.

In general, the hardware cannot tell when such a data reference is for a PTW (or any other critical data item which must be fetched from core - such as a status word from an I/O device). One proposed solution is to provide an instruction (or class of instructions) which ignores the cache when fetching data. This solution is very hard to implement in general as it is almost impossible for a translator (compiler) to know when it must generate one of these instructions. It is also asking for trouble when an entire class of programming techniques must be discarded because they may not work with the cache. This would be the case, here, for all users and system programmer's would have to be warned against using arbitrary instructions which might not ignore the cache. There is also the problem that if enough instructions were added to this set the performance of the system would needlessly be degraded.

Another class of problems which must be solved is the shared data base issue. If several users are simultaneously referencing the same data (under protection of some software locking strategy) problems may result if the cache in one CPU contains older copies of data modified by another CPU. For this case it is necessary that the caches be cleared before any reference is made to any such shared data item. Although clever software locking techniques have been worked out these have in general been aided by hardware support in the form of some kind of semaphore function. Even with this additional support, if multiple processors are simultaneously sharing the same data with caches, the before mentioned problems arise.

For the above reasons (and several others of a less critical nature) the following proposal is put forth:

(1) Provide a means in the hardware for the software to specify an entire data block (segment) should bypass the cache. The hardware must be able to easily and efficiently recognize the intent of the software and act accordingly.

(2) Provide software which can unerringly determine when a data block may be used in such a way that the use of the cache for the data block won't lead to trouble.

In the Multics system this is particularly easy in that (1) the software has control over all SDW's in the system and (2) the hardware references the SDW for a segment on each reference to that segment. In particular it is proposed that a bit of the SDW be defined as the "no cache" bit and that if an SDW has this bit on, the hardware is directed to bypass the cache with respect to all references to that segment.

The software must be able to determine when a segment is shared and if it is shared whether or not a problem may arise with the use of the cache. It is easy to see that a problem may arise if more than one processor can modify the segment at the same time. This is true only if more than one SDW of the segment has the write permit bit on. (Future development with tasking may bring about the sharing of SDW's within an address space which means that more than one processor can modify a segment even though there is only one SDW. This is easily remedied.) It is easy for the supervisor software to keep track of all SDW's for a segment (the Multics supervisor does this anyway for a different reason) and hence it is easy for the supervisor to set the "no cache" bit in all SDW's.

The shared supervisor data bases in Multics (including the storage system directories) have the same problems inherent in user data bases in a cache system. The above strategy would dictate that the "no cache" bit would be on for these segments. However, it is possible to remove the restriction, in these special cases, because of our complete knowledge and control over these data bases. We have a separate locking system which guarantees that only one processor will be referencing such data bases at a time. It is merely necessary to clear all cache at lock time. However, after locking, full advantage of caching the data is possible.

It is proposed here that if cache hardware is developed for the 6180 system the software be developed in two stages. First a system be generated which refuses to cache any data which is shared and modifiable (including most of the supervisor data bases). When this is working it will then be possible to selectively improve the system by caching those data bases which are protected in some other way.

Two further proposals to the hardware are:

(1) Provide meters (counters) to determine the effectiveness of the cache - for example, number of cache match successes in relation to number of cache match failures.

(2) A mechanism be provided to <u>selectively</u> clear the cache, i.e., clear only selected entries in the cache. This would seem quite useful (possibly necessary from a cost viewpoint) in that nearly all reasons for clearing the cache will be as a result of a page of core being replaced by new data thereby invalidating any data in the caches for those core address within the page. If only those core addresses affected (which probably aren't in the cache anyway or we wouldn't have taken that core block) are cleared the contents of the cache would remain valid for considerably longer. Preliminary estimates show that without selective clearing the cache will rarely become more than one-half to two-thirds full.

(3) It has been proposed that the cache lookup scheme might use segment number and offset rather than absolute core address in order to overlap cache lookup logic with other functions of the appending

unit. A consequence of this is that the cache would have to be cleared with each ldbr. This might be more frequent than is necessary.

Please make any comments about the above proposals to me.

Written by: Steven H. Webber Date: April 11, 1973
Steven H. Webber

Witnessed by: [illegible] Date: April 11, 1973
C. T. Clingen, Manager -
Cambridge Information
Systems Laboratory