

A CACHE LINE FLUSH INSTRUCTION AND METHOD, APPARATUS, AND SYSTEM FOR IMPLEMENTING THE SAME

BACKGROUND

Field of the Invention

The present invention relates in general to computer architecture, and in particular to a method and system that allow a processor to flush a cache line associated
5 with a linear memory address from all caches in the coherency domain.

Description of the Related Art

A cache memory device is a small, fast memory that is available to contain the most frequently accessed data (or "words") from a larger, slower memory.

Dynamic random access memory (DRAM) provides large amounts of storage
10 capacity at a relatively low cost. Unfortunately, access to dynamic random access memory is slow relative to the processing speed of modern microprocessors. A cost-effective solution providing cache memory is to provide a static random access memory (SRAM) cache memory, or cache memory physically located on the processor. Even though the storage capacity of the cache memory may be relatively small, it provides
15 high-speed access to the data stored therein.

The operating principle behind cache memory is as follows. The first time an instruction or data location is addressed, it must be accessed from the lower speed memory. The instruction or data is then stored in cache memory. Subsequent accesses to the same instruction or data are done via the faster cache memory, thereby
20 minimizing access time and enhancing overall system performance. However, since the storage capacity of the cache is limited, and typically is much smaller than the storage capacity of system memory, the cache is often filled and some of its contents must be changed as new instructions or data are accessed.

The cache is managed, in various ways, so that it stores the instruction or data
25 most likely to be needed at a given time. When the cache is accessed and contains the requested data, a cache "hit" occurs. Otherwise, if the cache does not contain the

11 MAR 2004

requested data, a cache "miss" occurs. Thus, the cache contents are typically managed in an attempt to maximize the cache hit-to-miss ratio.

With current systems, flushing a specific memory address in a cache requires knowledge of the cache memory replacement algorithm.

5 A cache, in its entirety, may be flushed periodically, or when certain predefined conditions are met. Furthermore, individual cache lines may be flushed as part of a replacement algorithm. In systems that contain a cache, a cache line is the complete data portion that is exchanged between the cache and the main memory. In each case, dirty data is written to main memory. Dirty data is defined as data, not yet written to
10 main memory, in the cache to be flushed or in the cache line to be flushed. Dirty bits, which identify blocks of a cache line containing dirty data, are then cleared. The flushed cache or flushed cache lines can then store new blocks of data.

 If a cache flush is scheduled or if predetermined conditions for a cache flush are met, the cache is flushed. That is, all dirty data in the cache is written to the main
15 memory.

 For the Intel family of P6 microprocessors (e.g., Pentium II, Celeron), for example, there exists a set of micro-operations used to flush cache lines at specified cache levels given a cache set and way; however, there is not such a micro-operation to flush a cache line given its memory address.

20 Systems that require high data access continuously flush data as it becomes dirty. The situation is particularly acute in systems that require high data flow between the processor and system memory, such as the case in high-end graphics pixel manipulation for 3-D and video performances. The problems with current systems are that high bandwidth between the cache and system memory is required to accommodate the
25 copies from write combining memory and write back memory.

 Thus, what is needed is a method and system that allow a processor to flush the cache line associated with a linear memory address from all caches in the coherency domain.

11 MAR 2004

SUMMARY

The cache line flush (CLFLUSH) micro-architectural implementation process and system allow a processor to flush a cache line associated with a linear memory address from all caches in the coherency domain. The processor receives a memory address. Once the memory address is received, it is determined whether the memory address is stored within a cache memory. If the memory address is stored within the cache, the memory address is flushed from the cache.

BRIEF DESCRIPTION OF THE DRAWINGS

The inventions claimed herein will be described in detail with reference to the drawings in which reference characters identify correspondingly throughout and wherein:

FIG. 1 illustrates a microprocessor architecture; and

FIG. 2 flowcharts an embodiment of the cache line flush process.

DETAILED DESCRIPTION

By definition, a cache line is either completely valid or completely invalid; a cache line may never been partially valid. For example, when the processor only wishes to read one byte, all the bytes of an applicable cache line must be stored in the cache; otherwise, a cache miss will occur. The cache line forms the actual cache memory. A cache directory is used only for cache management. Cache lines usually contain more data than it is possible to transfer in a single bus cycle. For this reason, most cache controllers implement a burst mode, in which pre-set address sequences enable data to be transferred more quickly through a bus. This is used for cache line fills, or for writing back cache lines, because such these cache lines represent a continuous and aligned address area.

A technique to flush the cache line can be associated with a linear memory address. Upon execution, the technique flushes the cache line associated with the operand from all caches in the coherency domain. In a multi-processor environment, for example, the specified cache line is flushed from all cache hierarchy levels in all microprocessors in the system (i.e. the coherency domain), depending on processor state. The MESI (Modified, Exclusive, Shared, Invalid) protocol, a write-invalidate

protocol, gives every cache line one of four states which are managed by two MESI-bits. The four states also identify the four possible states of a cache line. If the processor is found in "exclusive" or "shared" states, the flushing equates to the cache line being invalidated. Another example is true when the processor is found in "modified" state. If
5 a cache controller implements a write-back strategy and, with a cache hit, only writes data from the processor to its cache, the cache line content must be transferred to the main memory, and the cache line is invalidated.

When compared to other memory macroinstructions, the cache line flush (CLFLUSH) method is not strongly ordered, regardless of the memory type associated
10 with the cache line flush macroinstruction. In contrast, the behavior in the memory sub-system of the processor is weakly ordered. Other macro-instructions, can be used to strongly order and guarantee memory access loads, stores, fences, and other serializing instructions, immediately prior to and right after CLFLUSH.

A micro-operation, named "clflush_micro_op" is used to implement the
15 CLFLUSH macroinstruction.

Moving to FIG. 1, an example microprocessor's memory and bus subsystems is shown with the flow of loads and stores. In FIG. 1, two cache levels are assumed in the microprocessor: an on-chip ("L1") cache being the cache level closest to the processor, and second level ("L2") cache being the cache level farthest from the processor. An
20 instruction fetch unit 102 fetches macroinstructions for an instructions decoder unit 104. The decoder unit 104 decodes the macroinstructions into a stream of microinstructions, which are forwarded to a reservation station 106, and a reorder buffer and register file 108. As an instruction enters the memory subsystem, it is allocated in the load 112 or store buffer 114, depending on whether it is a read or a write memory macroinstruction,
25 respectively. In the unit of the memory subsystem where such buffers reside, the instruction goes through memory ordering checks by the memory ordering unit 110. If no memory dependencies exist, the instruction is dispatched to the next unit in the memory subsystem after undergoing the physical address translation. At the L1 cache controller 120, it is determined whether there is an L1 cache hit or miss. In the case of a
30 miss, the instruction is allocated into a set of buffers, from where it is dispatched to the bus sub-system 140 of the microprocessor. In case of a cacheable load miss, the

instruction is sent to read buffers 122, or in the case of a cacheable store miss, the instruction is sent to write buffers 130. The write buffers may be either weakly ordered write combining buffers 132 or non-write combining buffers 134. In the bus controller unit 140, the read or write micro-operation is allocated into an out-of-order queue 144. If
5 the micro-operation is cacheable, the L2 cache 146 is checked for a hit/miss. If a miss, the instruction is sent through an in-order queue 142 to the frontside bus 150 to retrieve or update the desired data from main memory.

The flow of the "clflush_micro_op" micro-operation through the processor memory subsystem is also described in FIG. 2. Initially, the instruction fetch unit 102
10 retrieves a cache line flush instruction, block 202. In block 204, the cache line flush instruction is decoded into the "clflush_micro_op" micro-operation by the instructions decoder unit 104. The micro-operation is then forwarded to a reservation station 106, and a recorder buffer and register file 108, block 206. The "clflush_micro_op" micro-operation is dispatched to the memory subsystem on a load port, block 208. It is
15 allocated an entry in the load buffer 112 in the memory ordering unit 110. For split accesses calculation in the memory ordering unit 110, the data size of the micro operation is masked to one byte in order to avoid cache line splits; however, upon execution, the whole cache line will be flushed.

The behavior of the "clflush_micro_op" in the memory-ordering unit 110 is
20 speculative. Simply put, this means that the "clflush_micro_op" can execute out of order respect to other CLFLUSH macroinstructions, loads and stores. Unless memory access fencing (termed "MFENCE") instructions are used appropriately, (immediately before and after of the CLFLUSH macro-instruction), execution of the "clflush_micro_op" with respect other memory loads and stores is not guaranteed to be
25 in order, provided there are no address dependencies. The behavior of CLFLUSH through the memory subsystem is weakly ordered. The following tables list the ordering constraints on CLFLUSH. Table 1 lists the ordering constraint affects of later memory access commands compared to an earlier CLFLUSH. Table 2 lists the converse of table 1, displaying the ordering constraint affects of earlier memory access commands
30 compared to a later CLFLUSH instruction. The memory access types listed are

11 MAR 2004

uncacheable (UC) memory, write back (WB) memory, and uncacheable speculative write combining (USWC) memory accesses.

Earlier access	Later access							
	UC memory		WB memory		USWC memory		CLFLUS	MFENCE
	Load	Store	Load	Store	Load	Store		
CLFLUS	N	N	Y	Y	Y	Y	Y	N

Note: N = Cannot pass, Y = can pass.

Table 1: Memory ordering of instructions with respect to an older CLFLUSH

Earlier access		Later access
		CLFLUSH
UC memory	Load	Y
	Store	Y
WB memory	Load	Y
	Store	Y
USWC memory	Load	Y
	Store	Y
CLFLUSH		Y
MFENCE		N

Table 2: Memory ordering of instructions with respect to a younger CLFLUSH

From the memory-ordering unit 110, the "clflush_micro_op" micro-operation is dispatched to the L1 cache controller unit 120, block 210. The "clflush_micro_op" micro-operation is dispatched on the load port; however, it is allocated in a write combining buffer 132, as if it were a store. From the L1 cache controller unit forward, the "clflush_micro_op" is switched from the load to the store pipe.

Decision block 212 determines whether no write combining buffers 132 are available. If none are available, flow returns to block 210. Otherwise, flow continues into block 214. Regardless of the memory type and whether it hits or misses the L1 cache, a write combining buffer 132 is allocated to service an incoming "clflush_micro_op," block 214. A control field is added to each write combining buffer 132 in the L1 cache controller unit to determine which self-snoop attributes need to be sent to the bus controller 140. This control bit, named "clflush_miss," is set exclusively for a "clflush_micro_op" that misses the L1 cache.

Upon entering the memory sub-system of the microprocessor, several bits of the address that enable cache line access of a "clflush_micro_op" are zeroed out, block 216. In the Pentium pro family of microprocessors, these would be the lower five bits of the

11 MAR 2004

address (address[4:0]). This is done in both the L1 cache and L2 cache controller units 120, upon executing the flush command. The zeroing out helps to determine a cache line hit or miss. Since only tag match determines a hit or miss, no byte enable comparison is needed. Note that by definition, no partial hit is possible. A hit or miss is always full line hit or miss. Zeroing out the address bits [4:0] also provides an alternative mechanism to the one used in the memory ordering unit 110 to mask line split accesses. In split accesses the data size of the transaction is masked one byte.

Another control bit added to each write combining buffer 132 in the L1 cache controller unit 120 is used to differentiate between a write combining buffer 132 allocated for a "clflush_micro_op" and another one allocated for a write combining store, block 218. This control bit, named "clflush_op," is exclusively set for those write combining buffers allocated to service a "clflush_micro_op". It is used to select the request type and flush attributes sent from the L1 cache controller 120 to the bus controller 140.

In the case of an L1 cache hit, as determined by decision block 222, both "flush L1" and "flush L2" attributes are sent to the bus controller 140 upon dispatch from the L1 cache controller unit 120, blocks 224 and 226. The bus controller 140 contains both the L2 cache 146 and external bus controller units.

Alternatively, in the case of a L1 cache miss, as determined by decision block 222, the "clflush_miss" control bit is set, and only the "flush L2" attribute is sent blocks 228 and 232. This helps improve performance by omitting the internal self-snoop to the L1 cache.

Upon its dispatch from the memory-ordering unit 110, the "clflush_micro_op" micro-operation is blocked by the L1 cache controller unit 120 if there are no write combining buffers 132 available, block 212. In such a case, it also evicts a write-combining buffer 132, as pointed by the write-combining circular allocation pointer. This guarantees no deadlock conditions due to the lack of free write combining buffers 132. If blocked, the "clflush_micro_op" is redispached once the blocking condition is removed. An example that would cause the dispatching of the "clflush_micro_op" instruction is the completed eviction of a previously allocated write-combining buffer 132.

6. The process of claim 5 further comprising:
executing the cache line flush instruction out of order with respect to a plurality of load and store instructions to uncacheable, writeback, and uncacheable speculative write combining memory type locations executed by the processor; and
maintaining order of the cache line flush instruction with respect to memory fence instructions executed by the processor.

7. A process comprising:
receiving a cache line flush instruction having an associated memory address; and
flushing each instance of the memory address stored within cache memories in a coherency domain by generating a single type of bus transaction in response to the cache line flush instruction regardless of a state of or presence or absence of a cache line corresponding to the memory address in a cache associated with a bus agent executing the cache line flush instruction.

8. The process of claim 7 wherein flushing further comprises:
performing a self snoop to flush the associated memory address from the first bus agent.

9. The process of claim 7 wherein flushing comprises generating a zero length access and an invalidation request.

10. The process of claim 7 wherein at least one bus transaction to request invalidation is generated for each cache line implicated by an operand of an executed cache line flush instruction.

11. An apparatus comprising:
a cache memory; and
a cache controller, the cache controller being operative to receive a memory address which is indicated by an operand of an instruction, wherein the cache controller is to flush a cache line associated with the memory address stored within the cache memory;

and a first bus controller to generate a predetermined bus transaction indicating the memory address in response to the instruction, whether the memory address is cached in a modified state or in another state in the cache memory.

12. The apparatus of claim 11, wherein the predetermined bus transaction is to request invalidation of a cache location corresponding to the memory address, further comprising:

at least one external cache memory; and

a second bus controller to receive the memory address from the first bus controller and to flush instances of the memory address stored within the at least one external cache memory.

13. The apparatus of claim 11, the cache controller further comprising write combining buffers operative to communicate to the bus controller, wherein one of the write combining buffers is allocated in response to the instruction to store an entry indicative of a cache flush operation without data from the cache memory corresponding to the memory address despite data corresponding to the memory address being present in the cache memory in the modified state, and further wherein the apparatus is to generate a self snoop which provides data for a separate bus transaction to write back the cache line associated with the memory address.

14. The apparatus of claim 11 wherein the bus transaction comprises a zero length access and an indication to perform an invalidation.

15. The apparatus of claim 11 wherein the apparatus is a processor and the instruction is a cache line flush instruction which is weakly ordered and can be executed by the processor out of order with respect to other instances of the cache line flush instruction executed by the processor.

16. The apparatus of claim 11 implemented using any combination of one or more of hardware, firmware, and instructions on a computer readable medium.

17. A computer architecture comprising:
a closest cache memory within a coherency domain;
at least one external cache memory within the coherency domain;
a cache controller with a write combining buffer, the cache controller being operative to receive a memory address, to determine whether the memory address is stored within the closest cache memory in the coherency domain, to flush the cache line associated with the memory address stored within the closest cache memory in the coherency domain, and to allocate the write combining buffer to a flush instruction with the memory address;
and
a bus controller, the bus controller being operative to receive the memory address from the write combining buffer and to flush instances of the memory address stored within the external cache memory within the coherency domain.

18. The architecture of claim 17, the bus controller receiving a first control bit indicating whether a write-combining buffer is allocated to the flush instruction with the memory address.

19. The architecture of claim 18, the bus controller further receiving a second control bit indicating whether the memory address is stored within the closest cache memory in the coherency domain.

20. The architecture of claim 19, the bus controller further receiving a third control bit indicating whether the flush instruction with the memory address should be evicted from the write-combining buffer.

21. A process comprising:
receiving a memory address by a cache controller;
determining whether the memory address is stored within a closest cache memory in a coherency domain,

flushing the cache line associated with the memory address stored within the closest cache memory in the coherency domain;
 allocating a flush instruction with the memory address to a write combining buffer;
 transmitting the flush instruction with the memory address to a bus controller through the write combining buffer;
 locating instances of the memory address stored within external cache memory within the coherency domain; and
 flushing instances of the memory address stored within the external cache memory within the coherency domain.

22. The process of claim 21 wherein a first control bit indicates to the bus controller whether a write-combining buffer is allocated to the flush instruction with the memory address.

23. The process of claim 22 wherein a second control bit indicates to the bus controller whether the memory address is stored within the closest cache memory in the coherency domain.

24. The process of claim 23 further comprising:
 evicting the flush instruction with the memory address from the write-combining buffer.

25. The process of claim 24 wherein a third control bit indicates to the bus controller the flush instruction with the memory address should be evicted from the write-combining buffer.

26. An apparatus comprising:
 a cache memory;
 control logic that, in response to an instruction having an associated address indicating a memory location, generates a first type of transaction requesting invalidation of the memory location in other caches regardless of a state of or presence or absence of data

for the memory location in the cache memory, and flushes the memory location from the cache memory, wherein the instruction is a cache line flush instruction, and wherein the associated address is a linear address which is an operand to the cache line flush instruction, and further wherein the first type of transaction is a zero length memory access and invalidate bus cycle that conveys a physical address which is a translation of the linear address.

27. The apparatus of claim 26 wherein the instruction is a weakly ordered instruction which can be executed out of order by the apparatus with respect to other instances of the instruction executed by the apparatus.

28. The apparatus of claim 26 wherein the control logic, in response to the instruction, causes a snoop of the cache memory which invalidates a cache line corresponding to the associated address.

29. The apparatus of claim 26 wherein the control logic comprises:
an instruction fetch unit to fetch the instruction, wherein the instruction is a cache line flush instruction;
an instruction decoder unit to decode the cache line flush instruction to generate a micro-operation;
a reservation station to receive the micro-operation from the instruction decoder unit;
a memory ordering unit to allow the micro-operation to be reordered with respect to other micro-operations but not with respect to fence micro-operations, the other micro-operations being load micro-operations, store micro-operations, and cache line flush micro-operations; and
a bus controller having a bus queue to queue bus transactions.

30. The apparatus of claim 29 further comprising a plurality of write combining buffers, wherein an entry is to be allocated in the plurality of write combining buffers in response to the instruction, and further wherein the entry indicates a cache line flush operation but is allocated without data associated with the memory location from the cache memory.

31. The apparatus of claim 30 wherein the memory ordering unit considers a data size of the micro-operation to be one byte for a split access calculation.

32. The apparatus of claim 30 wherein an entry for the cache line flush instruction is allocated in a write combining buffer as if the cache line flush instruction were a store, and further wherein the micro-operation is dispatched from a load port.

33. The apparatus of claim 32 wherein the memory location is flushed from the cache memory in response to a self snoop initiated by the control logic, and further wherein the self snoop is initiated only if a hit in the cache memory previously occurred in response to the instruction and associated address.

34. The apparatus of claim 26 wherein the first type of transaction comprises a memory access and an indication of a request to invalidate.

35. The apparatus of claim 26 wherein the control logic, in response to the instruction, is to generate at least one bus transaction indicating an implicated cache line for each associated address associated with any instance of the instruction that completes execution.

36. An apparatus comprising:
a cache memory;
instruction fetch logic to fetch a cache flush macroinstruction, the cache flush macroinstruction having an operand that indicates a memory address;
decode logic to decode the cache flush macroinstruction into a micro-operation;
execution logic;
memory ordering logic to allow the micro-operation to be reordered to complete execution by the execution logic out of order with respect to other micro-operations from other cache flush macroinstructions also executed by the execution logic; and

a bus controller that, in response to the micro-operation, generates an invalidate transaction indicative of the memory address and is to cause flushing of a memory location corresponding to the memory address from the cache memory.

37. The apparatus of claim 36 further comprising:

a plurality of write combining buffers, wherein a cache flush operation is received by the write combining buffers without data from the memory address from the cache memory.

38. The apparatus of claim 36, wherein the memory ordering logic allows reordering of the micro-operation with respect to load and store instructions processed by the memory ordering logic to any of uncacheable, writeback, and uncacheable speculative write combining memory type locations, but not with respect to a fence instruction.

39. The apparatus of claim 36, wherein the invalidate transaction is a single type of transaction that is generated in response to the micro-operation regardless of a state or presence or absence of a cache line corresponding to the memory address in the cache.

40. The apparatus of claim 39, wherein the invalidate transaction comprises a zero length memory access and an indication to invalidate the cache line.

41. A system comprising:

a first processor having a first cache caching a first modified data value at a first address;
a second processor having a second cache to cache a second modified data value at a second address, execute a first cache line flush instruction indicating the first address and a second cache line flush instruction indicating the second address, and
in response to the first and second cache line flush instructions to generate a first instance of a first type of bus cycle and a second instance of the first type of bus cycle to the first processor,

wherein the first processor writes back the first modified data value in response to the first instance of the first type of bus cycle generated by the second processor, and further wherein the second processor writes back the second modified data value for the second instance of the first type of bus cycle.

42. The system of claim 41 wherein the first type of bus cycle is generated in response to execution of a cache line flush instruction by the second processor regardless of presence or absence of or a state of a cache line corresponding to an address indicated by the cache line flush instruction in a cache of the second processor.

43. The system of claim 42 wherein the cache line flush instruction is a weakly ordered instruction executable out of order with respect to other memory operations including other cache line flush instructions executed by a single processor.

44. The system of claim 41 wherein the cycle comprises an access of zero length and invalidate line indication.

45. An apparatus comprising:
 a cache;
 an instruction fetch unit to receive a cache flush instruction having a linear address as an operand;
 ordering logic to allow the cache flush instruction to execute out of order with respect to other instructions including other load and store instructions and other instances of the cache flush instruction; and
 control logic to invalidate a cache line in the cache corresponding to the linear address if present in the cache and to signal other caches to invalidate any present cache lines corresponding to the linear address.

46. The apparatus of claim 45 wherein the control logic is to signal other caches regardless of a state of or presence or absence of the cache line in the cache.

47. The apparatus of claim 46 wherein the control logic is to generate a single type of bus transaction in response to the cache flush instruction regardless of a state of or presence or absence of the cache line in the cache.

48. The apparatus of claim 45 wherein the control logic is to invalidate the cache line as a part of a self snoop performed in conjunction with snooping other caches.

49. A system comprising:

a bus;

a main memory coupled to the bus; and

a plurality of caching agents coupled to the bus forming a coherency domain, wherein execution of a cache line flush instruction implicating a cache line by an executing one of the plurality of caching agents is to cause invalidation of the cache line in all of the plurality of caching agents in the coherency domain by generating a first type of bus transaction received by the plurality of caching agents whether or not the cache line is modified in the executing one of the plurality of caching agents.

50. The system of claim 49 wherein the executing one of the plurality of caching agents is to generate a memory access and invalidate cycle to snoop caches of each of the plurality of caching agents.

51. The system of claim 50 wherein the memory access and invalidate cycle comprises a zero length memory transaction and an indication requesting that an invalidation be performed.

52. The system of claim 51 wherein any one of the plurality of caching agents that is caching the cache line in a modified form, including the executing one, is to provide the cache line for the first type of bus transaction when it occurs.

53. The system of claim 52 further comprising means for buffering the cache line flush instruction with a caching agent.