

9800675-2

Pros copy

- 1 -

REPLACED SHEETS		SG
☒ Description	: paged 3 to paged -	
☒ Claim(s)	: paged 15 to paged -	
☐ Drawing(s)	: paged to paged	
☒ Abstract	: paged 16 to paged -	

**METHOD OF LAYERING CACHE AND ARCHITECTURAL
SPECIFIC FUNCTIONS**

BACKGROUND OF THE INVENTION

1. Technical Field:

The present invention relates in general to cache controllers in data processing systems and in particular to cache controllers which layer cache and architectural specific functions. Still more particularly, the present invention relates to layering cache and architectural specific functions within a controller to improve performance and simplify

design.

2. Description of the Related Art:

Data processing systems which utilize a level two (L2) cache typically include a cache controller for managing transactions affecting the cache. Such cache controllers are conventionally implemented on a functional level, as depicted in Figure 3. For example, a cache controller 302 may include logic 304 for maintaining the cache directory, logic 306 for implementing a least recently used (LRU) replacement policy, logic for managing reload buffers 308, and logic for managing store-back buffers 310. In traditional implementations, the cache is generally very visible to these and other architectural functions typically required for cache controllers, with the result that cache controller designs are specific to a particular processors such as the PowerPC™, Alpha™, or the x86 family of processors.

The prior art approach imposes stringent and complex design requirements on the controller implementation. The maximum frequency obtainable is limited by the interlocks required. Testing and formal verification of the design may be frustrated by the complexity of the design. It would be desirable, therefore, to simplify the design of a cache controller to eliminate such complexities.

SUMMARY OF THE INVENTION

It is therefore one object of the present invention to provide an improved cache controller for a data processing system.

It is another object of the present invention to provide an improved cache controller having layered cache and architectural specific functions.

It is yet another object of the present invention to layer cache and architectural specific functions within a controller to improve performance and simplify design.

The foregoing objects are achieved as is now described. Cache and architectural specific functions are layered within a controller, simplifying design requirements. Faster performance may be achieved and individual segments of the overall design may be individually tested and formally verified. Transition between memory consistency models is also facilitated. Different segments of the overall design may be implemented in distinct integrated circuits, allowing less expensive processes to be employed where suitable.

The above as well as additional objects, features, and advantages of the present invention will become apparent in the following detailed written description.

This Sheet Has Been Replaced



BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself however, as well as a preferred mode of use, further objects and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 depicts a data processing system implemented with a nonshared cache controller design in accordance with a preferred embodiment of the present invention;

Figure 2 is a block diagram of a nonshared cache controller design in accordance with a preferred embodiment of the present invention; and

Figure 3 depicts a prior art cache controller design.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

With reference now to the figures, and in particular with reference to Figure 1, a data processing system implemented with a nonshared cache controller design in accordance with a preferred embodiment of the present invention is depicted. Data processing system 100 may include only a single processor or may be a symmetric multiprocessor (SMP) system including a plurality of processors. A single processor system is shown in the example depicted. Processor 102 may be a superscalar reduced instruction set computing (RISC) processor including separate level one instruction and data caches 104 and 106 within the processor. A PowerPC™ processor may be utilized for processor 102.

Processor 102 is connected to a level two (L2) cache 108, which is a nonshared cache. A second processor (not shown) may be added to the system depicted, either with a separate L2 cache or sharing L2 cache 108 with processor 102. L2 cache 108 is connected to system bus 110 for data processing system 100. Local memory 112 is also connected to system bus 110, as is I/O bus bridge 114. Other devices, such as memory-mapped graphics adapter 116, may also be connected to system bus 110. I/O bus bridge 114 is connected to I/O bus 118, which may be connected to a variety of other devices such as local area network (LAN) adapter 120 and hard disk drive 122.

Those of ordinary skill in the art will appreciate that the hardware depicted in Figure 1 may vary. For example, other peripheral devices, such as optical disk drive and the like also may be used in addition or in place of the hardware depicted. The depicted example is not meant to imply

architectural imitations with respect to the present invention. In particular, a data processing system need not be limited to a single processor as shown in the depicted example to benefit from the present invention. The present invention may be employed, for example, to improve the performance of a data processing system having two processors, each with a corresponding L2 cache.

Referring to Figure 2, a block diagram of a nonshared cache controller design in accordance with a preferred embodiment of the present invention is illustrated. Controller 202 is implemented within cache 108 depicted in Figure 1. Controller 202 includes a bus interface unit (BIU) 204 connected to an upper bus 206 for a processor or a higher level cache, and a bus interface unit 208 to a lower bus 210, which may be a system bus or a bus to another cache. Upper bus 206 and lower bus 210 may differ; upper bus 206 may be, for example, a 60X bus, while lower bus 210 may be a different bus.

Cache and architectural specific functions within controller 202 are layered. Thus, controller 202 includes cache controller 212 and architectural controller 214. Operations are distinguished as "cache" or "architectural" operations. Only cache operations are handled by cache controller 212, and only cache controller 212 performs operations on cache 216. Architectural operations are handled by architectural controller 214 and are seen by cache controller 212 as system-side operations.

A third unit, noncacheable controller 218, is also contained within cache controller 202. Noncacheable

controller 218 is actually a counterpart to cache controller 212 in that it also handles only cache operations. Whereas cache controller 212 handles cache operations directed at cache memory locations, noncacheable controller 218 handles cache operations directed at memory locations which do not map to cache 216. It is advantageous, for reasons known to those skilled in the art, to treat part of the system memory as noncacheable. Such memory may be utilized, for example, by memory mapped devices. While cache controller 212 operates on full cache blocks, noncacheable controller 218 operates on smaller memory segments, typically less than 8-16 bytes. Moreover, noncacheable controller 218 does not store data, while cache controller 212 retains copies of data handled within cache 216.

Cache operations are typically those operations which read or write values to memory locations, and therefore may change or retrieve the value of data in a memory location. Cache operations are generally those simple operations which result from LOAD or STORE instructions, or equivalent instructions, within a processor. The remaining operations are defined as architectural operations. Unlike cache operations, architectural operations are more complicated and generally do not change the value of data in a memory location. An architectural operation may move the data to a different location within the cache hierarchy, change the status of data in a particular memory location, or perform other such functions. However, architectural operations generally do not directly alter the value of data within a memory location. An example of an exception is the data cache block zero (DCBZ) instruction in the PowerPC™ architecture, which is an architectural operation which alters the values of

data in a memory location on a cache block basis.

Cache operations, supported by cache controller 212, comprise the largest majority of operations affecting the system cache. Within the complete set of operations supported by a given processor, cache operations may not derive from the portion of the instruction set which is most frequently executed and/or consume the largest majority of processor cycles. However, disregarding instructions directed to other functional units within the processor, such as the floating point, fixed point, or branch units, cache operations are, collectively, executed most often and utilize the largest measure of time.

The remaining operations affecting a system cache--those employed for cache management, operating system management, page management, and synchronization, etc.--are layered out and supported by architectural controller 214. Virtually all processor architectures support such operations, which are utilized in real time operation much less frequently than cache operations. Additionally, individual operations among the architectural operations are generally implemented, if at all, in substantially divergent manners for different processors of interest.

Different designs may vary the set of operations supported by the cache controller and, by default, the remaining operations layered for support by the architectural controller. However, increasing the number of operations supported by the cache controller increases the complexity of logic required. Additionally, if instructions selected for

support by the cache controller are not supported by all processors of interest, the cache controller design loses its direct transferability to new controller designs.

While certain operations pass down only one path within controller 202--that is, through architectural controller 214 or cache controller 212--other operations are split and pass down both paths. Cache controller 212 employs a pass-through design, in which operations initiated at interface 220 generate a response at interface 222 while operations initiated at interface 222 produce a responsive action at interface 220.

Because cache and architectural operations are layered within controller 202, bus transactions and protocols may also be layered. That is, generic interfaces may be defined for cache controller 212, architectural controller 214, and noncacheable controller 218. Thus, interfaces 220-230 comprise generic protocol interfaces to bus interface units 204 and 208 which are, to the extent possible, not architecturally specific. This decouples the design for cache controller 212 from the specific protocols of bus 206 and bus 210, allowing the design for cache controller 212 to be reused. Bus interface units 204 and 208 are responsible for managing transactions and protocols to bus 206 and system bus 210, translating the specific bus transactions into the protocol for generic interfaces 220-230. By employing generic interfaces for interfaces 220-230, the designs for controllers 212, 214, and 218 are isolated from specific bus architectures and may be readily duplicated.

In contrast to traditional cache controllers, cache

controller 212 may thus be implemented in a manner independent of the two buses 206 and 210, responding only to cache operations. Although such cache operations are initiated by transactions on either bus 206 or bus 210, only certain bus transactions will prompt a response within cache controller 212. In a preferred embodiment, cache controller 212 only responds to instruction fetch operations (IFETCH), LOAD operations, and WRITE operations on bus 206, and to READ operations, WRITE operations, and traditional SNOOPS on bus 210. This results in substantially simplified design requirements for cache controller 212. This is accomplished by avoiding the usual practice of overlaying the highly irregular (semantically and temporally) architectural operations and cache operations. The burden of responding to the architectural operations is removed from the design of cache controller 212 and placed in architectural controller 214.

The cache operations handled by cache controller 212 are supported by every commercial processor of interest in substantially the same form. Only minor differences in specific implementation, from which cache controller 212 in the present invention is decoupled by generic interfaces 220 and 222, distinguish comparable instructions for different processors of interest.

By layering selected cache and architectural functions, and implementing generic interfaces to bus interface units 204 and 208, a large portion of the overall design of controller 202 may be directly transferred to new implementations. The cache controller logic may be reused without significant modification for cache operations. New sleeves of logic for

the bus interface units may be easily implemented for handling new bus protocols and converting the generic protocol interfaces 220-230 of cache, architectural, and noncacheable controllers 212, 214, and 218 to interfaces for bus 206 and bus 210. The most significant effort for implementing a design supporting a different processor is required by the architectural controller. Individual design of the logic supporting the architectural operations is required in any case since processor architectures vary dramatically. Overall, however, a significant savings in design effort for different processors may be achieved since only the semantics of operations handled by architectural controller 214 will change.

By layering cache and architectural functions, limiting cache controller 212 to responding to a few fundamental operations, the cache controller logic is greatly streamlined and simplified. In addition, the architectural controller logic is also simplified since, by separating the two classes of operations, issues of interrelationships between operations in different classes are eliminated. The cache and architectural controllers may be designed as individual units.

Layering cache and architectural functions allows those operations which are performed most frequently--cache operations--to be executed as fast as possible. The remaining architectural operations, which may occur as infrequently as once every 20,000 operations, are layered out handled in the existing (and necessary) snoop logic for controller 202.

Layering cache and architectural functions also provides benefits in designing cache controller 212. In addition to

being able to concentrate design efforts on simply running a few instructions as fast as possible, the design for cache controller 212 may be separately completed and tested. The complex logic required for architectural controller 214 need not impede completion of the cache controller design, which may be tested utilizing the generic interfaces.

Formal verification attempts to mathematically prove the operation of a design. Such formal verification is virtually impossible with extremely large, complicated designs. The modular approach of the present invention permits independent formal verification of different units within the overall design.

Independent verification of the cache controller design is important since cache controller 212 still contains the LRU logic and other essential logic discussed earlier. Additionally, bugs are more difficult to locate when contained within the mainstream path of the design if the mainstream path is complicated. By layering the more complex logic required for architectural controller 214 and the logic required for cache controller 212, the bugs become easier to locate. Moreover, with the layered design, bugs within the architectural controller 214 are less likely to impact operation of cache controller 212, which may continue to operate normally.

Because the complex, dedicated logic required for architectural operations is segregated, cache controller 212 may be pipelined and may operate at processor frequency while architectural controller 214 is run at a much lower frequency. In fact, the architectural controller 214 may be implemented in a separate chip or on a separate die, using a less

expensive process.

Since the operations handled by architectural controller 214 occur so infrequently, an overall performance increase in controller 202 may actually be realized. The design allows instructions which are executed most frequently to be performed as fast as possible by streamlining cache controller 212 to only the essential logic.

Providing dedicated hardware for architectural operations directly in the mainstream path of the controller enormously increases the complexity of the logic required and impacts pipelining and frequency within the core controller logic. Nonetheless, traditional designs include the slow, scalar, nonpipelined logic required to support architectural instructions within the mainstream path, slowing the entire cache controller.

Layering cache and architectural allows pipelining of cache operations, and simplifies design requirements by eliminating the need for interlocks. Simplifying the design directly impacts the clock frequency (MHz) which may be obtained, since more complicated logic requires more interlocks, increasing cycle time.

Layering allows the design to be implemented in simple, largely autonomous units. Layering also allows cache controller designs to be more resilient to changing from one memory consistency model to another. The controller design may be developed more expeditiously during migration--from one memory consistency model to another, one processor to another, or both--by altering the cache and architectural controllers

separately.

While the invention has been particularly shown and described with reference to a preferred embodiment, it will be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention.

This Sheet Has Been Replaced



CLAIMS

1. A method of layering operations in a controller, comprising:

performing cache operations in a first controller unit;
and

performing architectural operations in a second controller unit, wherein operations within said controller are layered.

2. A controller, comprising:

a first controller unit performing cache operations; and

a second controller unit performing architectural operations, wherein operations within said controller are layered.



* C L M S *