



166166

CACHE POLLUTION AVOIDANCE INSTRUCTIONS



G00002

BACKGROUND OF THE INVENTION1. Field of the Invention

The present invention relates in general to the field of computer systems, and in particular, to an apparatus and method for providing instructions which facilitate the control of cache accesses while minimizing cache pollution.

2. Description of the Related Art

The use of a cache memory with a processor facilitates the reduction of memory access time. The fundamental idea of cache organization is that by keeping the most frequently accessed instructions and data in the fast cache memory, the average memory access time will approach the access time of the cache. To achieve the optimal tradeoffs between cache size and performance, typical processors implement a cache hierarchy, that is, different levels of cache memory. The different levels of cache correspond to different distances from the processor core. The closer the cache is to the processor, the faster the data access. However, the closer the cache is to the processor, the more costly it is to implement. As a result, the closer the cache level, the faster and smaller the cache.

The performance of cache memory is frequently measured in terms of its hit ratio. When the processor refers to memory and finds the data in its cache, it is said to produce a hit. If the data is not found in cache, then it is in main memory and is counted as a miss. If a miss occurs, then an allocation is made at the entry indexed by the address of the access. The access can be for loading data to the processor or storing data from the processor to memory. The cached information is retained by the cache memory until it is no longer needed, made invalid or replaced by other data, in which instances the cache entry is de-allocated.

- 2 -

A computer system may utilize one or more levels of cache memory.

Allocation and de-allocation schemes implemented for the cache for various known computer systems are generally similar in practice. General practice has been to allocate an entry in the cache for all accesses required by the processor. Accordingly, 5 system architectures specify re-use of accessed data without notion of relevant cache hierarchy level. That is, all accesses are allocated in cache. A disadvantage of this approach is that in certain instances, certain data is referenced once and not reused in the immediate future. It is not desirable for such data, typically termed as non-temporal data, to overwrite data that is used frequently (i.e., temporal data) such as 10 application-cached code and data, when cache memory is full.

BRIEF SUMMARY OF THE INVENTION

A computer system and method for providing cache memory management is disclosed. The computer system comprises a main memory having a plurality of main memory addresses each having a corresponding data entry, and a processor
5 coupled to the main memory. At least one cache memory is coupled to the processor. The at least one cache memory has a cache directory with a plurality of addresses and a cache controller having a plurality of data entries corresponding to the plurality of addresses. The processor receives an instruction having an operand address and determines if the operand address matches one of the plurality of
10 addresses in the cache directory. If so, the processor updates a data entry in the cache controller corresponding to the matched address. Otherwise, a data entry corresponding to the operand address in the main memory is updated.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention is illustrated by way of example, and not limitation, in the figures. Like reference indicate similar elements.

Figure 1 illustrates an exemplary computer system in accordance with one
5 embodiment of the invention.

Figure 2 illustrates one embodiment of the organizational structure of a cache memory e.g., L0, L1 and/or L2 according to one embodiment of the invention.

Figure 3 illustrates one embodiment of the format of the cache pollution avoidance instruction 165 provided according to one embodiment of the invention.

10 Figures 4A and 4B illustrate the operation of the cache pollution avoidance instruction according to one embodiment of the invention.

Figures 5A and 5B illustrate the operation of the cache pollution avoidance instruction according to an alternate embodiment of the invention.

Figure 6 is a flowchart illustrating one embodiment of the process of the
15 present invention.

DETAILED DESCRIPTION OF THE INVENTION

In the following description, numerous specific details are set forth to provide a thorough understanding of the invention. However, it is understood that the invention may be practiced without these specific details. In other instances, well-known circuits, structures and techniques have not been shown in detail in order not to obscure the invention.

Figure 1 illustrates one embodiment of a computer system 100 which implements the principles of the present invention. Computer system 100 comprises a processor 105, a storage device 110, and a bus 115. The processor 105 is coupled to the storage device 110 by the bus 115. The storage device 110 represents one or more mechanisms for storing data. For example, the storage device 110 may include read only memory (ROM), random access memory (RAM), magnetic disk storage mediums, optical storage mediums, flash memory devices and/or other machine readable mediums. In addition, a number of user input/output devices, such as a keyboard 120 and a display 125, are also coupled to the bus 115. The processor 105 represents a central processing unit of any type of architecture, such as multi-threaded CISC, RISC, VLIW, or hybrid architecture. In addition, the processor 105 could be implemented on one or more chips. The storage device 110 represents one or more mechanisms for storing data. For example, the storage device 110 may include read only memory (ROM), random access memory (RAM), magnetic disk storage mediums, optical storage mediums, flash memory devices, and/or other machine-readable mediums. The bus 115 represents one or more buses (e.g., AGP, PCI, ISA, X-Bus, VESA, etc.) and bridges (also termed as bus controllers). While this embodiment is described in relation to a single processor computer system, the invention could be implemented in a multi-processor computer system.

- 6 -

In addition to other devices, one or more of a network 130, a TV broadcast signal receiver 131, a fax/modem 132, a digitizing unit 133, a sound unit 134, and a graphics unit 135 may optionally be coupled to bus 115. The network 130 and fax modem 132 represent one or more network connections for transmitting data over a machine readable media (e.g., carrier waves). The digitizing unit 133 represents one or more devices for digitizing images (i.e., a scanner, camera, etc.). The sound unit 134 represents one or more devices for inputting and/or outputting sound (e.g., microphones, speakers, magnetic main memories, etc.). The graphics unit 135 represents one or more devices for generating 3-D images (e.g., graphics card). Figure 1 also illustrates that the storage device 110 has stored therein data 136 and software 137. Data 136 represents data stored in one or more of the formats described herein. Software 137 represents the necessary code for performing any and/or all of the techniques described with reference to Figures 2, and 4-6. Of course, the storage device 110 preferably contains additional software (not shown), which is not necessary to understanding the invention.

Figure 1 additionally illustrates that the processor 105 includes decode unit 140, a set of registers 141, and execution unit 142, and an internal bus 143 for executing instructions. The processor 105 further includes two internal cache memories, a level 0 (L0) cache memory which is coupled to the execution unit 142, and a level 1 (L1) cache memory, which is coupled to the L0 cache. An external cache memory, i.e., a level 2 (L2) cache memory 172, is coupled to bus 115 via a cache controller 170. The actual placement of the various cache memories is a design choice or may be dictated by the processor architecture. Thus, it is appreciated that the L1 cache could be placed external to the processor 105. In alternate embodiments, more or less levels of cache (other than L1 and L2) may be implemented.

- 7 -

Of course, the processor 105 contains additional circuitry, which is not necessary to understanding the invention. The decode unit 140, registers 141 and execution unit 142 are coupled together by internal bus 143. The decode unit 140 is used for decoding instructions received by processor 105 into control signals and/or micro code entry points. In response to these control signals and/or micro code entry points, the execution unit 142 performs the appropriate operations. The decode unit 140 may be implemented using any number of different mechanisms (e.g., a look-up table, a hardware implementation, a PLA, etc.). While the decoding of the various instructions is represented herein by a series of if/then statements, it is understood that the execution of an instruction does not require a serial processing of these if/then statements. Rather, any mechanism for logically performing this if/then processing is considered to be within the scope of the implementation of the invention.

The decode unit 140 is shown including a fetching unit 150 which fetches instructions, and an instruction set 160 for performing operations on data. In one embodiment, the instruction set 160 includes a cache pollution avoidance instruction 165 provided in accordance with the present invention. Examples of the cache pollution avoidance instruction 165 includes the following instructions: a byte mask write instruction(s) (such as MASKMOVQ), a move 64 bits non temporal instruction(s) (such as MOVNTQ), and a move aligned four packed single-floating point (FP) non temporal instruction(s) (such as MOVNTPS). A byte mask instruction typically performs a write using a mask pattern. A mask pattern indicates the bit position at which a write can be performed. In one embodiment, a '0' corresponds to no write (mask) and a '1' corresponds to a write (unmask). Therefore, a mask pattern of 00100100 allows writing to the bit positions 2 and 5 while other bit positions are masked off. The MASKMOVQ and MOVNTQ

- 8 -

instructions are applicable to integer data. In particular, the MASKMOVQ instruction moves 64-bits representing integer data from a register to the memory location specified and the MOVNTQ instruction moves 64-bits representing integer operands from a register to memory. The MOVNTPS instruction is applicable to
5 packed floating point data, in which the results of an operation between two sets of numbers having a predetermined number of bits, are stored in a register having the same predetermined number of bits, i.e., the size or configuration of the operand is the same as that of the result register. While certain cache pollution avoidance instruction, are described for use with particular type and amount of data, alternate
10 embodiment, can support instructions that operate on different types and amounts of data.

In addition to the cache pollution avoidance instruction(s) 165, processor 105 can include new instructions and/or instructions similar to or the same as those found in existing general purpose processors. For example, in one embodiment the
15 processor 105 supports an instruction set which is compatible with the Intel® Architecture instruction set used by existing processors, such as the Pentium®II processor. Alternative embodiments of the invention may contain more or less, as well as different instructions and still utilize the teachings of the invention.

The registers 141 represent a storage area on processor 105 for storing
20 information, such as control/status information, scalar and/or packed integer data, floating point data, etc. It is understood that one aspect of the invention is the described instruction set. According to this aspect of the invention, the storage area used for storing the data is not critical. The term data processing system is used herein to refer to any machine for processing data, including the computer
25 systems(s) described with reference to Figure 1.

Figure 2 illustrates one embodiment of the organizational structure of a cache memory e.g., L0, L1 and/or L2. For present discussion purposes, the cache memory L0, L1 and/or L2 will be referred to as cache memory 200. Cache memory 200 comprises a cache directory 210 and a plurality of cache entries 220₁-220_n (collectively referred to as cache entries 220). The cache directory 210 stores a plurality of addresses 212₁-212_n of the memory locations in storage device 110, corresponding to the cache entries 220₁-220_n. These memory locations in storage device 110 have addresses 230₁-230_n.

Figure 3 illustrates one embodiment of the format of the cache pollution avoidance instruction 165 provided in accordance with the present invention. The cache pollution avoidance instruction 165 comprises an operational code (OP CODE) 310 which identifies the operation of the cache pollution avoidance instruction 165 and an operand 312 which specifies the address of the data object that the instruction 165 will be operating on. In order to understand the present invention, certain terminology must be understood in reference to the cache pollution avoidance technique. The present invention operates within the framework wherein the particular data being accessed will have (or not have) temporal locality. Temporal locality is an attribute associated with data and determined by how soon in the future a program will access the data. Thus, for each data pattern, the data may have temporal locality (T) or not have temporal locality (NT) with respect to a cache memory or a cache level in the cache hierarchy.

In Figure 1, three levels of cache hierarchy are shown (representing levels L0, L1 and L2), wherein at each level there are two possibilities for classifying the data pattern being accessed. The classifications are noted as 1) temporal (T); and 2) non-temporal (NT). The classifications are based on the attributes associated with the

data access for a computer system. It is appreciated that three levels of cache hierarchy are shown in Figure 1, but there could be more or less cache levels. For example, the present invention could be practiced where there is only one cache level (L0 only) or where there are only two cache levels (L0 and L1), or where there
5 are four or more cache levels.

In the practice of the invention, the temporal property is associated with how close to the processor 105 the data is stored or saved. Accordingly, the temporal property is associated with the use or re-use of data at a given level. For example, if a particular data pattern in the program is identified to be T with respect to L1, but
10 NT with respect to L2, then this data will be used in the near future in the L1 cache, but not so near in the future in the L2 cache. The temporal distance of how soon the data will be used or re-used is application dependent for a particular computer system and software. When data access is regarded as T at a given cache level, it will likely be re-used (assuming no branches taken, etc.) within a certain time frame (for
15 example, within x number of instructions) in the near future. Where data access is regarded as NT at a given level, it will likely not be re-used within the specified time frame.

Thus, the noted temporal criteria can be set based on the action required of the data. Subsequently, the present invention provides for a scheme in which
20 cache pollution avoidance can be implemented at the cache memory or at each cache hierarchy level depending on the attributes associated with the data at a particular level. In particular, by using the cache pollution avoidance instruction 165 to process data, the programmer is indicating that the associated data does not have temporal locality, i.e., the associated data is non-temporal. The associated data
25 is then processed according to the cache pollution avoidance instruction 165 used.

Thus, cache pollution avoidance can be based on none, one, or more than one, of the categories available at each of the cache levels.

In Figure 1, only L0, L1 and L2 levels are shown, but it is appreciated that more or less levels (e.g., one level) can be readily implemented. The embodiment shown in Figures 4-6 describes the use of the invention with respect to one cache level.

The cache pollution avoidance technique of the invention will now be described. Upon receiving the cache pollution avoidance instruction 165, the cache directory 210 of cache memory 200 is first polled to determine if the address specified in the operand 312 matches one of the addresses stored in the cache directory 210. If so, there is a cache hit, and the corresponding data object (i.e., contents of the cache memory corresponding to the addresses specified in the operand) is updated. In one embodiment where different levels of cache memories are used, for example, L0, L1, L2, any one cache level or any combination of the cache memories are updated. In an alternate embodiment, both the corresponding data object in cache memory and in main memory are updated.

If the address specified in the operand 312 does not match any of the addresses stored in the cache directory 210, there is a cache miss. In this case, there is no cache line fill, i.e., the corresponding data object in storage device 110 will not be transferred into the cache memory 210. By not performing the cache line fill, the processor 105 avoids polluting the cache.

Figures 4A and 4B illustrate the operation of the cache pollution avoidance instruction according to one embodiment of the invention, in which there is a cache hit. Upon receiving the cache pollution avoidance instruction 165', the cache

directory 210 of cache memory 200 is polled to determine if the address, e.g., address X, specified in the operand 312' matches one of the address stored in the cache directory 210. If the cache memory 200 is an internal cache memory, i.e., a L0 or L1 cache memory, the processor 105 conducts the polling. If the cache memory 200 is an external cache memory, e.g., L2 cache memory, polling is conducted by the cache controller. In one embodiment, a comparator 410 compares the address X specified in the operand 312' to determine if it matches one of the addresses 212₁-212_n stored in cache directory 210 (Figure 4A). In the present case, there is an address match, indicating that there is a cache hit, and the comparator 400 issues a first signal to the cache line fill logic 420, which determines if a signal should be sent to either the data object 212₄ corresponding to the address X in cache directory 210 or to both the data object 212₄ and to the corresponding data object in 110₄ in storage device 110 to update the data object 212₄ and/or 110₄. The required signal(s) are then issued and updating of the corresponding data object 212₄ and/or 110₄ is performed (Figure 4B).

Figures 5A and 5B illustrate the operation of the cache pollution avoidance instruction according to an alternate embodiment of the invention in which there is cache miss. Upon receiving the cache pollution avoidance instruction 165", the cache directory 210 of cache memory 200 is polled to determine if the address, e.g., address Y, specified in the operand 312" matches one of the address stored in the cache directory 210. In particular, the comparator 410 compares the address Y specified in the operand 312" to determine if it matches one of the addresses 212₁-212_n stored in cache directory 210 (Figure 5A). In this case, the address Y specified in the operand 312" does not match any one of the addresses 212₁-212_n stored in the cache directory 210. As a result, the comparator 410 issues a second signal to the cache line fill logic 420, indicating that there is a cache miss. In response, the cache line fill logic issues only a signal to the storage device 110 to update the contents of

- 13 -

address Y. However, a cache line fill is not performed, i.e., the corresponding data object 1105 in storage device 110 will not be transferred into the cache memory 210. The cache memory 210 therefore remains intact and is not polluted by the non-temporal data.

5 In one embodiment, the cache pollution avoidance instruction 165 is a weakly ordered instruction. That is, when processing a plurality of the cache pollution avoidance instructions 165, the execution of the plurality of cache pollution avoidance instructions are not performed in the same order as they are fetched by the fetching unit in the decoder. In an alternate embodiment, the cache
10 pollution avoidance instruction 165 has the property of write collapsing, whereby when executing successive stores to the same address, only the last store will be visible on the bus 115. In a further alternate embodiment, the cache pollution avoidance instruction 165 has the property of write combining, where cacheable store misses to the same cache line are combined before the write is issued to storage
15 device 110.

 Figure 6 is a flowchart illustrating one embodiment of the process of the present invention. Beginning from a start state, the process 600 proceeds to a decision block 610, where it determines if the address in the cache pollution avoidance instruction 165 matches one of the addresses in the cache directory 210. If
20 so, the process 600 advances to decision block 612, where it determines if only the corresponding cache entry is to be updated or if both the corresponding cache entry and the corresponding entry in storage device 110 have to be updated. If only the corresponding cache entry is to be updated, the process 600 proceeds to process block 614, where it updates the corresponding entry in cache memory (block 614).
25 Otherwise, it updates both the corresponding cache entry and the corresponding

entry in storage device 110. In either case, the process 600 proceeds to process block 618, where the process terminates. If it is determined, at decision block 610, that the address in the cache pollution avoidance instruction 165 does not match one of the addresses in the cache directory 210, the process S600 does not perform a cache line
5 fill, and instead, only updates the corresponding entry in storage device 110 (process block 620).

The use of the present invention thus enhances system performance by not performing a cache line fill during non-temporal accesses. In particular, since no cache line fill is performed during non-temporal access, system performance is
10 improved by reducing contention for the cache, as frequently used data (i.e., temporal data) is not evicted from the cache by non-temporal data. As a result, cache pollution or disturbance during non-temporal accesses is minimized.

While a preferred embodiment has been described, it is to be understood that the invention is not limited to such use. In addition, while the invention has been
15 described in terms of several embodiments, those skilled in the art will recognize that the invention is not limited to the embodiments described. The method and apparatus of the invention can be practiced with modification and alteration within the spirit and scope of the appended claims. The description is thus to be regarded as illustrative instead of limiting on the invention.

CLAIMS:

1. A computer system

comprising:

a main memory having a plurality of main memory addresses each having a corresponding data entry;

a bus;

a processor coupled to said main memory by the bus, the processor comprising:

at least one cache memory , said at least one cache memory having a cache directory with a plurality of addresses and a cache controller having a plurality of data entries corresponding to said plurality of addresses ;

a register;

an execution unit to execute an instruction having an opcode and an operand address to specify a memory location, wherein said processor is to perform a selective copying, based on a mask, of portions of data from the register to the memory location specified by the operand address and to prevent the data copied by execution of the instruction from being stored in said at least one cache memory in response to said opcode

2. The computer system of Claim 1, wherein said processor is to combine a plurality of non-temporal stores into a write transaction on the bus
3. The computer system of Claim 2, wherein said processor is to combine the plurality of non-temporal stores in response to each of said plurality of non-temporal stores being a non-temporal instruction
4. The computer system of Claim 1, wherein said processor determines if said operand address matches one of said plurality of addresses in said cache directory, if so, said processor updating a data entry in said cache controller corresponding to said matched address, otherwise updating a data entry corresponding to said operand address in said main memory
5. The computer system of Claim 1, wherein said instruction is a MASKMOVQ instruction which is a byte mask instruction to copy bytes indicated by the mask to the memory location, wherein said mask comprises a plurality of bits and wherein each single bit indicates whether a respective byte of the register is to be copied to the memory location.

6. The computer system of Claim 1, further comprising a fetching unit,
wherein said processor is to perform a plurality of non-temporal instructions in a
different order than fetched by the fetching unit in response to said plurality of non-
temporal instructions being non-temporal instructions

7. The computer system of Claim 6, further comprising:

a keyboard coupled to the bus;

a display coupled to the bus;

a sound unit coupled to the bus;

a modem coupled to the bus.

8. The computer system of Claim 5 wherein non-temporal data is treated
differently at different levels in a cache hierarchy within said processor, said cache

hierarchy comprising at least two cache memories

9. The computer system of Claim 5 further comprising a plurality of registers, wherein said execution unit is to execute a MOVNTPS move instruction to copy four packed floating point numbers from a first register in said plurality of registers to a first memory location and to prevent storing of said four packed floating point numbers in the at least one cache memory, and wherein said execution unit is to execute a MOVNTQ instruction to copy integer data from a second one of said plurality of registers to a second memory location and to prevent storing of said integer data to said at least one cache memory.

10. A method comprising:
receiving an instruction having an operand address;

selectively moving portions of data in a register to a memory location indicated by the operand address in response to said instruction based on a mask value;
preventing said memory location from being written to a cache memory in response to the instruction.

11. The method of Claim 10, further comprising:
determining if said operand address matches one of a plurality of addresses in a cache directory;
updating a data entry in a cache controller corresponding to said operand address if said operand address matches one of said plurality of addresses in said cache directory, otherwise updating a data entry corresponding to said operand address in a main memory.

12. The method of Claim 10
further comprising:

combining a plurality of non-temporal instructions into a single write transaction.

13. The method of Claim

12 wherein combining the plurality of non-temporal instructions into the single write transaction is to be done in response to each of the plurality of non-temporal instructions being a non-temporal instruction.

14. The method of Claim

13 further comprising:
completing a second plurality of non-temporal instructions in a different order than fetched.

15. The method of Claim 10, wherein said instruction is a MASKMOVB instruction which is a byte mask instruction to copy bytes indicated by the mask value from the register to the memory location, wherein said mask value comprises a plurality of bits and wherein each single bit indicates whether a respective byte of the register is to be copied to the memory location

16. The method of Claim 15, further comprising:

generating an output signal to one of a cache controller and a main memory to update a corresponding data entry in one of said cache controller and said main memory, if the operand address matches one of a plurality of addresses in a cache directory.

17. The method of Claim 15, further comprising:

executing a MOVNTPS move instruction to copy four packed floating point numbers from a first register to a first memory location;

preventing storing of said four packed floating point numbers to said cache memory;

executing a MOVNTQ instruction to copy integer data from a second register to a second memory location;

preventing storing of said integer data to said cache memory

18. The method of Claim 17, further comprising:

treating non-temporal data differently at different levels in a cache hierarchy

19. An apparatus comprising:

a cache memory to store a plurality of entries and a plurality of associated addresses;

a register;

an execution unit to execute a first instruction to perform a move operation using a

mask pattern, said mask pattern to indicate which portions of data from said

register are to be stored in a memory location, said first instruction to prevent data

moved by execution of said first instruction from being cached in said cache

memory.

20. The apparatus of claim 19 wherein data from said register is prevented from being cached in said cache memory at least if said memory location was not cached prior to execution of the first instruction.

21. The apparatus of claim 19 wherein said first instruction is a MASKMOVQ instruction, and wherein the register is a 64 bit integer register, and wherein said MASKMOVQ instruction is a byte mask instruction that copies bytes indicated by the mask pattern to the memory location without caching the bytes indicated by the mask pattern in the cache memory, and further wherein a single bit indicates whether a respective byte of the register is to be copied to memory, and further wherein a logical 0 in the mask pattern corresponds to no write (mask) and a logical 1 in the mask pattern corresponds to write (unmask).

22. The apparatus of claim 19 wherein said masked move instruction is a byte mask instruction that copies bytes indicated by the mask pattern to the memory location without caching the bytes in the cache memory.
23. The apparatus of claim 22 wherein said mask pattern comprises a plurality of bits, wherein a single bit indicates whether a respective byte of the register is to be copied to memory.
24. The apparatus of claim 23 wherein a logical 0 in the mask pattern corresponds to no write (mask) and a logical 1 in the mask pattern corresponds to write (unmask).
25. The apparatus of claim 24 wherein said apparatus is to store data from the register in the cache memory if the memory location is already cached in the cache memory and to otherwise not store data from the register in the cache memory and to not perform a cache fill in response to said first instruction.
26. The apparatus of claim 21 further comprising a second cache memory, wherein only one of the cache memory and the second cache memory is updated.
27. The apparatus of claim 24 wherein said apparatus is defined by data on a computer readable medium.

28. The apparatus of claim 19 wherein a cache line fill of a cache line in the cache memory corresponding to the memory location is prevented in response to the first instruction being a non-temporal instruction if said memory location is not cached in said cache memory.
29. The apparatus of claim 24 wherein said apparatus is a processor that supports an instruction set compatible with an Intel Architecture instruction set.
30. The apparatus of claim 24 wherein said execution unit is also to execute a first non-temporal move instruction and a second non-temporal move instruction, each storing a different type of data, wherein the apparatus, in response to either said first non-temporal move instruction or said second non-temporal move instruction is to prevent a cache line fill to a memory location specified by the first non-temporal move instruction or the second non-temporal move instruction.
31. The apparatus of claim 21 wherein said execution unit is further to execute a first non-temporal move instruction which is a MOVNTPS move instruction for copying four non-temporal packed floating-point data elements from the register to a first memory location, and wherein said execution unit is further to execute a second non-temporal move instruction which is a MOVNTQ move instruction for copying non-temporal integer data from the register to a second location, and further wherein both said MOVNTPS and said MOVNTQ move instructions prevent copied data from being cached.

32. The apparatus of claim 30 wherein said first non-temporal move instruction is a MOVNTPS move instruction for copying four non-temporal packed single precision floating-point data elements from the register to a first specified memory location, and wherein said second non-temporal move instruction is a MOVNTQ move instruction for copying non-temporal integer data from the register to a second specified memory location, and wherein said cache line fill is prevented in response to the MOVNTPS or the MOVNTQ move instruction being a non-temporal instruction.
33. The apparatus of claim 32 wherein said apparatus is to combine a plurality of instructions into a write transaction for issue onto a bus in response to each of the plurality of instructions being a non-temporal store instruction.
34. The apparatus of claim 19 wherein said apparatus is to combine a plurality of non-temporal stores into a write transaction for issue onto a bus.
35. The apparatus of claim 34 wherein said apparatus is to combine the plurality of non-temporal stores in response to each of the plurality of non-temporal stores having a property of write combining.
36. The apparatus of claim 35 wherein said plurality of non-temporal stores that are combined into the write transaction are stores to a single cache line.

37. The apparatus of claim 31 further comprising a fetching unit, wherein said apparatus is to perform a second plurality of non-temporal instructions in a different order than received by the fetching unit in response to said second plurality of non-temporal instructions being non-temporal instructions and therefore being weakly ordered instructions.
38. The apparatus of claim 19 wherein the first instruction has a property of write combining, wherein cacheable stores to a same cache line are combined before a write is issued.
39. The apparatus of claim 38 wherein non-temporal data is treated differently at different levels in a cache hierarchy comprising the cache memory and at least one other cache memory.
40. An apparatus comprising:
- a cache memory to store a plurality of entries and a plurality of associated addresses;
 - a register;
 - an execution unit to receive a first instruction having an opcode that indicates the first instruction is a non-temporal instruction, said first instruction indicating a store of a data item from the register to a memory location, said execution unit to prevent storing of the data item into the cache memory in response to said opcode, wherein a plurality of instructions, including the first instruction, are to be

combined into a write transaction to be issued to a bus in response to the plurality of instructions being a plurality of non-temporal store instructions.

41. The apparatus of claim 40 wherein said first instruction has an operand address, and wherein said cache memory is to update data written by the first instruction if said operand address of the first instruction is indicated by one of said plurality of associated addresses, and wherein said cache memory is, in response to said opcode indicating a non-temporal store, to not update data written by the first instruction if the operand address of the first instruction is not indicated by one of said plurality of associated addresses.
42. The apparatus of claim 40 wherein said plurality of non-temporal store instructions that are combined into the write transaction are to a single cache line.
43. The apparatus of claim 40 further comprising a fetching unit, wherein a plurality of non-temporal instructions received by the fetching unit are performed in a different order than received by the fetching unit.
44. The apparatus of claim 40 wherein only a single store is to be visible on an external bus after a plurality of successive non-temporal stores write to a single address.
45. The apparatus of claim 43 wherein each of the plurality of instructions has a property of write collapsing, allowing only a single store to be visible on an external bus after a plurality of successive non-temporal stores write to a single address.

46. The apparatus of claim 40 wherein the cache memory is to determine whether or not to update main memory in addition to updating said cache memory.
47. The apparatus of claim 45 wherein the cache memory is to determine whether or not to update main memory in addition to updating said cache memory.
48. The apparatus of claim 40 wherein non-temporal data is treated differently at different levels in a cache hierarchy comprising the cache memory and at least one other cache memory.
49. The apparatus of claim 47 wherein non-temporal data is treated differently at different levels in a cache hierarchy comprising the cache memory and at least one other cache memory.
50. The apparatus of claim 40 wherein said execution unit is to execute any non-temporal instructions chosen from a set consisting of:
- a move 64 bits non-temporal instruction;
 - a move four packed floating point instruction; and
 - a byte mask move instruction.
51. The apparatus of claim 49 wherein said apparatus is to process as non-temporal instructions an instruction set comprising:
- a move 64 bits non-temporal instruction;
 - a move four packed floating point instruction; and
 - a byte mask move instruction.

52. The apparatus of claim 50 wherein said byte mask move instruction uses a mask pattern to allow selective writing of portions of the register to the memory location.
53. The apparatus of claim 52 wherein portions of the register indicated by a logical '1' in the mask pattern are written to, and portions of the register indicated by a logical '0' in the mask pattern are masked.
54. The apparatus of claim 40 wherein said apparatus is defined by machine readable data on a machine readable medium.
55. The apparatus of claim 53 wherein said apparatus is defined by machine readable data on a machine readable medium.
56. A method comprising:
- receiving an instruction having an opcode that indicates a non-temporal instruction and an operand address which is an address at which an operand data from a register is to be stored in response to the instruction;
 - preventing said operand data from being written to a cache memory in response to the opcode;
 - combining a plurality of non-temporal stores, including the instruction, prior to issuing a store to a bus.
57. The method of claim 56 further comprising:
- if data associated with said operand address is already cached, then:
 - caching the operand data;

determining if the operand data is also to be written to a second memory and if
so writing the data;

if data associated with said operand address is not already cached, then writing the
operand data to the second memory;

58. The method of claim 57 further comprising:

performing a plurality of non-temporal store instructions in a different order than they
were fetched.

59. The method of claim 56 wherein combining the plurality of non-temporal writes
comprises combining a plurality of writes to a single cache line into a single write
transaction.

60. The method of claim 56 wherein combining the plurality of non-temporal writes
comprises collapsing a plurality of writes to a single address into a single write
transaction.

61. The method of claim 59 wherein combining the plurality of non-temporal writes
comprises combining the plurality of writes to the single cache line into a single write
transaction in response to each of the plurality of non-temporal writes being a non-
temporal instruction and therefore having a write combining property.

62. The method of claim 56 wherein said instruction is a mask move instruction and
wherein the method further comprises executing said mask move instruction by
selectively moving data based on a mask value.

63. The method of claim 62 wherein said instruction is a MASKMOVQ instruction

which is a byte mask instruction to copy bytes of the register indicated by the mask value to a memory location, wherein said mask value comprises a plurality of bits and wherein each single bit indicates whether a respective byte of the register is to be copied to the memory location.

64. A system comprising:

a main memory having a plurality of main memory addresses each having a

corresponding data entry;

a processor comprising a bus interface and coupled to said main memory;

a cache memory to store a plurality of entries and a plurality of associated addresses;

a register;

control logic to receive an instruction that is a non-temporal store instruction having

an opcode and an operand address, said control logic to prevent, in response to the

opcode, caching of data written from the register to a memory location in

accordance with said instruction, said control logic to combine a plurality of non-

temporal stores including said instruction into a write transaction issued on the

bus interface.

65. The system of claim 64 wherein said control logic is to cache data from said register if said operand address is indicated by one of said plurality of associated addresses, said control logic to not cache data from said register if the operand address is not indicated by one of said plurality of associated addresses in response to said opcode indicating a non-temporal store.

65. The system of claim 64 wherein the system is a single processor system.

66. The system of claim 64 wherein the instruction is a masked move instruction (MASKMOVQ), to perform a selective copying, based on a mask, of portions of data from the register.
67. The system of claim 64 wherein the instruction is a MOVNTPS instruction to move packed floating point data from the register.
68. The system of claim 64 wherein the instruction is a MOVNTQ instruction to move integer data from the register.
69. The system of claim 64 further comprising a plurality of registers, wherein said control logic is capable of executing a plurality of program instructions in a software routine in the main memory, said plurality of program instructions comprising:
- a MOVNTPS move instruction to copy four packed floating point numbers from a first register in said plurality of registers to a first memory location and to prevent storing of said four packed floating point numbers in the cache memory;
 - a MOVNTQ instruction to copy integer data from a second one of said plurality of registers to a second memory location and to prevent storing of said integer data to said cache memory; and
 - a MASKMOVQ instruction, to copy integer data from a 64 bit integer register, wherein said MASKMOVQ instruction is a byte mask instruction that is to copy bytes indicated by a mask pattern from a third one of said plurality of registers to a third memory location without caching the specified data in the cache memory, and further wherein a each of a plurality of single bits indicates whether respective bytes of the 64 bit integer register are to be

copied to memory, and further wherein a logical 0 in the mask pattern corresponds to no write (mask) and a logical 1 in the mask pattern corresponds to write (unmask).

70. The system of claim 69 wherein each of said plurality of program instructions has a property of write combining.
71. The system of claim 70 wherein each of said plurality of program instructions is a weakly ordered instruction.