

US Patent & Trademark Office

Patent Public Search | Text View

United States Patent Application Publication

20250265194

Kind Code

A1

Publication Date

August 21, 2025

Inventor(s)

Heremagalur Ramaprasad; Bipin Prasad et al.

PREFETCH MANAGEMENT IN A HIERARCHICAL CACHE SYSTEM

Abstract

An apparatus includes a CPU core, a first memory cache with a first line size, and a second memory cache having a second line size larger than the first line size. Each line of the second memory cache includes an upper half and a lower half. A memory controller subsystem is coupled to the CPU core and to the first and second memory caches. Upon a miss in the first memory cache for a first target address, the memory controller subsystem determines that the first target address resulting in the miss maps to the lower half of a line in the second memory cache, retrieves the entire line from the second memory cache, and returns the entire line from the second memory cache to the first memory cache.

Inventors: Heremagalur Ramaprasad; Bipin Prasad (Austin, TX), Thompson; David Matthew (Dallas, TX), Chachad; Abhijeet Ashok (Plano, TX), ONG; Hung (Plano, TX)

Applicant: Texas Instruments Incorporated (Dallas, TX)

Family ID: 1000008586696

Appl. No.: 19/199519

Filed: May 06, 2025

Related U.S. Application Data

parent US continuation 18102804 20230130 parent-grant-document US 12321277 child US 19199519

parent US continuation 17520805 20211108 parent-grant-document US 11567874 child US 18102804

parent US continuation 16856169 20200423 parent-grant-document US 11169924 child US 17520805

Publication Classification

Int. Cl.: G06F12/0862 (20160101); G06F9/38 (20180101); G06F12/0811 (20160101)

U.S. Cl.:

CPC G06F12/0862 (20130101); G06F9/3802 (20130101); G06F12/0811 (20130101);
G06F2212/602 (20130101)

Background/Summary

CROSS REFERENCE TO RELATED APPLICATIONS [0001] This application is a continuation of and claims priority to U.S. patent application Ser. No. 18/102,804, filed Jan. 30, 2023, which claims priority to U.S. patent application Ser. No. 17/520,805, filed Nov. 8, 2021, now U.S. Pat. No. 11,567,874, issued Jan. 31, 2023, which claims priority to U.S. Patent Application No. 16/856,169, filed Apr. 23, 2020, now U.S. Patent No. 11,169,924, issued Nov. 9, 2021, which claims priority to U.S. patent application Ser. No. 16/102,862, filed Aug. 14, 2018, now U.S. Pat. No. 10,642,742, issued May 5, 2020, all of which are hereby incorporated herein by reference in their entireties. [0002] This disclosure contains subject matter that may be related to the subject matter of commonly assigned U.S. patent application Ser. No. 16/102,931, filed Aug. 14, 2018, now U.S. Pat. No. 10,489,305, issued Nov. 26, 2019, U.S. patent application Ser. No. 16/694,751, filed Nov. 25, 2019, now U.S. Pat. No. 11,314,660, issued Apr. 26, 2022, U.S. patent application Ser. No. 17/727,921, filed Apr. 25, 2022, now U.S. Pat. No. 11,620,236, issued Apr. 4, 2023, U.S. patent application Ser. No. 18/194,708, filed Apr. 3, 2023, now U.S. Pat. No. 11,977,491, issued May 7, 2024, and U.S. patent application Ser. No. 18/630,098, filed Apr. 9, 2024, all of which are hereby incorporated herein by reference in their entireties.

BACKGROUND

[0003] Some memory systems include a multi-level cache system. Upon receipt from a processor core by a memory controller of a request for a particular memory address, the memory controller determines if data associated with the memory address is present in a first level cache (L1). If the data is present in the L1 cache, the data is returned from the L1 cache. If the data associated with the memory address is not present in the L1 cache, then the memory controller accesses a second level cache (L2) which may be larger and thus hold more data than the L1 cache. If the data is present in the L2 cache, the data is returned from the L2 cache to the processor core and a copy also is stored in the L1 cache in the event that the same data is again requested. Additional memory levels of the hierarchy are possible as well.

SUMMARY

[0004] In at least one example, an apparatus includes a central processing unit (CPU) core, a first memory cache to store instructions for execution by the CPU core, the first memory cache configured to have a first line size, and a second memory cache to store instructions for execution by the CPU core. The second memory cache has a second line size that is larger than the first line size, and each line of the second memory cache comprising an upper half and a lower half. A memory controller subsystem is coupled to the CPU core and to the first and second memory caches. Upon a miss in the first memory cache for a first target address, the memory controller subsystem determines that the first target address resulting in the miss maps to the lower half of a

line in the second memory cache, retrieves the entire line from the second memory cache, and returns the entire line from the second memory cache to the first memory cache.

Description

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] For a detailed description of various examples, reference will now be made to the accompanying drawings in which:

[0006] FIG. 1 illustrates a processor in accordance with an example.

[0007] FIG. 2 illustrates the promotion of an L1 memory cache access to a full L2 cache line access in accordance with an example.

[0008] FIG. 3 is a flow chart to illustrate a performance improvement in accordance with an example.

[0009] FIG. 4 is another flow chart to illustrate another performance improvement in accordance with an example.

[0010] FIG. 5 shows a system that includes the processor of FIG. 1.

DETAILED DESCRIPTION

[0011] FIG. 1 shows an example of a processor **100** that includes a hierarchical cache subsystem. The processor **100** in this example includes a central processing unit (CPU) core **102**, a memory controller subsystem **101**, an L1 data cache (L1D) **115**, an L1 program cache (L1P) **130**, and an L2 memory cache **155**. In this example, the memory controller subsystem **101** includes a data memory controller (DMC) **110**, a program memory controller (PMC) **120**, and a unified memory controller (UMC) **150**. In this example, at the L1 cache level, data and program instructions are divided into separate caches. Instructions to be executed by the CPU core **102** are stored in the L1P **130** to then be provided to the CPU core **102** for execution. Data, on the other hand, is stored in the L1D **115**. The CPU core **102** can read data from, or write data to, the L1D **115** but has read access to (no write access to) the L1P **130**. The L2 memory cache **155** can store both data and program instructions.

[0012] Although the sizes of the L1D **115**, L1P **130**, and L2 memory cache **155** can vary from implementation to implementation, in one example, the size of the L2 memory cache **155** is larger than the size of either the L1D **115** or the L1P **130**. For example, the size of the L1D **115** is 32 kbytes and the size of the L1P also is 32 kbytes, while the size of the L2 memory cache can be between 64 kbytes and 4 MB. Further, the cache line size of the L1D **115** is the same as the cache line size of the L2 memory cache **155** (e.g., 128 bytes), and the cache line size of the L1P **130** is smaller (e.g., 64 bytes).

[0013] Upon the need by the CPU core **102** for data, the DMC **110** receives an access request for the target data from the CPU core **102**. The access request may comprise an address (e.g., a virtual address) from the CPU core **102**. The DMC **110** determines whether the target data is present in the L1D **115**. If the data is present in the L1D **115**, the data is returned to the CPU core **102**. If, however, the data requested by the CPU core **102** is not present in the L1D **115**, the DMC **110** provides an access request to the UMC **150**. The access request may comprise a physical address that is generated by the DMC **110** based on the virtual address (VA) provided by the CPU core **102**. The UMC **150** determines whether the physical address provided by the DMC **110** is present in the L2 memory cache **155**. If the data is present in the L2 memory cache **155**, the data is returned to the CPU core **102** from the L2 memory cache **155** with a copy being stored in the L1D **115**. An additional hierarchy of the cache subsystem may be present as well. For example, an L3 memory cache or system memory may be available to be accessed. As such, if the data requested by the CPU core **102** is not present in either the L1D **115** or the L2 memory cache **155**, the data can be accessed in an additional cache level.

[0014] With regard to program instructions, when the CPU core **102** needs additional instructions to execute, the CPU core **102** provides a VA **103** to the PMC **120**. The PMC responds to the VA **103** provided by the CPU core **102** by initiating a work flow to return a prefetch packet **105** of program instructions back to the CPU core **102** for execution. Although the size of the prefetch packet can vary from implementation to implementation, in one example, the size of the prefetch packet equals the size of the cache line of the L1P **130**. If the L1P cache line size is, for example, 64 bytes, a prefetch packet returned to the CPU core **102** will also contain 64 bytes of program instructions.

[0015] The CPU core **102** also provides a prefetch count **104** to the PMC **120**. In some implementations, the prefetch count **104** is provided to the PMC **120** after the CPU core **102** provides the VA **103**. The prefetch count **104** indicates the number of prefetch units of program instructions following the prefetch unit starting at the VA **103**. For example, the CPU core **102** may provide a VA of **200h**. That VA is associated with a prefetch unit of 64 bytes that begins at virtual address **200h**. If the CPU core **102** wants the memory controller subsystem **101** to send additional instructions for execution following the prefetch unit associated with virtual address **200h**, the CPU core **102** submits a prefetch count with a value greater than 0. A prefetch count of 0 means that the CPU core **102** does not need any more prefetch units. A prefetch count of, for example, 6 means that the CPU core **102** requests an additional 6 prefetch units worth of instructions to be obtained and sent back to the CPU core **102** for execution. The return prefetch units are shown in FIG. **1** as prefetch packets **105**.

[0016] Referring still to the example of FIG. **1**, the PMC **120** includes a TAGRAM **121**, an address translator **122**, and a register **123**. The TAGRAM **121** includes a list of the virtual addresses whose contents (program instructions) have been cached to the L1P **130**. The address translator **122** translates virtual addresses to physical addresses (PAs). In one example, the address translator **122** generates the physical address directly from the virtual address. For example, the lower 12 bits of the VA may be used as the least significant 12 bits of the PA, with the most significant bits of the PA (above the lower 12 bits) being generated based on a set of tables configured in main memory prior to execution of the program. In this example, the L2 memory cache **155** is addressable using physical addresses, not virtual addresses. The register **123** stores a hit/miss indicator **124** from a TAGRAM **121** look-up, the physical address **125** generated by the address translator **122** and a valid bit **126** (also referred to herein as a status bit) to indicate whether the corresponding hit/miss indicator **124** and physical address **125** are valid or invalid.

[0017] Upon receipt of a VA **103** from the CPU **102**, the PMC **120** performs a TAGRAM **121** look-up to determine whether the L1P **130** includes program instructions associated with that virtual address. The result of the TAGRAM look-up is a hit or miss indicator **124**. A hit means that the VA is present in the L1P **130** and a miss means that the VA is not present in the L1P **130**. For an L1P **130** hit, the target prefetch unit is retrieved by the PMC **120** from the L1P **130** and returned as a prefetch packet **105** to the CPU core **102**.

[0018] For an L1P **130** miss, the PA (generated based on the VA) is provided by the PMC **120** to the UMC **150** as shown at **142**. A byte count **140** also is provided from the PMC **120** to the UMC **150**. The byte count indicates the number of bytes of the L2 memory cache **155** that is to be retrieved (if present) starting at the PA **142**. In one example, the byte count **140** is a multi-bit signal that encodes the number of bytes desired from the L2 memory cache **155**. In an example, the line size of the L2 memory cache is **128** bytes, and each line is divided into an upper half (64 bytes) and a lower half (64 bytes). The byte count **140** thus may encode the number 64 (if only the upper or lower half 64 bytes are needed from a given L2 memory cache line) or 128 (if an entire L2 memory cache line is needed). In another example, the byte count may be a single bit signal where one state (e.g., 1) implicitly encodes an entire L2 memory cache line and another state (e.g., 0) implicitly encodes half of an L2 memory cache line.

[0019] The UMC **150** also includes a TAGRAM **152**. The PA **142** received by the UMC **150** from the PMC **120** is used to perform a look-up into the TAGRAM **152** to determine whether the target

PA is a hit or a miss in the L2 memory cache **155**. If there is a hit in the L2 memory cache **155**, the target information, which may be one-half of the cache line or the entire cache line depending on the byte count **140**, the target information is returned to the CPU core **102** with a copy being stored in the L1P **130** from which the same program instructions will be provided to the CPU core **102** the next time that the CPU core **102** attempts to fetch the same program instructions.

[0020] In example of FIG. **1**, the CPU core **102** provides a VA **103** and a prefetch count **104** to the PMC **120**. The PMC **120** initiates the workflow to retrieve the prefetch packet from the L1P **130** or L2 memory cache **155** as describe above. Using the prefetch count **104** and the original VA **103**, the PMC **120** calculates additional virtual addresses and proceeds to retrieve prefetch packets corresponding to those calculated VAs from the L1P **130** or L2 memory cache **155**. For example, if the prefetch count is 2 and the VA **103** from the CPU core **102** is **200h**, the PMC **120** calculates the next two VAs as **240h** and **280h**, rather than the CPU core **102** providing each such VA to the PMC **120**.

[0021] FIG. **2** illustrates a specific example in which an optimization results in improved performance of the processor **100**. As noted above, the line width of L2 memory cache **155** is larger than the line width of L1P. In one example, the width of L1P is **64** bytes and the line width of L2 memory cache **155** is **128** bytes as shown in FIG. **2**. The L2 memory cache **155** is organized as an upper half **220** and a lower half **225**. The UMC **150** can read an entire **128** byte cache line from the L2 memory cache **155**, or only half (upper half **220** or lower half **225**) of an L2 memory cache line.

[0022] A given VA may translate into a particular PA that, if present in the L2 memory cache **155**, maps to the lower half **225** of a given line of the L2 memory cache or maps to the upper half **220**. Based on the addressing scheme used to represent VAs and PAs, the PMC **120** can determine whether a given VA would map to the lower half **225** or upper half **220**. For example, a particular bit within the VA (e.g., bit **6**) can be used to determine whether the corresponding PA would map to the upper or lower halves of the line of the L2memory cache. For example, bit **6** being a 0 may indicate the lower half and bit **6** being a 1 may indicate the upper half.

[0023] Reference numeral **202** shows an example of a VA of **200h** provided by the CPU core **102** to the PMC **120** and a corresponding prefetch count of 6. Reference numeral **210** illustrates that the list of VAs that are run through the cache pipeline described above include **200h** (received from the CPU core **102**) and the next 6 consecutive virtual address **240h**, **280h**, **2c0h**, **300h**, **340h**, and **380h** (calculated by the PMC **120**).

[0024] Each address from **200h** through **380h** is processed as described above. Any or all of the VAs may be a miss in the L1P **130**. The PMC **120** can package two consecutive VAs that miss in the L1P **130** into a single L2 cache line access attempt. That is, if **200h** and **240h** both miss in the L1P **130**, and the physical address corresponding to **200h** corresponds to the lower half **225** of a particular cache line of the L2 memory cache **155** and the physical address corresponding to **240h** corresponds to the upper half **225** of the same cache line of the L2 memory cache, the PMC **120** can issue a single PA **142** to the UMC **150** along with a byte count **140** specifying an entire cache line from the L2 memory cache. That is, two contiguous VA misses in L1P **130** can be promoted into a single full line L2 memory cache look-up.

[0025] If the last VA in a series of VAs initiated by the CPU core **102** (e.g., VA **380h** in VA series **210**) maps to the lower half **225** of a cache line of the L2 memory cache **155**, then in accordance with the described examples, the entire cache line of the L2 memory cache **155** is retrieved even though only the lower half **225** was needed. The same response occurs if the CPU provided a VA **103** to the PMC **120** with prefetch count of 0 meaning that the CPU **102** only wanted a single prefetch unit. Very little if any additional overhead, time, or power consumption is expended to retrieve the entire cache line and provide the entire cache line to the L1P **130**. Since program instructions are often executed in linear order, the probability is generally high that the program instructions in the upper half **220** would be executed following the execution of the instructions in the lower half **225** anyway. Thus, the next set of instructions are received at very little cost and

such instructions are likely to be needed anyway.

[0026] FIG. 2 illustrates through arrow **213** that VA **380h** maps to the lower half **225** of cache line **260** in the L2 memory cache **155**. The PMC **120** determines this mapping through an examination, for example, of one or more of the bits of the VA or its counterpart physical address following translation by the address translator **122**. The PMC **120** promotes the look-up process by the UMC **150** to a full cache line read by submitting the PA associated with VA **380h** along with a byte count **104** that specifies the entire cache line. The entire 128 byte cache line (if present in the L2 memory cache **155**) is then retrieved and written to the L1P **130** in two separate 64 byte cache lines as indicated at **265**.

[0027] If, however, the last VA in the series of VAs (or if there is only one VA for a prefetch count of 0) maps to the upper half **220** of a cache line of the L2 memory cache **155**, then the PMC **120** requests the UMC **150** to look-up in its TAGRAM **152** and return to the CPU core **102** and the L1P **130** only the upper half of the cache line. The next PA would be in the lower half **225** of the next cache line of the L2 memory cache **155** and additional time, overhead and power would be consumed to speculatively retrieve the next cache line, and it is not certain that the CPU core **102** would need to execute those instructions.

[0028] FIG. 3 shows an example of a flow chart **300** for the above-described method. The operations can be performed in the order shown, or in a different order. Further, the operations can be performed sequentially or two or more of the operations can be performed concurrently.

[0029] At **302**, the method includes receiving by the memory controller subsystem **101** an access request for N prefetch units of program instructions. In one implementation, this operation is performed by the CPU core **102** providing an address and count value to the PMC **120**. The address may be a virtual address or a physical address, and the count value may indicate the number of additional prefetch units that are needed by the CPU core **102**.

[0030] At **304**, an index value I is initialized to a value 1. This index value is used to determine when the last virtual address in a series of consecutive virtual addresses is to be processed by the PMC **120**. At **306**, the method determines whether prefetch unit I is a hit or a miss into the L1P **130**. This determination is made in some examples by determining if the virtual address is present in the PMC's TAGRAM **121**. Two results are possible from the determination **306**—a hit or a miss.

[0031] If the virtual address is a hit into the L1P **130**, then at **308**, the corresponding line of the L1P **130** which contains the desired prefetch unit is returned from the L1P **130** and provided back to the CPU core **102** as a prefetch packet **105**. The index is then incremented at **310** ($I=I+1$). If I has not yet reached N+1 (as determined at decision operation **312**), then the VA of the last of the prefetch units has not yet been evaluated for the hit/miss determination, and control loops back to **306** to evaluate the next Ith prefetch unit for a hit or miss in the L1P **130**. If I has reached N+1, then all N prefetch units have been evaluated and the corresponding program instructions have been supplied to the CPU core **102** and the process stops.

[0032] For a given Ith prefetch unit, if PMC **120** determines there to be a miss in the L1P **130** at **306**, then a determination is made at **314** as to whether I has reached the value of N. If I does not equal N (indicating that the last VA in a series of VAs has not been reached), then at **316**, the method includes the memory controller subsystem **101** obtaining program instructions from the L2 memory cache **155** (if present there, or from a third level cache or system memory if not present). The index value I is then incremented at **318** and control loops back to determination **306**.

[0033] If, at **314**, I has reached N (indicating that the last VA in the series of VAs has been reached), then at **320** the method includes determining whether the VA of the Ith prefetch unit maps to the lower half or the upper half of the cache line of the L2 memory cache **155**. An example of how this determination can be made is described above. If the VA of the Ith prefetch unit maps to the upper half, then at **322** the method includes obtaining the program instructions from only the upper half of the cache line of the L2 memory cache.

[0034] However, if the VA of Ith prefetch unit maps to the lower half, then at **324** the method

includes promoting the L2 memory cache access to a full cache line access and, at **326**, obtaining the program instructions from the full cache line of the L2 memory cache.

[0035] Referring back to FIG. 1, as described above, following submission from the CPU core **102** to the PMC **120** of a VA **103**, the CPU core **102** also can provide a prefetch count **104** to the PMC **120**. The prefetch count could be 0 meaning that the CPU core **102** does not need any more instructions other than those contained in the prefetch unit starting at the VA **103**. However, in between receipt of the VA **103** and the subsequent prefetch count, the PMC **120** has done some work as described below.

[0036] Upon receipt of the VA **103**, the PMC **120** performs a look-up in TAGRAM **121** to determine whether the first VA (provided by the CPU core **102**) is a hit or miss in L1P and also performs a VA to PA conversion using the address translator **122**. The PMC **120** also calculates a second VA (the next contiguous VA following the VA provided by the CPU core) before receiving the prefetch count **104**. The PMC **120** speculatively accesses the TAGRAM **121** and uses the address translator **122** to determine the hit/miss status of the second VA, and populates register **123** with the hit/miss indication **124** and the PA **125**. The valid bit **126** in the register **123** is set to a valid state to thereby permit further processing of the second VA as described above (e.g., retrieve the corresponding cache line from the L1P **130** if present, or from the L2 memory cache **155** if needed).

[0037] However, before any further processing of the second VA occurs, it is possible that the CPU core **102** sends a prefetch count of 0 to the PMC **120** meaning that the CPU core does not need any prefetch units besides the prefetch unit starting at the original VA **103**. At this point, the PMC **120** is provided a prefetch count of 0, and thus the prefetch unit associated with the second VA is not needed. However, the PMC also already determined the hit/miss status of the second VA and has generated the corresponding PA. Both the hit/miss indicator **124** and the PA **125** have been stored in the register **123** by the time the zero prefetch count has been received by the PMC **120**. The PMC **120** changes the status of the valid bit **126** to indicate an invalid state to thereby preclude any further processing of the second VA. This condition (valid bit set to an invalid state) is referred to as a “kill”, that is, the PMC **120** kills the processing of the second VA.

[0038] In some situations, however, the CPU core **102** may determine, despite the previous kill, that the prefetch unit associated with the second VA should indeed be obtained from the L1P **130** or L2 memory cache **155** as described above. For example, if the CPU core **102** has no further internal prediction information to inform the next required instruction address, the CPU core **102** will signal to the PMC **120** that it should continue prefetching linearly starting from the last requested address. This condition may occur, for example, due to a misprediction in the branch prediction logic in the CPU core **102**. The CPU core **102** thus issues a revive signal **106** to the PMC **120** for this purpose. The PMC **120** responds to the revive signal by changing the valid bit **126** back to the valid state to thereby permit the continued processing of the second VA through the memory subsystem pipeline as described above. As such, the CPU **102** need not submit the second VA directly to the PMC **120**. Instead, the PMC **120** retains the second VA in, for example, register **123** as well as its hit/miss indicator **124** thereby avoiding the power consumption and time spent to again determine the hit/miss status of the second VA and translate the second VA to a PA.

[0039] FIG. 4 shows an example of a flow chart **400** for initiating, then killing, and then reviving a memory address look-up. The operations can be performed in the order shown, or in a different order. Further, the operations can be performed sequentially or two or more of the operations can be performed concurrently.

[0040] At **402**, the method includes receiving by the memory controller subsystem **101** an access request at a first VA. In one implementation, this operation is performed by the CPU core **102** providing the first VA to the PMC **120**. At **404**, the method includes determining if the first VA is a hit or a miss in the L1P **30**. In one example, this operation is performed by accessing the PMC's TAGRAM **121** to determine the hit/miss condition of the first VA. The first VA is translated to a

first PA at **406** by using, for example, the address translator **122**.

[0041] At **408**, the method includes computing a second VA based on the first VA. The second VA may be computed by incrementing the first VA by a value to generate an address of a byte that is 64 bytes following the byte associated with the first VA. At **410**, the method includes determining if the second VA is a hit or a miss in the L1P **30**. In one example, this operation is performed by accessing the PMC's TAGRAM **121** to determine the hit/miss condition of the second VA. The second VA is translated to a second PA at **412** by using the address translator **122** as described above. At **414**, the method includes updating a register (e.g., register **123**) with the hit/miss indicator **124** and the second PA. Further, the valid bit **126** is configured to be a valid state.

[0042] The PMC **120** then receives a prefetch count at **416**. Then, at **418**, if the prefetch count is greater than zero, then at **420**, the program instructions from the L1P **130** or the L2 memory cache **155** (or additional level(s)) are retrieved as described above. However, if the prefetch count is zero, then at **422**, the valid bit **126** is changed to the invalid state. Despite having provided a prefetch count of zero to the PMC **120**, the CPU core **102** may then provide a revival indication to the PMC **120** (**424**). At **426**, the PMC **120** changes the valid bit **126** back to the valid state and memory controller subsystem **101** then obtains the program instructions associated with the second PA from the L1P, L2 memory caches, etc. as appropriate (**428**).

[0043] FIG. 5 shows an example use of the processor **100** described herein. In this example, the processor **100** is part of a system-on-chip (SoC) **500** that includes the processor **100** and one or more peripheral ports or devices. In this example, the peripherals include a universal asynchronous receiver transmitter (UART) **502**, a universal serial bus (USB) port **504**, and an Ethernet controller **506**. The SoC **500** can perform any of a variety of functions as implemented by, for example, the program instructions executed by the processor **100**. More than one processor **100** may be provided and, within a given processor **100**, more than one CPU core **102** may be included.

[0044] In this description, the term “couple” or “couples” means either an indirect or direct wired or wireless connection. Thus, if a first device couples to a second device, that connection may be through a direct connection or through an indirect connection via other devices and connections. The recitation “based on” means “based at least in part on.” Therefore, if X is based on Y, X may be a function of Y and any number of other factors.

[0045] Modifications are possible in the described embodiments, and other embodiments are possible, within the scope of the claims.

Claims

1. A device, comprising: a cache memory including a cache line that includes a first half portion and a second half portion; and a cache controller coupled to the cache memory and capable of: receiving a first request for a first program instruction; determining whether to generate a second request for the first program instruction or generate a second request for the first program instruction and a second program instruction, based on whether the first program instruction is stored in the first half portion or in the second half portion of the cache line of the cache memory; and providing the second request to the cache memory.
2. The device of claim 1, wherein the first request includes a virtual address.
3. The device of claim 2, wherein the second request includes a physical address.
4. The device of claim 3, wherein the second request includes a byte count that indicates whether the cache memory is to provide the second program instruction.
5. The device of claim 1, wherein: the first half portion is an upper portion of the cache line of the cache memory; and the second half portion is a lower portion of the cache line of the cache memory.
6. The device of claim 5, wherein the cache controller is capable of: generating the second request for the first program instruction based on that the first program instruction is stored in the upper

portion of the cache line; and generating the second request for the first program instruction and the second program instruction based on that the first program instruction is stored in the lower portion of the cache line and the second program instruction is stored in the upper portion of the cache line.

7. The device of claim 1, wherein: the cache memory is a level two (L2) cache memory.

8. The device of claim 7, wherein: the device further comprises a level one (L1) cache memory; and the cache line of the L2 cache memory is twice the size of a cache line of the L1 cache memory.

9. The device of claim 8, wherein the cache controller is capable of: prior to determining whether to generate the second request for the first program instruction or generate the second request for the first program instruction and the second program instruction, determining that the first request corresponds to a miss in the L1 cache memory.

10. A device, comprising: a processing core; a level one (L1) cache memory; a level two (L2) cache memory including a cache line that includes a first half portion and a second half portion; and a controller capable of: receiving, from the processing core, a first request for a first program instruction; determining whether the first request corresponds to a miss in the L1 cache memory; and based on determining that the first request corresponds to a miss in the L1 cache memory, determining whether to generate a second request for the first program instruction or generate a second request for the first program instruction and a second program instruction, based on whether the first program instruction is stored in the first half portion or in the second half portion of the cache line of the L2 cache memory; and providing the second request to the L2 cache memory.

11. The device of claim 10, wherein the first request includes a virtual address.

12. The device of claim 11, wherein the second request includes a physical address.

13. The device of claim 12, wherein the physical address is a translated address of the virtual address.

14. The device of claim 10, wherein: the first half portion is an upper portion of the cache line of the L2 cache memory; and the second half portion is a lower portion of the cache line of the L2 cache memory.

15. The device of claim 14, wherein the controller is capable of: generating the second request for the first program instruction based on that the first program instruction is stored in the upper portion of the cache line; and generating the second request for the first program instruction and the second program instruction based on that the first program instruction is stored in the lower portion of the cache line and the second program instruction is stored in the upper portion of the cache line.

16. The device of claim 10, wherein: the cache line of the L2 cache memory is twice the size of a cache line of the L1 cache.

17. A method, comprising: receiving, by a cache controller, a first request for a first program instruction; determining, by the cache controller, whether the first program instruction is stored in a first half portion or in a second half portion of a cache line of a cache memory; determining, by the cache controller, whether to generate a second request for the first program instruction or generate a second request for the first program instruction and a second program instruction, based on whether the first program instruction is stored in the first half portion or in the second half portion of the cache line; and providing, by the cache controller, the second request to the cache memory.

18. The method of claim 17, wherein: the first half portion is an upper portion of the cache line of the cache memory; and the second half portion is a lower portion of the cache line of the cache memory.

19. The method of claim 18, wherein determining whether to generate the second request for the first program instruction or generate the second request for the first program instruction and the second program instruction comprises: generating the second request for the first program instruction based on that the first program instruction is stored in the upper portion of the cache line; and generating the second request for the first program instruction and the second program instruction based on that the first program instruction is stored in the lower portion of the cache line

and the second program instruction is stored in the upper portion of the cache line.

20. The method of claim 17, wherein the cache memory is a level two (L2) cache memory, and wherein the method further comprises: prior to determining whether to generate the second request for the first program instruction or generate the second request for the first program instruction and the second program instruction, determining that the first request corresponds to a miss in a level one (L1) cache memory.
