



(51) International Patent Classification:  
*G06F 13/16* (2006.01) *G06F 12/0806* (2016.01)

(21) International Application Number:  
PCT/US2017/031182

(22) International Filing Date:  
05 May 2017 (05.05.2017)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:  
15/180,351 13 June 2016 (13.06.2016) US

(71) Applicant: INTEL CORPORATION [US/US]; 2200 Mission College Boulevard, Santa Clara, California 95054 (US).

(72) Inventors: SURY, Samantika S.; 11 Blueberry Lane, Westford, Massachusetts 01886 (US). BLANKENSHIP, Robert G.; 54 N. Salmon Beach, Tacoma, Washington 98407 (US). STEELY, JR., Simon C.; 8 Anna Louise Drive, Hudson, New Hampshire 03051 (US).

(74) Agent: GARZA, John C. et al.; Trop, Pruner & Hu, P.C., 1616 S. Voss Rd., Ste. 750, Houston, Texas 77057-2631 (US).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM,

(54) Title: SYNCHRONIZATION LOGIC FOR MEMORY REQUESTS

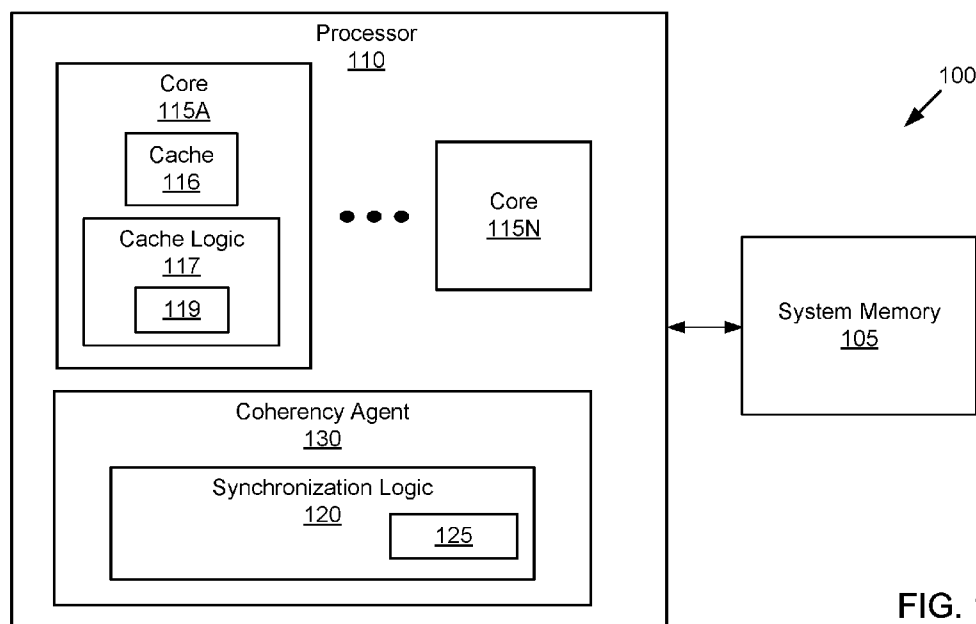


FIG. 1A

(57) Abstract: In an embodiment, a processor includes a plurality of cores and synchronization logic. The synchronization logic includes circuitry to: receive a first memory request and a second memory request; determine whether the second memory request is in contention with the first memory request; and in response to a determination that the second memory request is in contention with the first memory request, process the second memory request using a non-blocking cache coherence protocol. Other embodiments are described and claimed.

TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW,  
KM, ML, MR, NE, SN, TD, TG).

**Declarations under Rule 4.17:**

- *as to the identity of the inventor (Rule 4.17(i))*
- *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*
- *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

**Published:**

- *with international search report (Art. 21(3))*

## SYNCHRONIZATION LOGIC FOR MEMORY REQUESTS

### Field of Invention

[0001] Embodiments relate generally to computer processors. More particularly, embodiments are related to computer processors including functionality to process parallel threads.

### Background

[0002] Modern computer processors may include functionality to process information as parallel threads. For example, a processor may be implemented as a single physical processor die that includes multiple processing elements or "cores." These multiple processing elements can execute multiple threads in parallel. In some situations, processing threads in parallel may reduce the overall execution time of a program.

### Brief Description of the Drawings

[0003] FIG. 1A is a block diagram of a system in accordance with one or more embodiments.

[0004] FIG. 1B show an illustration of an example operation in accordance with one or more embodiments.

[0005] FIGs. 2A-2C are sequences in accordance with one or more embodiments.

[0006] FIG. 3A is a block diagram of a portion of a system in accordance with one or more embodiments.

[0007] FIG. 3B is a block diagram of a multi-domain processor in accordance with one or more embodiments.

[0008] FIG. 3C is a block diagram of a processor in accordance with one or more embodiments.

[0009] FIG. 4 is a block diagram of a processor including multiple cores in accordance with one or more embodiments.

[0010] FIG. 5 is a block diagram of a micro-architecture of a processor core in accordance with one or more embodiments.

[0011] FIG. 6 is a block diagram of a micro-architecture of a processor core in accordance with one or more embodiments.

[0012] FIG. 7 is a block diagram of a micro-architecture of a processor core in accordance with one or more embodiments.

[0013] FIG. 8 is a block diagram of a micro-architecture of a processor core in accordance with one or more embodiments.

[0014] FIG. 9 is a block diagram of a processor in accordance with one or more embodiments.

[0015] FIG. 10 is a block diagram of a representative SoC in accordance with one or more embodiments.

[0016] FIG. 11 is a block diagram of another example SoC in accordance with one or more embodiments.

[0017] FIG. 12 is a block diagram of an example system with which one or more embodiments can be used.

[0018] FIG. 13 is a block diagram of another example system with which one or more embodiments may be used.

[0019] FIG. 14 is a block diagram of a computer system in accordance with one or more embodiments.

[0020] FIG. 15 is a block diagram of a system in accordance with one or more embodiments.

### Detailed Description

[0021] Some processors may include multiple processing cores, with each core (or a subset of cores) having a dedicated cache memory (e.g., a first-level or "L1" cache). Each cache may store a copy of information that is located in system memory. However, a cache coherence problem may arise when multiple caches include a copy of the same memory location. For example, if two different threads attempt to simultaneously modify the same data element, two conflicting versions of the data element may be generated, thus resulting in a program error.

[0022] One technique for maintaining cache coherency is to use blocking cache coherence protocols. In such protocols, each memory request is processed in series, with only one core having access to the desired memory address at a given time. Further, the processing of each memory request may include multiple messages (e.g., to invalidate data in a first cache, to give permission to write to a second cache, etc.). However, when there are multiple requests for the same data, the processing of these requests in series may result in a processing bottleneck.

[0023] In accordance with some embodiments, a processor may include synchronization logic to manage cache coherence for multiple cores. The synchronization logic may be cache coherence logic that can determine whether first and second requests are in contention (i.e., are attempting to simultaneously access the same memory address at the same time). If the first and second requests are not in contention, the synchronization logic may process the requests using a blocking cache coherence protocol. However, if the first and second requests are in contention, the synchronization logic may process the requests using a non-blocking cache coherence protocol. Using the non-blocking cache coherence protocol may allow the requests to be processed in a temporary pipeline, such that at least some messages associated with the second request are sent before completing the processing of the first request. Thus, the synchronization logic may temporarily process requests in parallel if they are in contention. Accordingly, in some embodiments, the overall execution time of the program may be reduced.

[0024] Although the following embodiments are described with reference to particular implementations, embodiments are not limited in this regard. In particular, it is contemplated that similar techniques and teachings of embodiments described herein may be applied to other types of circuits, semiconductor devices, processors, systems, etc. For example, the disclosed embodiments may be implemented in any type of computer system, including server computers (e.g., tower, rack, blade, micro-server and so forth), communications systems, storage systems, desktop computers of any configuration, laptop, notebook, and tablet computers (including 2:1 tablets, phablets and so forth).

[0025] In addition, disclosed embodiments can also be used in other devices, such as handheld devices, systems on chip (SoCs), and embedded applications. Some examples of handheld devices include cellular phones such as smartphones, Internet protocol devices,

digital cameras, personal digital assistants (PDAs), and handheld PCs. Embedded applications may typically include a microcontroller, a digital signal processor (DSP), network computers (NetPC), set-top boxes, network hubs, wide area network (WAN) switches, wearable devices, or any other system that can perform the functions and operations taught below. Further, embodiments may be implemented in mobile terminals having standard voice functionality such as mobile phones, smartphones and phablets, and/or in non-mobile terminals without a standard wireless voice function communication capability, such as many wearables, tablets, notebooks, desktops, micro-servers, servers and so forth.

[0026] Referring now to FIG. 1A, shown is a block diagram of a system 100 in accordance with one or more embodiments. In some embodiments, the system 100 may be all or a portion of an electronic device or component. For example, the system 100 may be a cellular telephone, a computer, a server, a network device, a system on a chip (SoC), a controller, a wireless transceiver, a power supply unit, etc. Furthermore, in some embodiments, the system 100 may be any grouping of related or interconnected devices, such as a datacenter, a computing cluster, etc.

[0027] As shown in FIG. 1A, the system 100 may include a processor 110 operatively coupled to system memory 105. Further, although not shown in FIG 1A, the system 100 may include other components. In one or more embodiments, the system memory 105 can be implemented with any type(s) of computer memory (e.g., dynamic random access memory (DRAM), static random-access memory (SRAM), non-volatile memory (NVM), a combination of DRAM and NVM, etc.).

[0028] The processor 110 may be a general purpose hardware processor (e.g., a central processing unit (CPU)). As shown, the processor 110 can include a coherency agent 130 and any number of processing cores 115A-115N (referred to collectively as "cores 115"). Each core 115 may be a general purpose processing core. In one or more embodiments, each core 115 may include a cache 116 and cache logic 117. The cache logic 117 may include a memory buffer 119. The cache 116 and/or cache logic 117 may be implemented in circuitry and/or micro-architecture of the core 115. Note that, while FIG. 1A only shows the cache 116 and cache logic 117 as included in core 115A, it is contemplated that any or all cores 115A-115N may include the same components.

[0029] In some embodiments, the coherency agent 130 may manage cache coherency across the cores 115A-115N. For example, the coherency agent 130 may control access to copies of data stored in each cache 116 of the cores 115A-115N. In some embodiments, the coherency agent 130 may determine an order for contending memory requests (e.g., based on arrival time, priority, etc.). As shown in FIG. 1A, the coherency agent 130 may include synchronization logic 120. The coherency agent 130 and/or the synchronization logic 120 may be implemented in circuitry and/or micro-architecture of the processor 110. In some embodiments, the synchronization logic 120 may be external to each core 115 (i.e., in the uncore region of the processor 110).

[0030] In one or more embodiments, the synchronization logic 120 is cache coherence logic that can determine whether memory requests are processed using blocking or non-blocking cache coherence protocols. In some embodiments, the synchronization logic 120 may select a blocking cache coherence protocol or a non-blocking cache coherence protocol based on whether the memory requests are in contention. For example, the synchronization logic 120 may determine that two memory requests are in contention, and may thus process the memory requests using a non-blocking cache coherence protocol. As used herein, the term "blocking protocol" may be used to refer to a blocking cache coherence protocol. Further, the term "non-blocking protocol" may be used to refer to a non-blocking cache coherence protocol.

[0031] In some embodiments, the synchronization logic 120 may use a non-blocking protocol to form a temporary pipeline for processing memory requests in parallel. For example, using a non-blocking protocol to process a memory request may include sending one or more messages before a contending memory request that was ordered earlier is completed. In contrast, using a blocking protocol to process the memory request includes waiting until the contending memory request that was ordered earlier is completed.

[0032] In some embodiments, both the blocking protocol and the non-blocking protocol may include a predefined sequence of messages between the coherency agent 130 and the cores 115. For example, the sequence of messages may include a request message sent from a requesting core 115A to the coherency agent 130 (e.g., a request for ownership (RFO) message). The sequence of messages may also include an invalidation message sent from the coherency agent 130 to a second core 115N storing the requested data (e.g., an "Inv"

message), a data response from the second core 115N to the coherency agent 130, and an acknowledgement message from the coherency agent 130 to the requesting core 115A (e.g., a global observability (GO) message).

[0033] In some embodiments, the invalidation message used in the non-blocking protocol (referred to herein as a "non-blocking-protocol invalidation message") may be of a different type than the invalidation message used in the blocking protocol (referred to herein as a "blocking-protocol invalidation message"). For example, the non-blocking-protocol invalidation message may have a different identifier, metadata, format, fields, and/or parameter values than the blocking-protocol invalidation message. In some embodiments, the cache logic 117 in a core 115 may recognize a non-blocking-protocol invalidation message received from the coherency agent 130.

[0034] In some embodiments, the acknowledgement message used in the non-blocking protocol (referred to herein as a "non-blocking-protocol acknowledgement message") may be of a different type than the acknowledgement message used in the blocking protocol (referred to herein as a "blocking-protocol acknowledgement message"). For example, the non-blocking-protocol acknowledgement message may have a different identifier, metadata, format, fields, and/or parameter values than the blocking-protocol acknowledgement message. In some embodiments, the cache logic 117 in a core 115 may recognize a non-blocking-protocol acknowledgement message received from the coherency agent 130.

[0035] In some embodiments, the cache logic 117 may wait to receive both a non-blocking-protocol invalidation message and a non-blocking-protocol acknowledgement message before performing the corresponding protocol actions. Thus, in some embodiments, the cache logic 117 may store the last received non-blocking-protocol invalidation message in the memory buffer 119. The cache logic 117 may also store the last received non-blocking-protocol acknowledgement message in the memory buffer 119. Once both of these messages have been received, the cache logic 117 may perform the corresponding protocol actions.

[0036] In some embodiments, the memory buffer 119 may be implemented by extending an entry of an existing data structure storing cache requests to the uncore region (e.g., a "super queue"). In one or more embodiments, each entry may be extended to include one or more

fields describing a message (e.g., a valid bit, a snoop type, an identifier, a flag indicating the use of a non-blocking-protocol, and so forth).

[0037] In one or more embodiments, the synchronization logic 120 may include a memory buffer 125 to store information about any non-blocking-protocol messages that are being used in a temporary pipeline for parallel memory requests. The synchronization logic 120 may use the memory buffer 125 to track non-blocking-protocol messages that have been sent to the cores 115, and to determine the current state of the parallel memory requests in the temporary pipeline. For example, the memory buffer 125 may include information such as the order of memory requests, messages sent to and received from each core, which core currently has permission to modify the data, and so forth. In some embodiments, the memory buffer 125 may be implemented by extending an entry of an existing data structure in the uncore region. For example, in some embodiments, the memory buffer 125 may be implemented by extending a directory used for blocking-protocol messages. In other embodiments, the memory buffer 125 may be separate from a directory used for blocking-protocol messages.

[0038] In one or more embodiments, the synchronization logic 120 may limit the use of non-blocking-protocol messages to a number of slots in the memory buffer 125. For example, when the memory buffer 125 is full, the synchronization logic 120 may use blocking-protocol messages for all memory requests, without regard to whether such memory requests are in contention.

[0039] In some embodiments, the synchronization logic 120 may limit the use of non-blocking-protocol messages based on a user-configurable setting. For example, a user-configurable setting or parameter may cause the synchronization logic 120 to use only blocking-protocol messages, to use only non-blocking-protocol messages, to use only a maximum number of non-blocking-protocol messages, and so forth.

[0040] In some embodiments, the synchronization logic 120 may limit the use of non-blocking-protocol messages based on operating conditions. For example, the synchronization logic 120 may disable or reduce the use of non-blocking-protocol messages during periods of starvation, low contention, etc.

[0041] Referring now to FIG. 1B, shown is an illustration of an example operation, in accordance with one or more embodiments. Specifically, FIG. 1B illustrates various

messages that are sent between four cores A-D and a coherency agent (CA) over time periods 1-8.

[0042] As shown, at time period 1, the CA receives memory requests A1, B1, C1 from cores A, B, C, respectively. Assume that request A1 is ordered first, and that request C1 is ordered last. Assume also that the CA determines that these requests are in contention for the same data, and that the requested data is located in core D. In response to receiving these three memory requests, the CA may perform three corresponding sequences of messages, namely a first sequence A2-A4, a second sequence B2-B4, and a third sequence C2-C4. Initially, at time period 1, the CA sends an invalidation message A2 to core D.

[0043] At time period 2, in response to receiving the invalidation message A2, core D sends a data response A3. In some embodiments, the data response A3 may include the requested data, and may acknowledge that core D has invalidated its own copy of the requested data (i.e., in a cache of core D).

[0044] Also at time period 2, the CA sends a non-blocking-protocol invalidation message B2 to core A, where the invalidation message B2 is associated with the request B1 received from core B. Note that the CA does not wait until the first sequence (associated with request A1) is completed before sending the invalidation message B2.

[0045] At time period 3, core A receives the invalidation message B2 from the CA. However, core A is not yet ready to process the invalidation message B2, and thus stores the invalidation message B2 (e.g., in memory buffer 119 shown in FIG. 1A). Note that core A has not yet received the requested data, and has not performed any action on the requested data (e.g., read, modify, etc.).

[0046] Also at time period 3, the CA receives the data response A3 from core D. In response to receiving the data response A3, the CA sends a non-blocking-protocol acknowledgement message A4 to core A. In some embodiments, the non-blocking-protocol acknowledgement message A4 includes the requested data, and acknowledges that core A has permission to modify the requested data.

[0047] At time period 4, core A receives the acknowledgement message A4 from the CA. Note that core A has now received the invalidation message B2 and the acknowledgement message A4, and that both of these messages are in a non-blocking cache coherence protocol.

Thus, because both of the non-blocking-protocol messages have been received, core A may process the data included in the acknowledgement message A4, and may include the processed data in a data response B3 sent to the CA. Further, once the data response B3 is sent, core A may invalidate its own copy of the data in response to the invalidation message B2 (received at time period 3).

[0048] Also at time period 4, the CA sends a non-blocking-protocol invalidation message C2 to core B, where the invalidation message C2 is associated with the request C1 received from core B. Note that the CA does not wait until the second sequence (associated with request B1) is completed before sending the invalidation message C2.

[0049] At time period 5, core B receives the invalidation message C2 from the CA. However, core B is not yet ready to process the invalidation message C2, and thus stores the invalidation message C2.

[0050] Also at time period 5, the CA receives the data response B3 from core A. In response to receiving the data response B3, the CA sends a non-blocking-protocol acknowledgement message B4 to core B. The acknowledgement message B4 may include the requested data, and may acknowledge that core B has permission to modify the requested data.

[0051] At time period 6, core B receives the acknowledgement message B4 from the CA. Note that core B has now received the invalidation message C2 and the acknowledgement message B4, and that both of these messages are in a non-blocking cache coherence protocol. Thus, core B may process the data included in the acknowledgement message B4, and may include this data in a data response C3 sent to the CA. Further, once the data response C3 is sent, core B may invalidate its own copy of the data in response to the invalidation message C2 (received at time period 5).

[0052] At time period 7, the CA receives the data response C3 from core B. The CA may determine that there is no longer any contention for the requested data. Thus, in response to receiving the data response C3, the CA may send a blocking-protocol acknowledgement message C4 to core C. The acknowledgement message C4 may include the requested data, and may acknowledge that core C has permission to modify the requested data.

[0053] At time period 8, core C receives the acknowledgement message C4 from the CA. Note that the acknowledgement message C4 is in a blocking protocol. Thus, core C may proceed to process the data in the acknowledgement message C4 without waiting for any other messages.

[0054] Referring now to FIG. 2A, shown is a sequence 200 in accordance with one or more embodiments. In some embodiments, all or a portion of the sequence 200 may be implemented in the coherency agent 130 and/or the synchronization logic 120 shown in FIG. 1A. The sequence 200 may be implemented in hardware, software, and/or firmware. In hardware embodiments it may be implemented as circuitry and/or micro-architecture. Further, in firmware and software embodiments, it may be implemented by computer executed instructions stored in a non-transitory machine readable medium, such as an optical, semiconductor, or magnetic storage device. The machine readable medium may store data, which if used by at least one machine, causes the at least one machine to fabricate at least one integrated circuit to perform a method. For the sake of illustration, the steps involved in the sequence 200 may be described below with reference to FIGs. 1A-1B, which show examples in accordance with some embodiments. However, the scope of the various embodiments discussed herein is not limited in this regard.

[0055] At block 210, a first memory request and a second memory request may be received. For example, referring to FIG. 1B, the coherency agent (CA) receives the memory request A1 from core A, and receives the memory request B1 from core B.

[0056] At block 214, a determination is made about whether the first and second memory requests are in contention. For example, referring to FIG. 1A, the synchronization logic 120 may determine whether the first and second memory requests are simultaneously requesting the same memory address.

[0057] If it is determined at block 214 that the first and second memory requests are in contention, then at block 217, the first and second memory requests may be processed using a non-blocking cache coherence protocol. For example, referring to FIG. 1A, the synchronization logic 120 may process each request in series using a non-blocking cache coherence protocol. In some embodiments, the synchronization logic 120 may use the non-blocking cache coherence protocol to establish a temporary pipeline for processing the first

and second memory requests in parallel. Various aspects in the operation of block 217 are described further below with reference to FIG. 2B

[0058] However, if it is determined at block 214 that the first and second memory requests are not in contention, then at block 220, the first and second memory requests may be processed using a blocking cache coherence protocol. For example, referring to FIG. 1A, the synchronization logic 120 may process each request in series using a blocking cache coherence protocol. As such, the synchronization logic 120 may not process the second request until a sequence of messages associated with the first request are completed. After either block 214 or block 220, the sequence 200 terminates.

[0059] Referring now to FIG. 2B, shown is a sequence 230 in accordance with one or more embodiments. The sequence 230 illustrates an example of actions involved in processing contending requests using a non-blocking cache coherence protocol. As such, the sequence 230 may correspond generally to actions involved in (or related to) block 217 (shown in FIG. 2A).

[0060] In some embodiments, all or a portion of the sequence 230 may be implemented in the coherency agent 130 and/or the synchronization logic 120 shown in FIG. 1A. The sequence 230 may be implemented in hardware, software, and/or firmware. In hardware embodiments it may be implemented as circuitry and/or micro-architecture. Further, in firmware and software embodiments, it may be implemented by computer executed instructions stored in a non-transitory machine readable medium, such as an optical, semiconductor, or magnetic storage device. The machine readable medium may store data, which if used by at least one machine, causes the at least one machine to fabricate at least one integrated circuit to perform a method. For the sake of illustration, the steps involved in the sequence 230 may be described below with reference to FIGs. 1A-1B, which show examples in accordance with some embodiments. However, the scope of the various embodiments discussed herein is not limited in this regard.

[0061] At block 235, it may be determined that a first memory request from a first core and a second memory request from a second core are in contention. For example, referring to FIG. 1B, the CA may determine that memory request A1 from core A and memory request B1 from core B are requesting the same memory address.

[0062] At block 240, an invalidation message associated with the first request may be sent to a third core. For example, referring to FIG. 1B, the CA sends the invalidation request A2 to core D.

[0063] At block 244, a non-blocking-protocol invalidation message associated with the second request may be sent to the first core without waiting for a data response from the third core. For example, referring to FIG. 1B, the CA sends the non-blocking-protocol invalidation message B2 to core A.

[0064] At block 247, a data response associated with the first request may be received from the third core. For example, referring to FIG. 1B, core D sends the data response A3 to the CA.

[0065] At block 250, in response to the data response from the third core, a non-blocking-protocol acknowledgement message associated with the first request may be sent to the first core. For example, referring to FIG. 1B, in response to receiving the data response A3, the CA sends a non-blocking-protocol acknowledgement message A4 to core A.

[0066] At block 254, a data response associated with the second request may be received from the first core. For example, referring to FIG. 1B, core A sends the data response B3 to the CA.

[0067] At block 257, in response to the data response from the first core, a non-blocking-protocol acknowledgement message associated with the first request may be sent to the second core. For example, referring to FIG. 1B, in response to receiving the data response B3, the CA sends a non-blocking-protocol acknowledgement message B4 to core B. After block 257, the sequence 230 terminates.

[0068] Referring now to FIG. 2C, shown is a sequence 260 in accordance with one or more embodiments. In some embodiments, all or a portion of the sequence 260 may be implemented in the cache logic 117 in the core 115 shown in FIG. 1A. The sequence 260 may be implemented in hardware, software, and/or firmware. In hardware embodiments it may be implemented as circuitry and/or micro-architecture. Further, in firmware and software embodiments, it may be implemented by computer executed instructions stored in a non-transitory machine readable medium, such as an optical, semiconductor, or magnetic storage device. The machine readable medium may store data, which if used by at least one machine,

causes the at least one machine to fabricate at least one integrated circuit to perform a method. For the sake of illustration, the steps involved in the sequence 260 may be described below with reference to FIGs. 1A-1B, which show examples in accordance with some embodiments. However, the scope of the various embodiments discussed herein is not limited in this regard.

[0069] At block 264, a non-blocking-protocol message may be received. For example, referring to FIG. 1A, the cache logic 117 in the core 115 may receive a non-blocking-protocol message from the coherency agent 130.

[0070] At block 267, a determination is made about whether the received non-blocking-protocol message is an invalidation message ("INV") or an acknowledgement message ("ACK"). For example, referring to FIG. 1A, the cache logic 117 in the core 115 may determine whether the non-blocking-protocol message is an invalidation message or an acknowledgement message.

[0071] If it is determined at block 267 that the received non-blocking-protocol message is an invalidation message, then at block 270, a determination is made about whether a non-blocking-protocol acknowledgement message is stored in a buffer. For example, referring to FIG. 1A, the cache logic 117 may determine whether an acknowledgement message is stored in the memory buffer 119 in the core 115.

[0072] If it is determined at block 270 that a non-blocking-protocol acknowledgement message is not stored in a buffer, then at block 274, the received invalidation message is stored in a buffer, and a wait is begun for the corresponding non-blocking-protocol acknowledgement message. For example, referring to FIG. 1A, the cache logic 117 may store the received invalidation message in the memory buffer 119. After block 274, the sequence 260 returns to block 264 to wait for the acknowledgement message.

[0073] However, if it is determined at block 267 that the received non-blocking-protocol message is an acknowledgement message, then at block 280, a determination is made about whether a non-blocking-protocol invalidation message is stored in a buffer. For example, referring to FIG. 1A, the cache logic 117 may determine whether an invalidation message is stored in the memory buffer 119.

[0074] If it is determined at block 280 that a non-blocking-protocol invalidation message is not stored in a buffer, then at block 284, the received acknowledgement message is stored in a buffer, and a wait is begun for the corresponding non-blocking-protocol invalidation message. For example, referring to FIG. 1A, the cache logic 117 may store the received acknowledgement message in the memory buffer 119. After block 284, the sequence 260 returns to block 264 to wait for the invalidation message.

[0075] However, if it is determined at block 270 that a non-blocking-protocol acknowledgement message is stored in the buffer, or if it is determined at block 280 that a non-blocking-protocol invalidation message is stored in the buffer, then at block 290, data included in the acknowledgement message may be processed. For example, referring to FIG. 1B, core A may process the data included in the acknowledgement message A4.

[0076] At block 294, a local copy of the data may be invalidated based on the invalidation message. For example, referring to FIG. 1B, core A may invalidate its local copy of the requested data in response to the invalidation message B2. In some embodiments, core A may send the data response B3 to the CA. After block 294, the sequence 260 terminates.

[0077] It is noted that the examples shown in FIGs. 1A-1B and FIGs. 2A-2C are provided for the sake of illustration, and are not intended to limit any embodiments. In particular, it is contemplated that the system 100, the processor 110, the core 115, the coherency agent 130, the synchronization logic 120, and/or the cache logic 117 may include different components, additional components, different arrangements of components, and/or different numbers of components than shown in FIGs. 1A-1B. For example, referring to FIG. 1A, it is contemplated that the synchronization logic 120 may be implemented in a single core 115, in multiple cores 115, and so forth. In another example, it is contemplated that the coherency agent 130 and the synchronization logic 120 may be implemented as separate components. In still another example, referring to FIG. 1B, it is contemplated that non-blocking-protocol messages may be processed in parallel to blocking-protocol messages, in parallel to other non-blocking-protocol messages, and so forth. Further, it is contemplated that specifics in the examples shown in FIGs. 1A-1B and FIGs. 2A-2C may be used anywhere in one or more embodiments. Other variations are contemplated and may be used in various embodiments.

[0078] Referring now to FIG. 3A, shown is a block diagram of a system 300 in accordance with an embodiment of the present invention. As shown in FIG. 3A, system 300 may include various components, including a processor 303 which as shown is a multicore processor. Processor 303 may be coupled to a power supply 317 via an external voltage regulator 316, which may perform a first voltage conversion to provide a primary regulated voltage to processor 303.

[0079] As seen, processor 303 may be a single die processor including multiple cores 304<sub>a</sub> – 304<sub>n</sub>. In addition, each core 304 may be associated with an integrated voltage regulator (IVR) 308<sub>a</sub> – 308<sub>n</sub> which receives the primary regulated voltage and generates an operating voltage to be provided to one or more agents of the processor associated with the IVR 308. Accordingly, an IVR implementation may be provided to allow for fine-grained control of voltage and thus power and performance of each individual core 304. As such, each core 304 can operate at an independent voltage and frequency, enabling great flexibility and affording wide opportunities for balancing power consumption with performance. In some embodiments, the use of multiple IVRs 308 enables the grouping of components into separate power planes, such that power is regulated and supplied by the IVR 308 to only those components in the group. During power management, a given power plane of one IVR 308 may be powered down or off when the processor is placed into a certain low power state, while another power plane of another IVR 308 remains active, or fully powered.

[0080] Still referring to FIG. 3A, additional components may be present within the processor including an input/output interface 313, another interface 314, and an integrated memory controller 315. As seen, each of these components may be powered by another integrated voltage regulator 308<sub>x</sub>. In one embodiment, interface 313 may be in accordance with the Intel® Quick Path Interconnect (QPI) protocol, which provides for point-to-point (PtP) links in a cache coherent protocol that includes multiple layers including a physical layer, a link layer and a protocol layer. In turn, interface 314 may be in accordance with a Peripheral Component Interconnect Express (PCIe<sup>TM</sup>) specification, e.g., the PCI Express<sup>TM</sup> Specification Base Specification version 2.0 (published January 17, 2007).

[0081] Also shown is a power control unit (PCU) 312, which may include hardware, software and/or firmware to perform power management operations with regard to processor 303. As seen, PCU 312 provides control information to external voltage regulator 316 via a

digital interface to cause the external voltage regulator 316 to generate the appropriate regulated voltage. PCU 312 also provides control information to IVRs 308 via another digital interface to control the operating voltage generated (or to cause a corresponding IVR 308 to be disabled in a low power mode). In some embodiments, the control information provided to IVRs 308 may include a power state of a corresponding core 304.

[0082] In various embodiments, PCU 312 may include a variety of power management logic units to perform hardware-based power management. Such power management may be wholly processor controlled (e.g., by various processor hardware, and which may be triggered by workload and/or power, thermal or other processor constraints) and/or the power management may be performed responsive to external sources (such as a platform or management power management source or system software).

[0083] While not shown for ease of illustration, understand that additional components may be present within processor 303 such as uncore logic, and other components such as internal memories, e.g., one or more levels of a cache memory hierarchy and so forth. Furthermore, while shown in the implementation of FIG. 3A with an external voltage regulator, embodiments are not so limited.

[0084] Although not shown for ease of illustration in FIG. 3A, in some embodiments, processor 303 may include some or all of the components and/or functionality of the coherency agent 130 and/or the synchronization logic 120 described above with reference to FIGs. 1A-1B and 2A-2C. Further, in some embodiments, the cores 304<sub>a</sub> – 304<sub>n</sub> may include some or all of the components and/or functionality of the cache logic 117 described above with reference to FIGs. 1A-1B and 2A-2C.

[0085] Embodiments can be implemented in processors for various markets including server processors, desktop processors, mobile processors and so forth. Referring now to FIG. 3B, shown is a block diagram of a multi-domain processor 301 in accordance with one or more embodiments. As shown in the embodiment of FIG. 3B, processor 301 includes multiple domains. Specifically, a core domain 321 can include a plurality of cores 320<sub>0</sub>–320<sub>n</sub>, a graphics domain 324 can include one or more graphics engines, and a system agent domain 330 may further be present. In some embodiments, system agent domain 330 may execute at an independent frequency than the core domain and may remain powered on at all

times to handle power control events and power management such that domains 321 and 324 can be controlled to dynamically enter into and exit high power and low power states. Each of domains 321 and 324 may operate at different voltage and/or power. Note that while only shown with three domains, understand the scope of the present invention is not limited in this regard and additional domains can be present in other embodiments. For example, multiple core domains may be present, with each core domain including at least one core.

[0086] In general, each core 320 may further include low level caches in addition to various execution units and additional processing elements. In turn, the various cores may be coupled to each other and to a shared cache memory formed of a plurality of units of a last level cache (LLC) 322<sub>0</sub> – 322<sub>n</sub>. In various embodiments, LLC 322 may be shared amongst the cores and the graphics engine, as well as various media processing circuitry. As seen, a ring interconnect 323 thus couples the cores together, and provides interconnection between the cores 320, graphics domain 324 and system agent domain 330. In one embodiment, interconnect 323 can be part of the core domain 321. However, in other embodiments, the ring interconnect 323 can be of its own domain.

[0087] As further seen, system agent domain 330 may include display controller 332 which may provide control of and an interface to an associated display. In addition, system agent domain 330 may include a power control unit 335 to perform power management.

[0088] As further seen in FIG. 3B, processor 301 can further include an integrated memory controller (IMC) 342 that can provide for an interface to a system memory, such as a dynamic random access memory (DRAM). Multiple interfaces 340<sub>0</sub> – 340<sub>n</sub> may be present to enable interconnection between the processor and other circuitry. For example, in one embodiment at least one direct media interface (DMI) interface may be provided as well as one or more PCIe™ interfaces. Still further, to provide for communications between other agents such as additional processors or other circuitry, one or more interfaces in accordance with an Intel® Quick Path Interconnect (QPI) protocol may also be provided. Although shown at this high level in the embodiment of FIG. 3B, understand the scope of the present invention is not limited in this regard.

[0089] Although not shown for ease of illustration in FIG. 3B, in some embodiments, processor 301 may include some or all of the components and/or functionality of the

coherency agent 130 and/or the synchronization logic 120 described above with reference to FIGs. 1A-1B and 2A-2C. Further, in some embodiments, the cores 320 may include some or all of the components and/or functionality of the cache logic 117 described above with reference to FIGs. 1A-1B and 2A-2C.

[0090] Referring now to FIG. 3C, shown is a block diagram of a processor 302 in accordance with an embodiment of the present invention. As shown in FIG. 3C, processor 302 may be a multicore processor including a plurality of cores 370<sub>a</sub> – 370<sub>n</sub>. In one embodiment, each such core may be of an independent power domain and can be configured to enter and exit active states and/or maximum performance states based on workload. The various cores may be coupled via an interconnect 375 to a system agent or uncore 380 that includes various components. As seen, the uncore 380 may include a shared cache 382 which may be a last level cache. In addition, the uncore 380 may include an integrated memory controller 384 to communicate with a system memory (not shown in FIG. 3C), e.g., via a memory bus. Uncore 380 also includes various interfaces 386a-386n and a power control unit 388, which may include logic to perform the power management techniques described herein.

[0091] In addition, by interfaces 386a-386n, connection can be made to various off-chip components such as peripheral devices, mass storage and so forth. While shown with this particular implementation in the embodiment of FIG. 3C, the scope of the present invention is not limited in this regard.

[0092] Although not shown for ease of illustration in FIG. 3C, in some embodiments, processor 302 may include some or all of the components and/or functionality of the coherency agent 130 and/or the synchronization logic 120 described above with reference to FIGs. 1A-1B and 2A-2C. Further, in some embodiments, the cores 370 may include some or all of the components and/or functionality of the cache logic 117 described above with reference to FIGs. 1A-1B and 2A-2C.

[0093] Referring to FIG. 4, an embodiment of a processor including multiple cores is illustrated. Processor 400 includes any processor or processing device, such as a microprocessor, an embedded processor, a digital signal processor (DSP), a network processor, a handheld processor, an application processor, a co-processor, a system on a chip

(SoC), or other device to execute code. Processor 400, in one embodiment, includes at least two cores—cores 401 and 402, which may include asymmetric cores or symmetric cores (the illustrated embodiment). However, processor 400 may include any number of processing elements that may be symmetric or asymmetric.

[0094] In one embodiment, a processing element refers to hardware or logic to support a software thread. Examples of hardware processing elements include: a thread unit, a thread slot, a thread, a process unit, a context, a context unit, a logical processor, a hardware thread, a core, and/or any other element, which is capable of holding a state for a processor, such as an execution state or architectural state. In other words, a processing element, in one embodiment, refers to any hardware capable of being independently associated with code, such as a software thread, operating system, application, or other code. A physical processor typically refers to an integrated circuit, which potentially includes any number of other processing elements, such as cores or hardware threads.

[0095] A core often refers to logic located on an integrated circuit capable of maintaining an independent architectural state, wherein each independently maintained architectural state is associated with at least some dedicated execution resources. In contrast to cores, a hardware thread typically refers to any logic located on an integrated circuit capable of maintaining an independent architectural state, wherein the independently maintained architectural states share access to execution resources. As can be seen, when certain resources are shared and others are dedicated to an architectural state, the line between the nomenclature of a hardware thread and core overlaps. Yet often, a core and a hardware thread are viewed by an operating system as individual logical processors, where the operating system is able to individually schedule operations on each logical processor.

[0096] Physical processor 400, as illustrated in FIG. 4, includes two cores, cores 401 and 402. Here, cores 401 and 402 are considered symmetric cores, i.e., cores with the same configurations, functional units, and/or logic. In another embodiment, core 401 includes an out-of-order processor core, while core 402 includes an in-order processor core. However, cores 401 and 402 may be individually selected from any type of core, such as a native core, a software managed core, a core adapted to execute a native instruction set architecture (ISA), a core adapted to execute a translated ISA, a co-designed core, or other known core. Yet to

further the discussion, the functional units illustrated in core 401 are described in further detail below, as the units in core 402 operate in a similar manner.

[0097] As depicted, core 401 includes two hardware threads 401a and 401b, which may also be referred to as hardware thread slots 401a and 401b. Therefore, software entities, such as an operating system, in one embodiment potentially view processor 400 as four separate processors, i.e., four logical processors or processing elements capable of executing four software threads concurrently. As alluded to above, a first thread is associated with architecture state registers 401a, a second thread is associated with architecture state registers 401b, a third thread may be associated with architecture state registers 402a, and a fourth thread may be associated with architecture state registers 402b. Here, each of the architecture state registers (401a, 401b, 402a, and 402b) may be referred to as processing elements, thread slots, or thread units, as described above. As illustrated, architecture state registers 401a are replicated in architecture state registers 401b, so individual architecture states/contexts are capable of being stored for logical processor 401a and logical processor 401b. In core 401, other smaller resources, such as instruction pointers and renaming logic in allocator and renamer block 430 may also be replicated for threads 401a and 401b. Some resources, such as re-order buffers in reorder/retirement unit 435, ILTB 420, load/store buffers, and queues may be shared through partitioning. Other resources, such as general purpose internal registers, page-table base register(s), low-level data-cache and data-TLB 415, execution unit(s) 440, and portions of out-of-order unit 435 are potentially fully shared.

[0098] Processor 400 often includes other resources, which may be fully shared, shared through partitioning, or dedicated by/to processing elements. In FIG. 4, an embodiment of a purely exemplary processor with illustrative logical units/resources of a processor is illustrated. Note that a processor may include, or omit, any of these functional units, as well as include any other known functional units, logic, or firmware not depicted. As illustrated, core 401 includes a simplified, representative out-of-order (OOO) processor core. But an in-order processor may be utilized in different embodiments. The OOO core includes a branch target buffer 420 to predict branches to be executed/taken and an instruction-translation buffer (I-TLB) 420 to store address translation entries for instructions.

[0099] Core 401 further includes decode module 425 coupled to fetch unit 420 to decode fetched elements. Fetch logic, in one embodiment, includes individual sequencers associated

with thread slots 401a, 401b, respectively. Usually core 401 is associated with a first ISA, which defines/specifies instructions executable on processor 400. Often machine code instructions that are part of the first ISA include a portion of the instruction (referred to as an opcode), which references/specifies an instruction or operation to be performed. Decode logic 425 includes circuitry that recognizes these instructions from their opcodes and passes the decoded instructions on in the pipeline for processing as defined by the first ISA. For example, decoders 425, in one embodiment, include logic designed or adapted to recognize specific instructions, such as transactional instruction. As a result of the recognition by decoders 425, the architecture or core 401 takes specific, predefined actions to perform tasks associated with the appropriate instruction. It is important to note that any of the tasks, blocks, operations, and methods described herein may be performed in response to a single or multiple instructions; some of which may be new or old instructions.

[0100] In one example, allocator and renamer block 430 includes an allocator to reserve resources, such as register files to store instruction processing results. However, threads 401a and 401b are potentially capable of out-of-order execution, where allocator and renamer block 430 also reserves other resources, such as reorder buffers to track instruction results. Unit 430 may also include a register renamer to rename program/instruction reference registers to other registers internal to processor 400. Reorder/retirement unit 435 includes components, such as the reorder buffers mentioned above, load buffers, and store buffers, to support out-of-order execution and later in-order retirement of instructions executed out-of-order.

[0101] Scheduler and execution unit(s) block 440, in one embodiment, includes a scheduler unit to schedule instructions/operation on execution units. For example, a floating point instruction is scheduled on a port of an execution unit that has an available floating point execution unit. Register files associated with the execution units are also included to store information instruction processing results. Exemplary execution units include a floating point execution unit, an integer execution unit, a jump execution unit, a load execution unit, a store execution unit, and other known execution units.

[0102] Lower level data cache and data translation buffer (D-TLB) 450 are coupled to execution unit(s) 440. The data cache is to store recently used/operated on elements, such as data operands, which are potentially held in memory coherency states. The D-TLB is to store

recent virtual/linear to physical address translations. As a specific example, a processor may include a page table structure to break physical memory into a plurality of virtual pages.

[0103] Here, cores 401 and 402 share access to higher-level or further-out cache 410, which is to cache recently fetched elements. Note that higher-level or further-out refers to cache levels increasing or getting further away from the execution unit(s). In one embodiment, higher-level cache 410 is a last-level data cache—last cache in the memory hierarchy on processor 400—such as a second or third level data cache. However, higher level cache 410 is not so limited, as it may be associated with or includes an instruction cache. A trace cache—a type of instruction cache—instead may be coupled after decoder 425 to store recently decoded traces.

[0104] In the depicted configuration, processor 400 also includes bus interface module 405 and a power controller 460, which may perform power management in accordance with an embodiment of the present invention. In this scenario, bus interface 405 is to communicate with devices external to processor 400, such as system memory and other components.

[0105] A memory controller 470 may interface with other devices such as one or many memories. In an example, bus interface 405 includes a ring interconnect with a memory controller for interfacing with a memory and a graphics controller for interfacing with a graphics processor. In an SoC environment, even more devices, such as a network interface, coprocessors, memory, graphics processor, and any other known computer devices/interface may be integrated on a single die or integrated circuit to provide small form factor with high functionality and low power consumption.

[0106] Although not shown for ease of illustration in FIG. 4, in some embodiments, processor 400 may include some or all of the components and/or functionality of the coherency agent 130 and/or the synchronization logic 120 described above with reference to FIGs. 1A-1B and 2A-2C. Further, in some embodiments, the cores 401, 402 may include some or all of the components and/or functionality of the cache logic 117 described above with reference to FIGs. 1A-1B and 2A-2C.

[0107] Referring now to FIG. 5, shown is a block diagram of a micro-architecture of a processor core in accordance with one embodiment of the present invention. As shown in FIG. 5, processor core 500 may be a multi-stage pipelined out-of-order processor. Core 500

may operate at various voltages based on a received operating voltage, which may be received from an integrated voltage regulator or external voltage regulator.

[0108] As seen in FIG. 5, core 500 includes front end units 510, which may be used to fetch instructions to be executed and prepare them for use later in the processor pipeline. For example, front end units 510 may include a fetch unit 501, an instruction cache 503, and an instruction decoder 505. In some implementations, front end units 510 may further include a trace cache, along with microcode storage as well as a micro-operation storage. Fetch unit 501 may fetch macro-instructions, e.g., from memory or instruction cache 503, and feed them to instruction decoder 505 to decode them into primitives, i.e., micro-operations for execution by the processor.

[0109] Coupled between front end units 510 and execution units 520 is an out-of-order (OOO) engine 515 that may be used to receive the micro-instructions and prepare them for execution. More specifically OOO engine 515 may include various buffers to re-order micro-instruction flow and allocate various resources needed for execution, as well as to provide renaming of logical registers onto storage locations within various register files such as register file 530 and extended register file 535. Register file 530 may include separate register files for integer and floating point operations. Extended register file 535 may provide storage for vector-sized units, e.g., 256 or 512 bits per register.

[0110] Various resources may be present in execution units 520, including, for example, various integer, floating point, and single instruction multiple data (SIMD) logic units, among other specialized hardware. For example, such execution units may include one or more arithmetic logic units (ALUs) 522 and one or more vector execution units 524, among other such execution units.

[0111] Results from the execution units may be provided to retirement logic, namely a reorder buffer (ROB) 540. More specifically, ROB 540 may include various arrays and logic to receive information associated with instructions that are executed. This information is then examined by ROB 540 to determine whether the instructions can be validly retired and result data committed to the architectural state of the processor, or whether one or more exceptions occurred that prevent a proper retirement of the instructions. Of course, ROB 540 may handle other operations associated with retirement.

[0112] As shown in FIG. 5, ROB 540 is coupled to a cache 550 which, in one embodiment may be a low level cache (e.g., an L1 cache) although the scope of the present invention is not limited in this regard. Also, execution units 520 can be directly coupled to cache 550. From cache 550, data communication may occur with higher level caches, system memory and so forth. While shown with this high level in the embodiment of FIG. 5, understand the scope of the present invention is not limited in this regard. For example, while the implementation of FIG. 5 is with regard to an out-of-order machine such as of an Intel® x86 instruction set architecture (ISA), the scope of the present invention is not limited in this regard. That is, other embodiments may be implemented in an in-order processor, a reduced instruction set computing (RISC) processor such as an ARM-based processor, or a processor of another type of ISA that can emulate instructions and operations of a different ISA via an emulation engine and associated logic circuitry.

[0113] Although not shown for ease of illustration in FIG. 5, in some embodiments, the core 500 may include some or all of the components and/or functionality described above with reference to FIGs. 1A-1B and 2A-2C.

[0114] Referring now to FIG. 6, shown is a block diagram of a micro-architecture of a processor core in accordance with another embodiment. In the embodiment of FIG. 6, core 600 may be a low power core of a different micro-architecture, such as an Intel® Atom™-based processor having a relatively limited pipeline depth designed to reduce power consumption. As seen, core 600 includes an instruction cache 610 coupled to provide instructions to an instruction decoder 615. A branch predictor 605 may be coupled to instruction cache 610. Note that instruction cache 610 may further be coupled to another level of a cache memory, such as an L2 cache (not shown for ease of illustration in FIG. 6). In turn, instruction decoder 615 provides decoded instructions to an issue queue 620 for storage and delivery to a given execution pipeline. A microcode ROM 618 is coupled to instruction decoder 615.

[0115] A floating point pipeline 630 includes a floating point register file 632 which may include a plurality of architectural registers of a given bit with such as 128, 256 or 512 bits. Pipeline 630 includes a floating point scheduler 634 to schedule instructions for execution on one of multiple execution units of the pipeline. In the embodiment shown, such execution units include an ALU 635, a shuffle unit 636, and a floating point adder 638. In turn, results

generated in these execution units may be provided back to buffers and/or registers of register file 632. Of course understand while shown with these few example execution units, additional or different floating point execution units may be present in another embodiment.

[0116] An integer pipeline 640 also may be provided. In the embodiment shown, pipeline 640 includes an integer register file 642 which may include a plurality of architectural registers of a given bit with such as 128 or 256 bits. Pipeline 640 includes an integer scheduler 644 to schedule instructions for execution on one of multiple execution units of the pipeline. In the embodiment shown, such execution units include an ALU 645, a shifter unit 646, and a jump execution unit 648. In turn, results generated in these execution units may be provided back to buffers and/or registers of register file 642. Of course understand while shown with these few example execution units, additional or different integer execution units may be present in another embodiment.

[0117] A memory execution scheduler 650 may schedule memory operations for execution in an address generation unit 652, which is also coupled to a TLB 654. As seen, these structures may couple to a data cache 660, which may be a L0 and/or L1 data cache that in turn couples to additional levels of a cache memory hierarchy, including an L2 cache memory.

[0118] To provide support for out-of-order execution, an allocator/renamer 670 may be provided, in addition to a reorder buffer 680, which is configured to reorder instructions executed out of order for retirement in order. Although shown with this particular pipeline architecture in the illustration of FIG. 6, understand that many variations and alternatives are possible.

[0119] Note that in a processor having asymmetric cores, such as in accordance with the micro-architectures of FIGs. 5 and 6, workloads may be dynamically swapped between the cores for power management reasons, as these cores, although having different pipeline designs and depths, may be of the same or related ISA. Such dynamic core swapping may be performed in a manner transparent to a user application (and possibly kernel also).

[0120] Although not shown for ease of illustration in FIG. 6, in some embodiments, the core 600 may include some or all of the components and/or functionality described above with reference to FIGs. 1A-1B and 2A-2C.

[0121] Referring to FIG. 7, shown is a block diagram of a micro-architecture of a processor core in accordance with yet another embodiment. As illustrated in FIG. 7, a core 700 may include a multi-staged in-order pipeline to execute at very low power consumption levels. As one such example, processor 700 may have a micro-architecture in accordance with an ARM Cortex A53 design available from ARM Holdings, LTD., Sunnyvale, CA. In an implementation, an 8-stage pipeline may be provided that is configured to execute both 32-bit and 64-bit code. Core 700 includes a fetch unit 710 that is configured to fetch instructions and provide them to a decode unit 715, which may decode the instructions, e.g., macro-instructions of a given ISA such as an ARMv8 ISA. Note further that a queue 730 may couple to decode unit 715 to store decoded instructions. Decoded instructions are provided to an issue logic 725, where the decoded instructions may be issued to a given one of multiple execution units.

[0122] With further reference to FIG. 7, issue logic 725 may issue instructions to one of multiple execution units. In the embodiment shown, these execution units include an integer unit 735, a multiply unit 740, a floating point/vector unit 750, a dual issue unit 760, and a load/store unit 770. The results of these different execution units may be provided to a writeback unit 780. Understand that while a single writeback unit is shown for ease of illustration, in some implementations separate writeback units may be associated with each of the execution units. Furthermore, understand that while each of the units and logic shown in FIG. 7 is represented at a high level, a particular implementation may include more or different structures. A processor designed using one or more cores having a pipeline as in FIG. 7 may be implemented in many different end products, extending from mobile devices to server systems.

[0123] Although not shown for ease of illustration in FIG. 7, in some embodiments, the core 700 may include some or all of the components and/or functionality described above with reference to FIGs. 1A-1B and 2A-2C.

[0124] Referring now to FIG. 8, shown is a block diagram of a micro-architecture of a processor core in accordance with a still further embodiment. As illustrated in FIG. 8, a core 800 may include a multi-stage multi-issue out-of-order pipeline to execute at very high performance levels (which may occur at higher power consumption levels than core 700 of FIG. 7). As one such example, processor 800 may have a microarchitecture in accordance

with an ARM Cortex A57 design. In an implementation, a 15 (or greater)-stage pipeline may be provided that is configured to execute both 32-bit and 64-bit code. In addition, the pipeline may provide for 3 (or greater)-wide and 3 (or greater)-issue operation. Core 800 includes a fetch unit 810 that is configured to fetch instructions and provide them to a decoder/renamer/dispatcher 815, which may decode the instructions, e.g., macro-instructions of an ARMv8 instruction set architecture, rename register references within the instructions, and dispatch the instructions (eventually) to a selected execution unit. Decoded instructions may be stored in a queue 825. Note that while a single queue structure is shown for ease of illustration in FIG 8, understand that separate queues may be provided for each of the multiple different types of execution units.

[0125] Also shown in FIG. 8 is an issue logic 830 from which decoded instructions stored in queue 825 may be issued to a selected execution unit. Issue logic 830 also may be implemented in a particular embodiment with a separate issue logic for each of the multiple different types of execution units to which issue logic 830 couples.

[0126] Decoded instructions may be issued to a given one of multiple execution units. In the embodiment shown, these execution units include one or more integer units 835, a multiply unit 840, a floating point/vector unit 850, a branch unit 860, and a load/store unit 870. In an embodiment, floating point/vector unit 850 may be configured to handle SIMD or vector data of 128 or 256 bits. Still further, floating point/vector execution unit 850 may perform IEEE-754 double precision floating-point operations. The results of these different execution units may be provided to a writeback unit 880. Note that in some implementations separate writeback units may be associated with each of the execution units. Furthermore, understand that while each of the units and logic shown in FIG. 8 is represented at a high level, a particular implementation may include more or different structures.

[0127] Note that in a processor having asymmetric cores, such as in accordance with the micro-architectures of FIGs. 7 and 8, workloads may be dynamically swapped for power management reasons, as these cores, although having different pipeline designs and depths, may be of the same or related ISA. Such dynamic core swapping may be performed in a manner transparent to a user application (and possibly kernel also).

[0128] Although not shown for ease of illustration in FIG. 8, in some embodiments, the core 800 may include some or all of the components and/or functionality described above with reference to FIGs. 1A-1B and 2A-2C.

[0129] A processor designed using one or more cores having pipelines as in any one or more of FIGs. 5-8 may be implemented in many different end products, extending from mobile devices to server systems. Referring now to FIG. 9, shown is a block diagram of a processor in accordance with another embodiment of the present invention. In the embodiment of FIG. 9, processor 900 may be a SoC including multiple domains, each of which may be controlled to operate at an independent operating voltage and operating frequency. As a specific illustrative example, processor 900 may be an Intel® Architecture Core™-based processor such as an i3, i5, i7 or another such processor available from Intel Corporation. However, other low power processors such as available from Advanced Micro Devices, Inc. (AMD) of Sunnyvale, CA, an ARM-based design from ARM Holdings, Ltd. or licensee thereof or a MIPS-based design from MIPS Technologies, Inc. of Sunnyvale, CA, or their licensees or adopters may instead be present in other embodiments such as an Apple A7 processor, a Qualcomm Snapdragon processor, or Texas Instruments OMAP processor. Such SoC may be used in a low power system such as a smartphone, tablet computer, phablet computer, Ultrabook™ computer or other portable computing device.

[0130] In the high level view shown in FIG. 9, processor 900 includes a plurality of core units 910<sub>0</sub>-910<sub>n</sub>. Each core unit may include one or more processor cores, one or more cache memories and other circuitry. Each core unit 910 may support one or more instructions sets (e.g., an x86 instruction set (with some extensions that have been added with newer versions); a MIPS instruction set; an ARM instruction set (with optional additional extensions such as NEON)) or other instruction set or combinations thereof. Note that some of the core units may be heterogeneous resources (e.g., of a different design). In addition, each such core may be coupled to a cache memory (not shown) which in an embodiment may be a shared level (L2) cache memory. A non-volatile storage 930 may be used to store various program and other data. For example, this storage may be used to store at least portions of microcode, boot information such as a BIOS, other system software or so forth.

[0131] Each core unit 910 may also include an interface such as a bus interface unit to enable interconnection to additional circuitry of the processor. In an embodiment, each core

unit 910 couples to a coherent fabric that may act as a primary cache coherent on-die interconnect that in turn couples to a memory controller 935. In turn, memory controller 935 controls communications with a memory such as a DRAM (not shown for ease of illustration in FIG. 9).

[0132] In addition to core units, additional processing engines are present within the processor, including at least one graphics unit 920 which may include one or more graphics processing units (GPUs) to perform graphics processing as well as to possibly execute general purpose operations on the graphics processor (so-called GPGPU operation). In addition, at least one image signal processor 925 may be present. Signal processor 925 may be configured to process incoming image data received from one or more capture devices, either internal to the SoC or off-chip.

[0133] Other accelerators also may be present. In the illustration of FIG. 9, a video coder 950 may perform coding operations including encoding and decoding for video information, e.g., providing hardware acceleration support for high definition video content. A display controller 955 further may be provided to accelerate display operations including providing support for internal and external displays of a system. In addition, a security processor 945 may be present to perform security operations such as secure boot operations, various cryptography operations and so forth.

[0134] Each of the units may have its power consumption controlled via a power manager 940, which may include control logic to perform the various power management techniques described herein.

[0135] In some embodiments, SoC 900 may further include a non-coherent fabric coupled to the coherent fabric to which various peripheral devices may couple. One or more interfaces 960a-960d enable communication with one or more off-chip devices. Such communications may be according to a variety of communication protocols such as PCIe™, GPIO, USB, I<sup>2</sup>C, UART, MIPI, SDIO, DDR, SPI, HDMI, among other types of communication protocols. Although shown at this high level in the embodiment of FIG. 9, understand the scope of the present invention is not limited in this regard.

[0136] Although not shown for ease of illustration in FIG. 9, in some embodiments, the SoC 900 may include some or all of the components and/or functionality described above with reference to FIGs. 1A-1B and 2A-2C.

[0137] Referring now to FIG. 10, shown is a block diagram of a representative SoC. In the embodiment shown, SoC 1000 may be a multi-core SoC configured for low power operation to be optimized for incorporation into a smartphone or other low power device such as a tablet computer or other portable computing device. As an example, SoC 1000 may be implemented using asymmetric or different types of cores, such as combinations of higher power and/or low power cores, e.g., out-of-order cores and in-order cores. In different embodiments, these cores may be based on an Intel® Architecture™ core design or an ARM architecture design. In yet other embodiments, a mix of Intel and ARM cores may be implemented in a given SoC.

[0138] As seen in FIG. 10, SoC 1000 includes a first core domain 1010 having a plurality of first cores 1012<sub>0</sub> – 1012<sub>3</sub>. In an example, these cores may be low power cores such as in-order cores. In one embodiment these first cores may be implemented as ARM Cortex A53 cores. In turn, these cores couple to a cache memory 1015 of core domain 1010. In addition, SoC 1000 includes a second core domain 1020. In the illustration of FIG. 10, second core domain 1020 has a plurality of second cores 1022<sub>0</sub> – 1022<sub>3</sub>. In an example, these cores may be higher power-consuming cores than first cores 1012. In an embodiment, the second cores may be out-of-order cores, which may be implemented as ARM Cortex A57 cores. In turn, these cores couple to a cache memory 1025 of core domain 1020. Note that while the example shown in FIG. 10 includes 4 cores in each domain, understand that more or fewer cores may be present in a given domain in other examples.

[0139] With further reference to FIG. 10, a graphics domain 1030 also is provided, which may include one or more graphics processing units (GPUs) configured to independently execute graphics workloads, e.g., provided by one or more cores of core domains 1010 and 1020. As an example, GPU domain 1030 may be used to provide display support for a variety of screen sizes, in addition to providing graphics and display rendering operations.

[0140] As seen, the various domains couple to a coherent interconnect 1040, which in an embodiment may be a cache coherent interconnect fabric that in turn couples to an integrated

memory controller 1050. Coherent interconnect 1040 may include a shared cache memory, such as an L3 cache, some examples. In an embodiment, memory controller 1050 may be a direct memory controller to provide for multiple channels of communication with an off-chip memory, such as multiple channels of a DRAM (not shown for ease of illustration in FIG. 10).

[0141] In different examples, the number of the core domains may vary. For example, for a low power SoC suitable for incorporation into a mobile computing device, a limited number of core domains such as shown in FIG. 10 may be present. Still further, in such low power SoCs, core domain 1020 including higher power cores may have fewer numbers of such cores. For example, in one implementation two cores 1022 may be provided to enable operation at reduced power consumption levels. In addition, the different core domains may also be coupled to an interrupt controller to enable dynamic swapping of workloads between the different domains.

[0142] In yet other embodiments, a greater number of core domains, as well as additional optional IP logic may be present, in that an SoC can be scaled to higher performance (and power) levels for incorporation into other computing devices, such as desktops, servers, high performance computing systems, base stations forth. As one such example, 4 core domains each having a given number of out-of-order cores may be provided. Still further, in addition to optional GPU support (which as an example may take the form of a GPGPU), one or more accelerators to provide optimized hardware support for particular functions (e.g. web serving, network processing, switching or so forth) also may be provided. In addition, an input/output interface may be present to couple such accelerators to off-chip components.

[0143] Although not shown for ease of illustration in FIG. 10, in some embodiments, the SoC 1000 may include some or all of the components and/or functionality described above with reference to FIGs. 1A-1B and 2A-2C.

[0144] Referring now to FIG. 11, shown is a block diagram of another example SoC. In the embodiment of FIG. 11, SoC 1100 may include various circuitry to enable high performance for multimedia applications, communications and other functions. As such, SoC 1100 is suitable for incorporation into a wide variety of portable and other devices, such as smartphones, tablet computers, smart TVs and so forth. In the example shown, SoC 1100

includes a central processor unit (CPU) domain 1110. In an embodiment, a plurality of individual processor cores may be present in CPU domain 1110. As one example, CPU domain 1110 may be a quad core processor having 4 multithreaded cores. Such processors may be homogeneous or heterogeneous processors, e.g., a mix of low power and high power processor cores.

[0145] In turn, a GPU domain 1120 is provided to perform advanced graphics processing in one or more GPUs to handle graphics and compute APIs. A DSP unit 1130 may provide one or more low power DSPs for handling low-power multimedia applications such as music playback, audio/video and so forth, in addition to advanced calculations that may occur during execution of multimedia instructions. In turn, a communication unit 1140 may include various components to provide connectivity via various wireless protocols, such as cellular communications (including 3G/4G LTE), wireless local area techniques such as Bluetooth<sup>TM</sup>, IEEE 802.11, and so forth.

[0146] Still further, a multimedia processor 1150 may be used to perform capture and playback of high definition video and audio content, including processing of user gestures. A sensor unit 1160 may include a plurality of sensors and/or a sensor controller to interface to various off-chip sensors present in a given platform. An image signal processor 1170 may be provided with one or more separate ISPs to perform image processing with regard to captured content from one or more cameras of a platform, including still and video cameras.

[0147] A display processor 1180 may provide support for connection to a high definition display of a given pixel density, including the ability to wirelessly communicate content for playback on such display. Still further, a location unit 1190 may include a GPS receiver with support for multiple GPS constellations to provide applications highly accurate positioning information obtained using as such GPS receiver. Understand that while shown with this particular set of components in the example of FIG. 11, many variations and alternatives are possible.

[0148] Although not shown for ease of illustration in FIG. 11, in some embodiments, the SoC 1100 may include some or all of the components and/or functionality described above with reference to FIGs. 1A-1B and 2A-2C.

[0149] Referring now to FIG. 12, shown is a block diagram of an example system with which embodiments can be used. As seen, system 1200 may be a smartphone or other wireless communicator. A baseband processor 1205 is configured to perform various signal processing with regard to communication signals to be transmitted from or received by the system. In turn, baseband processor 1205 is coupled to an application processor 1210, which may be a main CPU of the system to execute an OS and other system software, in addition to user applications such as many well-known social media and multimedia apps. Application processor 1210 may further be configured to perform a variety of other computing operations for the device.

[0150] In turn, application processor 1210 can couple to a user interface/display 1220, e.g., a touch screen display. In addition, application processor 1210 may couple to a memory system including a non-volatile memory, namely a flash memory 1230 and a system memory, namely a dynamic random access memory (DRAM) 1235. As further seen, application processor 1210 further couples to a capture device 1240 such as one or more image capture devices that can record video and/or still images.

[0151] Still referring to FIG. 12, a universal integrated circuit card (UICC) 1240 comprising a subscriber identity module and possibly a secure storage and cryptoprocessor is also coupled to application processor 1210. System 1200 may further include a security processor 1250 that may couple to application processor 1210. A plurality of sensors 1225 may couple to application processor 1210 to enable input of a variety of sensed information such as accelerometer and other environmental information. An audio output device 1295 may provide an interface to output sound, e.g., in the form of voice communications, played or streaming audio data and so forth.

[0152] As further illustrated, a near field communication (NFC) contactless interface 1260 is provided that communicates in a NFC near field via an NFC antenna 1265. While separate antennae are shown in FIG. 12, understand that in some implementations one antenna or a different set of antennae may be provided to enable various wireless functionality.

[0153] A power management integrated circuit (PMIC) 1215 couples to application processor 1210 to perform platform level power management. To this end, PMIC 1215 may issue power management requests to application processor 1210 to enter certain low power

states as desired. Furthermore, based on platform constraints, PMIC 1215 may also control the power level of other components of system 1200.

[0154] To enable communications to be transmitted and received, various circuitry may be coupled between baseband processor 1205 and an antenna 1290. Specifically, a radio frequency (RF) transceiver 1270 and a wireless local area network (WLAN) transceiver 1275 may be present. In general, RF transceiver 1270 may be used to receive and transmit wireless data and calls according to a given wireless communication protocol such as 3G or 4G wireless communication protocol such as in accordance with a code division multiple access (CDMA), global system for mobile communication (GSM), long term evolution (LTE) or other protocol. In addition a GPS sensor 1280 may be present. Other wireless communications such as receipt or transmission of radio signals, e.g., AM/FM and other signals may also be provided. In addition, via WLAN transceiver 1275, local wireless communications, such as according to a Bluetooth™ standard or an IEEE 802.11 standard such as IEEE 802.11a/b/g/n can also be realized.

[0155] Although not shown for ease of illustration in FIG. 12, in some embodiments, the system 1200 may include some or all of the components and/or functionality described above with reference to FIGs. 1A-1B and 2A-2C.

[0156] Referring now to FIG. 13, shown is a block diagram of another example system with which embodiments may be used. In the illustration of FIG 13, system 1300 may be mobile low-power system such as a tablet computer, 2:1 tablet, phablet or other convertible or standalone tablet system. As illustrated, a SoC 1310 is present and may be configured to operate as an application processor for the device.

[0157] A variety of devices may couple to SoC 1310. In the illustration shown, a memory subsystem includes a flash memory 1340 and a DRAM 1345 coupled to SoC 1310. In addition, a touch panel 1320 is coupled to the SoC 1310 to provide display capability and user input via touch, including provision of a virtual keyboard on a display of touch panel 1320. To provide wired network connectivity, SoC 1310 couples to an Ethernet interface 1330. A peripheral hub 1325 is coupled to SoC 1310 to enable interfacing with various peripheral devices, such as may be coupled to system 1300 by any of various ports or other connectors.

[0158] In addition to internal power management circuitry and functionality within SoC 1310, a PMIC 1380 is coupled to SoC 1310 to provide platform-based power management, e.g., based on whether the system is powered by a battery 1390 or AC power via an AC adapter 1395. In addition to this power source-based power management, PMIC 1380 may further perform platform power management activities based on environmental and usage conditions. Still further, PMIC 1380 may communicate control and status information to SoC 1310 to cause various power management actions within SoC 1310.

[0159] Still referring to FIG. 13, to provide for wireless capabilities, a WLAN unit 1350 is coupled to SoC 1310 and in turn to an antenna 1355. In various implementations, WLAN unit 1350 may provide for communication according to one or more wireless protocols, including an IEEE 802.11 protocol, a Bluetooth<sup>TM</sup> protocol or any other wireless protocol.

[0160] As further illustrated, a plurality of sensors 1360 may couple to SoC 1310. These sensors may include various accelerometer, environmental and other sensors, including user gesture sensors. Finally, an audio codec 1365 is coupled to SoC 1310 to provide an interface to an audio output device 1370. Of course understand that while shown with this particular implementation in FIG. 13, many variations and alternatives are possible.

[0161] Although not shown for ease of illustration in FIG. 13, in some embodiments, the system 1300 may include some or all of the components and/or functionality described above with reference to FIGs. 1A-1B and 2A-2C.

[0162] Referring now to FIG. 14, a block diagram of a representative computer system 1400 such as notebook, Ultrabook<sup>TM</sup> or other small form factor system. A processor 1410, in one embodiment, includes a microprocessor, multi-core processor, multithreaded processor, an ultra low voltage processor, an embedded processor, or other known processing element. In the illustrated implementation, processor 1410 acts as a main processing unit and central hub for communication with many of the various components of the system 1400. As one example, processor 1410 is implemented as a SoC.

[0163] Processor 1410, in one embodiment, communicates with a system memory 1415. As an illustrative example, the system memory 1415 is implemented via multiple memory devices or modules to provide for a given amount of system memory.

[0164] To provide for persistent storage of information such as data, applications, one or more operating systems and so forth, a mass storage 1420 may also couple to processor 1410. In various embodiments, to enable a thinner and lighter system design as well as to improve system responsiveness, this mass storage may be implemented via a SSD or the mass storage may primarily be implemented using a hard disk drive (HDD) with a smaller amount of SSD storage to act as a SSD cache to enable non-volatile storage of context state and other such information during power down events so that a fast power up can occur on re-initiation of system activities. Also shown in FIG. 14, a flash device 1422 may be coupled to processor 1410, e.g., via a serial peripheral interface (SPI). This flash device may provide for non-volatile storage of system software, including a basic input/output software (BIOS) as well as other firmware of the system.

[0165] Various input/output (I/O) devices may be present within system 1400. Specifically shown in the embodiment of FIG. 14 is a display 1424 which may be a high definition LCD or LED panel that further provides for a touch screen 1425. In one embodiment, display 1424 may be coupled to processor 1410 via a display interconnect that can be implemented as a high performance graphics interconnect. Touch screen 1425 may be coupled to processor 1410 via another interconnect, which in an embodiment can be an I<sup>2</sup>C interconnect. As further shown in FIG. 14, in addition to touch screen 1425, user input by way of touch can also occur via a touch pad 1430 which may be configured within the chassis and may also be coupled to the same I<sup>2</sup>C interconnect as touch screen 1425.

[0166] For perceptual computing and other purposes, various sensors may be present within the system and may be coupled to processor 1410 in different manners. Certain inertial and environmental sensors may couple to processor 1410 through a sensor hub 1440, e.g., via an I<sup>2</sup>C interconnect. In the embodiment shown in FIG. 14, these sensors may include an accelerometer 1441, an ambient light sensor (ALS) 1442, a compass 1443 and a gyroscope 1444. Other environmental sensors may include one or more thermal sensors 1446 which in some embodiments couple to processor 1410 via a system management bus (SMBus) bus.

[0167] Also seen in FIG. 14, various peripheral devices may couple to processor 1410 via a low pin count (LPC) interconnect. In the embodiment shown, various components can be coupled through an embedded controller 1435. Such components can include a keyboard 1436 (e.g., coupled via a PS2 interface), a fan 1437, and a thermal sensor 1439. In some

embodiments, touch pad 1430 may also couple to EC 1435 via a PS2 interface. In addition, a security processor such as a trusted platform module (TPM) 1438 in accordance with the Trusted Computing Group (TCG) TPM Specification Version 1.2, dated Oct. 2, 2003, may also couple to processor 1410 via this LPC interconnect.

[0168] System 1400 can communicate with external devices in a variety of manners, including wirelessly. In the embodiment shown in FIG. 14, various wireless modules, each of which can correspond to a radio configured for a particular wireless communication protocol, are present. One manner for wireless communication in a short range such as a near field may be via a NFC unit 1445 which may communicate, in one embodiment with processor 1410 via an SMBus. Note that via this NFC unit 1445, devices in close proximity to each other can communicate.

[0169] As further seen in FIG. 14, additional wireless units can include other short range wireless engines including a WLAN unit 1450 and a Bluetooth unit 1452. Using WLAN unit 1450, Wi-Fi™ communications in accordance with a given IEEE 802.11 standard can be realized, while via Bluetooth unit 1452, short range communications via a Bluetooth protocol can occur. These units may communicate with processor 1410 via, e.g., a USB link or a universal asynchronous receiver transmitter (UART) link. Or these units may couple to processor 1410 via an interconnect according to a PCIe™ protocol or another such protocol such as a serial data input/output (SDIO) standard.

[0170] In addition, wireless wide area communications, e.g., according to a cellular or other wireless wide area protocol, can occur via a WWAN unit 1456 which in turn may couple to a subscriber identity module (SIM) 1457. In addition, to enable receipt and use of location information, a GPS module 1455 may also be present. Note that in the embodiment shown in FIG. 14, WWAN unit 1456 and an integrated capture device such as a camera module 1454 may communicate via a given USB protocol such as a USB 2.0 or 3.0 link, or a UART or I<sup>2</sup>C protocol.

[0171] An integrated camera module 1454 can be incorporated in the lid. To provide for audio inputs and outputs, an audio processor can be implemented via a digital signal processor (DSP) 1460, which may couple to processor 1410 via a high definition audio (HDA) link. Similarly, DSP 1460 may communicate with an integrated coder/decoder

(CODEC) and amplifier 1462 that in turn may couple to output speakers 1463 which may be implemented within the chassis. Similarly, amplifier and CODEC 1462 can be coupled to receive audio inputs from a microphone 1465 which in an embodiment can be implemented via dual array microphones (such as a digital microphone array) to provide for high quality audio inputs to enable voice-activated control of various operations within the system. Note also that audio outputs can be provided from amplifier/CODEC 1462 to a headphone jack 1464. Although shown with these particular components in the embodiment of FIG. 14, understand the scope of the present invention is not limited in this regard.

[0172] Although not shown for ease of illustration in FIG. 14, in some embodiments, the system 1400 may include some or all of the components and/or functionality described above with reference to FIGs. 1A-1B and 2A-2C.

[0173] Embodiments may be implemented in many different system types. Referring now to FIG. 15, shown is a block diagram of a system in accordance with an embodiment of the present invention. As shown in FIG. 15, multiprocessor system 1500 is a point-to-point interconnect system, and includes a first processor 1570 and a second processor 1580 coupled via a point-to-point interconnect 1550. As shown in FIG. 15, each of processors 1570 and 1580 may be multicore processors, including first and second processor cores (i.e., processor cores 1574a and 1574b and processor cores 1584a and 1584b), although potentially many more cores may be present in the processors. Each of the processors can include a PCU or other power management logic to perform processor-based power management as described herein.

[0174] Still referring to FIG. 15, first processor 1570 further includes a memory controller hub (MCH) 1572 and point-to-point (P-P) interfaces 1576 and 1578. Similarly, second processor 1580 includes a MCH 1582 and P-P interfaces 1586 and 1588. As shown in FIG. 15, MCH's 1572 and 1582 couple the processors to respective memories, namely a memory 1532 and a memory 1534, which may be portions of system memory (e.g., DRAM) locally attached to the respective processors. First processor 1570 and second processor 1580 may be coupled to a chipset 1590 via P-P interconnects 1562 and 1564, respectively. As shown in FIG. 15, chipset 1590 includes P-P interfaces 1594 and 1598.

[0175] Furthermore, chipset 1590 includes an interface 1592 to couple chipset 1590 with a high performance graphics engine 1538, by a P-P interconnect 1539. In turn, chipset 1590 may be coupled to a first bus 1516 via an interface 1596. As shown in FIG. 15, various input/output (I/O) devices 1514 may be coupled to first bus 1516, along with a bus bridge 1518 which couples first bus 1516 to a second bus 1520. Various devices may be coupled to second bus 1520 including, for example, a keyboard/mouse 1522, communication devices 1526 and a data storage unit 1528 such as a disk drive or other mass storage device which may include code 1530, in one embodiment. Further, an audio I/O 1524 may be coupled to second bus 1520. Embodiments can be incorporated into other types of systems including mobile devices such as a smart cellular telephone, tablet computer, netbook, Ultrabook™, or so forth.

[0176] Although not shown for ease of illustration in FIG. 15, in some embodiments, the system 1500 may include some or all of the components and/or functionality described above with reference to FIGs. 1A-1B and 2A-2C.

[0177] Embodiments may be implemented in code and may be stored on a non-transitory storage medium having stored thereon instructions which can be used to program a system to perform the instructions. The storage medium may include, but is not limited to, any type of disk including floppy disks, optical disks, solid state drives (SSDs), compact disk read-only memories (CD-ROMs), compact disk rewritables (CD-RWs), and magneto-optical disks, semiconductor devices such as read-only memories (ROMs), random access memories (RAMs) such as dynamic random access memories (DRAMs), static random access memories (SRAMs), erasable programmable read-only memories (EPROMs), flash memories, electrically erasable programmable read-only memories (EEPROMs), magnetic or optical cards, or any other type of media suitable for storing electronic instructions.

[0178] The following clauses and/or examples pertain to further embodiments.

[0179] In one example, a processor for managing memory requests comprises a plurality of cores and synchronization logic. The synchronization logic comprises circuitry to: receive a first memory request and a second memory request; determine whether the second memory request is in contention with the first memory request; and in response to a determination that

the second memory request is in contention with the first memory request, process the second memory request using a non-blocking cache coherence protocol.

[0180] In an example, the circuitry is further to: in response to a determination that the second memory request is not in contention with the first memory request, process the second memory request using a blocking cache coherence protocol.

[0181] In an example, the first memory request is received from a first core; the second memory request is received from a second core; the first memory request and the second memory request are in contention for a first memory address; and at a time when the first memory request is received, the first memory address is cached in a third core. In an example, the circuitry is further to: send, to the third core, a blocking-protocol invalidation message associated with the first request; and send, to the first core, a non-blocking-protocol invalidation message associated with the second request without waiting for a data response from the third core. In an example, the circuitry is further to: receive, from the third core, a data response associated with the first request; and in response to a receipt of the data response from the third core, send, to the first core, a non-blocking-protocol acknowledgement message associated with the first request. In an example, the circuitry is further to: receive, from the first core, a data response associated with the second request; and in response to a receipt of the data response from the first core, send, to the second core, a non-blocking-protocol acknowledgement message associated with the second request. In an example, the first core comprises cache logic implemented in circuitry, where the cache logic of the first core is to: receive the non-blocking-protocol invalidation message associated with the second request; and store the non-blocking-protocol invalidation message associated with the second request in a memory buffer of the cache logic until a receipt of the non-blocking-protocol acknowledgement message associated with the first request. In an example, the cache logic of the first core is to further to, in response to a receipt of the non-blocking-protocol acknowledgement message associated with the first request: process data included in the received non-blocking-protocol acknowledgement message to generate processed data; send a data response including the processed data; and process the stored non-blocking-protocol invalidation message to invalidate a local copy of the processed data.

[0182] In one example, a method for managing memory requests comprises: receiving a first memory request from a first core of a processor; receiving a second memory request

from a second core of the processor; and in response to a determination that the second memory request is in contention with the first memory request, processing the second memory request using a non-blocking cache coherence protocol.

[0183] In an example, the method further comprises: in response to a determination that the second memory request is not in contention with the first memory request, processing the second memory request using a blocking cache coherence protocol, where the second memory request is not processed until the first memory request is completed.

[0184] In an example, the first memory request and the second memory request are in contention for a first memory address; and, when the first memory request is received, the first memory address is cached in a third core. In an example, the method further comprises: sending, to the third core, a blocking-protocol invalidation message associated with the first request; and sending, to the first core, a non-blocking-protocol invalidation message associated with the second request without waiting for a data response from the third core. In an example, the method further comprises: receiving, from the third core, a data response associated with the first request; and in response to the data response from the third core, sending, to the first core, a non-blocking-protocol acknowledgement message associated with the first request. In an example, the method further comprises: receiving, from the first core, a data response associated with the second request; and in response to the data response from the first core, sending, to the second core, a non-blocking-protocol acknowledgement message associated with the second request.

[0185] In one example, a machine readable medium has stored thereon data, which if used by at least one machine, causes the at least one machine to fabricate at least one integrated circuit to perform a method according to any one of the above examples.

[0186] In one example, an apparatus for processing instructions, is configured to perform the method of any one of the above examples.

[0187] In one example, a system for managing memory requests comprises: a processor comprising a plurality of cores and synchronization logic, and a system memory coupled to the processor. The synchronization logic comprises circuitry to: receive a first memory request and a second memory request; determine whether the second memory request is in contention with the first memory request; and in response to a determination that the second

memory request is in contention with the first memory request, process the second memory request using a non-blocking cache coherence protocol.

[0188] In an example, the circuitry is further to: in response to a determination that the second memory request is not in contention with the first memory request, process the second memory request using a blocking cache coherence protocol.

[0189] In an example, the first memory request is received from a first core; the second memory request is received from a second core; the first memory request and the second memory request are in contention for a first memory address; and at a time when the first memory request is received, the first memory address is cached in a third core. In an example, the circuitry is further to: send, to the third core, a blocking-protocol invalidation message associated with the first request; and send, to the first core, a non-blocking-protocol invalidation message associated with the second request without waiting for a data response from the third core. In an example, the circuitry is further to: receive, from the third core, a data response associated with the first request; and in response to the data response from the third core, send, to the first core, a non-blocking-protocol acknowledgement message associated with the first request. In an example, the first core comprises cache logic implemented in circuitry. The cache logic of the first core is to: receive the non-blocking-protocol invalidation message associated with the second request; and store the non-blocking-protocol invalidation message associated with the second request in a buffer of the cache logic until a receipt of the non-blocking-protocol acknowledgement message associated with the first request.

[0190] In one example, a machine-readable medium has stored thereon data, which if used by at least one machine, causes the at least one machine to fabricate at least one integrated circuit to perform a method. The method comprises: receiving a first memory request from a first core of a processor; receiving a second memory request from a second core of the processor; and in response to a determination that the second memory request is in contention with the first memory request, processing the second memory request using a non-blocking cache coherence protocol.

[0191] In an example, the method further comprises: in response to a determination that the second memory request is not in contention with the first memory request, processing the

second memory request using a blocking cache coherence protocol, where the second memory request is not processed until the first memory request is completed.

[0192] In an example, the first memory request and the second memory request are in contention for a first memory address; and, when the first memory request is received, the first memory address is cached in a third core. In an example, the method further comprises: sending, to the third core, a blocking-protocol invalidation message associated with the first request; and sending, to the first core, a non-blocking-protocol invalidation message associated with the second request without waiting for a data response from the third core. In an example, the method further comprises: receiving, from the third core, a data response associated with the first request; and in response to the data response from the third core, sending, to the first core, a non-blocking-protocol acknowledgement message associated with the first request. In an example, the method further comprises: receiving, from the first core, a data response associated with the second request; and in response to the data response from the first core, sending, to the second core, a non-blocking-protocol acknowledgement message associated with the second request.

[0193] Understand that various combinations of the above examples are possible. Embodiments may be used in many different types of systems. For example, in one embodiment a communication device can be arranged to perform the various methods and techniques described herein. Of course, the scope of the present invention is not limited to a communication device, and instead other embodiments can be directed to other types of apparatus for processing instructions, or one or more machine readable media including instructions that in response to being executed on a computing device, cause the device to carry out one or more of the methods and techniques described herein.

[0194] References throughout this specification to “one embodiment” or “an embodiment” mean that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one implementation encompassed within the present invention. Thus, appearances of the phrase “one embodiment” or “in an embodiment” are not necessarily referring to the same embodiment. Furthermore, the particular features, structures, or characteristics may be instituted in other suitable forms other than the particular embodiment illustrated and all such forms may be encompassed within the claims of the present application.

[0195] While the present invention has been described with respect to a limited number of embodiments, those skilled in the art will appreciate numerous modifications and variations therefrom. It is intended that the appended claims cover all such modifications and variations as fall within the true spirit and scope of this present invention.

What is claimed is:

- 1 1. A processor comprising:  
2 a plurality of cores; and  
3 synchronization logic comprising circuitry, the circuitry to:  
4 receive a first memory request and a second memory request;  
5 determine whether the second memory request is in contention with the first  
6 memory request; and  
7 in response to a determination that the second memory request is in contention  
8 with the first memory request, process the second memory request using a non-blocking  
9 cache coherence protocol.
- 1 2. The processor of claim 1, the circuitry further to:  
2 in response to a determination that the second memory request is not in contention  
3 with the first memory request, process the second memory request using a blocking cache  
4 coherence protocol.
- 1 3. The processor of claim 1, wherein:  
2 the first memory request is received from a first core;  
3 the second memory request is received from a second core;  
4 the first memory request and the second memory request are in contention for a first  
5 memory address; and  
6 at a time when the first memory request is received, the first memory address is  
7 cached in a third core.
- 1 4. The processor of claim 3, the circuitry further to:  
2 send, to the third core, a blocking-protocol invalidation message associated with the  
3 first request; and  
4 send, to the first core, a non-blocking-protocol invalidation message associated with  
5 the second request without waiting for a data response from the third core.
- 1 5. The processor of claim 4, the circuitry further to:  
2 receive, from the third core, a data response associated with the first request; and

3           in response to a receipt of the data response from the third core, send, to the first core,  
4   a non-blocking-protocol acknowledgement message associated with the first request.

1   6.     The processor of claim 5, the circuitry further to:  
2           receive, from the first core, a data response associated with the second request; and  
3           in response to a receipt of the data response from the first core, send, to the second  
4   core, a non-blocking-protocol acknowledgement message associated with the second request.

1   7.     The processor of claim 5, the first core comprising cache logic implemented in  
2   circuitry, the cache logic of the first core to:  
3           receive the non-blocking-protocol invalidation message associated with the second  
4   request; and  
5           store the non-blocking-protocol invalidation message associated with the second  
6   request in a memory buffer of the cache logic until a receipt of the non-blocking-protocol  
7   acknowledgement message associated with the first request.

1   8.     The processor of claim 7, the cache logic of the first core further to:  
2           in response to a receipt of the non-blocking-protocol acknowledgement message  
3   associated with the first request:  
4                 process data included in the received non-blocking-protocol acknowledgement  
5   message to generate processed data;  
6                 send a data response including the processed data; and  
7                 process the stored non-blocking-protocol invalidation message to invalidate a  
8   local copy of the processed data.

1   9.     A method comprising:  
2           receiving a first memory request from a first core of a processor;  
3           receiving a second memory request from a second core of the processor; and  
4           in response to a determination that the second memory request is in contention with  
5   the first memory request, processing the second memory request using a non-blocking cache  
6   coherence protocol.

- 1    10.    The method of claim 9, further comprising:  
2            in response to a determination that the second memory request is not in contention  
3    with the first memory request, processing the second memory request using a blocking cache  
4    coherence protocol, wherein the second memory request is not processed until the first  
5    memory request is completed.
- 1    11.    The method of claim 9, wherein the first memory request and the second memory  
2    request are in contention for a first memory address; and wherein, when the first memory  
3    request is received, the first memory address is cached in a third core.
- 1    12.    The method of claim 11, further comprising:  
2            sending, to the third core, a blocking-protocol invalidation message associated with  
3    the first request; and  
4            sending, to the first core, a non-blocking-protocol invalidation message associated  
5    with the second request without waiting for a data response from the third core.
- 1    13.    The method of claim 12, further comprising:  
2            receiving, from the third core, a data response associated with the first request; and  
3            in response to the data response from the third core, sending, to the first core, a non-  
4    blocking-protocol acknowledgement message associated with the first request.
- 1    14.    The method of claim 13, further comprising:  
2            receiving, from the first core, a data response associated with the second request; and  
3            in response to the data response from the first core, sending, to the second core, a non-  
4    blocking-protocol acknowledgement message associated with the second request.
- 1    15.    A machine readable medium having stored thereon data, which if used by at least one  
2    machine, causes the at least one machine to fabricate at least one integrated circuit to perform  
3    a method according to any one of claims 9 to 14.
- 1    16.    An apparatus for processing instructions, configured to perform the method of any  
2    one of claims 9 to 14.

- 1 17. A system comprising:  
2 a processor comprising a plurality of cores and synchronization logic, the  
3 synchronization logic comprising circuitry to:  
4 receive a first memory request and a second memory request;  
5 determine whether the second memory request is in contention with the first  
6 memory request; and  
7 in response to a determination that the second memory request is in contention  
8 with the first memory request, process the second memory request using a non-blocking  
9 cache coherence protocol; and  
10 a system memory coupled to the processor.
- 1 18. The system of claim 17, the circuitry further to:  
2 in response to a determination that the second memory request is not in contention  
3 with the first memory request, process the second memory request using a blocking cache  
4 coherence protocol.
- 1 19. The system of claim 17, wherein:  
2 the first memory request is received from a first core;  
3 the second memory request is received from a second core;  
4 the first memory request and the second memory request are in contention for a first  
5 memory address; and  
6 at a time when the first memory request is received, the first memory address is  
7 cached in a third core.
- 1 20. The system of claim 19, the circuitry further to:  
2 send, to the third core, a blocking-protocol invalidation message associated with the  
3 first request; and  
4 send, to the first core, a non-blocking-protocol invalidation message associated with  
5 the second request without waiting for a data response from the third core.

- 1    21.    The system of claim 20, the circuitry further to:  
2           receive, from the third core, a data response associated with the first request; and  
3           in response to the data response from the third core, send, to the first core, a non-  
4 blocking-protocol acknowledgement message associated with the first request.
- 1    22.    The system of claim 21, the first core comprising cache logic implemented in  
2 circuitry, the cache logic of the first core to:  
3           receive the non-blocking-protocol invalidation message associated with the second  
4 request; and  
5           store the non-blocking-protocol invalidation message associated with the second  
6 request in a buffer of the cache logic until a receipt of the non-blocking-protocol  
7 acknowledgement message associated with the first request.

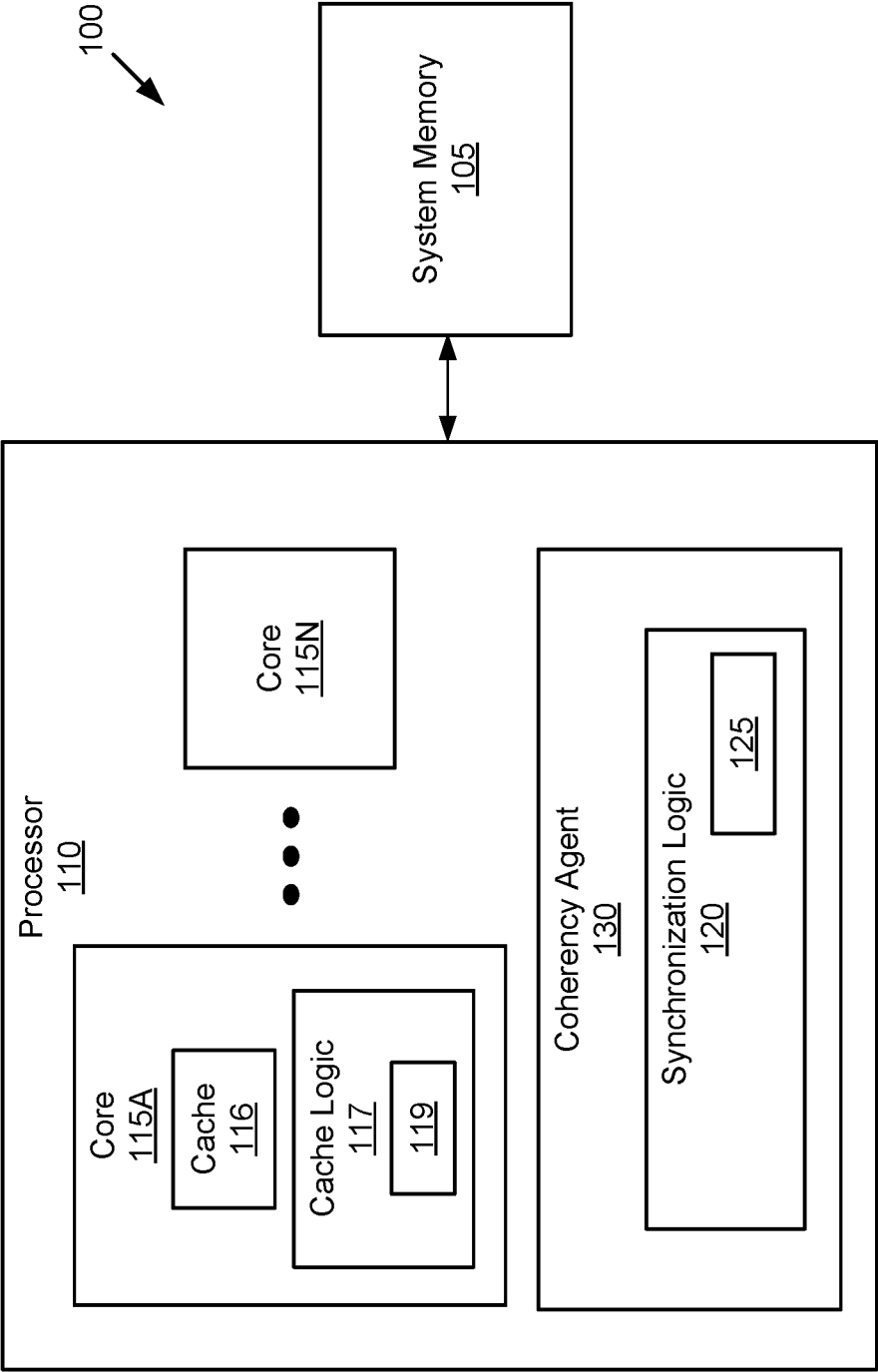


FIG. 1A

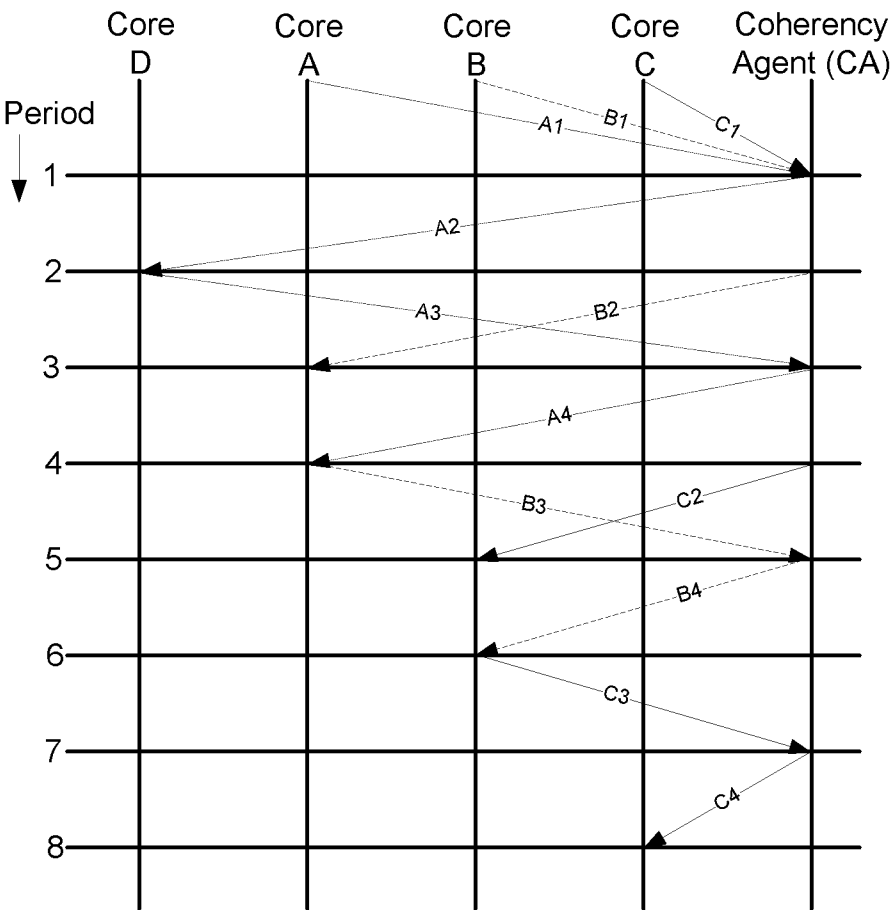


FIG. 1B

3/20

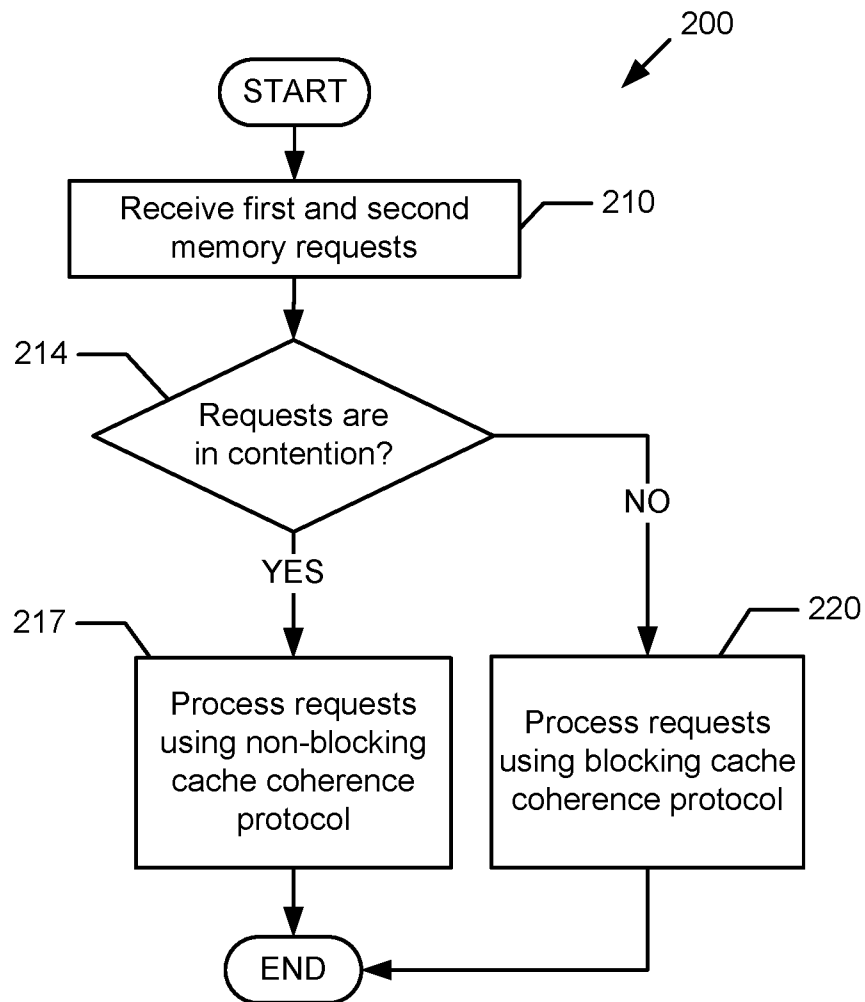


FIG. 2A

4/20

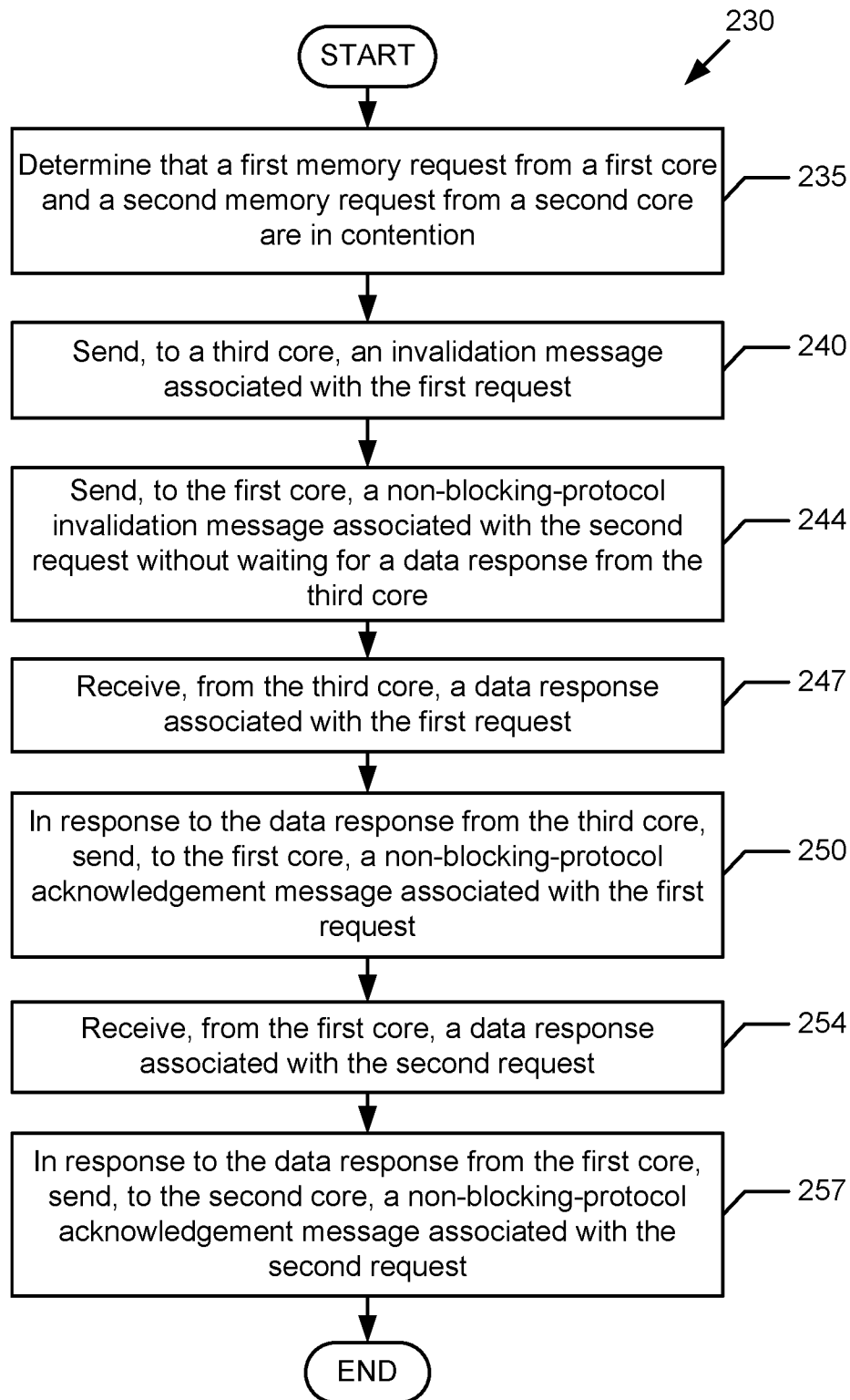


FIG. 2B

5/20

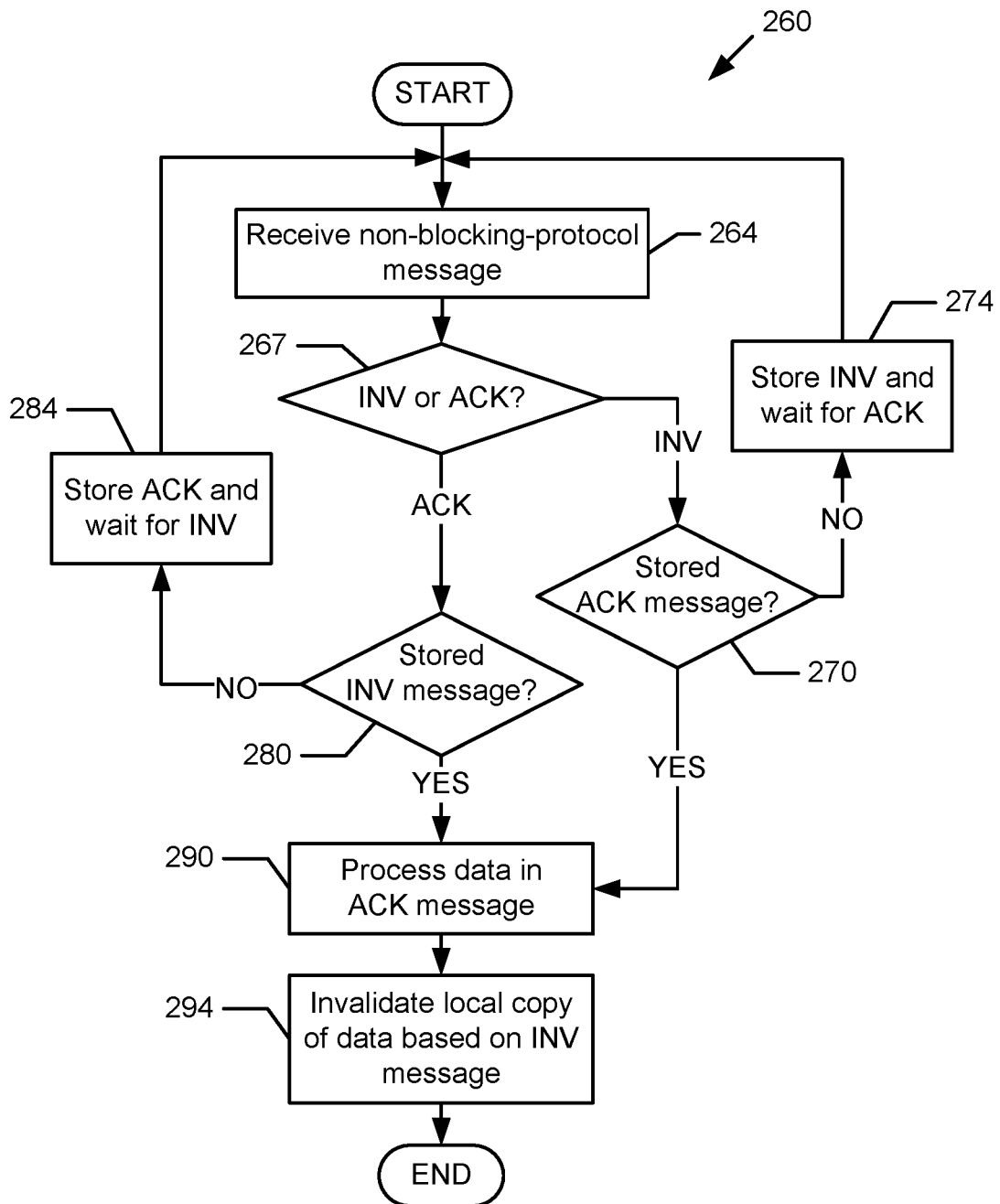


FIG. 2C

6/20

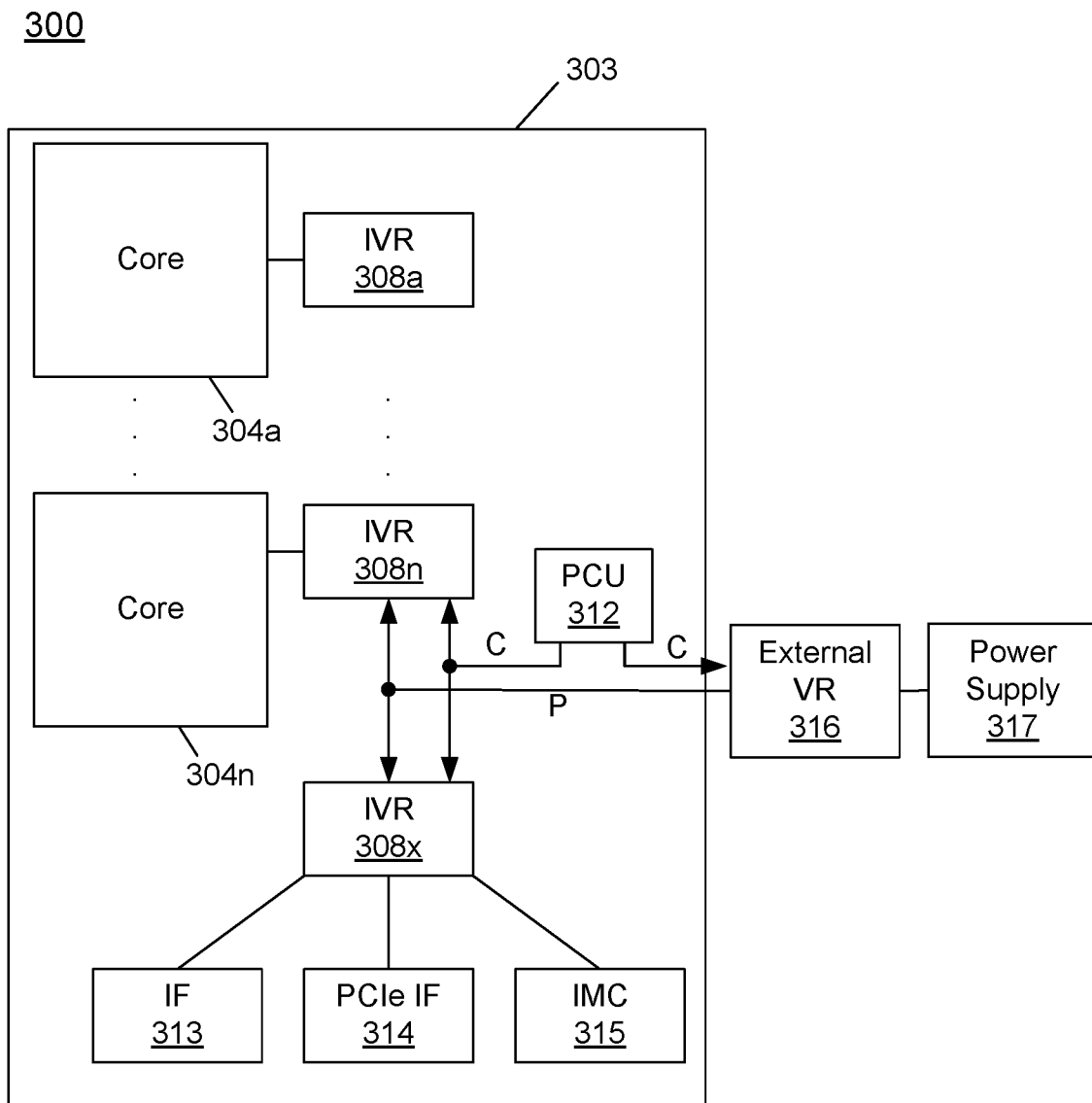


FIG. 3A

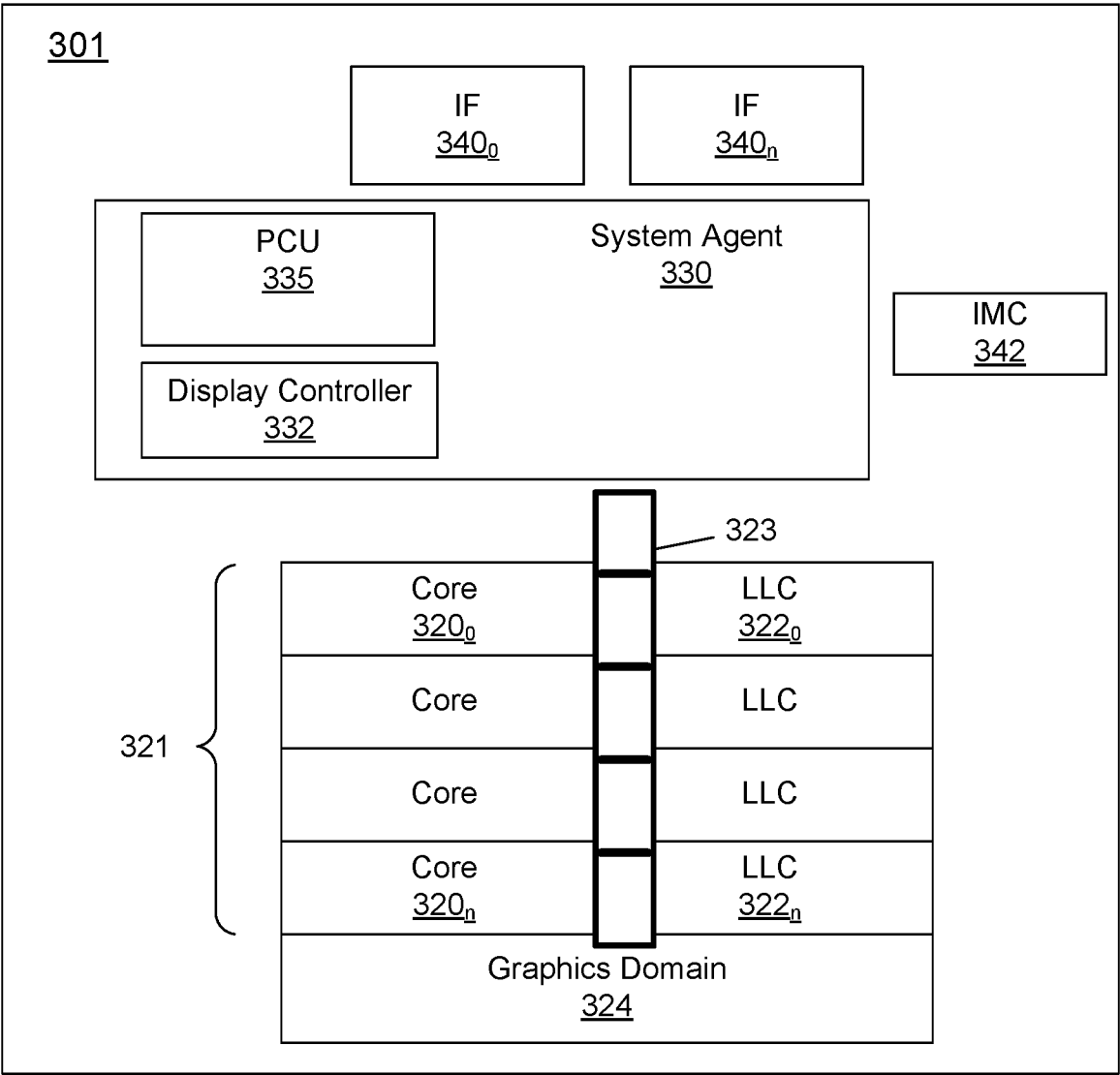


FIG. 3B

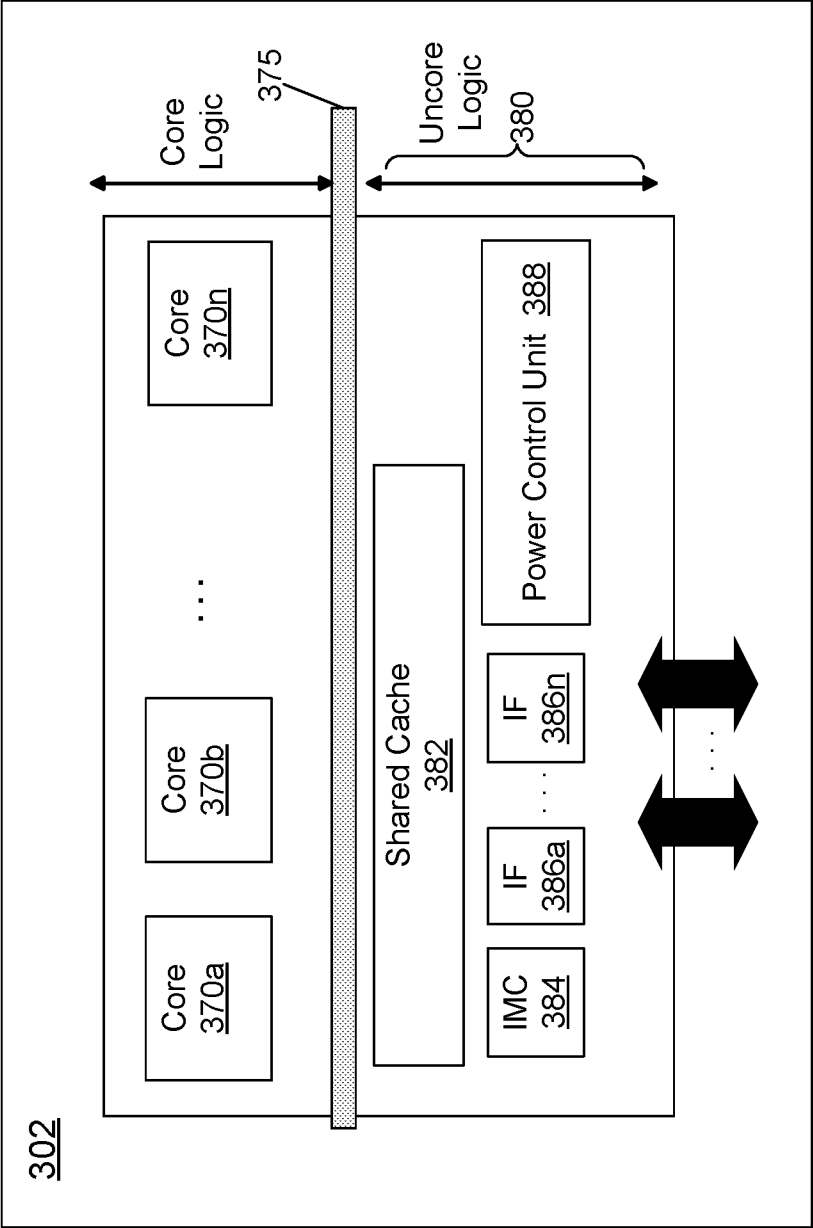


FIG. 3C

9/20

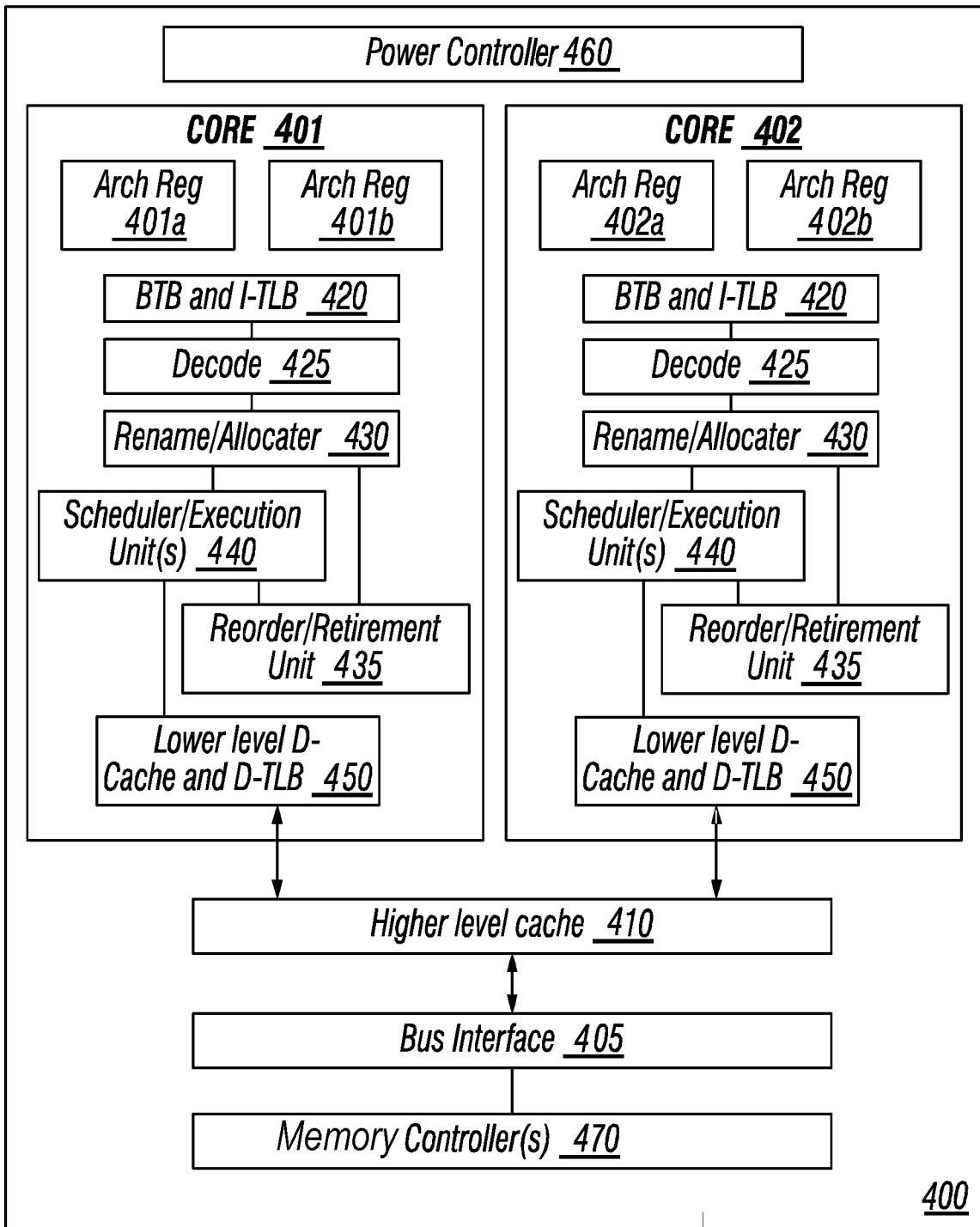


FIG. 4

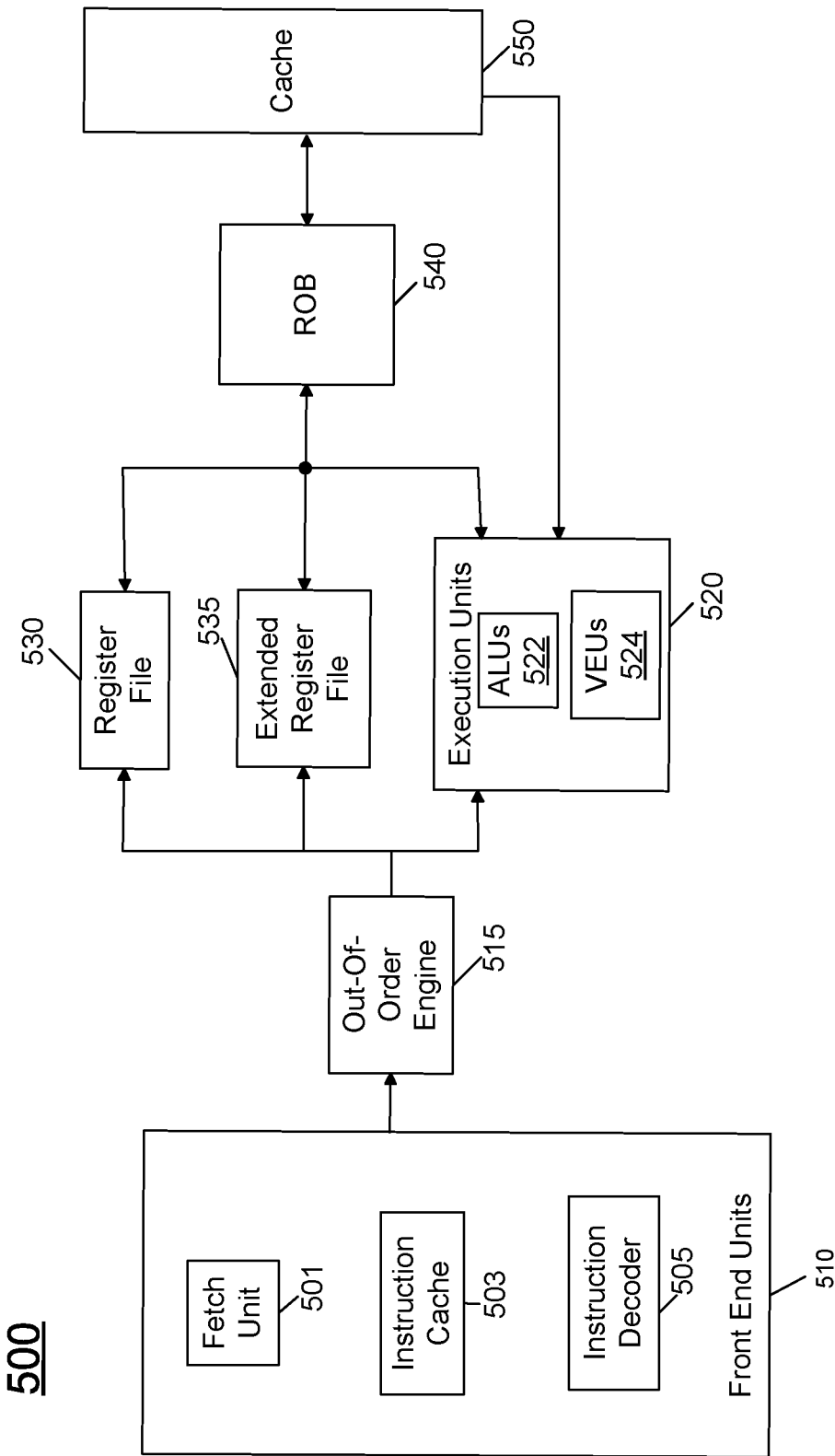


FIG. 5

11/20

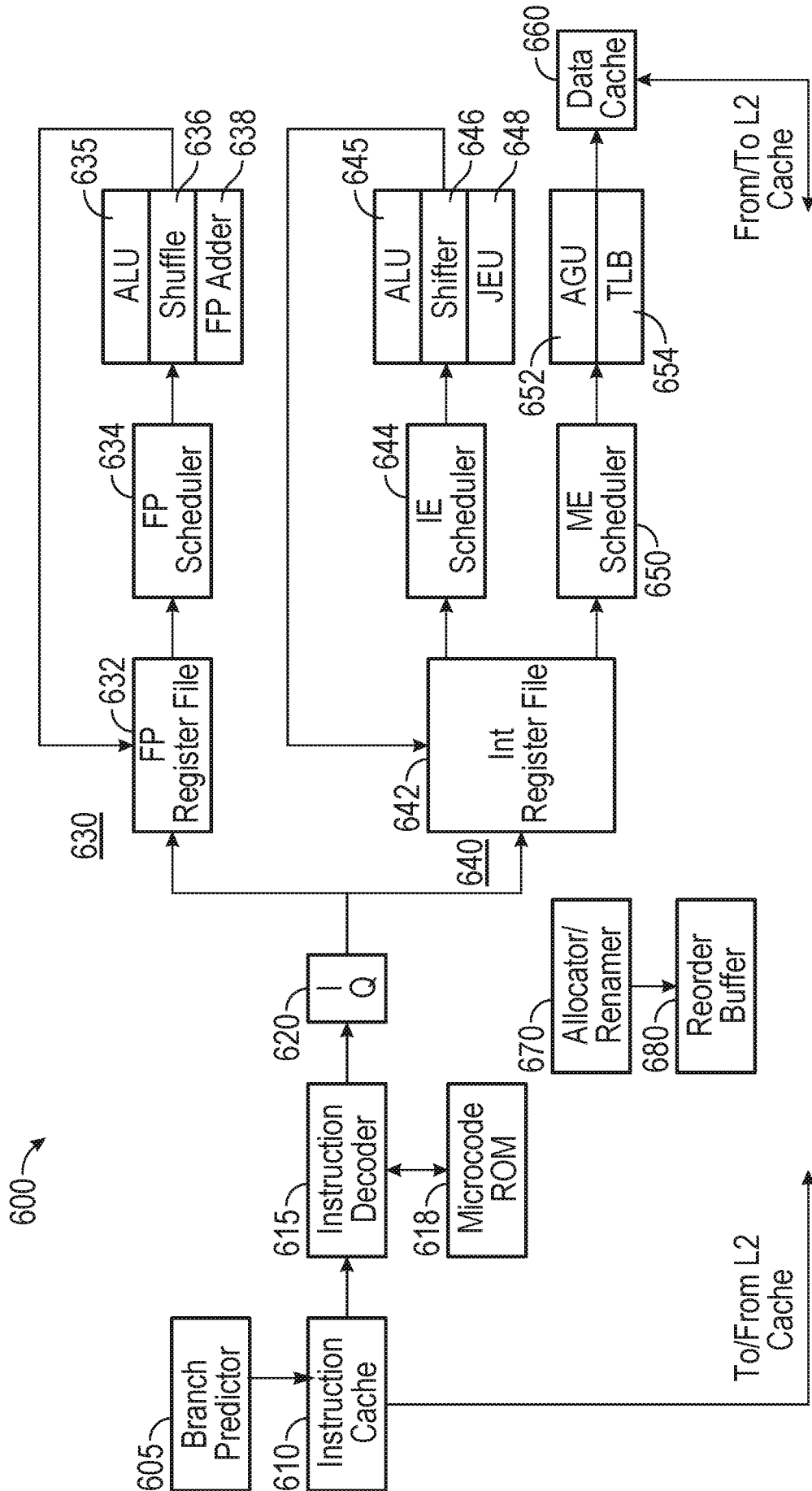


FIG. 6

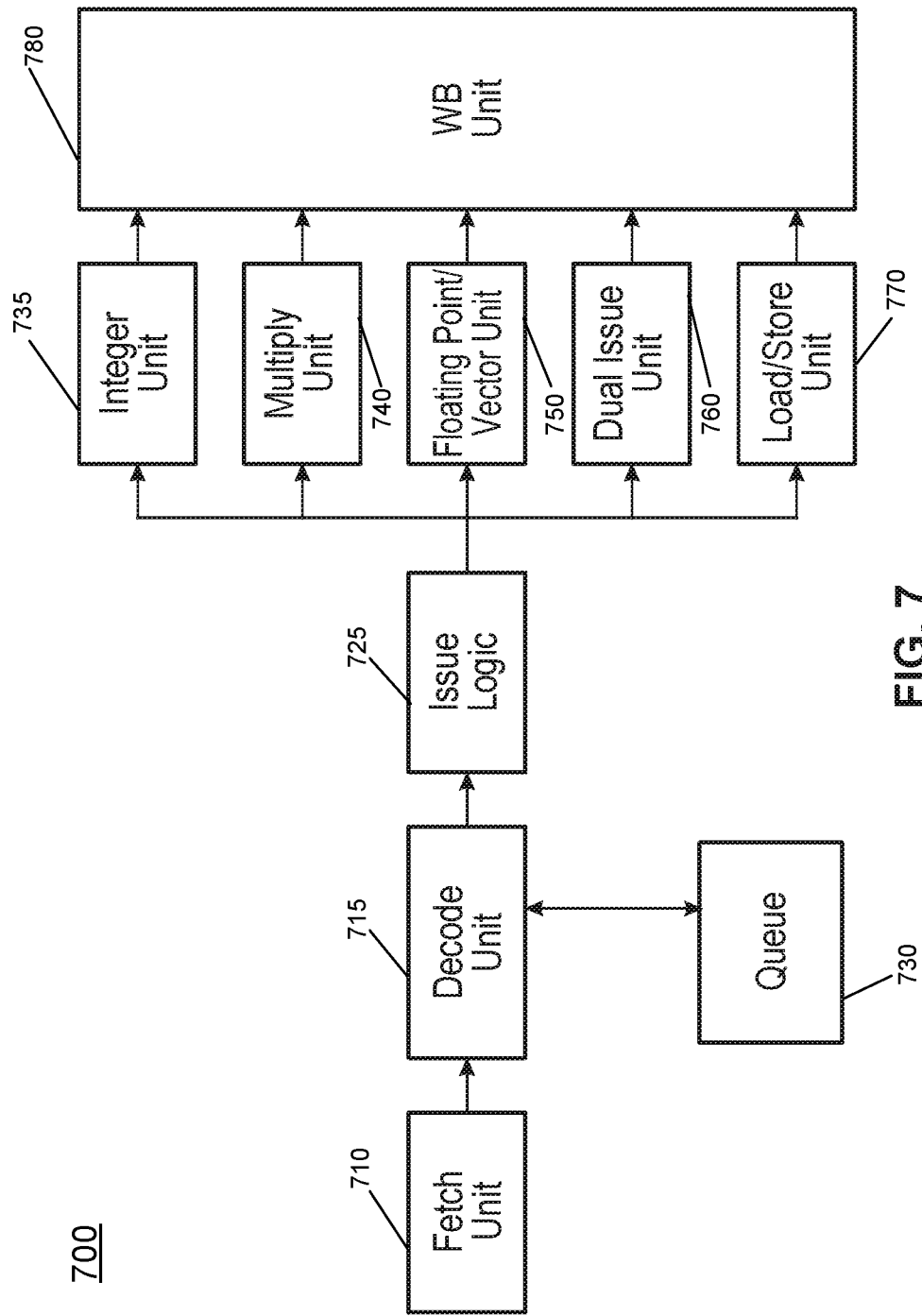
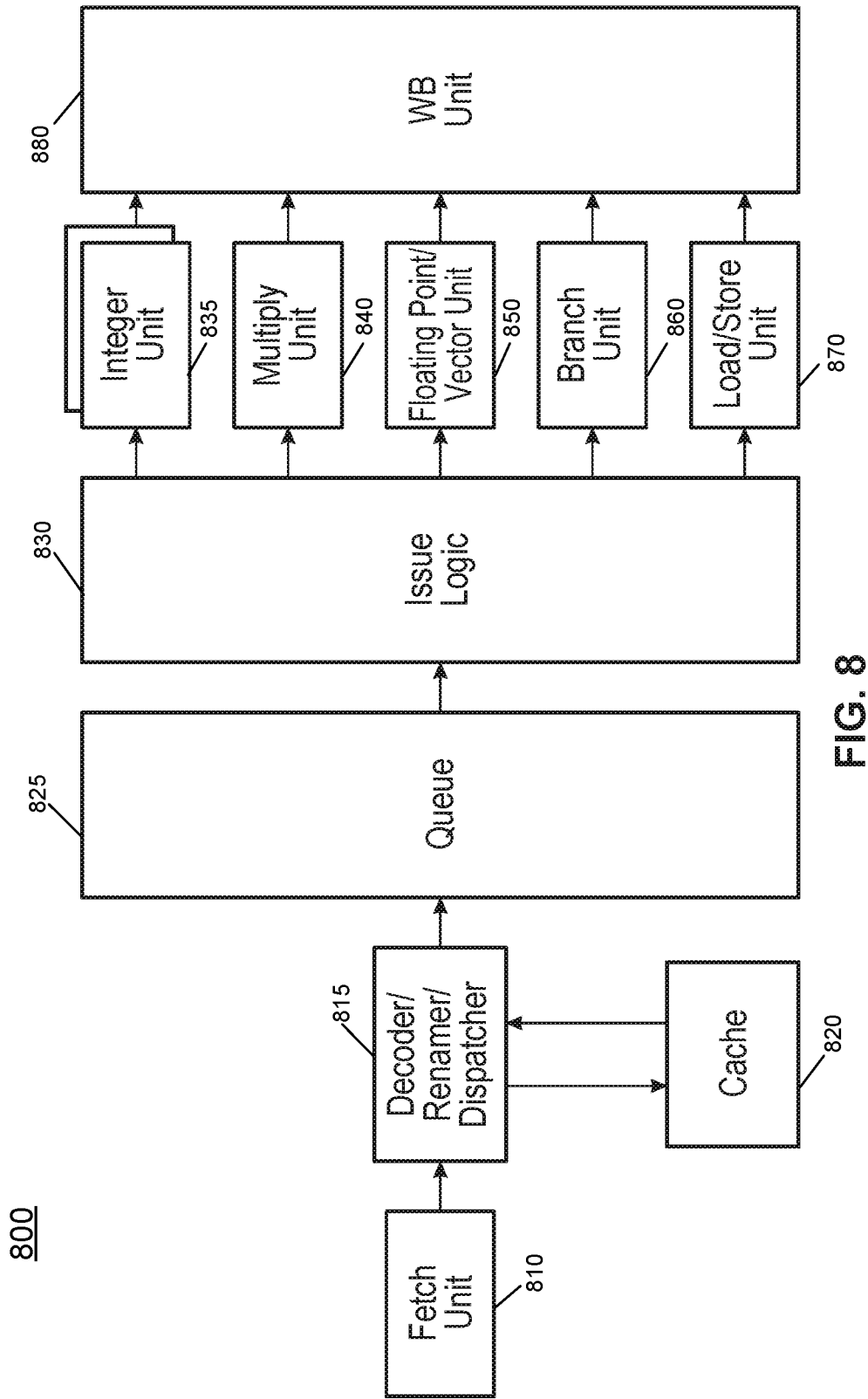


FIG. 7

700



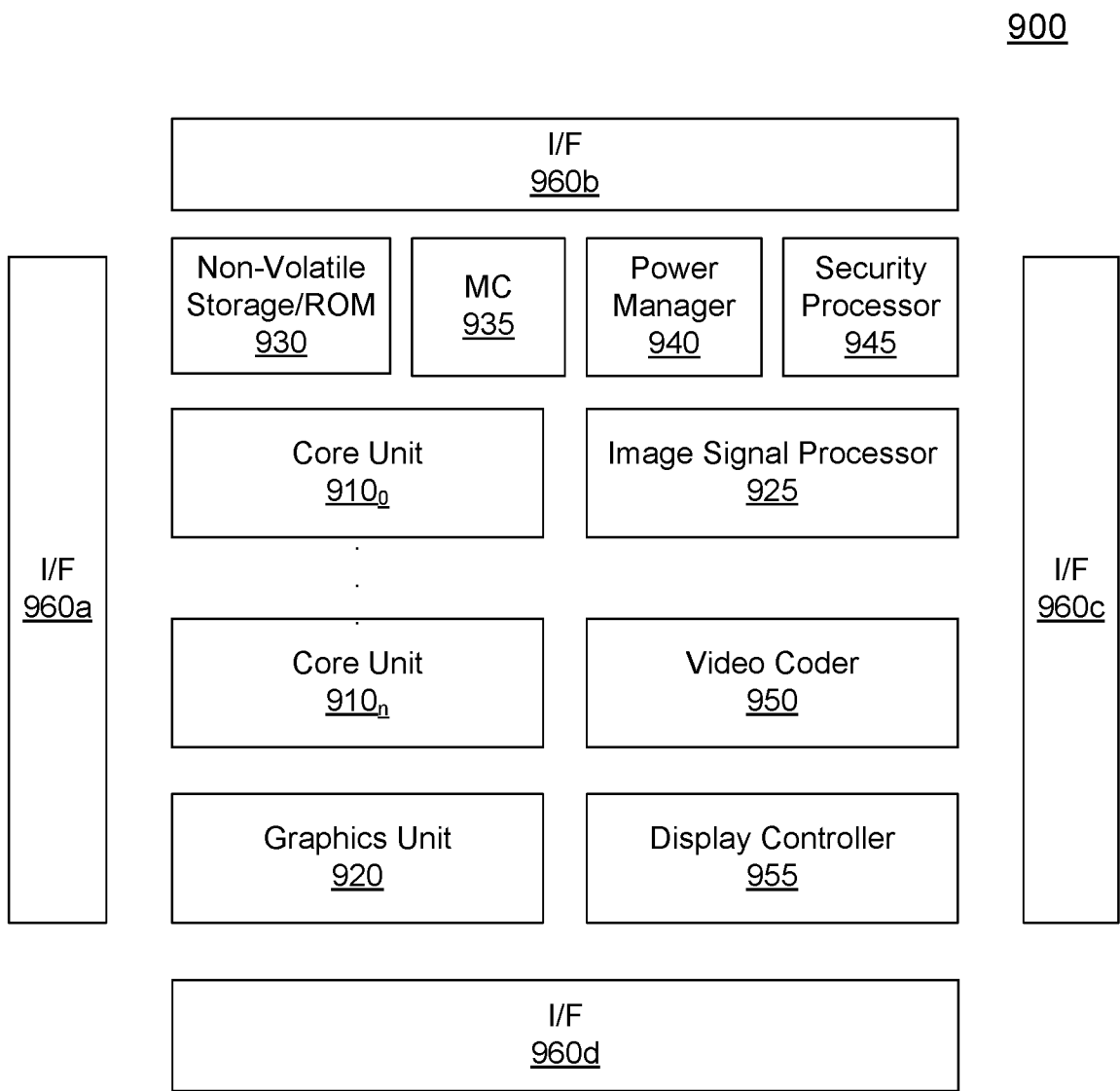


FIG. 9

1000

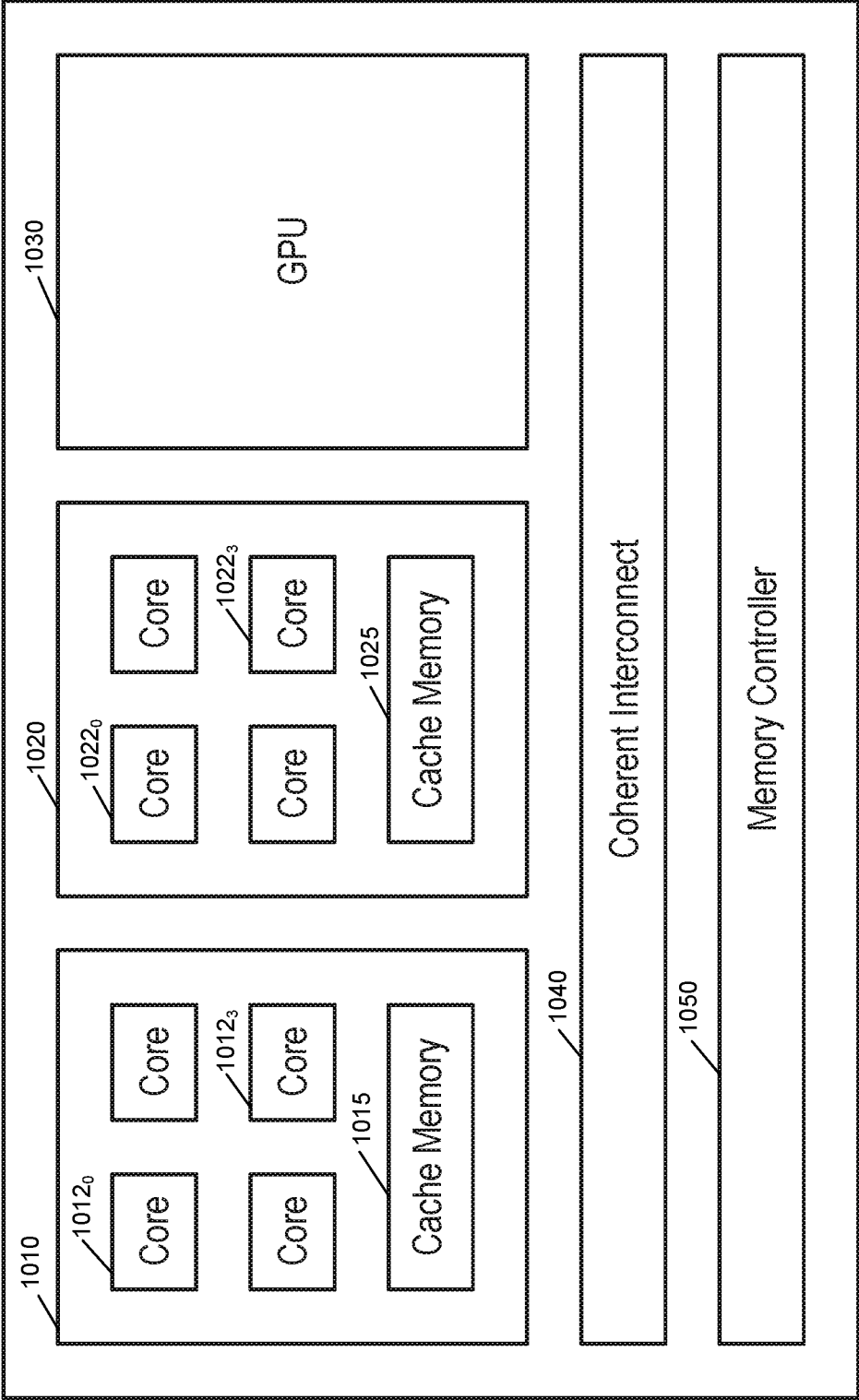


FIG. 10

16/20

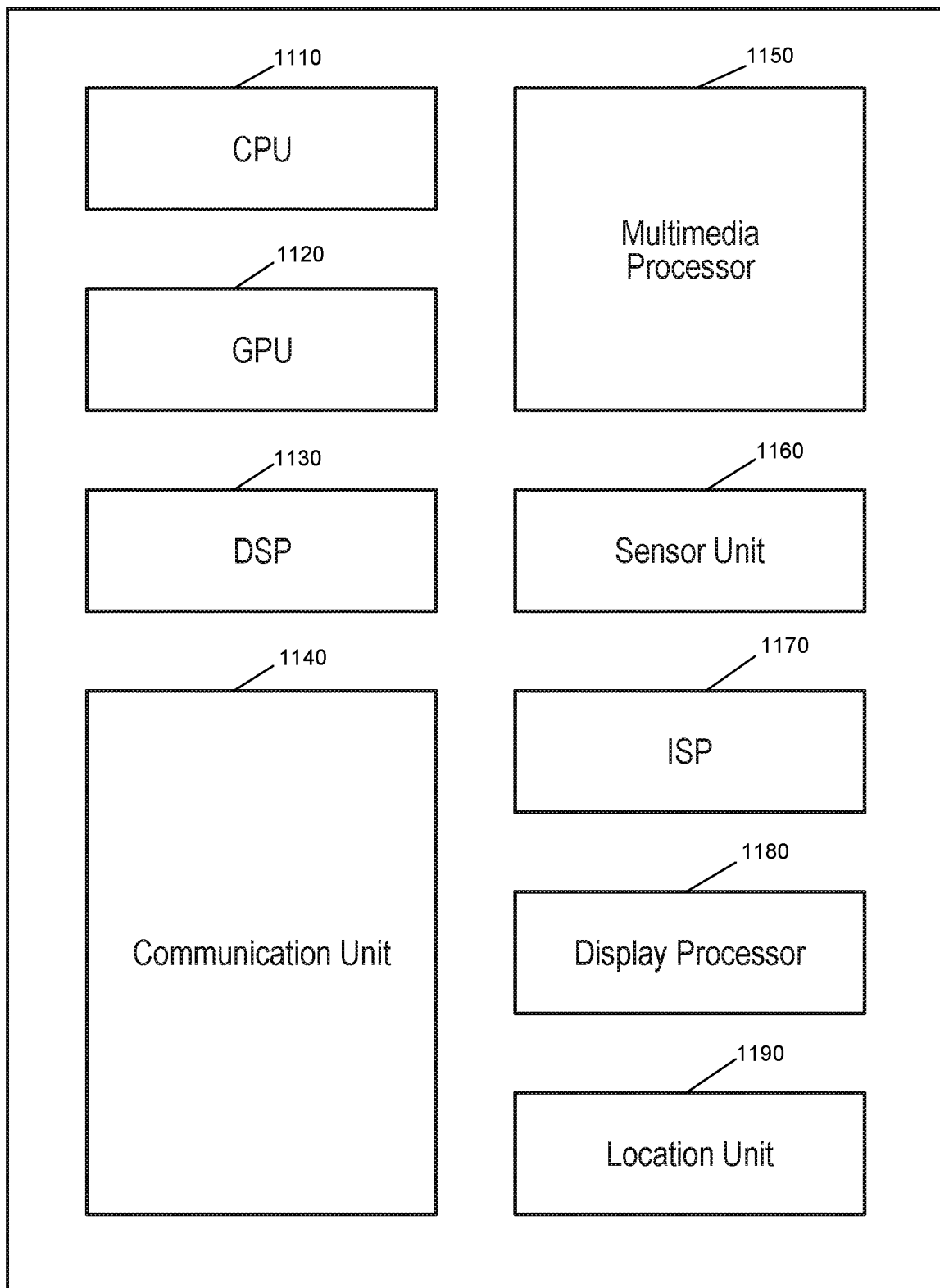
1100

FIG. 11

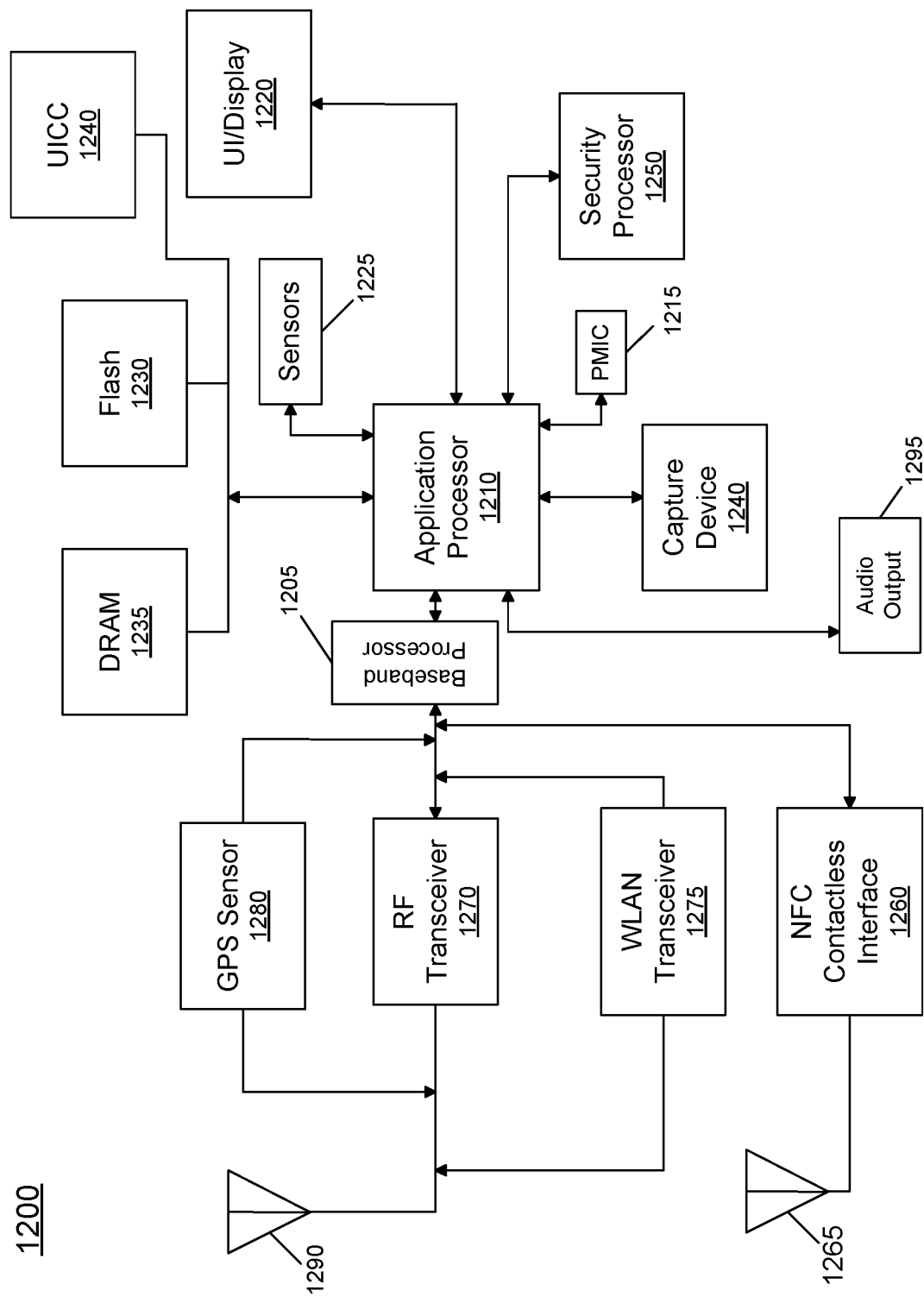


FIG. 12

1300

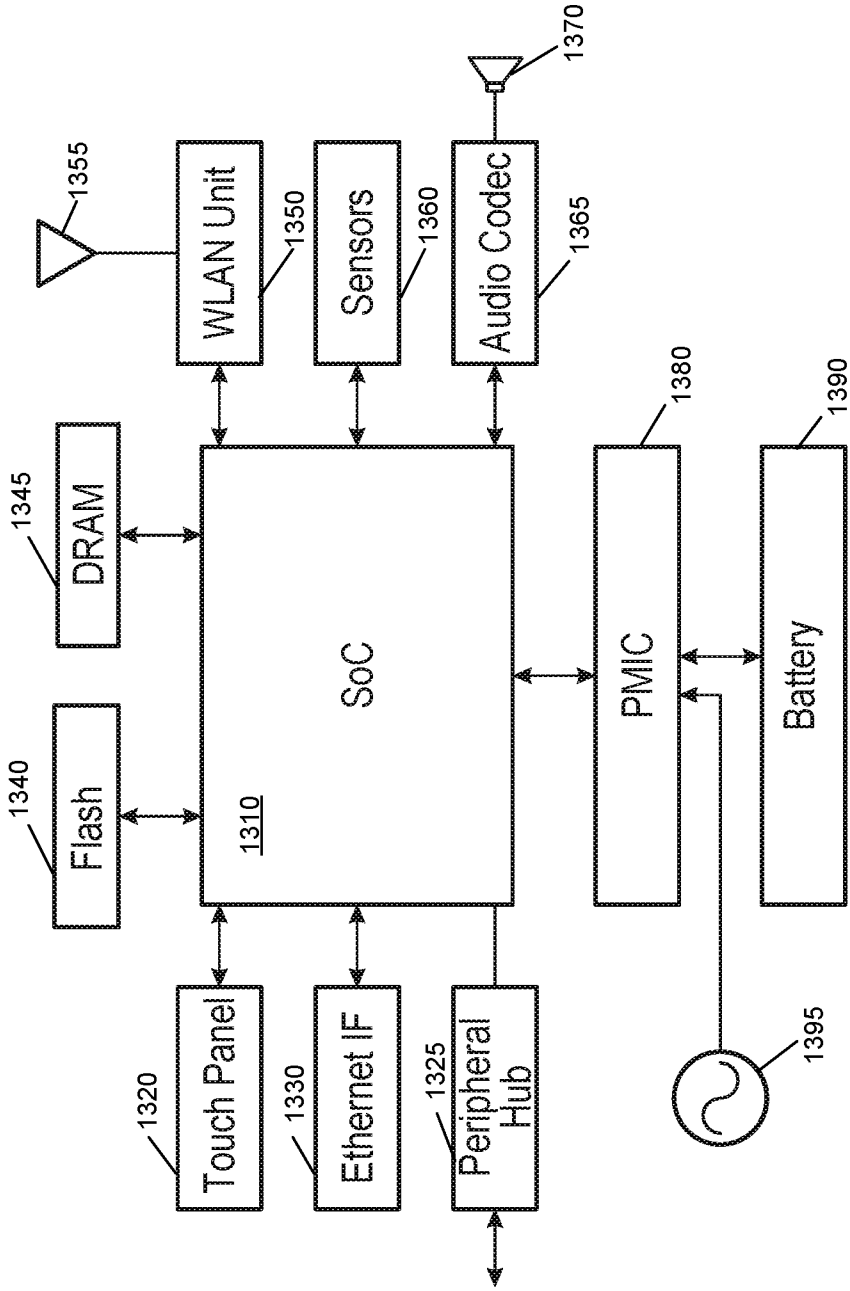


**FIG. 13**

19/20

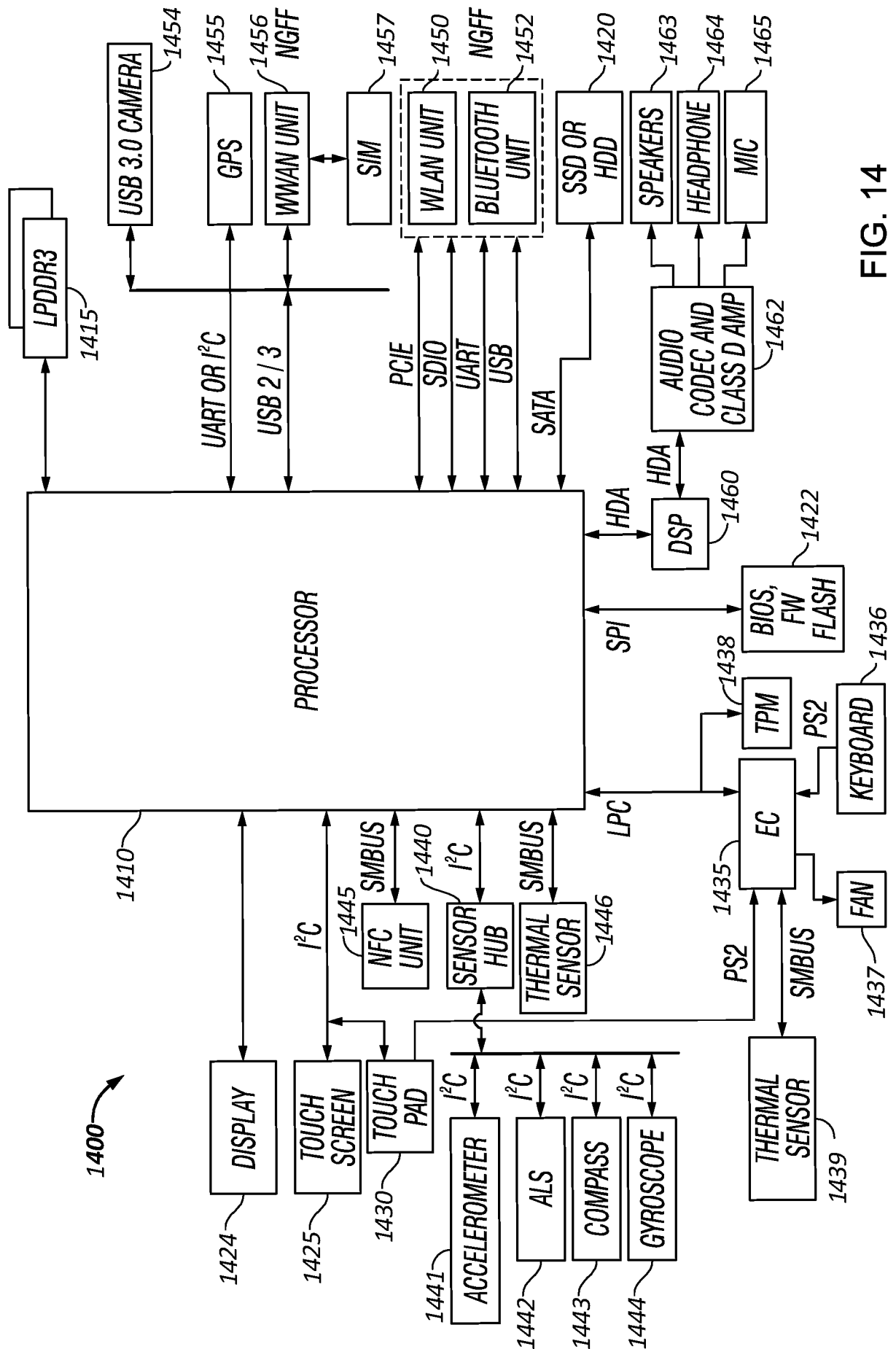


FIG. 14

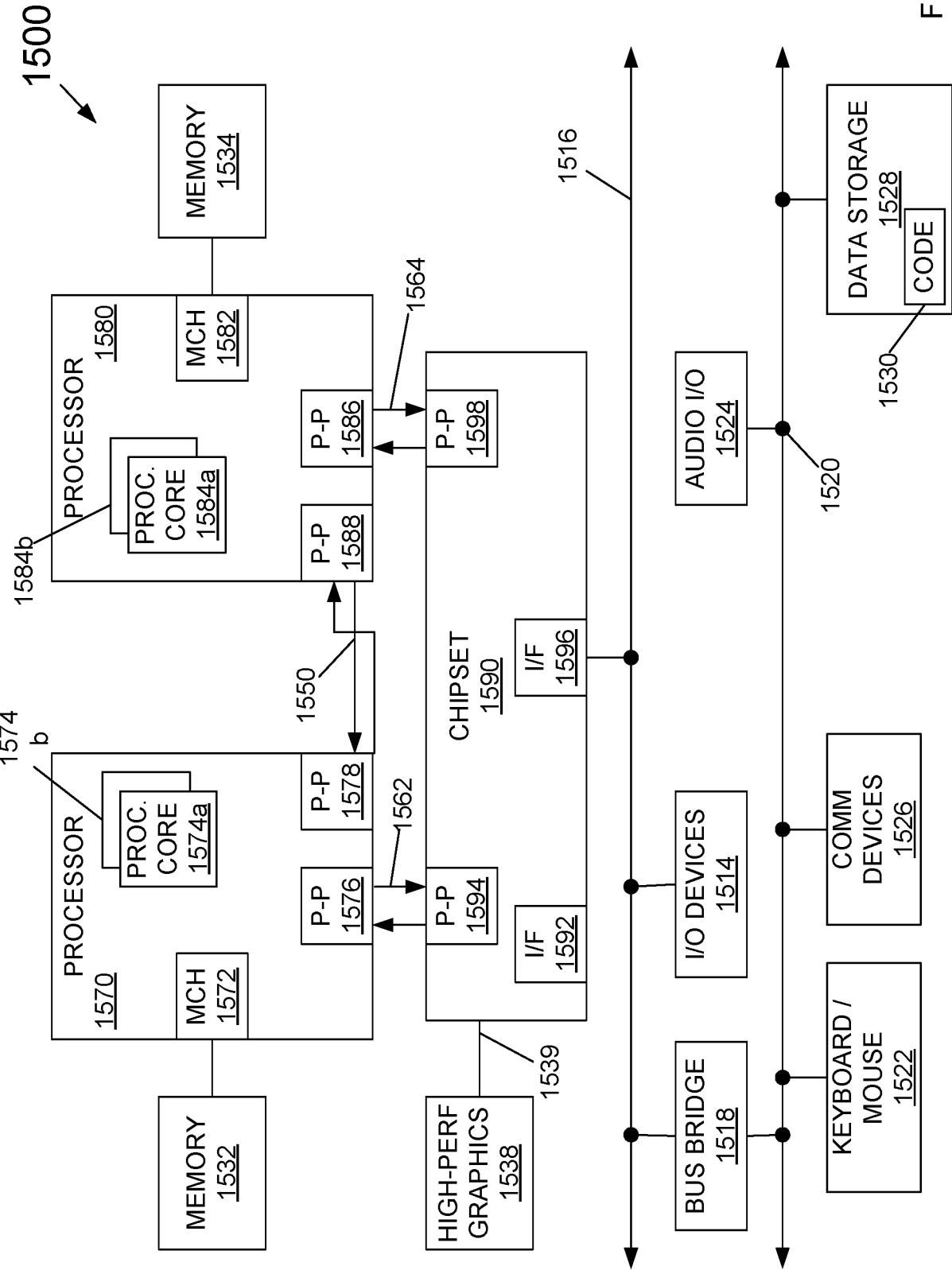


FIG. 15

## INTERNATIONAL SEARCH REPORT

International application No.  
**PCT/US2017/031182****A. CLASSIFICATION OF SUBJECT MATTER****G06F 13/16(2006.01)i, G06F 12/0806(2016.01)i**

According to International Patent Classification (IPC) or to both national classification and IPC

**B. FIELDS SEARCHED**Minimum documentation searched (classification system followed by classification symbols)  
G06F 13/16; G06F 9/30; G06F 12/00; G06F 13/14; G06F 12/08; G06F 13/00; G06F 12/0806Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched  
Korean utility models and applications for utility models  
Japanese utility models and applications for utility modelsElectronic data base consulted during the international search (name of data base and, where practicable, search terms used)  
eKOMPASS(KIPO internal) & Keywords: memory request, contention, non-blocking cache coherence protocol, multi cores, and similar terms.**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

| Category* | Citation of document, with indication, where appropriate, of the relevant passages                                                                                                                                               | Relevant to claim No. |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|
| Y         | US 6,516,393 B1 (MICHAEL FEE et al.) 04 February 2003<br>See column 3, lines 28-30; column 4, lines 66-67; column 6, lines 41-51;<br>column 9, lines 51-61;<br>claim 1; and figures 1 and 4.                                     | 1-3, 9-11, 15-19      |
| A         |                                                                                                                                                                                                                                  | 4-8, 12-14, 20-22     |
| Y         | HUANG YONGQIN et al., 'A Novel Directory-Based Non-Busy, Non-Blocking Cache Coherence,' International Forum on Computer Science-Technology and Applications (IFCSTA '09), pp. 374-379, 25-27 December 2009<br>See pages 374-375. | 1-3, 9-11, 15-19      |
| A         | US 2011-0252204 A1 (BRETT W. COON et al.) 13 October 2011<br>See paragraphs [0087]-[0100] and figures 6-7.                                                                                                                       | 1-22                  |
| A         | US 2003-0005229 A1 (SRINIVASA RANGAN) 02 January 2003<br>See paragraphs [0022]-[0025] and figure 2.                                                                                                                              | 1-22                  |
| A         | US 5,432,918 A (REBECCA L. STAMM) 11 July 1995<br>See column 35, line 44 - column 36, line 29; and figure 18.                                                                                                                    | 1-22                  |



Further documents are listed in the continuation of Box C.



See patent family annex.

\* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&amp;" document member of the same patent family

Date of the actual completion of the international search

18 August 2017 (18.08.2017)

Date of mailing of the international search report

**18 August 2017 (18.08.2017)**

Name and mailing address of the ISA/KR

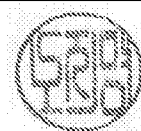
International Application Division  
Korean Intellectual Property Office  
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea

Facsimile No. +82-42-481-8578

Authorized officer

NHO, Ji Myong

Telephone No. +82-42-481-8528



**INTERNATIONAL SEARCH REPORT**

Information on patent family members

International application No.

**PCT/US2017/031182**

| Patent document<br>cited in search report | Publication<br>date | Patent family<br>member(s)                                                                                                                                                                                                             | Publication<br>date                                                                                                                                                                              |
|-------------------------------------------|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| US 6516393 B1                             | 04/02/2003          | CN 1222890 C<br>CN 1347041 A<br>JP 2002-182976 A<br>JP 3871305 B2                                                                                                                                                                      | 12/10/2005<br>01/05/2002<br>28/06/2002<br>24/01/2007                                                                                                                                             |
| US 2011-0252204 A1                        | 13/10/2011          | US 2012-0221808 A1<br>US 8176265 B2<br>US 8645638 B2                                                                                                                                                                                   | 30/08/2012<br>08/05/2012<br>04/02/2014                                                                                                                                                           |
| US 2003-0005229 A1                        | 02/01/2003          | US 6711654 B2                                                                                                                                                                                                                          | 23/03/2004                                                                                                                                                                                       |
| US 5432918 A                              | 11/07/1995          | CA 2045715 A1<br>CA 2045808 A1<br>EP 0465319 A2<br>EP 0465319 B1<br>EP 0468831 A2<br>EP 0468831 B1<br>JP 06-095963 A<br>JP 06-095964 A<br>US 5155843 A<br>US 5317720 A<br>US 5347648 A<br>US 5404482 A<br>US 5404483 A<br>US 5471591 A | 30/12/1991<br>30/12/1991<br>08/01/1992<br>02/08/1995<br>29/01/1992<br>15/10/1997<br>08/04/1994<br>08/04/1994<br>13/10/1992<br>31/05/1994<br>13/09/1994<br>04/04/1995<br>04/04/1995<br>28/11/1995 |