



- (51) International Patent Classification:  
*G06F 9/48* (2006.01)
- (21) International Application Number:  
PCT/US2013/048793
- (22) International Filing Date:  
29 June 2013 (29.06.2013)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:  
13/626,441 25 September 2012 (25.09.2012) US
- (71) Applicant: INTEL CORPORATION [US/US]; 2200 Mission College Boulevard, Santa Clara, California 95054 (US).
- (72) Inventors: SHANBHOGUE, Vedvyas; 12531 W Hwy 71, Apt P2102, Austin, Texas 78738 (US). ROBINSON, Stephen J.; 2013 Maize Bend Dr., Austin, Texas 78727 (US).
- (74) Agent: ROZMAN, Mark J.; 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, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, 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, 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, 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,

[Continued on next page]

(54) Title: ENABLING VIRTUALIZATION OF A PROCESSOR RESOURCE

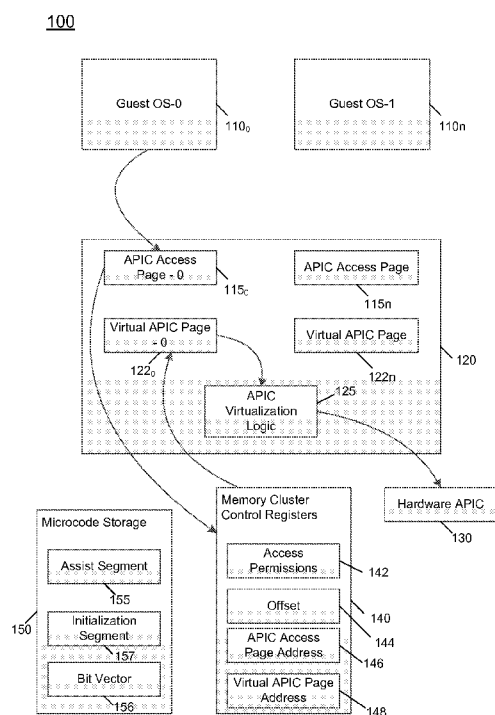


FIG. 1

(57) Abstract: In one embodiment, a processor includes an access logic to determine whether an access request from a virtual machine is to a device access page associated with a device of the processor and if so, to re-map the access request to a virtual device page in a system memory associated with the VM, based at least in part on information stored in a control register of the processor. 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).

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

**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))*

**Published:**

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

## ENABLING VIRTUALIZATION OF A PROCESSOR RESOURCE

### Background

[0001] A conventional virtual machine monitor (VMM) executes on a computing system and presents to other software the abstraction of one or more virtual machines (VMs). Each virtual machine may function as a self-contained platform, running its own guest operating system (OS) hosted by the VMM and other software, collectively referred to as guest software. The guest software expects to operate as if it were running on a dedicated computer rather than a virtual machine. That is, the guest software expects to control various events and have access to hardware resources. The hardware resources may include processor resident resources, resources that reside in memory and resources that reside on the underlying hardware platform. The events may include interrupts, exceptions, platform events, execution of certain instructions, and so forth.

[0002] In a virtual machine environment, the VMM should be able to have ultimate control over these events and hardware resources to provide proper operation of guest software running on the virtual machines and for protection from and between guest software running on the virtual machines. To achieve this, the VMM typically receives control when guest software accesses a protected resource or when other events (such as interrupts or exceptions) occur. For example, when an operation in a virtual machine supported by the VMM causes a system device to generate an interrupt, the currently running virtual machine is interrupted and control of the processor is passed to the VMM. The VMM then receives the interrupt, and handles the interrupt itself or delivers the interrupt to the appropriate virtual machine. However, this switching between guest and VMM is an inefficient use of processor cycles.

### Brief Description of the Drawings

[0003] FIG. 1 is a block diagram of a portion of a system in accordance with an embodiment of the present invention.

[0004] FIG. 2 is a flow diagram of a method in accordance with an embodiment of the present invention.

[0005] FIG. 3 is a flow diagram of a method for performing APIC virtualization microcode operations in accordance with an embodiment of the present invention.

[0006] FIG. 4 is a block diagram of one embodiment of a virtual machine environment in accordance with an embodiment of the present invention.

[0007] FIG. 5 is a block diagram of a processor core in accordance with one embodiment of the present invention.

[0008] FIG. 6 is a block diagram of a processor in accordance with an embodiment of the present invention.

[0009] FIG. 7 is a block diagram of an embodiment of a processor including multiple cores.

[0010] FIG. 8 is a block diagram of a system in accordance with an embodiment of the present invention.

#### Detailed Description

[0011] In various embodiments, one or more resources of a processor such as a multicore processor can be virtualized using control structures and logic as described herein. More specifically, embodiments are directed to virtualization of a memory mapped processor resource such as an advanced programmable interrupt controller (APIC). In general, an APIC is a hardware resource of a processor such as a microcontroller or logic to receive incoming interrupts from various agents, both internal and external to the processor and to prioritize and select an appropriate resource for handling the interrupt, such as one or more cores of the processor.

[0012] Using embodiments of the present invention, this APIC may be virtualized such that multiple virtual agents e.g., individual guests executing on the processor, can access the APIC in a time sliced manner.

[0013] As will be described further below, to effect APIC virtualization, multiple addresses may be stored into a control structure for a virtual machine, referred to herein as a virtual machine control structure (VMCS), which may be provided for each virtual machine running on a processor and which can be stored in a system memory. Namely, these addresses may include a physical access page address

and a virtual access page address. The physical access page address is the physical address to which the guest OS performs input/output (IO) operations. In the example of virtualizing the APIC, this is the address to which the guest performs memory mapped IO operations. The virtual access page address is the address of a page in memory where the state of the virtualized device is stored. In the example of virtualizing an APIC this page stores the state of the virtual APIC. Specifically, the physical access page may correspond to an APIC access page and the virtual access page may correspond to a virtual APIC page, in the example of virtualizing an APIC. However, more generally these addresses may be for a device access page and a virtual device page. Guest accesses to valid locations within this APIC access page may be emulated by remapping such accesses to the virtual APIC page. In this way, a write access stores data to the corresponding offset within the virtual APIC page, and in turn a load access returns data from the corresponding offset in the virtual APIC page to the requester. Accesses to invalid locations within the APIC access page can lead to an exit from a virtual machine so that a virtual machine monitor or other supervisor software can handle the condition.

[0014] To effect such operation efficiently, embodiments may provide for combined hardware controlled and microcode controlled mechanisms of the processor. More specifically, this microcode may be invoked in certain cases when an access to an APIC access page is detected. In general, this microcode may verify that various attributes of the access is valid and if so, re-start instruction execution after programming of hardware to remap the APIC access page to an offset within the virtual APIC page for efficient access upon a next access to the APIC access page.

[0015] In addition, embodiments further provide hardware that can be used in place of the microcode assist to maintain correct operation even with a lack of instruction inclusion in an instruction cache, where incorrect operation may occur due to fetching of an instruction from an instruction cache that has been changed since the remapping. Embodiments may thus combine virtualization hardware and microcode assist mechanisms to ensure correct operation even without a guarantee of instruction inclusion in an instruction cache.

[0016] In various embodiments, access logic within a memory cluster of a processor may provide for matching of a programmable offset on the APIC access page using information from a control register in accordance with an embodiment of the present invention. When a guest is entered (e.g., via a VMLAUNCH or VMRESUME instruction), the microcode may program a first address field in a page address field of this control register (e.g., an EMULATED\_REG\_OFFSET field) with an offset to a predetermined offset value (e.g., 0x80H) within the APIC access page. In addition, as a default the control register may be programmed to provide for both emulation of read and write accesses via first and second permission indicators stored in permission fields (e.g., EMULATE\_READ and EMULATE\_WRITE fields). On a VM entry, the APIC access page address and the vAPIC page address are read out of the VMCS and written into these control registers.

[0017] Then upon detection of an access to the APIC access page, the address of this access request may be remapped to a corresponding offset in a vAPIC page, e.g., corresponding to the address stored in the control register (namely the vAPIC page address, which may be originally obtained from the VMCS upon entry into the guest).

[0018] The memory cluster then verifies the permissions of the access against those programmed in the control register, as follows: if the access is a load and bits 11:0 of the access address match an EMULATED\_REG\_OFFSET value within the control register and the EMULATE\_READ indicator is programmed to be set (e.g., a logic 1), the access is allowed; and if the access is a store and bits 11:0 of the access address match the EMULATES\_REG\_OFFSET value and the EMULATE\_WRITE indicator is programmed 1 the access is allowed. When the access is allowed the memory cluster remaps the address of the load or store such that it is performed to the virtual APIC page.

[0019] If these conditions are not met (access is not to an offset programmed by microcode, access is a load and EMULATE\_READ is 0, or access is a store and EMULATE\_WRITE is 0), hardware may invoke a microcode assist. The hardware also provides details of the access, e.g., offset, size, and type (read/write), to the microcode assist.

[0020] In this microcode assist, the offset, size and type (load or store) of the access may be evaluated to determine if the access is to a valid location. If so, the microcode may program appropriate values (e.g., as to the permissions and the APIC access page offset) into the control register and restart the instruction.

[0021] If the restarted instruction is the same as the one that originally caused the assist, the access will now match in the memory cluster and will be emulated and redirected to the vAPIC page. Any subsequent instructions that match this offset of a type allowed for emulation will continue to be emulated and remapped without a microcode assist. This behavior allows the remaps to happen in hardware without a microcode assist for the most commonly accessed offsets.

[0022] Referring now to FIG. 1, shown is a block diagram of a portion of a system in accordance with an embodiment of the present invention. The portion of the system shown in FIG. 1 details the interaction between various layers of software and hardware. Specifically, one or more guest operating systems (OS) may operate within a corresponding virtual machine 110<sub>0</sub>-110<sub>n</sub> (generically VM or guest 110). Of course, additional software may execute within each VM, such as various guest software, e.g., one or more user-level applications.

[0023] Relevant to the discussion here, a guest OS executing in a guest 110<sub>0</sub> can issue an access request, ostensibly to a memory structure. However, due to virtualization, instead of communicating directly with underlying hardware such as a hardware APIC 130, the access instead may be made to an APIC access page 115<sub>0</sub>, which is a valid physical address within a virtual machine monitor (VMM) 120 that is mapped into the guest address space. This access page may be a page in physical memory associated with a given guest. Note that there may be an APIC access page provided per guest.

[0024] This access to APIC access page 115 may in turn lead to an access to a virtual APIC (vAPIC) page 122<sub>0</sub> within system memory used by VMM 120. That is, depending upon an offset to which the access is directed within this page, an access logic 125 of the VMM (also referred to herein as APIC virtualization logic) can

determine whether to allow this access to flow to the underlying hardware APIC 130, which may be a memory mapped hardware structure of the processor.

[0025] To enable such access to occur, first it can be determined whether the access from the guest is allowed. To this end, additional hardware, namely a control register 140 (that can be implemented as one or more control registers of a memory cluster of a processor), may be provided to store information regarding access permissions, address information and so forth for a particular guest that is executing. In the embodiment shown, control register 140 may include an access permission storage 142. In various embodiments, this storage may include one or more fields to store indicators to indicate whether particular types of accesses are allowed to an APIC access page. In one embodiment, this permission storage can include a read permission indicator and a write permission indicator. In addition, an offset storage 144 may be present. The offset storage may store an offset to a location within an APIC access page to emulate. A vAPIC storage 146 also may be provided. This storage may be used to store a base address of the vAPIC page for the corresponding guest. In addition to a vAPIC storage may be provided. This storage may be used to store a base address of the vAPIC page to which the emulated access is to be redirected.

[0026] Thus before an access within a virtualized environment to hardware APIC 130 is permitted, it can be determined whether the access is allowed based at least in part on information stored in control register 140. In addition as will be described further below, the processor may include a microcode storage 150 that includes various microcode to be executed by underlying hardware of the processor. As part of this microcode, an assist microcode segment 155 may be provided that can be used to assist in emulation operations with regard to the APIC flow as described herein. Also for use in emulation operations, access can be made to a bit vector 156 within the microcode storage, details of which are described further below. The microcode may also include an initialization segment 157 to perform initialization operations for the processor, including programming default values for the control register and to perform further operations on entry into a VM. Also note that while the hardware device emulation in the embodiments described herein is for an APIC,

understand the scope of the present invention is not limited in this regard and in other embodiments, other types of hardware devices can be emulated such as a network controller. Although shown at this high level in the embodiment of FIG. 1, understand the scope of the present invention is not limited in this regard.

[0027] Referring now to FIG. 2, shown is a flow diagram of a method in accordance with an embodiment of the present invention. As shown in FIG. 2, method 200 may be performed by logic within a memory cluster of a processor that can handle incoming memory requests and determine whether to allow access in accordance with an embodiment of the present invention. As shown in FIG. 2, method 200 may begin by determining whether an access request received in the memory cluster hits on an APIC access page. As described above, the APIC access page may be a page in VMM physical address space that is mapped into the guest's physical address space. If the access is not for such a page, control passes to block 215 where conventional operations for handling an access, e.g., a load or store access, may be performed.

[0028] If instead it is determined that the access request hits an APIC access page, control passes to diamond 220 to determine whether the virtual APIC page corresponding to this APIC access page of the VM from which the access was performed is valid. In an embodiment, this determination may be based on a valid bit in a control register. As an example, control register 140 shown in FIG. 1 may include in addition to a vAPIC page address a valid bit to indicate whether the address stored in this portion of the control register is valid (which may also be stored in the control register). If so, control passes next to diamond 230 where it can be determined whether the access is to an emulated register. Although the scope of the present invention is not limited in this regard, this determination may be based on various constituent determinations. Specifically, in one embodiment these determinations may include: whether the access is for a data read or write (not an instruction fetch); access size is at most 32 bits; access is contained within low 4 bytes of a naturally aligned 16 byte region; bits 11:0 of the access address match EMULATED\_REG\_OFFSET (e.g., stored in offset field 144); access is a data read

and EMULATED\_READ is 1, or access is a data write and EMULATED\_WRITE is 1 (e.g., both stored in access permission field 142).

[0029] Thus if the determination at diamond 230 is in positive that the access is to an emulated register, control passes to block 240 where an address of the access request, e.g., a load/store access operation may be remapped to an offset in the vAPIC page. As such, for a read operation, requested information may be read from the corresponding offset within the virtual APIC page and reported back to the guest. Or, if it is a store request, the corresponding data of the write request may be stored at the offset indicated in the vAPIC page, namely the offset in the vAPIC page corresponding to the offset in the APIC access page, in other words, the offset specified by bits 11:0 of the access.

[0030] Still referring to FIG. 2 from both diamond 220 and diamond 230 if the determinations of either of these points are in the negative, control passes to block 225 where an APIC virtualization microcode assist may be invoked. Further details of this assist are described further below.

[0031] Referring now to FIG. 3, shown is a flow diagram of a method for performing APIC virtualization microcode operations in accordance with an embodiment of the present invention. As seen in FIG. 3, method 250 may begin by determining whether the requested access is to an emulated offset. This determination, as with all other operations shown in FIG. 3, may be performed by microcode. More specifically, a given processor may include a microcode segment, referred to herein as a microcode assist segment that includes microcode for execution on hardware of the processor to efficiently perform the illustrated operations. In an embodiment, this determination of access to an emulated offset may be based on a bit vector for valid locations within an APIC access page. In one embodiment, this bit vector may be stored in a non-volatile storage, e.g., as part of a processor's microcode. In another embodiment, this bit vector may be provided by the VMM through the VMCS and stored in a processor control register for subsequent use by this microcode. Although the scope of the present invention is not limited in this regard, in one embodiment a plurality of valid offsets may be provided for read emulation and a plurality of valid offsets may be provided for write emulation. In one particular

embodiment, approximately 40 valid offsets may be provided for read emulation and approximately 20 valid offsets provided for write emulations, although the scope of the present invention is not limited in this regard.

[0032] If the access is not to one of these emulated offsets, e.g., as determined based on a bit vector, control passes to block 290 where an exit may be performed from a virtual machine environment to the VMM. More specifically, a VM exit may occur to thus pass control to the VMM with an indication that the fault is due to an APIC access failure. Responsive to such an indication, the VMM may take appropriate action such as terminating the guest or performing emulated device specific actions like logging an error or so forth.

[0033] Still referring to FIG. 3, if instead the access is to a valid emulated offset, control passes to block 270 where various information may be programmed to allow an access to an virtual APIC page to occur (as described above with regard to the memory cluster logic). In one embodiment, this programming may be of the control register described above. More specifically, this control register may be programmed with permissions regarding emulated reads and writes as well as an emulated register offset to thus indicate that an access to this emulated offset is to be allowed.

[0034] Control next passes to block 280 where the assisting instruction may be restarted. More specifically, the microcode may issue an instruction to a front end unit of the processor to cause the access request instruction (e.g., a load/store) to be re-fetched and re-executed. Note that it is possible that this instruction may still be present in a caching structure of the processor, such as an instruction cache. If so, the instruction can be obtained with minimal latency and passed for execution. Otherwise if the instruction has been evicted for any of various reasons such as a capacity eviction, the instruction may again be fetched from a memory hierarchy. In any event, method 250 concludes and control passes back to further operations for this access instruction. For example, the operations of FIG. 2 may again be performed for the re-fetched instruction. However at this time (assuming the same instruction is obtained) the determinations at both diamonds 220 and 230 should be in the affirmative and thus a remapping may be performed at block 240. Although

shown at this high level in the embodiment of FIG. 3, understand the scope of the present invention is not limited in this regard.

[0035] FIG. 4 illustrates one embodiment of a virtual machine environment 300. In this embodiment, bare platform hardware 310 is a computing platform, such as a given server, desktop, laptop, Ultrabook™, tablet, smartphone or so forth to execute an OS or VMM, such as VMMs 325. The platform hardware 310 includes at least one processor 312, memory 320 and possibly other platform hardware (e.g. input output devices), not shown.

[0036] Processor 312 can be any type of processor capable of executing software, such as a microprocessor, digital signal processor, microcontroller, or the like. The processor 312 may include microcode, programmable logic or hardcoded logic for performing virtualization of processor components according to embodiments of the present invention. To this end, access logic 330 may be present to control access to a virtualized resource such as an APIC (not shown for ease of illustration).

[0037] Memory 320 can be a hard disk, random access memory (RAM), read only memory (ROM), flash memory, or other non-volatile storage, or any combination of the above devices, or any other type of non-transitory machine readable storage medium readable by processor 312. Memory 320 may store instructions or data for performing the execution of method embodiments of the present invention. As seen memory 320 includes one or more VMCSs 322 that include addresses for one or more device access pages, and corresponding virtual device pages, as described herein.

[0038] Each VMM 325, though typically implemented in software, may emulate and export a bare machine interface to higher level software. Such higher level software may be a standard or real-time OS. The VMMs 325 may be implemented, for example, in hardware, software, firmware, or by a combination of various techniques.

[0039] When running, each VMM 325 presents to guest software (software other than that of the VMMs 325) the abstraction of one or more VMs. The VMMs 325 may provide the same or different abstractions to the various guests. The guest software running on each VM may include a guest OS (e.g., a guest OS 334, 344 or

354) and various guest software applications (e.g., applications 336, 346 and 356). Collectively, guest OS and software applications are referred to herein as guest software 303, 305 and 315.

[0040] Guest software 303, 305 and 315 expects to access physical resources (e.g., processor registers, memory and I/O devices) within the VMs 332, 342 and 352 on which the guest software is running. The VMMs 325 facilitate access to resources desired by guest software while retaining ultimate control over resources within the platform hardware 310. In addition, the guest software 303, 305 and 315 expect to handle various events such as exceptions, interrupts and platform events.

[0041] Embodiments can be implemented in many different systems. For example, embodiments can be realized in a processor such as a multicore processor.

Referring now to FIG. 5, shown is a block diagram of a processor core in accordance with one embodiment of the present invention. As shown in FIG. 5, processor core 400 may be one core of a multicore processor, and is shown as a multi-stage pipelined out-of-order processor. Processor core 400 is shown with a relatively simplified view in FIG. 5 to illustrate various features used in connection with virtualization of a processor resource in accordance with an embodiment of the present invention.

[0042] As shown in FIG. 5, core 400 includes front end units 410, which may be used to fetch instructions to be executed and prepare them for use later in the processor. For example, front end units 410 may include a fetch unit 401, an instruction cache 403, and an instruction decoder 405. In some embodiments, the instruction cache may maintain inclusiveness, keeping entries until a corresponding instruction is retired. In other embodiments, the instruction cache may be a non-inclusive cache such that an entry corresponding to a pending instruction in the core may be evicted prior to retirement of the instruction. In some implementations, front end units 410 may further include a trace cache, along with microcode storage as well as a micro-operation storage. Fetch unit 401 may fetch macro-instructions, e.g., from memory or instruction cache 403, and feed them to instruction decoder 405 to decode them into primitives, i.e., micro-operations for execution by the processor.

Also present may be a microcode storage 407 to store assist code in accordance with an embodiment of the present invention.

[0043] Coupled between front end units 410 and execution units 420 is an out-of-order (OOO) engine 415 that may be used to receive the micro-instructions and prepare them for execution. More specifically OOO engine 415 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 430 and extended register file 435 such as by using renaming logic of the engine. Register file 430 may include separate register files for integer and floating point operations. Extended register file 435 may provide storage for vector-sized units, e.g., 256 or 512 bits per register. A set of control registers 436 may be present, including a register 438 to store information associated with programmable virtualization of a processor resource such as an APIC, as described herein.

[0044] Various resources may be present in execution units 420, 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) 422. Of course other execution units such as multiply-accumulate units and so forth may further be present. Results may be provided to a retirement logic, which may be implemented within a memory subsystem 460 of the processor. Various processor structures including execution units and front end logic, for example, may be coupled to a memory subsystem 460. This memory subsystem may provide an interface between processor structures and further portions of a memory hierarchy, e.g., an on or off-chip cache and a system memory. As seen the subsystem has various components including a memory order buffer (MOB) 440. More specifically, MOB 440 may include various arrays and logic to receive information associated with instructions that are executed. This information is then examined by MOB 440 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, MOB 440 may handle other operations associated with retirement.

[0045] As shown in FIG. 5, MOB 440 is coupled to a cache 450 which, in one embodiment may be a low level cache (e.g., an L1 cache). Memory subsystem 460 also may include an integrated memory controller 470 to provide for communication with a system memory (not shown for ease of illustration in FIG. 5). Memory subsystem 460 may further include a memory execution unit 475 that handles various operations to initiate memory requests and handle return of data from memory. For example, as shown in the embodiment of FIG. 5, MEU 475 may include a page miss handler 476 to obtain translations from logical addresses to physical addresses when a translation lookaside buffer miss occurs. An access logic 477 of the MEU may perform access checks to determine whether an access request is to an emulated region, and if so, to re-map the request to a redirection region (e.g., an APIC access page-to-vAPIC page remapping). Also present may be an arbitrator 479 to arbitrate among various memory requests also may be present. Further, while not shown understand that other structures such as buffers, schedulers and so forth may be present in the MEU.

[0046] From memory subsystem 460, 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 a so-called x86 instruction set architecture (ISA) architecture, 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.

[0047] That is, in other embodiments, a processor architecture may include emulation features such that the processor can execute instructions of a first ISA, referred to as a source ISA, where the architecture is according to a second ISA, referred to as a target ISA. In general, software, including both the OS and

application programs, is compiled to the source ISA, and hardware implements the target ISA designed specifically for a given hardware implementation with special performance and/or energy efficiency features.

[0048] Referring now to FIG. 6, shown is a block diagram of a processor in accordance with an embodiment of the present invention. As shown in FIG. 6, processor 500 may be a multicore processor including a plurality of cores 510a – 510n in a core domain 510. In one embodiment, each such core may be of an independent power domain and can be configured to operate at an independent voltage and/or frequency, and to enter turbo mode when available headroom exists, or the cores can be uniformly controlled as a single domain. The cores may be coupled via an interconnect 515 to a system agent or uncore 520 that includes various components. As seen, the uncore 520 may include a shared cache 530 which may be a last level cache. In addition, the uncore may include an integrated memory controller 540, various interfaces 550 and a power control unit 555 to control power consumption by the components of the processor. An APIC 545 may be virtualized among multiple VMs executing on the cores using access logic of memory clusters of the cores as described herein.

[0049] With further reference to FIG. 6, processor 500 may communicate with a system memory 560, e.g., via a memory bus. In addition, by interfaces 550, 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. 6, the scope of the present invention is not limited in this regard.

[0050] Referring to FIG. 7, an embodiment of a processor including multiple cores is illustrated. Processor 1100 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 1100, in one embodiment, includes at least two cores—cores 1101 and 1102, which may include asymmetric cores or symmetric cores (the illustrated embodiment). However, processor 1100 may include any number of processing elements that may be symmetric or asymmetric.

[0051] 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.

[0052] 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.

[0053] Physical processor 1100, as illustrated in FIG. 7, includes two cores, cores 1101 and 1102. Here, cores 1101 and 1102 are considered symmetric cores, i.e., cores with the same configurations, functional units, and/or logic. In another embodiment, core 1101 includes an out-of-order processor core, while core 1102 includes an in-order processor core. However, cores 1101 and 1102 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 1101 are

described in further detail below, as the units in core 1102 operate in a similar manner.

[0054] As depicted, core 1101 includes two hardware threads 1101a and 1101b, which may also be referred to as hardware thread slots 1101a and 1101b. Therefore, software entities, such as an operating system, in one embodiment potentially view processor 1100 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 1101a, a second thread is associated with architecture state registers 1101b, a third thread may be associated with architecture state registers 1102a, and a fourth thread may be associated with architecture state registers 1102b. Here, each of the architecture state registers (1101a, 1101b, 1102a, and 1102b) may be referred to as processing elements, thread slots, or thread units, as described above. As illustrated, architecture state registers 1101a are replicated in architecture state registers 1101b, so individual architecture states/contexts are capable of being stored for logical processor 1101a and logical processor 1101b. In core 1101, other smaller resources, such as instruction pointers and renaming logic in allocator and renamer block 1130 may also be replicated for threads 1101a and 1101b. Some resources, such as re-order buffers in reorder/retirement unit 1135, ILTB 1120, 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 1115, execution unit(s) 1140, and portions of out-of-order unit 1135 are potentially fully shared.

[0055] Processor 1100 often includes other resources, which may be fully shared, shared through partitioning, or dedicated by/to processing elements. In FIG. 7, 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 1101 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 1120 to

predict branches to be executed/taken and an instruction-translation buffer (I-TLB) 1120 to store address translation entries for instructions.

[0056] Core 1101 further includes decode module 1125 coupled to fetch unit 1120 to decode fetched elements. Fetch logic, in one embodiment, includes individual sequencers associated with thread slots 1101a, 1101b, respectively. Usually core 1101 is associated with a first ISA, which defines/specifies instructions executable on processor 1100. 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 1125 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 1125, in one embodiment, include logic designed or adapted to recognize specific instructions, such as transactional instruction. As a result of the recognition by decoders 1125, the architecture or core 1101 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.

[0057] In one example, allocator and renamer block 1130 includes an allocator to reserve resources, such as register files to store instruction processing results. However, threads 1101a and 1101b are potentially capable of out-of-order execution, where allocator and renamer block 1130 also reserves other resources, such as reorder buffers to track instruction results. Unit 1130 may also include a register renamer to rename program/instruction reference registers to other registers internal to processor 1100. Reorder/retirement unit 1135 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.

[0058] Scheduler and execution unit(s) block 1140, 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.

[0059] Lower level data cache and data translation buffer (D-TLB) 1150 are coupled to execution unit(s) 1140. 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.

[0060] Here, cores 1101 and 1102 share access to higher-level or further-out cache 1110, 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 1110 is a last-level data cache—last cache in the memory hierarchy on processor 1100—such as a second or third level data cache. However, higher level cache 1110 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 1125 to store recently decoded traces.

[0061] In the depicted configuration, processor 1100 also includes bus interface module 1105 and a power controller 1160, which may perform power sharing control in accordance with an embodiment of the present invention. Historically, controller 1170 has been included in a computing system external to processor 1100. In this scenario, bus interface 1105 is to communicate with devices external to processor 1100, such as system memory 1175, a chipset (often including a memory controller hub to connect to memory 1175 and an I/O controller hub to connect peripheral devices), a memory controller hub, a northbridge, or other integrated circuit. And in this scenario, bus 1105 may include any known interconnect, such as multi-drop bus, a point-to-point interconnect, a serial interconnect, a parallel bus, a coherent (e.g.

cache coherent) bus, a layered protocol architecture, a differential bus, and a GTL bus.

[0062] Memory 1175 may be dedicated to processor 1100 or shared with other devices in a system. Common examples of types of memory 1175 include DRAM, SRAM, non-volatile memory (NV memory), and other known storage devices. Note that device 1180 may include a graphic accelerator, processor or card coupled to a memory controller hub, data storage coupled to an I/O controller hub, a wireless transceiver, a flash device, an audio controller, a network controller, or other known device.

[0063] Note however, that in the depicted embodiment, the controller 1170 is illustrated as part of processor 1100. Recently, as more logic and devices are being integrated on a single die, such as SOC, each of these devices may be incorporated on processor 1100. For example in one embodiment, memory controller hub 1170 is on the same package and/or die with processor 1100. Here, a portion of the core (an on-core portion) includes one or more controller(s) 1170 for interfacing with other devices such as memory 1175 or a graphics device 1180. The configuration including an interconnect and controllers for interfacing with such devices is often referred to as an on-core (or un-core configuration). As an example, bus interface 1105 includes a ring interconnect with a memory controller for interfacing with memory 1175 and a graphics controller for interfacing with graphics processor 1180. Yet, in the SOC environment, even more devices, such as the network interface, co-processors, memory 1175, graphics processor 1180, 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.

[0064] Embodiments may be implemented in many different system types. Referring now to FIG. 8, shown is a block diagram of a system in accordance with an embodiment of the present invention. As shown in FIG. 8, multiprocessor system 600 is a point-to-point interconnect system, and includes a first processor 670 and a second processor 680 coupled via a point-to-point interconnect 650. As shown in FIG. 8, each of processors 670 and 680 may be multicore processors, including first and second processor cores (i.e., processor cores 674a and 674b and processor

cores 684a and 684b), although potentially many more cores may be present in the processors. Each of the processors can include virtualization logic to enable multiple guests to access a memory mapped processor resource such as an APIC, as described herein.

[0065] Still referring to FIG. 8, first processor 670 further includes a memory controller hub (MCH) 672 and point-to-point (P-P) interfaces 676 and 678. Similarly, second processor 680 includes a MCH 682 and P-P interfaces 686 and 688. As shown in FIG. 8, MCH's 672 and 682 couple the processors to respective memories, namely a memory 632 and a memory 634, which may be portions of system memory (e.g., DRAM) locally attached to the respective processors. First processor 670 and second processor 680 may be coupled to a chipset 690 via P-P interconnects 652 and 654, respectively. As shown in FIG. 8, chipset 690 includes P-P interfaces 694 and 698.

[0066] Furthermore, chipset 690 includes an interface 692 to couple chipset 690 with a high performance graphics engine 638, by a P-P interconnect 639. In turn, chipset 690 may be coupled to a first bus 616 via an interface 696. As shown in FIG. 8, various input/output (I/O) devices 614 may be coupled to first bus 616, along with a bus bridge 618 which couples first bus 616 to a second bus 620. Various devices may be coupled to second bus 620 including, for example, a keyboard/mouse 622, communication devices 626 and a data storage unit 628 such as a disk drive or other mass storage device which may include code 630, in one embodiment. Further, an audio I/O 624 may be coupled to second bus 620. Embodiments can be incorporated into other types of systems including mobile devices such as a smart cellular telephone, Ultrabook™, tablet computer, netbook, or so forth.

[0067] 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.

[0068] 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 execution means for executing instructions;  
3 control register means including a first indicator field to store a first permission  
4 indicator to indicate whether an emulated memory access to an advanced  
5 programmable interrupt controller (APIC) access page in a system memory is  
6 allowed, and a first address field to store an offset value to identify an offset within  
7 the APIC access page; and  
8 means for determining whether an access request is to the APIC access page  
9 and if so, re-mapping the access request to a virtual APIC (vAPIC) page in the  
10 system memory associated with a requester of the access request.
- 1 2. The processor of claim 1, wherein the first permission indicator comprises a  
2 read access permission indicator, the control register means further comprising a  
3 second indicator field to store a second permission indicator to indicate whether an  
4 emulated write access to the APIC access page is allowed.
- 1 3. The processor of claim 1, wherein the control register means further  
2 comprises a second address field to store a vAPIC page address corresponding to  
3 an address of the vAPIC page in the system memory.
- 1 4. The processor of claim 1, further comprising a microcode storage means  
2 including an assist segment to program the control register means responsive to the  
3 access request not matching the offset value.
- 1 5. The processor of claim 4, wherein the microcode storage means further  
2 includes an initialization segment to store, upon entry into a guest, a first value in the  
3 first address field and to store a first indicator value in the first indicator field to  
4 indicate that the emulated memory access to the APIC access page is allowed.

1 6. The processor of claim 4, further comprising a bit vector including a plurality of  
2 fields, each to indicate whether an offset within the APIC access page is valid.

1 7. The processor of claim 6, wherein the assist segment is to enable the  
2 processor to determine whether the access request is to a valid offset within the  
3 APIC access page based at least in part on the bit vector, and to cause an exit from  
4 a virtual machine if the access request is not to a valid offset.

1 8. The processor of claim 7, wherein the assist segment is to program the  
2 control register means responsive to the access request being to the valid offset.

1 9. The processor of claim 7, wherein the assist segment is to cause the  
2 processor to restart the access request from a front end unit of the processor, the  
3 front end unit including an instruction cache, wherein the instruction cache is not an  
4 inclusive cache.

1 10. The processor of claim 1, wherein the requester comprises a virtual machine.

1 11. A method comprising:  
2 receiving a memory access request to an address of a system memory in an  
3 access logic of a processor and determining if the address is within a device access  
4 page of a device of the processor;  
5 if so, determining whether the memory access request is for a type having a  
6 valid permission indicator stored in a permission field of a control register of the  
7 processor and a portion of the address corresponds to an emulated offset value  
8 stored in a first address field of the control register; and  
9 if so, remapping the address to a virtual device page of the system memory  
10 associated with a requester of the memory access request.

1 12. The method of claim 11, further comprising remapping the address using an  
2 address of the virtual device page obtained from a second address field of the  
3 control register.

1 13. The method of claim 12, further comprising obtaining the virtual device page  
2 address from a virtual machine control structure for the requester, and storing the  
3 virtual device page address in the second address field.

1 14. The method of claim 13, further comprising obtaining and storing the virtual  
2 device page address using a virtualization initialization segment of a microcode of  
3 the processor.

1 15. The method of claim 11, further comprising if the memory access request is  
2 not for a type having a valid permission indicator or a portion of the address does not  
3 correspond to an emulated offset value, invoking a virtualization assist segment of  
4 microcode of the processor to determine if the memory address access request is to  
5 an emulated offset within the device access page and if so, programming the control  
6 register with the permission indicator and the emulated offset value, and re-starting  
7 the memory access request in a front end unit of the processor.

1 16. The method of claim 15, further comprising if the memory access request is  
2 not to the emulated offset within the device access page, causing an exit from a  
3 guest to a virtual machine monitor executing on the processor.

1 17. A system comprising:

2 a processor including an access means for determining whether an access  
3 request from a virtual machine (VM) is to a device access page associated with a  
4 device of the processor and if so, re-mapping the access request to a virtual device  
5 page in a system memory associated with the VM, based at least in part on  
6 information stored in a control register of the processor including a first indicator field  
7 to store a first permission indicator to indicate whether the access request is allowed  
8 and a first address field to store an offset value to identify a valid offset within the  
9 device access page; and

10 the system memory comprising a dynamic random access memory (DRAM)  
11 coupled to the processor.

1 18. The system of claim 17, wherein the control register further comprises a  
2 second address field to store a virtual device page address corresponding to an  
3 address of the virtual device page in the system memory.

1 19. The system of claim 17, further comprising a microcode storage means  
2 including an assist segment to program the control register responsive to a portion of  
3 an address of the access request not matching the offset value.

1 20. The system of claim 19, wherein the microcode storage means further  
2 comprising:  
3 an initialization segment to store, upon entry into the VM, a first value in the  
4 first address field and to store a first indicator value in the first indicator field to  
5 indicate that an access request of a first type to the device access page is allowed;  
6 and a bit vector including a plurality of fields, each to indicate whether an access  
7 request from a VM to an offset within the device access page is allowed, wherein the  
8 assist segment is to enable the processor to determine whether the access request  
9 to a first offset within the device access page is allowed based at least in part on the  
10 bit vector, and to cause an exit from the virtual machine if the access request is not  
11 allowed.

1 21. A communication device arranged to perform the method of any one of claims  
2 11 to 16.

1 22. At least one machine readable medium comprising a plurality of instructions  
2 that in response to being executed on a computing device, cause the computing  
3 device to carry out a method according to any one of claims 11 to 16.

1 23. An apparatus for processing instructions, configured to perform the method of  
2 any one of claims 11 to 16.

- 1 24. An apparatus comprising means for performing the method of any one of
- 2 claims 11 to 16.

1/8

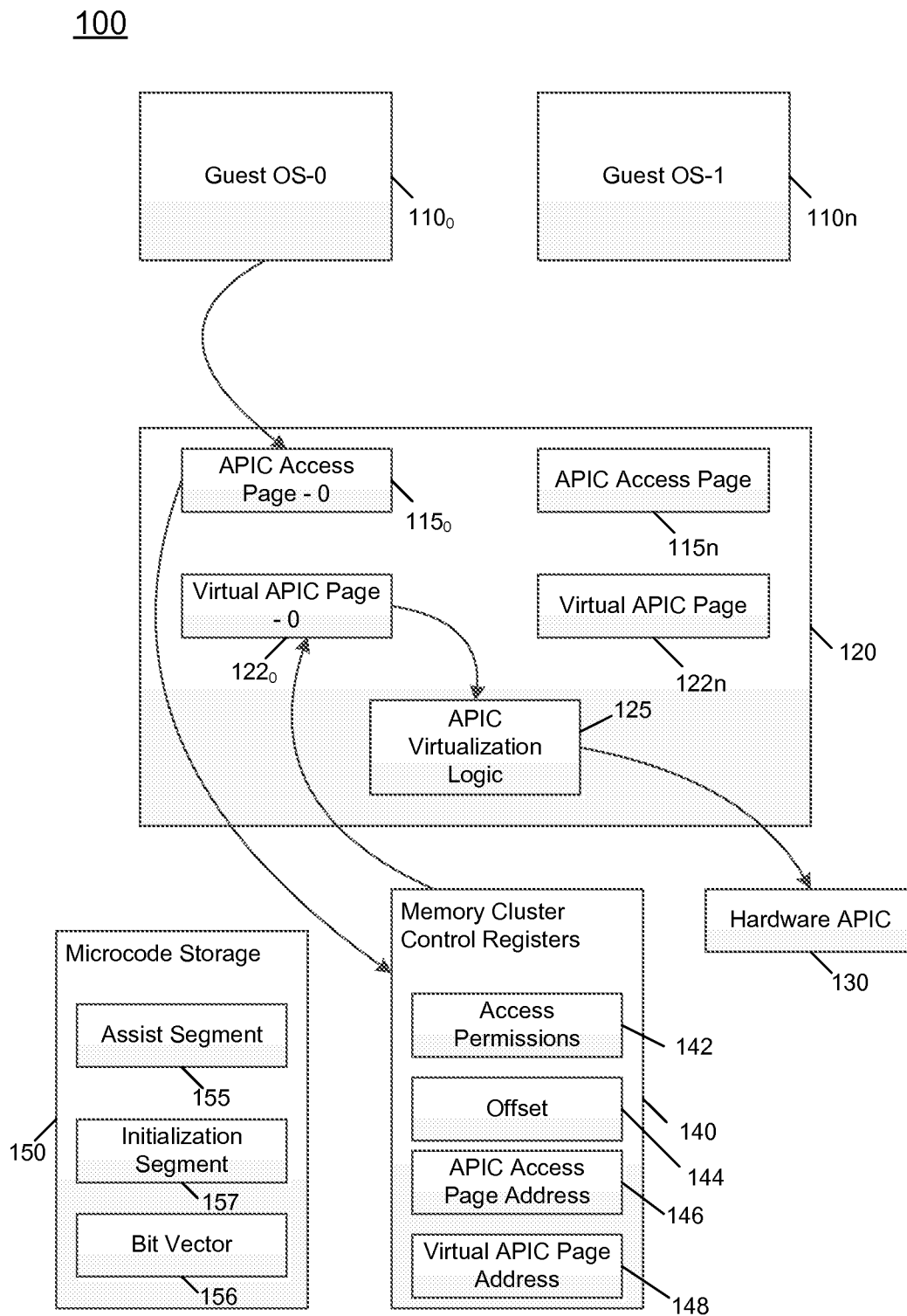


FIG. 1

2/8

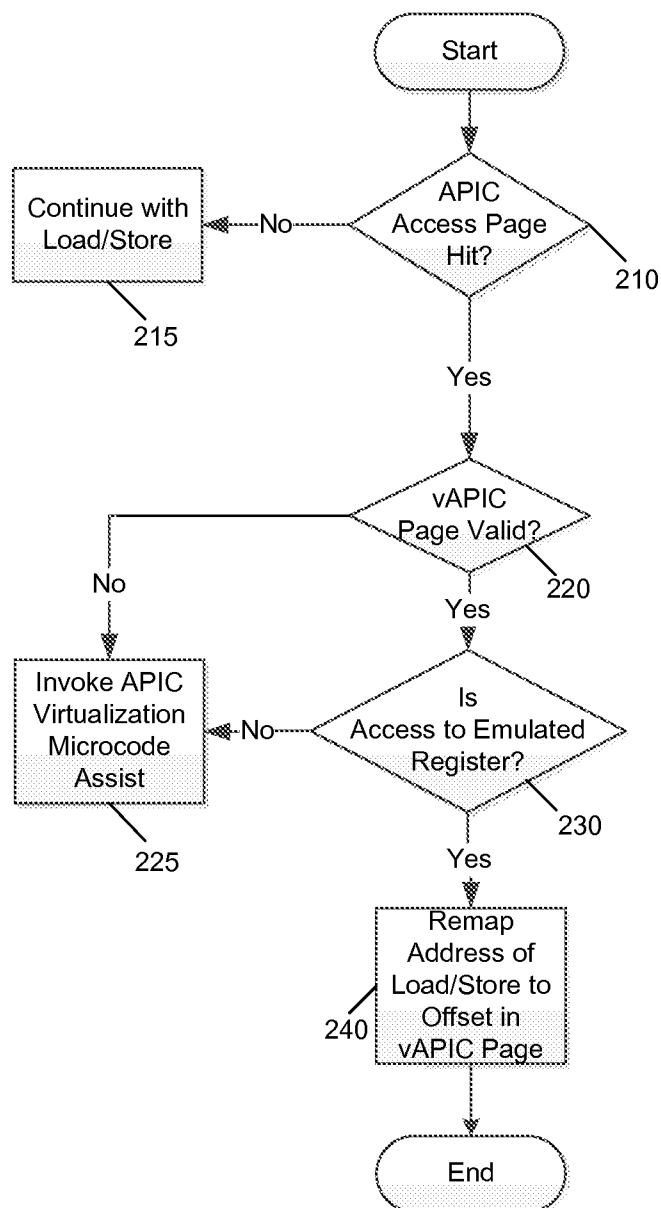
200

FIG. 2

3/8

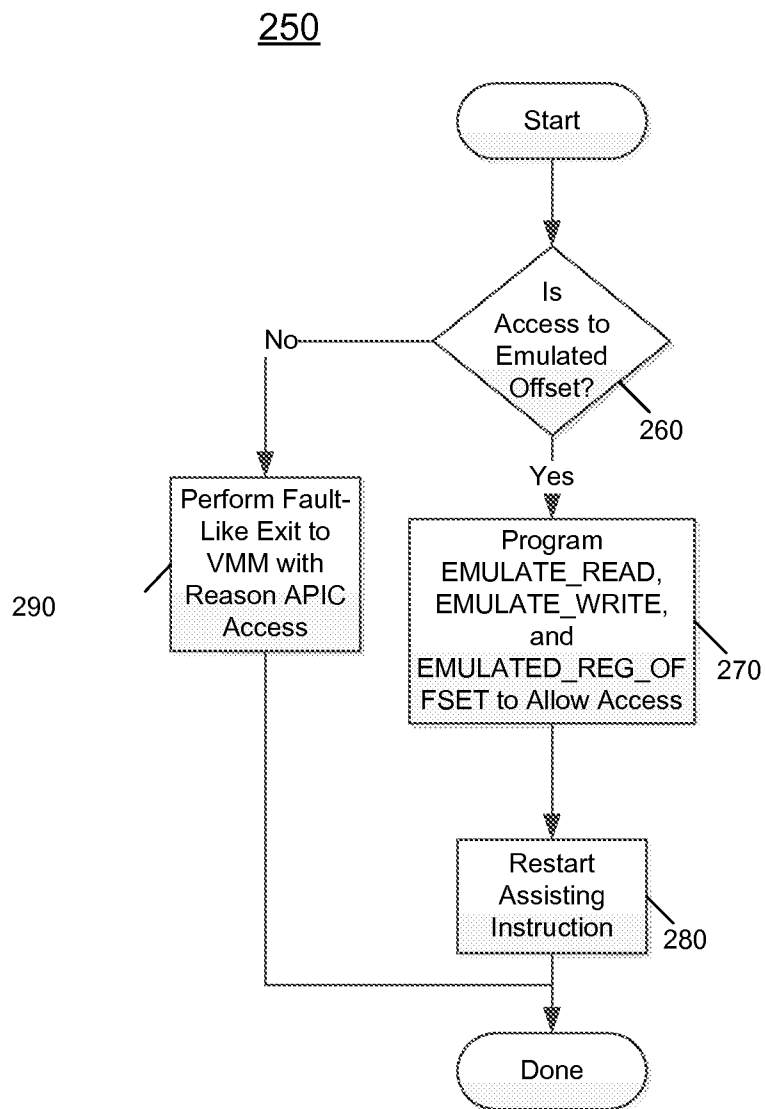


FIG. 3

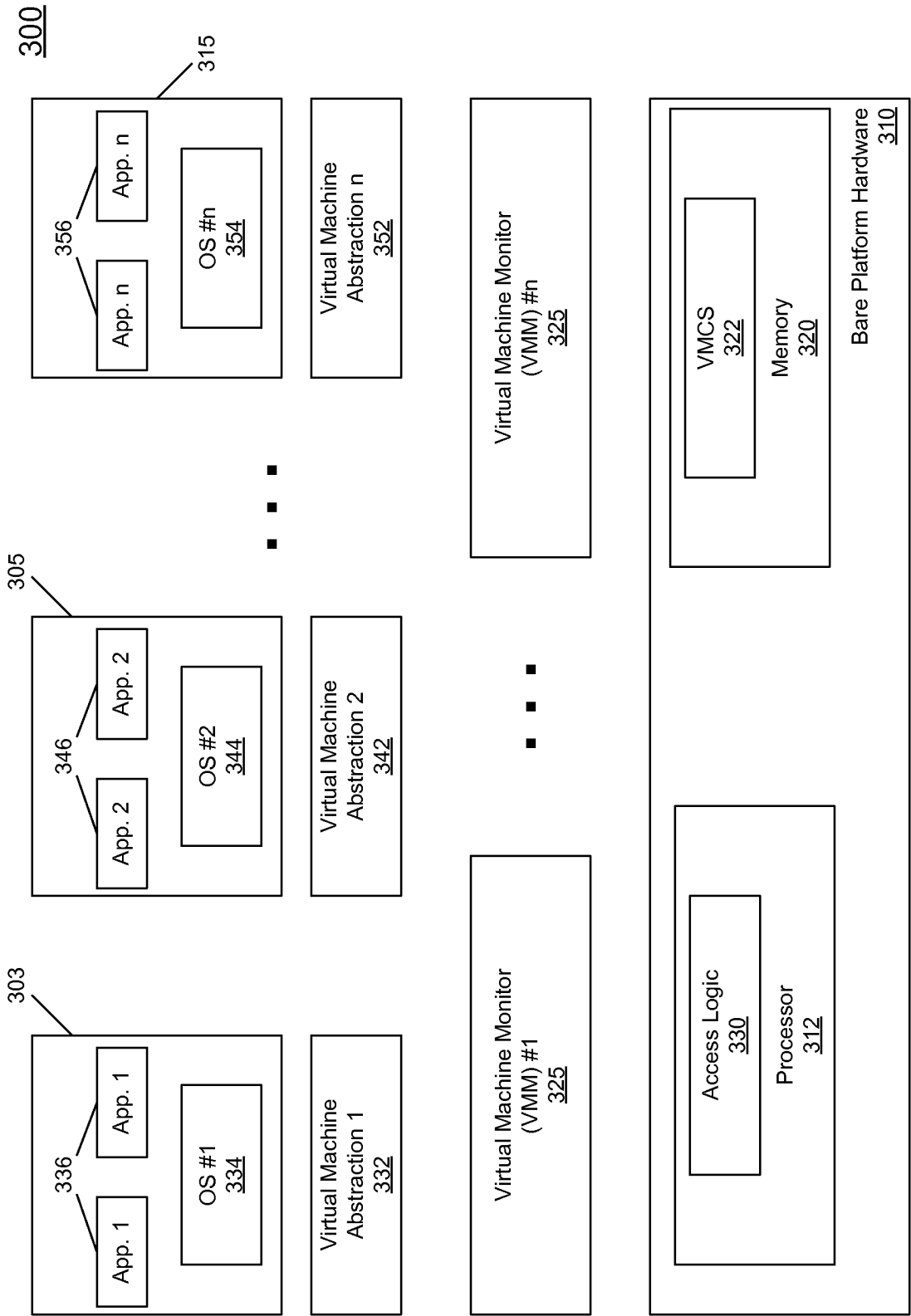


FIG. 4

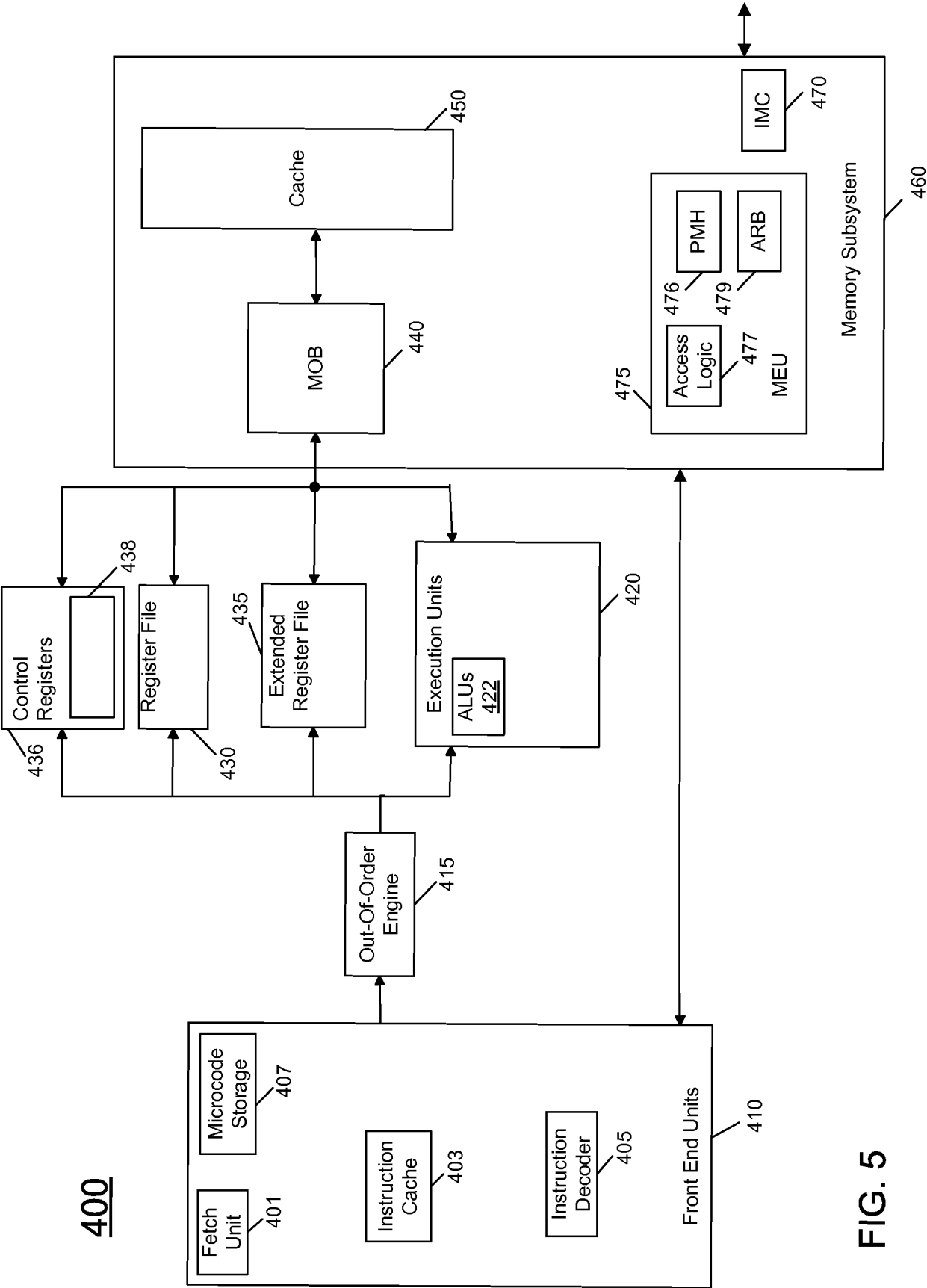


FIG. 5

500

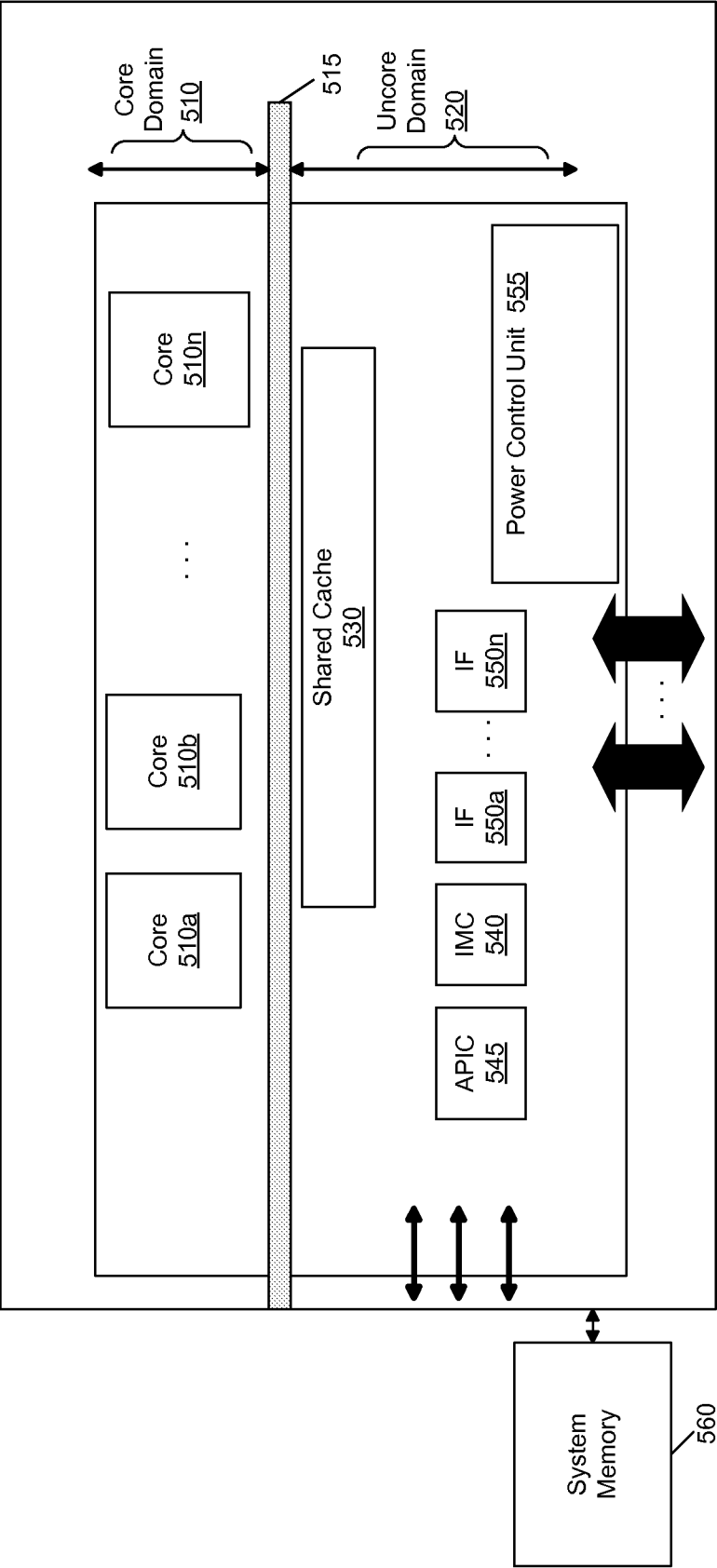


FIG. 6

7/8

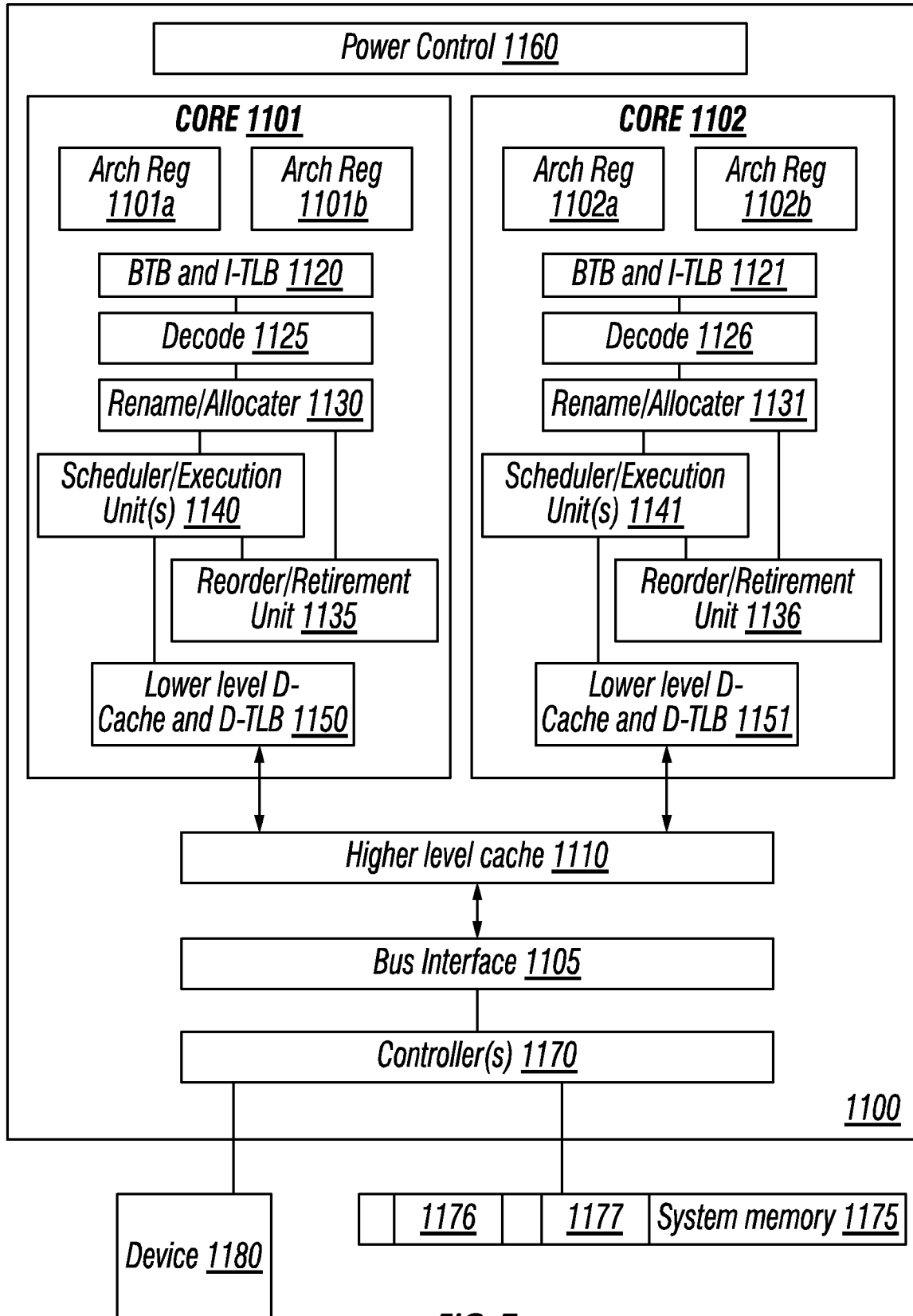


FIG. 7

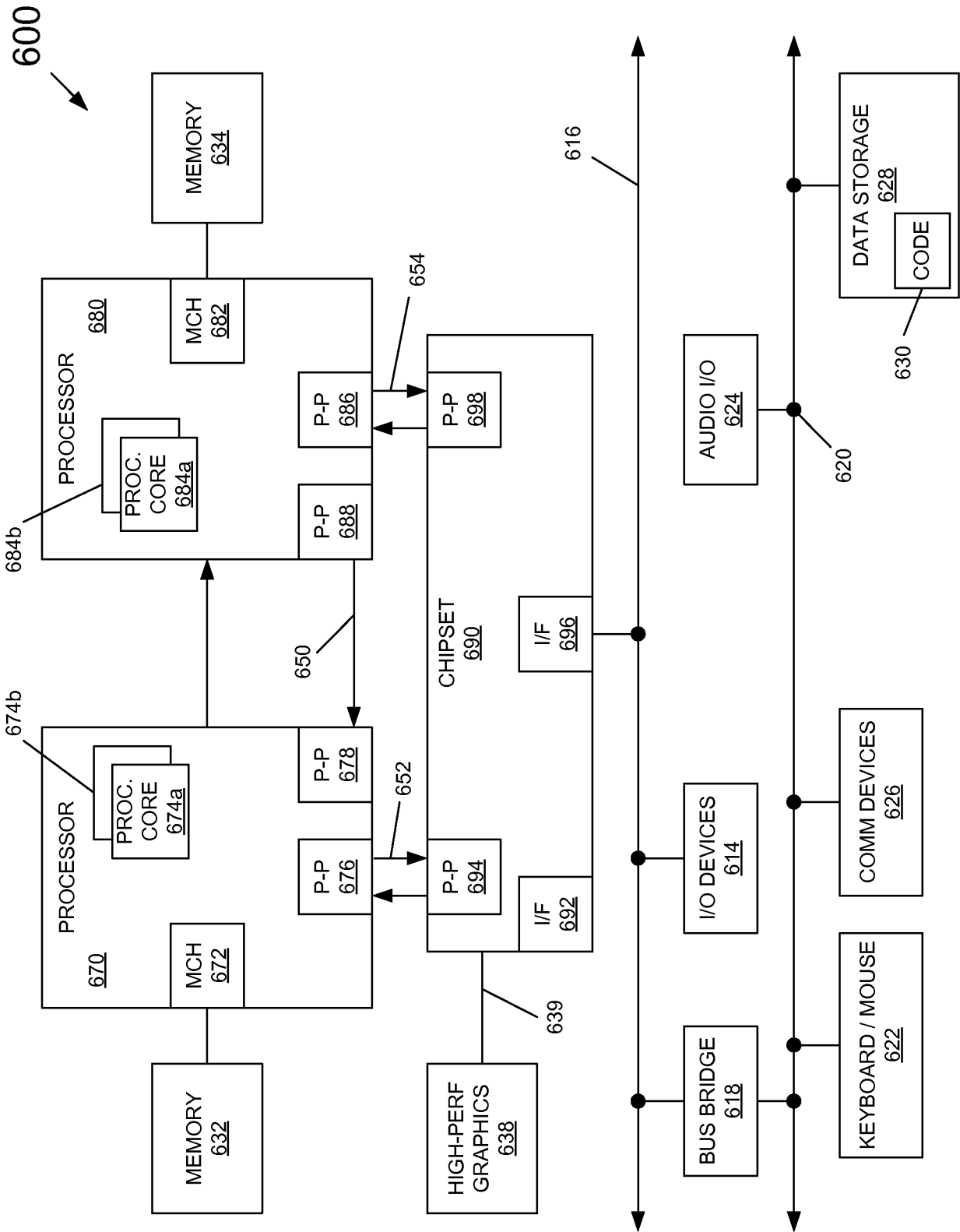


FIG. 8

## INTERNATIONAL SEARCH REPORT

International application No.  
**PCT/US2013/048793****A. CLASSIFICATION OF SUBJECT MATTER****G06F 9/48(2006.01)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 9/48; G06F 9/455; G06F 13/24

Documentation 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 models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

eKOMPASS(KIPO internal) &amp; Keywords:virtualization, APIC

**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

| Category* | Citation of document, with indication, where appropriate, of the relevant passages                                                         | Relevant to claim No. |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|
| A         | US 2007-0157197 A1 (GILBERT NEIGER et al.) 05 July 2007<br>See abstract, paragraphs [0011]j-[0029], and figures 1-3.                       | 1-24                  |
| A         | US 2010-0191885 A1 (SEREBRIN BENJAMIN C. et al.) 29 July 2010<br>See abstract, paragraphs [0005]-[0008], and figures 1-8                   | 1-24                  |
| A         | US 2011-0161541 A1 (MADUKKARUMUKUMANA RAJESH SANKARAN et al.) 30 June 2011<br>See abstract, paragraphs [0023]-[0034], and figures 1-6.     | 1-24                  |
| A         | US 2011-0197004 A1 (SEREBRIN BENJAMIN C. et al.) 11 August 2011<br>See abstract, paragraphs [0008]-[0009], [0037]-[0050], and figures 1-3. | 1-24                  |



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

28 November 2013 (28.11.2013)

Date of mailing of the international search report

**29 November 2013 (29.11.2013)**

Name and mailing address of the ISA/KR

Korean Intellectual Property Office  
189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City,  
302-701, Republic of Korea

Facsimile No. +82-42-472-7140

Authorized officer

JI, Jeong Hoon

Telephone No. +82-42-481-5688



**INTERNATIONAL SEARCH REPORT**

Information on patent family members

International application No.

**PCT/US2013/048793**

| Patent document<br>cited in search report | Publication<br>date | Patent family<br>member(s) | Publication<br>date |
|-------------------------------------------|---------------------|----------------------------|---------------------|
| US 2007-0157197 A1                        | 05/07/2007          | CN 101004726 A             | 25/07/2007          |
|                                           |                     | CN 101004726 C0            | 25/07/2007          |
|                                           |                     | CN 102945186 A             | 27/02/2013          |
|                                           |                     | EP 1804164 A1              | 04/07/2007          |
|                                           |                     | JP 05-122597B2             | 16/01/2013          |
|                                           |                     | JP 2007-183951A            | 19/07/2007          |
|                                           |                     | JP 2010-176693A            | 12/08/2010          |
|                                           |                     | TW 200818019 A             | 16/04/2008          |
|                                           |                     | TW I375913B                | 01/11/2012          |
|                                           |                     | TW I375913I                | 01/11/2012          |
|                                           |                     | US 2012-331467 A1          | 27/12/2012          |
|                                           |                     | US 8286162 B2              | 09/10/2012          |
|                                           |                     |                            |                     |
| US 2010-0191885 A1                        | 29/07/2010          | CN 102341783 A             | 01/02/2012          |
|                                           |                     | EP 2389628 A1              | 30/11/2011          |
|                                           |                     | JP 2012-515995A            | 12/07/2012          |
|                                           |                     | KR 10-2011-0113756 A       | 18/10/2011          |
|                                           |                     | KR 20110113756A            | 18/10/2011          |
|                                           |                     | US 2010-0191887 A1         | 29/07/2010          |
|                                           |                     | US 2010-0191888 A1         | 29/07/2010          |
|                                           |                     | US 2010-0191889 A1         | 29/07/2010          |
|                                           |                     | US 8055827 B2              | 08/11/2011          |
|                                           |                     | US 8180944 B2              | 15/05/2012          |
|                                           |                     | US 8234429 B2              | 31/07/2012          |
|                                           |                     | US 8234432 B2              | 31/07/2012          |
|                                           |                     | WO 2010-085804 A1          | 29/07/2010          |
|                                           |                     |                            |                     |
| US 2011-0161541 A1                        | 30/06/2011          | US 2013-232288 A1          | 05/09/2013          |
| US 2011-0197004 A1                        | 11/08/2011          | CN 102792272 A             | 21/11/2012          |
|                                           |                     | CN 102804143 A             | 28/11/2012          |
|                                           |                     | EP 2531918 A2              | 12/12/2012          |
|                                           |                     | EP 2531919 A2              | 12/12/2012          |
|                                           |                     | JP 2013-519169 A           | 23/05/2013          |
|                                           |                     | JP 2013-519170 A           | 23/05/2013          |
|                                           |                     | KR 10-2012-0131175 A       | 04/12/2012          |
|                                           |                     | KR 10-2013-0035996 A       | 09/04/2013          |
|                                           |                     | US 2011-0197003 A1         | 11/08/2011          |
|                                           |                     | US 8489789 B2              | 16/07/2013          |
|                                           |                     | WO 2011-097588 A2          | 11/08/2011          |
|                                           |                     | WO 2011-097588 A3          | 29/09/2011          |
|                                           |                     | WO 2011-097589 A2          | 11/08/2011          |
|                                           |                     | WO 2011-097589 A3          | 29/09/2011          |