



(51) International Patent Classification:  
**G06F 9/30** (2006.01)

(21) International Application Number:  
PCT/US2017/030204

(22) International Filing Date:  
28 April 2017 (28.04.2017)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:  
15/190,436 23 June 2016 (23.06.2016) US

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

(72) Inventor: **AKENINE-MOLLER, Tomas**; Rosenvägen  
13, 227 38 Lund, M (SE).

(74) Agent: **MALLIE, Michael, J.** et al.; Blakely, Sokoloff,  
Taylor & Zafman LLP, 1279 Oakmead Parkway, Sunnyvale,  
California 94085 (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,  
TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW,  
KM, ML, MR, NE, SN, TD, TG).

Published:  
— with international search report (Art. 21(3))

(54) Title: EXTENSION OF REGISTER FILES FOR LOCAL PROCESSING OF DATA IN COMPUTING ENVIRONMENTS

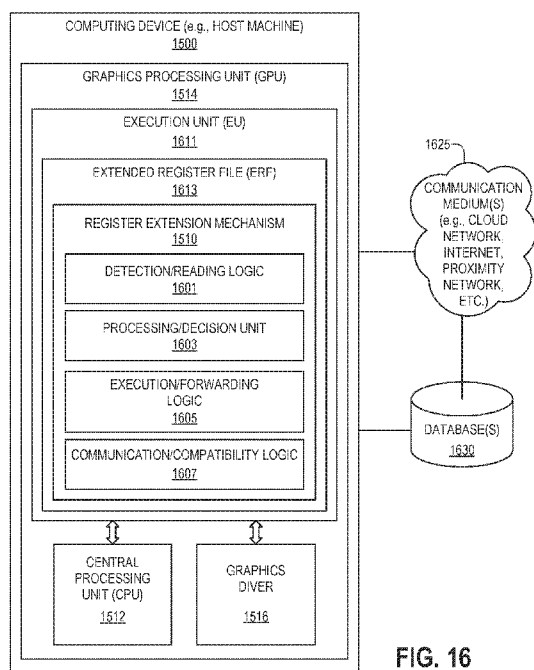


FIG. 16

(57) Abstract: A mechanism is described for facilitating extension of register files in computing environments. A method of embodiments, as described herein, includes facilitating, inside an extended register file, performance of one or more tasks relating to an instruction, where the one or more tasks are performed by an extension mechanism being hosted inside the extended register file of a computing device.



## EXTENSION OF REGISTER FILES FOR LOCAL PROCESSING OF DATA IN COMPUTING ENVIRONMENTS

### FIELD

5       **[0001]** Embodiments described herein generally relate to computers. More particularly, embodiments are described for facilitating extension of register files for local processing of data in computing environments.

### BACKGROUND

10       **[0002]** The use of registers in computing devices is well-known and thus several techniques have been developed over the years to continue to improve data processing associated with such registers. However, conventional registers are still regarded as complex and expensive, such as in terms of the number and type of ports they host and processing tasks they handle, which often results in significant inefficiency and high latency.

### BRIEF DESCRIPTION OF THE DRAWINGS

15       **[0003]** Embodiments are illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings in which like reference numerals refer to similar elements.

**[0004]** **Figure 1** is a block diagram of a processing system, according to an embodiment.

20       **[0005]** **Figure 2** is a block diagram of an embodiment of a processor having one or more processor cores, an integrated memory controller, and an integrated graphics processor.

**[0006]** **Figure 3** is a block diagram of a graphics processor, which may be a discrete graphics processing unit, or may be a graphics processor integrated with a plurality of processing cores.

25       **[0007]** **Figure 4** is a block diagram of a graphics processing engine of a graphics processor in accordance with some embodiments.

**[0008]** **Figure 5** is a block diagram of another embodiment of a graphics processor.

[0009] **Figure 6** illustrates thread execution logic including an array of processing elements employed in some embodiments of a graphics processing engine.

[00010] **Figure 7** is a block diagram illustrating a graphics processor instruction formats according to some embodiments.

5 [00011] **Figure 8** is a block diagram of another embodiment of a graphics processor.

[00012] **Figure 9A** is a block diagram illustrating a graphics processor command format according to an embodiment.

[00013] **Figure 9B** is a block diagram illustrating a graphics processor command sequence according to an embodiment.

10 [00014] **Figure 10** illustrates exemplary graphics software architecture for a data processing system according to some embodiments.

[00015] **Figure 11** is a block diagram illustrating an IP core development system that may be used to manufacture an integrated circuit to perform operations according to an embodiment.

15 [00016] **Figure 12** is a block diagram illustrating an exemplary system on a chip integrated circuit that may be fabricated using one or more IP cores, according to an embodiment.

[00017] **Figure 13** is a block diagram illustrating an exemplary graphics processor of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to an embodiment.

[00018] **Figure 14** is a block diagram illustrating an additional exemplary graphics processor of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to an embodiment.

[00019] **Figure 15** illustrates a computing device employing a register extension mechanism according to one embodiment.

25

[00020] **Figure 16** illustrates a register extension mechanism according to one embodiment.

[00021] **Figure 17** illustrates an architectural placement including a conventional register file.

5 [00022] **Figure 18** illustrates an architectural placement including an extended register file according to one embodiment.

[00023] **Figure 19** illustrates a method for facilitating and using extended register files at computing devices according to one embodiment.

#### DETAILED DESCRIPTION

10 [00024] In the following description, numerous specific details are set forth. However, embodiments, as described herein, may be practiced without these specific details. In other instances, well-known circuits, structures and techniques have not been shown in details in order not to obscure the understanding of this description.

[00025] Embodiments provide for a novel technique to extend a register file (RF) into an  
15 extended RF (ERF) by adding logic, such as hardware or firmware logic, to the register file, where the logic may be used to perform some of the operations within or inside the register file without having to go back and forth between the register file and its corresponding execution unit. For example, write ports are often more expensive than read ports (e.g., 3-4 read ports are typically supported) and further, the register area grows as the square of the number of ports  
20 grows and thus, it is desirable to reduce the number of ports in a register file. This novel technique allows for removing the need for more than one write port in a register file for certain instructions that would typically require two or more write ports. It is contemplated that embodiments are not limited as such in that for example, although 3-4 read ports are common, in some cases, this technique may also allow for using fewer read ports, such as there may be only  
25 one read port in a register file, but two read ports may be simulated by simply executing an instruction inside the register file.

[00026] It is known that each execution unit (EU) or streaming multiprocessor (SM) includes a register file that is capable of having any number of registers (e.g., single instruction, multiple data (SIMD) register). For example, in certain architectures, there may be 128 registers per thread, 7 threads per EU, while each register may be SIMD8 x 32 bits, such as 32B x 12B  
5 equaling 4kB registers per thread, where 4kB x 7 equals 28kB registers per EU. In one embodiment, a novel technique is introduced to offer an extended register file (ERF) that is capable of using merely one write port even for instructions that would normally require two or more write ports.

[00027] It is contemplated that in one embodiment, this novel register extension logic or  
10 component(s) may be implemented as part of or hosted by a register file of an EU of a graphics processor of a computing device, while, in another embodiment, this novel register extension logic or component(s) may be implemented as part of or hosted by a register file of an arithmetic logic unit (ALU) of an application processor of a computing device. In yet another embodiment, both the graphics and application processors may host the register extension logic or  
15 component(s), illustrated as register extension mechanism 1510 in **Figure 15**. It is contemplated that embodiments are not limited to any particular number or type of graphics processors, application processors, EUs, ALUs, register files, registers, and/or the like.

[00028] It is contemplated that terms like “request”, “query”, “job”, “work”, “work item”, and “workload” may be referenced interchangeably throughout this document. Similarly,  
20 an “application” or “agent” may refer to or include a computer program, a software application, a game, a workstation application, etc., offered through an application programming interface (API), such as a free rendering API, such as Open Graphics Library (OpenGL®), DirectX® 11, DirectX® 12, etc., where “dispatch” may be interchangeably referred to as “work unit” or “draw” and similarly, “application” may be interchangeably referred to as “workflow” or simply  
25 “agent”. For example, a workload, such as that of a three-dimensional (3D) game, may include and issue any number and type of “frames” where each frame may represent an image (e.g.,

sailboat, human face). Further, each frame may include and offer any number and type of work units, where each work unit may represent a part (e.g., mast of sailboat, forehead of human face) of the image (e.g., sailboat, human face) represented by its corresponding frame. However, for the sake of consistency, each item may be referenced by a single term (e.g., “dispatch”, “agent”, etc.) throughout this document.

[00029] In some embodiments, terms like “display screen” and “display surface” may be used interchangeably referring to the visible portion of a display device while the rest of the display device may be embedded into a computing device, such as a smartphone, a wearable device, etc. It is contemplated and to be noted that embodiments are not limited to any particular computing device, software application, hardware component, display device, display screen or surface, protocol, standard, etc. For example, embodiments may be applied to and used with any number and type of real-time applications on any number and type of computers, such as desktops, laptops, tablet computers, smartphones, head-mounted displays and other wearable devices, and/or the like. Further, for example, rendering scenarios for efficient performance using this novel technique may range from simple scenarios, such as desktop compositing, to complex scenarios, such as 3D games, augmented reality applications, etc.

### **System Overview**

[00030] **Figure 1** is a block diagram of a processing system 100, according to an embodiment. In various embodiments the system 100 includes one or more processors 102 and one or more graphics processors 108, and may be a single processor desktop system, a multiprocessor workstation system, or a server system having a large number of processors 102 or processor cores 107. In one embodiment, the system 100 is a processing platform incorporated within a system-on-a-chip (SoC) integrated circuit for use in mobile, handheld, or embedded devices.

[00031] An embodiment of system 100 can include, or be incorporated within a server-based gaming platform, a game console, including a game and media console, a mobile gaming

console, a handheld game console, or an online game console. In some embodiments system 100 is a mobile phone, smart phone, tablet computing device or mobile Internet device. Data processing system 100 can also include, couple with, or be integrated within a wearable device, such as a smart watch wearable device, smart eyewear device, augmented reality device, or virtual reality device. In some embodiments, data processing system 100 is a television or set top box device having one or more processors 102 and a graphical interface generated by one or more graphics processors 108.

[00032] In some embodiments, the one or more processors 102 each include one or more processor cores 107 to process instructions which, when executed, perform operations for system and user software. In some embodiments, each of the one or more processor cores 107 is configured to process a specific instruction set 109. In some embodiments, instruction set 109 may facilitate Complex Instruction Set Computing (CISC), Reduced Instruction Set Computing (RISC), or computing via a Very Long Instruction Word (VLIW). Multiple processor cores 107 may each process a different instruction set 109, which may include instructions to facilitate the emulation of other instruction sets. Processor core 107 may also include other processing devices, such a Digital Signal Processor (DSP).

[00033] In some embodiments, the processor 102 includes cache memory 104. Depending on the architecture, the processor 102 can have a single internal cache or multiple levels of internal cache. In some embodiments, the cache memory is shared among various components of the processor 102. In some embodiments, the processor 102 also uses an external cache (e.g., a Level-3 (L3) cache or Last Level Cache (LLC)) (not shown), which may be shared among processor cores 107 using known cache coherency techniques. A register file 106 is additionally included in processor 102 which may include different types of registers for storing different types of data (e.g., integer registers, floating point registers, status registers, and an instruction pointer register). Some registers may be general-purpose registers, while other registers may be specific to the design of the processor 102.

[00034] In some embodiments, processor 102 is coupled with a processor bus 110 to transmit communication signals such as address, data, or control signals between processor 102 and other components in system 100. In one embodiment the system 100 uses an exemplary 'hub' system architecture, including a memory controller hub 116 and an Input Output (I/O) controller hub 130. A memory controller hub 116 facilitates communication between a memory device and other components of system 100, while an I/O Controller Hub (ICH) 130 provides connections to I/O devices via a local I/O bus. In one embodiment, the logic of the memory controller hub 116 is integrated within the processor.

[00035] Memory device 120 can be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, phase-change memory device, or some other memory device having suitable performance to serve as process memory. In one embodiment the memory device 120 can operate as system memory for the system 100, to store data 122 and instructions 121 for use when the one or more processors 102 executes an application or process. Memory controller hub 116 also couples with an optional external graphics processor 112, which may communicate with the one or more graphics processors 108 in processors 102 to perform graphics and media operations.

[00036] In some embodiments, ICH 130 enables peripherals to connect to memory device 120 and processor 102 via a high-speed I/O bus. The I/O peripherals include, but are not limited to, an audio controller 146, a firmware interface 128, a wireless transceiver 126 (e.g., Wi-Fi, Bluetooth), a data storage device 124 (e.g., hard disk drive, flash memory, etc.), and a legacy I/O controller 140 for coupling legacy (e.g., Personal System 2 (PS/2)) devices to the system. One or more Universal Serial Bus (USB) controllers 142 connect input devices, such as keyboard and mouse 144 combinations. A network controller 134 may also couple with ICH 130. In some embodiments, a high-performance network controller (not shown) couples with processor bus 110. It will be appreciated that the system 100 shown is exemplary and not limiting, as other types of data processing systems that are differently configured may also be

used. For example, the I/O controller hub 130 may be integrated within the one or more processor 102, or the memory controller hub 116 and I/O controller hub 130 may be integrated into a discreet external graphics processor, such as the external graphics processor 112.

[00037] **Figure 2** is a block diagram of an embodiment of a processor 200 having one or more processor cores 202A-202N, an integrated memory controller 214, and an integrated graphics processor 208. Those elements of **Figure 2** having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are not limited to such. Processor 200 can include additional cores up to and including additional core 202N represented by the dashed lined boxes.

Each of processor cores 202A-202N includes one or more internal cache units 204A-204N. In some embodiments each processor core also has access to one or more shared cached units 206.

[00038] The internal cache units 204A-204N and shared cache units 206 represent a cache memory hierarchy within the processor 200. The cache memory hierarchy may include at least one level of instruction and data cache within each processor core and one or more levels of shared mid-level cache, such as a Level 2 (L2), Level 3 (L3), Level 4 (L4), or other levels of cache, where the highest level of cache before external memory is classified as the LLC. In some embodiments, cache coherency logic maintains coherency between the various cache units 206 and 204A-204N.

[00039] In some embodiments, processor 200 may also include a set of one or more bus controller units 216 and a system agent core 210. The one or more bus controller units 216 manage a set of peripheral buses, such as one or more Peripheral Component Interconnect buses (e.g., PCI, PCI Express). System agent core 210 provides management functionality for the various processor components. In some embodiments, system agent core 210 includes one or more integrated memory controllers 214 to manage access to various external memory devices (not shown).

[00040] In some embodiments, one or more of the processor cores 202A-202N include support for simultaneous multi-threading. In such embodiment, the system agent core 210 includes components for coordinating and operating cores 202A-202N during multi-threaded processing. System agent core 210 may additionally include a power control unit (PCU), which includes logic and components to regulate the power state of processor cores 202A-202N and graphics processor 208.

[00041] In some embodiments, processor 200 additionally includes graphics processor 208 to execute graphics processing operations. In some embodiments, the graphics processor 208 couples with the set of shared cache units 206, and the system agent core 210, including the one or more integrated memory controllers 214. In some embodiments, a display controller 211 is coupled with the graphics processor 208 to drive graphics processor output to one or more coupled displays. In some embodiments, display controller 211 may be a separate module coupled with the graphics processor via at least one interconnect, or may be integrated within the graphics processor 208 or system agent core 210.

[00042] In some embodiments, a ring based interconnect unit 212 is used to couple the internal components of the processor 200. However, an alternative interconnect unit may be used, such as a point-to-point interconnect, a switched interconnect, or other techniques, including techniques well known in the art. In some embodiments, graphics processor 208 couples with the ring interconnect 212 via an I/O link 213.

[00043] The exemplary I/O link 213 represents at least one of multiple varieties of I/O interconnects, including an on package I/O interconnect which facilitates communication between various processor components and a high-performance embedded memory module 218, such as an eDRAM module. In some embodiments, each of the processor cores 202A-202N and graphics processor 208 use embedded memory modules 218 as a shared Last Level Cache.

[00044] In some embodiments, processor cores 202A-202N are homogenous cores executing the same instruction set architecture. In another embodiment, processor cores 202A-

202N are heterogeneous in terms of instruction set architecture (ISA), where one or more of processor cores 202A-202N execute a first instruction set, while at least one of the other cores executes a subset of the first instruction set or a different instruction set. In one embodiment processor cores 202A-202N are heterogeneous in terms of microarchitecture, where one or more  
5 cores having a relatively higher power consumption couple with one or more power cores having a lower power consumption. Additionally, processor 200 can be implemented on one or more chips or as an SoC integrated circuit having the illustrated components, in addition to other components.

[00045] **Figure 3** is a block diagram of a graphics processor 300, which may be a  
10 discrete graphics processing unit, or may be a graphics processor integrated with a plurality of processing cores. In some embodiments, the graphics processor communicates via a memory mapped I/O interface to registers on the graphics processor and with commands placed into the processor memory. In some embodiments, graphics processor 300 includes a memory interface 314 to access memory. Memory interface 314 can be an interface to local memory, one or more  
15 internal caches, one or more shared external caches, and/or to system memory.

[00046] In some embodiments, graphics processor 300 also includes a display controller 302 to drive display output data to a display device 320. Display controller 302 includes hardware for one or more overlay planes for the display and composition of multiple layers of video or user interface elements. In some embodiments, graphics processor 300 includes a video  
20 codec engine 306 to encode, decode, or transcode media to, from, or between one or more media encoding formats, including, but not limited to Moving Picture Experts Group (MPEG) formats such as MPEG-2, Advanced Video Coding (AVC) formats such as H.264/MPEG-4 AVC, as well as the Society of Motion Picture & Television Engineers (SMPTE) 421M/VC-1, and Joint Photographic Experts Group (JPEG) formats such as JPEG, and Motion JPEG (MJPEG) formats.

25 [00047] In some embodiments, graphics processor 300 includes a block image transfer (BLIT) engine 304 to perform two-dimensional (2D) rasterizer operations including, for

example, bit-boundary block transfers. However, in one embodiment, 2D graphics operations are performed using one or more components of graphics processing engine (GPE) 310. In some embodiments, GPE 310 is a compute engine for performing graphics operations, including three-dimensional (3D) graphics operations and media operations.

5           **[00048]** In some embodiments, GPE 310 includes a 3D pipeline 312 for performing 3D operations, such as rendering three-dimensional images and scenes using processing functions that act upon 3D primitive shapes (e.g., rectangle, triangle, etc.). The 3D pipeline 312 includes programmable and fixed function elements that perform various tasks within the element and/or spawn execution threads to a 3D/Media sub-system 315. While 3D pipeline 312 can be used to  
10 perform media operations, an embodiment of GPE 310 also includes a media pipeline 316 that is specifically used to perform media operations, such as video post-processing and image enhancement.

**[00049]** In some embodiments, media pipeline 316 includes fixed function or programmable logic units to perform one or more specialized media operations, such as video  
15 decode acceleration, video de-interlacing, and video encode acceleration in place of, or on behalf of video codec engine 306. In some embodiments, media pipeline 316 additionally includes a thread spawning unit to spawn threads for execution on 3D/Media sub-system 315. The spawned threads perform computations for the media operations on one or more graphics execution units included in 3D/Media sub-system 315.

20           **[00050]** In some embodiments, 3D/Media subsystem 315 includes logic for executing threads spawned by 3D pipeline 312 and media pipeline 316. In one embodiment, the pipelines send thread execution requests to 3D/Media subsystem 315, which includes thread dispatch logic for arbitrating and dispatching the various requests to available thread execution resources. The execution resources include an array of graphics execution units to process the 3D and media  
25 threads. In some embodiments, 3D/Media subsystem 315 includes one or more internal caches for thread instructions and data. In some embodiments, the subsystem also includes shared

memory, including registers and addressable memory, to share data between threads and to store output data.

### **Graphics Processing Engine**

[00051] **Figure 4** is a block diagram of a graphics processing engine 410 of a graphics processor in accordance with some embodiments. In one embodiment, the graphics processing engine (GPE) 410 is a version of the GPE 310 shown in **Figure 3**. Elements of **Figure 4** having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are not limited to such. For example, the 3D pipeline 312 and media pipeline 316 of **Figure 3** are illustrated. The media pipeline 316 is optional in some embodiments of the GPE 410 and may not be explicitly included within the GPE 410. For example and in at least one embodiment, a separate media and/or image processor is coupled to the GPE 410.

[00052] In some embodiments, GPE 410 couples with or includes a command streamer 403, which provides a command stream to the 3D pipeline 312 and/or media pipelines 316. In some embodiments, command streamer 403 is coupled with memory, which can be system memory, or one or more of internal cache memory and shared cache memory. In some embodiments, command streamer 403 receives commands from the memory and sends the commands to 3D pipeline 312 and/or media pipeline 316. The commands are directives fetched from a ring buffer, which stores commands for the 3D pipeline 312 and media pipeline 316. In one embodiment, the ring buffer can additionally include batch command buffers storing batches of multiple commands. The commands for the 3D pipeline 312 can also include references to data stored in memory, such as but not limited to vertex and geometry data for the 3D pipeline 312 and/or image data and memory objects for the media pipeline 316. The 3D pipeline 312 and media pipeline 316 process the commands and data by performing operations via logic within the respective pipelines or by dispatching one or more execution threads to a graphics core array 414.

[00053] In various embodiments the 3D pipeline 312 can execute one or more shader programs, such as vertex shaders, geometry shaders, pixel shaders, fragment shaders, compute shaders, or other shader programs, by processing the instructions and dispatching execution threads to the graphics core array 414. The graphics core array 414 provides a unified block of execution resources. Multi-purpose execution logic (e.g., execution units) within the graphic core array 414 includes support for various 3D API shader languages and can execute multiple simultaneous execution threads associated with multiple shaders.

[00054] In some embodiments the graphics core array 414 also includes execution logic to perform media functions, such as video and/or image processing. In one embodiment, the execution units additionally include general-purpose logic that is programmable to perform parallel general purpose computational operations, in addition to graphics processing operations. The general purpose logic can perform processing operations in parallel or in conjunction with general purpose logic within the processor core(s) 107 of **Figure 1** or core 202A-202N as in **Figure 2**.

[00055] Output data generated by threads executing on the graphics core array 414 can output data to memory in a unified return buffer (URB) 418. The URB 418 can store data for multiple threads. In some embodiments the URB 418 may be used to send data between different threads executing on the graphics core array 414. In some embodiments the URB 418 may additionally be used for synchronization between threads on the graphics core array and fixed function logic within the shared function logic 420.

[00056] In some embodiments, graphics core array 414 is scalable, such that the array includes a variable number of graphics cores, each having a variable number of execution units based on the target power and performance level of GPE 410. In one embodiment the execution resources are dynamically scalable, such that execution resources may be enabled or disabled as needed.

[00057] The graphics core array 414 couples with shared function logic 420 that includes multiple resources that are shared between the graphics cores in the graphics core array. The shared functions within the shared function logic 420 are hardware logic units that provide specialized supplemental functionality to the graphics core array 414. In various embodiments, shared function logic 420 includes but is not limited to sampler 421, math 422, and inter-thread communication (ITC) 423 logic. Additionally, some embodiments implement one or more cache(s) 425 within the shared function logic 420. A shared function is implemented where the demand for a given specialized function is insufficient for inclusion within the graphics core array 414. Instead a single instantiation of that specialized function is implemented as a stand-alone entity in the shared function logic 420 and shared among the execution resources within the graphics core array 414. The precise set of functions that are shared between the graphics core array 414 and included within the graphics core array 414 varies between embodiments.

[00058] **Figure 5** is a block diagram of another embodiment of a graphics processor 500. Elements of **Figure 5** having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are not limited to such.

[00059] In some embodiments, graphics processor 500 includes a ring interconnect 502, a pipeline front-end 504, a media engine 537, and graphics cores 580A-580N. In some embodiments, ring interconnect 502 couples the graphics processor to other processing units, including other graphics processors or one or more general-purpose processor cores. In some embodiments, the graphics processor is one of many processors integrated within a multi-core processing system.

[00060] In some embodiments, graphics processor 500 receives batches of commands via ring interconnect 502. The incoming commands are interpreted by a command streamer 503 in the pipeline front-end 504. In some embodiments, graphics processor 500 includes scalable execution logic to perform 3D geometry processing and media processing via the graphics

core(s) 580A-580N. For 3D geometry processing commands, command streamer 503 supplies commands to geometry pipeline 536. For at least some media processing commands, command streamer 503 supplies the commands to a video front end 534, which couples with a media engine 537. In some embodiments, media engine 537 includes a Video Quality Engine (VQE) 530 for video and image post-processing and a multi-format encode/decode (MFX) 533 engine to provide hardware-accelerated media data encode and decode. In some embodiments, geometry pipeline 536 and media engine 537 each generate execution threads for the thread execution resources provided by at least one graphics core 580A.

[00061] In some embodiments, graphics processor 500 includes scalable thread execution resources featuring modular cores 580A-580N (sometimes referred to as core slices), each having multiple sub-cores 550A-550N, 560A-560N (sometimes referred to as core sub-slices). In some embodiments, graphics processor 500 can have any number of graphics cores 580A through 580N. In some embodiments, graphics processor 500 includes a graphics core 580A having at least a first sub-core 550A and a second sub-core 560A. In other embodiments, the graphics processor is a low power processor with a single sub-core (e.g., 550A). In some embodiments, graphics processor 500 includes multiple graphics cores 580A-580N, each including a set of first sub-cores 550A-550N and a set of second sub-cores 560A-560N. Each sub-core in the set of first sub-cores 550A-550N includes at least a first set of execution units 552A-552N and media/texture samplers 554A-554N. Each sub-core in the set of second sub-cores 560A-560N includes at least a second set of execution units 562A-562N and samplers 564A-564N. In some embodiments, each sub-core 550A-550N, 560A-560N shares a set of shared resources 570A-570N. In some embodiments, the shared resources include shared cache memory and pixel operation logic. Other shared resources may also be included in the various embodiments of the graphics processor.

#### **Execution Units**

[00062] **Figure 6** illustrates thread execution logic 600 including an array of processing elements employed in some embodiments of a GPE. Elements of **Figure 6** having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere herein, but are not limited to such.

5        [00063] In some embodiments, thread execution logic 600 includes a shader processor 602, a thread dispatcher 604, instruction cache 606, a scalable execution unit array including a plurality of execution units 608A-608N, a sampler 610, a data cache 612, and a data port 614. In one embodiment the scalable execution unit array can dynamically scale by enabling or disabling one or more execution units (e.g., any of execution unit 608A, 608B, 608C, 608D, through  
10    608N-1 and 608N) based on the computational requirements of a workload. In one embodiment the included components are interconnected via an interconnect fabric that links to each of the components. In some embodiments, thread execution logic 600 includes one or more connections to memory, such as system memory or cache memory, through one or more of instruction cache 606, data port 614, sampler 610, and execution units 608A-608N. In some  
15    embodiments, each execution unit (e.g. 608A) is a stand-alone programmable general purpose computational unit that is capable of executing multiple simultaneous hardware threads while processing multiple data elements in parallel for each thread. In various embodiments, the array of execution units 608A-608N is scalable to include any number individual execution units.

20        [00064] In some embodiments, the execution units 608A-608N are primarily used to execute shader programs. A shader processor 602 can process the various shader programs and dispatch execution threads associated with the shader programs via a thread dispatcher 604. In one embodiment the thread dispatcher includes logic to arbitrate thread initiation requests from the graphics and media pipelines and instantiate the requested threads on one or more execution unit in the execution units 608A-608N. For example, the geometry pipeline (e.g., 536 of **Figure**  
25    **5**) can dispatch vertex, tessellation, or geometry shaders to the thread execution logic 600

(Figure 6) for processing. In some embodiments, thread dispatcher 604 can also process runtime thread spawning requests from the executing shader programs.

[00065] In some embodiments, the execution units 608A-608N support an instruction set that includes native support for many standard 3D graphics shader instructions, such that

5 shader programs from graphics libraries (e.g., Direct 3D and OpenGL) are executed with a minimal translation. The execution units support vertex and geometry processing (e.g., vertex programs, geometry programs, vertex shaders), pixel processing (e.g., pixel shaders, fragment shaders) and general-purpose processing (e.g., compute and media shaders). Each of the execution units 608A-608N is capable of multi-issue single instruction multiple data (SIMD)

10 execution and multi-threaded operation enables an efficient execution environment in the face of higher latency memory accesses. Each hardware thread within each execution unit has a dedicated high-bandwidth register file and associated independent thread-state. Execution is

multi-issue per clock to pipelines capable of integer, single and double precision floating point operations, SIMD branch capability, logical operations, transcendental operations, and other

15 miscellaneous operations. While waiting for data from memory or one of the shared functions, dependency logic within the execution units 608A-608N causes a waiting thread to sleep until the requested data has been returned. While the waiting thread is sleeping, hardware resources may be devoted to processing other threads. For example, during a delay associated with a vertex shader operation, an execution unit can perform operations for a pixel shader, fragment

20 shader, or another type of shader program, including a different vertex shader.

[00066] Each execution unit in execution units 608A-608N operates on arrays of data elements. The number of data elements is the “execution size,” or the number of channels for the instruction. An execution channel is a logical unit of execution for data element access, masking, and flow control within instructions. The number of channels may be independent of

25 the number of physical Arithmetic Logic Units (ALUs) or Floating Point Units (FPUs) for a

particular graphics processor. In some embodiments, execution units 608A-608N support integer and floating-point data types.

[00067] The execution unit instruction set includes SIMD instructions. The various data elements can be stored as a packed data type in a register and the execution unit will process the various elements based on the data size of the elements. For example, when operating on a 256-bit wide vector, the 256 bits of the vector are stored in a register and the execution unit operates on the vector as four separate 64-bit packed data elements (Quad-Word (QW) size data elements), eight separate 32-bit packed data elements (Double Word (DW) size data elements), sixteen separate 16-bit packed data elements (Word (W) size data elements), or thirty-two separate 8-bit data elements (byte (B) size data elements). However, different vector widths and register sizes are possible.

[00068] One or more internal instruction caches (e.g., 606) are included in the thread execution logic 600 to cache thread instructions for the execution units. In some embodiments, one or more data caches (e.g., 612) are included to cache thread data during thread execution. In some embodiments, a sampler 610 is included to provide texture sampling for 3D operations and media sampling for media operations. In some embodiments, sampler 610 includes specialized texture or media sampling functionality to process texture or media data during the sampling process before providing the sampled data to an execution unit.

[00069] During execution, the graphics and media pipelines send thread initiation requests to thread execution logic 600 via thread spawning and dispatch logic. Once a group of geometric objects has been processed and rasterized into pixel data, pixel processor logic (e.g., pixel shader logic, fragment shader logic, etc.) within the shader processor 602 is invoked to further compute output information and cause results to be written to output surfaces (e.g., color buffers, depth buffers, stencil buffers, etc.). In some embodiments, a pixel shader or fragment shader calculates the values of the various vertex attributes that are to be interpolated across the rasterized object. In some embodiments, pixel processor logic within the shader processor 602

then executes an application programming interface (API)-supplied pixel or fragment shader program. To execute the shader program, the shader processor 602 dispatches threads to an execution unit (e.g., 608A) via thread dispatcher 604. In some embodiments, pixel shader 602 uses texture sampling logic in the sampler 610 to access texture data in texture maps stored in memory. Arithmetic operations on the texture data and the input geometry data compute pixel color data for each geometric fragment, or discards one or more pixels from further processing.

[00070] In some embodiments, the data port 614 provides a memory access mechanism for the thread execution logic 600 output processed data to memory for processing on a graphics processor output pipeline. In some embodiments, the data port 614 includes or couples to one or more cache memories (e.g., data cache 612) to cache data for memory access via the data port.

[00071] **Figure 7** is a block diagram illustrating a graphics processor instruction formats 700 according to some embodiments. In one or more embodiment, the graphics processor execution units support an instruction set having instructions in multiple formats. The solid lined boxes illustrate the components that are generally included in an execution unit instruction, while the dashed lines include components that are optional or that are only included in a sub-set of the instructions. In some embodiments, instruction format 700 described and illustrated are macro-instructions, in that they are instructions supplied to the execution unit, as opposed to micro-operations resulting from instruction decode once the instruction is processed.

[00072] In some embodiments, the graphics processor execution units natively support instructions in a 128-bit instruction format 710. A 64-bit compacted instruction format 730 is available for some instructions based on the selected instruction, instruction options, and number of operands. The native 128-bit instruction format 710 provides access to all instruction options, while some options and operations are restricted in the 64-bit format 730. The native instructions available in the 64-bit format 730 vary by embodiment. In some embodiments, the instruction is compacted in part using a set of index values in an index field 713. The execution

unit hardware references a set of compaction tables based on the index values and uses the compaction table outputs to reconstruct a native instruction in the 128-bit instruction format 710.

[00073] For each format, instruction opcode 712 defines the operation that the execution unit is to perform. The execution units execute each instruction in parallel across the multiple data elements of each operand. For example, in response to an add instruction the execution unit performs a simultaneous add operation across each color channel representing a texture element or picture element. By default, the execution unit performs each instruction across all data channels of the operands. In some embodiments, instruction control field 714 enables control over certain execution options, such as channels selection (e.g., predication) and data channel order (e.g., swizzle). For instructions in the 128-bit instruction format 710 an exec-size field 716 limits the number of data channels that will be executed in parallel. In some embodiments, exec-size field 716 is not available for use in the 64-bit compact instruction format 730.

[00074] Some execution unit instructions have up to three operands including two source operands, src0 720, src1 722, and one destination 718. In some embodiments, the execution units support dual destination instructions, where one of the destinations is implied. Data manipulation instructions can have a third source operand (e.g., SRC2 724), where the instruction opcode 712 determines the number of source operands. An instruction's last source operand can be an immediate (e.g., hard-coded) value passed with the instruction.

[00075] In some embodiments, the 128-bit instruction format 710 includes an access/address mode field 726 specifying, for example, whether direct register addressing mode or indirect register addressing mode is used. When direct register addressing mode is used, the register address of one or more operands is directly provided by bits in the instruction.

[00076] In some embodiments, the 128-bit instruction format 710 includes an access/address mode field 726, which specifies an address mode and/or an access mode for the instruction. In one embodiment the access mode is used to define a data access alignment for the instruction. Some embodiments support access modes including a 16-byte aligned access mode

and a 1-byte aligned access mode, where the byte alignment of the access mode determines the access alignment of the instruction operands. For example, when in a first mode, the instruction may use byte-aligned addressing for source and destination operands and when in a second mode, the instruction may use 16-byte-aligned addressing for all source and destination  
5 operands.

**[00077]** In one embodiment, the address mode portion of the access/address mode field 726 determines whether the instruction is to use direct or indirect addressing. When direct register addressing mode is used bits in the instruction directly provide the register address of one or more operands. When indirect register addressing mode is used, the register address of  
10 one or more operands may be computed based on an address register value and an address immediate field in the instruction.

**[00078]** In some embodiments instructions are grouped based on opcode 712 bit-fields to simplify Opcode decode 740. For an 8-bit opcode, bits 4, 5, and 6 allow the execution unit to determine the type of opcode. The precise opcode grouping shown is merely an example. In  
15 some embodiments, a move and logic opcode group 742 includes data movement and logic instructions (e.g., move (mov), compare (cmp)). In some embodiments, move and logic group 742 shares the five most significant bits (MSB), where move (mov) instructions are in the form of 0000xxxxb and logic instructions are in the form of 0001xxxxb. A flow control instruction group 744 (e.g., call, jump (jmp)) includes instructions in the form of 0010xxxxb (e.g., 0x20). A  
20 miscellaneous instruction group 746 includes a mix of instructions, including synchronization instructions (e.g., wait, send) in the form of 0011xxxxb (e.g., 0x30). A parallel math instruction group 748 includes component-wise arithmetic instructions (e.g., add, multiply (mul)) in the form of 0100xxxxb (e.g., 0x40). The parallel math group 748 performs the arithmetic operations in parallel across data channels. The vector math group 750 includes arithmetic instructions  
25 (e.g., dp4) in the form of 0101xxxxb (e.g., 0x50). The vector math group performs arithmetic such as dot product calculations on vector operands.

### **Graphics Pipeline**

[00079] **Figure 8** is a block diagram of another embodiment of a graphics processor 800. Elements of **Figure 8** having the same reference numbers (or names) as the elements of any other figure herein can operate or function in any manner similar to that described elsewhere  
5 herein, but are not limited to such.

[00080] In some embodiments, graphics processor 800 includes a graphics pipeline 820, a media pipeline 830, a display engine 840, thread execution logic 850, and a render output pipeline 870. In some embodiments, graphics processor 800 is a graphics processor within a multi-core processing system that includes one or more general purpose processing cores. The  
10 graphics processor is controlled by register writes to one or more control registers (not shown) or via commands issued to graphics processor 800 via a ring interconnect 802. In some embodiments, ring interconnect 802 couples graphics processor 800 to other processing components, such as other graphics processors or general-purpose processors. Commands from ring interconnect 802 are interpreted by a command streamer 803, which supplies instructions to  
15 individual components of graphics pipeline 820 or media pipeline 830.

[00081] In some embodiments, command streamer 803 directs the operation of a vertex fetcher 805 that reads vertex data from memory and executes vertex-processing commands provided by command streamer 803. In some embodiments, vertex fetcher 805 provides vertex data to a vertex shader 807, which performs coordinate space transformation and lighting  
20 operations to each vertex. In some embodiments, vertex fetcher 805 and vertex shader 807 execute vertex-processing instructions by dispatching execution threads to execution units 852A-852B via a thread dispatcher 831.

[00082] In some embodiments, execution units 852A-852B are an array of vector processors having an instruction set for performing graphics and media operations. In some  
25 embodiments, execution units 852A-852B have an attached L1 cache 851 that is specific for each array or shared between the arrays. The cache can be configured as a data cache, an

instruction cache, or a single cache that is partitioned to contain data and instructions in different partitions.

[00083] In some embodiments, graphics pipeline 820 includes tessellation components to perform hardware-accelerated tessellation of 3D objects. In some embodiments, a programmable hull shader 811 configures the tessellation operations. A programmable domain shader 817 provides back-end evaluation of tessellation output, where tessellator 813 operates at the direction of hull shader 811 and contains special purpose logic to generate a set of detailed geometric objects based on a coarse geometric model that is provided as input to graphics pipeline 820. In some embodiments, if tessellation is not used, tessellation components (e.g., hull shader 811, tessellator 813, and domain shader 817) can be bypassed.

[00084] In some embodiments, complete geometric objects can be processed by a geometry shader 819 via one or more threads dispatched to execution units 852A-852B, or can proceed directly to the clipper 829. In some embodiments, the geometry shader operates on entire geometric objects, rather than vertices or patches of vertices as in previous stages of the graphics pipeline. If the tessellation is disabled the geometry shader 819 receives input from the vertex shader 807. In some embodiments, geometry shader 819 is programmable by a geometry shader program to perform geometry tessellation if the tessellation units are disabled.

[00085] Before rasterization, a clipper 829 processes vertex data. The clipper 829 may be a fixed function clipper or a programmable clipper having clipping and geometry shader functions. In some embodiments, a rasterizer and depth test component 873 in the render output pipeline 870 dispatches pixel shaders to convert the geometric objects into their per pixel representations. In some embodiments, pixel shader logic is included in thread execution logic 850. In some embodiments, an application can bypass the rasterizer and depth test component 873 and access un-rasterized vertex data via a stream out unit 823.

[00086] The graphics processor 800 has an interconnect bus, interconnect fabric, or some other interconnect mechanism that allows data and message passing amongst the major

components of the processor. In some embodiments, execution units 852A-852B and associated cache(s) 851, texture and media sampler 854, and texture/sampler cache 858 interconnect via a data port 856 to perform memory access and communicate with render output pipeline components of the processor. In some embodiments, sampler 854, caches 851, 858 and  
5 execution units 852A-852B each have separate memory access paths.

[00087] In some embodiments, render output pipeline 870 contains a rasterizer and depth test component 873 that converts vertex-based objects into an associated pixel-based representation. In some embodiments, the rasterizer logic includes a windower/masker unit to perform fixed function triangle and line rasterization. An associated render cache 878 and depth  
10 cache 879 are also available in some embodiments. A pixel operations component 877 performs pixel-based operations on the data, though in some instances, pixel operations associated with 2D operations (e.g. bit block image transfers with blending) are performed by the 2D engine 841, or substituted at display time by the display controller 843 using overlay display planes. In some embodiments, a shared L3 cache 875 is available to all graphics components, allowing the  
15 sharing of data without the use of main system memory.

[00088] In some embodiments, graphics processor media pipeline 830 includes a media engine 837 and a video front end 834. In some embodiments, video front end 834 receives pipeline commands from the command streamer 803. In some embodiments, media pipeline 830 includes a separate command streamer. In some embodiments, video front-end 834 processes  
20 media commands before sending the command to the media engine 837. In some embodiments, media engine 837 includes thread spawning functionality to spawn threads for dispatch to thread execution logic 850 via thread dispatcher 831.

[00089] In some embodiments, graphics processor 800 includes a display engine 840. In some embodiments, display engine 840 is external to processor 800 and couples with the  
25 graphics processor via the ring interconnect 802, or some other interconnect bus or fabric. In some embodiments, display engine 840 includes a 2D engine 841 and a display controller 843.

In some embodiments, display engine 840 contains special purpose logic capable of operating independently of the 3D pipeline. In some embodiments, display controller 843 couples with a display device (not shown), which may be a system integrated display device, as in a laptop computer, or an external display device attached via a display device connector.

5       **[00090]** In some embodiments, graphics pipeline 820 and media pipeline 830 are configurable to perform operations based on multiple graphics and media programming interfaces and are not specific to any one application programming interface (API). In some embodiments, driver software for the graphics processor translates API calls that are specific to a particular graphics or media library into commands that can be processed by the graphics  
10 processor. In some embodiments, support is provided for the Open Graphics Library (OpenGL), Open Computing Language (OpenCL), and/or Vulkan graphics and compute API, all from the Khronos Group. In some embodiments, support may also be provided for the Direct3D library from the Microsoft Corporation. In some embodiments, a combination of these libraries may be supported. Support may also be provided for the Open Source Computer Vision Library  
15 (OpenCV). A future API with a compatible 3D pipeline would also be supported if a mapping can be made from the pipeline of the future API to the pipeline of the graphics processor.

#### **Graphics Pipeline Programming**

**[00091]** **Figure 9A** is a block diagram illustrating a graphics processor command format 900 according to some embodiments. **Figure 9B** is a block diagram illustrating a graphics  
20 processor command sequence 910 according to an embodiment. The solid lined boxes in **Figure 9A** illustrate the components that are generally included in a graphics command while the dashed lines include components that are optional or that are only included in a sub-set of the graphics commands. The exemplary graphics processor command format 900 of **Figure 9A** includes data fields to identify a target client 902 of the command, a command operation code (opcode) 904,  
25 and the relevant data 906 for the command. A sub-opcode 905 and a command size 908 are also included in some commands.

[00092] In some embodiments, client 902 specifies the client unit of the graphics device that processes the command data. In some embodiments, a graphics processor command parser examines the client field of each command to condition the further processing of the command and route the command data to the appropriate client unit. In some embodiments, the graphics processor client units include a memory interface unit, a render unit, a 2D unit, a 3D unit, and a media unit. Each client unit has a corresponding processing pipeline that processes the commands. Once the command is received by the client unit, the client unit reads the opcode 904 and, if present, sub-opcode 905 to determine the operation to perform. The client unit performs the command using information in data field 906. For some commands an explicit command size 908 is expected to specify the size of the command. In some embodiments, the command parser automatically determines the size of at least some of the commands based on the command opcode. In some embodiments commands are aligned via multiples of a double word.

[00093] The flow diagram in **Figure 9B** shows an exemplary graphics processor command sequence 910. In some embodiments, software or firmware of a data processing system that features an embodiment of a graphics processor uses a version of the command sequence shown to set up, execute, and terminate a set of graphics operations. A sample command sequence is shown and described for purposes of example only as embodiments are not limited to these specific commands or to this command sequence. Moreover, the commands may be issued as batch of commands in a command sequence, such that the graphics processor will process the sequence of commands in at least partially concurrence.

[00094] In some embodiments, the graphics processor command sequence 910 may begin with a pipeline flush command 912 to cause any active graphics pipeline to complete the currently pending commands for the pipeline. In some embodiments, the 3D pipeline 922 and the media pipeline 924 do not operate concurrently. The pipeline flush is performed to cause the active graphics pipeline to complete any pending commands. In response to a pipeline flush, the

command parser for the graphics processor will pause command processing until the active drawing engines complete pending operations and the relevant read caches are invalidated. Optionally, any data in the render cache that is marked 'dirty' can be flushed to memory. In some embodiments, pipeline flush command 912 can be used for pipeline synchronization or  
5 before placing the graphics processor into a low power state.

[00095] In some embodiments, a pipeline select command 913 is used when a command sequence requires the graphics processor to explicitly switch between pipelines. In some embodiments, a pipeline select command 913 is required only once within an execution context before issuing pipeline commands unless the context is to issue commands for both pipelines. In  
10 some embodiments, a pipeline flush command 912 is required immediately before a pipeline switch via the pipeline select command 913.

[00096] In some embodiments, a pipeline control command 914 configures a graphics pipeline for operation and is used to program the 3D pipeline 922 and the media pipeline 924. In some embodiments, pipeline control command 914 configures the pipeline state for the active  
15 pipeline. In one embodiment, the pipeline control command 914 is used for pipeline synchronization and to clear data from one or more cache memories within the active pipeline before processing a batch of commands.

[00097] In some embodiments, return buffer state commands 916 are used to configure a set of return buffers for the respective pipelines to write data. Some pipeline operations require  
20 the allocation, selection, or configuration of one or more return buffers into which the operations write intermediate data during processing. In some embodiments, the graphics processor also uses one or more return buffers to store output data and to perform cross thread communication. In some embodiments, the return buffer state 916 includes selecting the size and number of return buffers to use for a set of pipeline operations.

25 [00098] The remaining commands in the command sequence differ based on the active pipeline for operations. Based on a pipeline determination 920, the command sequence is

tailored to the 3D pipeline 922 beginning with the 3D pipeline state 930 or the media pipeline 924 beginning at the media pipeline state 940.

[00099] The commands to configure the 3D pipeline state 930 include 3D state setting commands for vertex buffer state, vertex element state, constant color state, depth buffer state, and other state variables that are to be configured before 3D primitive commands are processed. The values of these commands are determined at least in part based on the particular 3D API in use. In some embodiments, 3D pipeline state 930 commands are also able to selectively disable or bypass certain pipeline elements if those elements will not be used.

[000100] In some embodiments, 3D primitive 932 command is used to submit 3D primitives to be processed by the 3D pipeline. Commands and associated parameters that are passed to the graphics processor via the 3D primitive 932 command are forwarded to the vertex fetch function in the graphics pipeline. The vertex fetch function uses the 3D primitive 932 command data to generate vertex data structures. The vertex data structures are stored in one or more return buffers. In some embodiments, 3D primitive 932 command is used to perform vertex operations on 3D primitives via vertex shaders. To process vertex shaders, 3D pipeline 922 dispatches shader execution threads to graphics processor execution units.

[000101] In some embodiments, 3D pipeline 922 is triggered via an execute 934 command or event. In some embodiments, a register write triggers command execution. In some embodiments execution is triggered via a 'go' or 'kick' command in the command sequence. In one embodiment, command execution is triggered using a pipeline synchronization command to flush the command sequence through the graphics pipeline. The 3D pipeline will perform geometry processing for the 3D primitives. Once operations are complete, the resulting geometric objects are rasterized and the pixel engine colors the resulting pixels. Additional commands to control pixel shading and pixel back end operations may also be included for those operations.

[000102] In some embodiments, the graphics processor command sequence 910 follows the media pipeline 924 path when performing media operations. In general, the specific use and manner of programming for the media pipeline 924 depends on the media or compute operations to be performed. Specific media decode operations may be offloaded to the media pipeline  
5 during media decode. In some embodiments, the media pipeline can also be bypassed and media decode can be performed in whole or in part using resources provided by one or more general purpose processing cores. In one embodiment, the media pipeline also includes elements for general-purpose graphics processor unit (GPGPU) operations, where the graphics processor is used to perform SIMD vector operations using computational shader programs that are not  
10 explicitly related to the rendering of graphics primitives.

[000103] In some embodiments, media pipeline 924 is configured in a similar manner as the 3D pipeline 922. A set of commands to configure the media pipeline state 940 are dispatched or placed into a command queue before the media object commands 942. In some embodiments, media pipeline state commands 940 include data to configure the media pipeline  
15 elements that will be used to process the media objects. This includes data to configure the video decode and video encode logic within the media pipeline, such as encode or decode format. In some embodiments, media pipeline state commands 940 also support the use of one or more pointers to “indirect” state elements that contain a batch of state settings.

[000104] In some embodiments, media object commands 942 supply pointers to media  
20 objects for processing by the media pipeline. The media objects include memory buffers containing video data to be processed. In some embodiments, all media pipeline states must be valid before issuing a media object command 942. Once the pipeline state is configured and media object commands 942 are queued, the media pipeline 924 is triggered via an execute command 944 or an equivalent execute event (e.g., register write). Output from media pipeline  
25 924 may then be post processed by operations provided by the 3D pipeline 922 or the media

pipeline 924. In some embodiments, GPGPU operations are configured and executed in a similar manner as media operations.

### **Graphics Software Architecture**

[000105] **Figure 10** illustrates exemplary graphics software architecture for a data processing system 1000 according to some embodiments. In some embodiments, software architecture includes a 3D graphics application 1010, an operating system 1020, and at least one processor 1030. In some embodiments, processor 1030 includes a graphics processor 1032 and one or more general-purpose processor core(s) 1034. The graphics application 1010 and operating system 1020 each execute in the system memory 1050 of the data processing system.

[000106] In some embodiments, 3D graphics application 1010 contains one or more shader programs including shader instructions 1012. The shader language instructions may be in a high-level shader language, such as the High Level Shader Language (HLSL) or the OpenGL Shader Language (GLSL). The application also includes executable instructions 1014 in a machine language suitable for execution by the general-purpose processor core 1034. The application also includes graphics objects 1016 defined by vertex data.

[000107] In some embodiments, operating system 1020 is a Microsoft® Windows® operating system from the Microsoft Corporation, a proprietary UNIX-like operating system, or an open source UNIX-like operating system using a variant of the Linux kernel. The operating system 1020 can support a graphics API 1022 such as the Direct3D API, the OpenGL API, or the Vulkan API. When the Direct3D API is in use, the operating system 1020 uses a front-end shader compiler 1024 to compile any shader instructions 1012 in HLSL into a lower-level shader language. The compilation may be a just-in-time (JIT) compilation or the application can perform shader pre-compilation. In some embodiments, high-level shaders are compiled into low-level shaders during the compilation of the 3D graphics application 1010. In some embodiments, the shader instructions 1012 are provided in an intermediate form, such as a version of the Standard Portable Intermediate Representation (SPIR) used by the Vulkan API.

[000108] In some embodiments, user mode graphics driver 1026 contains a back-end shader compiler 1027 to convert the shader instructions 1012 into a hardware specific representation. When the OpenGL API is in use, shader instructions 1012 in the GLSL high-level language are passed to a user mode graphics driver 1026 for compilation. In some  
5     embodiments, user mode graphics driver 1026 uses operating system kernel mode functions 1028 to communicate with a kernel mode graphics driver 1029. In some embodiments, kernel mode graphics driver 1029 communicates with graphics processor 1032 to dispatch commands and instructions.

### **IP Core Implementations**

10     [000109] One or more aspects of at least one embodiment may be implemented by representative code stored on a machine-readable medium which represents and/or defines logic within an integrated circuit such as a processor. For example, the machine-readable medium may include instructions which represent various logic within the processor. When read by a machine, the instructions may cause the machine to fabricate the logic to perform the techniques  
15     described herein. Such representations, known as “IP cores,” are reusable units of logic for an integrated circuit that may be stored on a tangible, machine-readable medium as a hardware model that describes the structure of the integrated circuit. The hardware model may be supplied to various customers or manufacturing facilities, which load the hardware model on fabrication machines that manufacture the integrated circuit. The integrated circuit may be fabricated such  
20     that the circuit performs operations described in association with any of the embodiments described herein.

[000110] **Figure 11** is a block diagram illustrating an IP core development system 1100 that may be used to manufacture an integrated circuit to perform operations according to an embodiment. The IP core development system 1100 may be used to generate modular, re-usable  
25     designs that can be incorporated into a larger design or used to construct an entire integrated circuit (e.g., an SOC integrated circuit). A design facility 1130 can generate a software

simulation 1110 of an IP core design in a high level programming language (e.g., C/C++). The software simulation 1110 can be used to design, test, and verify the behavior of the IP core using a simulation model 1112. The simulation model 1112 may include functional, behavioral, and/or timing simulations. A register transfer level (RTL) design 1115 can then be created or  
5 synthesized from the simulation model 1112. The RTL design 1115 is an abstraction of the behavior of the integrated circuit that models the flow of digital signals between hardware registers, including the associated logic performed using the modeled digital signals. In addition to an RTL design 1115, lower-level designs at the logic level or transistor level may also be created, designed, or synthesized. Thus, the particular details of the initial design and simulation  
10 may vary.

[000111] The RTL design 1115 or equivalent may be further synthesized by the design facility into a hardware model 1120, which may be in a hardware description language (HDL), or some other representation of physical design data. The HDL may be further simulated or tested to verify the IP core design. The IP core design can be stored for delivery to a 3<sup>rd</sup> party  
15 fabrication facility 1165 using non-volatile memory 1140 (e.g., hard disk, flash memory, or any non-volatile storage medium). Alternatively, the IP core design may be transmitted (e.g., via the Internet) over a wired connection 1150 or wireless connection 1160. The fabrication facility 1165 may then fabricate an integrated circuit that is based at least in part on the IP core design. The fabricated integrated circuit can be configured to perform operations in accordance with at  
20 least one embodiment described herein.

### **Exemplary System on a Chip Integrated Circuit**

[000112] Figs. 12-14 illustrated exemplary integrated circuits and associated graphics processors that may be fabricated using one or more IP cores, according to various embodiments described herein. In addition to what is illustrated, other logic and circuits may be included,  
25 including additional graphics processors/cores, peripheral interface controllers, or general purpose processor cores.

[000113] **Figure 12** is a block diagram illustrating an exemplary system on a chip integrated circuit 1200 that may be fabricated using one or more IP cores, according to an embodiment. Exemplary integrated circuit 1200 includes one or more application processor(s) 1205 (e.g., CPUs), at least one graphics processor 1210, and may additionally include an image processor 1215 and/or a video processor 1220, any of which may be a modular IP core from the same or multiple different design facilities. Integrated circuit 1200 includes peripheral or bus logic including a USB controller 1225, UART controller 1230, an SPI/SDIO controller 1235, and an I<sup>2</sup>S/I<sup>2</sup>C controller 1240. Additionally, the integrated circuit can include a display device 1245 coupled to one or more of a high-definition multimedia interface (HDMI) controller 1250 and a mobile industry processor interface (MIPI) display interface 1255. Storage may be provided by a flash memory subsystem 1260 including flash memory and a flash memory controller. Memory interface may be provided via a memory controller 1265 for access to SDRAM or SRAM memory devices. Some integrated circuits additionally include an embedded security engine 1270.

[000114] **Figure 13** is a block diagram illustrating an exemplary graphics processor 1310 of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to an embodiment. Graphics processor 1310 can be a variant of the graphics processor 1210 of **Figure 12**. Graphics processor 1310 includes a vertex processor 1305 and one or more fragment processor(s) 1315A-1315N (e.g., 1315A, 1315B, 1315C, 1315D, through 1315N-1, and 1315N). Graphics processor 1310 can execute different shader programs via separate logic, such that the vertex processor 1305 is optimized to execute operations for vertex shader programs, while the one or more fragment processor(s) 1315A-1315N execute fragment (e.g., pixel) shading operations for fragment or pixel shader programs. The vertex processor 1305 performs the vertex processing stage of the 3D graphics pipeline and generates primitives and vertex data. The fragment processor(s) 1315A-1315N use the primitive and vertex data generated by the vertex processor 1305 to produce a framebuffer that is displayed on a display device. In one

embodiment, the fragment processor(s) 1315A-1315N are optimized to execute fragment shader programs as provided for in the OpenGL API, which may be used to perform similar operations as a pixel shader program as provided for in the Direct 3D API.

[000115] Graphics processor 1310 additionally includes one or more memory management units (MMUs) 1320A-1320B, cache(s) 1325A-1325B, and circuit interconnect(s) 1330A-1330B. The one or more MMU(s) 1320A-1320B provide for virtual to physical address mapping for integrated circuit 1310, including for the vertex processor 1305 and/or fragment processor(s) 1315A-1315N, which may reference vertex or image/texture data stored in memory, in addition to vertex or image/texture data stored in the one or more cache(s) 1325A-1325B. In one embodiment the one or more MMU(s) 1325A-1325B may be synchronized with other MMUs within the system, including one or more MMUs associated with the one or more application processor(s) 1205, image processor 1215, and/or video processor 1220 of **Figure 12**, such that each processor 1205-1220 can participate in a shared or unified virtual memory system. The one or more circuit interconnect(s) 1330A-1330B enable graphics processor 1310 to interface with other IP cores within the SoC, either via an internal bus of the SoC or via a direct connection, according to embodiments.

[000116] **Figure 14** is a block diagram illustrating an additional exemplary graphics processor 1410 of a system on a chip integrated circuit that may be fabricated using one or more IP cores, according to an embodiment. Graphics processor 1410 can be a variant of the graphics processor 1210 of **Figure 12**. Graphics processor 1410 includes the one or more MMU(s) 1320A-1320B, caches 1325A-1325B, and circuit interconnects 1330A-1330B of the integrated circuit 1300 of **Figure 13**.

[000117] Graphics processor 1410 includes one or more shader core(s) 1415A-1415N (e.g., 1415A, 1415B, 1415C, 1415D, 1415E, 1415F, through 1415N-1, and 1415N), which provides for a unified shader core architecture in which a single core or type or core can execute all types of programmable shader code, including shader program code to implement vertex

shaders, fragment shaders, and/or compute shaders. The exact number of shader cores present can vary among embodiments and implementations. Additionally, graphics processor 1410 includes an inter-core task manager 1405, which acts as a thread dispatcher to dispatch execution threads to one or more shader cores 1415A-1415N and a tiling unit 1418 to accelerate tiling operations for tile-based rendering, in which rendering operations for a scene are subdivided in image space, for example to exploit local spatial coherence within a scene or to optimize use of internal caches.

[000118] **Figure 15** illustrates a computing device 1500 employing a register extension mechanism (“extension mechanism”) 1510 according to one embodiment. Computing device 1500 (e.g., smart wearable devices, virtual reality (VR) devices, head-mounted display (HMDs), mobile computers, Internet of Things (IoT) devices, laptop computers, desktop computers, server computers, etc.) may be the same as data processing system 100 of **Figure 1** and accordingly, for brevity, clarity, and ease of understanding, many of the details stated above with reference to **Figures 1-14** are not further discussed or repeated hereafter. As illustrated, in one embodiment, computing device 1500 is shown as hosting extension mechanism 1510.

[000119] As illustrated, in one embodiment, extension mechanism 1510 may be hosted by or part of firmware of graphics processing unit (“GPU” or “graphics processor”) 1514. For example, as further illustrated with respect to **Figure 16**, extension mechanism 1510 may be hosted within or inside a register file that is part of an EU of GPU 1514, where this hosting of extension mechanism 1510 converts this GPU-based register file into an extended register file, such as extended register file 1613 of **Figure 16**.

[000120] Similarly, in one embodiment, extension mechanism 1510 may be hosted by or part of firmware of central processing unit (“CPU” or “application processor”) 1512. For example, extension mechanism 1510 may be hosted within or inside a register file that is part of an ALU of CPU 1512, where this hosting of extension mechanism 1510 converts this CPU-based register file into an extended register file. For brevity, clarity, and ease of understanding,

throughout the rest of this document, extension mechanism 1510 is shown and discussed as part of GPU 1514; however, embodiments are not limited as such.

[000121] For example, in another embodiment, extension mechanism 1510 may be hosted as software or firmware logic by operating system 1506. Similarly, in yet another embodiment, extension mechanism 1510 may be hosted by graphics driver 1516. In yet another embodiment, extension mechanism 1510 may be partially and simultaneously hosted by multiple components of computing device 1500, such as one or more of graphics driver 1516, GPU 1514, GPU firmware, CPU 1512, CPU firmware, operating system 1506, and/or the like. It is contemplated that extension mechanism 1510 or one or more of its components may be implemented as hardware, software, and/or firmware.

[000122] Throughout the document, term “user” may be interchangeably referred to as “viewer”, “observer”, “person”, “individual”, “end-user”, and/or the like. It is to be noted that throughout this document, terms like “graphics domain” may be referenced interchangeably with “graphics processing unit”, “graphics processor”, or simply “GPU” and similarly, “CPU domain” or “host domain” may be referenced interchangeably with “computer processing unit”, “application processor”, or simply “CPU”.

[000123] Computing device 1500 may include any number and type of communication devices, such as large computing systems, such as server computers, desktop computers, etc., and may further include set-top boxes (e.g., Internet-based cable television set-top boxes, etc.), global positioning system (GPS)-based devices, etc. Computing device 1500 may include mobile computing devices serving as communication devices, such as cellular phones including smartphones, personal digital assistants (PDAs), tablet computers, laptop computers, e-readers, smart televisions, television platforms, wearable devices (e.g., glasses, watches, bracelets, smartcards, jewelry, clothing items, etc.), media players, etc. For example, in one embodiment, computing device 1500 may include a mobile computing device employing a computer platform hosting an integrated circuit (“IC”), such as system on a chip (“SoC” or “SOC”), integrating

various hardware and/or software components of computing device 1500 on a single chip.

[000124] As illustrated, in one embodiment, computing device 1500 may include any number and type of hardware and/or software components, such as (without limitation) GPU 1514, graphics driver (also referred to as “GPU driver”, “graphics driver logic”, “driver logic”, user-mode driver (UMD), UMD, user-mode driver framework (UMDF), UMDF, or simply “driver”) 1516, CPU 1512, memory 1508, network devices, drivers, or the like, as well as input/output (I/O) sources 1504, such as touchscreens, touch panels, touch pads, virtual or regular keyboards, virtual or regular mice, ports, connectors, etc. Computing device 1500 may include operating system (OS) 1506 serving as an interface between hardware and/or physical resources of the computer device 1500 and a user. It is contemplated that CPU 1512 may include one or more processors, such as processor(s) 102 of **Figure 1**, while GPU 1514 may include one or more graphics processors, such as graphics processor(s) 108 of **Figure 1**.

[000125] It is to be noted that terms like “node”, “computing node”, “server”, “server device”, “cloud computer”, “cloud server”, “cloud server computer”, “machine”, “host machine”, “device”, “computing device”, “computer”, “computing system”, and the like, may be used interchangeably throughout this document. It is to be further noted that terms like “application”, “software application”, “program”, “software program”, “package”, “software package”, and the like, may be used interchangeably throughout this document. Also, terms like “job”, “input”, “request”, “message”, and the like, may be used interchangeably throughout this document.

[000126] It is contemplated and as further described with reference to **Figures 1-12**, some processes of the graphics pipeline as described above are implemented in software, while the rest are implemented in hardware. A graphics pipeline may be implemented in a graphics coprocessor design, where CPU 1512 is designed to work with GPU 1514 which may be included in or co-located with CPU 1512. In one embodiment, GPU 1514 may employ any number and type of conventional software and hardware logic to perform the conventional

functions relating to graphics rendering as well as novel software and hardware logic to execute any number and type of instructions, such as instructions 121 of **Figure 1**, to perform the various novel functions of hybrid mechanism 1510 as disclosed throughout this document.

[000127] As aforementioned, memory 1508 may include a random access memory (RAM) comprising application database having object information. A memory controller hub, such as memory controller hub 116 of **Figure 1**, may access data in the RAM and forward it to GPU 1514 for graphics pipeline processing. RAM may include double data rate RAM (DDR RAM), extended data output RAM (EDO RAM), etc. CPU 1512 interacts with a hardware graphics pipeline, as illustrated with reference to **Figure 3**, to share graphics pipelining functionality. Processed data is stored in a buffer in the hardware graphics pipeline, and state information is stored in memory 1508. The resulting image is then transferred to I/O sources 1504, such as a display component, such as display device 320 of **Figure 3**, for displaying of the image. It is contemplated that the display device may be of various types, such as Cathode Ray Tube (CRT), Thin Film Transistor (TFT), Liquid Crystal Display (LCD), Organic Light Emitting Diode (OLED) array, etc., to display information to a user.

[000128] Memory 1508 may comprise a pre-allocated region of a buffer (e.g., frame buffer); however, it should be understood by one of ordinary skill in the art that the embodiments are not so limited, and that any memory accessible to the lower graphics pipeline may be used. Computing device 1500 may further include input/output (I/O) control hub (ICH) 150 as referenced in **Figure 1**, one or more I/O sources 1504, etc.

[000129] CPU 1512 may include one or more processors to execute instructions in order to perform whatever software routines the computing system implements. The instructions frequently involve some sort of operation performed upon data. Both data and instructions may be stored in system memory 1508 and any associated cache. Cache is typically designed to have shorter latency times than system memory 1508; for example, cache might be integrated onto the same silicon chip(s) as the processor(s) and/or constructed with faster static RAM (SRAM) cells

whilst the system memory 1508 might be constructed with slower dynamic RAM (DRAM) cells. By tending to store more frequently used instructions and data in the cache as opposed to the system memory 1508, the overall performance efficiency of computing device 1500 improves. It is contemplated that in some embodiments, GPU 1514 may exist as part of CPU 1512 (such as part of a physical CPU package) in which case, memory 1508 may be shared by CPU 1512 and GPU 1514 or kept separated.

[000130] System memory 1508 may be made available to other components within the computing device 1500. For example, any data (e.g., input graphics data) received from various interfaces to the computing device 1500 (e.g., keyboard and mouse, printer port, Local Area Network (LAN) port, modem port, etc.) or retrieved from an internal storage element of the computer device 1500 (e.g., hard disk drive) are often temporarily queued into system memory 1508 prior to their being operated upon by the one or more processor(s) in the implementation of a software program. Similarly, data that a software program determines should be sent from the computing device 1500 to an outside entity through one of the computing system interfaces, or stored into an internal storage element, is often temporarily queued in system memory 1508 prior to its being transmitted or stored.

[000131] Further, for example, an ICH, such as ICH 130 of **Figure 1**, may be used for ensuring that such data is properly passed between the system memory 1508 and its appropriate corresponding computing system interface (and internal storage device if the computing system is so designed) and may have bi-directional point-to-point links between itself and the observed I/O sources/devices 1504. Similarly, an MCH, such as MCH 116 of **Figure 1**, may be used for managing the various contending requests for system memory 1508 accesses amongst CPU 1512 and GPU 1514, interfaces and internal storage elements that may proximately arise in time with respect to one another.

[000132] I/O sources 1504 may include one or more I/O devices that are implemented for transferring data to and/or from computing device 1500 (e.g., a networking adapter); or, for a

large scale non-volatile storage within computing device 1500 (e.g., hard disk drive). User input device, including alphanumeric and other keys, may be used to communicate information and command selections to GPU 1514. Another type of user input device is cursor control, such as a mouse, a trackball, a touchscreen, a touchpad, or cursor direction keys to communicate direction  
5 information and command selections to GPU 1514 and to control cursor movement on the display device. Camera and microphone arrays of computer device 1500 may be employed to observe gestures, record audio and video and to receive and transmit visual and audio commands.

[000133] Computing device 1500 may further include network interface(s) to provide  
10 access to a network, such as a LAN, a wide area network (WAN), a metropolitan area network (MAN), a personal area network (PAN), Bluetooth, a cloud network, a mobile network (e.g., 3<sup>rd</sup> Generation (3G), 4<sup>th</sup> Generation (4G), etc.), an intranet, the Internet, etc. Network interface(s) may include, for example, a wireless network interface having antenna, which may represent one or more antenna(e). Network interface(s) may also include, for example, a wired network  
15 interface to communicate with remote devices via network cable, which may be, for example, an Ethernet cable, a coaxial cable, a fiber optic cable, a serial cable, or a parallel cable.

[000134] Network interface(s) may provide access to a LAN, for example, by conforming to IEEE 802.11b and/or IEEE 802.11g standards, and/or the wireless network interface may provide access to a personal area network, for example, by conforming to  
20 Bluetooth standards. Other wireless network interfaces and/or protocols, including previous and subsequent versions of the standards, may also be supported. In addition to, or instead of, communication via the wireless LAN standards, network interface(s) may provide wireless communication using, for example, Time Division, Multiple Access (TDMA) protocols, Global Systems for Mobile Communications (GSM) protocols, Code Division, Multiple Access  
25 (CDMA) protocols, and/or any other type of wireless communications protocols.

[000135] Network interface(s) may include one or more communication interfaces,

such as a modem, a network interface card, or other well-known interface devices, such as those used for coupling to the Ethernet, token ring, or other types of physical wired or wireless attachments for purposes of providing a communication link to support a LAN or a WAN, for example. In this manner, the computer system may also be coupled to a number of peripheral devices, clients, control surfaces, consoles, or servers via a conventional network infrastructure, including an Intranet or the Internet, for example.

[000136] It is to be appreciated that a lesser or more equipped system than the example described above may be preferred for certain implementations. Therefore, the configuration of computing device 1500 may vary from implementation to implementation depending upon numerous factors, such as price constraints, performance requirements, technological improvements, or other circumstances. Examples of the electronic device or computer system 1500 may include (without limitation) a mobile device, a personal digital assistant, a mobile computing device, a smartphone, a cellular telephone, a handset, a one-way pager, a two-way pager, a messaging device, a computer, a personal computer (PC), a desktop computer, a laptop computer, a notebook computer, a handheld computer, a tablet computer, a server, a server array or server farm, a web server, a network server, an Internet server, a work station, a mini-computer, a main frame computer, a supercomputer, a network appliance, a web appliance, a distributed computing system, multiprocessor systems, processor-based systems, consumer electronics, programmable consumer electronics, television, digital television, set top box, wireless access point, base station, subscriber station, mobile subscriber center, radio network controller, router, hub, gateway, bridge, switch, machine, or combinations thereof.

[000137] Embodiments may be implemented as any or a combination of: one or more microchips or integrated circuits interconnected using a parentboard, hardwired logic, software stored by a memory device and executed by a microprocessor, firmware, an application specific integrated circuit (ASIC), and/or a field programmable gate array (FPGA). The term "logic" may include, by way of example, software or hardware and/or combinations of software and

hardware.

[000138] Embodiments may be provided, for example, as a computer program product which may include one or more machine-readable media having stored thereon machine-executable instructions that, when executed by one or more machines such as a computer,  
5 network of computers, or other electronic devices, may result in the one or more machines carrying out operations in accordance with embodiments described herein. A machine-readable medium may include, but is not limited to, floppy diskettes, optical disks, CD-ROMs (Compact Disc-Read Only Memories), and magneto-optical disks, ROMs, RAMs, EPROMs (Erasable Programmable Read Only Memories), EEPROMs (Electrically Erasable Programmable Read  
10 Only Memories), magnetic or optical cards, flash memory, or other type of media/machine-readable medium suitable for storing machine-executable instructions.

[000139] Moreover, embodiments may be downloaded as a computer program product, wherein the program may be transferred from a remote computer (e.g., a server) to a requesting computer (e.g., a client) by way of one or more data signals embodied in and/or modulated by a  
15 carrier wave or other propagation medium via a communication link (e.g., a modem and/or network connection).

[000140] **Figure 16** illustrates extension mechanism 1510 of **Figure 15** according to one embodiment. For brevity, many of the details already discussed with reference to **Figures 1-15** are not repeated or discussed hereafter. In one embodiment, extension mechanism 1510 may  
20 include any number and type of components, such as (without limitations): detection/reading logic 1601; processing/decision unit 1603; execution/forwarding logic 1605; and communication/compatibility logic 1607.

[000141] As previously discussed, extension mechanism 1510 may be implemented as part of CPU 1512, such as being part of a register file within an ALU of CPU 1512. However,  
25 for the sake of brevity and clarity, in the illustrated embodiment, extension mechanism 1510 is shown as being implemented as part of GPU 1514 to facilitate extended register file 1613 of EU

1611, where GPU 1514 is shown in communication with CPU 1512 and/or graphics driver 1516. As previously mentioned, in one embodiment, it is the hosting or implementation of extension mechanism 1510 within or inside a register file that makes the register file an extended register file, such as extended register file or simply ERF 1613.

5           **[000142]** Computing device 1500 is further shown to be in communication with one or more repositories, datasets, and/or databases, such as database(s) 1630 (e.g., cloud storage, non-cloud storage, etc.), where database(s) 1630 may reside at a local storage or a remote storage over communication medium(s) 1625, such as one or more networks (e.g., cloud network, proximity network, mobile network, intranet, Internet, etc.).

10           **[000143]** It is contemplated that a software application running at computing device 1500 may be responsible for performing or facilitating performance of any number and type of tasks using one or more components (e.g., GPU 1514, graphics driver 1516, CPU 1512, etc.) of computing device 1500. When performing such tasks, as defined by the software application, one or more components, such as GPU 1514, graphics driver 1516, CPU 1512, etc., may  
15           communicate with each other to ensure accurate and timely processing and completion of those tasks.

**[000144]** Prior to discussing the workings of extension mechanism 1510, let us elaborate on some of the previous discussion, such as to make a register file simpler, a number of write ports may be limited to merely one, while the number of read ports are to remain at 3 or  
20           even 4. For example, a multiply-and-add (MAD) instruction, such as MAD dest\_reg, src1\_reg, src2\_reg, src3\_reg that can result in  $\text{dest\_reg} = \text{src1\_reg} * \text{src2\_reg} + \text{src3\_reg}$ , where “\*\_reg” refers to register. However, issues or problems are encountered with instructions when they lead to more than one result, such as needing at least two write ports. For example a sort instruction, such as SORT dest1\_reg, dest2\_reg, src1\_reg, src2\_reg can lead to more than one result in  
25           that the smallest of src1\_reg and src2\_reg may be written back to dest1\_reg, and the bigger to dest2\_reg. This may be represented in an EU, such as EU 1611, as: 1) read src1\_reg and

src2\_reg using read ports in RF; 2) make a comparison in the EU as to whether src1\_reg > src2\_reg; 3) if true, a) write back src2\_reg to dest1\_reg in RF using existing write port, and b) write back src1\_reg to dest2\_reg in RF using existing write port, else c) write back src1\_reg to dest1\_reg in RF using existing write port, and d) write back src2\_reg to dest2\_reg in RF using existing write port; and 5) done.

[000145] Now, for example, one option might be to implement an RF with at least two write ports (such as to handle 3a and 3b (or 3c and 3d), simultaneously), but that would make the RF much more expensive to implement in terms of the number of gates. Another option might be to let 3a and 3b (or 3c and 3d) finish on different clock cycles, but this is likely to make the latency of the instruction longer, while the risk of hazards become bigger (due to longer latency), which may then need further logic support.

[000146] Continuing with the sort instruction above, such as SORT dest1\_reg, dest2\_reg, src1\_reg, src2\_reg, sorting may be performed in place where the instruction may simply be SORT src\_dest1\_reg, src\_dest2\_reg, where the two registers are now serving as both source and destination registers.

[000147] For example, the sort instruction may see to it that src\_dest1\_reg <= src\_dest2\_reg after the instruction has finished, such as sorting of the two registers happened in place (e.g., without the need for extra destination registers). However, this technique does not allow for getting rid of the writing to the two registers. If src\_dest1\_reg <= src\_dest2\_reg, then the instruction may not need to do anything; however, if that is not the case, then the content of the registers are to be swapped, which in turn necessitates two write ports.

[000148] Since swapping is expected to happen on nearly every other execution of the sort instruction (based on the likelihood of random input data), the cost associated with an extra clock cycle may be acceptable and the two registers may be written to a single existing write port, such as first written to register1 and second, on the next clock, written to register2.

Although, on average, this may be regarded as an acceptable solution, it may still complicate pipelining and further, other instructions that require more than one write port may not be as well-suited for this solution.

[000149] Referring now to extension mechanism 1510, which is implemented inside a register file converting the register file into ERF 1613, a novel technique is offered where any type and amount of processing, such as sorting or swapping in case of the aforementioned sort instruction, may be performed in place inside ERF 1613 so that none of the registers would have to be read from or written back to an EU-based processing engine (“EU engine”) of its corresponding or associated EU, such as EU 1611. For example, referring back to the sort instruction, SORT r0, r1 may result in  $r0 \leq r1$  after the instruction has been executed, so the EU engine of the corresponding EU, such as EU 1611, which is requesting two registers, r0 and r1, may simply send a short message to the RF, namely: Instruction: SORT // say 2 bits to handle 4 different instructions; Register0: 7 bits // to handle 128 registers; Register1: 7 bits, so the message may be 16 bits. It is contemplated that this sort instruction is merely used as an example, such as in case of an SIMD RF, masks may be added so that only certain SIMD lanes are sorted, etc. This can be extended in any number of ways. Also, any existing read ports are likely to be exploited to send most of the message, while some bits are needed to distinguish these operations from normal read operations.

[000150] In one embodiment, having extension mechanism 1510 inside a register file allows for extension of the register file into ERF 1613, where this ERF 1613, using certain logic or components of extension mechanism 1510, is made capable of performing and processing any number and type of instructions within or inside ERF 1613, without having to go back and forth between one or more registers of ERF 1613 and the EU engine of EU 1611, as further illustrated with respect to **Figure 18**.

[000151] Continuing with the sort instruction, such as in case of a SORT, in one embodiment, detection/reading logic 1601 may be used to detect the SORT-instruction and

then read out the two registers, Register0 and Register1, so that they can be used inside ERF 1613. In one embodiment, execution/forwarding logic 1605 may then be triggered to forward or send the contents of these two registers to processing/decision unit 1603, such having a comparison unit, to perform comparison of the contents of the two registers and  
5 allow the result of that comparison to determine whether the registers should be swapped. If the registers are not to be swapped, they remain without any change. If, however, the swap is to be performed, in one embodiment, execution/forwarding logic 1605 is triggered again to write back Register0 (which has already been read) to Register1 and similarly, write back Register1 (which has already been read) to Register0. As such, no content of  
10 any register ever leaves ERF 1613 for this instruction, making the processing far more efficient, faster, and resource-friendly than having to go back and forth between the registers and EU 1611.

[000152] Typically, it is the arithmetic/logic engine of EU, such as EU 1611, that holds the necessary processing capabilities, which is why registers in conventional register  
15 files have to communicate and go back and forth with their corresponding EUs to perform various tasks relating to instructions, such as the aforementioned comparison and decision-making tasks relating to the sort instruction. Embodiments provide for a novel technique to employ all or most or at least some of the processing capabilities locally by extending the RF into ERF 1613, such that any number and type of processes or tasks relating to any  
20 number and type of instructions may be performed or processed locally within or inside ERF 1613 by or as facilitated by various components of extension mechanism 1510.

[000153] In case of any number and type of instructions, detection/reading logic 1601 may be triggered to detect an instruction and further read from any registers of its ERF 1613 because the contents are sent over to processing/decision unit 1603 by  
25 execution/forwarding logic 1605. Once the contents are processed and decided upon, execution/forwarding logic 1603 may then be triggered to execute or apply the decisions or

subsequent processes as determined by processing/decision unit 1603, such as swapping when it is necessitated, forwarding contents or results for further processing, etc.

[000154] With regard to processing/decision unit 1603, it may include any number and type of processing units, such as one or more comparison units (capable of comparing  
5 or matching contents and/or mathematical statements or routines of registers, such as detecting equal to, greater than, less than, etc.), one or more arithmetic units (capable of performing additions, multiplications, subtractions, divisions, and other arithmetic routines, etc.), one or more functional units (capable of performing OR functions, AND  
10 functions, etc.), one or more decision units (capable of determining and deciding on the instruction or the next process relating to the instruction that is to be taken, such as whether the swap is needed or not in case of the sort instruction, etc.), and/or the like.

[000155] In some embodiments, processing/decision unit 1603 may be further used to decide whether an instruction, such as the SORT-instruction, or its relevant data is even qualified to be processed locally by extension mechanism 1510. For example, in some  
15 embodiments, selective computations or decision-making tasks may be continued on in being sent to an EU engine that is part of EU 1611 and outside ERF 1613. For example, upon receiving or detecting an incoming instruction by detection/reading logic 1601, the instruction may then be evaluated locally and in runtime by processing/decision unit 1603 to determine and decide on whether the instruction is qualified (such due to simplicity,  
20 complexity, certain computations, known factors, efficiency, speed, reducing latency, etc.) to be processed by extension mechanism 1510 inside the ERF 1613. If yes, the instruction is processed by extension mechanism 1510, but if not (such as due to certain complexities, unknown factors, dependencies, etc.), the instruction or the contents relating to the instruction may be communicated back and forth with the EU engine of EU 1611.

25 [000156] In some embodiments, the qualification of the instruction may be pre-determined, such as the instruction may be labeled with a qualification status or note as to

whether it is to be processed inside ERF 1613 by extension mechanism 1510 or remotely by the EU engine of EU 1611. In this case, processing/decision unit 1603 may be used to confirm the status of the instruction and allow for the processing to proceed accordingly.

[000157] Stated differently, processing/decision unit 1603 is not limited to any particular type or number of processes or decision-making tasks as embodiments are not limited to any particular number or type of instructions. Accordingly, it is contemplated that the aforementioned SORT-instruction is merely used as an example for brevity and clarity, but that embodiments are not limited as such.

[000158] For example, other instructions having more than one destination register may also be implemented using extension mechanism 1510 inside ERF 1613. One example is a 32-bit times 32-bit multiplication, where any overflow is stored in a second destination register. In this case, for example, a multiplication unit may be made part of processing/decision unit 1603 so that it may be made part of ERF 1613 as opposed to being EU 1611 outside of ERF 1613. Similarly, processing/decision unit 1603 and the rest of the components of extension mechanism 1510 may be further exploited and used for any number of other types of instructions, such as MAD, DOT, PLANE, etc.

[000159] Another type of instruction capable of being used with extension mechanism 1510 may relate to creating predicates for SIMD lanes, such as `setp.lt.s32 plq, a, b; // p = (a < b); q = !(a < b);` where p and q are destination registers. Note that these are SIMD instructions and that only one bit is generated per p and q per SIMD lane, where “lt” means less than, which may be replaced by any other comparison or other function (OR and AND) and s32 means that the input is signed 32-bit numbers. These are some of examples of operations that capable of being performed inside ERF 1613 using extension mechanism 1510. Further, for example, such predications may be generated with a `cmp-` instruction using one or more of the following subscripts: e (equal), n (not equal), g (greater), ge (greater or equal), l (less), and lt (less than or equal).

[000160] Similarly, such instructions may be useful for ray-tracing using the EUs, such as EU 1611, where, for example, a ray-box intersection test may involve three instances of: MIN  $r_2, r_0, r_1$  and MAX  $r_3, r_0, r_1$ , etc., when merely sort  $r_0$  and  $r_1$  may be sorted by extension mechanism 1510 so that  $r_0 \leq r_1$ . For example, for each ray, between 5 45 and 82 ray versus box intersection tests may be executed and thus, using extension mechanism 1501 may help speedup the ray tracing processes since it can help reduce the number of instructions (minimum/maximum) by 50% or 2 to 1 (using sort).

[000161] Communication/compatibility logic 1607 may be used to facilitate dynamic communication and compatibility between computing device 1500 and any number and type of 10 other computing devices (such as mobile computing device, desktop computer, server computing device, etc.); processing devices or components (such as CPUs, GPUs, etc.); capturing/sensing/detecting devices (such as capturing/sensing components including cameras, depth sensing cameras, camera sensors, red green blue (RGB) sensors, microphones, etc.); display devices (such as output components including display screens, display areas, display 15 projectors, etc.); user/context-awareness components and/or identification/verification sensors/devices (such as biometric sensors/detectors, scanners, etc.); database(s) 1630, such as memory or storage devices, databases, and/or data sources (such as data storage devices, hard drives, solid-state drives, hard disks, memory cards or devices, memory circuits, etc.); communication medium(s) 1625, such as one or more communication channels or networks 20 (e.g., Cloud network, the Internet, intranet, cellular network, proximity networks, such as Bluetooth, Bluetooth low energy (BLE), Bluetooth Smart, Wi-Fi proximity, Radio Frequency Identification (RFID), Near Field Communication (NFC), Body Area Network (BAN), etc.); wireless or wired communications and relevant protocols (e.g., Wi-Fi®, WiMAX, Ethernet, etc.); connectivity and location management techniques; software applications/websites (e.g., 25 social and/or business networking websites, etc., business applications, games and other entertainment applications, etc.); and programming languages, etc., while ensuring compatibility

with changing technologies, parameters, protocols, standards, etc.

[000162] Throughout this document, terms like "logic", "component", "module", "framework", "engine", "mechanism", and the like, may be referenced interchangeably and include, by way of example, software, hardware, and/or any combination of software and hardware, such as firmware. In one example, "logic" may refer to or include a software component that is capable of working with one or more of an operating system (e.g., operating system 1506), a graphics driver (e.g., graphics driver 1516), etc., of a computing device, such as computing device 1500. In another example, "logic" may refer to or include a hardware component that is capable of being physically installed along with or as part of one or more system hardware elements, such as an application processor (e.g., CPU 1512), a graphics processor (e.g., GPU 1514), etc., of a computing device, such as computing device 1500. In yet another embodiment, "logic" may refer to or include a firmware component that is capable of being part of system firmware, such as firmware of an application processor (e.g., CPU 1512) or a graphics processor (e.g., GPU 1514), etc., of a computing device, such as computing device 1500.

[000163] Further, any use of a particular brand, word, term, phrase, name, and/or acronym, such as "GPU", "GPU domain", "GPGPU", "CPU", "CPU domain", "graphics driver", "workload", "application", "graphics pipeline", "pipeline processes", "register", "register file", "RF", "extended register file", "ERF", "execution unit", "EU", "instruction", "API", "3D API", "OpenGL®", "DirectX®", "fragment shader", "YUV texture", "shader execution", "existing UAV capabilities", "existing backend", "hardware", "software", "agent", "graphics driver", "kernel mode graphics driver", "user-mode driver", "user-mode driver framework", "buffer", "graphics buffer", "task", "process", "operation", "software application", "game", etc., should not be read to limit embodiments to software or devices that carry that label in products or in literature external to this document.

[000164] It is contemplated that any number and type of components may be added to

and/or removed from extension mechanism 1510 to facilitate various embodiments including adding, removing, and/or enhancing certain features. For brevity, clarity, and ease of understanding of extension mechanism 1510, many of the standard and/or known components, such as those of a computing device, are not shown or discussed here. It is contemplated that  
5   embodiments, as described herein, are not limited to any particular technology, topology, system, architecture, and/or standard and are dynamic enough to adopt and adapt to any future changes.

[000165] **Figure 17** illustrates an architectural setup having an execution unit 1701 employing a conventional register file 1711. For brevity, many of the details previously discussed with reference to **Figures 1-16** may not be discussed or repeated hereafter. As  
10   illustrated, EU 1701 includes RF 1711 which includes register 0 1713 and register 1 1715. EU 1701 is further shown as having processing engine 1703 (e.g., arithmetic/logic engine) in communication with registers 0 1713 and 1 1715 for performing any number and type of processes relating to contents of registers 0 1713 and/or 1 1713 based on instructions received at RF 1711. Stated different, for each instruction-related process or tasks, registers 1713, 1715 are  
15   expected to communicated back and forth with processing engine 1703, which wastes valuable processing resources, causes latency in clock cycles, and/or the like.

[000166] **Figure 18** illustrates an architectural placement having an execution unit 1611 employing an extended register file 1613 according to one embodiment. For brevity, many of the details previously discussed with reference to **Figures 1-17** may not be discussed or repeated  
20   hereafter. It is to be noted that embodiments are not limited to any particular number or type of use-case scenarios, component setups, architectural placements, etc., such as this illustrated architectural placement of EU 1611. Further, it is contemplated and to be noted that for brevity and simplicity, EU 1611 is shown being part of and employed at a graphics processor, such as GPU 1514 of **Figure 15**, but embodiments are not limited as such in that EU 1611 may very well  
25   be an ALU being part of and employed at an application processor, such as CPU 1512 of **Figure 15**.

[000167] In the illustrated embodiment, EU 1611 is shown as having processing engine 1803 (e.g., arithmetic/logic engine) in communication with registers 0 1813 and 1 1815 for performing any number and type of processes relating to various instructions and/or instruction-based contents at registers 0 1813 and/or 1 1813. In the illustrated embodiment, ERF 1613 is not  
5 a conventional RF, such as RF 1711 of **Figure 17**, but that this, ERF 1613, is extended or modified by having employed extension mechanism 1510. As further illustrated and previously discussed with reference to **Figures 15-16**, in one embodiment, register 0 1813 and 1 1815 are shown in communication with extension mechanism 1510, where extension mechanism 1510 is used to perform any number and type of instruction-related processes and tasks (such as  
10 comparison, multiplication, addition, decision-making, etc.) without having to refer to or rely on processing engine 1803.

[000168] In one embodiment, this novel technique of using a localized logic, such as extension mechanism 1510, that is inside a register file, such as ERF 1613, allows for much faster and efficient processing of instruction-related tasks, while further allowing for the use of  
15 merely one write port in ERF 1613 even for those instructions that normally necessitate two or more ports. It is contemplated that embodiments are not limited to any of these illustrations, such as the architectural placement, logical components setup, etc., of **Figures 15, 16 and 18**. For example, it is to be noted that in some embodiments, an RF and/or ERF 1613, may be located outside next to EU 1611, as opposed to inside EU 1611 as shown in **Figures 16 and 18**,  
20 such that EU 1611 may be connected to the read and write ports of RF and/or ERF 1611.

[000169] **Figure 19** illustrates a method 1900 for facilitating and using extended register files at computing devices according to one embodiment. Method 1900 may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, programmable logic, etc.), software (such as instructions run on a processing device), or a  
25 combination thereof, as facilitated by extension mechanism 1510 of **Figure 15**. The processes of method 1900 are illustrated in linear sequences for brevity and clarity in presentation; however,

it is contemplated that any number of them can be performed in parallel, asynchronously, or in different orders. For brevity, many of the details discussed with reference to **Figures 1-18** may not be discussed or repeated hereafter.

[000170] Method 1900 begins at block 1901 with detecting or receiving of an  
5 instruction at an extended register file of an execution unit of a graphics processor at a computing device, as facilitated by detection/reading logic 1601 of **Figure 16**. In another embodiment, the extended register file may be part of an arithmetic logic unit of an application processor of the computing device. At block 1903, a determination may be made by the EU as to whether the instruction is qualified to be processed inside the ERF by extension mechanism  
10 1510 of **Figure 15**, where this qualifying of the instruction may be based on whether the ERF can execute the relevant work relating to the instruction inside the ERF and if so, a task is sent by the EU to the ERF to carry out the relevant work at block 1905.

[000171] Once the task is received, at block 1907, the task may be executed to carry out the relevant work, such as performing arithmetic computations, swapping contents of registers,  
15 executing or applying any decisions, forwarding contents for further processing, etc., such as by execution/forwarding logic 1605 of **Figure 16**.

[000172] If the instruction is not qualified or cannot be processed inside the ERF, it may then be processed by the EU, such as an arithmetic/logic engine of the EU. Once the instruction is processed, whether it done by the EU or inside the ERF, method 1900 then ends at  
20 block 1911.

[000173] References to “one embodiment”, “an embodiment”, “example embodiment”, “various embodiments”, etc., indicate that the embodiment(s) so described may include particular features, structures, or characteristics, but not every embodiment necessarily includes the particular features, structures, or characteristics. Further, some embodiments may have  
25 some, all, or none of the features described for other embodiments.

[000174] In the foregoing specification, embodiments have been described with

reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of embodiments as set forth in the appended claims. The Specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

5       **[000175]** In the following description and claims, the term “coupled” along with its derivatives, may be used. “Coupled” is used to indicate that two or more elements co-operate or interact with each other, but they may or may not have intervening physical or electrical components between them.

10       **[000176]** As used in the claims, unless otherwise specified the use of the ordinal adjectives “first”, “second”, “third”, etc., to describe a common element, merely indicate that different instances of like elements are being referred to, and are not intended to imply that the elements so described must be in a given sequence, either temporally, spatially, in ranking, or in any other manner.

15       **[000177]** The following clauses and/or examples pertain to further embodiments or examples. Specifics in the examples may be used anywhere in one or more embodiments. The various features of the different embodiments or examples may be variously combined with some features included and others excluded to suit a variety of different applications. Examples may include subject matter such as a method, means for performing acts of the method, at least one machine-readable medium including instructions that, when performed by a machine cause  
20       the machine to performs acts of the method, or of an apparatus or system for facilitating hybrid communication according to embodiments and examples described herein.

25       **[000178]** Some embodiments pertain to Example 1 that includes an apparatus to facilitate extension of register files in computing environments, the apparatus comprising: an extended register file having registers and an extension mechanism, wherein the extension mechanism to facilitate, inside the extended register file, performance of one or more tasks relating to an instruction.

[000179] Example 2 includes the subject matter of Example 1, further comprising a graphics processor having an execution unit, wherein the execution unit to host the extended register file.

5 [000180] Example 3 includes the subject matter of Example 1, further comprising an application processor having an arithmetic logic unit, wherein the arithmetic logic unit to host the extended register file.

[000181] Example 4 includes the subject matter of Example 1, further comprising: detection/reading logic to detect the instruction; and processing/decision unit to process the one or more tasks, wherein processing of the one or more tasks includes managing one or more  
10 operations relating to contents of one or more of the registers, wherein the one or more operations include one or more of a comparison operation, a swapping operation, an arithmetic operation, and a decision-making operation.

[000182] Example 5 includes the subject matter of Example 4, further comprising execution/forwarding logic to execute results associated with the one or more operations to  
15 complete the performance of the one or more tasks relating to the instruction, wherein the execution/forwarding logic is further to facilitate communication of at least one of the results, the contents, and other relevant data within or between one or more of the extension mechanism, the extended register file, the execution unit, and the arithmetic logic unit.

[000183] Example 6 includes the subject matter of Example 2, wherein the execution  
20 unit to determine whether the instruction is qualified to be processed by the extension mechanism inside the extended register file, wherein if the instruction is qualified, the one or more tasks relating to the instruction are performed by the extension mechanism inside the extended register file.

[000184] Example 7 includes the subject matter of Example 6, wherein if the instruction  
25 is not qualified, the one or more tasks relating to the instruction are performed by a processing

engine of the execution unit, wherein the execution unit-based processing engine includes an arithmetic/logic unit located outside the extended register file.

[000185] Example 8 includes the subject matter of Example 3, wherein the arithmetic logic unit to determine whether the instruction is qualified to be processed by the extension  
5 mechanism inside the extended register file, wherein if the instruction is qualified, the one or more tasks relating to the instruction are performed by the extension mechanism inside the extended register file, and wherein if the instruction is not qualified, the one or more tasks relating to the instruction are performed by a processing engine of the arithmetic logic unit, wherein the arithmetic logic unit-based processing engine includes an arithmetic/logic engine  
10 located outside the extended register file.

[000186] Some embodiments pertain to Example 9 that includes a method for facilitating extension of register files in computing environments, the method comprising: facilitating, inside an extended register file, performance of one or more tasks relating to an instruction, wherein the one or more tasks are performed by an extension mechanism being  
15 hosted inside the extended register file of a computing device.

[000187] Example 10 includes the subject matter of Example 9, further comprising: facilitating an execution unit of a graphics processor to host the extended register.

[000188] Example 11 includes the subject matter of Example 9, further comprising: facilitating an arithmetic logic unit of an application processor to host the extended register file.

20 [000189] Example 12 includes the subject matter of Example 9, further comprising: detecting the instruction; and processing the one or more tasks, wherein processing of the one or more tasks includes managing one or more operations relating to contents of one or more of the registers, wherein the one or more operations include one or more of a comparison operation, a swapping operation, an arithmetic operation, and a decision-making operation.

25 [000190] Example 13 includes the subject matter of Example 12, further comprising: executing results associated with the one or more operations to complete the performance of the

one or more tasks relating to the instruction; and facilitating communication of at least one of the results, the contents, and other relevant data within or between one or more of the extension mechanism, the extended register file, the execution unit, and the arithmetic logic unit.

**[000191]** Example 14 includes the subject matter of Example 10, further comprising:

5 determining, by the execution unit, whether the instruction is qualified to be processed by the extension mechanism inside the extended register file, wherein if the instruction is qualified, the one or more tasks relating to the instruction are performed by the extension mechanism inside the extended register file.

**[000192]** Example 15 includes the subject matter of Example 14, wherein if the

10 instruction is not qualified, the one or more tasks relating to the instruction are performed by a processing engine of the execution unit, wherein the execution unit-based processing engine includes an arithmetic/logic unit located outside the extended register file.

**[000193]** Example 16 includes the subject matter of Example 11, further comprising:

determining, by the arithmetic logic unit, whether the instruction is qualified to be processed by  
15 the extension mechanism inside the extended register file, wherein if the instruction is qualified, the one or more tasks relating to the instruction are performed by the extension mechanism inside the extended register file, and wherein if the instruction is not qualified, the one or more tasks relating to the instruction are performed by a processing engine of the arithmetic logic unit, wherein the arithmetic logic unit-based processing engine includes an arithmetic/logic engine  
20 located outside the extended register file.

**[000194]** Some embodiments pertain to Example 17 includes a system comprising a

computing device including a storage device having instructions, and a processor to execute the instructions, wherein the system comprises: an extended register file having registers and an extension mechanism, wherein the extension mechanism to facilitate, inside the extended register  
25 file, performance of one or more tasks relating to an instruction.

[000195] Example 18 includes the subject matter of Example 17, further comprising a graphics processor having an execution unit, wherein the execution unit to host the extended register file.

[000196] Example 19 includes the subject matter of Example 17, further comprising an application processor having an arithmetic logic unit, wherein the arithmetic logic unit to host the extended register file.

[000197] Example 20 includes the subject matter of Example 17, wherein the extension mechanism to: detect the instruction; and process the one or more tasks, wherein processing of the one or more tasks includes managing one or more operations relating to contents of one or more of the registers, wherein the one or more operations include one or more of a comparison operation, a swapping operation, an arithmetic operation, and a decision-making operation.

[000198] Example 21 includes the subject matter of Example 20, wherein the extension mechanism to: execute results associated with the one or more operations to complete the performance of the one or more tasks relating to the instruction; and facilitate communication of at least one of the results, the contents, and other relevant data within or between one or more of the extension mechanism, the extended register file, the execution unit, and the arithmetic logic unit.

[000199] Example 22 includes the subject matter of Example 18, wherein the execution unit to determine whether the instruction is qualified to be processed by the extension mechanism inside the extended register file, wherein if the instruction is qualified, the one or more tasks relating to the instruction are performed by the extension mechanism inside the extended register file.

[000200] Example 23 includes the subject matter of Example 22, wherein if the instruction is not qualified, the one or more tasks relating to the instruction are performed by a processing engine of the execution unit, wherein the execution unit-based processing engine includes an arithmetic/logic unit located outside the extended register file.

[000201] Example 24 includes the subject matter of Example 19, wherein the arithmetic logic unit to determine whether the instruction is qualified to be processed by the extension mechanism inside the extended register file, wherein if the instruction is qualified, the one or more tasks relating to the instruction are performed by the extension mechanism inside the extended register file, and wherein if the instruction is not qualified, the one or more tasks relating to the instruction are performed by a processing engine of the arithmetic logic unit, wherein the arithmetic logic unit-based processing engine includes an arithmetic/logic engine located outside the extended register file.

[000202] Some embodiments pertain to Example 25 includes an apparatus comprising: means for facilitating, inside an extended register file, performance of one or more tasks relating to an instruction, wherein the one or more tasks are performed by an extension mechanism being hosted inside the extended register file of a computing device.

[000203] Example 26 includes the subject matter of Example 25, further comprising: means for facilitating an execution unit of a graphics processor to host the extended register.

[000204] Example 27 includes the subject matter of Example 25, further comprising: means for facilitating an arithmetic logic unit of an application processor to host the extended register file.

[000205] Example 28 includes the subject matter of Example 25, further comprising: means for detecting the instruction; and means for processing the one or more tasks, wherein processing of the one or more tasks includes managing one or more operations relating to contents of one or more of the registers, wherein the one or more operations include one or more of a comparison operation, a swapping operation, an arithmetic operation, and a decision-making operation.

[000206] Example 29 includes the subject matter of Example 28, further comprising: means for executing results associated with the one or more operations to complete the performance of the one or more tasks relating to the instruction; and means for facilitating

communication of at least one of the results, the contents, and other relevant data within or between one or more of the extension mechanism, the extended register file, the execution unit, and the arithmetic logic unit.

**[000207]** Example 30 includes the subject matter of Example 26, further comprising:

5 means for determining, by the execution unit, whether the instruction is qualified to be processed by the extension mechanism inside the extended register file, wherein if the instruction is qualified, the one or more tasks relating to the instruction are performed by the extension mechanism inside the extended register file.

**[000208]** Example 31 includes the subject matter of Example 30, wherein if the

10 instruction is not qualified, the one or more tasks relating to the instruction are performed by a processing engine of the execution unit, wherein the execution unit-based processing engine includes an arithmetic/logic unit located outside the extended register file.

**[000209]** Example 32 includes the subject matter of Example 27, further comprising:

means for determining, by the arithmetic logic unit, whether the instruction is qualified to be  
15 processed by the extension mechanism inside the extended register file, wherein if the instruction is qualified, the one or more tasks relating to the instruction are performed by the extension mechanism inside the extended register file, and wherein if the instruction is not qualified, the one or more tasks relating to the instruction are performed by a processing engine of the arithmetic logic unit, wherein the arithmetic logic unit-based processing engine includes an  
20 arithmetic/logic engine located outside the extended register file.

**[000210]** Example 33 includes at least one non-transitory or tangible machine-readable

medium comprising a plurality of instructions, when executed on a computing device, to implement or perform a method as claimed in any of claims or examples 9-16.

**[000211]** Example 34 includes at least one machine-readable medium comprising a

25 plurality of instructions, when executed on a computing device, to implement or perform a method as claimed in any of claims or examples 9-16.

[000212] Example 35 includes a system comprising a mechanism to implement or perform a method as claimed in any of claims or examples 9-16.

[000213] Example 36 includes an apparatus comprising means for performing a method as claimed in any of claims or examples 9-16.

5 [000214] Example 37 includes a computing device arranged to implement or perform a method as claimed in any of claims or examples 9-16.

[000215] Example 38 includes a communications device arranged to implement or perform a method as claimed in any of claims or examples 9-16.

10 [000216] Example 39 includes at least one machine-readable medium comprising a plurality of instructions, when executed on a computing device, to implement or perform a method or realize an apparatus as claimed in any preceding claims.

[000217] Example 40 includes at least one non-transitory or tangible machine-readable medium comprising a plurality of instructions, when executed on a computing device, to implement or perform a method or realize an apparatus as claimed in any preceding claims.

15 [000218] Example 41 includes a system comprising a mechanism to implement or perform a method or realize an apparatus as claimed in any preceding claims.

[000219] Example 42 includes an apparatus comprising means to perform a method as claimed in any preceding claims.

20 [000220] Example 43 includes a computing device arranged to implement or perform a method or realize an apparatus as claimed in any preceding claims.

[000221] Example 44 includes a communications device arranged to implement or perform a method or realize an apparatus as claimed in any preceding claims.

[000222] The drawings and the forgoing description give examples of embodiments. Those skilled in the art will appreciate that one or more of the described elements may well be combined into a single functional element. Alternatively, certain elements may be split into  
25 multiple functional elements. Elements from one embodiment may be added to another

embodiment. For example, orders of processes described herein may be changed and are not limited to the manner described herein. Moreover, the actions of any flow diagram need not be implemented in the order shown; nor do all of the acts necessarily need to be performed. Also, those acts that are not dependent on other acts may be performed in parallel with the other acts.

- 5 The scope of embodiments is by no means limited by these specific examples. Numerous variations, whether explicitly given in the specification or not, such as differences in structure, dimension, and use of material, are possible. The scope of embodiments is at least as broad as given by the following claims.

## CLAIMS

What is claimed is:

1. An apparatus to facilitate extension of register files in computing environments,  
5 the apparatus comprising:

an extended register file having registers and an extension mechanism, wherein the extension mechanism to facilitate, inside the extended register file, performance of one or more tasks relating to an instruction.

2. The apparatus of claim 1, further comprising a graphics processor having an  
10 execution unit, wherein the execution unit to host the extended register file.

3. The apparatus of claim 1, further comprising an application processor having an arithmetic logic unit, wherein the arithmetic logic unit to host the extended register file.

4. The apparatus of claim 1, further comprising:  
detection/reading logic to detect the instruction; and  
15 processing/decision unit to process the one or more tasks, wherein processing of the one or more tasks includes managing one or more operations relating to contents of one or more of the registers, wherein the one or more operations include one or more of a comparison operation, a swapping operation, an arithmetic operation, and a decision-making operation.

5. The apparatus of claim 4, further comprising execution/forwarding logic to  
20 execute results associated with the one or more operations to complete the performance of the one or more tasks relating to the instruction, wherein the execution/forwarding logic is further to facilitate communication of at least one of the results, the contents, and other relevant data within or between one or more of the extension mechanism, the extended register file, the execution unit, and the arithmetic logic unit.

25 6. The apparatus of claim 2, wherein the execution unit to determine whether the instruction is qualified to be processed by the extension mechanism inside the extended register file,

wherein if the instruction is qualified, the one or more tasks relating to the instruction are performed by the extension mechanism inside the extended register file.

7. The apparatus of claim 6, wherein if the instruction is not qualified, the one or more tasks relating to the instruction are performed by a processing engine of the execution unit,  
5 wherein the execution unit-based processing engine includes an arithmetic/logic unit located outside the extended register file.

8. The apparatus of claim 3, wherein the arithmetic logic unit to determine whether the instruction is qualified to be processed by the extension mechanism inside the extended register file,

10 wherein if the instruction is qualified, the one or more tasks relating to the instruction are performed by the extension mechanism inside the extended register file, and

wherein if the instruction is not qualified, the one or more tasks relating to the instruction are performed by a processing engine of the arithmetic logic unit, wherein the arithmetic logic unit-based processing engine includes an arithmetic/logic engine located outside the extended  
15 register file.

9. A method for facilitating extension of register files in computing environments, the method comprising:

facilitating, inside an extended register file, performance of one or more tasks relating to an instruction, wherein the one or more tasks are performed by an extension mechanism being  
20 hosted inside the extended register file of a computing device.

10. The method of claim 9, further comprising: facilitating an execution unit of a graphics processor to host the extended register.

11. The method of claim 9, further comprising: facilitating an arithmetic logic unit of an application processor to host the extended register file.

25 12. The method of claim 9, further comprising:  
detecting the instruction; and

processing the one or more tasks, wherein processing of the one or more tasks includes managing one or more operations relating to contents of one or more of the registers, wherein the one or more operations include one or more of a comparison operation, a swapping operation, an arithmetic operation, and a decision-making operation.

5           13.     The method of claim 12, further comprising:

executing results associated with the one or more operations to complete the performance of the one or more tasks relating to the instruction; and

facilitating communication of at least one of the results, the contents, and other relevant data within or between one or more of the extension mechanism, the extended register file, the  
10   execution unit, and the arithmetic logic unit.

14.     The method of claim 10, further comprising: determining, by the execution unit, whether the instruction is qualified to be processed by the extension mechanism inside the extended register file,

wherein if the instruction is qualified, the one or more tasks relating to the instruction are  
15   performed by the extension mechanism inside the extended register file.

15.     The method of claim 14, wherein if the instruction is not qualified, the one or more tasks relating to the instruction are performed by a processing engine of the execution unit, wherein the execution unit-based processing engine includes an arithmetic/logic unit located outside the extended register file.

20           16.     The method of claim 11, further comprising: determining, by the arithmetic logic unit, whether the instruction is qualified to be processed by the extension mechanism inside the extended register file,

wherein if the instruction is qualified, the one or more tasks relating to the instruction are performed by the extension mechanism inside the extended register file, and

25           wherein if the instruction is not qualified, the one or more tasks relating to the instruction are performed by a processing engine of the arithmetic logic unit, wherein the arithmetic logic

unit-based processing engine includes an arithmetic/logic engine located outside the extended register file.

17. At least one machine-readable medium comprising a plurality of instructions, when executed on a computing device, to implement or perform a method as claimed in any of  
5 claims 9-16.

18. A system comprising a mechanism to implement or perform a method as claimed in any of claims 9-16.

19. An apparatus comprising means for performing a method as claimed in any of claims 9-16.

10 20. A computing device arranged to implement or perform a method as claimed in any of claims 9-16.

21. A communications device arranged to implement or perform a method as claimed in any of claims 9-16.

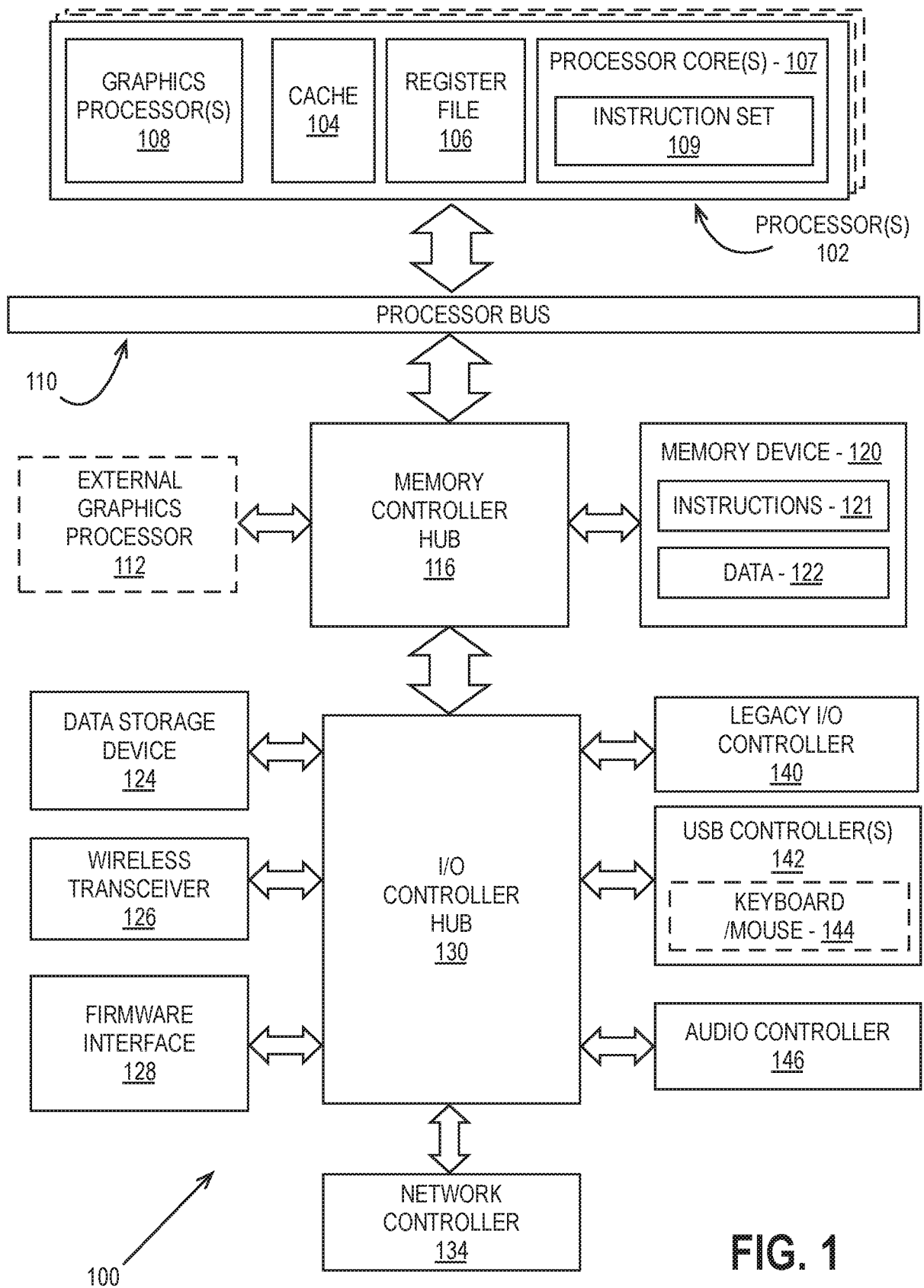


FIG. 1

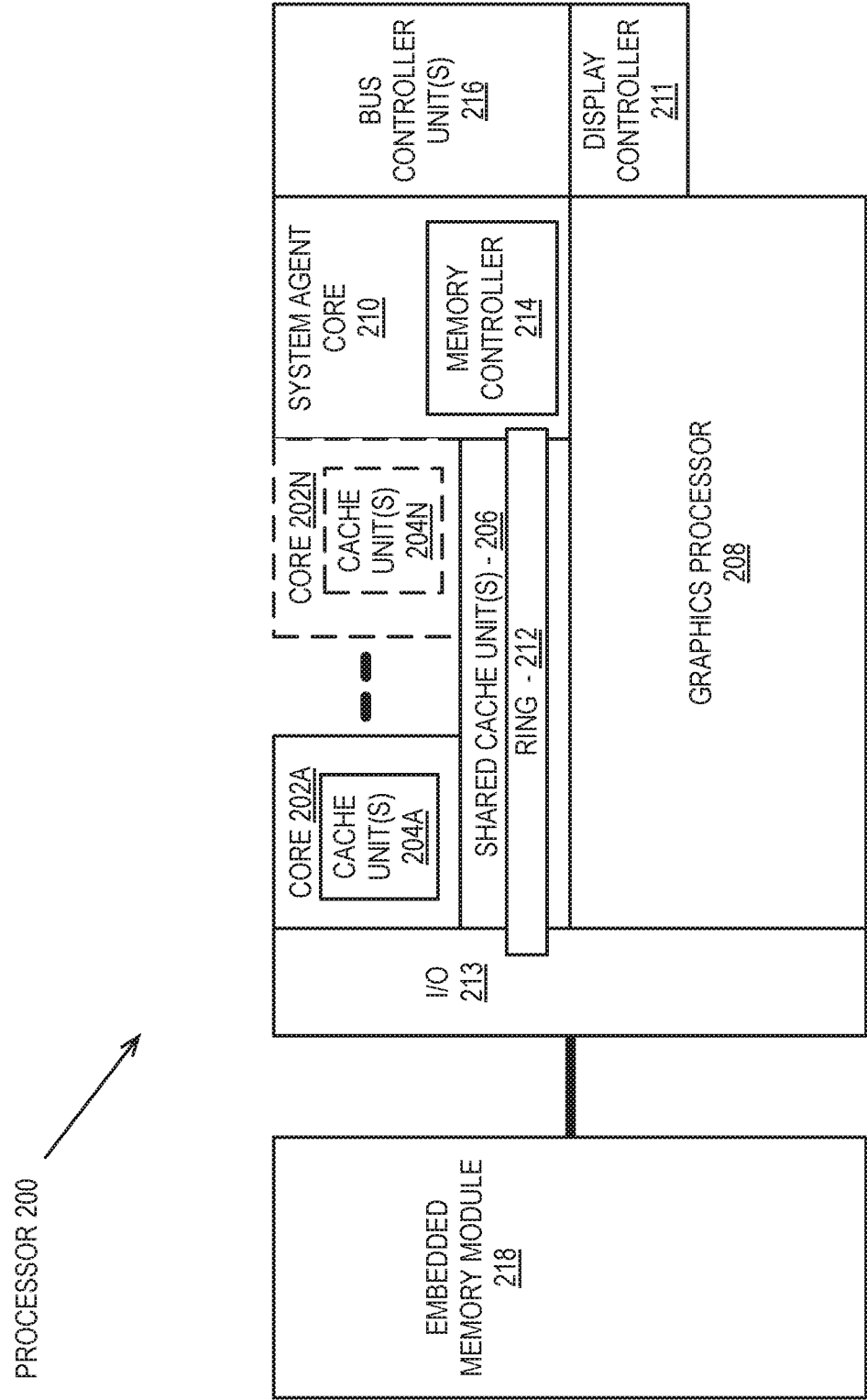


FIG. 2

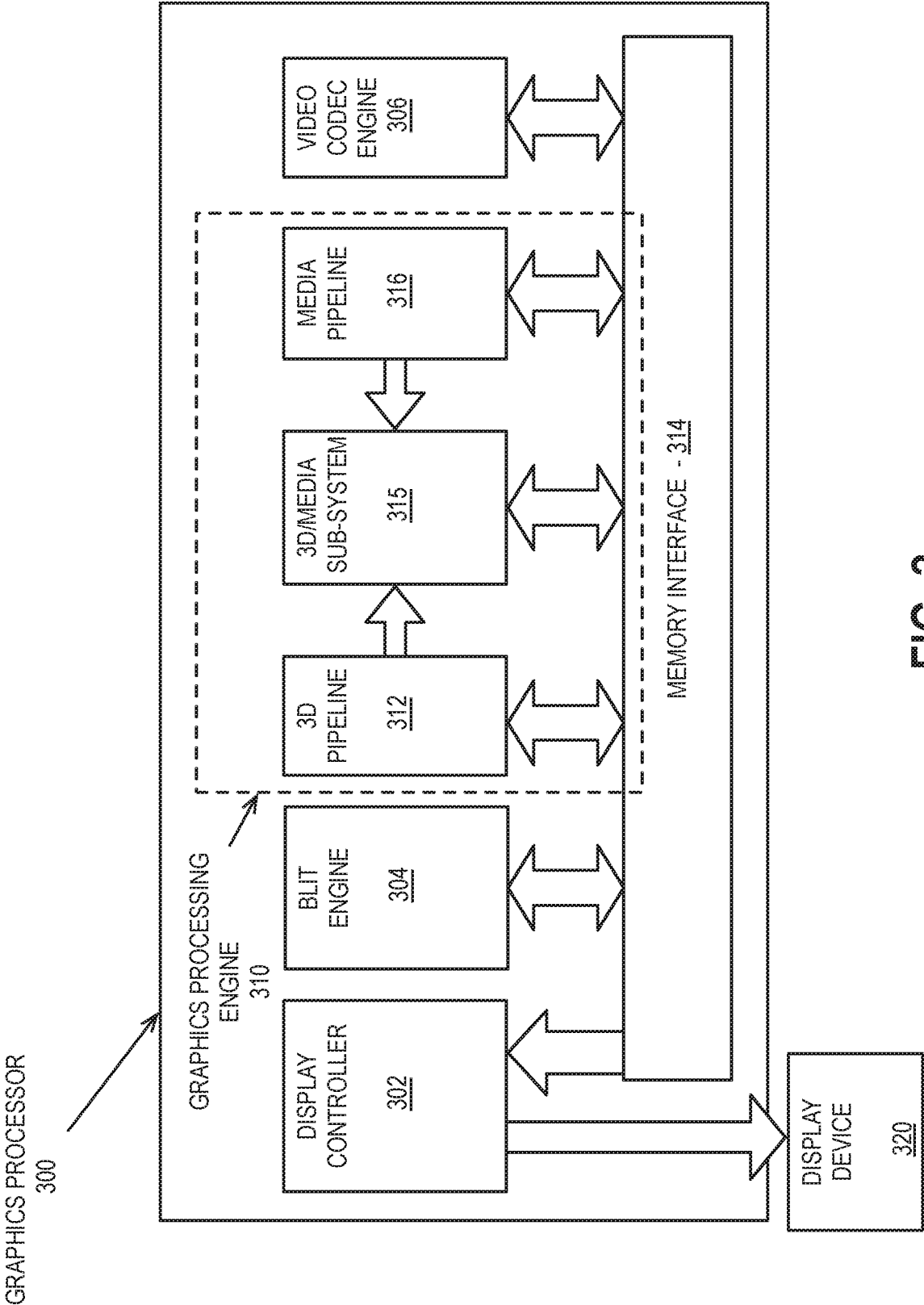


FIG. 3

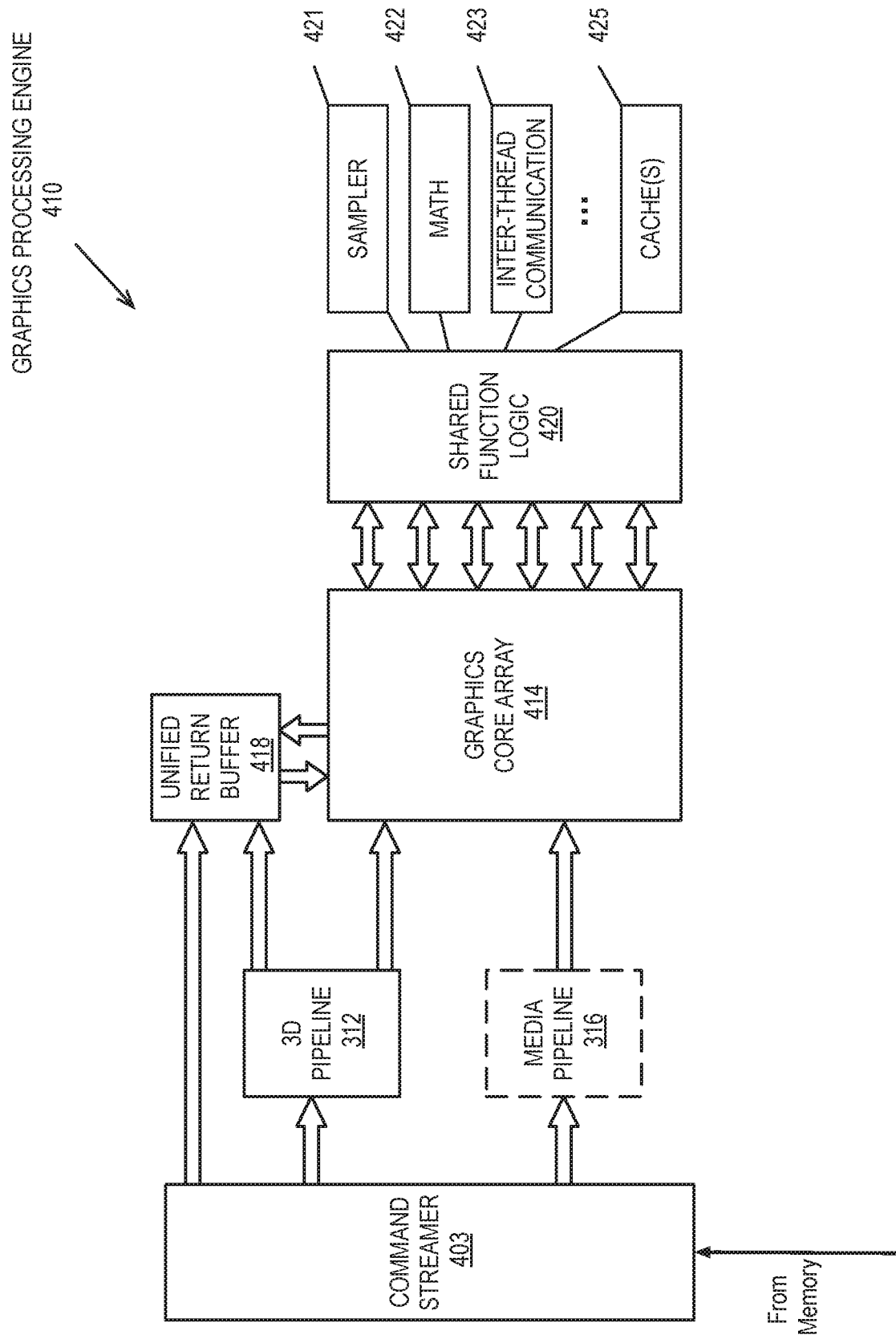


FIG. 4

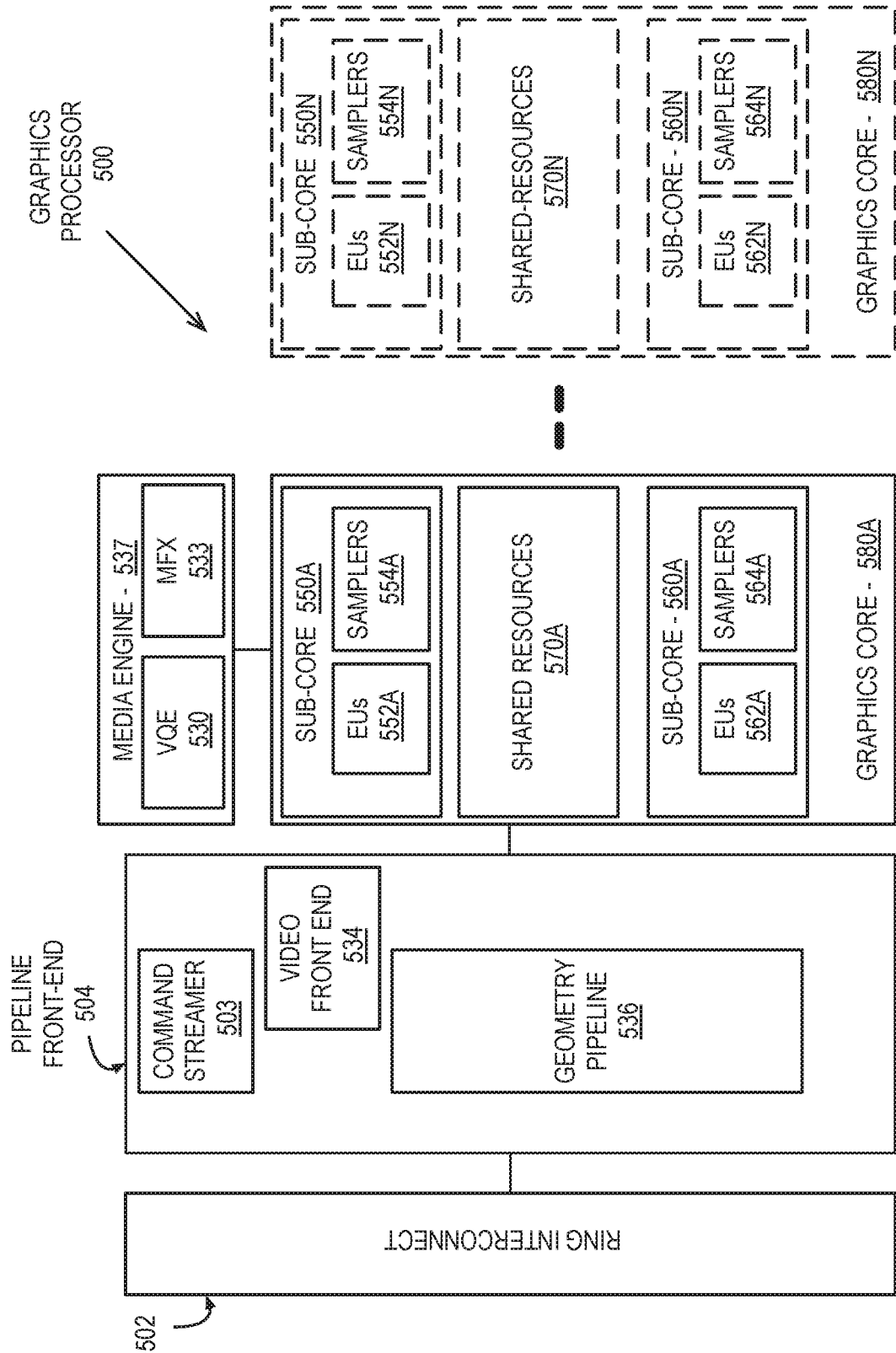


FIG. 5

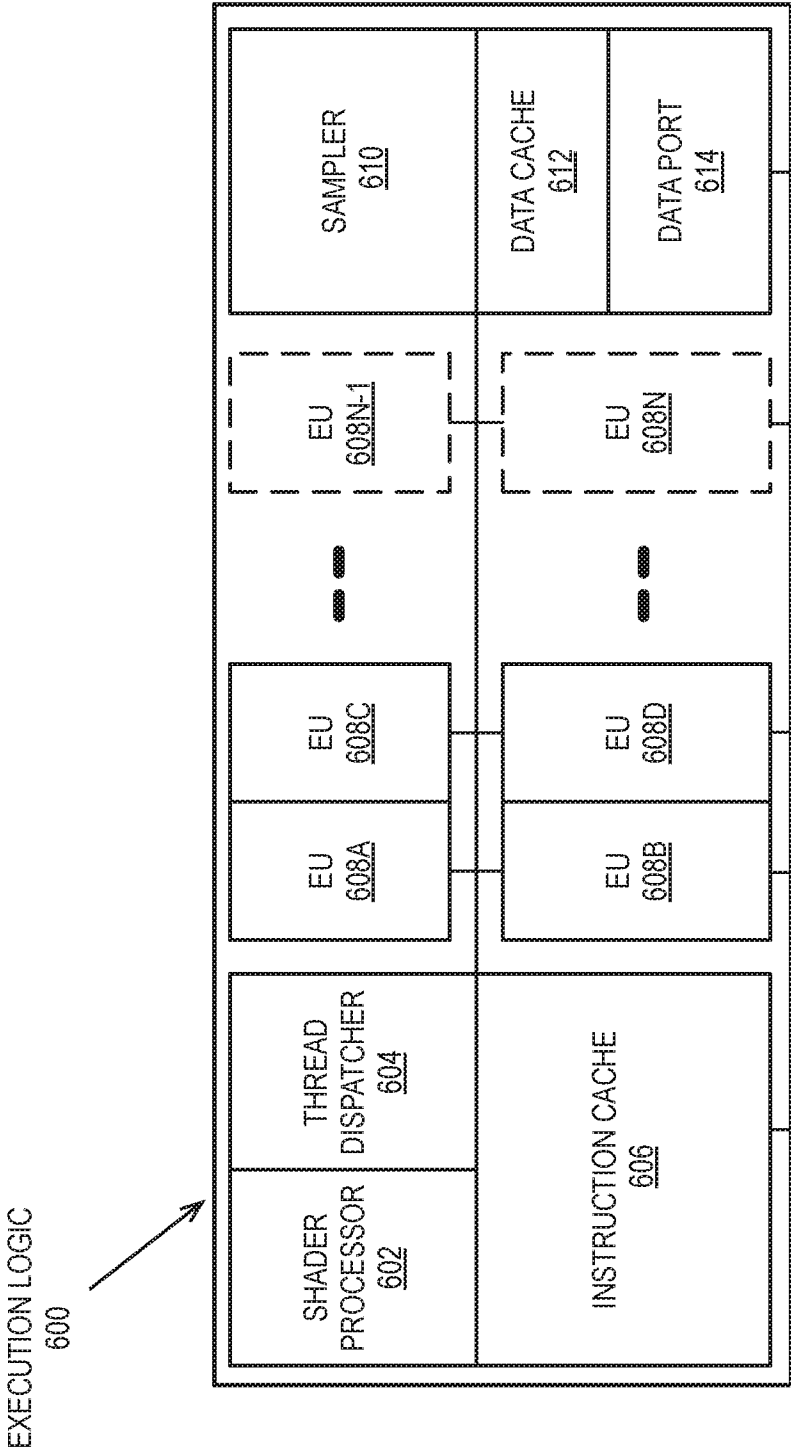


FIG. 6

GRAPHICS PROCESSOR INSTRUCTION FORMATS

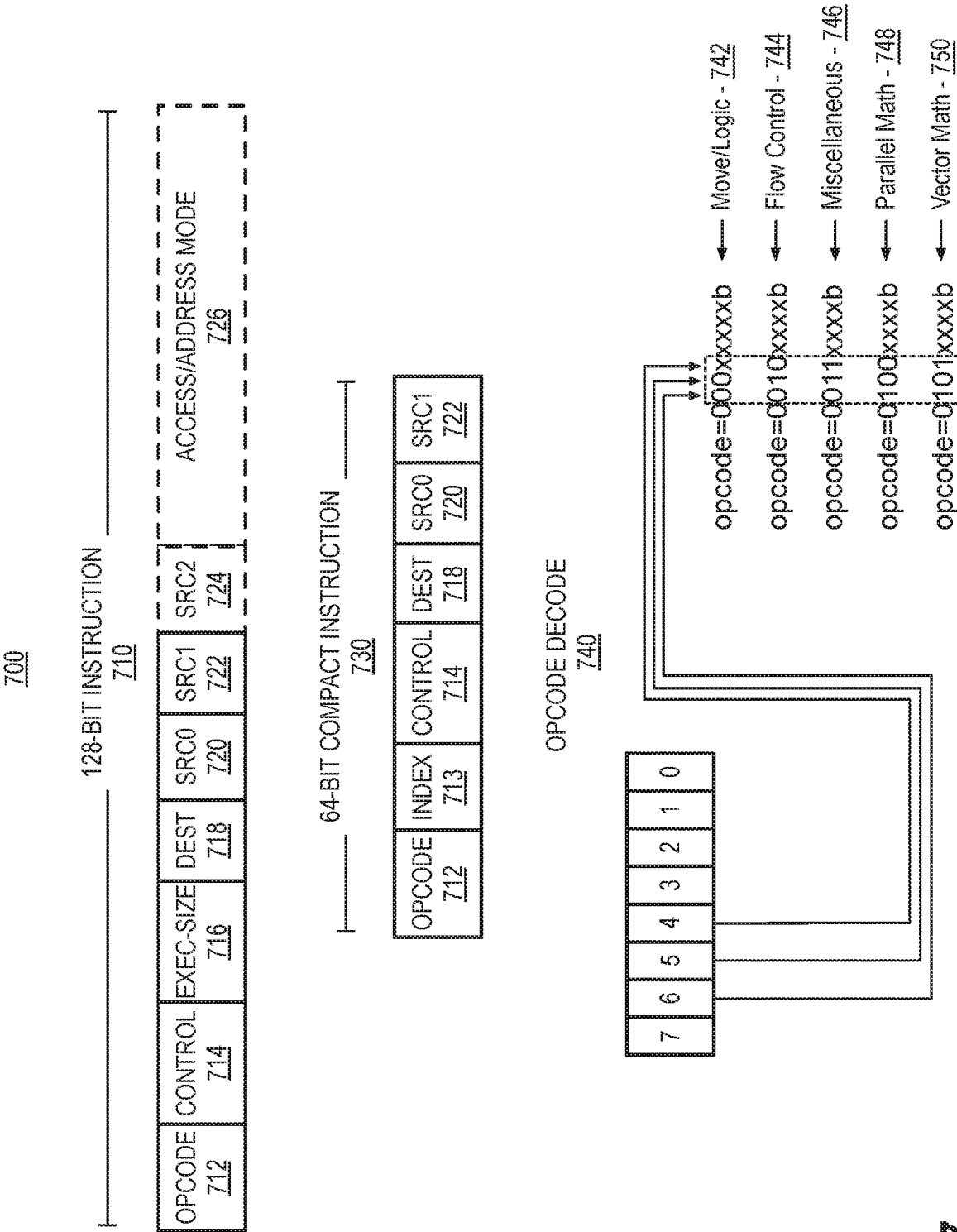


FIG. 7

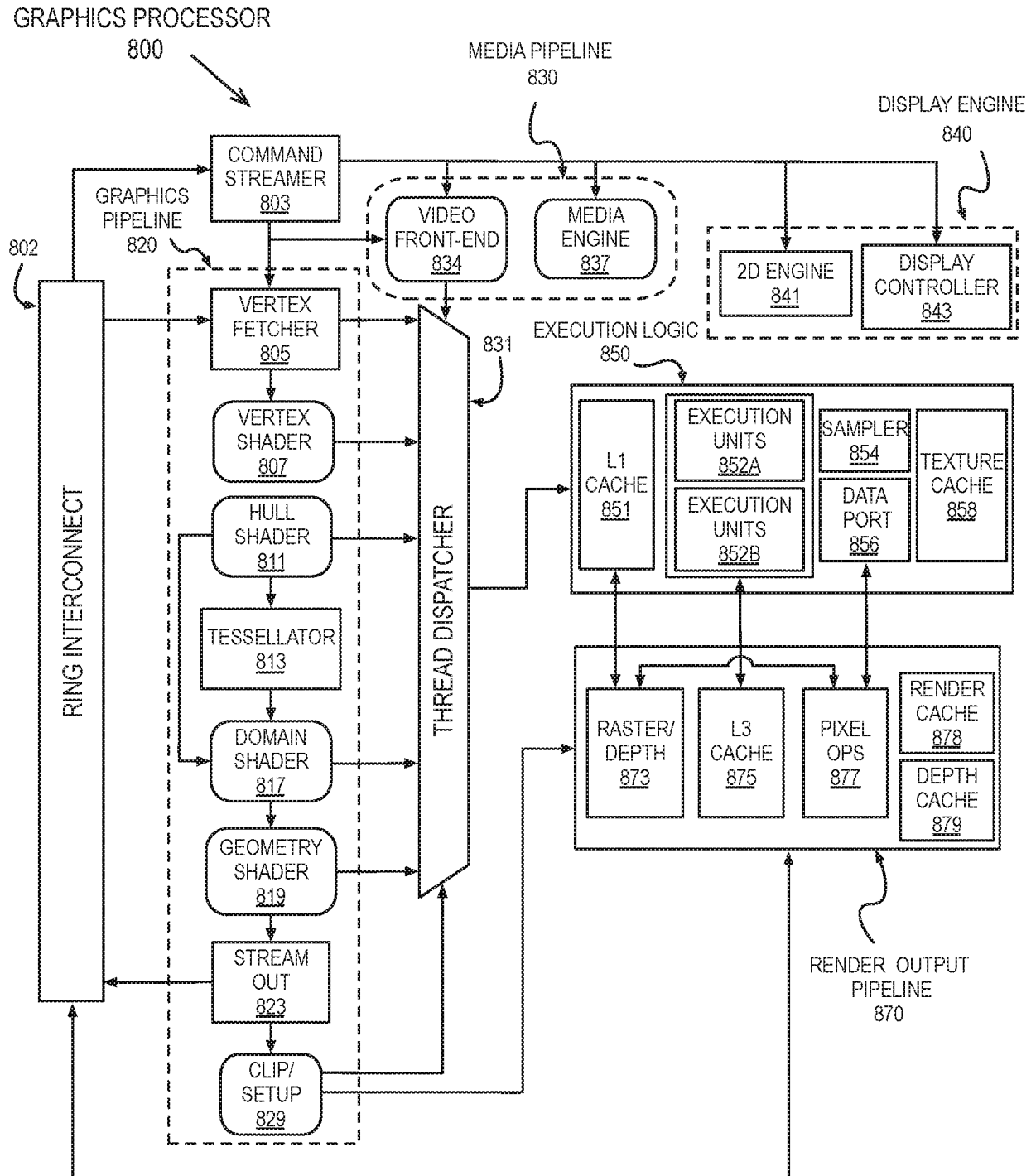
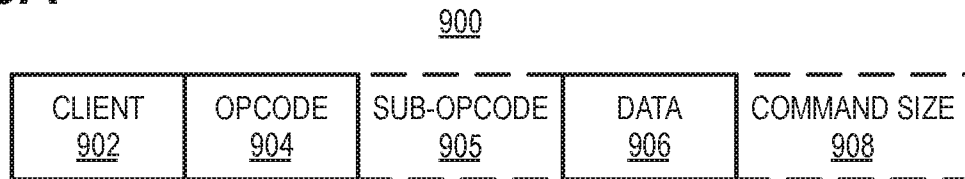
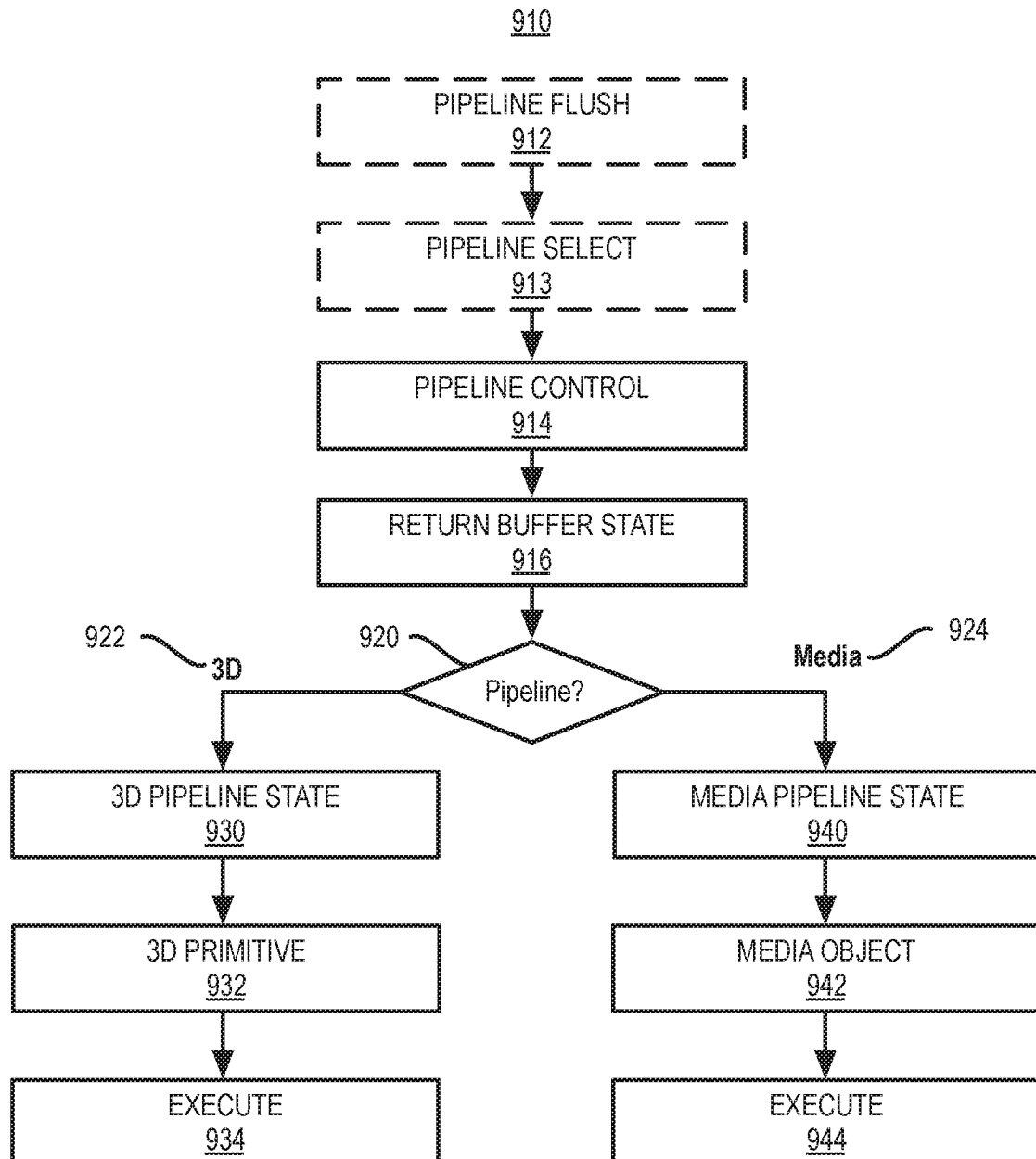


FIG. 8

**FIG. 9A****GRAPHICS PROCESSOR COMMAND FORMAT****FIG. 9B****GRAPHICS PROCESSOR COMMAND SEQUENCE**

## DATA PROCESSING SYSTEM -1000

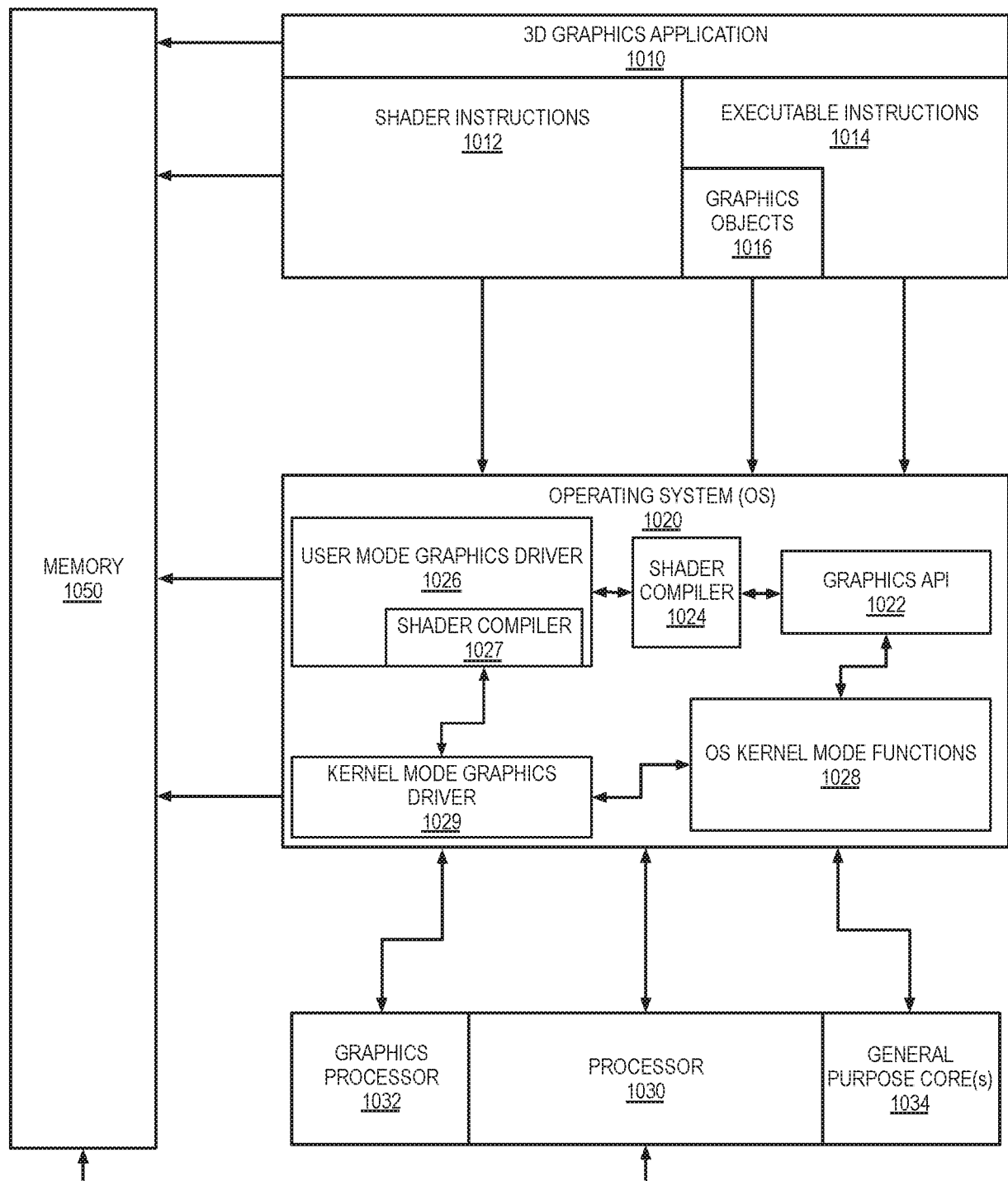


FIG. 10

IP CORE DEVELOPMENT - 1100

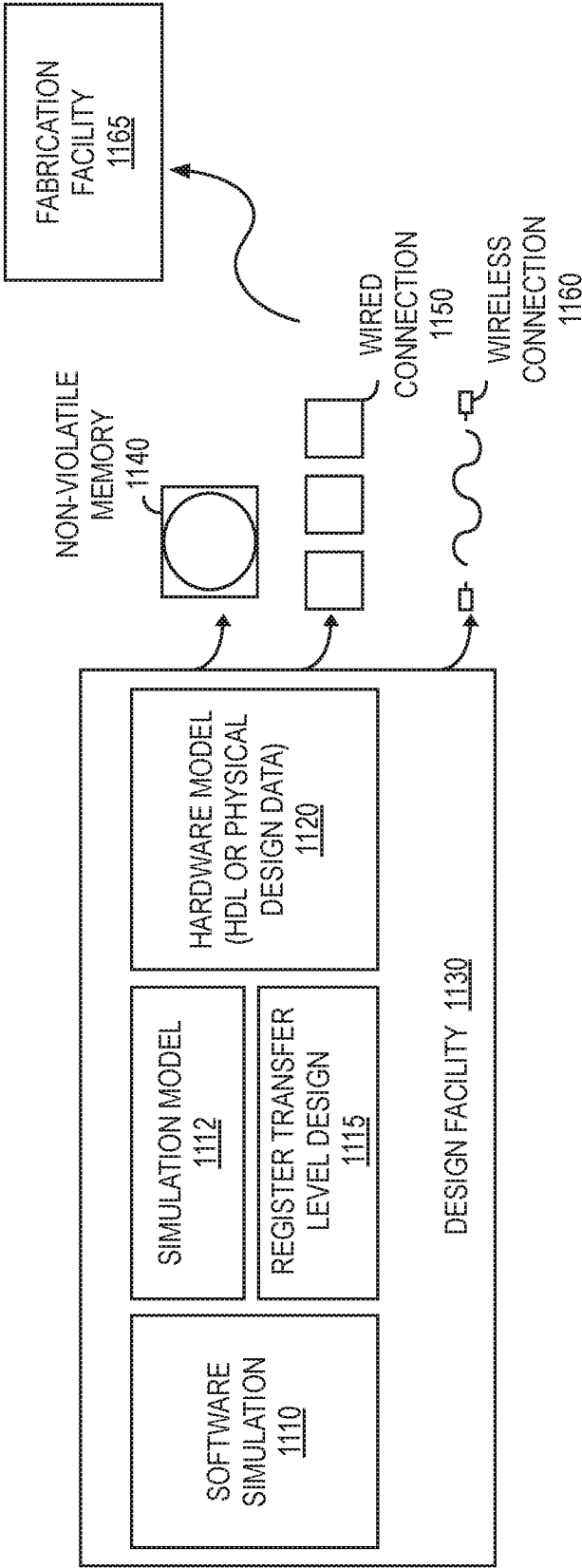


FIG. 11

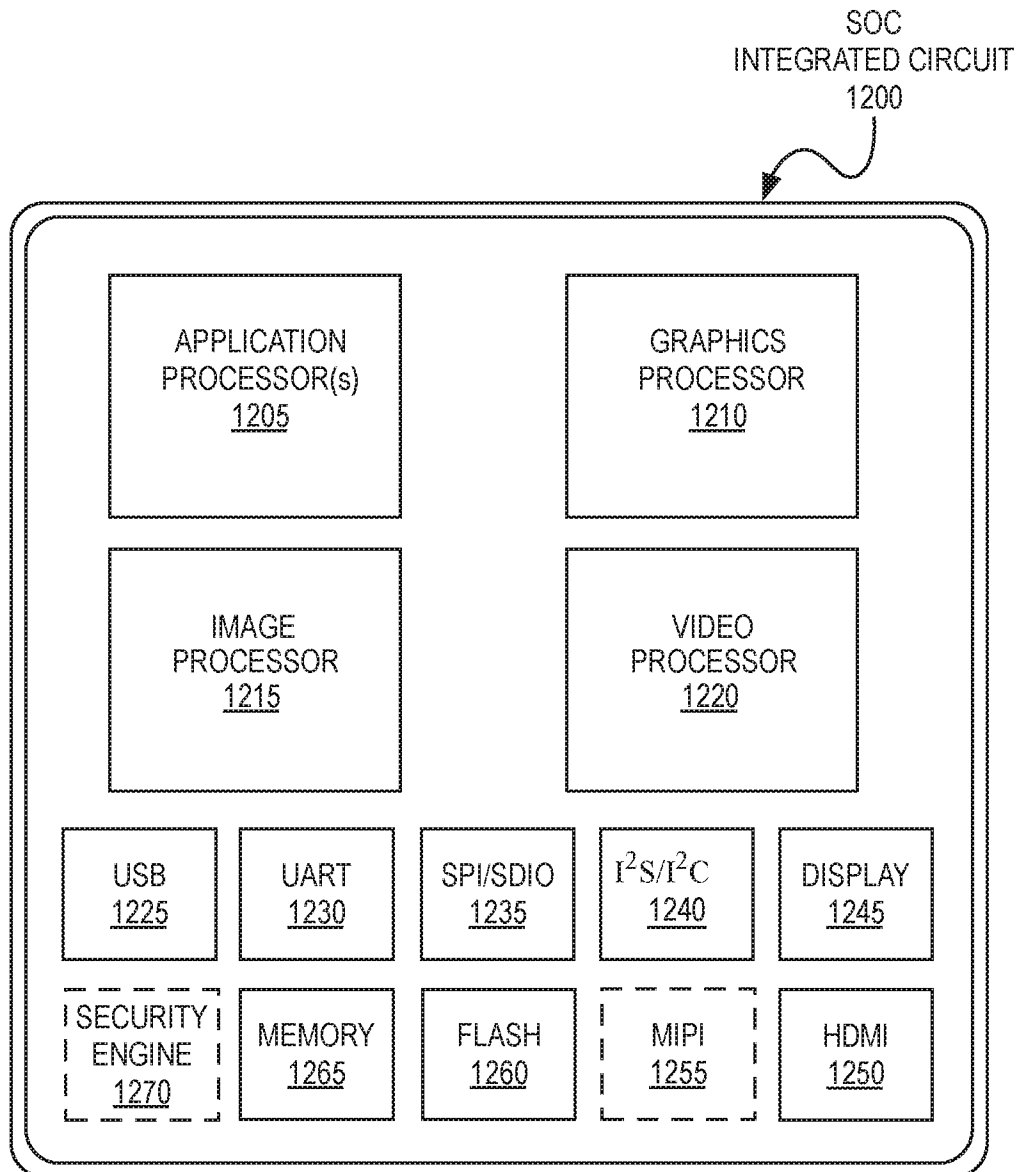


FIG. 12

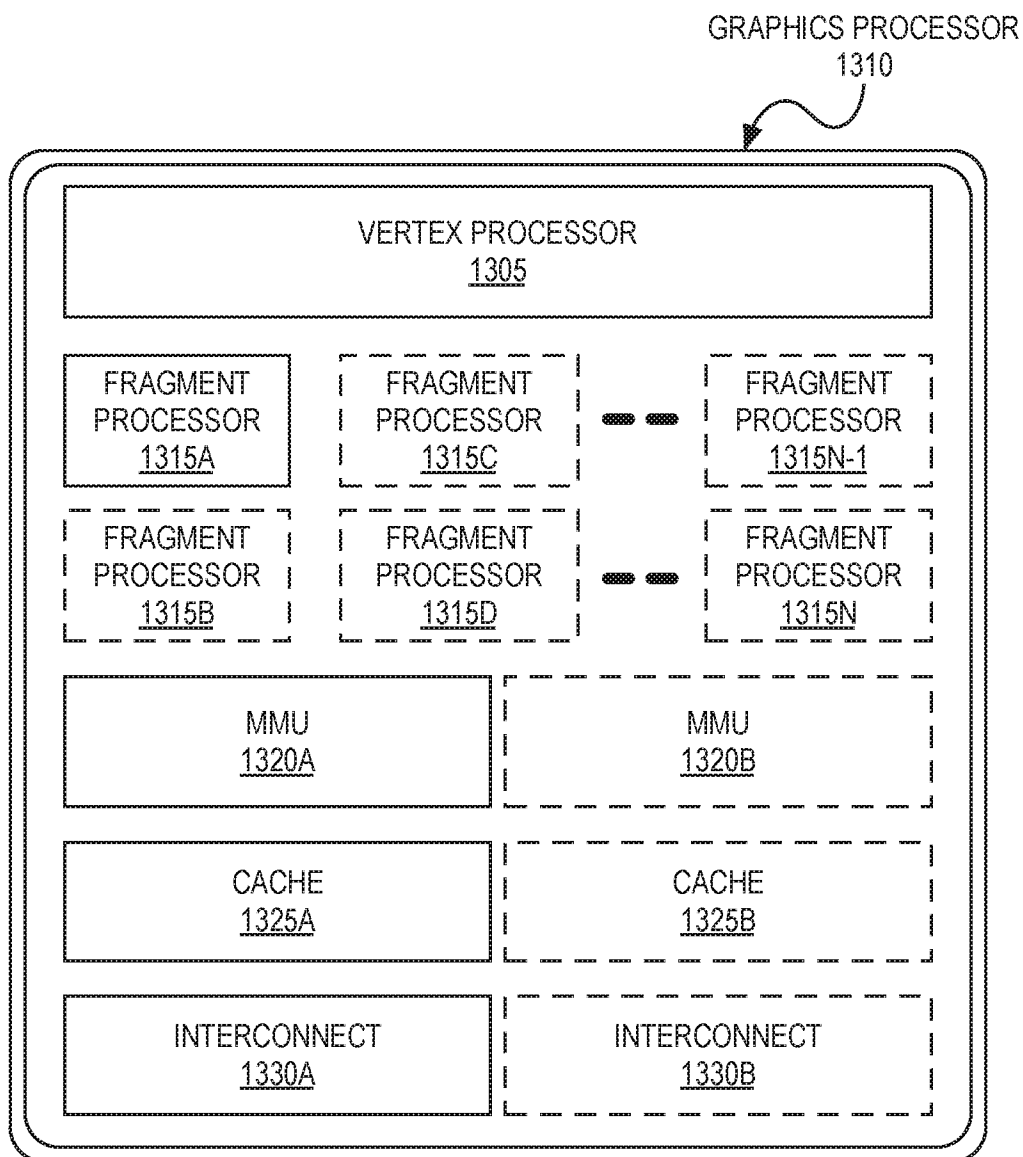


FIG. 13

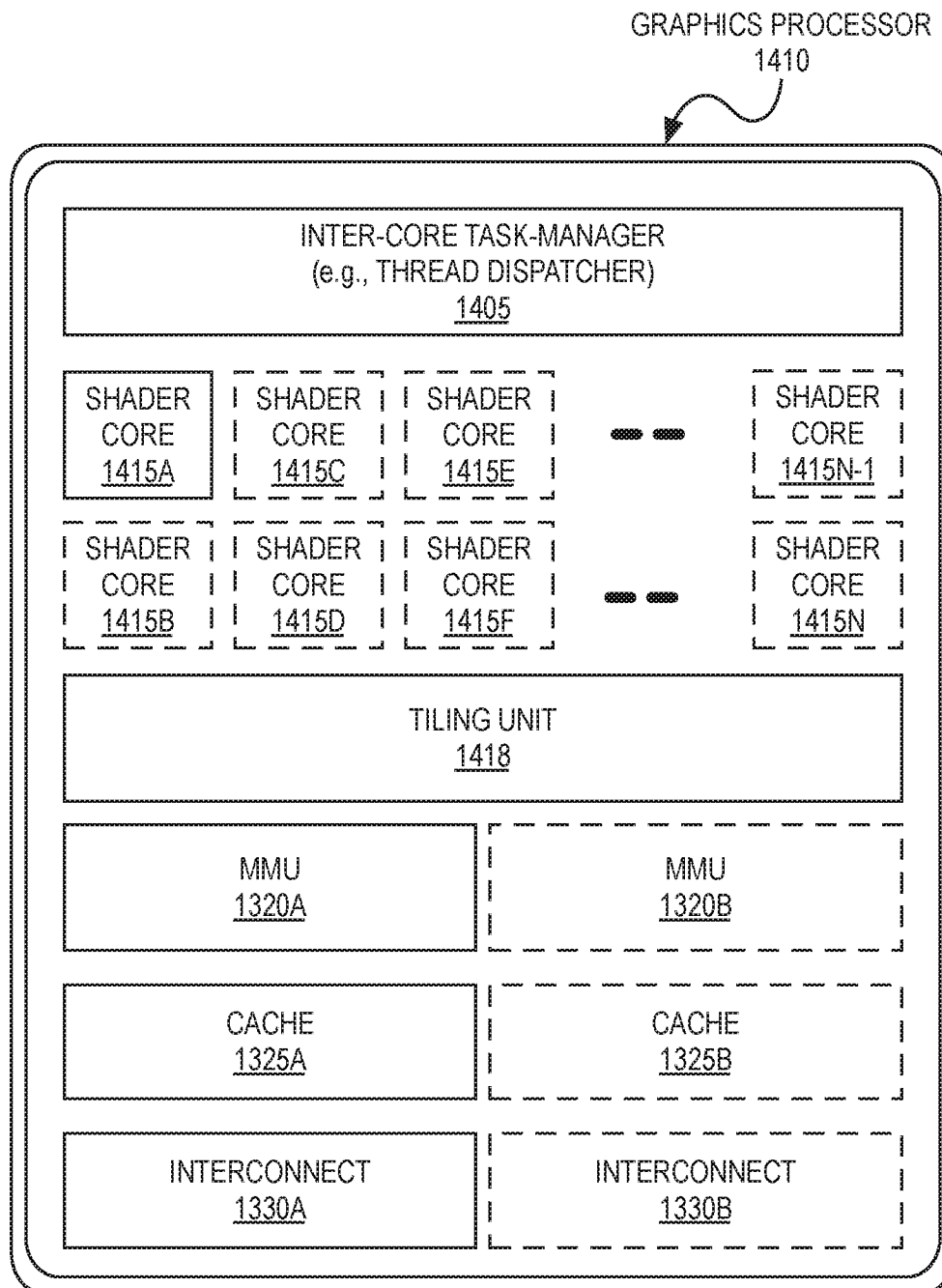


FIG. 14

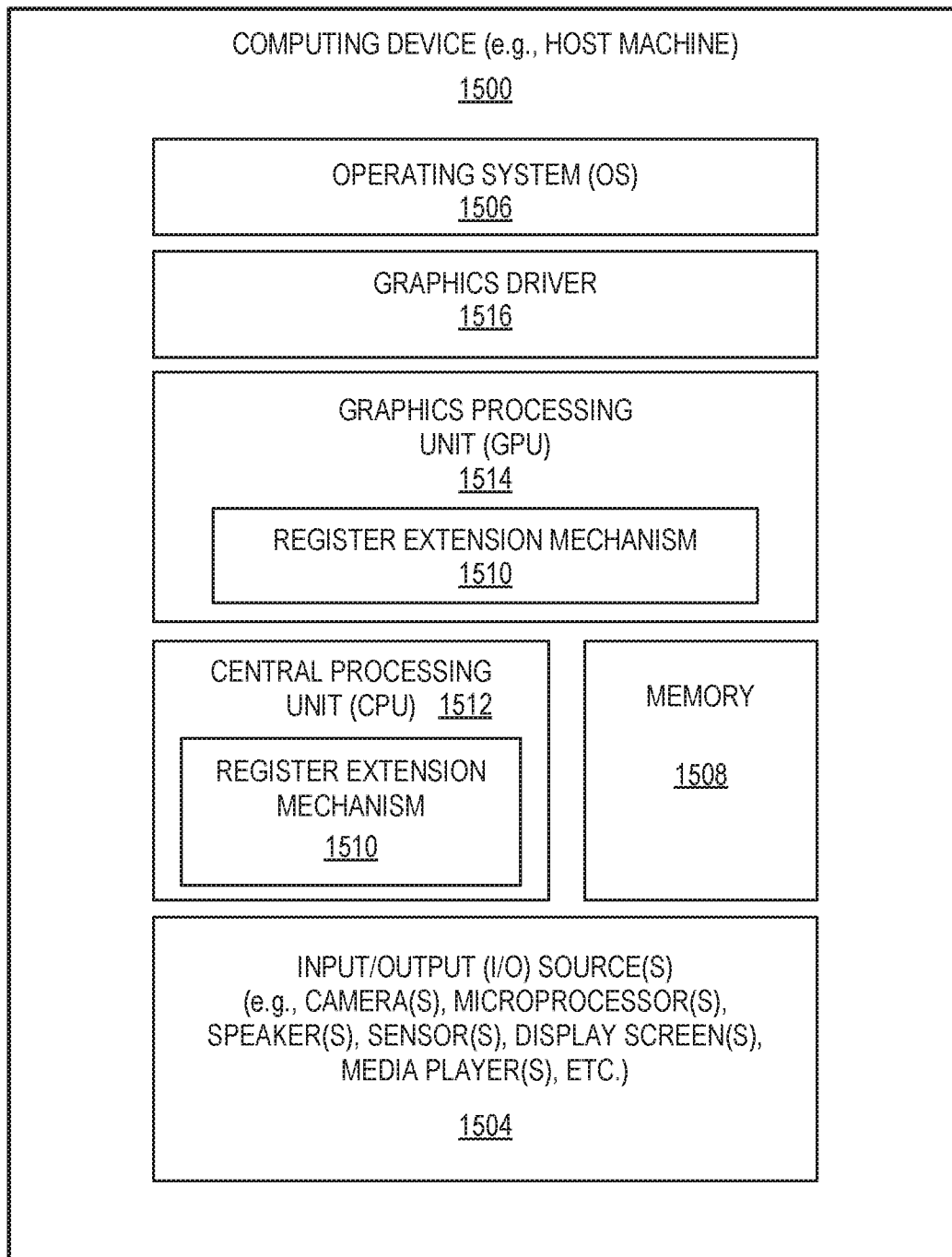
**FIG. 15**

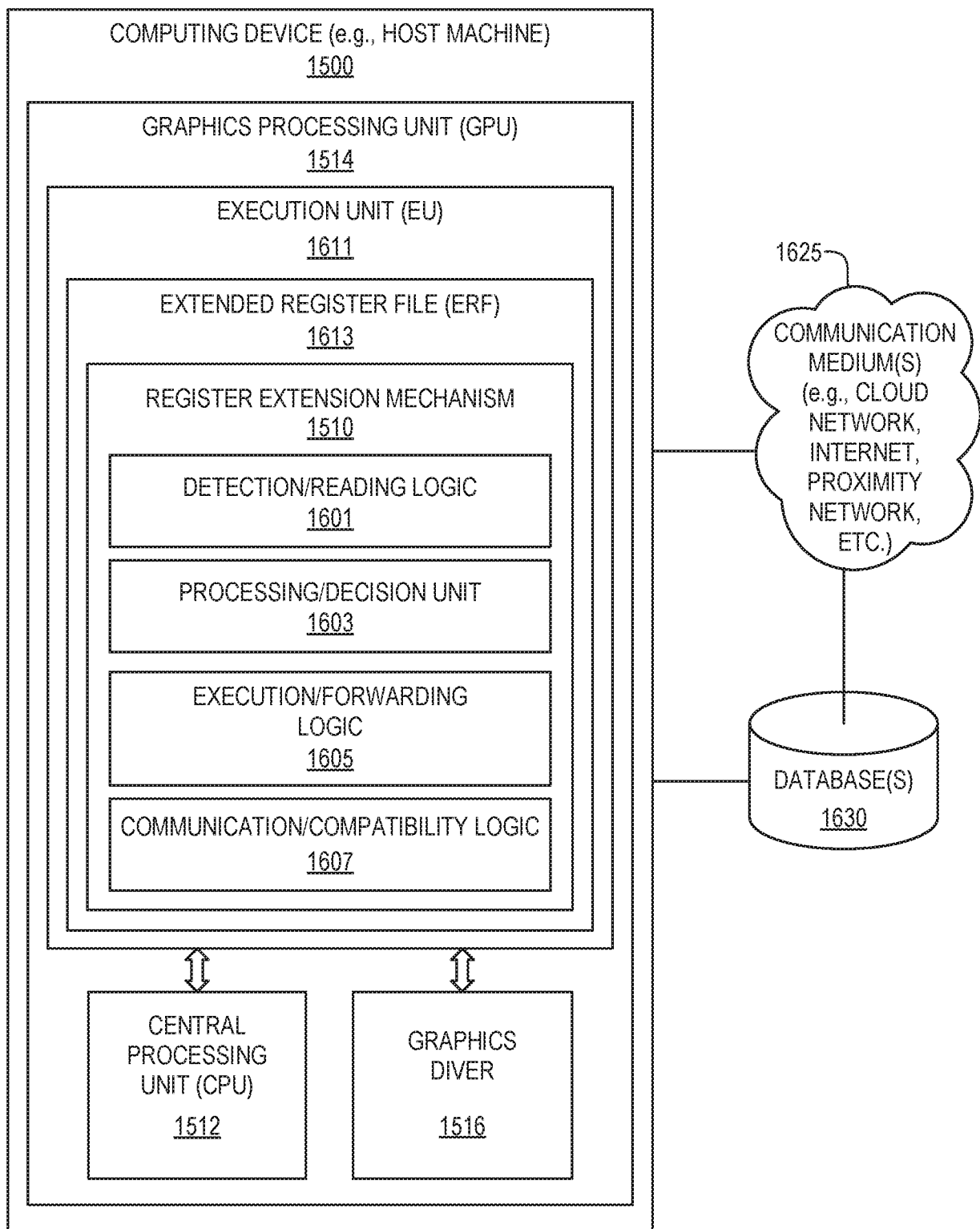
**FIG. 16**

**FIG. 17**  
**PRIOR ART**

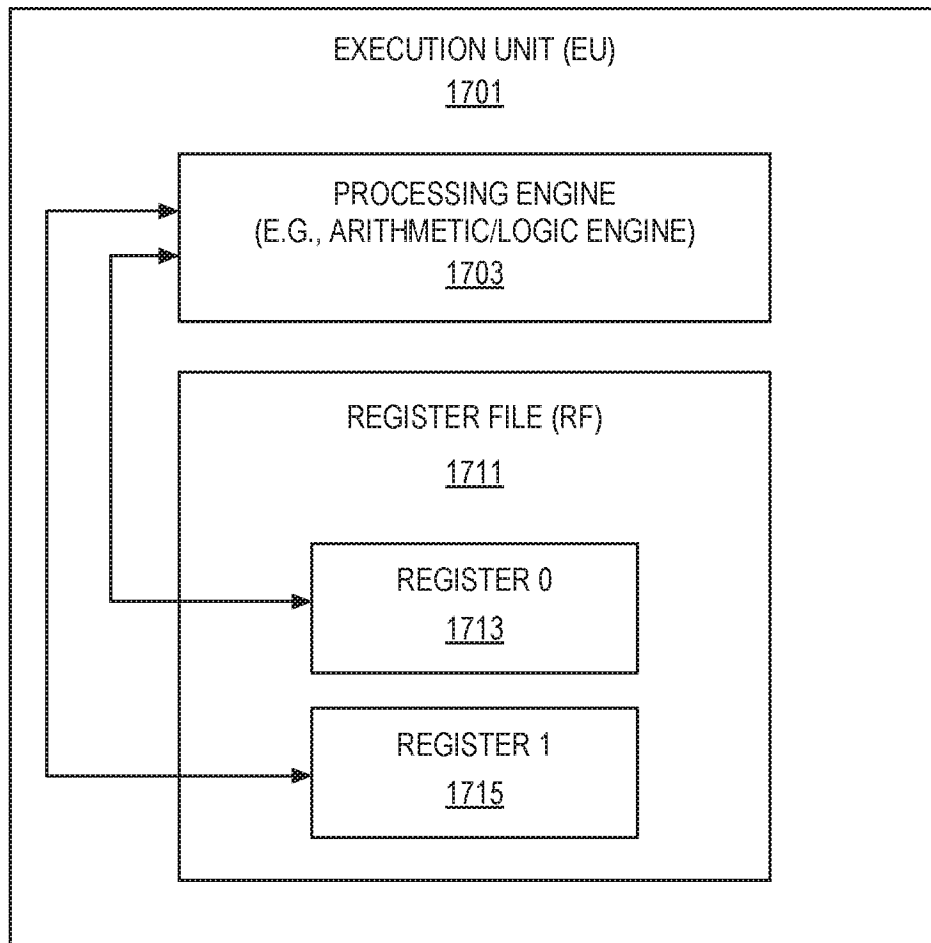


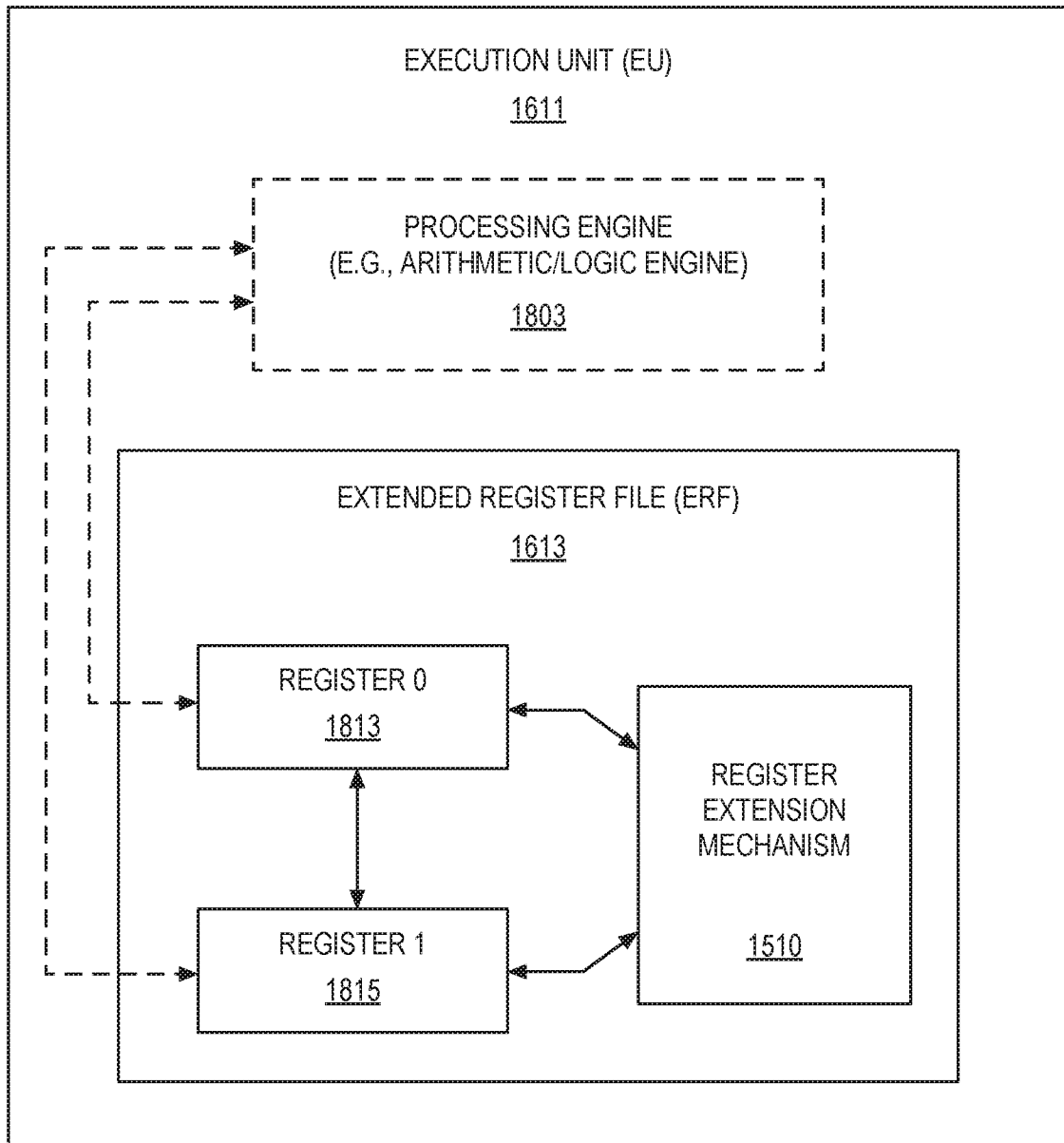
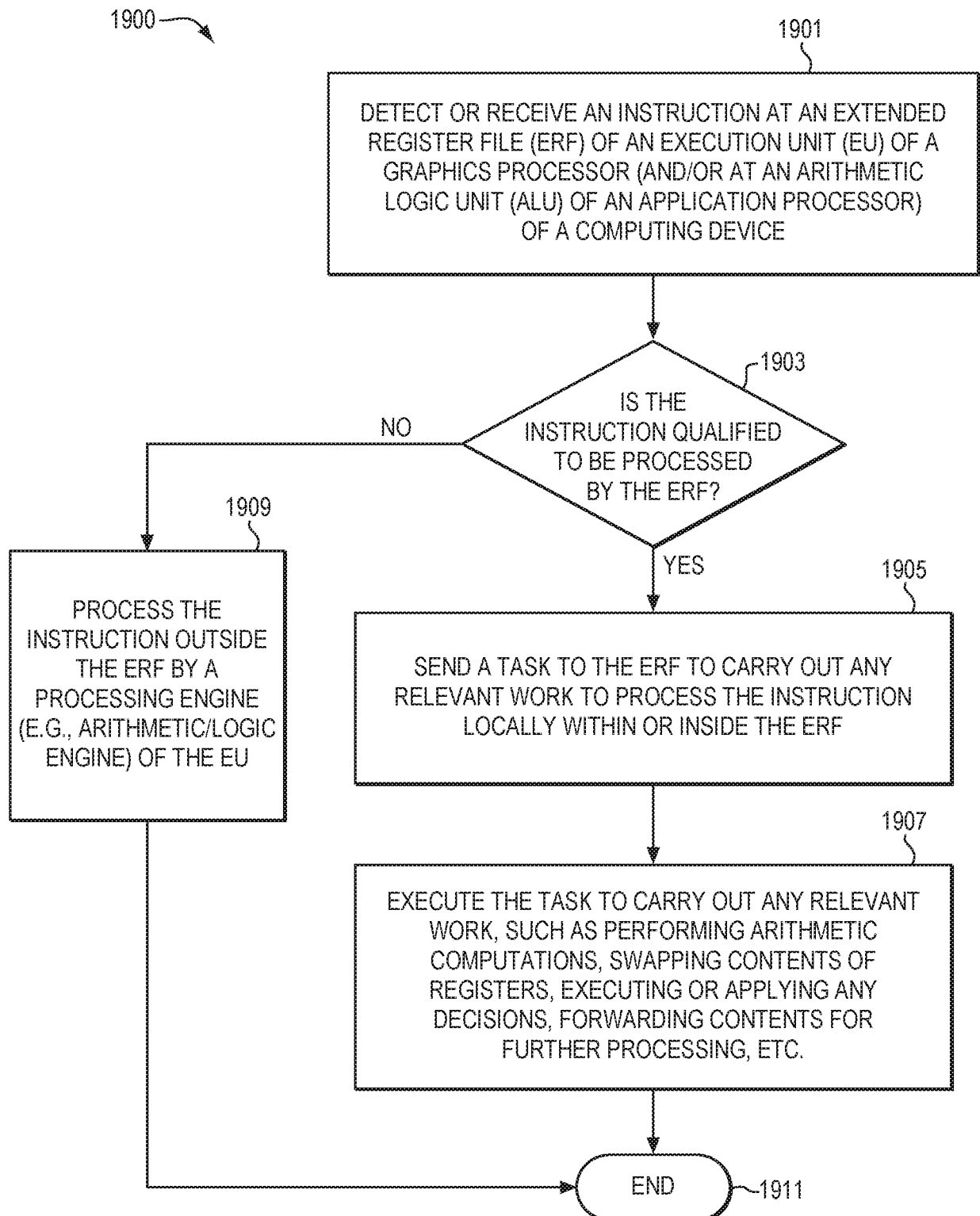
**FIG. 18**

FIG. 19



**A. CLASSIFICATION OF SUBJECT MATTER****G06F 9/30(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 9/30; G06F 15/00; G06F 7/00; G06F 12/04; G06F 9/312

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: extended register file, extension mechanism, task, instruction, execution unit, arithmetic logic unit

**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

| Category* | Citation of document, with indication, where appropriate, of the relevant passages                                                                      | Relevant to claim No. |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|
| X         | US 2015-0026435 A1 (INTERNATIONAL BUSINESS MACHINES CORPORATION)<br>22 January 2015<br>See paragraphs [0013]-[0014], [0052]-[0080]; and figures 3-5, 9. | 1-21                  |
| A         | US 2001-0011342 A1 (GERALD G. PECHANKE et al.) 02 August 2001<br>See paragraph [0020]; claims 1-4; and figure 1C.                                       | 1-21                  |
| A         | US 2013-0246761 A1 (JEFFREY H. DERBY et al.) 19 September 2013<br>See paragraphs [0038]-[0043]; claim 1; and figure 1.                                  | 1-21                  |
| A         | US 2003-0023646 A1 (DERRICK CHU LIN et al.) 30 January 2003<br>See paragraphs [0064]-[0090]; and figures 2A-3C.                                         | 1-21                  |
| A         | US 6877084 B1 (DAVID S. CHRISTIE) 05 April 2005<br>See column 8, line 27 - column 10, line 24; and figure 7.                                            | 1-21                  |



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

12 September 2017 (12.09.2017)

Date of mailing of the international search report

**12 September 2017 (12.09.2017)**

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

Lee, Eun Kyu

Telephone No. +82-42-481-3580



# INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

**PCT/US2017/030204**

| Patent document<br>cited in search report | Publication<br>date | Patent family<br>member(s)                                                                                                                                                                                                                                                                                                                                                                                                                                | Publication<br>date                                                                                                                                                                                                                                                                                                                          |
|-------------------------------------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| US 2015-0026435 A1                        | 22/01/2015          | CN 104331267 A<br>CN 104331267 B<br>US 9507599 B2                                                                                                                                                                                                                                                                                                                                                                                                         | 04/02/2015<br>12/04/2017<br>29/11/2016                                                                                                                                                                                                                                                                                                       |
| US 2001-0011342 A1                        | 02/08/2001          | US 6343356 B1<br>US 6430677 B2                                                                                                                                                                                                                                                                                                                                                                                                                            | 29/01/2002<br>06/08/2002                                                                                                                                                                                                                                                                                                                     |
| US 2013-0246761 A1                        | 19/09/2013          | None                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                                                                                                                                                                                                                                                              |
| US 2003-0023646 A1                        | 30/01/2003          | CA 2205830 A1<br>CA 2205830 C<br>JP 03924307 B2<br>JP 10-512069 A<br>JP 2006-172486 A<br>KR 10-0252411 B1<br>US 2001-0016861 A1<br>US 2004-0024800 A1<br>US 2004-0215681 A1<br>US 2005-0219897 A1<br>US 2006-0235914 A1<br>US 2007-0239810 A1<br>US 5666298 A<br>US 5818739 A<br>US 6275834 B1<br>US 6631389 B2<br>US 6738793 B2<br>US 6901420 B2<br>US 7117232 B2<br>US 7451169 B2<br>US 7461109 B2<br>US 7480686 B2<br>WO 96-17289 A1<br>WO 97-33222 A1 | 06/06/1996<br>15/08/2000<br>06/06/2007<br>17/11/1998<br>29/06/2006<br>15/04/2000<br>23/08/2001<br>05/02/2004<br>28/10/2004<br>06/10/2005<br>19/10/2006<br>11/10/2007<br>09/09/1997<br>06/10/1998<br>14/08/2001<br>07/10/2003<br>18/05/2004<br>31/05/2005<br>03/10/2006<br>11/11/2008<br>02/12/2008<br>20/01/2009<br>06/06/1996<br>12/09/1997 |
| US 6877084 B1                             | 05/04/2005          | AU 8316001 A<br>CN 1243304 C<br>CN 1446332 A<br>DE 60103414 T2<br>EP 1320800 A1<br>EP 1320800 B1<br>JP 2004-506263 A<br>KR 10-0880681 B1<br>KR 10-2003-0024850 A<br>WO 02-13005 A1                                                                                                                                                                                                                                                                        | 18/02/2002<br>22/02/2006<br>01/10/2003<br>02/06/2005<br>25/06/2003<br>19/05/2004<br>26/02/2004<br>30/01/2009<br>26/03/2003<br>14/02/2002                                                                                                                                                                                                     |