



## (51) International Patent Classification:

G06F 9/30 (2006.01) G06F 9/45 (2006.01)  
G06F 9/38 (2006.01)

## (21) International Application Number:

PCT/IB2016/000612

## (22) International Filing Date:

1 April 2016 (01.04.2016)

## (25) Filing Language:

English

## (26) Publication Language:

English

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

(72) Inventors: PODKORYTOV, Evgeniy; 34, Parkovaya str., apt. 60, Dolgoprudny, 141700 (RU). CHUDNOVETS, Andrey; 17 pos.Kievskiy apt. 63, Moscow, 143381 (RU). TITOV, Alexandr; 13 Yubileynaya str., bldg b, apartment 18, Severodvinsk, 164500 (RU). MASLENNIKOV, Dimitry; Aviatsionnaya 63-226, Moscow, 123182 (RU).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM,

AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, 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: APPARATUS AND METHOD FOR IMPROVING PERFORMANCE OF INTER-STRAND COMMUNICATIONS

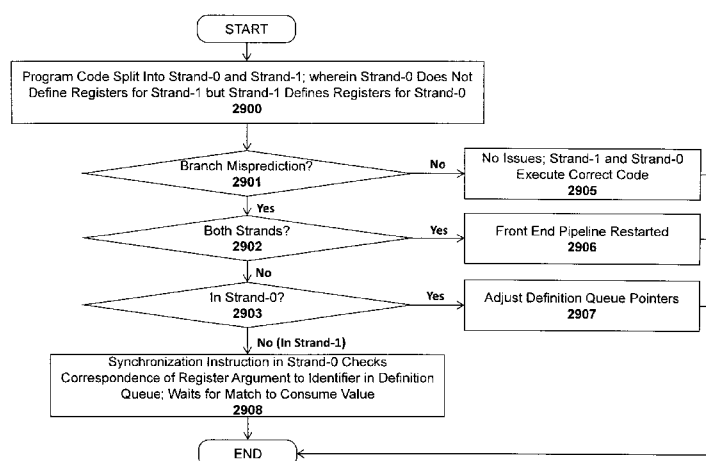


Fig. 29

(57) Abstract: An apparatus and method are described for inter-strand processing. For example, one embodiment of a processor comprises: a plurality of cores to concurrently execute a plurality of strands of program code; a first circuit to separate an instruction sequence into at least a first strand and a second strand; a second circuit comprising a plurality of entries to track associations between one or more instructions in the first strand and one or more instructions in the second strand, the second circuit to store a register identifier in a first entry responsive to execution of a first instruction in the first strand; a third circuit to compare an argument of a second instruction in a second strand with the register identifier in the first entry in response to detecting a branch misprediction in the second strand, wherein if the argument and the register identifier do not match, then the third circuit to cause the second strand to wait until the argument and the register identifier match before consuming a register value.

## APPARATUS AND METHOD FOR IMPROVING PERFORMANCE OF INTER-STRAND COMMUNICATIONS

### BACKGROUND

#### Field of the Invention

[0001] This invention relates generally to the field of computer processors. More particularly, the invention relates to an apparatus and method for improving performance of inter-strand communications.

#### Description of the Related Art

[0002] In order to take advantage of multiprocessor systems, program code to be executed may be separated into multiple threads for execution by various processing entities. Once separated, the threads may be executed in parallel on the processing resources of the multiprocessor system. In order to increase the utility of the processing resources, out-of-order execution may be employed, executing instructions as input to such instructions becomes available. Thus, an instruction that appears later in a code sequence may be executed before an instruction appearing earlier in a code sequence.

### BRIEF DESCRIPTION OF THE DRAWINGS

[0003] A better understanding of the present invention can be obtained from the following detailed description in conjunction with the following drawings, in which: FIG. 1A is a block diagram of an exemplary computer system formed with a processor that may include execution units to execute an instruction, in accordance with embodiments of the present disclosure;

[0004] FIG. 1B illustrates a data processing system, in accordance with embodiments of the present disclosure;

[0005] FIG. 1C illustrates other embodiments of a data processing system for performing text string comparison operations;

[0006] FIG. 2 is a block diagram of the micro-architecture for a processor that may include logic circuits to perform instructions, in accordance with embodiments of the present disclosure;

[0007] FIG. 3A illustrates various packed data type representations in multimedia registers, in accordance with embodiments of the present disclosure;

[0008] FIG. 3B illustrates possible in-register data storage formats, in accordance with embodiments of the present disclosure;

[0009] FIG. 3C illustrates various signed and unsigned packed data type representations in multimedia registers, in accordance with embodiments of the present disclosure;

[0010] FIG. 3D illustrates an embodiment of an operation encoding format;

- [0011]** FIG. 3E illustrates another possible operation encoding format having forty or more bits, in accordance with embodiments of the present disclosure;
- [0012]** FIG. 3F illustrates yet another possible operation encoding format, in accordance with embodiments of the present disclosure;
- [0013]** FIG. 4A is a block diagram illustrating an in-order pipeline and a register renaming stage, out-of-order issue/execution pipeline, in accordance with embodiments of the present disclosure;
- [0014]** FIG. 4B is a block diagram illustrating an in-order architecture core and a register renaming logic, out-of-order issue/execution logic to be included in a processor, in accordance with embodiments of the present disclosure;
- [0015]** FIG. 5A is a block diagram of a processor, in accordance with embodiments of the present disclosure;
- [0016]** FIG. 5B is a block diagram of an example implementation of a core, in accordance with embodiments of the present disclosure;
- [0017]** FIG. 6 is a block diagram of a system, in accordance with embodiments of the present disclosure;
- [0018]** FIG. 7 is a block diagram of a second system, in accordance with embodiments of the present disclosure;
- [0019]** FIG. 8 is a block diagram of a third system in accordance with embodiments of the present disclosure;
- [0020]** FIG. 9 is a block diagram of a system-on-a-chip, in accordance with embodiments of the present disclosure;
- [0021]** FIG. 10 illustrates a processor containing a central processing unit and a graphics processing unit which may perform at least one instruction, in accordance with embodiments of the present disclosure;
- [0022]** FIG. 11 is a block diagram illustrating the development of IP cores, in accordance with embodiments of the present disclosure;
- [0023]** FIG. 12 illustrates how an instruction of a first type may be emulated by a processor of a different type, in accordance with embodiments of the present disclosure;
- [0024]** FIG. 13 illustrates a block diagram contrasting the use of a software instruction converter to convert binary instructions in a source instruction set to binary instructions in a target instruction set, in accordance with embodiments of the present disclosure;
- [0025]** FIG. 14 is a block diagram of an instruction set architecture of a processor, in accordance with embodiments of the present disclosure;
- [0026]** FIG. 15 is a more detailed block diagram of an instruction set architecture of a processor, in accordance with embodiments of the present disclosure;
- [0027]** FIG. 16 is a block diagram of an execution pipeline for an instruction set architecture of a processor, in accordance with embodiments of the present disclosure;
- [0028]** FIG. 17 is a block diagram of an electronic device for utilizing a processor, in accordance with embodiments of the present disclosure;
- [0029]** FIG. 18 illustrates a block diagram of a system for implementing inter-strand communication

via a shared register file, in accordance with embodiments of the present disclosure;

**[0030]** FIG. 19 illustrates example operation of a system for implementing an inter-strand communication via a shared register, in accordance with embodiments of the present disclosure;

**[0031]** FIG. 20 illustrates a more detailed block diagram of a system for implementing inter-strand communication via a shared register file, in accordance with embodiments of the present disclosure;

**[0032]** FIGS. 21A-21E illustrate further example operation of a system for implementing inter-strand communication via a shared register, in accordance with embodiments of the present disclosure;

**[0033]** FIG. 22 is an illustration of an example method for inter-strand communication via a shared register, in accordance with embodiments of the present disclosure;

**[0034]** FIG. 23 illustrates one embodiment in which the performance of inter-strand communication is improved;

**[0035]** FIGS. 24-25 illustrates the operation of exemplary program code strands without inter-strand recovery;

**[0036]** FIG. 26-27 illustrates the operation of exemplary program code strands with inter-strand recovery;

**[0037]** FIG. 28 illustrates scenarios in which jump execution clears (JEClears) occur in different strands of related program code; and

**[0038]** FIG. 29 illustrates a method in accordance with one embodiment of the invention.

## DETAILED DESCRIPTION

**[0039]** In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the embodiments of the invention described below. It will be apparent, however, to one skilled in the art that the embodiments of the invention may be practiced without some of these specific details. In other instances, well-known structures and devices are shown in block diagram form to avoid obscuring the underlying principles of the embodiments of the invention.

**[0040]** The following description describes an apparatus and method for inter-strand communication via a shared register file in a processor, virtual processor, package, computer system, or other processing apparatus. Such a processing apparatus may include an out-of-order processor. The apparatus and method may make use of processing logic and one or more instructions. In the following description, numerous specific details such as processing logic, processor types, micro-architectural conditions, events, enablement mechanisms, and the like are set forth in order to provide a more thorough understanding of embodiments of the present disclosure. It will be appreciated, however, by one skilled in the art that the embodiments may be practiced without such specific details. Additionally, some well-known structures, circuits, and the like have not been shown in detail to avoid unnecessarily obscuring embodiments of the present disclosure.

### EXEMPLARY ARCHITECTURES

**[0041]** Although the following embodiments are described with reference to a processor, other embodiments are applicable to other types of integrated circuits and logic devices. Similar techniques and teachings of embodiments of the present disclosure may be applied to other types of circuits or semiconductor devices that may benefit from higher pipeline throughput and improved performance. The teachings of embodiments of the present disclosure are applicable to any processor or machine that performs data manipulations. However, the embodiments are not limited to processors or machines that perform 512-bit, 256-bit, 128-bit, 64-bit, 32-bit, or 16-bit data operations and may be applied to any processor and machine in which manipulation or management of data may be performed. In addition, the following description provides examples, and the accompanying drawings show various examples for the purposes of illustration. However, these examples should not be construed in a limiting sense as they are merely intended to provide examples of embodiments of the present disclosure rather than to provide an exhaustive list of all possible implementations of embodiments of the present disclosure.

**[0042]** Although the below examples describe instruction handling and distribution in the context of execution units and logic circuits, other embodiments of the present disclosure may be accomplished by way of a data or instructions stored on a machine-readable, tangible medium, which when performed by a machine cause the machine to perform functions consistent with at least one embodiment of the disclosure. In one embodiment, functions associated with embodiments of the present disclosure are embodied in machine-executable instructions. The instructions may be used to cause a general-purpose or special-purpose processor that may be programmed with the instructions to perform the steps of the present disclosure. Embodiments of the present disclosure may be provided as a computer program product or software which may include a machine or computer-readable medium having stored thereon

instructions which may be used to program a computer (or other electronic devices) to perform one or more operations according to embodiments of the present disclosure. Furthermore, steps of embodiments of the present disclosure might be performed by specific hardware components that contain fixed-function logic for performing the steps, or by any combination of programmed computer components and fixed-function hardware components.

**[0043]** Instructions used to program logic to perform embodiments of the present disclosure may be stored within a memory in the system, such as DRAM, cache, flash memory, or other storage. Furthermore, the instructions may be distributed via a network or by way of other computer-readable media. Thus a machine-readable medium may include any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computer), but is not limited to, floppy diskettes, optical disks, Compact Discs, Read-Only Memory (CD-ROMs), and magneto-optical disks, Read-Only Memory (ROMs), Random Access Memory (RAM), Erasable Programmable Read-Only Memory (EPROM), Electrically Erasable Programmable Read-Only Memory (EEPROM), magnetic or optical cards, flash memory, or a tangible, machine-readable storage used in the transmission of information over the Internet via electrical, optical, acoustical or other forms of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.). Accordingly, the computer-readable medium may include any type of tangible machine-readable medium suitable for storing or transmitting electronic instructions or information in a form readable by a machine (e.g., a computer).

**[0044]** A design may go through various stages, from creation to simulation to fabrication. Data representing a design may represent the design in a number of manners. First, as may be useful in simulations, the hardware may be represented using a hardware description language or another functional description language. Additionally, a circuit level model with logic and/or transistor gates may be produced at some stages of the design process. Furthermore, designs, at some stage, may reach a level of data representing the physical placement of various devices in the hardware model. In cases wherein some semiconductor fabrication techniques are used, the data representing the hardware model may be the data specifying the presence or absence of various features on different mask layers for masks used to produce the integrated circuit. In any representation of the design, the data may be stored in any form of a machine-readable medium. A memory or a magnetic or optical storage such as a disc may be the machine-readable medium to store information transmitted via optical or electrical wave modulated or otherwise generated to transmit such information. When an electrical carrier wave indicating or carrying the code or design is transmitted, to the extent that copying, buffering, or retransmission of the electrical signal is performed, a new copy may be made. Thus, a communication provider or a network provider may store on a tangible, machine-readable medium, at least temporarily, an article, such as information encoded into a carrier wave, embodying techniques of embodiments of the present disclosure.

**[0045]** In modern processors, a number of different execution units may be used to process and execute a variety of code and instructions. Some instructions may be quicker to complete while others may take a number of clock cycles to complete. The faster the throughput of instructions, the better the overall performance of the processor. Thus it would be advantageous to have as many instructions execute as fast as possible. However, there may be certain instructions that have greater complexity and require more in terms of execution time and processor resources, such as floating point instructions, load/store operations, data moves, etc.

**[0046]** As more computer systems are used in internet, text, and multimedia applications, additional processor support has been introduced over time. In one embodiment, an instruction set may be associated with one or more computer architectures, including data types, instructions, register architecture, addressing modes, memory architecture, interrupt and exception handling, and external input and output (I/O).

**[0047]** In one embodiment, the instruction set architecture (ISA) may be implemented by one or more micro-architectures, which may include processor logic and circuits used to implement one or more instruction sets. Accordingly, processors with different micro-architectures may share at least a portion of a common instruction set. For example, Intel® Pentium 4 processors, Intel® Core™ processors, and processors from Advanced Micro Devices, Inc. of Sunnyvale CA implement nearly identical versions of the x86 instruction set (with some extensions that have been added with newer versions), but have different internal designs. Similarly, processors designed by other processor development companies, such as ARM Holdings, Ltd., MIPS, or their licensees or adopters, may share at least a portion a common instruction set, but may include different processor designs. For example, the same register architecture of the ISA may be implemented in different ways in different micro-architectures using new or well-known techniques, including dedicated physical registers, one or more dynamically allocated physical registers using a register renaming mechanism (e.g., the use of a Register Alias Table (RAT) or Register Renaming Table (RRT), a Reorder Buffer (ROB) and a retirement register file. In one embodiment, registers may include one or more registers, register architectures, register files, or other register sets that may or may not be addressable by a software programmer.

**[0048]** An instruction may include one or more instruction formats. In one embodiment, an instruction format may indicate various fields (number of bits, location of bits, etc.) to specify, among other things, the operation to be performed and the operands on which that operation will be performed. In a further embodiment, some instruction formats may be further defined by instruction templates (or sub-formats). For example, the instruction templates of a given instruction format may be defined to have different subsets of the instruction format's fields and/or defined to have a given field interpreted differently. In one embodiment, an instruction may be expressed using an instruction format (and, if defined, in a given one of the instruction templates of that instruction format) and specifies or indicates the operation and the operands upon which the operation will operate.

**[0049]** Scientific, financial, auto-vectorized general purpose, RMS (recognition, mining, and synthesis), and visual and multimedia applications (e.g., 2D/3D graphics, image processing, video compression/decompression, voice recognition algorithms and audio manipulation) may require the same operation to be performed on a large number of data items. In one embodiment, Single Instruction Multiple Data (SIMD) refers to a type of instruction that causes a processor to perform an operation on multiple data elements. SIMD technology may be used in processors that may logically divide the bits in a register into a number of fixed-sized or variable-sized data elements, each of which represents a separate value. For example, in one embodiment, the bits in a 64-bit register may be organized as a source operand containing four separate 16-bit data elements, each of which represents a separate 16-bit value. This type of data may be referred to as 'packed' data type or 'vector' data type, and operands of this data type may be referred to as packed data operands or vector operands. In one embodiment, a packed data item or vector may be a sequence of packed data elements stored within a single register, and a packed data operand or a vector operand may a source or destination operand of a SIMD

instruction (or 'packed data instruction' or a 'vector instruction'). In one embodiment, a SIMD instruction specifies a single vector operation to be performed on two source vector operands to generate a destination vector operand (also referred to as a result vector operand) of the same or different size, with the same or different number of data elements, and in the same or different data element order.

**[0050]** SIMD technology, such as that employed by the Intel® Core™ processors having an instruction set including x86, MMX™, Streaming SIMD Extensions (SSE), SSE2, SSE3, SSE4.1, and SSE4.2 instructions, ARM processors, such as the ARM Cortex® family of processors having an instruction set including the Vector Floating Point (VFP) and/or NEON instructions, and MIPS processors, such as the Loongson family of processors developed by the Institute of Computing Technology (ICT) of the Chinese Academy of Sciences, has enabled a significant improvement in application performance (Core™ and MMX™ are registered trademarks or trademarks of Intel Corporation of Santa Clara, Calif.).

**[0051]** In one embodiment, destination and source registers/data may be generic terms to represent the source and destination of the corresponding data or operation. In some embodiments, they may be implemented by registers, memory, or other storage areas having other names or functions than those depicted. For example, in one embodiment, "DEST1" may be a temporary storage register or other storage area, whereas "SRC1" and "SRC2" may be a first and second source storage register or other storage area, and so forth. In other embodiments, two or more of the SRC and DEST storage areas may correspond to different data storage elements within the same storage area (e.g., a SIMD register). In one embodiment, one of the source registers may also act as a destination register by, for example, writing back the result of an operation performed on the first and second source data to one of the two source registers serving as a destination registers.

**[0052]** Figure 1A is a block diagram of an exemplary computer system formed with a processor that may include execution units to execute an instruction, in accordance with embodiments of the present disclosure. System 100 may include a component, such as a processor 102 to employ execution units including logic to perform algorithms for process data, in accordance with the present disclosure, such as in the embodiment described herein. System 100 may be representative of processing systems based on the PENTIUM® III, PENTIUM® 4, Xeon™, Itanium®, XScale™ and/or StrongARM™ microprocessors available from Intel Corporation of Santa Clara, California, although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and the like) may also be used. In one embodiment, sample system 100 may execute a version of the WINDOWSTM operating system available from Microsoft Corporation of Redmond, Washington, although other operating systems (UNIX and Linux for example), embedded software, and/or graphical user interfaces, may also be used. Thus, embodiments of the present disclosure are not limited to any specific combination of hardware circuitry and software.

**[0053]** Embodiments are not limited to computer systems. Embodiments of the present disclosure may be used in other devices such as handheld devices and embedded applications. Some examples of handheld devices include cellular phones, Internet Protocol devices, digital cameras, personal digital assistants (PDAs), and handheld PCs. Embedded applications may include a micro controller, a digital signal processor (DSP), system on a chip, network computers (NetPC), set-top boxes, network hubs, wide area network (WAN) switches, or any other system that may perform one or more instructions in accordance with at least one embodiment.

**[0054]** Computer system 100 may include a processor 102 that may include one or more execution units 108 to perform an algorithm to perform at least one instruction in accordance with one embodiment of the present disclosure. One embodiment may be described in the context of a single processor desktop or server system, but other embodiments may be included in a multiprocessor system. System 100 may be an example of a 'hub' system architecture. System 100 may include a processor 102 for processing data signals. Processor 102 may include a complex instruction set computer (CISC) microprocessor, a reduced instruction set computing (RISC) microprocessor, a very long instruction word (VLIW) microprocessor, a processor implementing a combination of instruction sets, or any other processor device, such as a digital signal processor, for example. In one embodiment, processor 102 may be coupled to a processor bus 110 that may transmit data signals between processor 102 and other components in system 100. The elements of system 100 may perform conventional functions that are well known to those familiar with the art.

**[0055]** In one embodiment, processor 102 may include a Level 1 (L1) internal cache memory 104. Depending on the architecture, the processor 102 may have a single internal cache or multiple levels of internal cache. In another embodiment, the cache memory may reside external to processor 102. Other embodiments may also include a combination of both internal and external caches depending on the particular implementation and needs. Register file 106 may store different types of data in various registers including integer registers, floating point registers, status registers, and instruction pointer register.

**[0056]** Execution unit 108, including logic to perform integer and floating point operations, also resides in processor 102. Processor 102 may also include a microcode (ucode) ROM that stores microcode for certain macroinstructions. In one embodiment, execution unit 108 may include logic to handle a packed instruction set 109. By including the packed instruction set 109 in the instruction set of a general-purpose processor 102, along with associated circuitry to execute the instructions, the operations used by many multimedia applications may be performed using packed data in a general-purpose processor 102. Thus, many multimedia applications may be accelerated and executed more efficiently by using the full width of a processor's data bus for performing operations on packed data. This may eliminate the need to transfer smaller units of data across the processor's data bus to perform one or more operations one data element at a time.

**[0057]** Embodiments of an execution unit 108 may also be used in micro controllers, embedded processors, graphics devices, DSPs, and other types of logic circuits. System 100 may include a memory 120. Memory 120 may be implemented as a Dynamic Random Access Memory (DRAM) device, a Static Random Access Memory (SRAM) device, flash memory device, or other memory device. Memory 120 may store instructions and/or data represented by data signals that may be executed by processor 102.

**[0058]** A system logic chip 116 may be coupled to processor bus 110 and memory 120. System logic chip 116 may include a memory controller hub (MCH). Processor 102 may communicate with MCH 116 via a processor bus 110. MCH 116 may provide a high bandwidth memory path 118 to memory 120 for instruction and data storage and for storage of graphics commands, data and textures. MCH 116 may direct data signals between processor 102, memory 120, and other components in system 100 and to bridge the data signals between processor bus 110, memory 120, and system I/O 122. In some embodiments, the system logic chip 116 may provide a graphics port for coupling to a graphics controller

112. MCH 116 may be coupled to memory 120 through a memory interface 118. Graphics card 112 may be coupled to MCH 116 through an Accelerated Graphics Port (AGP) interconnect 114.

**[0059]** System 100 may use a proprietary hub interface bus 122 to couple MCH 116 to I/O controller hub (ICH) 130. In one embodiment, ICH 130 may provide direct connections to some I/O devices via a local I/O bus. The local I/O bus may include a high-speed I/O bus for connecting peripherals to memory 120, chipset, and processor 102. Examples may include the audio controller, firmware hub (flash BIOS) 128, wireless transceiver 126, data storage 124, legacy I/O controller containing user input and keyboard interfaces, a serial expansion port such as Universal Serial Bus (USB), and a network controller 134. Data storage device 124 may comprise a hard disk drive, a floppy disk drive, a CD-ROM device, a flash memory device, or other mass storage device.

**[0060]** For another embodiment of a system, an instruction in accordance with one embodiment may be used with a system on a chip. One embodiment of a system on a chip comprises of a processor and a memory. The memory for one such system may include a flash memory. The flash memory may be located on the same die as the processor and other system components. Additionally, other logic blocks such as a memory controller or graphics controller may also be located on a system on a chip.

**[0061]** Figure 1B illustrates a data processing system 140 which implements the principles of embodiments of the present disclosure. It will be readily appreciated by one of skill in the art that the embodiments described herein may operate with alternative processing systems without departure from the scope of embodiments of the disclosure.

**[0062]** Computer system 140 comprises a processing core 159 for performing at least one instruction in accordance with one embodiment. In one embodiment, processing core 159 represents a processing unit of any type of architecture, including but not limited to a CISC, a RISC or a VLIW-type architecture. Processing core 159 may also be suitable for manufacture in one or more process technologies and by being represented on a machine-readable media in sufficient detail, may be suitable to facilitate said manufacture.

**[0063]** Processing core 159 comprises an execution unit 142, a set of register files 145, and a decoder 144. Processing core 159 may also include additional circuitry (not shown) which may be unnecessary to the understanding of embodiments of the present disclosure. Execution unit 142 may execute instructions received by processing core 159. In addition to performing typical processor instructions, execution unit 142 may perform instructions in packed instruction set 143 for performing operations on packed data formats. Packed instruction set 143 may include instructions for performing embodiments of the disclosure and other packed instructions. Execution unit 142 may be coupled to register file 145 by an internal bus. Register file 145 may represent a storage area on processing core 159 for storing information, including data. As previously mentioned, it is understood that the storage area may store the packed data might not be critical. Execution unit 142 may be coupled to decoder 144. Decoder 144 may decode instructions received by processing core 159 into control signals and/or microcode entry points. In response to these control signals and/or microcode entry points, execution unit 142 performs the appropriate operations. In one embodiment, the decoder may interpret the opcode of the instruction, which will indicate what operation should be performed on the corresponding data indicated within the instruction.

**[0064]** Processing core 159 may be coupled with bus 141 for communicating with various other system devices, which may include but are not limited to, for example, Synchronous Dynamic Random Access Memory (SDRAM) control 146, Static Random Access Memory (SRAM) control 147, burst flash memory interface 148, Personal Computer Memory Card International Association (PCMCIA)/Compact Flash (CF) card control 149, Liquid Crystal Display (LCD) control 150, Direct Memory Access (DMA) controller 151, and alternative bus master interface 152. In one embodiment, data processing system 140 may also comprise an I/O bridge 154 for communicating with various I/O devices via an I/O bus 153. Such I/O devices may include but are not limited to, for example, Universal Asynchronous Receiver/Transmitter (UART) 155, Universal Serial Bus (USB) 156, Bluetooth wireless UART 157 and I/O expansion interface 158.

**[0065]** One embodiment of data processing system 140 provides for mobile, network and/or wireless communications and a processing core 159 that may perform SIMD operations including a text string comparison operation. Processing core 159 may be programmed with various audio, video, imaging and communications algorithms including discrete transformations such as a Walsh-Hadamard transform, a fast Fourier transform (FFT), a discrete cosine transform (DCT), and their respective inverse transforms; compression/decompression techniques such as color space transformation, video encode motion estimation or video decode motion compensation; and modulation/demodulation (MODEM) functions such as pulse coded modulation (PCM).

**[0066]** Figure 1C illustrates other embodiments of a data processing system that performs SIMD text string comparison operations. In one embodiment, data processing system 160 may include a main processor 166, a SIMD coprocessor 161, a cache memory 167, and an input/output system 168. Input/output system 168 may optionally be coupled to a wireless interface 169. SIMD coprocessor 161 may perform operations including instructions in accordance with one embodiment. In one embodiment, processing core 170 may be suitable for manufacture in one or more process technologies and by being represented on a machine-readable media in sufficient detail, may be suitable to facilitate the manufacture of all or part of data processing system 160 including processing core 170.

**[0067]** In one embodiment, SIMD coprocessor 161 comprises an execution unit 162 and a set of register files 164. One embodiment of main processor 165 comprises a decoder 165 to recognize instructions of instruction set 163 including instructions in accordance with one embodiment for execution by execution unit 162. In other embodiments, SIMD coprocessor 161 also comprises at least part of decoder 165 to decode instructions of instruction set 163. Processing core 170 may also include additional circuitry (not shown) which may be unnecessary to the understanding of embodiments of the present disclosure.

**[0068]** In operation, main processor 166 executes a stream of data processing instructions that control data processing operations of a general type including interactions with cache memory 167, and input/output system 168. Embedded within the stream of data processing instructions may be SIMD coprocessor instructions. Decoder 165 of main processor 166 recognizes these SIMD coprocessor instructions as being of a type that should be executed by an attached SIMD coprocessor 161. Accordingly, main processor 166 issues these SIMD coprocessor instructions (or control signals representing SIMD coprocessor instructions) on the coprocessor bus 166. From coprocessor bus 166, these instructions may be received by any attached SIMD coprocessors. In this case, SIMD coprocessor 161 may accept and execute any received SIMD coprocessor instructions intended for it.

**[0069]** Data may be received via wireless interface 169 for processing by the SIMD coprocessor instructions. For one example, voice communication may be received in the form of a digital signal, which may be processed by the SIMD coprocessor instructions to regenerate digital audio samples representative of the voice communications. For another example, compressed audio and/or video may be received in the form of a digital bit stream, which may be processed by the SIMD coprocessor instructions to regenerate digital audio samples and/or motion video frames. In one embodiment of processing core 170, main processor 166, and a SIMD coprocessor 161 may be integrated into a single processing core 170 comprising an execution unit 162, a set of register files 164, and a decoder 165 to recognize instructions of instruction set 163 including instructions in accordance with one embodiment.

**[0070]** Figure 2 is a block diagram of the micro-architecture for a processor 200 that may include logic circuits to perform instructions, in accordance with embodiments of the present disclosure. In some embodiments, an instruction in accordance with one embodiment may be implemented to operate on data elements having sizes of byte, word, doubleword, quadword, etc., as well as datatypes, such as single and double precision integer and floating point datatypes. In one embodiment, in-order front end 201 may implement a part of processor 200 that may fetch instructions to be executed and prepares the instructions to be used later in the processor pipeline. Front end 201 may include several units. In one embodiment, instruction prefetcher 226 fetches instructions from memory and feeds the instructions to an instruction decoder 228 which in turn decodes or interprets the instructions. For example, in one embodiment, the decoder decodes a received instruction into one or more operations called "micro-instructions" or "micro-operations" (also called micro op or uops) that the machine may execute. In other embodiments, the decoder parses the instruction into an opcode and corresponding data and control fields that may be used by the micro-architecture to perform operations in accordance with one embodiment. In one embodiment, trace cache 230 may assemble decoded uops into program ordered sequences or traces in uop queue 234 for execution. When trace cache 230 encounters a complex instruction, microcode ROM 232 provides the uops needed to complete the operation.

**[0071]** Some instructions may be converted into a single micro-op, whereas others need several micro-ops to complete the full operation. In one embodiment, if more than four micro-ops are needed to complete an instruction, decoder 228 may access microcode ROM 232 to perform the instruction. In one embodiment, an instruction may be decoded into a small number of micro-ops for processing at instruction decoder 228. In another embodiment, an instruction may be stored within microcode ROM 232 should a number of micro-ops be needed to accomplish the operation. Trace cache 230 refers to an entry point programmable logic array (PLA) to determine a correct micro-instruction pointer for reading the micro-code sequences to complete one or more instructions in accordance with one embodiment from micro-code ROM 232. After microcode ROM 232 finishes sequencing micro-ops for an instruction, front end 201 of the machine may resume fetching micro-ops from trace cache 230.

**[0072]** Out-of-order execution engine 203 may prepare instructions for execution. The out-of-order execution logic has a number of buffers to smooth out and re-order the flow of instructions to optimize performance as they go down the pipeline and get scheduled for execution. The allocator logic allocates the machine buffers and resources that each uop needs in order to execute. The register renaming logic renames logic registers onto entries in a register file. The allocator also allocates an entry for each uop in one of the two uop queues, one for memory operations and one for non-memory operations, in front of the instruction schedulers: memory scheduler, fast scheduler 202, slow/general floating point scheduler

204, and simple floating point scheduler 206. Uop schedulers 202, 204, 206, determine when a uop is ready to execute based on the readiness of their dependent input register operand sources and the availability of the execution resources the uops need to complete their operation. Fast scheduler 202 of one embodiment may schedule on each half of the main clock cycle while the other schedulers may only schedule once per main processor clock cycle. The schedulers arbitrate for the dispatch ports to schedule uops for execution.

**[0073]** Register files 208, 210 may be arranged between schedulers 202, 204, 206, and execution units 212, 214, 216, 218, 220, 222, 224 in execution block 211. Each of register files 208, 210 perform integer and floating point operations, respectively. Each register file 208, 210, may include a bypass network that may bypass or forward just completed results that have not yet been written into the register file to new dependent uops. Integer register file 208 and floating point register file 210 may communicate data with the other. In one embodiment, integer register file 208 may be split into two separate register files, one register file for low-order thirty-two bits of data and a second register file for high order thirty-two bits of data. Floating point register file 210 may include 128-bit wide entries because floating point instructions typically have operands from 64 to 128 bits in width.

**[0074]** Execution block 211 may contain execution units 212, 214, 216, 218, 220, 222, 224. Execution units 212, 214, 216, 218, 220, 222, 224 may execute the instructions. Execution block 211 may include register files 208, 210 that store the integer and floating point data operand values that the micro-instructions need to execute. In one embodiment, processor 200 may comprise a number of execution units: address generation unit (AGU) 212, AGU 214, fast Arithmetic Logic Unit (ALU) 216, fast ALU 218, slow ALU 220, floating point ALU 222, floating point move unit 224. In another embodiment, floating point execution blocks 222, 224, may execute floating point, MMX, SIMD, and SSE, or other operations. In yet another embodiment, floating point ALU 222 may include a 64-bit by 64-bit floating point divider to execute divide, square root, and remainder micro-ops. In various embodiments, instructions involving a floating point value may be handled with the floating point hardware. In one embodiment, ALU operations may be passed to high-speed ALU execution units 216, 218. High-speed ALUs 216, 218 may execute fast operations with an effective latency of half a clock cycle. In one embodiment, most complex integer operations go to slow ALU 220 as slow ALU 220 may include integer execution hardware for long-latency type of operations, such as a multiplier, shifts, flag logic, and branch processing. Memory load/store operations may be executed by AGUs 212, 214. In one embodiment, integer ALUs 216, 218, 220 may perform integer operations on 64-bit data operands. In other embodiments, ALUs 216, 218, 220 may be implemented to support a variety of data bit sizes including sixteen, thirty-two, 128, 256, etc. Similarly, floating point units 222, 224 may be implemented to support a range of operands having bits of various widths. In one embodiment, floating point units 222, 224, may operate on 128-bit wide packed data operands in conjunction with SIMD and multimedia instructions.

**[0075]** In one embodiment, uops schedulers 202, 204, 206, dispatch dependent operations before the parent load has finished executing. As uops may be speculatively scheduled and executed in processor 200, processor 200 may also include logic to handle memory misses. If a data load misses in the data cache, there may be dependent operations in flight in the pipeline that have left the scheduler with temporarily incorrect data. A replay mechanism tracks and re-executes instructions that use incorrect data. Only the dependent operations might need to be replayed and the independent ones may

be allowed to complete. The schedulers and replay mechanism of one embodiment of a processor may also be designed to catch instruction sequences for text string comparison operations.

**[0076]** The term "registers" may refer to the on-board processor storage locations that may be used as part of instructions to identify operands. In other words, registers may be those that may be usable from the outside of the processor (from a programmer's perspective). However, in some embodiments registers might not be limited to a particular type of circuit. Rather, a register may store data, provide data, and perform the functions described herein. The registers described herein may be implemented by circuitry within a processor using any number of different techniques, such as dedicated physical registers, dynamically allocated physical registers using register renaming, combinations of dedicated and dynamically allocated physical registers, etc. In one embodiment, integer registers store 32-bit integer data. A register file of one embodiment also contains eight multimedia SIMD registers for packed data. For the discussions below, the registers may be understood to be data registers designed to hold packed data, such as 64-bit wide MMXTM registers (also referred to as 'mm' registers in some instances) in microprocessors enabled with MMX technology from Intel Corporation of Santa Clara, California. These MMX registers, available in both integer and floating point forms, may operate with packed data elements that accompany SIMD and SSE instructions. Similarly, 128-bit wide XMM registers relating to SSE2, SSE3, SSE4, or beyond (referred to generically as "SSEx") technology may hold such packed data operands. In one embodiment, in storing packed data and integer data, the registers do not need to differentiate between the two data types. In one embodiment, integer and floating point may be contained in the same register file or different register files. Furthermore, in one embodiment, floating point and integer data may be stored in different registers or the same registers.

**[0077]** In the examples of the following figures, a number of data operands may be described. Figure 3A illustrates various packed data type representations in multimedia registers, in accordance with embodiments of the present disclosure. Figure 3A illustrates data types for a packed byte 310, a packed word 320, and a packed doubleword (dword) 330 for 128-bit wide operands. Packed byte format 310 of this example may be 128 bits long and contains sixteen packed byte data elements. A byte may be defined, for example, as eight bits of data. Information for each byte data element may be stored in bit 7 through bit 0 for byte 0, bit 15 through bit 8 for byte 1, bit 23 through bit 16 for byte 2, and finally bit 120 through bit 127 for byte 15. Thus, all available bits may be used in the register. This storage arrangement increases the storage efficiency of the processor. As well, with sixteen data elements accessed, one operation may now be performed on sixteen data elements in parallel.

**[0078]** Generally, a data element may include an individual piece of data that is stored in a single register or memory location with other data elements of the same length. In packed data sequences relating to SSEx technology, the number of data elements stored in a XMM register may be 128 bits divided by the length in bits of an individual data element. Similarly, in packed data sequences relating to MMX and SSE technology, the number of data elements stored in an MMX register may be 64 bits divided by the length in bits of an individual data element. Although the data types illustrated in Figure 3A may be 128 bits long, embodiments of the present disclosure may also operate with 64-bit wide or other sized operands. Packed word format 320 of this example may be 128 bits long and contains eight packed word data elements. Each packed word contains sixteen bits of information. Packed doubleword format 330 of Figure 3A may be 128 bits long and contains four packed doubleword data elements. Each

packed doubleword data element contains thirty-two bits of information. A packed quadword may be 128 bits long and contain two packed quad-word data elements.

**[0079]** Figure 3B illustrates possible in-register data storage formats, in accordance with embodiments of the present disclosure. Each packed data may include more than one independent data element. Three packed data formats are illustrated; packed half 341, packed single 342, and packed double 343. One embodiment of packed half 341, packed single 342, and packed double 343 contain fixed-point data elements. For another embodiment one or more of packed half 341, packed single 342, and packed double 343 may contain floating-point data elements. One embodiment of packed half 341 may be 128 bits long containing eight 16-bit data elements. One embodiment of packed single 342 may be 128 bits long and contains four 32-bit data elements. One embodiment of packed double 343 may be 128 bits long and contains two 64-bit data elements. It will be appreciated that such packed data formats may be further extended to other register lengths, for example, to 96-bits, 160-bits, 192-bits, 224-bits, 256-bits or more.

**[0080]** Figure 3C illustrates various signed and unsigned packed data type representations in multimedia registers, in accordance with embodiments of the present disclosure. Unsigned packed byte representation 344 illustrates the storage of an unsigned packed byte in a SIMD register. Information for each byte data element may be stored in bit 7 through bit 0 for byte 0, bit 15 through bit 8 for byte 1, bit 23 through bit 16 for byte 2, and finally bit 120 through bit 127 for byte 15. Thus, all available bits may be used in the register. This storage arrangement may increase the storage efficiency of the processor. As well, with sixteen data elements accessed, one operation may now be performed on sixteen data elements in a parallel fashion. Signed packed byte representation 345 illustrates the storage of a signed packed byte. Note that the eighth bit of every byte data element may be the sign indicator. Unsigned packed word representation 346 illustrates how word seven through word zero may be stored in a SIMD register. Signed packed word representation 347 may be similar to the unsigned packed word in-register representation 346. Note that the sixteenth bit of each word data element may be the sign indicator. Unsigned packed doubleword representation 348 shows how doubleword data elements are stored. Signed packed doubleword representation 349 may be similar to unsigned packed doubleword in-register representation 348. Note that the necessary sign bit may be the thirty-second bit of each doubleword data element.

**[0081]** Figure 3D illustrates an embodiment of an operation encoding (opcode). Furthermore, format 360 may include register/memory operand addressing modes corresponding with a type of opcode format described in the "IA-32 Intel Architecture Software Developer's Manual Volume 2: Instruction Set Reference," which is available from Intel Corporation, Santa Clara, CA on the world-wide-web (www) at [intel.com/design/litcentr](http://intel.com/design/litcentr). In one embodiment, an instruction may be encoded by one or more of fields 361 and 362. Up to two operand locations per instruction may be identified, including up to two source operand identifiers 364 and 365. In one embodiment, destination operand identifier 366 may be the same as source operand identifier 364, whereas in other embodiments they may be different. In another embodiment, destination operand identifier 366 may be the same as source operand identifier 365, whereas in other embodiments they may be different. In one embodiment, one of the source operands identified by source operand identifiers 364 and 365 may be overwritten by the results of the text string comparison operations, whereas in other embodiments identifier 364 corresponds to a source register

element and identifier 365 corresponds to a destination register element. In one embodiment, operand identifiers 364 and 365 may identify 32-bit or 64-bit source and destination operands.

**[0082]** Figure 3E illustrates another possible operation encoding (opcode) format 370, having forty or more bits, in accordance with embodiments of the present disclosure. Opcode format 370 corresponds with opcode format 360 and comprises an optional prefix byte 378. An instruction according to one embodiment may be encoded by one or more of fields 378, 371, and 372. Up to two operand locations per instruction may be identified by source operand identifiers 374 and 375 and by prefix byte 378. In one embodiment, prefix byte 378 may be used to identify 32-bit or 64-bit source and destination operands. In one embodiment, destination operand identifier 376 may be the same as source operand identifier 374, whereas in other embodiments they may be different. For another embodiment, destination operand identifier 376 may be the same as source operand identifier 375, whereas in other embodiments they may be different. In one embodiment, an instruction operates on one or more of the operands identified by operand identifiers 374 and 375 and one or more operands identified by operand identifiers 374 and 375 may be overwritten by the results of the instruction, whereas in other embodiments, operands identified by identifiers 374 and 375 may be written to another data element in another register. Opcode formats 360 and 370 allow register to register, memory to register, register by memory, register by register, register by immediate, register to memory addressing specified in part by MOD fields 363 and 373 and by optional scale-index-base and displacement bytes.

**[0083]** Figure 3F illustrates yet another possible operation encoding (opcode) format, in accordance with embodiments of the present disclosure. 64-bit single instruction multiple data (SIMD) arithmetic operations may be performed through a coprocessor data processing (CDP) instruction. Operation encoding (opcode) format 380 depicts one such CDP instruction having CDP opcode fields 382 and 389. The type of CDP instruction, for another embodiment, operations may be encoded by one or more of fields 383, 384, 387, and 388. Up to three operand locations per instruction may be identified, including up to two source operand identifiers 385 and 390 and one destination operand identifier 386. One embodiment of the coprocessor may operate on eight, sixteen, thirty-two, and 64-bit values. In one embodiment, an instruction may be performed on integer data elements. In some embodiments, an instruction may be executed conditionally, using condition field 381. For some embodiments, source data sizes may be encoded by field 383. In some embodiments, Zero (Z), negative (N), carry (C), and overflow (V) detection may be done on SIMD fields. For some instructions, the type of saturation may be encoded by field 384.

**[0084]** Figure 4A is a block diagram illustrating an in-order pipeline and a register renaming stage, out-of-order issue/execution pipeline, in accordance with embodiments of the present disclosure. Figure 4B is a block diagram illustrating an in-order architecture core and a register renaming logic, out-of-order issue/execution logic to be included in a processor, in accordance with embodiments of the present disclosure. The solid lined boxes in Figure 4A illustrate the in-order pipeline, while the dashed lined boxes illustrates the register renaming, out-of-order issue/execution pipeline. Similarly, the solid lined boxes in Figure 4B illustrate the in-order architecture logic, while the dashed lined boxes illustrates the register renaming logic and out-of-order issue/execution logic.

**[0085]** In Figure 4A, a processor pipeline 400 may include a fetch stage 402, a length decode stage 404, a decode stage 406, an allocation stage 408, a renaming stage 410, a scheduling (also known as a

dispatch or issue) stage 412, a register read/memory read stage 414, an execute stage 416, a write-back/memory-write stage 418, an exception handling stage 422, and a commit stage 424.

**[0086]** In Figure 4B, arrows denote a coupling between two or more units and the direction of the arrow indicates a direction of data flow between those units. Figure 4B shows processor core 490 including a front end unit 430 coupled to an execution engine unit 450, and both may be coupled to a memory unit 470.

**[0087]** Core 490 may be a Reduced Instruction Set Computing (RISC) core, a Complex Instruction Set Computing (CISC) core, a Very Long Instruction Word (VLIW) core, or a hybrid or alternative core type. In one embodiment, core 490 may be a special-purpose core, such as, for example, a network or communication core, compression engine, graphics core, or the like.

**[0088]** Front end unit 430 may include a branch prediction unit 432 coupled to an instruction cache unit 434. Instruction cache unit 434 may be coupled to an instruction Translation Lookaside Buffer (TLB) 436. TLB 436 may be coupled to an instruction fetch unit 438, which is coupled to a decode unit 440. Decode unit 440 may decode instructions, and generate as an output one or more micro-operations, micro-code entry points, microinstructions, other instructions, or other control signals, which may be decoded from, or which otherwise reflect, or may be derived from, the original instructions. The decoder may be implemented using various different mechanisms. Examples of suitable mechanisms include, but are not limited to, look-up tables, hardware implementations, programmable logic arrays (PLAs), microcode read-only memories (ROMs), etc. In one embodiment, instruction cache unit 434 may be further coupled to a level 2 (L2) cache unit 476 in memory unit 470. Decode unit 440 may be coupled to a rename/allocator unit 452 in execution engine unit 450.

**[0089]** Execution engine unit 450 may include rename/allocator unit 452 coupled to a retirement unit 454 and a set of one or more scheduler units 456. Scheduler units 456 represent any number of different schedulers, including reservations stations, central instruction window, etc. Scheduler units 456 may be coupled to physical register file units 458. Each of physical register file units 458 represents one or more physical register files, different ones of which store one or more different data types, such as scalar integer, scalar floating point, packed integer, packed floating point, vector integer, vector floating point, etc., status (e.g., an instruction pointer that is the address of the next instruction to be executed), etc. Physical register file units 458 may be overlapped by retirement unit 154 to illustrate various ways in which register renaming and out-of-order execution may be implemented (e.g., using one or more reorder buffers and one or more retirement register files, using one or more future files, one or more history buffers, and one or more retirement register files; using register maps and a pool of registers; etc.). Generally, the architectural registers may be visible from the outside of the processor or from a programmer's perspective. The registers might not be limited to any known particular type of circuit. Various different types of registers may be suitable as long as they store and provide data as described herein. Examples of suitable registers include, but might not be limited to, dedicated physical registers, dynamically allocated physical registers using register renaming, combinations of dedicated and dynamically allocated physical registers, etc. Retirement unit 454 and physical register file units 458 may be coupled to execution clusters 460. Execution clusters 460 may include a set of one or more execution units 462 and a set of one or more memory access units 464. Execution units 462 may perform various operations (e.g., shifts, addition, subtraction, multiplication) and on various types of data (e.g., scalar floating point, packed integer, packed floating point, vector integer, vector floating point). While some

embodiments may include a number of execution units dedicated to specific functions or sets of functions, other embodiments may include only one execution unit or multiple execution units that all perform all functions. Scheduler units 456, physical register file units 458, and execution clusters 460 are shown as being possibly plural because certain embodiments create separate pipelines for certain types of data/operations (e.g., a scalar integer pipeline, a scalar floating point/packed integer/packed floating point/vector integer/vector floating point pipeline, and/or a memory access pipeline that each have their own scheduler unit, physical register file unit, and/or execution cluster – and in the case of a separate memory access pipeline, certain embodiments may be implemented in which only the execution cluster of this pipeline has memory access units 464). It should also be understood that where separate pipelines are used, one or more of these pipelines may be out-of-order issue/execution and the rest in-order.

**[0090]** The set of memory access units 464 may be coupled to memory unit 470, which may include a data TLB unit 472 coupled to a data cache unit 474 coupled to a level 2 (L2) cache unit 476. In one exemplary embodiment, memory access units 464 may include a load unit, a store address unit, and a store data unit, each of which may be coupled to data TLB unit 472 in memory unit 470. L2 cache unit 476 may be coupled to one or more other levels of cache and eventually to a main memory.

**[0091]** By way of example, the exemplary register renaming, out-of-order issue/execution core architecture may implement pipeline 400 as follows: 1) instruction fetch 438 may perform fetch and length decoding stages 402 and 404; 2) decode unit 440 may perform decode stage 406; 3) rename/allocator unit 452 may perform allocation stage 408 and renaming stage 410; 4) scheduler units 456 may perform schedule stage 412; 5) physical register file units 458 and memory unit 470 may perform register read/memory read stage 414; execution cluster 460 may perform execute stage 416; 6) memory unit 470 and physical register file units 458 may perform write-back/memory-write stage 418; 7) various units may be involved in the performance of exception handling stage 422; and 8) retirement unit 454 and physical register file units 458 may perform commit stage 424.

**[0092]** Core 490 may support one or more instructions sets (e.g., the x86 instruction set (with some extensions that have been added with newer versions); the MIPS instruction set of MIPS Technologies of Sunnyvale, CA; the ARM instruction set (with optional additional extensions such as NEON) of ARM Holdings of Sunnyvale, CA).

**[0093]** It should be understood that the core may support multithreading (executing two or more parallel sets of operations or threads) in a variety of manners. Multithreading support may be performed by, for example, including time sliced multithreading, simultaneous multithreading (where a single physical core provides a logical core for each of the threads that physical core is simultaneously multithreading), or a combination thereof. Such a combination may include, for example, time sliced fetching and decoding and simultaneous multithreading thereafter such as in the Intel® Hyperthreading technology.

**[0094]** While register renaming may be described in the context of out-of-order execution, it should be understood that register renaming may be used in an in-order architecture. While the illustrated embodiment of the processor may also include a separate instruction and data cache units 434/474 and a shared L2 cache unit 476, other embodiments may have a single internal cache for both instructions and data, such as, for example, a Level 1 (L1) internal cache, or multiple levels of internal cache. In some embodiments, the system may include a combination of an internal cache and an external cache that may

be external to the core and/or the processor. In other embodiments, all of the cache may be external to the core and/or the processor.

**[0095]** Figure 5A is a block diagram of a processor 500, in accordance with embodiments of the present disclosure. In one embodiment, processor 500 may include a multicore processor. Processor 500 may include a system agent 510 communicatively coupled to one or more cores 502. Furthermore, cores 502 and system agent 510 may be communicatively coupled to one or more caches 506. Cores 502, system agent 510, and caches 506 may be communicatively coupled via one or more memory control units 552. Furthermore, cores 502, system agent 510, and caches 506 may be communicatively coupled to a graphics module 560 via memory control units 552.

**[0096]** Processor 500 may include any suitable mechanism for interconnecting cores 502, system agent 510, and caches 506, and graphics module 560. In one embodiment, processor 500 may include a ring-based interconnect unit 508 to interconnect cores 502, system agent 510, and caches 506, and graphics module 560. In other embodiments, processor 500 may include any number of well-known techniques for interconnecting such units. Ring-based interconnect unit 508 may utilize memory control units 552 to facilitate interconnections.

**[0097]** Processor 500 may include a memory hierarchy comprising one or more levels of caches within the cores, one or more shared cache units such as caches 506, or external memory (not shown) coupled to the set of integrated memory controller units 552. Caches 506 may include any suitable cache. In one embodiment, caches 506 may include one or more mid-level caches, such as Level 2 (L2), Level 3 (L3), Level 4 (L4), or other levels of cache, a last level cache (LLC), and/or combinations thereof.

**[0098]** In various embodiments, one or more of cores 502 may perform multi-threading. System agent 510 may include components for coordinating and operating cores 502. System agent unit 510 may include for example a Power Control Unit (PCU). The PCU may be or include logic and components needed for regulating the power state of cores 502. System agent 510 may include a display engine 512 for driving one or more externally connected displays or graphics module 560. System agent 510 may include an interface 1214 for communications busses for graphics. In one embodiment, interface 1214 may be implemented by PCI Express (PCIe). In a further embodiment, interface 1214 may be implemented by PCI Express Graphics (PEG). System agent 510 may include a direct media interface (DMI) 516. DMI 516 may provide links between different bridges on a motherboard or other portion of a computer system. System agent 510 may include a PCIe bridge 1218 for providing PCIe links to other elements of a computing system. PCIe bridge 1218 may be implemented using a memory controller 1220 and coherence logic 1222.

**[0099]** Cores 502 may be implemented in any suitable manner. Cores 502 may be homogenous or heterogeneous in terms of architecture and/or instruction set. In one embodiment, some of cores 502 may be in-order while others may be out-of-order. In another embodiment, two or more of cores 502 may execute the same instruction set, while others may execute only a subset of that instruction set or a different instruction set.

**[00100]** Processor 500 may include a general-purpose processor, such as a Core™ i3, i5, i7, 2 Duo and Quad, Xeon™, Itanium™, XScale™ or StrongARM™ processor, which may be available from Intel Corporation, of Santa Clara, Calif. Processor 500 may be provided from another company, such as ARM Holdings, Ltd, MIPS, etc. Processor 500 may be a special-purpose processor, such as, for example, a

network or communication processor, compression engine, graphics processor, co-processor, embedded processor, or the like. Processor 500 may be implemented on one or more chips. Processor 500 may be a part of and/or may be implemented on one or more substrates using any of a number of process technologies, such as, for example, BiCMOS, CMOS, or NMOS.

**[00101]** In one embodiment, a given one of caches 506 may be shared by multiple ones of cores 502. In another embodiment, a given one of caches 506 may be dedicated to one of cores 502. The assignment of caches 506 to cores 502 may be handled by a cache controller or other suitable mechanism. A given one of caches 506 may be shared by two or more cores 502 by implementing time-slices of a given cache 506.

**[00102]** Graphics module 560 may implement an integrated graphics processing subsystem. In one embodiment, graphics module 560 may include a graphics processor. Furthermore, graphics module 560 may include a media engine 565. Media engine 565 may provide media encoding and video decoding.

**[00103]** Figure 5B is a block diagram of an example implementation of a core 502, in accordance with embodiments of the present disclosure. Core 502 may include a front end 570 communicatively coupled to an out-of-order engine 580. Core 502 may be communicatively coupled to other portions of processor 500 through cache hierarchy 503.

**[00104]** Front end 570 may be implemented in any suitable manner, such as fully or in part by front end 201 as described above. In one embodiment, front end 570 may communicate with other portions of processor 500 through cache hierarchy 503. In a further embodiment, front end 570 may fetch instructions from portions of processor 500 and prepare the instructions to be used later in the processor pipeline as they are passed to out-of-order execution engine 580.

**[00105]** Out-of-order execution engine 580 may be implemented in any suitable manner, such as fully or in part by out-of-order execution engine 203 as described above. Out-of-order execution engine 580 may prepare instructions received from front end 570 for execution. Out-of-order execution engine 580 may include an allocate module 1282. In one embodiment, allocate module 1282 may allocate resources of processor 500 or other resources, such as registers or buffers, to execute a given instruction. Allocate module 1282 may make allocations in schedulers, such as a memory scheduler, fast scheduler, or floating point scheduler. Such schedulers may be represented in Figure 5B by resource schedulers 584. Allocate module 1282 may be implemented fully or in part by the allocation logic described in conjunction with Figure 2. Resource schedulers 584 may determine when an instruction is ready to execute based on the readiness of a given resource's sources and the availability of execution resources needed to execute an instruction. Resource schedulers 584 may be implemented by, for example, schedulers 202, 204, 206 as discussed above. Resource schedulers 584 may schedule the execution of instructions upon one or more resources. In one embodiment, such resources may be internal to core 502, and may be illustrated, for example, as resources 586. In another embodiment, such resources may be external to core 502 and may be accessible by, for example, cache hierarchy 503. Resources may include, for example, memory, caches, register files, or registers. Resources internal to core 502 may be represented by resources 586 in Figure 5B. As necessary, values written to or read from resources 586 may be coordinated with other portions of processor 500 through, for example, cache hierarchy 503. As instructions are assigned resources, they may be placed into a reorder buffer 588. Reorder buffer 588 may track instructions as they are executed and may selectively reorder their execution based upon any suitable criteria of processor 500. In one embodiment, reorder buffer 588 may

identify instructions or a series of instructions that may be executed independently. Such instructions or a series of instructions may be executed in parallel from other such instructions. Parallel execution in core 502 may be performed by any suitable number of separate execution blocks or virtual processors. In one embodiment, shared resources—such as memory, registers, and caches—may be accessible to multiple virtual processors within a given core 502. In other embodiments, shared resources may be accessible to multiple processing entities within processor 500.

**[00106]** Cache hierarchy 503 may be implemented in any suitable manner. For example, cache hierarchy 503 may include one or more lower or mid-level caches, such as caches 572, 574. In one embodiment, cache hierarchy 503 may include an LLC 595 communicatively coupled to caches 572, 574. In another embodiment, LLC 595 may be implemented in a module 590 accessible to all processing entities of processor 500. In a further embodiment, module 590 may be implemented in an uncore module of processors from Intel, Inc. Module 590 may include portions or subsystems of processor 500 necessary for the execution of core 502 but might not be implemented within core 502. Besides LLC 595, Module 590 may include, for example, hardware interfaces, memory coherency coordinators, interprocessor interconnects, instruction pipelines, or memory controllers. Access to RAM 599 available to processor 500 may be made through module 590 and, more specifically, LLC 595. Furthermore, other instances of core 502 may similarly access module 590. Coordination of the instances of core 502 may be facilitated in part through module 590.

**[00107]** Figures 6-8 may illustrate exemplary systems suitable for including processor 500, while Figure 9 may illustrate an exemplary System on a Chip (SoC) that may include one or more of cores 502. Other system designs and implementations known in the arts for laptops, desktops, handheld PCs, personal digital assistants, engineering workstations, servers, network devices, network hubs, switches, embedded processors, DSPs, graphics devices, video game devices, set-top boxes, micro controllers, cell phones, portable media players, hand held devices, and various other electronic devices, may also be suitable. In general, a huge variety of systems or electronic devices that incorporate a processor and/or other execution logic as disclosed herein may be generally suitable.

**[00108]** Figure 6 illustrates a block diagram of a system 600, in accordance with embodiments of the present disclosure. System 600 may include one or more processors 610, 615, which may be coupled to Graphics Memory Controller Hub (GMCH) 620. The optional nature of additional processors 615 is denoted in Figure 6 with broken lines.

**[00109]** Each processor 610, 615 may be some version of processor 500. However, it should be noted that integrated graphics logic and integrated memory control units might not exist in processors 610, 615. Figure 6 illustrates that GMCH 620 may be coupled to a memory 640 that may be, for example, a dynamic random access memory (DRAM). The DRAM may, for at least one embodiment, be associated with a non-volatile cache.

**[00110]** GMCH 620 may be a chipset, or a portion of a chipset. GMCH 620 may communicate with processors 610, 615 and control interaction between processors 610, 615 and memory 640. GMCH 620 may also act as an accelerated bus interface between the processors 610, 615 and other elements of system 600. In one embodiment, GMCH 620 communicates with processors 610, 615 via a multi-drop bus, such as a frontside bus (FSB) 695.

**[00111]** Furthermore, GMCH 620 may be coupled to a display 645 (such as a flat panel display). In one embodiment, GMCH 620 may include an integrated graphics accelerator. GMCH 620 may be further coupled to an input/output (I/O) controller hub (ICH) 650, which may be used to couple various peripheral devices to system 600. External graphics device 660 may include be a discrete graphics device coupled to ICH 650 along with another peripheral device 670.

**[00112]** In other embodiments, additional or different processors may also be present in system 600. For example, additional processors 610, 615 may include additional processors that may be the same as processor 610, additional processors that may be heterogeneous or asymmetric to processor 610, accelerators (such as, e.g., graphics accelerators or digital signal processing (DSP) units), field programmable gate arrays, or any other processor. There may be a variety of differences between the physical resources 610, 615 in terms of a spectrum of metrics of merit including architectural, micro-architectural, thermal, power consumption characteristics, and the like. These differences may effectively manifest themselves as asymmetry and heterogeneity amongst processors 610, 615. For at least one embodiment, various processors 610, 615 may reside in the same die package.

**[00113]** Figure 7 illustrates a block diagram of a second system 700, in accordance with embodiments of the present disclosure. As shown in Figure 7, multiprocessor system 700 may include a point-to-point interconnect system, and may include a first processor 770 and a second processor 780 coupled via a point-to-point interconnect 750. Each of processors 770 and 780 may be some version of processor 500 as one or more of processors 610, 615.

**[00114]** While Figure 7 may illustrate two processors 770, 780, it is to be understood that the scope of the present disclosure is not so limited. In other embodiments, one or more additional processors may be present in a given processor.

**[00115]** Processors 770 and 780 are shown including integrated memory controller units 772 and 782, respectively. Processor 770 may also include as part of its bus controller units point-to-point (P-P) interfaces 776 and 778; similarly, second processor 780 may include P-P interfaces 786 and 788. Processors 770, 780 may exchange information via a point-to-point (P-P) interface 750 using P-P interface circuits 778, 788. As shown in Figure 7, IMCs 772 and 782 may couple the processors to respective memories, namely a memory 732 and a memory 734, which in one embodiment may be portions of main memory locally attached to the respective processors.

**[00116]** Processors 770, 780 may each exchange information with a chipset 790 via individual P-P interfaces 752, 754 using point to point interface circuits 776, 794, 786, 798. In one embodiment, chipset 790 may also exchange information with a high-performance graphics circuit 738 via a high-performance graphics interface 739.

**[00117]** A shared cache (not shown) may be included in either processor or outside of both processors, yet connected with the processors via P-P interconnect, such that either or both processors' local cache information may be stored in the shared cache if a processor is placed into a low power mode.

**[00118]** Chipset 790 may be coupled to a first bus 716 via an interface 796. In one embodiment, first bus 716 may be a Peripheral Component Interconnect (PCI) bus, or a bus such as a PCI Express bus or another third generation I/O interconnect bus, although the scope of the present disclosure is not so limited.

**[00119]** As shown in Figure 7, various I/O devices 714 may be coupled to first bus 716, along with a bus bridge 718 which couples first bus 716 to a second bus 720. In one embodiment, second bus 720 may be a Low Pin Count (LPC) bus. Various devices may be coupled to second bus 720 including, for example, a keyboard and/or mouse 722, communication devices 727 and a storage unit 728 such as a disk drive or other mass storage device which may include instructions/code and data 730, in one embodiment. Further, an audio I/O 724 may be coupled to second bus 720. Note that other architectures may be possible. For example, instead of the point-to-point architecture of Figure 7, a system may implement a multi-drop bus or other such architecture.

**[00120]** Figure 8 illustrates a block diagram of a third system 800 in accordance with embodiments of the present disclosure. Like elements in Figures 7 and 8 bear like reference numerals, and certain aspects of Figure 7 have been omitted from Figure 8 in order to avoid obscuring other aspects of Figure 8.

**[00121]** Figure 8 illustrates that processors 870, 880 may include integrated memory and I/O Control Logic ("CL") 872 and 882, respectively. For at least one embodiment, CL 872, 882 may include integrated memory controller units such as that described above in connection with Figures 5 and 7. In addition, CL 872, 882 may also include I/O control logic. Figure 8 illustrates that not only memories 832, 834 may be coupled to CL 872, 882, but also that I/O devices 814 may also be coupled to control logic 872, 882. Legacy I/O devices 815 may be coupled to chipset 890.

**[00122]** Figure 9 illustrates a block diagram of a SoC 900, in accordance with embodiments of the present disclosure. Similar elements in Figure 5 bear like reference numerals. Also, dashed lined boxes may represent optional features on more advanced SoCs. An interconnect units 902 may be coupled to: an application processor 910 which may include a set of one or more cores 902A-N and shared cache units 906; a system agent unit 910; a bus controller units 916; an integrated memory controller units 914; a set or one or more media processors 920 which may include integrated graphics logic 908, an image processor 924 for providing still and/or video camera functionality, an audio processor 926 for providing hardware audio acceleration, and a video processor 928 for providing video encode/decode acceleration; an SRAM unit 930; a DMA unit 932; and a display unit 940 for coupling to one or more external displays.

**[00123]** Figure 10 illustrates a processor containing a Central Processing Unit (CPU) and a graphics processing unit (GPU), which may perform at least one instruction, in accordance with embodiments of the present disclosure. In one embodiment, an instruction to perform operations according to at least one embodiment could be performed by the CPU. In another embodiment, the instruction could be performed by the GPU. In still another embodiment, the instruction may be performed through a combination of operations performed by the GPU and the CPU. For example, in one embodiment, an instruction in accordance with one embodiment may be received and decoded for execution on the GPU. However, one or more operations within the decoded instruction may be performed by a CPU and the result returned to the GPU for final retirement of the instruction. Conversely, in some embodiments, the CPU may act as the primary processor and the GPU as the co-processor.

**[00124]** In some embodiments, instructions that benefit from highly parallel, throughput processors may be performed by the GPU, while instructions that benefit from the performance of processors that benefit from deeply pipelined architectures may be performed by the CPU. For example, graphics, scientific applications, financial applications and other parallel workloads may benefit from the

performance of the GPU and be executed accordingly, whereas more sequential applications, such as operating system kernel or application code may be better suited for the CPU.

**[00125]** In Figure 10, processor 1000 includes a CPU 1005, GPU 1010, image processor 1015, video processor 1020, USB controller 1025, UART controller 1030, SPI/SDIO controller 1035, display device 1040, memory interface controller 1045, MIPI controller 1050, flash memory controller 1055, Dual Data Rate (DDR) controller 1060, security engine 1065, and I2S/I2C controller 1070. Other logic and circuits may be included in the processor of Figure 10, including more CPUs or GPUs and other peripheral interface controllers.

**[00126]** One or more aspects of at least one embodiment may be implemented by representative data stored on a machine-readable medium which represents various logic within the processor, which when read by a machine causes the machine to fabricate logic to perform the techniques described herein. Such representations, known as "IP cores" may be stored on a tangible, machine-readable medium ("tape") and supplied to various customers or manufacturing facilities to load into the fabrication machines that actually make the logic or processor. For example, IP cores, such as the Cortex™ family of processors developed by ARM Holdings, Ltd. and Loongson IP cores developed the Institute of Computing Technology (ICT) of the Chinese Academy of Sciences may be licensed or sold to various customers or licensees, such as Texas Instruments, Qualcomm, Apple, or Samsung and implemented in processors produced by these customers or licensees.

**[00127]** Figure 11 illustrates a block diagram illustrating the development of IP cores, in accordance with embodiments of the present disclosure. Storage 1130 may include simulation software 1120 and/or hardware or software model 1110. In one embodiment, the data representing the IP core design may be provided to storage 1130 via memory 1140 (e.g., hard disk), wired connection (e.g., internet) 1150 or wireless connection 1160. The IP core information generated by the simulation tool and model may then be transmitted to a fabrication facility where it may be fabricated by a third party to perform at least one instruction in accordance with at least one embodiment.

**[00128]** In some embodiments, one or more instructions may correspond to a first type or architecture (e.g., x86) and be translated or emulated on a processor of a different type or architecture (e.g., ARM). An instruction, according to one embodiment, may therefore be performed on any processor or processor type, including ARM, x86, MIPS, a GPU, or other processor type or architecture.

**[00129]** Figure 12 illustrates how an instruction of a first type may be emulated by a processor of a different type, in accordance with embodiments of the present disclosure. In Figure 12, program 1205 contains some instructions that may perform the same or substantially the same function as an instruction according to one embodiment. However the instructions of program 1205 may be of a type and/or format that is different from or incompatible with processor 1215, meaning the instructions of the type in program 1205 may not be able to execute natively by the processor 1215. However, with the help of emulation logic, 1210, the instructions of program 1205 may be translated into instructions that may be natively be executed by the processor 1215. In one embodiment, the emulation logic may be embodied in hardware. In another embodiment, the emulation logic may be embodied in a tangible, machine-readable medium containing software to translate instructions of the type in program 1205 into the type natively executable by processor 1215. In other embodiments, emulation logic may be a combination of fixed-function or programmable hardware and a program stored on a tangible, machine-readable medium. In one embodiment, the processor contains the emulation logic, whereas in other embodiments, the emulation

logic exists outside of the processor and may be provided by a third party. In one embodiment, the processor may load the emulation logic embodied in a tangible, machine-readable medium containing software by executing microcode or firmware contained in or associated with the processor.

**[00130]** Figure 13 illustrates a block diagram contrasting the use of a software instruction converter to convert binary instructions in a source instruction set to binary instructions in a target instruction set, in accordance with embodiments of the present disclosure. In the illustrated embodiment, the instruction converter may be a software instruction converter, although the instruction converter may be implemented in software, firmware, hardware, or various combinations thereof. Figure 13 shows a program in a high level language 1302 may be compiled using an x86 compiler 1304 to generate x86 binary code 1306 that may be natively executed by a processor with at least one x86 instruction set core 1316. The processor with at least one x86 instruction set core 1316 represents any processor that may perform substantially the same functions as a Intel processor with at least one x86 instruction set core by compatibly executing or otherwise processing (1) a substantial portion of the instruction set of the Intel x86 instruction set core or (2) object code versions of applications or other software targeted to run on an Intel processor with at least one x86 instruction set core, in order to achieve substantially the same result as an Intel processor with at least one x86 instruction set core. x86 compiler 1304 represents a compiler that may be operable to generate x86 binary code 1306 (e.g., object code) that may, with or without additional linkage processing, be executed on the processor with at least one x86 instruction set core 1316. Similarly, Figure 13 shows the program in high level language 1302 may be compiled using an alternative instruction set compiler 1308 to generate alternative instruction set binary code 1310 that may be natively executed by a processor without at least one x86 instruction set core 1314 (e.g., a processor with cores that execute the MIPS instruction set of MIPS Technologies of Sunnyvale, CA and/or that execute the ARM instruction set of ARM Holdings of Sunnyvale, CA). Instruction converter 1312 may be used to convert x86 binary code 1306 into code that may be natively executed by the processor without an x86 instruction set core 1314. This converted code might not be the same as alternative instruction set binary code 1310; however, the converted code will accomplish the general operation and be made up of instructions from the alternative instruction set. Thus, instruction converter 1312 represents software, firmware, hardware, or a combination thereof that, through emulation, simulation or any other process, allows a processor or other electronic device that does not have an x86 instruction set processor or core to execute x86 binary code 1306.

**[00131]** Figure 14 is a block diagram of an instruction set architecture 1400 of a processor, in accordance with embodiments of the present disclosure. Instruction set architecture 1400 may include any suitable number or kind of components.

**[00132]** For example, instruction set architecture 1400 may include processing entities such as one or more cores 1406, 1407 and a graphics processing unit 1415. Cores 1406, 1407 may be communicatively coupled to the rest of instruction set architecture 1400 through any suitable mechanism, such as through a bus or cache. In one embodiment, cores 1406, 1407 may be communicatively coupled through an L2 cache control 1408, which may include a bus interface unit 1409 and an L2 cache 1410. Cores 1406, 1407 and graphics processing unit 1415 may be communicatively coupled to each other and to the remainder of instruction set architecture 1400 through interconnect 1410. In one embodiment, graphics processing unit 1415 may use a video code 1420 defining the manner in which particular video signals will be encoded and decoded for output.

**[00133]** Instruction set architecture 1400 may also include any number or kind of interfaces, controllers, or other mechanisms for interfacing or communicating with other portions of an electronic device or system. Such mechanisms may facilitate interaction with, for example, peripherals, communications devices, other processors, or memory. In the example of Figure 14, instruction set architecture 1400 may include an LCD video interface 1425, a Subscriber Interface Module (SIM) interface 1430, a boot ROM interface 1435, an SDRAM controller 1440, a flash controller 1445, and a Serial Peripheral Interface (SPI) master unit 1450. LCD video interface 1425 may provide output of video signals from, for example, GPU 1415 and through, for example, a Mobile Industry Processor Interface (MIPI) 1490 or a High-Definition Multimedia Interface (HDMI) 1495 to a display. Such a display may include, for example, an LCD. SIM interface 1430 may provide access to or from a SIM card or device. SDRAM controller 1440 may provide access to or from memory such as an SDRAM chip or module. Flash controller 1445 may provide access to or from memory such as flash memory or other instances of RAM. SPI master unit 1450 may provide access to or from communications modules, such as a Bluetooth module 1470, high-speed 3G modem 1475, global positioning system module 1480, or wireless module 1485 implementing a communications standard such as 802.11.

**[00134]** Figure 15 is a more detailed block diagram of an instruction set architecture 1500 of a processor, in accordance with embodiments of the present disclosure. Instruction architecture 1500 may implement one or more aspects of instruction set architecture 1400. Furthermore, instruction set architecture 1500 may illustrate modules and mechanisms for the execution of instructions within a processor.

**[00135]** Instruction architecture 1500 may include a memory system 1540 communicatively coupled to one or more execution entities 1565. Furthermore, instruction architecture 1500 may include a caching and bus interface unit such as unit 1510 communicatively coupled to execution entities 1565 and memory system 1540. In one embodiment, loading of instructions into execution entities 1564 may be performed by one or more stages of execution. Such stages may include, for example, instruction prefetch stage 1530, dual instruction decode stage 1550, register rename stage 155, issue stage 1560, and writeback stage 1570.

**[00136]** In one embodiment, memory system 1540 may include an executed instruction pointer 1580. Executed instruction pointer 1580 may store a value identifying the oldest, undispached instruction within a batch of instructions. The oldest instruction may correspond to the lowest Program Order (PO) value. A PO may include a unique number of an instruction. Such an instruction may be a single instruction within a thread represented by multiple strands. A PO may be used in ordering instructions to ensure correct execution semantics of code. A PO may be reconstructed by mechanisms such as evaluating increments to PO encoded in the instruction rather than an absolute value. Such a reconstructed PO may be known as an "RPO." Although a PO may be referenced herein, such a PO may be used interchangeably with an RPO. A strand may include a sequence of instructions that are data dependent upon each other. The strand may be separated from a larger sequence of instructions by, for example, compiler 1806 at compilation time. Hardware executing a strand may execute the instructions of a given strand in order according to PO of the various instructions. A thread may include multiple strands such that instructions of different strands may depend upon each other. A PO of a given strand may be the PO of the oldest instruction in the strand which has not yet been dispatched to execution from an issue stage. Accordingly, given a thread of multiple strands, each strand including instructions ordered by PO,

executed instruction pointer 1580 may store the oldest—illustrated by the lowest number—PO in the thread.

**[00137]** In another embodiment, memory system 1540 may include a retirement pointer 1582. Retirement pointer 1582 may store a value identifying the PO of the last retired instruction. Retirement pointer 1582 may be set by, for example, retirement unit 454. If no instructions have yet been retired, retirement pointer 1582 may include a null value.

**[00138]** Execution entities 1565 may include any suitable number and kind of mechanisms by which a processor may execute instructions. In the example of Figure 15, execution entities 1565 may include ALU/Multiplication Units (MUL) 1566, ALUs 1567, and Floating Point Units (FPU) 1568. In one embodiment, such entities may make use of information contained within a given address 1569. Execution entities 1565 in combination with stages 1530, 1550, 1555, 1560, 1570 may collectively form an execution unit.

**[00139]** Unit 1510 may be implemented in any suitable manner. In one embodiment, unit 1510 may perform cache control. In such an embodiment, unit 1510 may thus include a cache 1525. Cache 1525 may be implemented, in a further embodiment, as an L2 unified cache with any suitable size, such as zero, 128k, 256k, 512k, 1M, or 2M bytes of memory. In another, further embodiment, cache 1525 may be implemented in error-correcting code memory. In another embodiment, unit 1510 may perform bus interfacing to other portions of a processor or electronic device. In such an embodiment, unit 1510 may thus include a bus interface unit 1520 for communicating over an interconnect, intraprocessor bus, interprocessor bus, or other communication bus, port, or line. Bus interface unit 1520 may provide interfacing in order to perform, for example, generation of the memory and input/output addresses for the transfer of data between execution entities 1565 and the portions of a system external to instruction architecture 1500.

**[00140]** To further facilitate its functions, bus interface unit 1520 may include an interrupt control and distribution unit 1511 for generating interrupts and other communications to other portions of a processor or electronic device. In one embodiment, bus interface unit 1520 may include a snoop control unit 1512 that handles cache access and coherency for multiple processing cores. In a further embodiment, to provide such functionality, snoop control unit 1512 may include a cache-to-cache transfer unit that handles information exchanges between different caches. In another, further embodiment, snoop control unit 1512 may include one or more snoop filters 1514 that monitors the coherency of other caches (not shown) so that a cache controller, such as unit 1510, does not have to perform such monitoring directly. Unit 1510 may include any suitable number of timers 1515 for synchronizing the actions of instruction architecture 1500. Also, unit 1510 may include an AC port 1516.

**[00141]** Memory system 1540 may include any suitable number and kind of mechanisms for storing information for the processing needs of instruction architecture 1500. In one embodiment, memory system 1504 may include a load store unit 1530 for storing information such as buffers written to or read back from memory or registers. In another embodiment, memory system 1504 may include a translation lookaside buffer (TLB) 1545 that provides look-up of address values between physical and virtual addresses. In yet another embodiment, bus interface unit 1520 may include a Memory Management Unit (MMU) 1544 for facilitating access to virtual memory. In still yet another embodiment, memory system 1504 may include a prefetcher 1543 for requesting instructions from memory before such instructions are actually needed to be executed, in order to reduce latency.

**[00142]** The operation of instruction architecture 1500 to execute an instruction may be performed through different stages. For example, using unit 1510 instruction prefetch stage 1530 may access an instruction through prefetcher 1543. Instructions retrieved may be stored in instruction cache 1532. Prefetch stage 1530 may enable an option 1531 for fast-loop mode, wherein a series of instructions forming a loop that is small enough to fit within a given cache are executed. In one embodiment, such an execution may be performed without needing to access additional instructions from, for example, instruction cache 1532. Determination of what instructions to prefetch may be made by, for example, branch prediction unit 1535, which may access indications of execution in global history 1536, indications of target addresses 1537, or contents of a return stack 1538 to determine which of branches 1557 of code will be executed next. Such branches may be possibly prefetched as a result. Branches 1557 may be produced through other stages of operation as described below. Instruction prefetch stage 1530 may provide instructions as well as any predictions about future instructions to dual instruction decode stage.

**[00143]** Dual instruction decode stage 1550 may translate a received instruction into microcode-based instructions that may be executed. Dual instruction decode stage 1550 may simultaneously decode two instructions per clock cycle. Furthermore, dual instruction decode stage 1550 may pass its results to register rename stage 1555. In addition, dual instruction decode stage 1550 may determine any resulting branches from its decoding and eventual execution of the microcode. Such results may be input into branches 1557.

**[00144]** Register rename stage 1555 may translate references to virtual registers or other resources into references to physical registers or resources. Register rename stage 1555 may include indications of such mapping in a register pool 1556. Register rename stage 1555 may alter the instructions as received and send the result to issue stage 1560.

**[00145]** Issue stage 1560 may issue or dispatch commands to execution entities 1565. Such issuance may be performed in an out-of-order fashion. In one embodiment, multiple instructions may be held at issue stage 1560 before being executed. Issue stage 1560 may include an instruction queue 1561 for holding such multiple commands. Instructions may be issued by issue stage 1560 to a particular processing entity 1565 based upon any acceptable criteria, such as availability or suitability of resources for execution of a given instruction. In one embodiment, issue stage 1560 may reorder the instructions within instruction queue 1561 such that the first instructions received might not be the first instructions executed. Based upon the ordering of instruction queue 1561, additional branching information may be provided to branches 1557. Issue stage 1560 may pass instructions to executing entities 1565 for execution.

**[00146]** Upon execution, writeback stage 1570 may write data into registers, queues, or other structures of instruction set architecture 1500 to communicate the completion of a given command. Depending upon the order of instructions arranged in issue stage 1560, the operation of writeback stage 1570 may enable additional instructions to be executed. Performance of instruction set architecture 1500 may be monitored or debugged by trace unit 1575.

**[00147]** Figure 16 is a block diagram of an execution pipeline 1600 for an instruction set architecture of a processor, in accordance with embodiments of the present disclosure. Execution pipeline 1600 may illustrate operation of, for example, instruction architecture 1500 of Figure 15.

**[00148]** Execution pipeline 1600 may include any suitable combination of steps or operations. In 1605, predictions of the branch that is to be executed next may be made. In one embodiment, such predictions may be based upon previous executions of instructions and the results thereof. In 1610, instructions corresponding to the predicted branch of execution may be loaded into an instruction cache. In 1615, one or more such instructions in the instruction cache may be fetched for execution. In 1620, the instructions that have been fetched may be decoded into microcode or more specific machine language. In one embodiment, multiple instructions may be simultaneously decoded. In 1625, references to registers or other resources within the decoded instructions may be reassigned. For example, references to virtual registers may be replaced with references to corresponding physical registers. In 1630, the instructions may be dispatched to queues for execution. In 1640, the instructions may be executed. Such execution may be performed in any suitable manner. In 1650, the instructions may be issued to a suitable execution entity. The manner in which the instruction is executed may depend upon the specific entity executing the instruction. For example, at 1655, an ALU may perform arithmetic functions. The ALU may utilize a single clock cycle for its operation, as well as two shifters. In one embodiment, two ALUs may be employed, and thus two instructions may be executed at 1655. At 1660, a determination of a resulting branch may be made. A program counter may be used to designate the destination to which the branch will be made. 1660 may be executed within a single clock cycle. At 1665, floating point arithmetic may be performed by one or more FPUs. The floating point operation may require multiple clock cycles to execute, such as two to ten cycles. At 1670, multiplication and division operations may be performed. Such operations may be performed in four clock cycles. At 1675, loading and storing operations to registers or other portions of pipeline 1600 may be performed. The operations may include loading and storing addresses. Such operations may be performed in four clock cycles. At 1680, write-back operations may be performed as required by the resulting operations of 1655-1675.

**[00149]** Figure 17 is a block diagram of an electronic device 1700 for utilizing a processor 1710, in accordance with embodiments of the present disclosure. Electronic device 1700 may include, for example, a notebook, an ultrabook, a computer, a tower server, a rack server, a blade server, a laptop, a desktop, a tablet, a mobile device, a phone, an embedded computer, or any other suitable electronic device.

**[00150]** Electronic device 1700 may include processor 1710 communicatively coupled to any suitable number or kind of components, peripherals, modules, or devices. Such coupling may be accomplished by any suitable kind of bus or interface, such as I2C bus, System Management Bus (SMBus), Low Pin Count (LPC) bus, SPI, High Definition Audio (HDA) bus, Serial Advance Technology Attachment (SATA) bus, USB bus (versions 1, 2, 3), or Universal Asynchronous Receiver/Transmitter (UART) bus.

**[00151]** Such components may include, for example, a display 1724, a touch screen 1725, a touch pad 1730, a Near Field Communications (NFC) unit 1745, a sensor hub 1740, a thermal sensor 1746, an Express Chipset (EC) 1735, a Trusted Platform Module (TPM) 1738, BIOS/firmware/flash memory 1722, a DSP 1760, a drive 1720 such as a Solid State Disk (SSD) or a Hard Disk Drive (HDD), a wireless local area network (WLAN) unit 1750, a Bluetooth unit 1752, a Wireless Wide Area Network (WWAN) unit 1756, a Global Positioning System (GPS), a camera 1754 such as a USB 3.0 camera, or a Low Power Double Data Rate (LPDDR) memory unit 1715 implemented in, for example, the LPDDR3 standard. These components may each be implemented in any suitable manner.

**[00152]** Furthermore, in various embodiments other components may be communicatively coupled to processor 1710 through the components discussed above. For example, an accelerometer 1741, Ambient Light Sensor (ALS) 1742, compass 1743, and gyroscope 1744 may be communicatively coupled to sensor hub 1740. A thermal sensor 1739, fan 1737, keyboard 1746, and touch pad 1730 may be communicatively coupled to EC 1735. Speaker 1763, headphones 1764, and a microphone 1765 may be communicatively coupled to an audio unit 1764, which may in turn be communicatively coupled to DSP 1760. Audio unit 1764 may include, for example, an audio codec and a class D amplifier. A SIM card 1757 may be communicatively coupled to WWAN unit 1756. Components such as WLAN unit 1750 and Bluetooth unit 1752, as well as WWAN unit 1756 may be implemented in a Next Generation Form Factor (NGFF).

#### INTER-STRAND COMMUNICATION

**[00153]** Figure 18 illustrates an example of a system 1800 for implementing inter-strand communication, in accordance with embodiments of the present disclosure. In one embodiment, such inter-strand communication may be performed with a shared register file which may be a physical register file. In another embodiment, such inter-strand communication may be facilitated through particular indications or denotations in instructions to be executed by system 1800. The denotations may be added to code in strands that are split off from an original sequence of instructions, and to code in the original strand that is to be coordinated. In a further embodiment, the denotations may include markers that register references in a given strand to registers that are shared between the strands. Execution of the two strands may be synchronized by reference to tracking mechanisms of such denotations. Reservation station entries may be shared by denotations in both strands, such as GLOBDEF and SYNCREG instructions. Furthermore, a definition queue may be used to track arrival of respective GLOBDEF and SYNCREG instructions. The entries may be populated with a reservation station identifier and a label for the physical register associated with the first of GLOBDEF or SYNCREG to arrive. The entry may be used to assign the same reservation station identifier and label for the physical register to the second of GLOBDEF or SYNCREG to arrive. Thus, the two instructions may share the same physical register to logical register. This may facilitate sharing of data through such a physical register between the strands by subsequent instructions.

**[00154]** System 1800 may fetch, dispatch, execute, and retire instructions, such as those in instruction stream 1804, out-of-order. System 1800 may utilize any suitable number or kind of out-of-order processors or processing entities. System 1800 may illustrate elements of such a processor, which may include any processor core, logical processor, processor, or other processing entity such as those illustrated in Figures 1-17.

**[00155]** System 1800 may be implemented in any suitable manner to perform inter-strand communication while executing instruction stream 1804. System 1800 may include a front end 1812 communicatively coupled to one or more execution units 1822 and an out-of-order execution engine 1824. Front end 1812 may process instructions so that they may be executed by execution units 1822 and out-of-order execution engine 1824. System 1800 may include as many execution units 1822 and out-of-order execution engines 1824 as necessary. Front end 1812 may be implemented in part by, for example, front end 201, front end unit 430, or front end 570. Out-of-order execution engine 1824 may be implemented in part by, for example, out-of-order execution engine 203, execution engine unit 450, or out

of order engine 580. Front end 1812 may include decoder 1816 to decode and interpret instructions from instruction stream 1804, which may indicate what operation should be performed on the corresponding data indicated within the instruction operands and data to be applied. Front end 1812 may send instructions to execution units through reservation station 1836.

**[00156]** System 1800 may execute instructions in any suitable number of strands. A single sequence of instructions may be divided into separate sequences of instructions, or strands, that may be executed in parallel on one or more cores, execution units, or processors. In one embodiment, multiple strands may be present in instruction stream 1804 as they are to be executed by processor 1802. Front end 1812 may switch between strands, fetching and decoding instructions from respective strands. In some cases, processor 1802 may include multiple front ends 1812 (such as for multiple cores), and in such cases multiple front ends may each handle a given strand.

**[00157]** Out-of-order execution engine 1824 may include any suitable number or kind of elements. For example, out-of-order execution engine 1824 may include schedulers, allocators, retirement units, reorder buffers, and register files. In one embodiment, out-of-order execution engine 1824 may include mechanisms for renaming logical registers in the instructions and assigning these to physical registers. For example, out-of-order execution engine 1824 may include renaming logic 1826 which may manage assignment of logical registers to physical registers. Furthermore, renaming logic 1826 may manage inter-strand communication via a shared register file. In various embodiments, processor 1802 may include a core that may include out-of-order execution engine 1824, execution units 1822, and front end 1812.

**[00158]** In various embodiments out-of-order execution engine 1824 may include a freelist indicating physical registers that are available for assignment to logical registers, a Rename Renaming Table (RRT) or Register Alias Table (RAT) for each strand, such as Strand0 RRT 1828 and Strand1 RRT 1830, as well as a reservation station 1836. Reservation station 1836 may track dynamic scheduling of instructions for register renaming. Instructions from instruction stream 1804 may be issued and designate entries in reservation station 1836 that are to be used for accessing the results of their execution. Instructions are executed as they are ready, when operands and inputs are available. Instructions may be issued to reservation station 1836, which may buffer the instruction and its operands and inputs. When all operands and inputs are available, execution of the instruction may begin.

**[00159]** Register renaming data structures, such as those described above, may include mappings of logical resources to physical resources. The resources may include registers, memory locations, or registers implemented as memory locations. By mapping logical resources to physical resources, these may facilitate out-of-order processing. Within instruction stream 1804, repeated references may be made to a given logical resource such as a register. However, different references to and operations of the same logical resource may not depend upon each other. In such circumstances, uses of the same logical resource may be performed out-of-order, in parallel, or in any other non-sequential manner. To allow multiple, perhaps simultaneous use of the same logical resource, different uses of the logical resource in instruction stream 1804 may actually be made to different physical resources. The mappings of multiple instances of a single logical resource to different physical resources may be stored in register renaming data structures. As used herein, a logical resource may be described, for example, by an identifiable register name and then an associated physical resource may be described. For example, instances of EAX registers may include "EAX {10}" or "EAX {40}", wherein certain references to EAX within instruction

stream 1804 may be redirected to physical location "10" and other references to EAX may be redirected to physical location "40".

**[00160]** Register renaming data structures may be populated as necessary by, for example, an allocator based upon instant needs to process instruction stream 1804 out-of-order. The allocator may be implemented fully or in part by, for example, rename/allocator unit 452, allocation stage 408, allocate 582, out of order engine 580, register rename stage 1555, issue stage 1560, issue stage 1560, execution engine unit 450, or out of order engine 203.

**[00161]** Furthermore, system 1800 may reclaim entries in register renaming data structures when they are no longer needed. System 1800 may include retirement units to perform such reclamation, implemented fully or in part by, for example by write back/memory write 418, execution engine unit 450, retirement unit 454, write back stage 1570, write back stage 1680, out of order engine 580, or out of order engine 203. For example, after no more references to "EAX {10}" are to be made, use of the mapping between "EAX" and "{10}" may be unnecessary.

**[00162]** The RATs may be implemented in any suitable manner such as by a table, array, or other suitable data structure. RATs 1828, 1830 may implement some of the functionality of register files, such as the mapping or aliasing between physical and logical resources. These may track physical registers that need to be reclaimed once their associated instructions are committed and retired. Once an instruction has been retired, resources identified in RRT 1832 with the instruction may be removed from RATs and then identified in a freelist so that the resource may be assigned to other virtual resources.

**[00163]** In one embodiment, out-of-order execution engine 1824 may include a definition queue (DQ) for tracking communication between strands. The communication may be indicated by denotations in instruction stream 1804 that certain operations upon a logical register by each strand are to be performed upon the same physical resource mapped to the logical register. Management of DQ 1832 may be performed by renaming logic 1826. Renaming logic 1826 may be implemented by, for example, digital circuitry. If an instruction in instruction stream 1804 is identified as associated with an instruction in another strand, the connection between these instructions in different strands may be tracked in DQ 1832. DQ 1832 may be implemented in any suitable manner, such as by a circular queue or other suitable data structure.

**[00164]** The denotations in instruction stream 1804 that a given instruction in a strand is to communicate or be coordinated with an instruction in another strand may be added or implemented in any suitable manner. In one embodiment, a compiler 1806 may modify instruction stream 1804 to include denotations that instructions between strands are to be coordinated. Compiler 1806 may be implemented within system 1800, within processor 1802, or in another system. Such modifications may be performed before instructions are received at processor 1802 or during processing by processor 1802. In another embodiment, a binary translator 1814 or other processing mechanism may amend instruction stream 1804 to include such denotations. The denotations may be added to programming code, machine-level instructions, decoded instructions, or to any other suitable form of instructions that are to be executed by system 1800. The result of adding denotations to instruction stream 1804 may be modified instructions 1810. As instruction stream 1804 may be divided into multiple strands, modified instructions 1810 may be modified multi-strand instructions 1810. Some of these instructions (belonging to any of the strands) may contain the denotations described herein. In various embodiments, denotations may be added to

instruction stream 1804 in conjunction with determinations of what instructions will be executed in various strands.

**[00165]** For example, compiler 1806, binary translator 1814, or another suitable portion of system 1800 may determine that instruction stream 1804 is to be divided into two or more strands of sequential instructions for execution by one or more execution units 1822. In one embodiment, system 1800 may allow instructions dependent upon the same logical-physical resource mapping to be placed into separate strands for execution. In another embodiment, system 1800 may allow instructions placed into separate strands for execution to communicate. Such communication may include indications of use of given logical-physical resource pairs.

**[00166]** In one embodiment, given a definition of a logical resource in one such strand, wherein another strand will use the defined logical resource, instruction stream 1804 may be amended with a denotation to be placed in the original strand that other strands are to use the same definition of the logical resource. In another embodiment, given the definition of the logical resource in the original, wherein another strand will use the defined logical resource, instruction stream 1804 may be amended with a denotation to be placed in the other strand that the original strand is to define the logical resource. The denotations may be implemented in any suitable manner. For example, new instructions may be inserted into instruction stream to make such denotations. In one embodiment, these new instructions may replace previous instructions.

**[00167]** In the example of Figure 18, these instructions may include GLOBDEF and SYNCREG. GLOBDEF may accept as a parameter a logical resource that is to be defined in a given strand. In one embodiment, GLOBDEF is placed in the strand that produces a register value. GLOBDEF may replace other instructions that are to define a logical resource. In one embodiment, GLOBDEF may perform the replaced instruction's action to define the logical register as associated with the physical resource. In another embodiment, GLOBDEF may signal to renaming logic 1826 that communication with the other strands may be performed with respect to the instance of the logical register. Such communication may be performed to synchronize use of the logical register until another definition of the logical register is made. GLOBDEF may be placed within the original, given strand.

**[00168]** SYNCREG may accept as a parameter a logical resource that is to be defined by GLOBDEF and used within the other strand. SYNCREG may be placed within the other strand, in which the value of the register is consumed. In one embodiment, SYNCREG may signal to renaming logic 1826 that communication with the other strands may be performed with respect to the instance of the logical register. Such communication may be performed to synchronize use of the logical register until another definition of the logical register is made. Use of SYNCREG may signal to renaming logic 1826 that a definition of the logical register has been made, to which execution may be coordinated, has been or is to be made on the original strand. In one embodiment, execution between the strands may be coordinated such that execution in respective strands is made for the same physical-logical resource mapping for certain instructions between the strands.

**[00169]** In other embodiments, GLOBDEF and SYNCREG may be implemented by flags, parameters, bits, or other aspects of existing instructions. For example, a definition instruction may include a flag to specify whether the instruction is to be handled as a GLOBDEF.

**[00170]** The GLOBDEF and SYNCREG instructions may be decoded, dispatched and then executed. After execution, they may be retired.

**[00171]** In one embodiment, an instance of a definition instruction may be replaced in the original strand by GLOBDEF. In another embodiment, SYNCREG may be added to the other strand before instances of instructions using the logical resource.

**[00172]** Figure 19 illustrates example operation of system 1800 for inter-strand communication via a shared register file, in accordance with embodiments of the present disclosure. An original segment of code 1902 may include a sequence of instructions, four of which are illustrated. Other instructions may be present before, between, and after such illustrated instructions. A first definition instruction may define, produce, or write to an instance of the logical register EAX. For such a write, the logical register EAX may be assigned a physical register. The definition instruction may include, for example, a write or save operation to the EAX register. The definition instruction may be followed by a use of the defined instance of logical register EAX, which may include a read. Then, a subsequent definition of the logical register EAX may be made, wherein the logical register EAX is assigned another physical register, followed by a subsequent use of this second definition. The definition of different physical resources to the different uses of the EAX register may allow the uses to be performed in parallel or out-of-order.

**[00173]** Code segment 1902 may be split into multiple strands for execution in parallel. For example, code segment 1902 may result in strand code 1904, wherein code segment 1902 has been divided into two strands for execution by one or more cores of processor 1802. The strands may be denoted as Strand0 and Strand1. Strand0 may be to execute the definition instructions, while Strand1 may be to execute the use instructions.

**[00174]** Register renaming may be performed upon the code. Code 1906 illustrates correct register renaming upon strand code 1904. The first definition instruction of EAX results in assignment of physical register R10. The second definition instruction of EAX results in assignment of physical register R20. If the first use instruction (in Strand1) occurs after the first definition (in Strand0), but before the second definition (in Strand0), then the first use instruction will have the correct physical register, R10. If the second use instruction (in Strand1) occurs after the second definition (in Strand0), then the second use instruction will have the correct physical register, R20. However, this example, without coordination between strands, may have performed correctly simply by luck that the timing between the strands worked such that no conflicts in register renaming occurred. The sequential nature of the instructions in Strand0 and the sequential nature of instructions in Strand1 provide no assurances of correct execution between the strands.

**[00175]** In another example, such as that illustrated by code 1908 may illustrate incorrect register renaming as a result of a lack of coordination between the strands. The first definition instruction of EAX results in assignment of physical register R10 in Strand0. The first use of EAX in Strand1 uses the correct physical register, R10. However, if the second use of EAX in Strand1 happens before the second definition of EAX in Strand0, R10 may be incorrectly used, rather than R20.

**[00176]** Code 1910 illustrates sequential code that may be separated into strands and marked for inter-strand communication by system 1800. Four combinations of definitions and uses of EAX may be included in code 1910. In the first combination, the definition of EAX may be in Strand0 while the use of EAX may be in Strand1. These two instructions are to have the same physical resource assignment to

the logical register EAX. This physical resource may, at the time of execution, be R1, although this might not be known at the time that code is divided into strands. In one embodiment, denotations may be made to instruction stream 1804 for such a combination so that their execution may be coordinated and the strands may communicate. In the second combination, both the uses and definition of EAX may be within Strand1. Accordingly, in one embodiment, denotations might not be made to instruction stream 1804 for such a combination. Likewise, in the third combination, both the uses and definitions of EAX may be within Strand0. Accordingly, in one embodiment, denotations might not be made to instruction stream 1804 for such a combination. In the fourth combination, the definition of EAX may be in Strand0 while the use of EAX may be in Strand1. These two instructions are to have the same physical resource assignment to the logical register EAX. This physical resource may, at the time of execution, be R4, although this might not be known at the time that code is divided into strands. In one embodiment, denotations may be made to instruction stream 1804 for such a combination so that their execution may be coordinated and the strands may communicate.

**[00177]** Code 1912 illustrates how code 1910 is divided into two strands and how code 1910 may be modified for inter-strand communication. In one embodiment, the first instance of defining EAX may be replaced in Strand0 with a GLOBDEF to both define EAX and to signal that its mapping is to be shared with the other strand. In another embodiment, a SYNCREG may be inserted in Strand1 before the first instance of using EAX. This may signal that the mapping for EAX is to be the same as that in corresponding segments of Strand0. The mapping may be that EAX is assigned to a physical resource R1. Renaming logic 1826 may identify the use of GLOBDEF and SYNCREG in the strands and, in one embodiment, stall execution of uses of EAX in Strand1 until the assignment of EAX to R1 is made in Strand0.

**[00178]** The second and third combinations of definitions and uses may be placed within Strand0 and Strand1 as planned.

**[00179]** In one embodiment, the fourth instance of defining EAX may be replaced in Strand0 with a GLOBDEF to both define EAX and to signal that its mapping is to be shared with the other strand. In another embodiment, a SYNCREG may be inserted in Strand1 before the fourth instance of using EAX. This may signal that the mapping for EAX is to be the same as that in corresponding segments of Strand0. The mapping may be that EAX is assigned to a physical resource R4. Renaming logic 1826 may identify the use of GLOBDEF and SYNCREG in the strands and, in one embodiment, stall execution of uses of EAX in Strand1 until the assignment of EAX to R4 is made in Strand0.

**[00180]** The coordination of execution by renaming logic 1826 may be performed in any suitable manner. Figure 20 illustrates a more detailed block diagram of a system for implementing inter-strand communication via a shared register file, in accordance with embodiments of the present disclosure. Figure 20 illustrates example embodiments for implementing the coordination of execution by renaming logic 1826.

**[00181]** For example, as SYNCREG or GLOBDEF instructions are executed, denotations of the associated physical resource may be placed within DQ 1832. In one embodiment, the physical resource may be assigned to the logical resource upon the first instance of either SYNCREG or GLOBDEF and entered into DQ 1832. In the example of Figure 20, a first mapping of logical resources to R1 may have been made by a SYNCREG-GLOBDEF pair, and a second mapping of logical resources to R4 may have been made by a second SYNCREG-GLOBDEF pair.

**[00182]** DQ 1832 may include tracking mechanisms for indicating whether a given strand has encountered one of a SYNCREG-GLOBDEF pair. In one embodiment, tracking of which strand has first encountered either a SYNCREG or GLOBDEF of linked pair may be performed by pointers in DQ 1832. DQ 1832 may include pointers for each respective strand, such as Strand0Ptr 2002 and Strand1Ptr 2004. Upon receipt of a SYNCREG or GLOBDEF at either Strand0 or Strand1, the associated physical resource may be entered to the entry in DQ 1832 pointed to by the corresponding pointer, if necessary, and the associated strand pointer will be incremented. Upon receipt of the other of SYNCREG or GLOBDEF at the other strand, DQ 1832 might already include the mapped physical resource and so the respective strand pointer will be incremented. Thus, through the use of the strand pointers, DQ 1832 may track whether a SYNCREG or GLOBDEF is expected based upon a past SYNCREG or GLOBDEF instruction. Moreover, in one embodiment if an expected GLOBDEF has not yet arrived, execution of instructions using the logical register that follow the previously received SYNCREG will be stalled until the expected GLOBDEF arrives. At such a point, the instructions using the logical register that follow the previously received SYNCREG may be safely executed.

**[00183]** The contents of Strand0 RRT 1828 and Strand1 RRT 1830 may change depending upon the current GLOBDEF or SYNCREG instructions. The mapping of logical register EAX may be to a physical register value as determined through received GLOBDEF or SYNCREG instructions, as illustrated in further detail below.

**[00184]** Figures 21A-21E illustrate further example operation of a system for implementing inter-strand communication via a shared register, in accordance with embodiments of the present disclosure.

**[00185]** At Figure 21A, a GLOBDEF instruction for the EAX register may be received for execution from Strand0. Strand0Ptr 2002 and Strand1Ptr 2004 may be pointing to the same value in DQ 1832, which may be effectively empty before the arrival of the first instruction. In one embodiment, an entry for the instruction which transfer the EAX register may be added to DQ 1832. The value for the entry may be based upon the assigned physical resource. For example, R1, in a position 'loc1' in a register freelist (RFL) 2006 may be available and assigned to this instance of EAX. The physical resource name and its reservation station identifier (RSID) may be written to DQ 1832. Moreover, Strand0Ptr 2002 may be incremented. In one embodiment, this may indicate that system 1800 is awaiting a SYNCREG corresponding to the GLOBDEF. Strand0 RRT 1830 may be updated with the logical-physical mapping. An entry in reservation station 1836 may be entered.

**[00186]** In one embodiment, because the association of EAX and R1 has been properly initiated by the GLOBDEF, this association may be used by subsequent execution, whether in Strand0 or Strand1. The association and safe execution may be enabled by a denotation in reservation station 1836 that the physical resource mapping is ready to use. Otherwise, the physical resource might be identified in reservation station 1836 as not ready to use and subsequent execution using the physical resource might be stalled.

**[00187]** At Figure 21B, the corresponding SYNCREG might be received at Strand1. Strand1Ptr 2004 may be incremented. In one embodiment, as the values between the pointers are the same, the value stored in DQ 1832 may be popped and stored in Strand1 RRT 1830.

**[00188]** At Figure 21C, other instructions not associated with SYNCREG or GLOBDEF may be received and executed in an otherwise normal manner, except that reservation station 1836 may be

checked the physical register readiness. Such a condition might exist when, for example, execution is to be made of a physical register identified in reservation station 1836 as not ready for execution.

Otherwise, necessary physical registers may be obtained and RRTs may be populated as normal.

**[00189]** At Figure 21D, a SYNCREG for EAX may be received. The corresponding GLOBDEF might not yet have been received. This situation, without intervention from system 1800, might otherwise result in the problem illustrated by code 1908, discussed above. However, as system 1800 provides inter-strand communication, such a problem may be avoided.

**[00190]** Strand1Ptr 2004 may be incremented. This may indicate that the received SYNCREG has arrived before the corresponding GLOBDEF. In one embodiment, the name of a free physical register, such as R4, and its RSID in RFL 2006 may be obtained and inserted into DQ 1832. The information of the physical register may be provided to Strand1 RRT 1830. In one embodiment, the information may be entered into reservation station 1836. In a further embodiment, the readiness of the physical register to be used by subsequent execution may be set to "NOT READY." Subsequent attempted use of the physical register, such as in other instructions in Strand1, may be stalled until the corresponding GLOBDEF is received.

**[00191]** At Figure 21E, the corresponding GLOBDEF may be received. Strand0Ptr 2002 may be incremented. As this pointer was behind the pointer for Strand1, renaming logic 1826 may determine that the GLOBDEF corresponds to a previously received SYNCREG. The entry in reservation station 1836 for the GLOBDEF and physical register for the logical EAX register were assigned while SYNCREG was being renamed, and the results of this assignment are located in the entry in DQ 1832 entry pointed to by Strand0Ptr 2002. The value from DQ 1832 may be popped to Strand0 RRT 1828. The entry allocated by the SYNCREG for the GLOBDEF in reservation station 1836 and whose RSID number was communicated from the SYNCREG to the GLOBDEF through DQ 1832 may change its status from NOT READY to READY depending on the status of GLOBDEF source operands.

**[00192]** Figure 22 is an illustration of an example method 2200 for inter-strand communication via a shared register, in accordance with embodiments of the present disclosure. Method 2200 may begin at any suitable point and may execute in any suitable order. In one embodiment, method 2200 may begin at 2005. In various embodiments, method 2200 may be performed during the execution of an out-of-order processor.

**[00193]** At 2205, code to be executed may be divided into strands to be executed in parallel with each other. In one embodiment, code resident on different strands that are to share the same physical resource mappings to logical registers may be identified. It may be necessary for these to have the same physical resource mapping in order to transfer data produced by an instruction in one strand to an instruction in the other strand. To enable such communication, both strands might need to share the register mapping to the same physical resource. For a given set of instructions that will share the same physical resource, in another embodiment, instructions for the definition of such a register may be replaced or appended with an instruction for defining the register and signaling the start of inter-strand communication for the set of instructions. This instruction may be denoted as GLOBDEF and may be placed in a strand in which the instruction producing the register value appeared. In yet another embodiment, an instruction may be inserted into the other strands of code before the first execution using the same assigned physical register. This instruction may be denoted as SYNCREG. 2205 may be repeated and performed in parallel with other steps of method 2200.

**[00194]** At 2210, strands for execution may be dispatched to one or more cores for execution. 2210 may be repeated and performed in parallel with other steps of method 2200. 2215-2270 may be repeated for each strand as long as instructions are to be executed. Moreover, 2215-2270 may be repeated for each strand in parallel with other instances of 2215-2270 for other strands.

**[00195]** At 2215, at a given strand in one embodiment it may be determined whether an instruction to be executed is a GLOBDEF or SYNCREG instruction. If not, if the instruction is to use a physical register that is associated with stalled execution, such as due to an unarrived GLOBDEF instruction, the instruction may still be renamed. Otherwise, the instruction may be executed in an otherwise normal manner. For example, renaming of any necessary physical registers may be performed, the physical register may be obtained from a free list, a reservation station ID may be obtained from a free list, and an identifier of the physical register may be written to an RRT at an index for the associated logical register. Method 2200 may repeat as necessary. Otherwise, if the instruction is a GLOBDEF or SYNCREG instruction, method 2200 may proceed to 2225.

**[00196]** At 2225, it may be determined whether the given strand's pointer in the DQ is greater than or equal to the other strand's pointer in the DQ. In one embodiment, this may signify whether the given strand is waiting on a GLOBDEF or SYNCREG because the counterpart was already received at the other strand. If the given strand's pointer is great than or equal to the other strand's pointer, method 2200 may proceed to 2240. Otherwise, method 2200 may proceed to 2230.

**[00197]** At 2230, it may be determined that the counterpart to the instruction has already arrived at the other strand. Accordingly, the physical register and reservation station ID (RSID) might have already been determined for the received instruction.

**[00198]** At 2235, an entry at the strand's pointer in the DQ may be popped and used for register renaming. It may be used to populate the strand's RRT. Moreover, the value may be used to populate the RRT at an entry indexed by the logical register. The RSID may be obtained from the entry in the DQ and, if necessary, the entry allocated by the SYNCREG for the GLOBDEF in the reservation station and whose RSID number was communicated from the SYNCREG to the GLOBDEF through the DQ may change its status from NOT READY to READY depending on the status of GLOBDEF source operands. The instruction may be executed. Subsequent execution using the physical register may be allowed. Method 2200 may proceed to 2250.

**[00199]** At 2240, it may be determined that the instruction is a first of a GLOBDEF-SYNCREG pair to arrive. At 2245, in one embodiment the DQ may be populated with a new entry. The physical register may be obtained from a free list. Moreover, an indicator of the physical register may be written to an RRT. An RSID may be obtained from a free list. An indicator of the physical register and the corresponding RSID may be written to the DQ entry. In one embodiment, the entry may be indexed at the strand's DQ pointer. The physical entry indicator may be written to the RRT at an index for the logical register associated with the received instruction.

**[00200]** At 2250, in one embodiment the strand's DQ pointer may be incremented. The logical destination may be renamed to the physical register.

**[00201]** At 2255, in one embodiment, it may be determined if the instruction is a SYNCREG. Such a determination may be used to determine whether the instruction should be stalled until the corresponding

GLOBDEF arrives. If the instruction is a SYNCREG, method 2200 may proceed to 2265. Otherwise, method 2200 may proceed to 2260.

**[00202]** At 2260, the instruction is a GLOBDEF. The received instruction and physical register may be written to the reservation station at the obtained RSID as the index. The readiness may be set as ready, assuming that other factors, beyond the scope of this disclosure, do not prevent the GLOBDEF instruction from dispatching for execution. Method 2200 may repeat.

**[00203]** At 2265, it may be determined whether the RSID and physical determinations were obtained from freelists in 2245. In one embodiment, this decision may be a reference to the previous decision made at 2225, to whether the given strand's pointer in the DQ is greater than or equal to the other strand's pointer in the DQ, signifying whether the given strand is waiting on a GLOBDEF or SYNCREG because the counterpart was already received at the other strand. If so, in another embodiment it may need to be denoted in the reservation station that the SYNCREG was received ahead of the GLOBDEF, and thus execution should be stalled until the GLOBDEF is received. If so, in yet another embodiment the reservation station entry may be marked as "NOT READY" in 2270. The reservation station may be accessed at an entry indexed by the obtained RSID. Method 2200 may repeat.

**[00204]** Method 2200 may be initiated by any suitable criteria. Furthermore, although method 2200 describes an operation of particular elements, method 2200 may be performed by any suitable combination or type of elements. For example, method 2200 may be implemented by the elements illustrated in Figures 1-21 or any other system operable to implement method 2200. As such, the preferred initialization point for method 2200 and the order of the elements comprising method 2200 may depend on the implementation chosen. In some embodiments, some elements may be optionally omitted, reorganized, repeated, or combined.

**[00205]** Embodiments of the mechanisms disclosed herein may be implemented in hardware, software, firmware, or a combination of such implementation approaches. Embodiments of the disclosure may be implemented as computer programs or program code executing on programmable systems comprising at least one processor, a storage system (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device.

**[00206]** Program code may be applied to input instructions to perform the functions described herein and generate output information. The output information may be applied to one or more output devices, in known fashion. For purposes of this application, a processing system may include any system that has a processor, such as, for example; a digital signal processor (DSP), a microcontroller, an application specific integrated circuit (ASIC), or a microprocessor.

**[00207]** The program code may be implemented in a high level procedural or object oriented programming language to communicate with a processing system. The program code may also be implemented in assembly or machine language, if desired. In fact, the mechanisms described herein are not limited in scope to any particular programming language. In any case, the language may be a compiled or interpreted language.

**[00208]** One or more aspects of at least one embodiment may be implemented by representative instructions stored on a machine-readable medium which represents various logic within the processor, which when read by a machine causes the machine to fabricate logic to perform the techniques described herein. Such representations, known as "IP cores" may be stored on a tangible, machine-readable

medium and supplied to various customers or manufacturing facilities to load into the fabrication machines that actually make the logic or processor.

**[00209]** Such machine-readable storage media may include, without limitation, non-transitory, tangible arrangements of articles manufactured or formed by a machine or device, including storage media such as hard disks, any other type of disk including floppy disks, optical disks, Compact Disk Read-Only Memories (CD-ROMs), Compact Disk Rewritables (CD-RWs), and magneto-optical disks, semiconductor devices such as Read-Only Memories (ROMs), Random Access Memories (RAMs) such as Dynamic Random Access Memories (DRAMs), Static Random Access Memories (SRAMs), Erasable Programmable Read-Only Memories (EPROMs), flash memories, Electrically Erasable Programmable Read-Only Memories (EEPROMs), magnetic or optical cards, or any other type of media suitable for storing electronic instructions.

**[00210]** Accordingly, embodiments of the disclosure may also include non-transitory, tangible machine-readable media containing instructions or containing design data, such as Hardware Description Language (HDL), which defines structures, circuits, apparatuses, processors and/or system features described herein. Such embodiments may also be referred to as program products.

**[00211]** In some cases, an instruction converter may be used to convert an instruction from a source instruction set to a target instruction set. For example, the instruction converter may translate (e.g., using static binary translation, dynamic binary translation including dynamic compilation), morph, emulate, or otherwise convert an instruction to one or more other instructions to be processed by the core. The instruction converter may be implemented in software, hardware, firmware, or a combination thereof. The instruction converter may be on processor, off processor, or part-on and part-off processor.

#### APPARATUS AND METHOD FOR IMPROVING THE PERFORMANCE OF INTER-STRAND COMMUNICATION

**[00212]** The techniques described above for enabling communication between strands are performed in a two-way SMT processor core through a common register file. In this implementation, the core may report a branch miss-prediction back to the front end of the pipeline in the form of a command such as “jump-execution-clear” (JEClear). Each JEClear causes the front-end pipeline to restart the in-order stream of logical micro-ops (fetch) in the current strand and also in other strands in case it has already consumed the wrong register value from the definition queue (DQ) produced by the jecleared strand.

**[00213]** The embodiments of the invention described below includes techniques for reducing the number of front-end pipeline restarts caused by a JEClear signal (or other command to indicate a miss-prediction), from the strand-producer feeding a potentially wrong value via DQ to a Register File.

**[00214]** Figure 23 illustrates one embodiment of a processor 2300 which includes circuitry and/or logic for reducing the number of front-end 2312 restarts caused by JEClear signals and other miss-prediction signals. In particular, the disclosed embodiment takes advantage of the fact that, in most cases, front-end 2312 restarting may be skipped. In one embodiment, each SYNCREG instruction 2308 may be supplied with an additional flushing bit (FB) 2307 by the compiler, denoting whether the front-end pipeline 2312 should be restarted (or not) by the JEClear signal from the other strand. In one embodiment, the hardware may be implemented substantially as described above, but with the following additions. First, in one embodiment, each Definition Queue (DQ) 2326 entry is extended by at least one

more field 2390-2393 to hold some unique identifier. For example, in one embodiment, the architectural register number (ARN) may be used for this purpose. It should be noted, however, that the underlying principles of the invention are not limited to use of the ARN; any suitable identifier may be used.

**[00215]** As in the embodiments described above, in one embodiment, inter-strand communication may be facilitated through particular indications or denotations in instructions to be executed by processor 2300. The denotations may be added to code in strands that are split off from an original sequence of instructions, and to code in the original strand that is to be coordinated. Execution of the two strands may be synchronized by reference to tracking mechanisms of such denotations. Reservation station 2327 entries may be shared by denotations in both strands, such as GLOBDEF and SYNCREG instructions and the definition queue (DQ) 2326 may track arrival of respective GLOBDEF and SYNCREG instructions. The entries may be populated with a reservation station identifier and a label for the physical register associated with the first of GLOBDEF or SYNCREG to arrive. The entry may be used to assign the same reservation station identifier and label for the physical register to the second of GLOBDEF or SYNCREG to arrive. Thus, the two instructions may share the same physical register to logical register to facilitate sharing of data through such a physical register between the strands by subsequent instructions.

**[00216]** Processor 2300 may be implemented in any suitable manner to perform inter-strand communication while executing instruction stream 2304, using a front end 2312 communicatively coupled to one or more execution units 2322 and an out-of-order execution engine 2312. A fetch unit 2312 fetches instructions 2304 from system memory and a decoder 2316 decodes and interprets the instructions to indicate what operation should be performed on the corresponding data indicated within the instruction operands and data to be applied. Front end 2312 may send instructions to execution units 2322 through reservation station 2327.

**[00217]** Out-of-order execution engine 2312 may include a renamer 2326 which may manage assignments of logical registers to physical registers. Furthermore, renamer 2326 may manage inter-strand communication via a shared register file. In various embodiments, processor 2300 may include multiple cores, each of which may include an out-of-order execution engine 2312, execution units 2322, and front end 2312.

**[00218]** In one embodiment, out-of-order execution engine 2312 may include a freelist indicating physical registers that are available for assignment to logical registers, a Rename Renaming Table (RRT) or Register Alias Table (RAT) for each strand, such as Strand<sub>0</sub> RRT 2328 and Strand<sub>1</sub> RRT 2330, as well as a reservation station 2327. Reservation station 2327 may track dynamic scheduling of instructions for register renaming. Instructions from instruction stream 2304 may be issued and designate entries in reservation station 2327 that are to be used for accessing the results of their execution. Instructions are executed as they are ready, when operands and inputs are available. Instructions may be issued to reservation station 2327, which may buffer the instruction and its operands and inputs. When all operands and inputs are available, execution of the instruction may begin.

**[00219]** The RATs may be implemented in any suitable manner such as by a table, array, or other suitable data structure. RRTs 2328, 2330 may implement some of the functionality of register files, such as the mapping or aliasing between physical and logical resources. These may track physical registers that need to be reclaimed once their associated instructions are committed and retired. Once an

instruction has been retired, resources identified in RRT with the instruction may be removed from RATs and then identified in a freelist so that the resource may be assigned to other virtual resources.

**[00220]** As mentioned, a definition queue (DQ) 2326 may track communication between strands as indicated by denotations in instruction stream 2304 that certain operations upon a logical register by each strand are to be performed upon the same physical resource mapped to the logical register.

Management of DQ 2326 may be performed by renamer 2326. If an instruction in instruction stream 2304 is identified as associated with an instruction in another strand, the connection between these instructions in different strands may be tracked in DQ 2326. The denotations in instruction stream 2304 that a given instruction in a strand is to communicate or be coordinated with an instruction in another strand may be added or implemented in any suitable manner. In one embodiment, a compiler 2306 may modify instruction stream 2304 to include denotations that instructions between strands are to be coordinated. Such modifications may be performed before instructions are received at processor 2300 or during processing by processor 2300. In another embodiment, a binary translator or other processing mechanism may dynamically amend instruction stream 2304 (e.g., at runtime) to include such denotations. As instruction stream 1804 may be divided into multiple strands, modified instructions 1810 may be modified multi-strand instructions 2310, some of which may contain the denotations described herein. In various embodiments, denotations may be added to instruction stream 2304 in conjunction with determinations of what instructions will be executed in various strands.

**[00221]** For example, compiler 2306, binary translator another suitable portion of system may determine that instruction stream 2304 is to be divided into two or more strands of sequential instructions for execution by one or more execution units 2322. One embodiment may allow instructions dependent upon the same logical-physical resource mapping to be placed into separate strands for execution as described in detail above. In another embodiment, instructions may be separated into strands for execution to communicate. Such communication may include indications of use of given logical-physical resource pairs.

**[00222]** In the embodiment of **Figure 23**, these instructions may include GLOBDEF and SYNCREG. AS described above, GLOBDEF may accept as a parameter a logical resource that is to be defined in a given strand. In one embodiment, GLOBDEF is placed in the strand that produces a register value and may replace other instructions that are to define a logical resource. In one embodiment, GLOBDEF may perform the replaced instruction's action to define the logical register as associated with the physical resource. In another embodiment, GLOBDEF may signal to renamer 2326 that communication with the other strands may be performed with respect to the instance of the logical register. Such communication may be performed to synchronize use of the logical register until another definition of the logical register is made. GLOBDEF may be placed within the original, given strand.

**[00223]** In one embodiment, a SYNCREG instruction 2308 may accept as a parameter a logical resource that is to be defined by GLOBDEF and used within the other strand. SYNCREG may be placed within the other strand, in which the value of the register is consumed. In one embodiment, SYNCREG may signal to renamer 2326 that communication with the other strands may be performed with respect to the instance of the logical register. Such communication may be performed to synchronize use of the logical register until another definition of the logical register is made. Use of SYNCREG may signal to renaming logic 1826 that a definition of the logical register has been made, to which execution may be coordinated, has been or is to be made on the original strand. In one embodiment, execution between

the strands may be coordinated such that execution in respective strands is made for the same physical-logical resource mapping for certain instructions between the strands.

**[00224]** In one embodiment, an instance of a definition instruction may be replaced in the original strand by GLOBDEF. In another embodiment, SYNCREG may be added to the other strand before instances of instructions using the logical resource. In addition, each GLOBDEF instruction may add a new identification (ID) field 2390-2393 to the definition queue (DQ) 2326. In one embodiment, the ID field comprises the architectural register number (ARN) (i.e., in addition to the name of its destination's physical register assigned by the renamer 2326). Moreover, in one embodiment, each SYNCREG instruction checks correspondence of its parameter (argument) to the ARN 2390-2393 in the DQ 2326 and does not consume the destination's physical register number from the DQ 2326 if the ARN is not the same as the SYNCREG instruction's argument. In the case of a JEClear signal from the other strand, each SYNCREG 2308 may check its own flushing bit (FB) 2307 and cancel/skip front-end pipeline restarting by the JEClear signal from the other strand if the FB 2307 is set to zero.

**[00225]** The above-described methodology of communication between strands does not describe what is happening in case of strand recovery after JEClears. **Figure 24** illustrates an example without inter-strand recovery. In particular, **Figure 24** shows how original program code 2401 (e.g., x86 code) is split into two strands 2402-2403 with an out-of-order fetch (OOOF) approach in which the compiler 2306 isolated a highly mispredicted branch into a separate strand. So JECleared strand-0 2402 does not produce any register for the other strand-1 2403. Consequently, as illustrated in **Figure 25**, in response to the JEClear 2505 in Strand-0 2501, Strand-1 2502 is not affected and the DQ pointers 2510, 2511 are moved as illustrated.

**[00226]** **Figure 26** illustrates an embodiment which utilizes inter-strand recovery in which original program code 2601 has again been separated into Strand-0 2602 and Strand-1 2603. In particular, **Figure 26** shows a situation where GLOBDEF may be executed speculatively and the JEClear on a mispredicted branch in Strand-1 will trigger a front-end pipeline restart and then GLOBDEF re-execution which will produce new value in ebx. As illustrated in **Figure 27**, Strand-0 2602 will consume wrong ebx register value and, to ensure correct code execution, the front-end pipeline should be restarted.

**[00227]** As mentioned, to limit the number of pipeline restarts, one embodiment of the invention adds new identification (ID) field to the DefQ entry. The architectural register number (ARN) may be used as the new ID field and its correspondence may be checked to the SYNCREG argument on SYNCREG execution. In example described above with respect to **Figures 26** and **27**, the ARN will always be the same as the SYNCREG argument (i.e., ebx). As such, SYNCREG may still consume the wrong register value. In one embodiment, the compiler 2306 will recognize such a situation and set the flushing bit (FB) 2307 to 1 on SYNCREG. Significantly, there are more common cases where the front-end restarting in the consuming strand could be canceled or skipped.

**[00228]** One such example is illustrated in **Figure 28**. By splitting the original program code 2801 into strands 2802-2803 as shown here the flushing bit (FB) can be set to zero for both SYNCREGs 1\* and 4\*. The front-end pipeline restarting on strand-0 by JEClear in strand-1 is now not required for the following reasons. There may be four different scenarios:

1. No JEClears. In this case, both strands execute the correct code and there are no issues.

2. A JEClear occurs in both strands. In this case, the front-end pipeline will be restarted and execution will remain correct.

3. A JEClear occurs only in strand-0. This is similar to the situation described above with respect to **Figures 24-25**. The JECleared strand-0 does not produce any register for the other strand-1 and after the JEClear the DQ pointers 2510-2511 are moved back.

4. JEClear happened only in strand-1. This is the most interesting case which may result in two sub-options:

a. The correct execution path goes to the right shoulder after Jcc. Strand-0 predicts this and executes SYNCREG 1\*. But strand-1 predicts path to the left shoulder and executes GLOBDEF 4. In this case, SYNCREG 1\* will check correspondence of its argument eax to the ARN in the DQ 2326 which is ebx and will not consume the wrong value. It will wait until the ARN becomes eax.

**[00229]** After JEClear in strand-1, the front-end pipeline will be restarted and the correct path to the right shoulder will be executed. GLOBDEF 1 will refresh the DQ entry with ARN=eax. Thus front-end pipeline restarting for strand-0 is not required. The flushing bit set to 0 for SYNCREG 1\* will also indicate that JEClearing is not required.

b. Correct execution path goes to the left shoulder after Jcc, strand-0 predicts this and executes SYNCREG 4\*. However, strand-1 predicts path to the right shoulder and executes GLOBDEF 1. In this case, SYNCREG 4\* will check correspondence of its argument ebx to the ARN in DQ 2326 which is eax and will not consume the wrong value. It will wait for the ARN to become ebx.

**[00230]** After JEClear in strand-1, the front-end pipeline will be restarted and the correct path to the left shoulder will be executed. GLOBDEF 4 will refresh the DQ entry, storing ARN=ebx. Thus, restarting the front-end pipeline for strand-0 is not required. The flushing bit (FB) set to 0 for SYNCREG 4\* will also indicate that JEClearing is not required.

**[00231]** A method in accordance with one embodiment of the invention is illustrated in **Figure 29**. The method may be implemented within the context of the architectures described above but is not limited to any particular processor or system architecture.

**[00232]** At 2900, program code is split into two strands: strand-0 and strand-1. In this particular example, strand-0 does not define registers for strand-1 but strand-1 defines registers for strand-0. At 2901, a determination is made as to whether a branch misprediction has occurred (e.g., in response to detecting a JEClear or similar operation). If not, then there are no issues and strand-1 and strand-0 execute normally. If so, then a determination is made at 2902 as to whether the misprediction has occurred in both strands. If so, then the front end pipeline for the strands is restarted (e.g., responsive to the JEClear). If not, then at 2903 a determination is made as to whether the misprediction was in strand-0. If so, then because the JECleared strand-0 does not produce any register for strand-1, all that is required after the JEClear is the adjustment of the definition queue (DQ) (e.g., which may be moved back as described above).

**[00233]** If the misprediction occurred in strand-1 then at 2908, a synchronization instruction in strand-0 (e.g., SYNCREG) checks the correspondence between its register argument (e.g., eax as in Figure 28) and the identifier stored in the DQ. If there is no match, then strand-0 will wait until there is a match before consuming the value (i.e., it will not consume the incorrect value). Thus, in 2908, the front end pipeline does not need to be restarted for strand-0.

**[00234]** The embodiments of the invention reduce the number of front-end pipeline restarts in one strand due to a JEClear signal from the other strand and thus improve the performance of inter-strand processing. In some embodiments, modifications to both Software and Hardware components are performed. For example, in one embodiment, the compiler 2306 will analyze resulting strands, supply SYNCREG instructions 2308 with additional flushing bit 2307 to ensure that no incorrect register values will be consumed. In addition, in one embodiment, the DQ 2326 and execution of GLOBDEFs and SYNCREGs may be updated so that a new ID field 2390-2393 is included in each DQ entry – using the architectural register number (ARN) in one embodiment. GLOBDEF may fill this field and SYNCREG will check correspondence of its argument to the ARN from the DQ entry if flushing bit (FB) is set to zero and should not consume destination's physical register number from DQ if the ARN is not the same as SYNCREG's argument. Only if the FB set to 1 does the front-end pipeline need to be restarted by a JEClear signal from other strand.

**[00235]** Techniques for performing one or more instructions according to at least one embodiment are disclosed. While certain exemplary embodiments have been described and shown in the accompanying drawings, it is to be understood that such embodiments are merely illustrative of and not restrictive on other embodiments, and that such embodiments not be limited to the specific constructions and arrangements shown and described, since various other modifications may occur to those ordinarily skilled in the art upon studying this disclosure. In an area of technology such as this, where growth is fast and further advancements are not easily foreseen, the disclosed embodiments may be readily modifiable in arrangement and detail as facilitated by enabling technological advancements without departing from the principles of the present disclosure or the scope of the accompanying claims.

**[00236]** Embodiments of the invention may include various steps, which have been described above. The steps may be embodied in machine-executable instructions which may be used to cause a general-purpose or special-purpose processor to perform the steps. Alternatively, these steps may be performed by specific hardware components that contain hardwired logic for performing the steps, or by any combination of programmed computer components and custom hardware components.

**[00237]** As described herein, instructions may refer to specific configurations of hardware such as application specific integrated circuits (ASICs) configured to perform certain operations or having a predetermined functionality or software instructions stored in memory embodied in a non-transitory computer readable medium. Thus, the techniques shown in the Figures can be implemented using code and data stored and executed on one or more electronic devices (e.g., an end station, a network element, etc.). Such electronic devices store and communicate (internally and/or with other electronic devices over a network) code and data using computer machine-readable media, such as non-transitory computer machine-readable storage media (e.g., magnetic disks; optical disks; random access memory; read only memory; flash memory devices; phase-change memory) and transitory computer machine-readable communication media (e.g., electrical, optical, acoustical or other form of propagated signals – such as carrier waves, infrared signals, digital signals, etc.). In addition, such electronic devices typically include a set of one or more processors coupled to one or more other components, such as one or more storage devices (non-transitory machine-readable storage media), user input/output devices (e.g., a keyboard, a touchscreen, and/or a display), and network connections. The coupling of the set of processors and other components is typically through one or more busses and bridges (also termed as bus controllers). The storage device and signals carrying the network traffic respectively represent one or more machine-

readable storage media and machine-readable communication media. Thus, the storage device of a given electronic device typically stores code and/or data for execution on the set of one or more processors of that electronic device. Of course, one or more parts of an embodiment of the invention may be implemented using different combinations of software, firmware, and/or hardware. Throughout this detailed description, for the purposes of explanation, numerous specific details were set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the invention may be practiced without some of these specific details. In certain instances, well known structures and functions were not described in elaborate detail in order to avoid obscuring the subject matter of the present invention. Accordingly, the scope and spirit of the invention should be judged in terms of the claims which follow.

**CLAIMS**

What is claimed is:

1. A processor comprising:

a plurality of cores to concurrently execute a plurality of strands of program code;  
a first circuit to separate an instruction sequence into at least a first strand and a second strand;

a second circuit comprising a plurality of entries to track associations between one or more instructions in the first strand and one or more instructions in the second strand, the second circuit to store a register identifier in a first entry responsive to execution of a first instruction in the first strand;

a third circuit to compare an argument of a second instruction in a second strand with the register identifier in the first entry in response to detecting a branch misprediction in the second strand, wherein if the argument and the register identifier do not match, then the third circuit to cause the second strand to wait until the argument and the register identifier match before consuming a register value.

2. The processor as in claim 1 further comprising:

a front end pipeline to fetch and decode instructions in the first and second strands, wherein if a branch misprediction is detected in both the first and second strands, then the front end pipeline is to be restarted.

3. The processor as in claim 1 wherein detecting a branch misprediction comprises detecting a jump execution clear (JEClear) instruction.

4. The processor as in claim 1 further comprising:

a fourth circuit to check a bit field within the second instruction, wherein if the bit field has a first value then a front end pipeline will restart execution of the second strand and if the bit field has a second value, then the third circuit will cause the second strand to wait until the argument and the register identifier match before consuming a register value.

5. The processor as in claim 1 wherein if a branch misprediction is detected within the first strand, then the second circuit is to adjust one or more pointers to the entries.

6. The processor as in claim 1 further comprising:
  - a renamer to perform register renaming operations to map architectural registers to physical registers for the first strand and the second strand.
7. The processor as in claim 6 further comprising:
  - a first register renaming table to store register rename entries associated with the first strand; and
  - a second register renaming table to store register rename entries associated with the second strand.
8. The processor as in claim 7 further comprising:
  - a plurality of execution resources; and
  - a reservation station to schedule execution of instructions in the first strand and the second strand on the execution resources.
9. The processor as in claim 1 wherein the register identifier comprises an architectural register number (ARN).
10. A system comprising:
  - a memory to store instructions and data;
  - a processor to execute the instructions and process the data;
  - a graphics processor to perform graphics operations in response to graphics instructions;
  - a network interface to receive and transmit data over a network;
  - an interface for receiving user input from a mouse or cursor control device, the plurality of cores executing the instructions and processing the data responsive to the user input;
  - the processor comprising:
    - a plurality of cores to concurrently execute a plurality of strands of program code;
    - a first circuit to separate an instruction sequence into at least a first strand and a second strand;
    - a second circuit comprising a plurality of entries to track associations between one or more instructions in the first strand and one or more instructions in the second strand, the second circuit to store a register identifier in a first entry responsive to execution of a first instruction in the first strand;

a third circuit to compare an argument of a second instruction in a second strand with the register identifier in the first entry in response to detecting a branch misprediction in the second strand, wherein if the argument and the register identifier do not match, then the third circuit to cause the second strand to wait until the argument and the register identifier match before consuming a register value.

11. The system as in claim 10 further comprising:

a front end pipeline to fetch and decode instructions in the first and second strands, wherein if a branch misprediction is detected in both the first and second strands, then the front end pipeline is to be restarted.

12. The system as in claim 10 wherein detecting a branch misprediction comprises detecting a jump execution clear (JEClear) instruction.

13. The system as in claim 10 further comprising:

a fourth circuit to check a bit field within the second instruction, wherein if the bit field has a first value then a front end pipeline will restart execution of the second strand and if the bit field has a second value, then the third circuit will cause the second strand to wait until the argument and the register identifier match before consuming a register value.

14. The system as in claim 10 wherein if a branch misprediction is detected within the first strand, then the second circuit is to adjust one or more pointers to the entries.

15. The system as in claim 10 further comprising:

a renamer to perform register renaming operations to map architectural registers to physical registers for the first strand and the second strand.

16. The system as in claim 15 further comprising:

a first register renaming table to store register rename entries associated with the first strand; and

a second register renaming table to store register rename entries associated with the second strand.

17. The system as in claim 16 further comprising:  
a plurality of execution resources; and  
a reservation station to schedule execution of instructions in the first strand and the second strand on the execution resources.

18. A method comprising:  
separating an instruction sequence into at least a first strand and a second strand;  
tracking associations between one or more instructions in the first strand and one or more instructions in the second strand within a first circuit comprising a plurality of entries;  
storing a register identifier in a first entry responsive to execution of a first instruction in the first strand;  
detecting a branch misprediction in the second strand;  
responsively comparing an argument of a second instruction in a second strand with the register identifier in the first entry;  
causing the second strand to wait until the argument and the register identifier match before consuming a register value if the argument and the register identifier do not match.

19. The method as in claim 18 further comprising:  
fetching and decoding instructions in the first and second strands, wherein if a branch misprediction is detected in both the first and second strands, then restarting the front end pipeline.

20. The method as in claim 18 wherein detecting a branch misprediction comprises detecting a jump execution clear (JEClear) instruction.

21. The method as in claim 18 further comprising:  
checking a bit field within the second instruction, wherein if the bit field has a first value then a front end pipeline will restart execution of the second strand and if the bit field has a second value, then causing the second strand to wait until the argument and the register identifier match before consuming a register value.

22. The method as in claim 18 wherein if a branch misprediction is detected within the first strand, then adjusting one or more pointers to the entries.

23. The method as in claim 18 further comprising:  
performing register renaming operations to map architectural registers to physical registers for the first strand and the second strand.

24. The method as in claim 23 further comprising:  
storing register rename entries associated with the first strand in a first register renaming table; and  
storing register rename entries associated with the second strand in a second register renaming table.

25. The method as in claim 24 further comprising:  
scheduling execution of instructions in the first strand and the second strand on a plurality of execution resources.

1/37

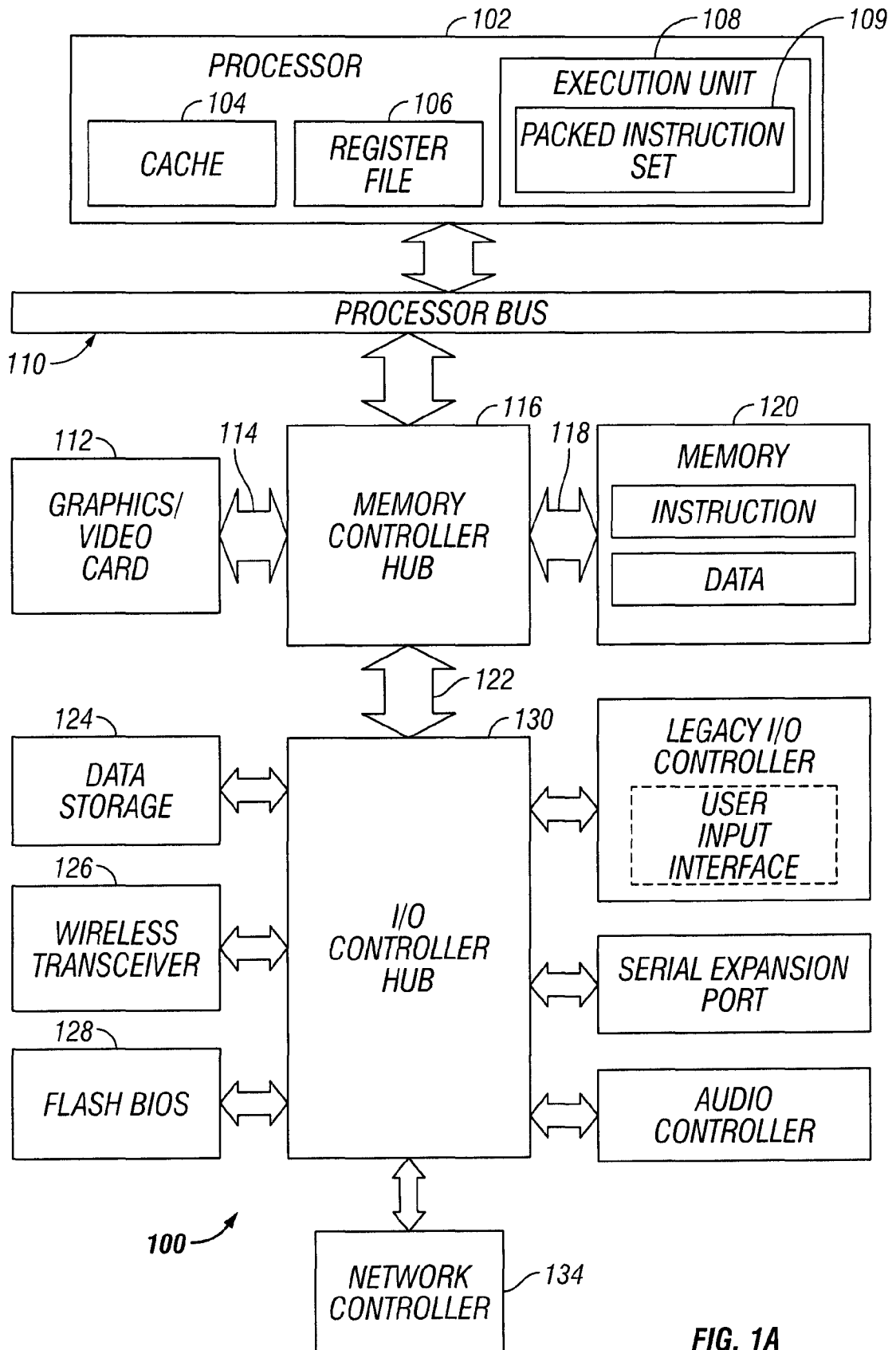


FIG. 1A

2/37

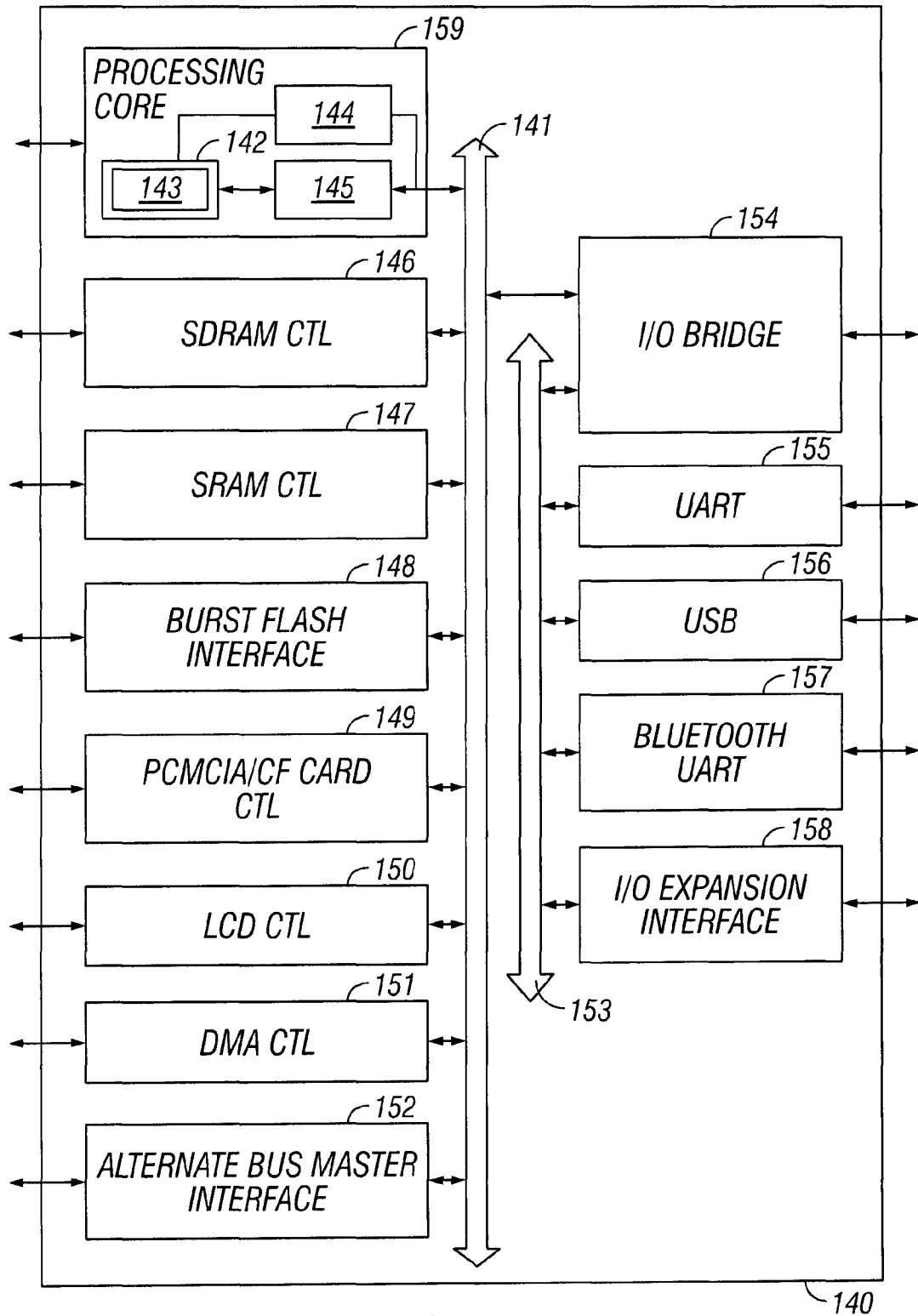


FIG. 1B

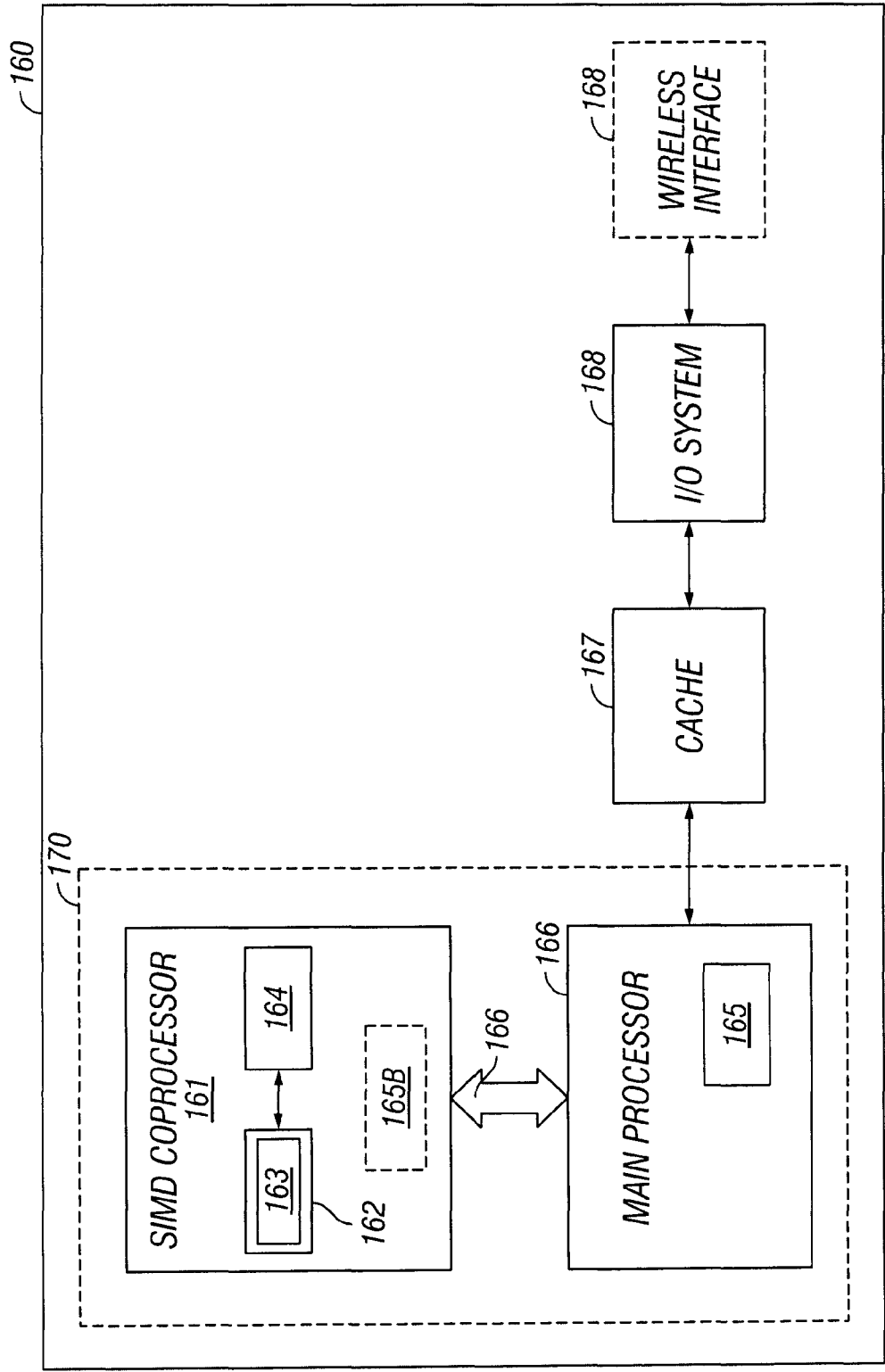


FIG. 1C

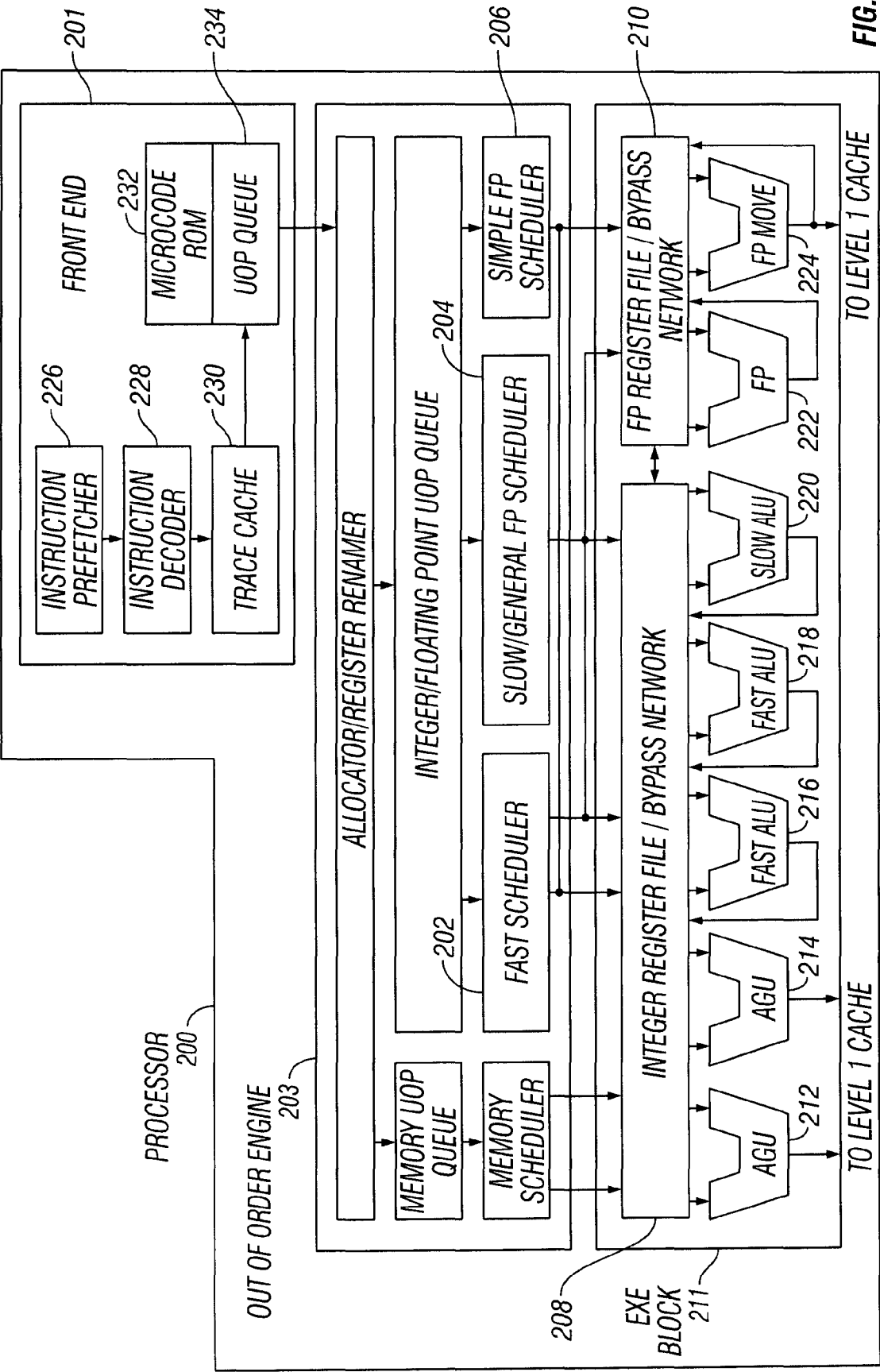


FIG. 2

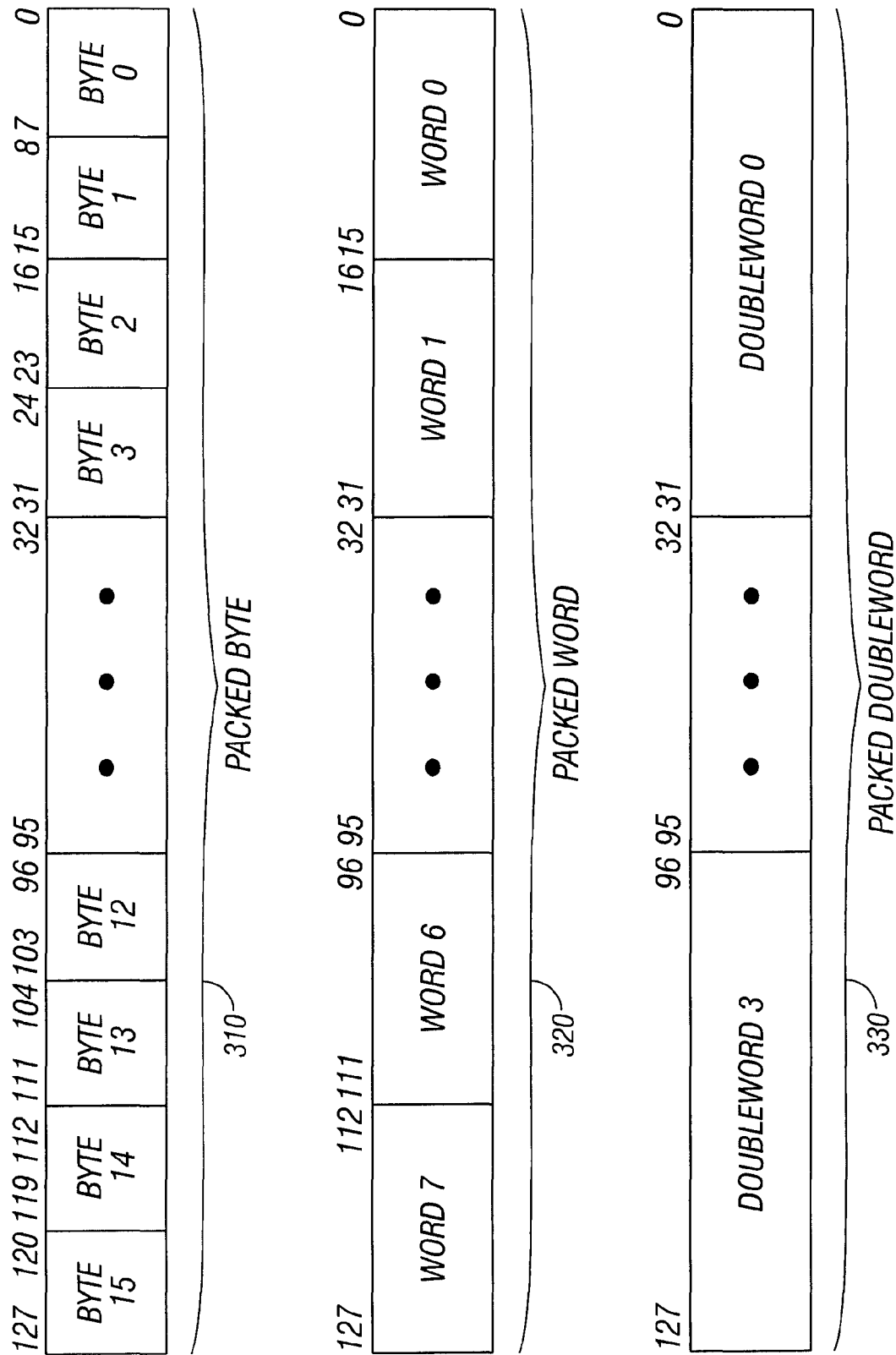


FIG. 3A

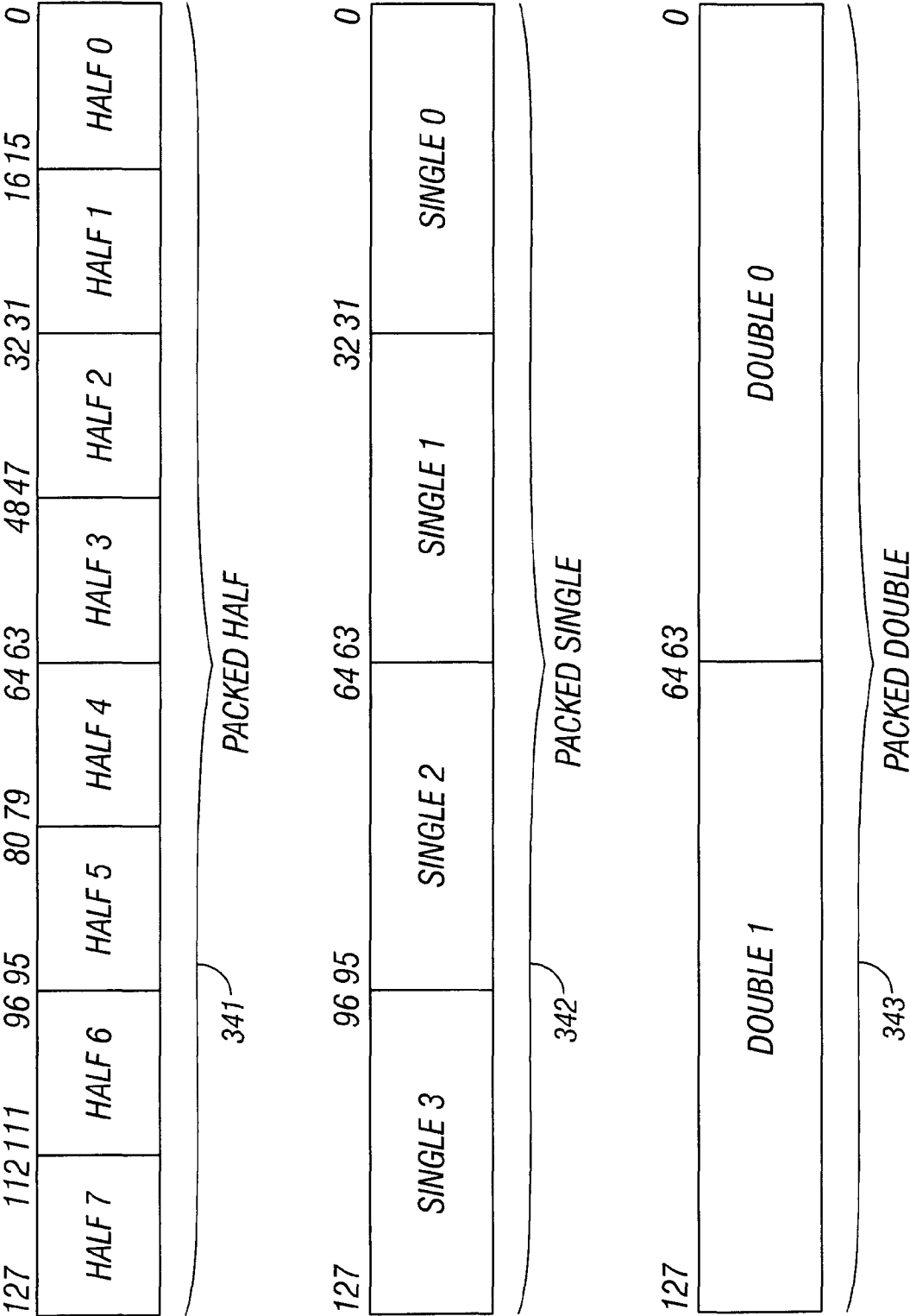
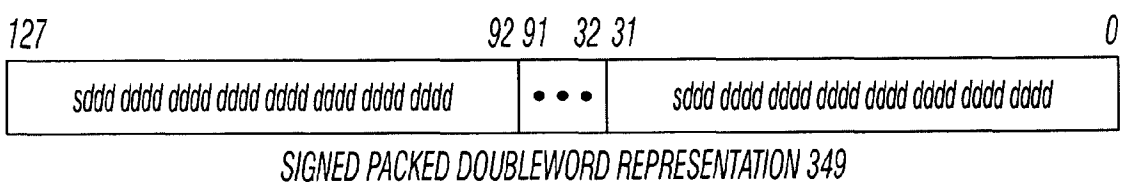
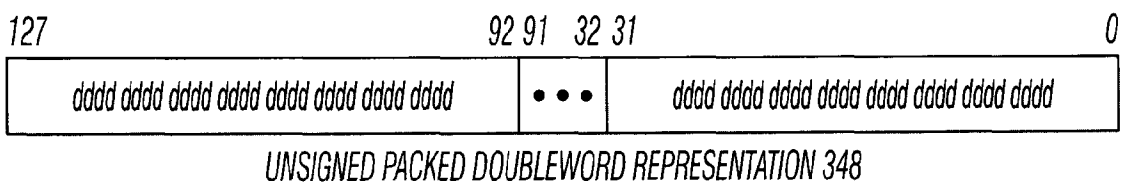
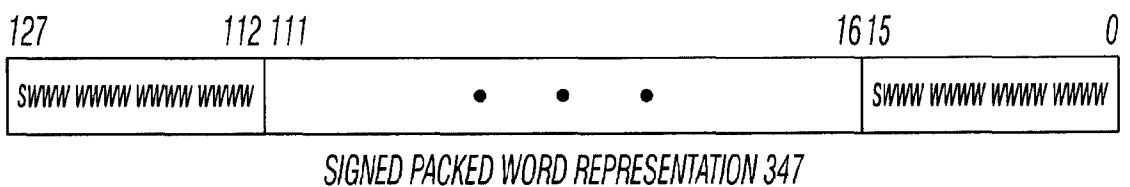
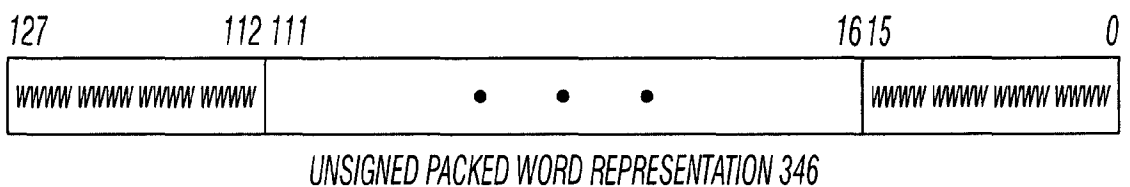
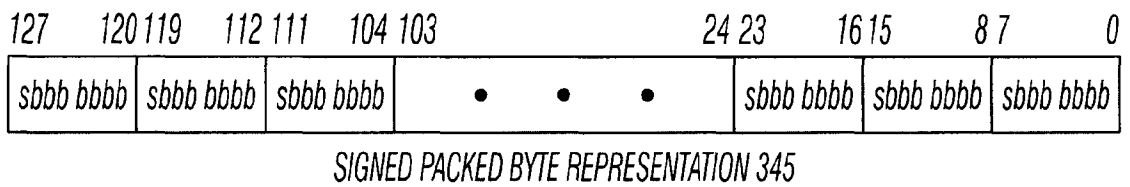
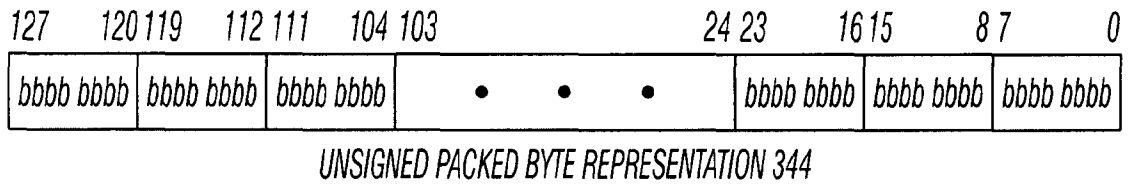


FIG. 3B

7/37

**FIG. 3C**

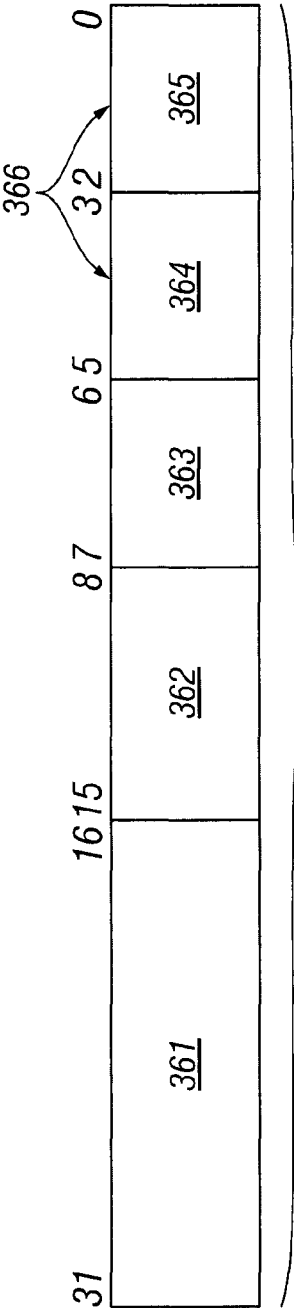


FIG. 3D

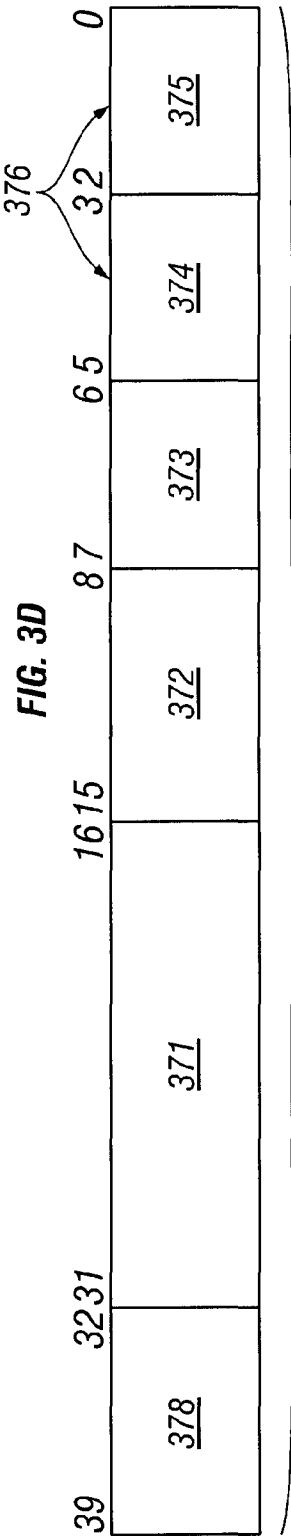


FIG. 3E

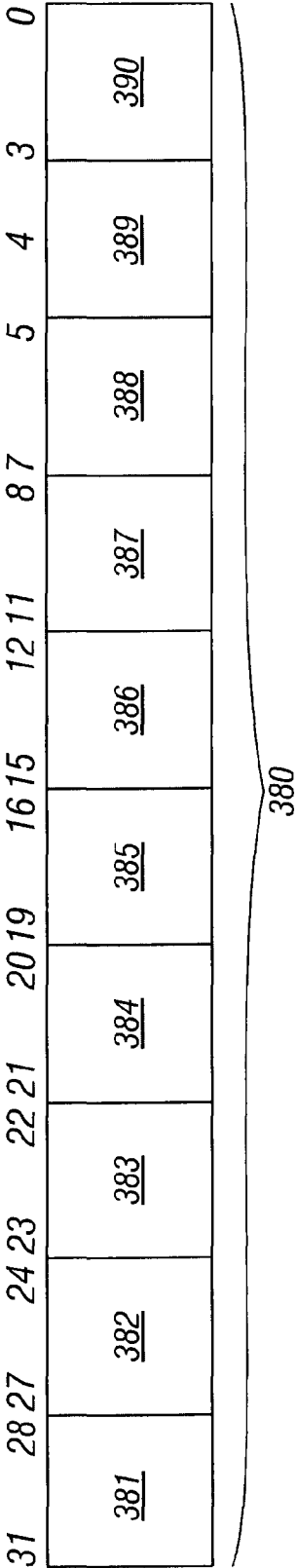


FIG. 3F

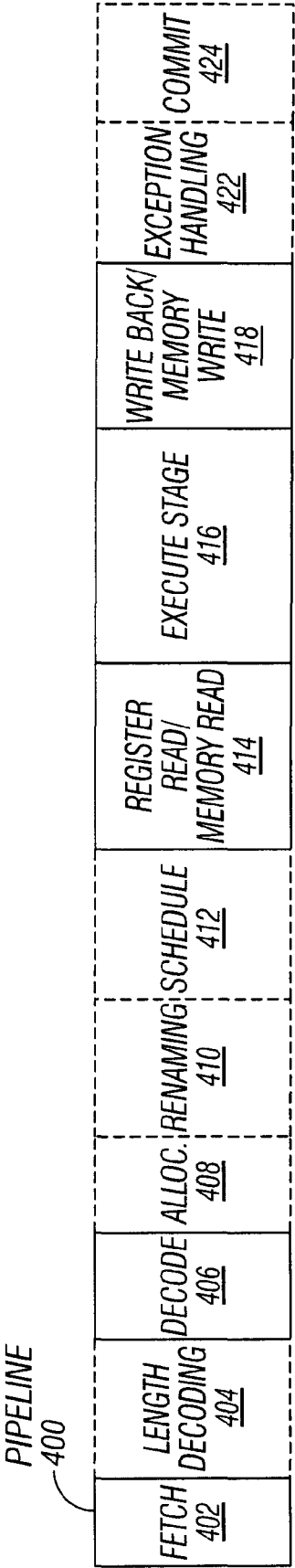


FIG. 4A

10/37

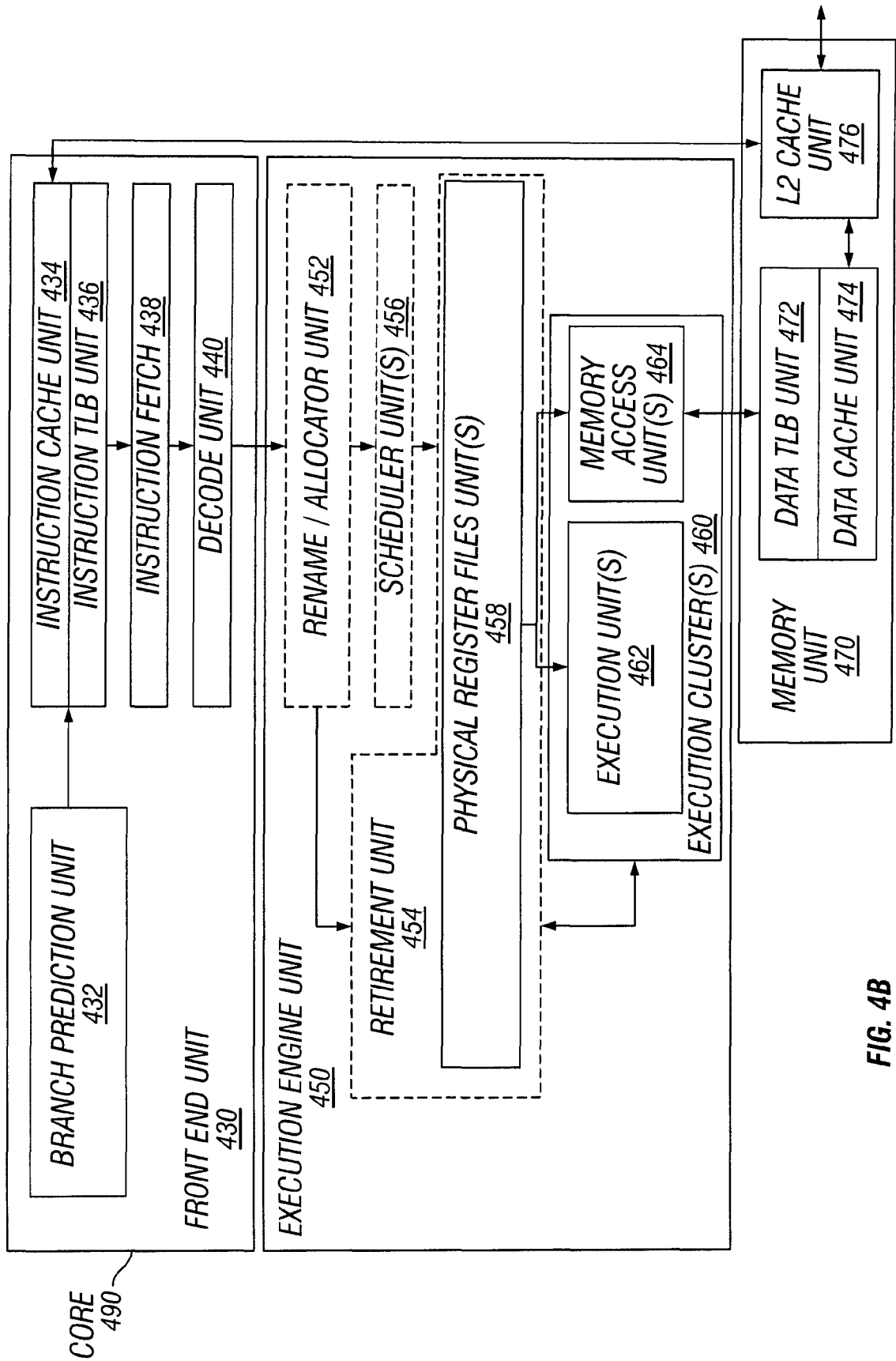
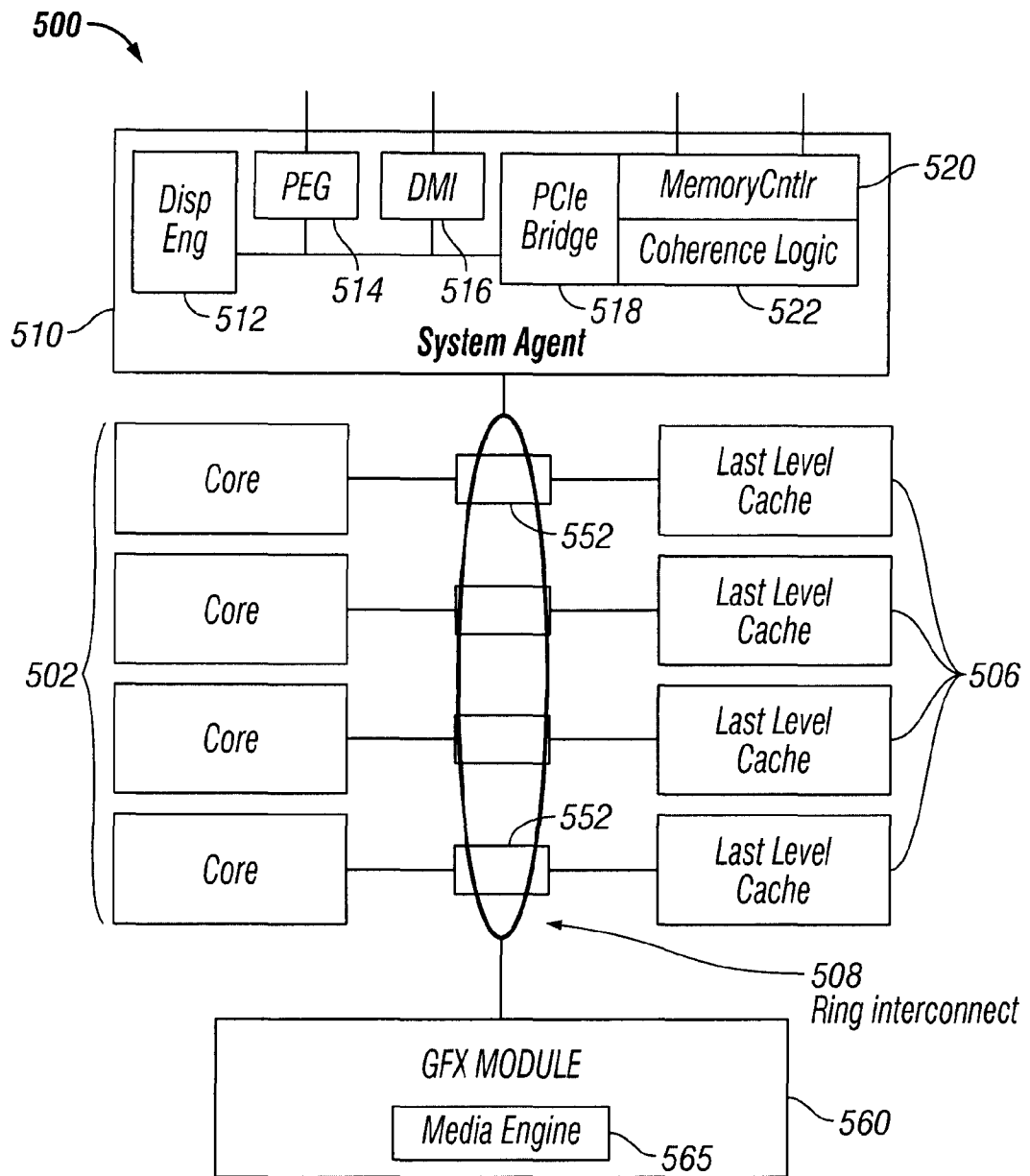


FIG. 4B

11/37

**FIG. 5A**

12/37

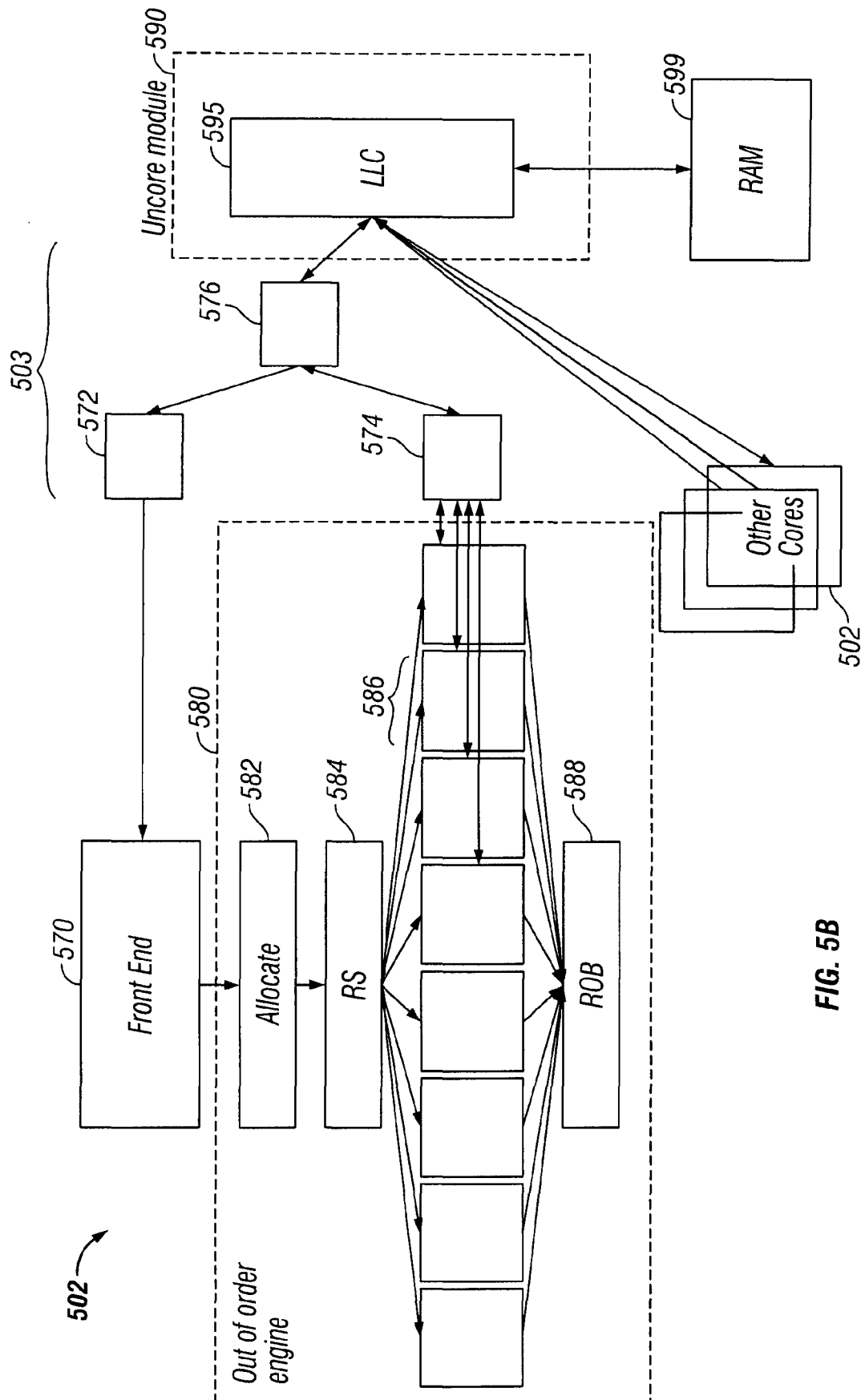
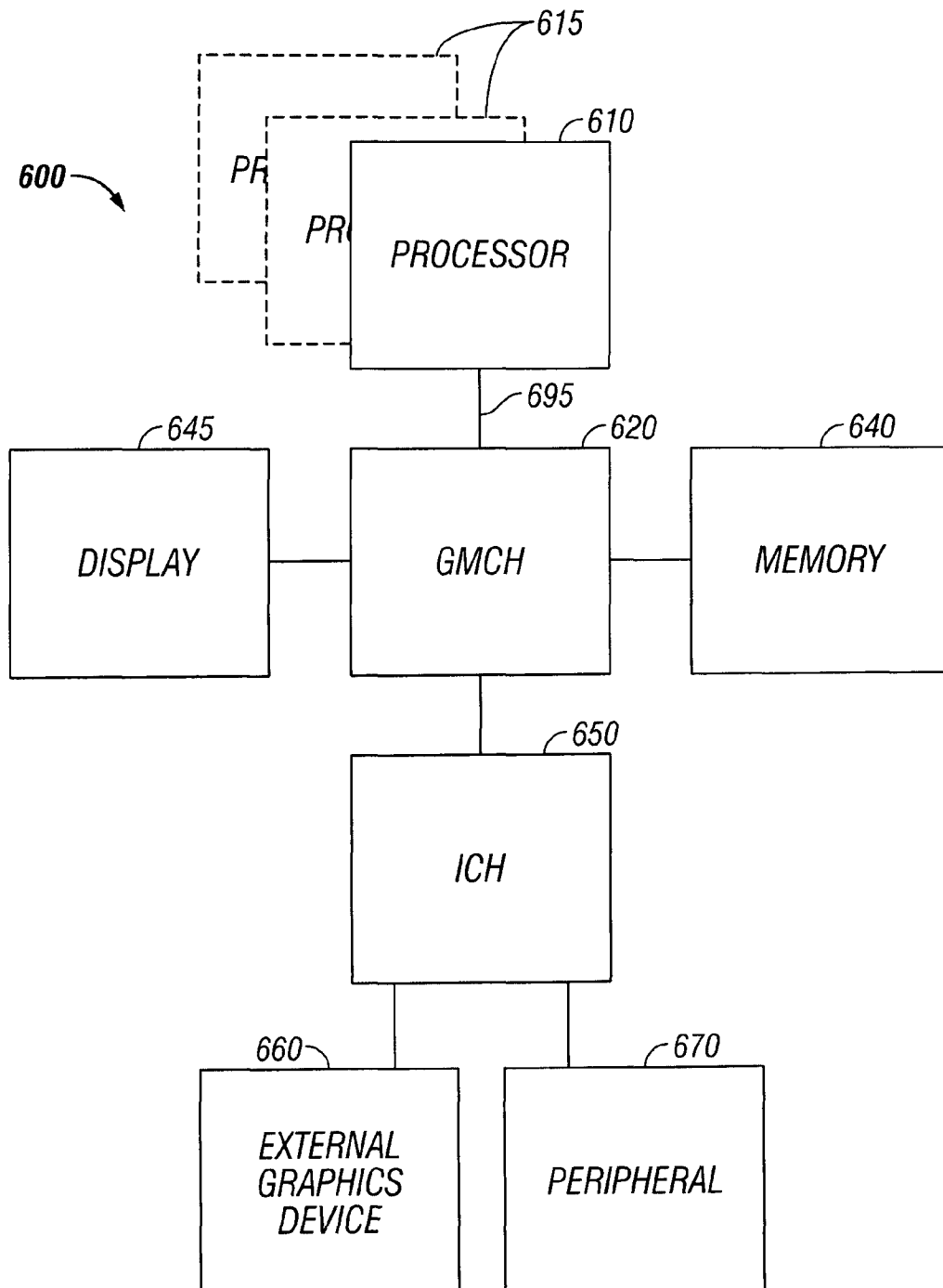


FIG. 5B

13/37

**FIG. 6**

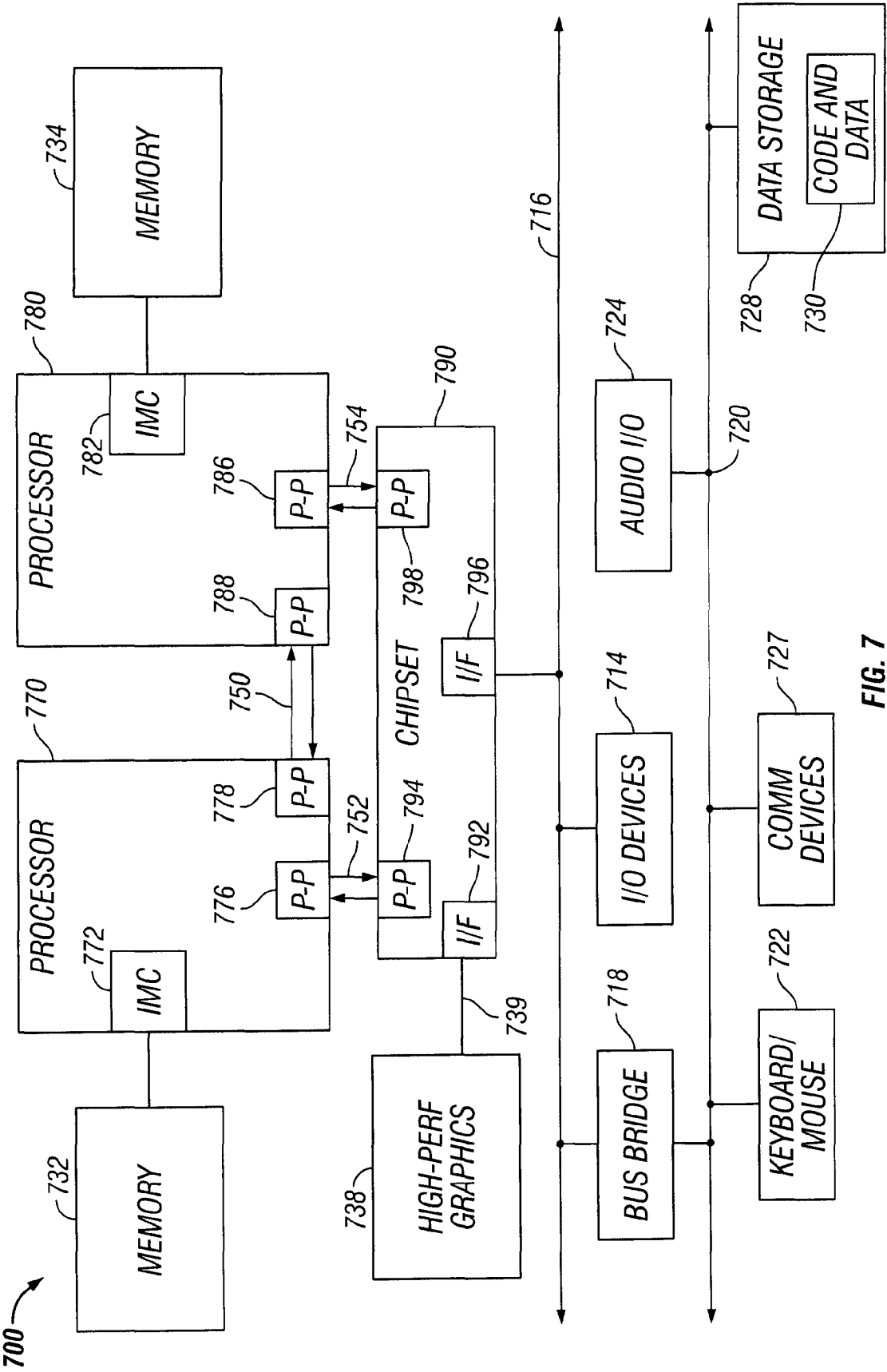


FIG. 7

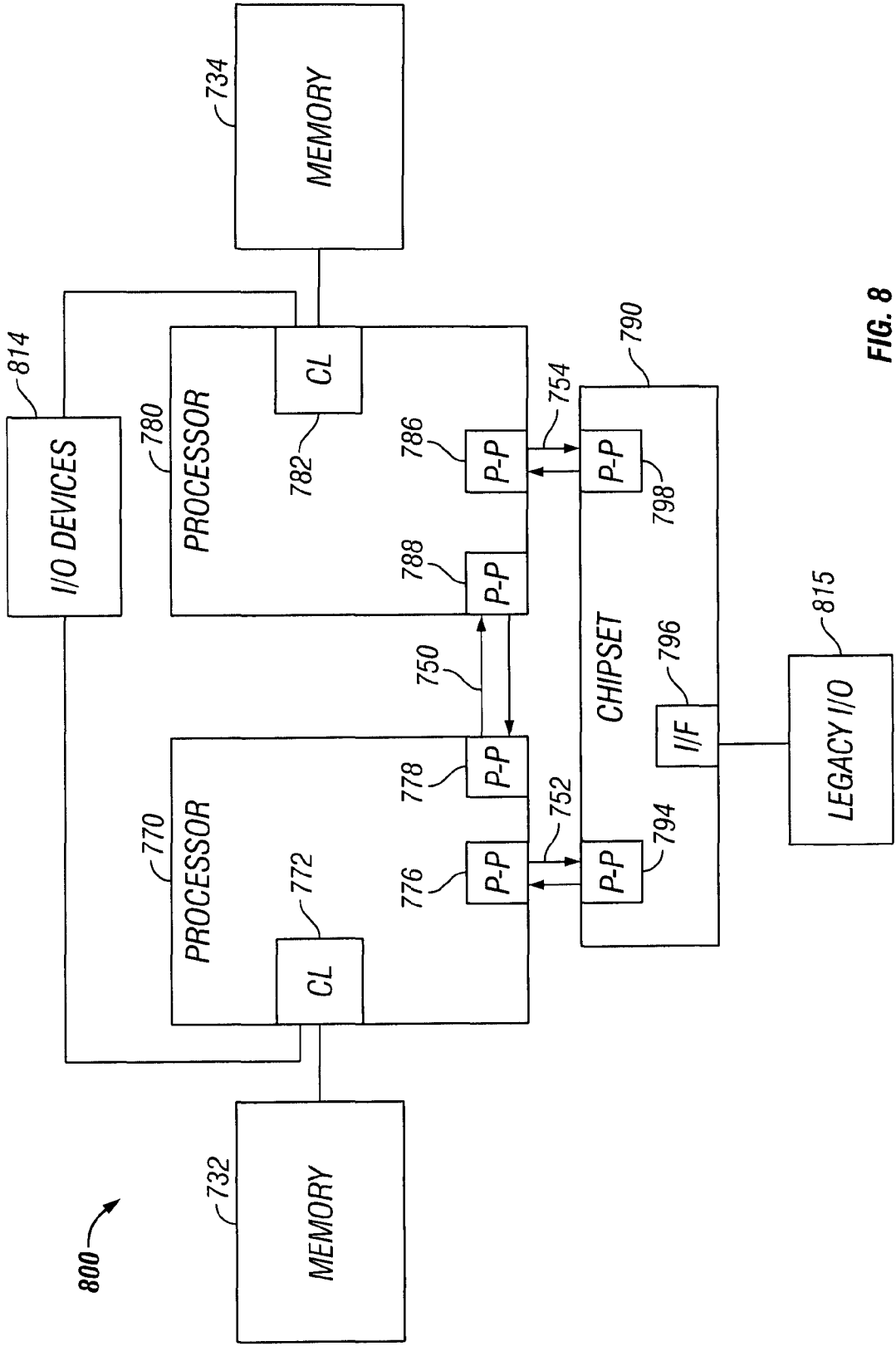


FIG. 8

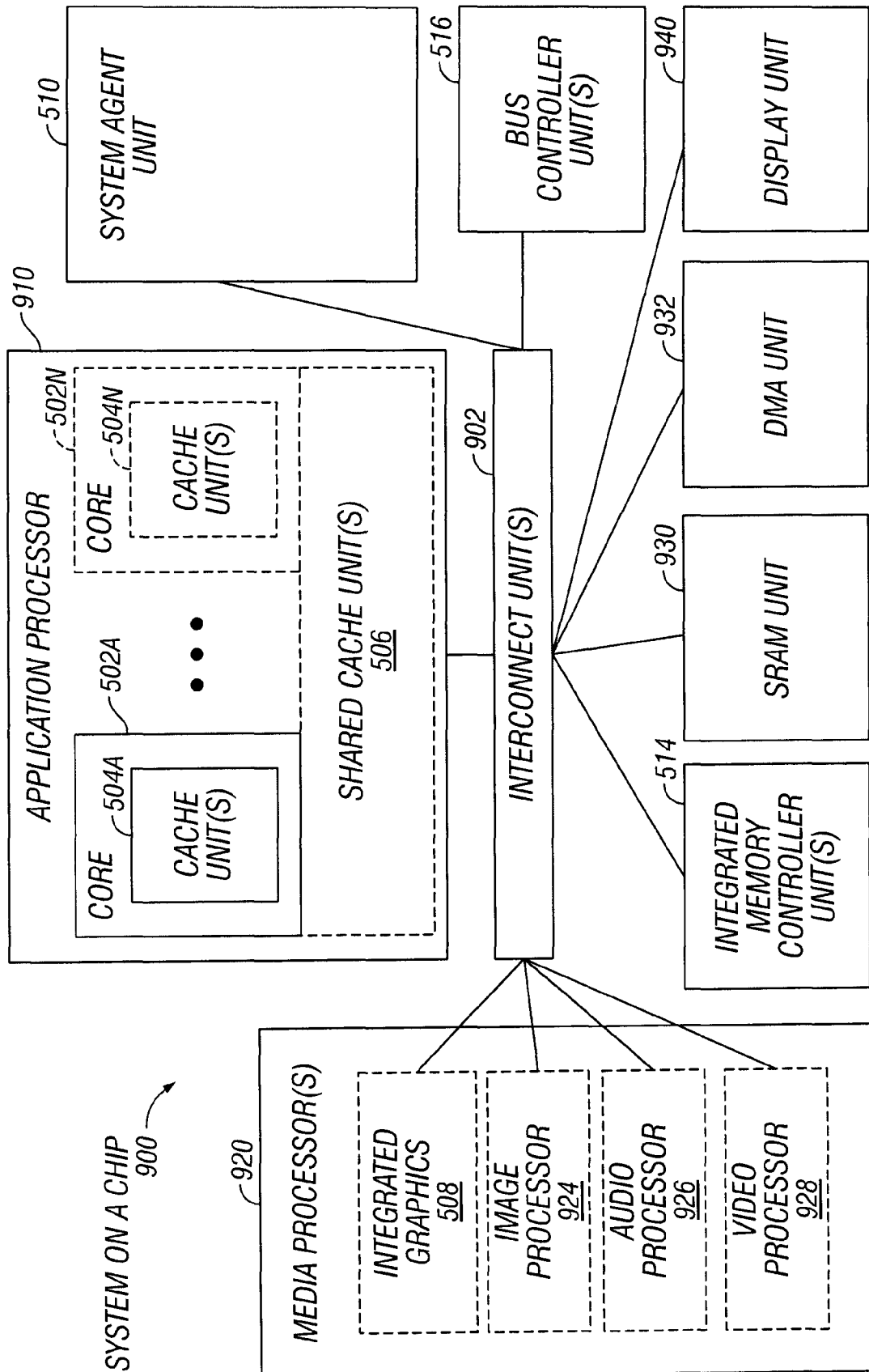
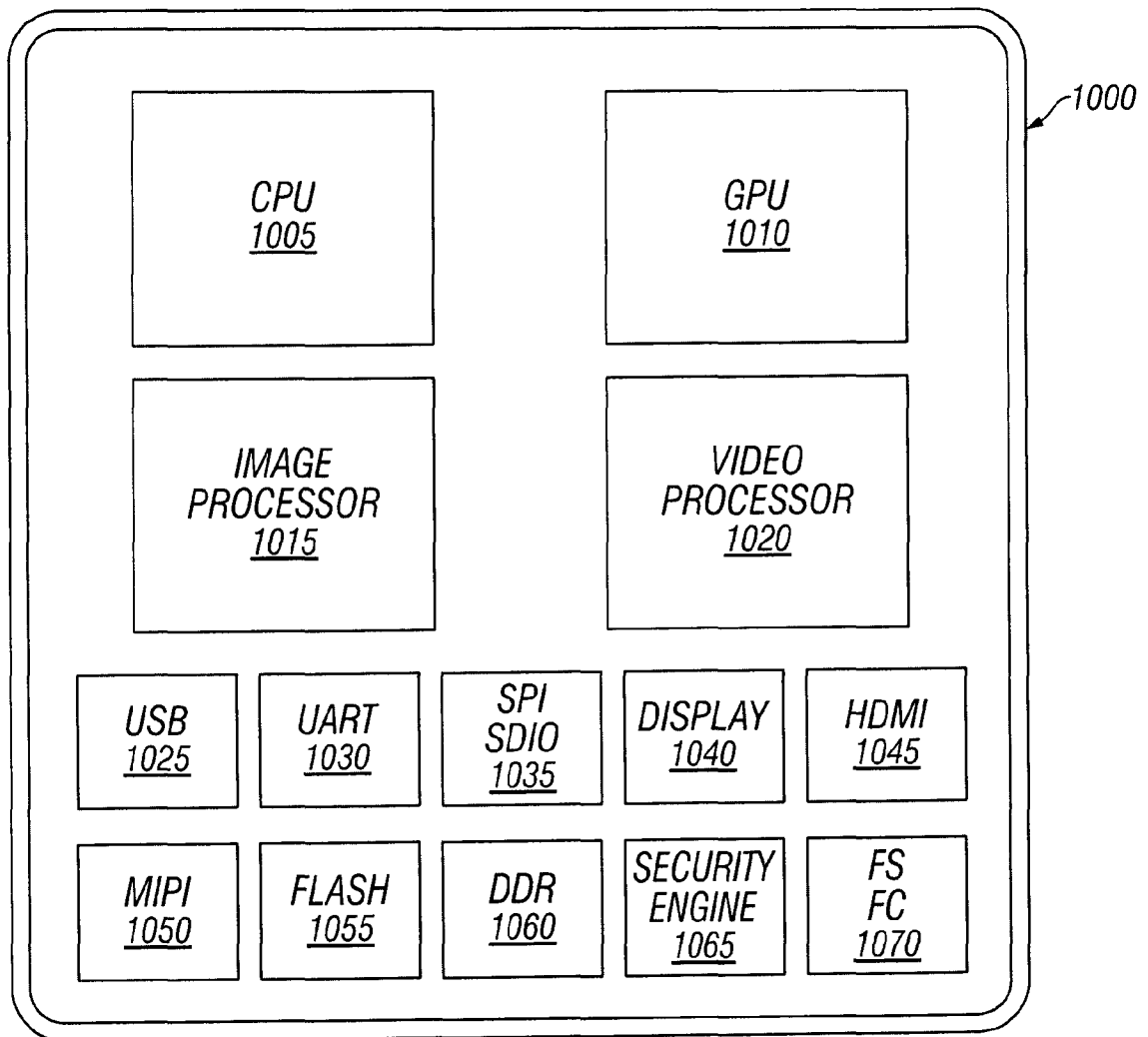


FIG. 9

17/37

**FIG. 10**

18/37

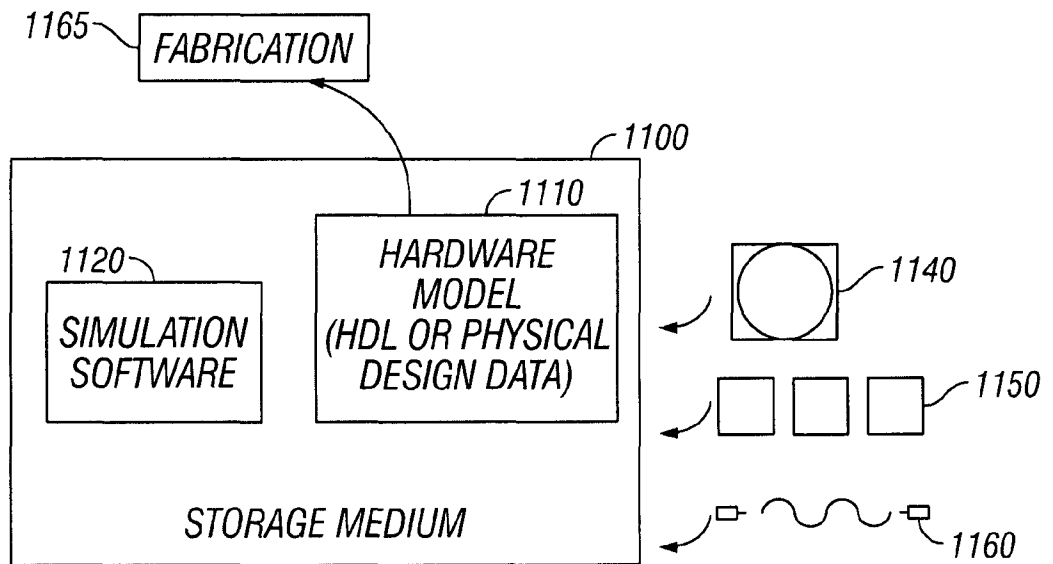


FIG. 11

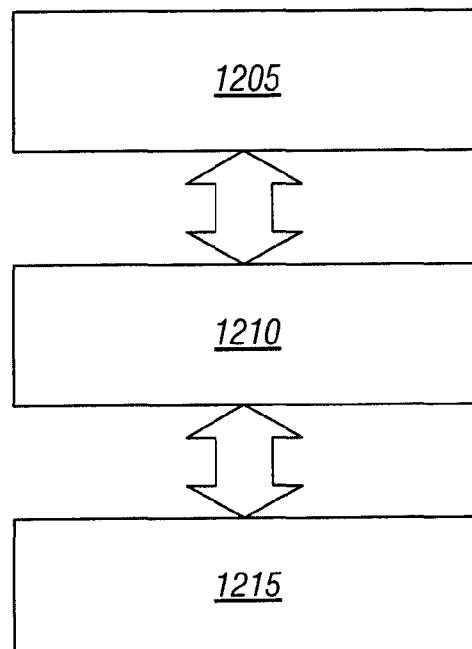


FIG. 12

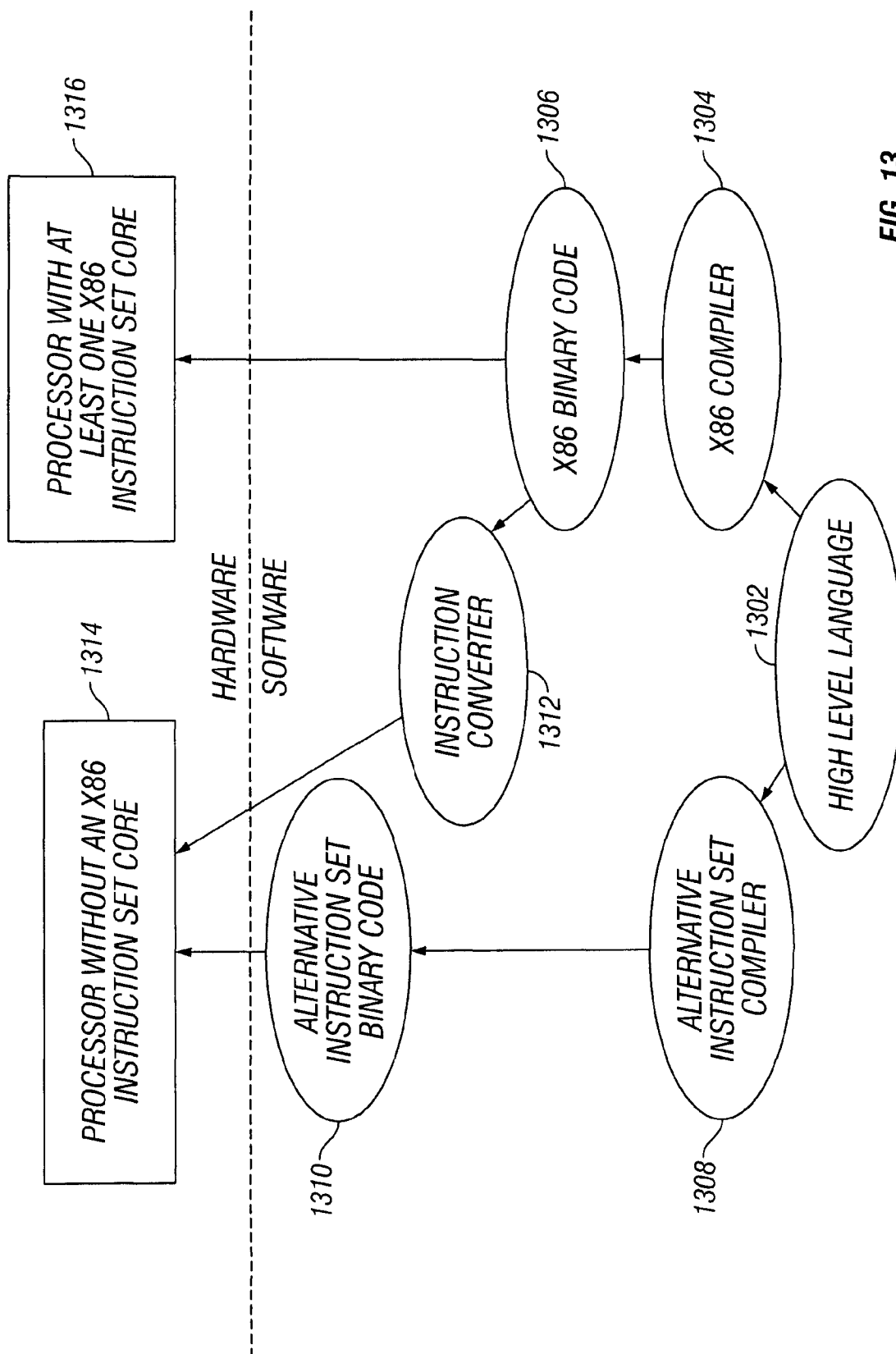


FIG. 13

20/37

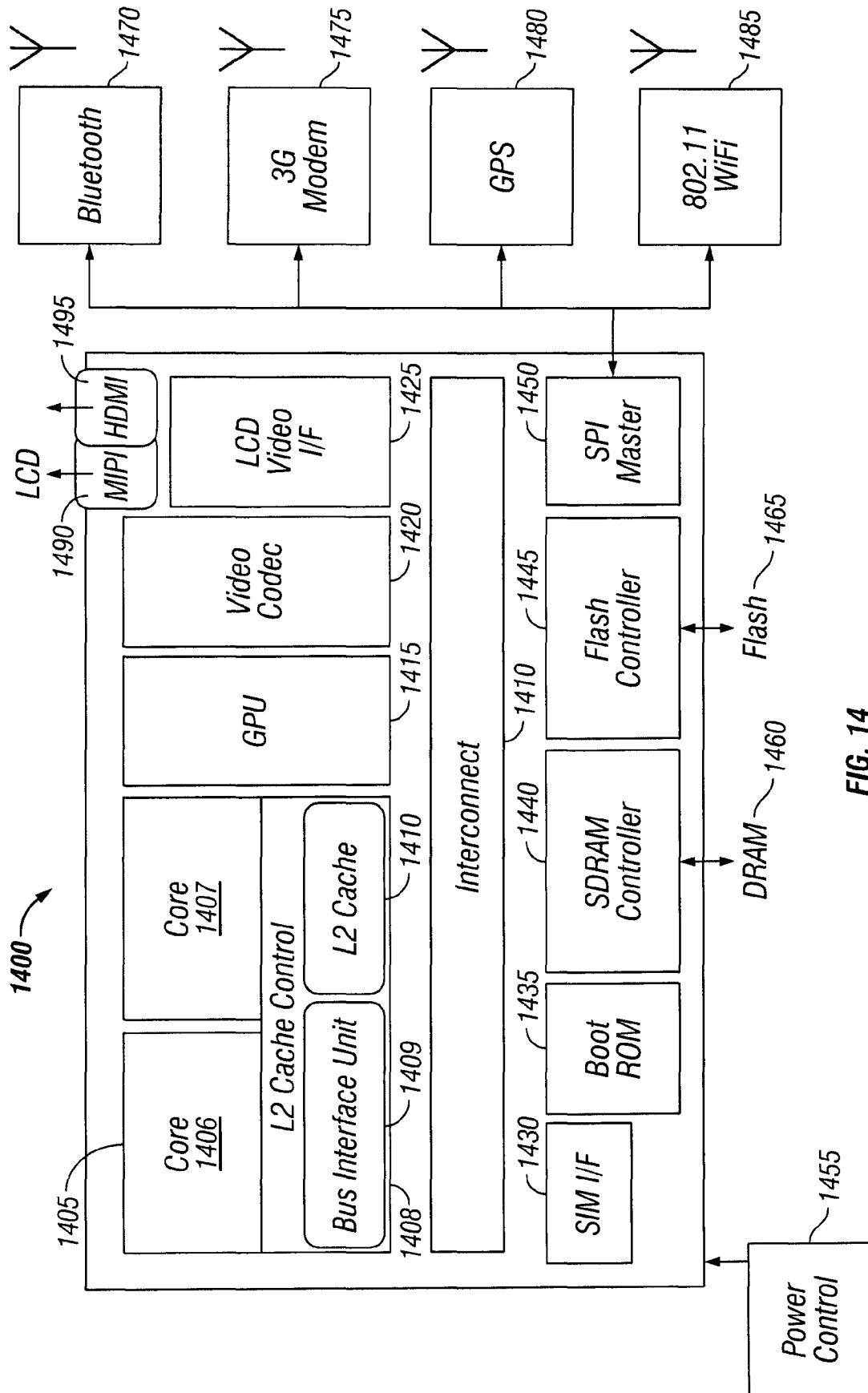


FIG. 14

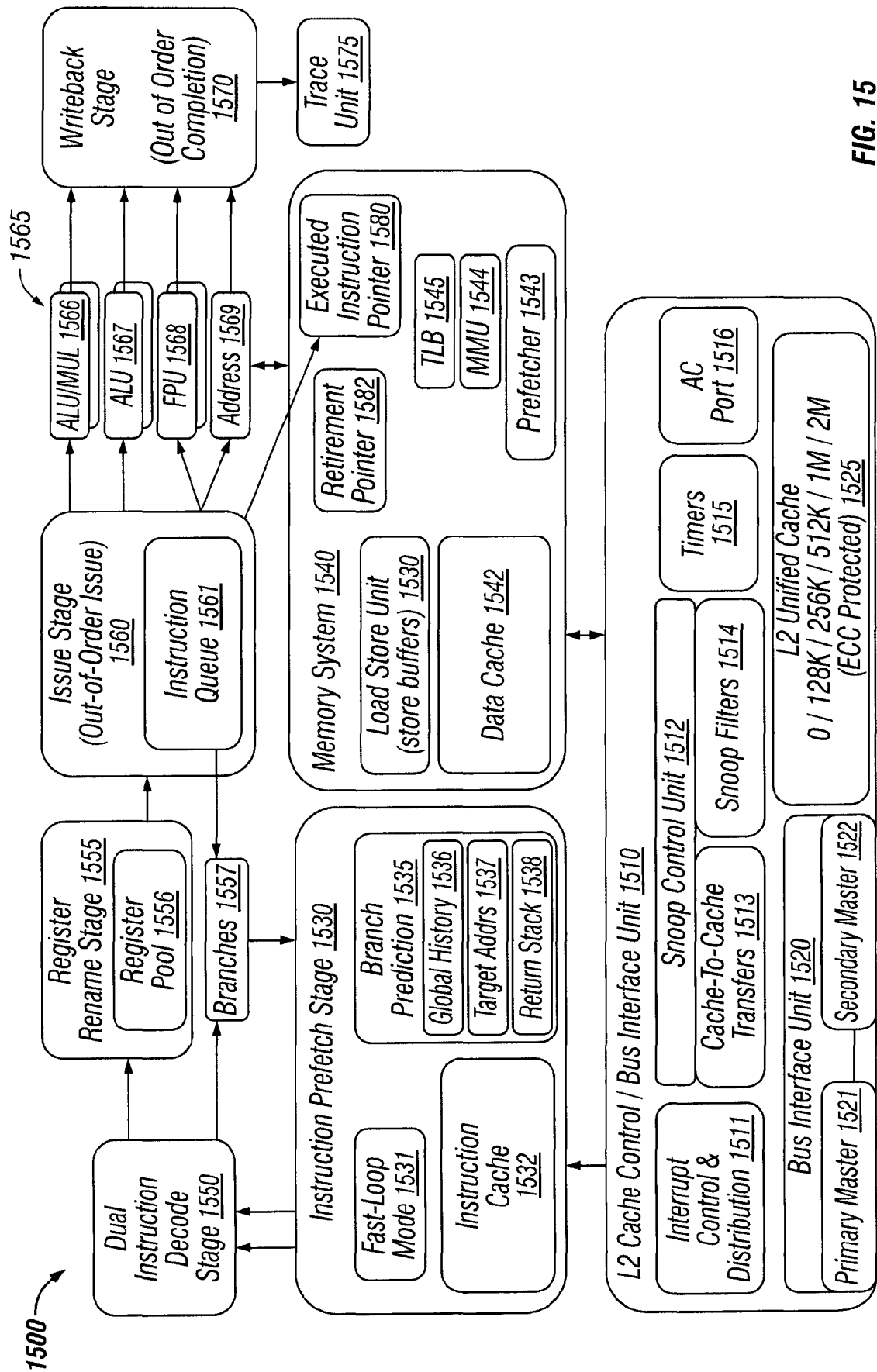


FIG. 15

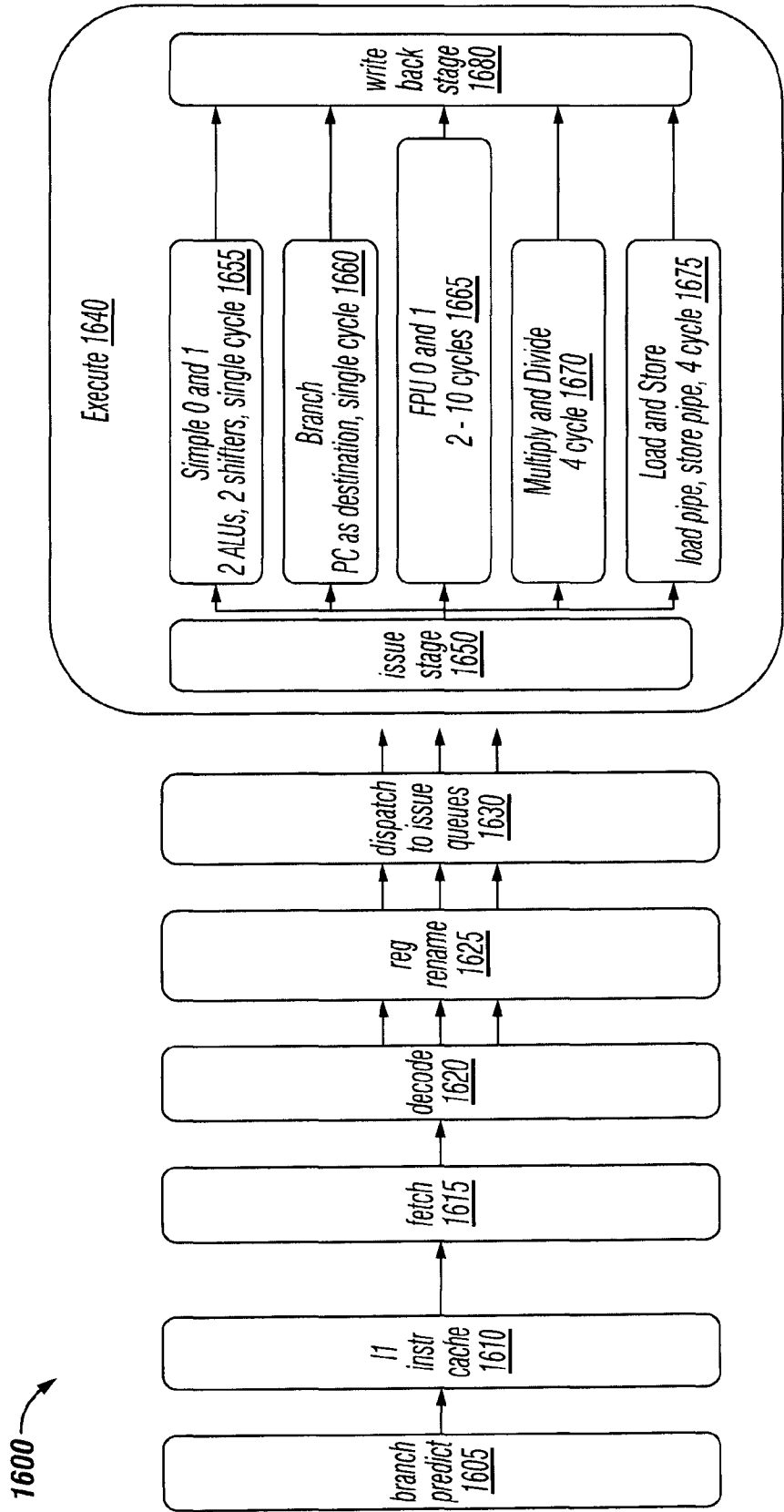


FIG. 16

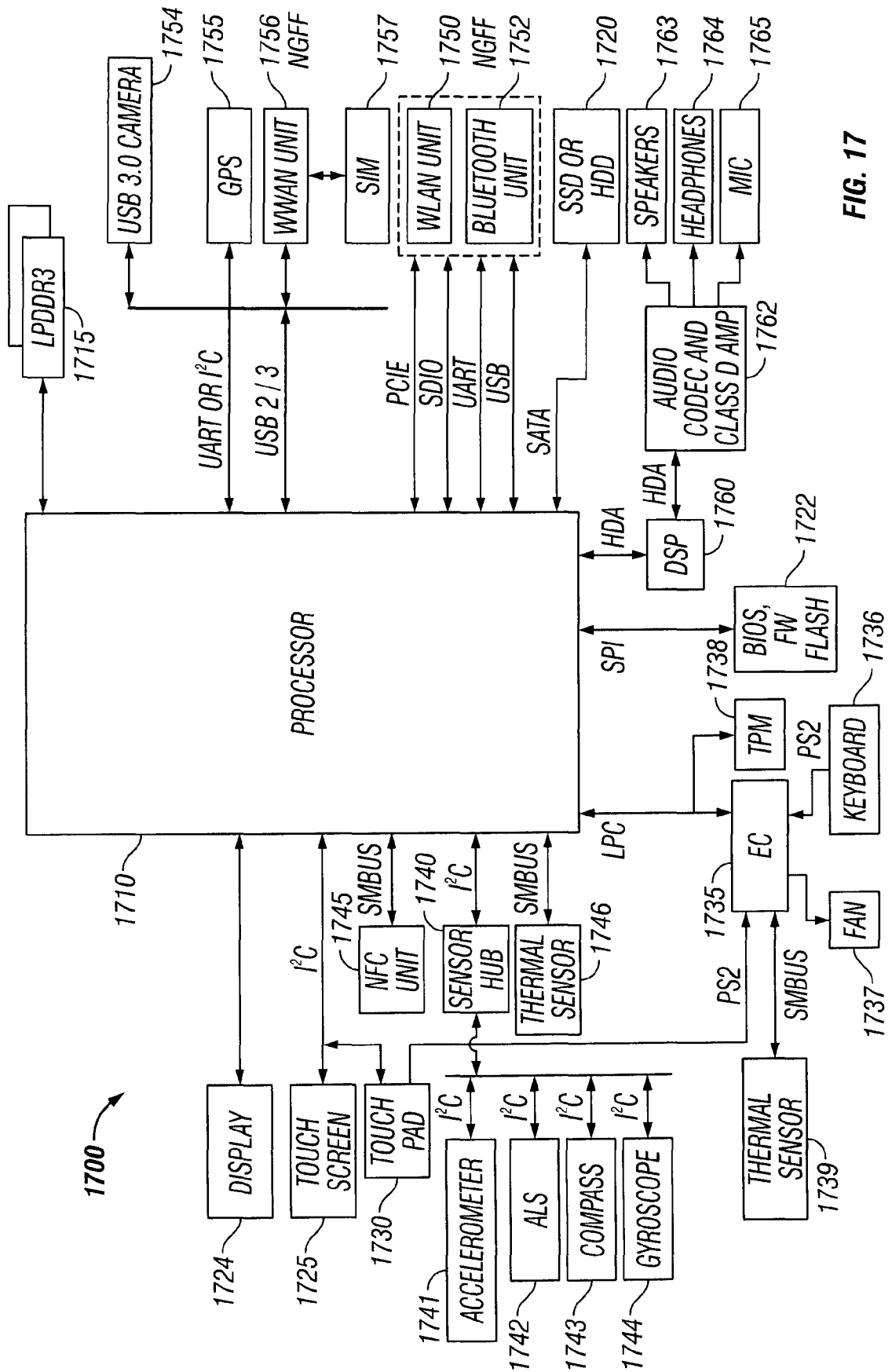


FIG. 17

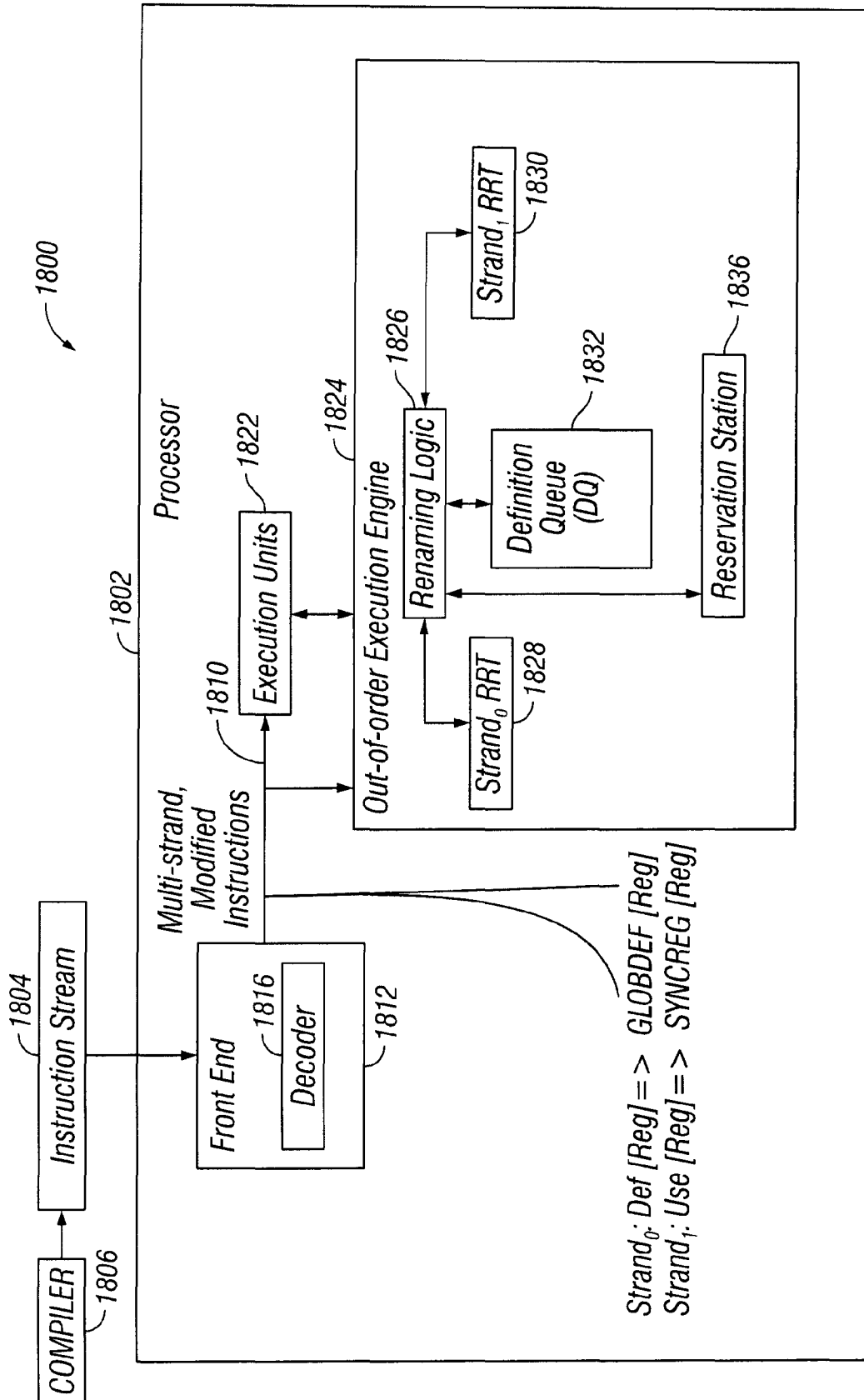


FIG. 18

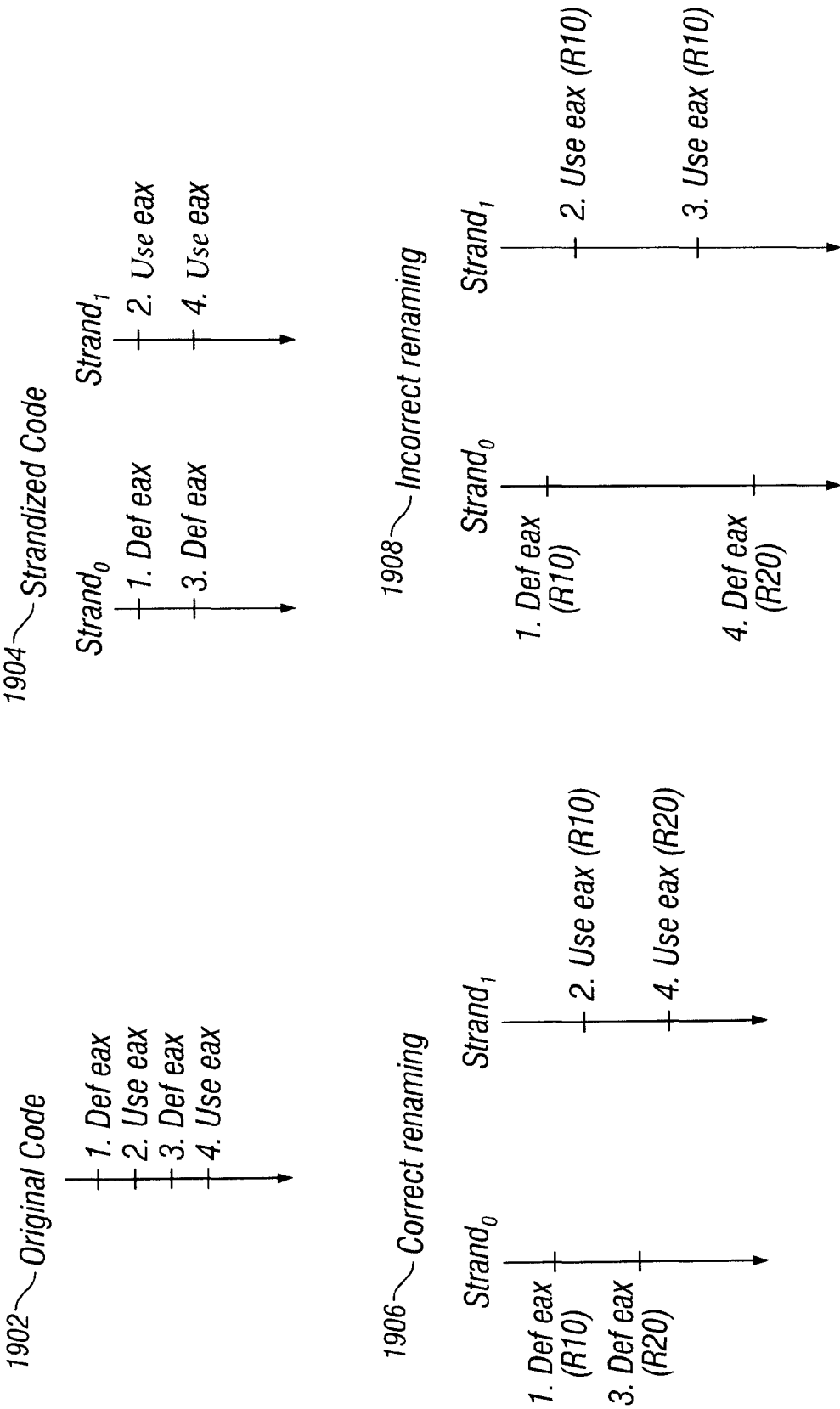


FIG. 19

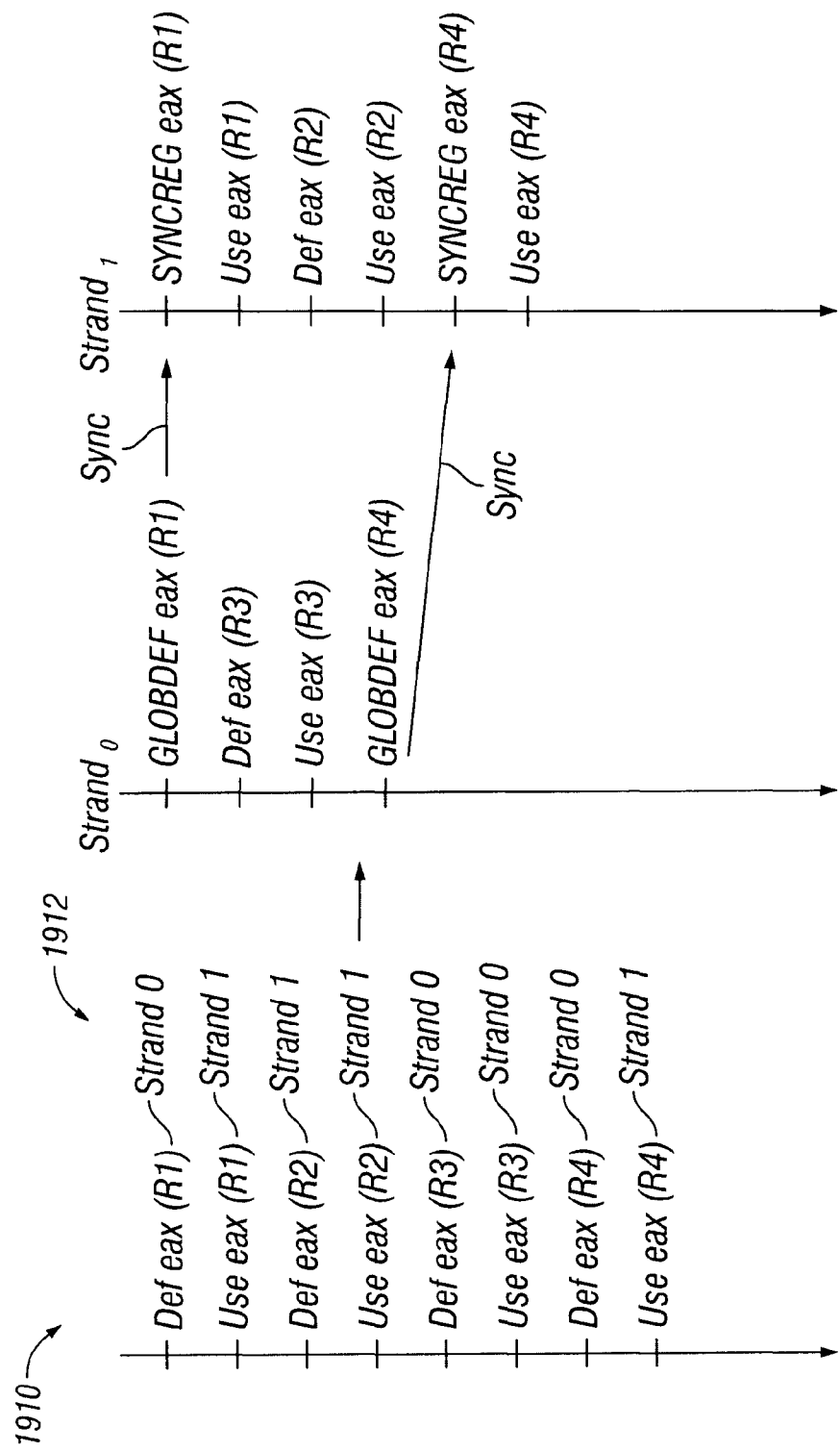


FIG. 19  
(Cont'd)

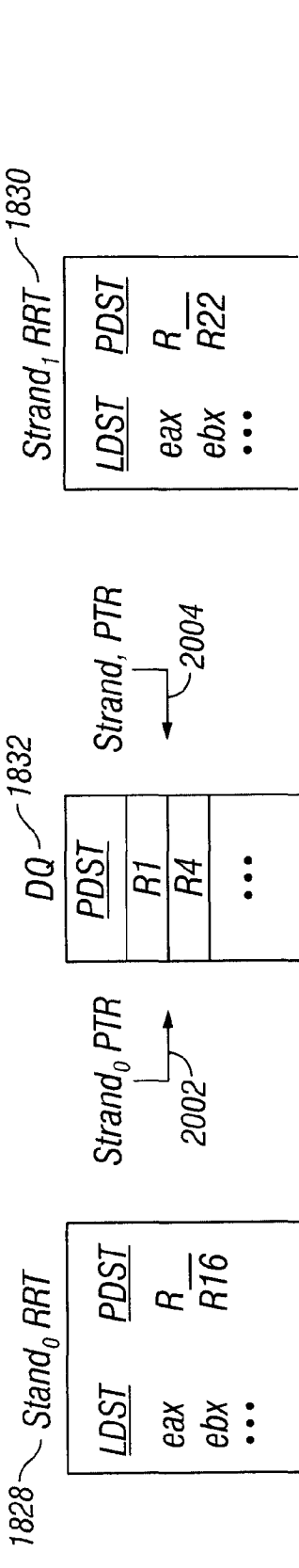


FIG. 20

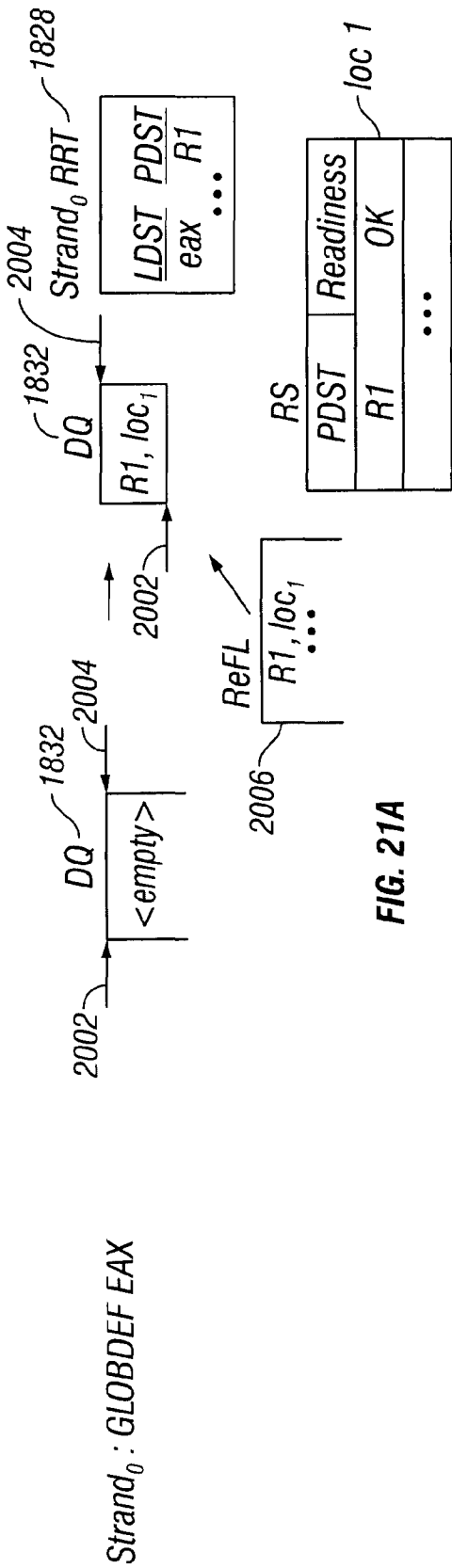


FIG. 21A

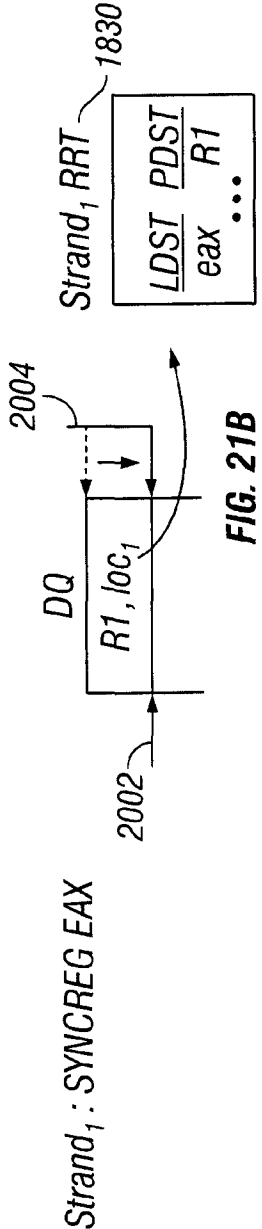


FIG. 21B

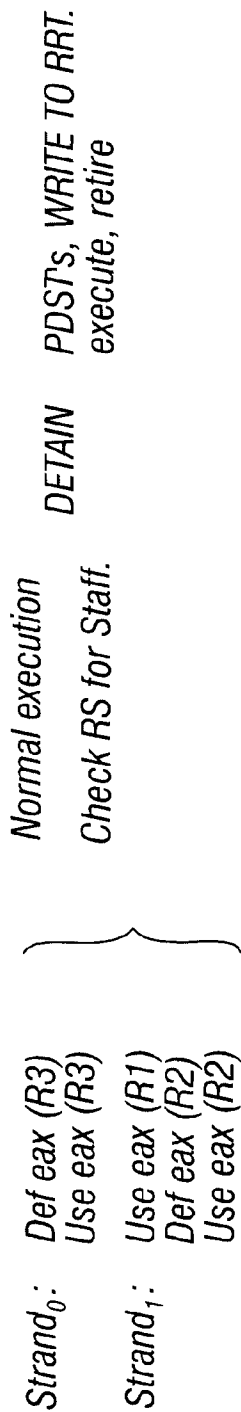


FIG. 21C

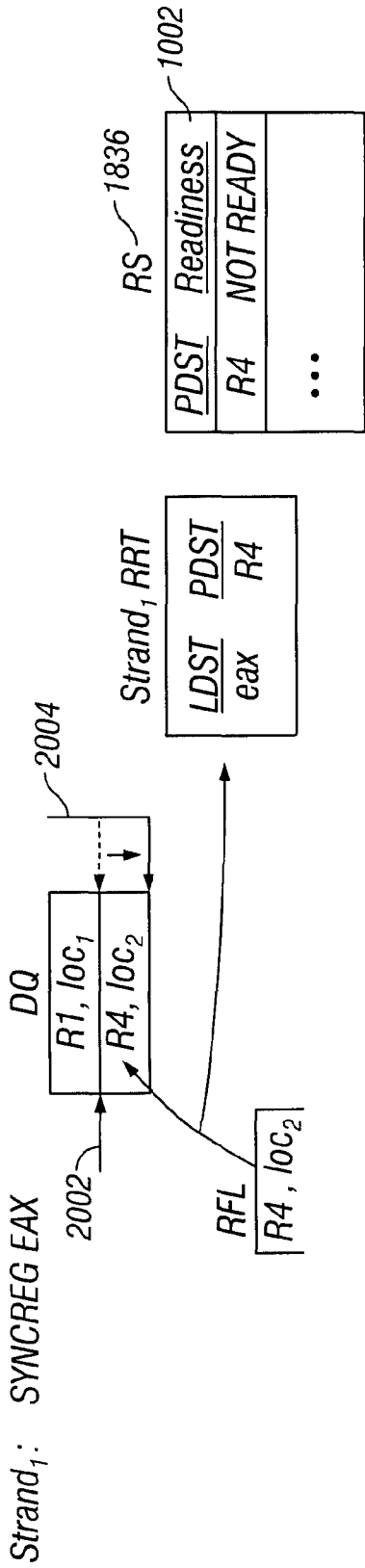
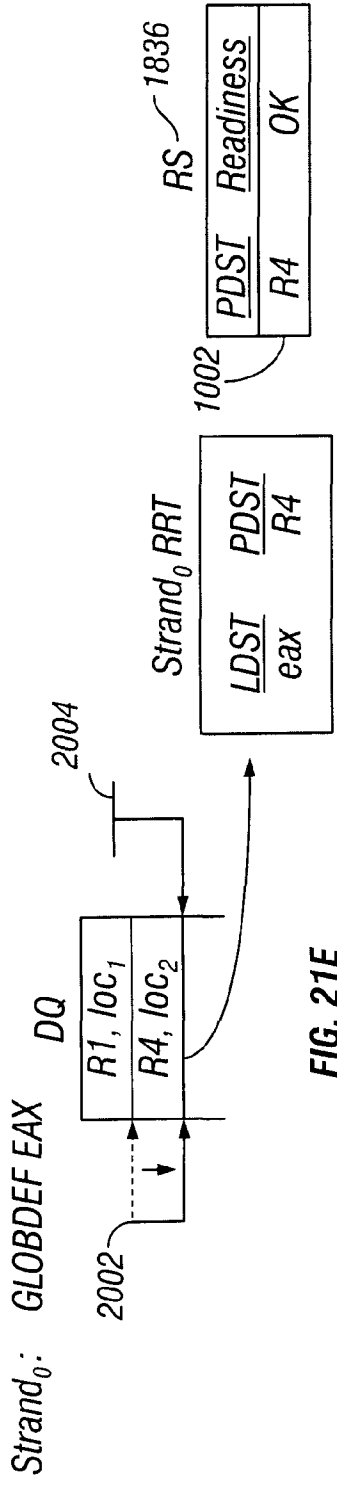


FIG. 21D



29/37

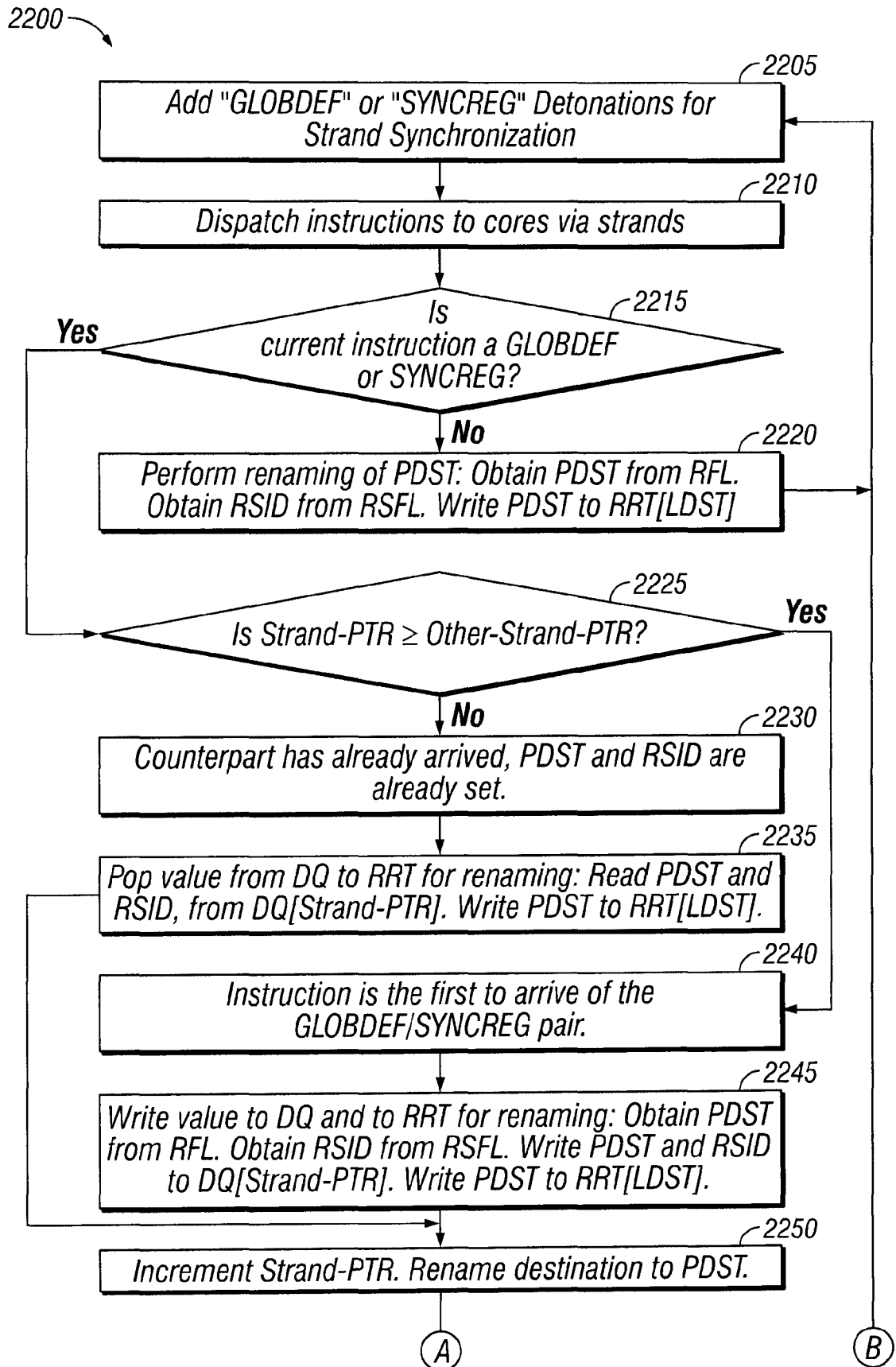


FIG. 22

30/37

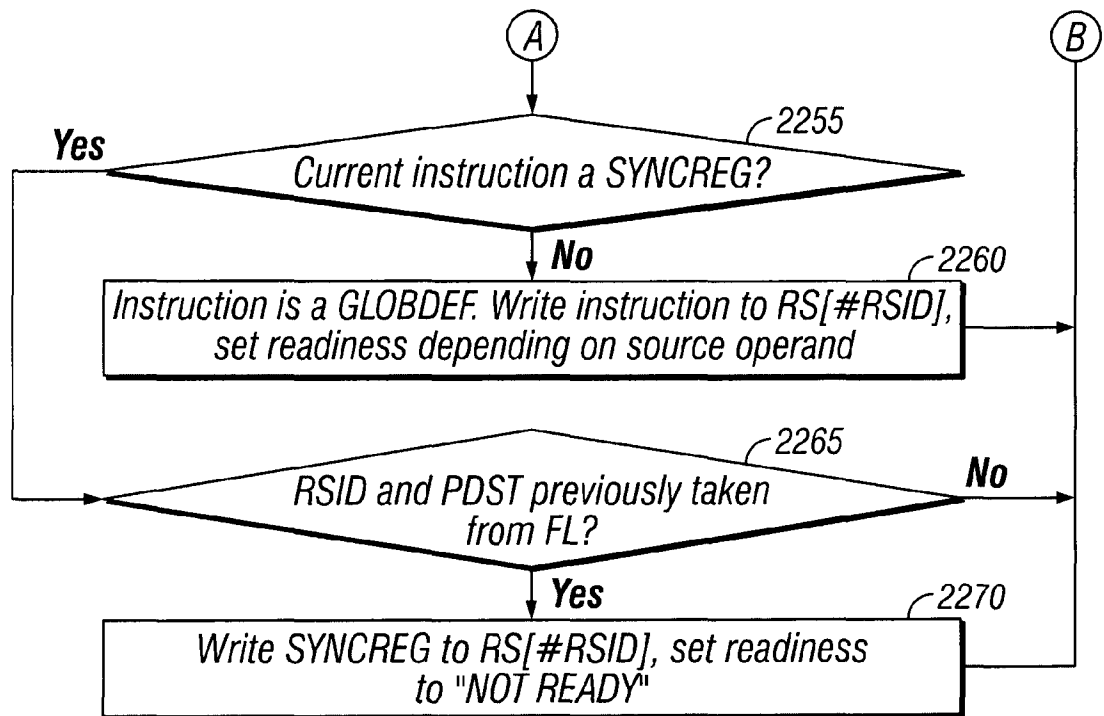


**FIG. 22**  
**(Cont'd)**

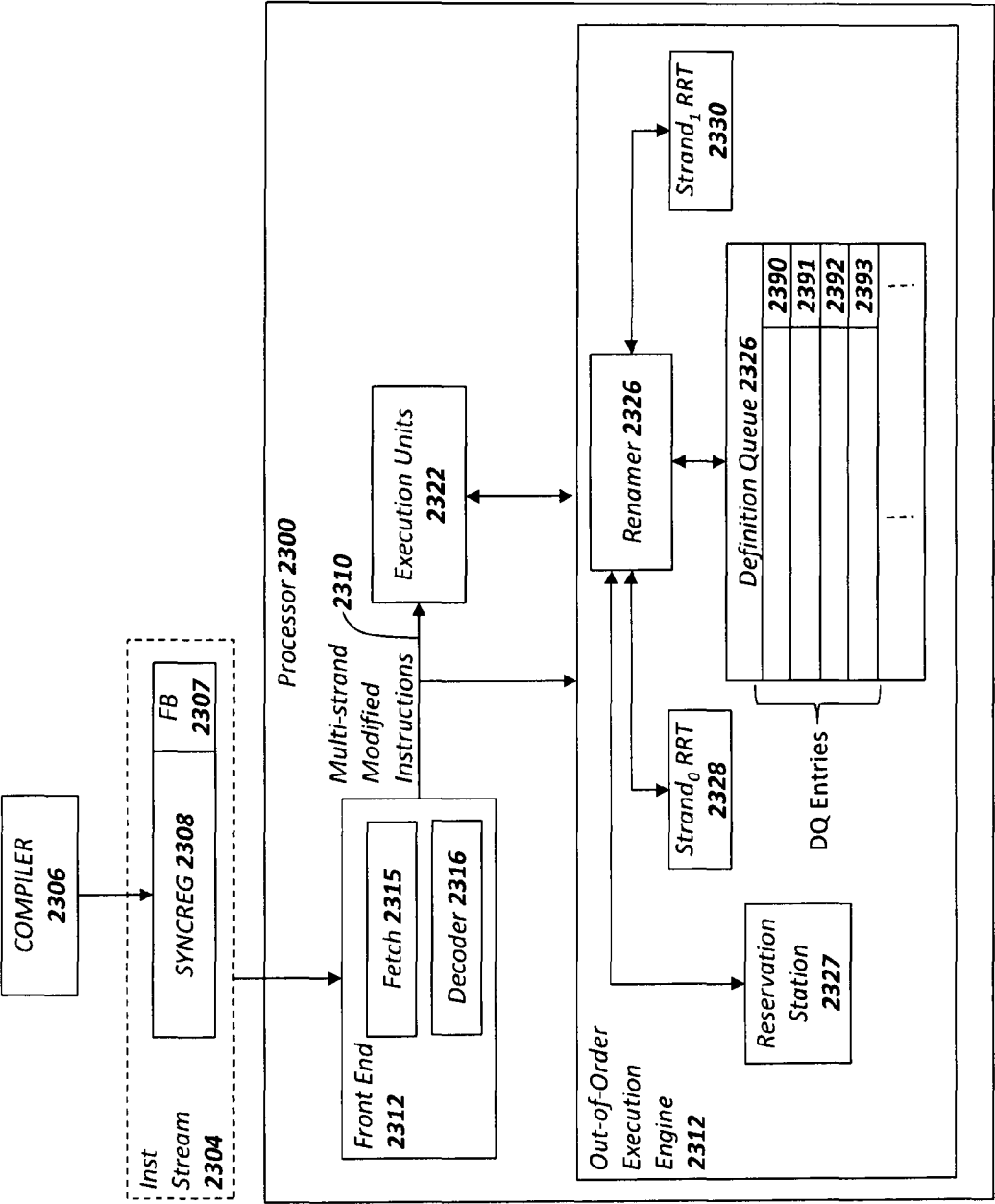


Fig. 23

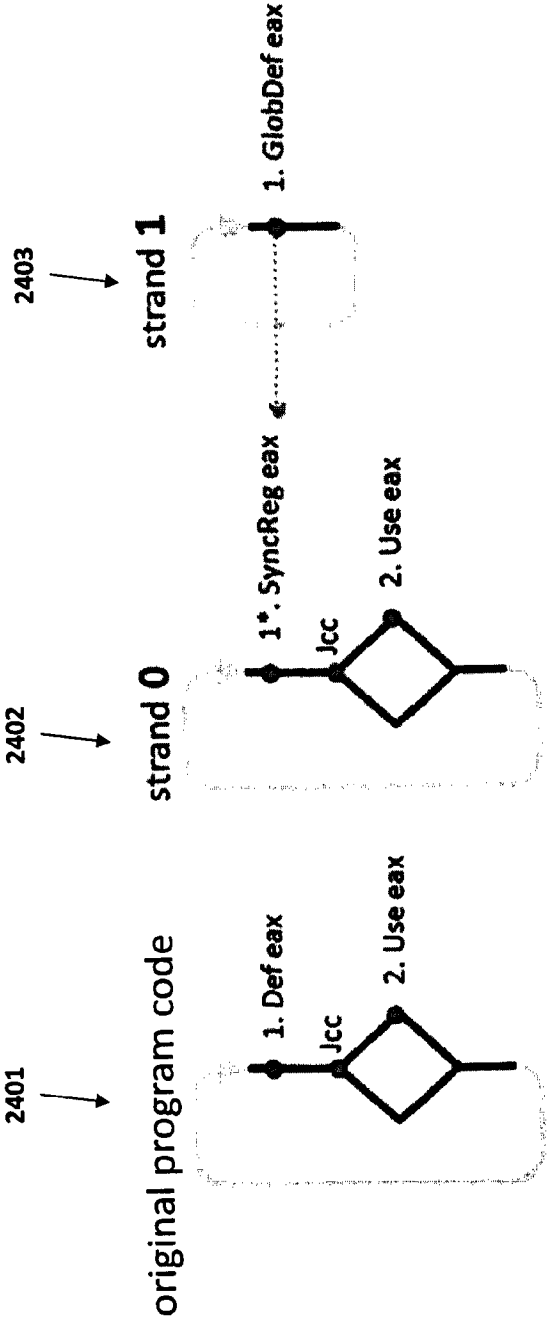


Fig. 24

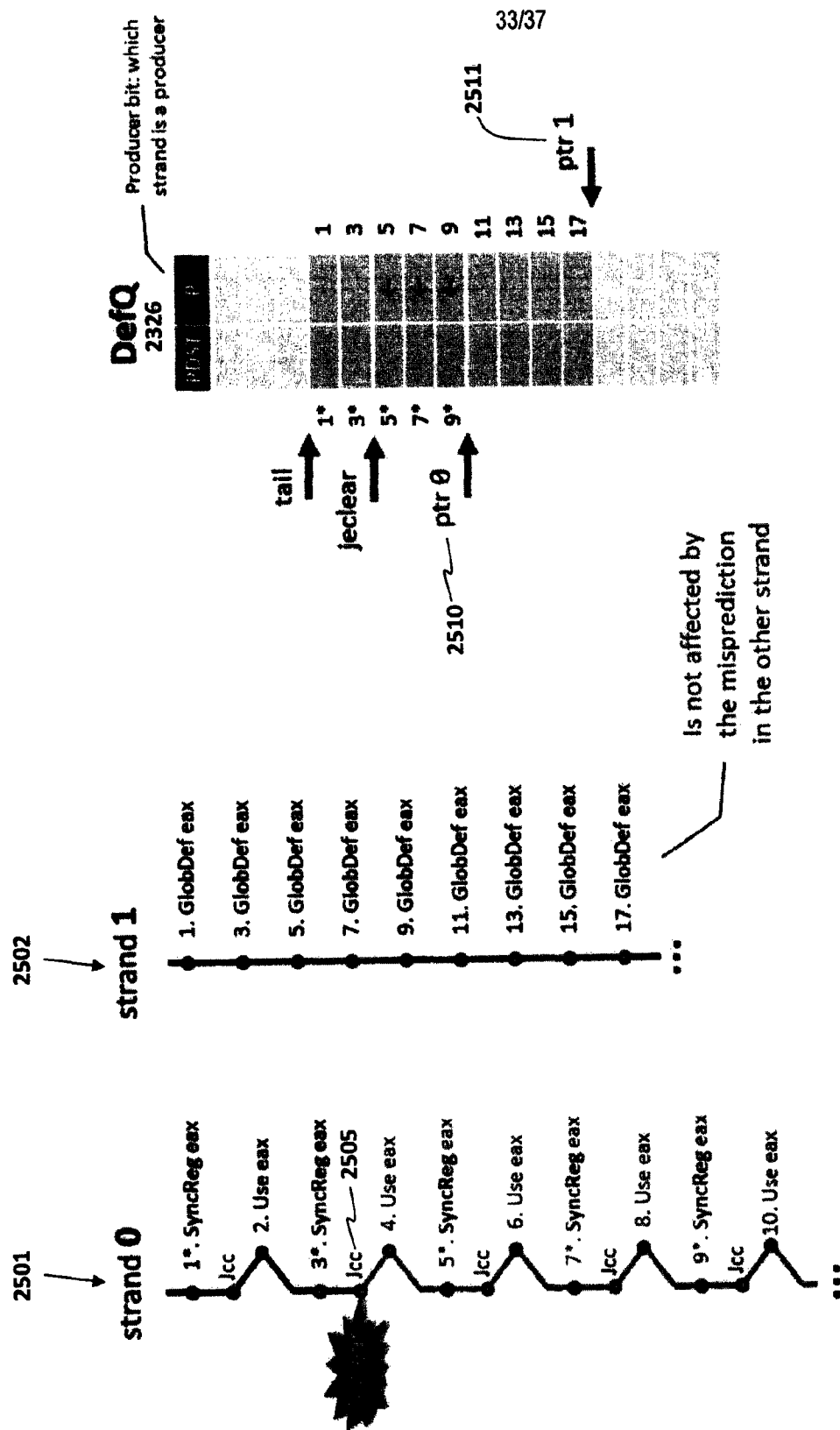


Fig. 25

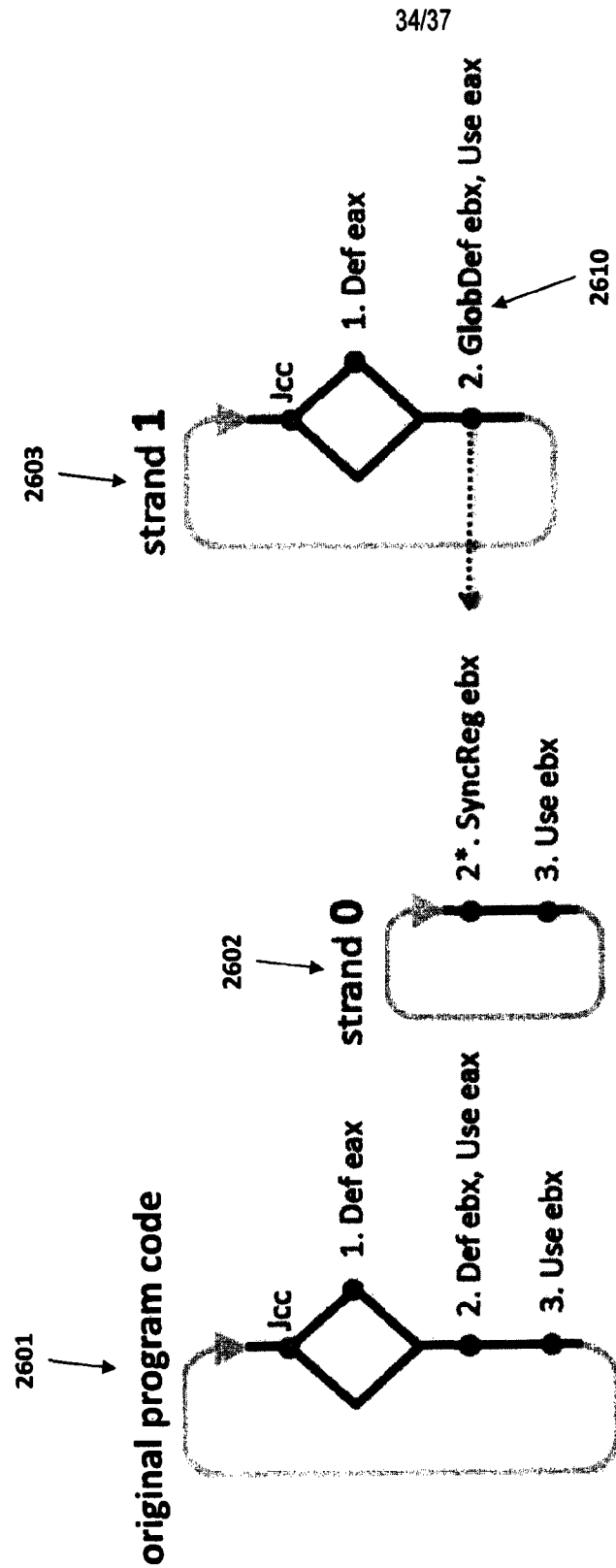


Fig. 26

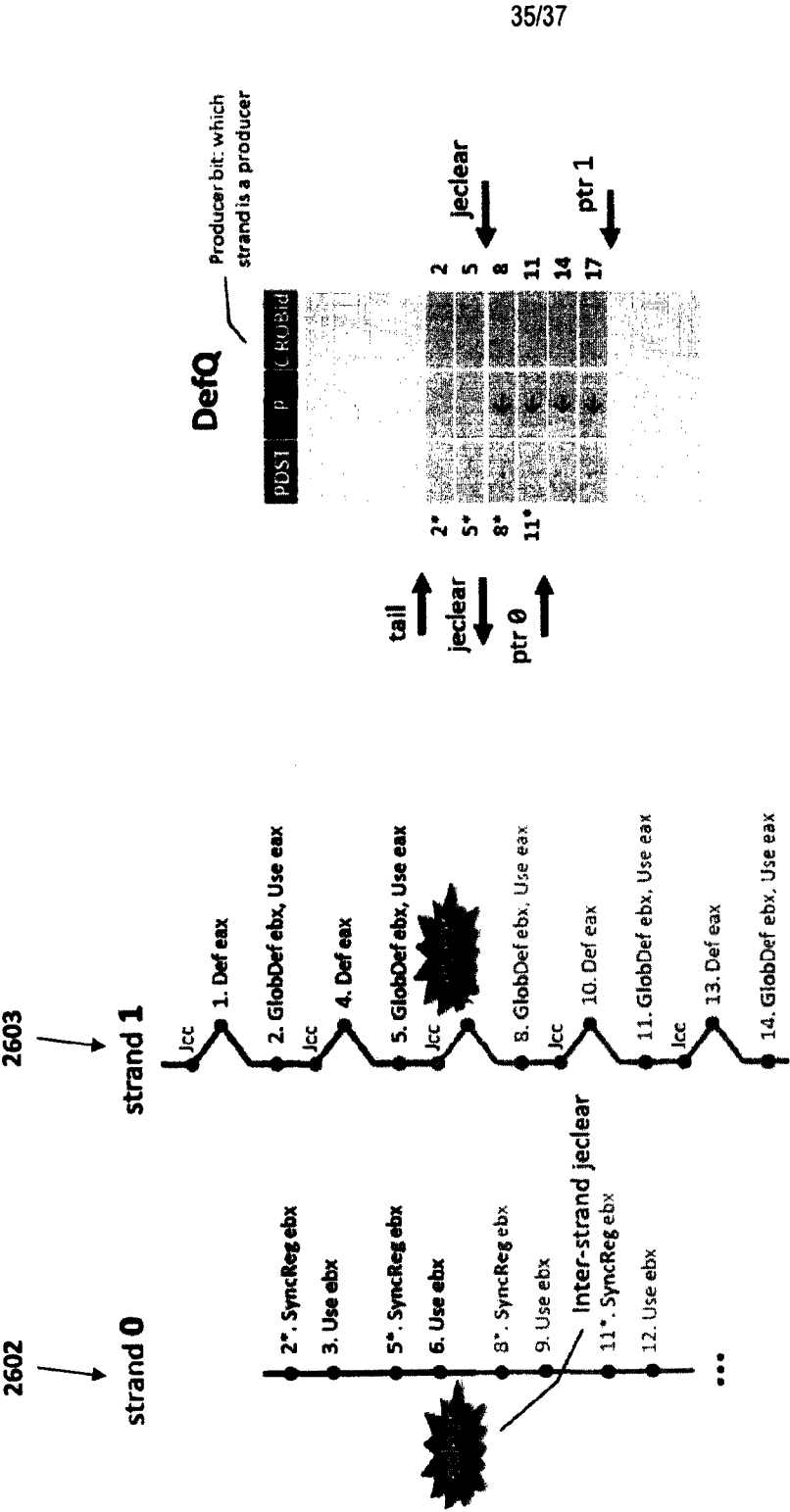


Fig. 27

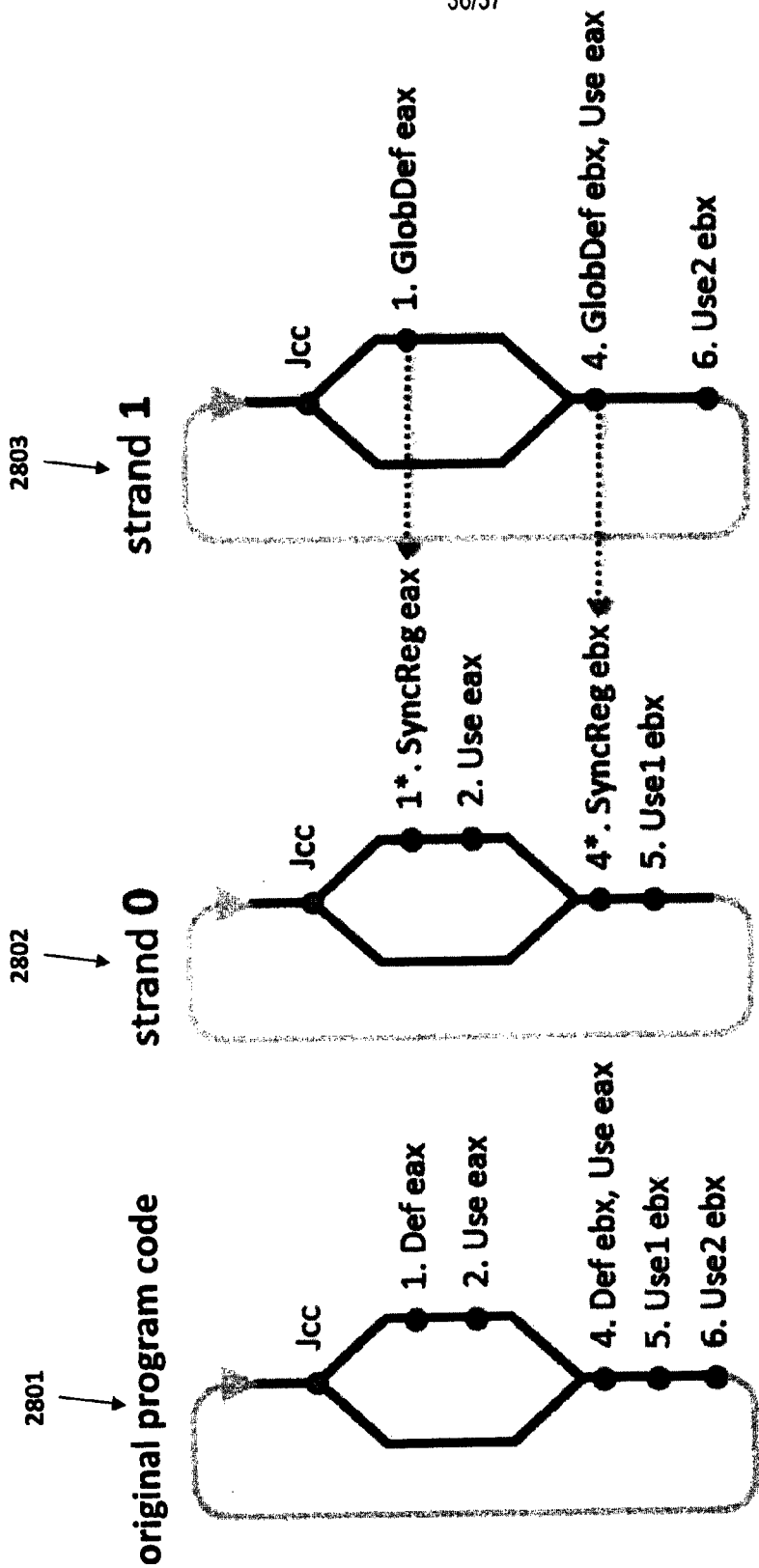


Fig. 28

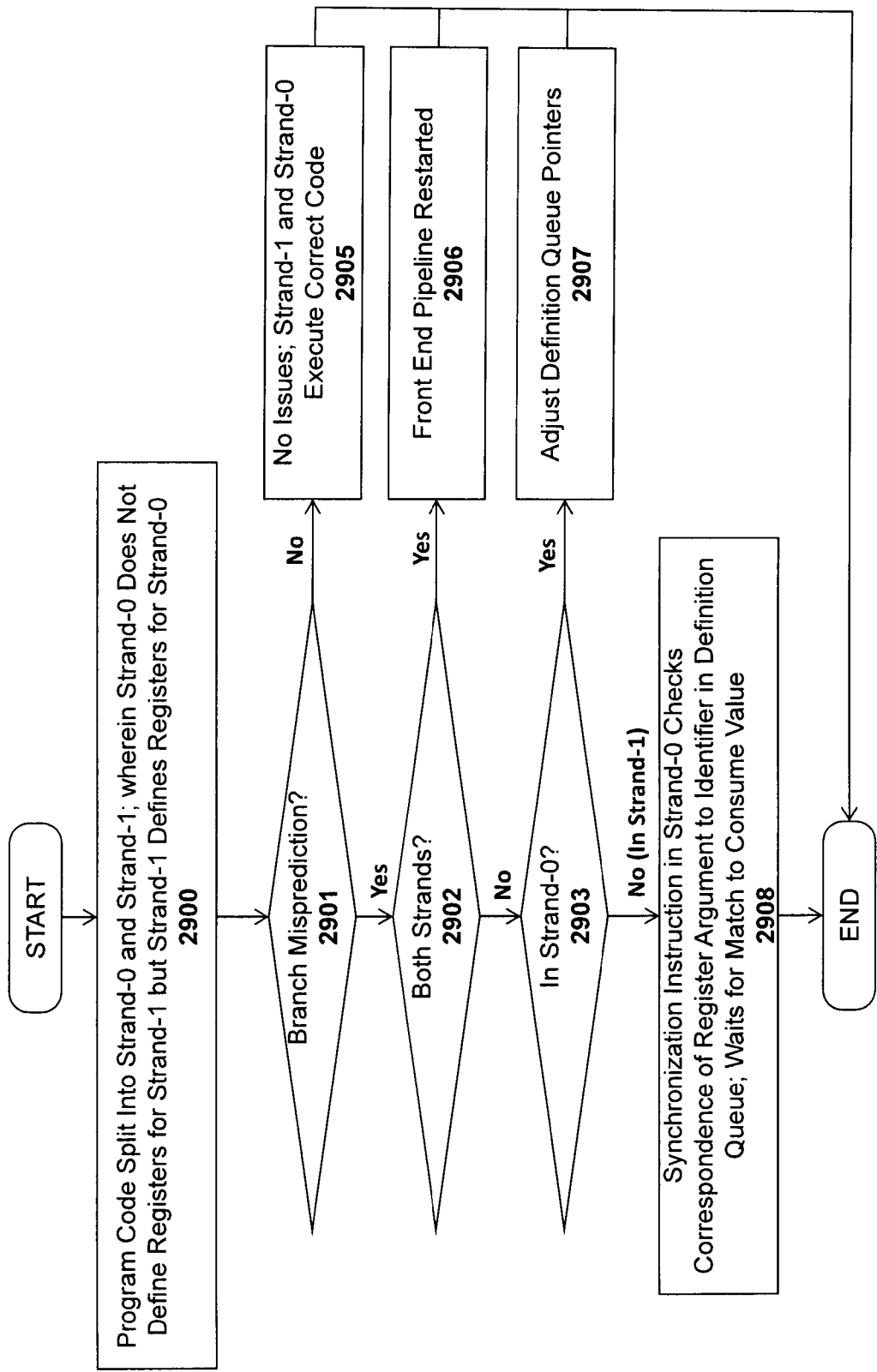


Fig. 29

## INTERNATIONAL SEARCH REPORT

International application No  
PCT/IB2016/000612

A. CLASSIFICATION OF SUBJECT MATTER  
INV. G06F9/30 G06F9/38 G06F9/45  
ADD.

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

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

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

EPO-Internal, WPI Data

## C. DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages                                                                     | Relevant to claim No. |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|
| A         | WO 2013/006566 A2 (INTEL CORP [US]; BABAYAN BORIS A [RU]; PENTKOVSKI VLADIMIR M [US]; BUT) 10 January 2013 (2013-01-10)<br>the whole document<br>----- | 1,10,18               |
| A         | US 2008/016324 A1 (BURKY WILLIAM E [US] ET AL) 17 January 2008 (2008-01-17)<br>the whole document<br>-----                                             | 1,10,18               |
| A         | US 5 961 639 A (MALLICK SOUMMYA [US] ET AL) 5 October 1999 (1999-10-05)<br>the whole document<br>-----                                                 | 1,10,18               |



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

"&" document member of the same patent family

Date of the actual completion of the international search

12 December 2016

Date of mailing of the international search report

22/12/2016

Name and mailing address of the ISA/

European Patent Office, P.B. 5818 Patentlaan 2  
NL - 2280 HV Rijswijk  
Tel. (+31-70) 340-2040,  
Fax: (+31-70) 340-3016

Authorized officer

Klocke, Lynn

# INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/IB2016/000612

| Patent document<br>cited in search report | Publication<br>date | Patent family<br>member(s) | Publication<br>date |
|-------------------------------------------|---------------------|----------------------------|---------------------|
| WO 2013006566 A2                          | 10-01-2013          | TW 201312460 A             | 16-03-2013          |
|                                           |                     | US 2013007415 A1           | 03-01-2013          |
|                                           |                     | WO 2013006566 A2           | 10-01-2013          |
| -----                                     |                     |                            |                     |
| US 2008016324 A1                          | 17-01-2008          | US 2008016324 A1           | 17-01-2008          |
|                                           |                     | US 2009055631 A1           | 26-02-2009          |
| -----                                     |                     |                            |                     |
| US 5961639 A                              | 05-10-1999          | NONE                       |                     |
| -----                                     |                     |                            |                     |