



- (51) **International Patent Classification:**
G06F 9/30 (2006.01) *G06F 9/06* (2006.01)
- (21) **International Application Number:**
PCT/US201 1/062673
- (22) **International Filing Date:**
30 November 2011 (30.11.2011)
- (25) **Filing Language:** English
- (26) **Publication Language:** English
- (71) **Applicant (for all designated States except US):** INTEL CORPORATION [US/US]; 2200 Mission College Boulevard, MS: RNB-4-150, Santa Clara, California 95052 (US).
- (72) **Inventors; and**
- (75) **Inventors/Applicants (for US only):** COMBS, Jonathan, D. [US/US]; 8612 Neider Drive, Austin, Texas 78749 (US). BRANDT, Jason, W. [US/US]; 1106 Maufrains St., Austin, Texas 78703 (US). VALENTINE, Robert [US/IL]; Rechov Hadganiot 33-5, 36054 Kiryat Tivon (IL). SMITH, Kevin, B. [US/US]; 1440 NW 317th Ave., Hillsboro, Oregon 97124 (US). ANSARI, Zia [US/US]; 17632 SW 79th PL., Durham, Oregon 97224 (US). LOKTYUKHIN, Maxim [RU/US]; 922 John Murray Way, Folsom, California 95630 (US).
- (74) **Agent:** SHAO, Kevin, G.; Blakely Sokoloff Taylor & Zafman LLP, 1279 Oakmead Parkway, Sunnyvale, CA 94085 (US).

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

Declarations under Rule 4.17:

— of inventorship (Rule 4.17(iv))

Published:

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

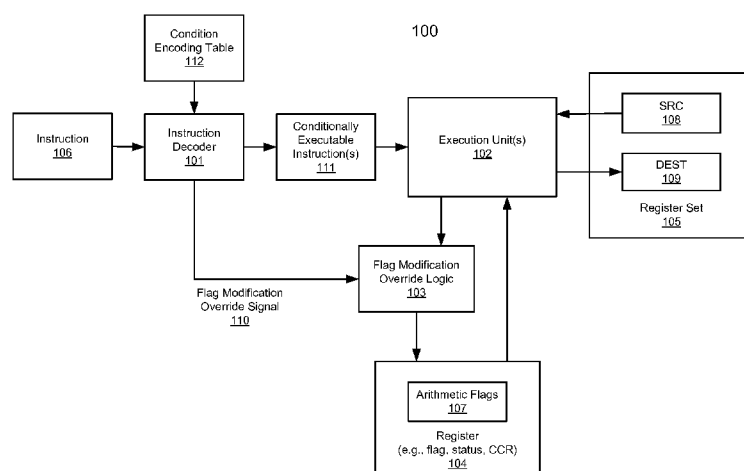
(54) **Title:** CONDITIONAL EXECUTION SUPPORT FOR ISA INSTRUCTIONS USING PREFIXES

FIG. 1

(57) **Abstract:** In one embodiment, a processor includes an instruction decoder to receive a first instruction having a prefix and an opcode and to generate, by an instruction decoder of the processor, a second instruction executable based on a condition determined based on the prefix, and an execution unit to conditionally execute the second instruction based on the condition determined based on the prefix.

CONDITIONAL EXECUTION SUPPORT FOR ISA INSTRUCTIONS USING PREFIXES5 FIELD OF THE INVENTION

Embodiments of the present invention relate generally to instruction processing apparatuses. More particularly, embodiments of the invention relate to an instruction processing apparatus with conditional execution extension for ISA instructions using prefixes.

10

BACKGROUND

Instructions involved in a general purpose register (GPR) such as integer registers are widely used in instruction set architectures (ISA) such as x86 architecture. Most x86 GPR (e.g., integer registers other than x87 or single instruction multiple data (SEVID)/advanced vector extension (AVX) registers) operations write to an integer register as well as manipulate an x86 arithmetic flag register. For example, an addition instruction such as ADD instruction modifies a flag (e.g., OF, SF, ZF, AF, CF, and PF flags) and updates a GPR destination. An increment instruction such as INC also modifies a flag (e.g., OF, SF, ZF, AF, and PF) and updates a GPR destination. Often, this flag modification is unnecessary and yet adds complexity and potential performance implications due to unnecessary dependency chains.

An instruction can also be conditionally executed by a processor based on one or more conditions. Typically, x86 GPR instructions require multiple opcodes or instructions in order to perform a conditional execution, for example, based on one or more test fields as shown in a condition code table in Figure 6. Such a requirement may consume more resources and reduce the performance of a processor.

BRIEF DESCRIPTION OF THE DRAWINGS

Embodiments of the invention are illustrated by way of example and not limitation in the figures of the accompanying drawings in which like references indicate similar elements.

Figure 1 is a block diagram illustrating an example of a processor according one embodiment.

Figure 2 is a block diagram illustrating an example of an instruction according to one embodiment.

Figure 3 is a block diagram illustrating an example of a processor according one embodiment.

5 Figure 4 is a block diagram illustrating an example of a flag register.

Figure 5 is a block diagram illustrating VEX prefix encoding utilized with AVX-based instructions according to one embodiment.

Figure 6 is a block diagram illustrating a conventional condition code table.

10 Figure 7 is a block diagram illustrating a prefix encoding table according to one embodiment.

Figure 8 is a flow diagram illustrating a method to process instructions according to one embodiment.

Figure 9 is a block diagram illustrating an example of a data processing system according to one embodiment.

15 Figure 10 is a block diagram illustrating an example of a data processing system according to another embodiment.

DETAILED DESCRIPTION

20 Various embodiments and aspects of the inventions will be described with reference to details discussed below, and the accompanying drawings will illustrate the various embodiments. The following description and drawings are illustrative of the invention and are not to be construed as limiting the invention. Numerous specific details are described to provide a thorough understanding of various embodiments of the present invention. However, in certain
25 instances, well-known or conventional details are not described in order to provide a concise discussion of embodiments of the present inventions.

Reference in the specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in conjunction with the embodiment can be included in at least one embodiment of the invention. The appearances of the phrase "in one
30 embodiment" in various places in the specification do not necessarily all refer to the same embodiment.

According to some embodiments, an instruction received at a processor includes a prefix and an opcode. One or more bits of the prefix of the instruction are utilized to indicate one or more conditions, which when satisfied, cause the opcode of the instruction to be executed.

Different bit patterns of the prefix may be utilized to specify different conditions for conditional execution of the opcode. The opcode of the instruction may represent an integer general purpose register (GPR) or memory instruction such as an x86 compatible opcode (e.g., x86 legacy opcodes). As a result, a single instruction can include information specifying one or more conditions of conditional execution of an opcode.

In addition, according to one embodiment, a prefix of an instruction is also utilized to provide an indication whether a flag of a register should be modified when executing an opcode of the instruction by a processor. In one embodiment, when an instruction is received at a processor, processing logic such as an instruction decoder, is configured to examine a bit pattern of one or more bits of a prefix to determine whether an opcode of the instruction, which when being executed, should be allowed to modify a flag of a register such as a flag register, a status register, or a condition code register, etc. If the bit pattern matches a predetermined pattern, an execution unit executing the opcode is prevented from modifying a flag that would normally be modified by the execution of the opcode.

In one embodiment, a bit pattern of a prefix of the instruction can be utilized to indicate: 1) one or more conditions which determine whether the opcode of the instruction will be executed; and 2) whether the conditional execution of the opcode will be allowed to modify one or more flags of a register such as a flag register, a status register, and/or a condition code register, etc. Thus, a single instruction having a prefix and an opcode can include all of the information in the prefix to determine one or more conditions for conditionally executing the opcode and/or whether the conditional execution of the opcode can modify one or more flags of a register.

Figure 1 is a block diagram illustrating an example of a processor according one embodiment. Referring to Figure 1, processor 100 may represent any kind of instruction processing apparatuses. For example, processor 101 may be a general-purpose processor. Processor 100 may be any of various complex instruction set computing (CISC) processors, various reduced instruction set computing (RISC) processors, various very long instruction word (VLIW) processors, various hybrids thereof, or other types of processors entirely. In one embodiment, processor 100 includes instruction decoder 101 to receive and decode instruction 106. Instruction decoder 101 may generate and output one or more micro-operations, micro-code, entry points, microinstructions, other instructions, or other control signals, which reflect, or are derived from, instruction 106. Instruction decoder 101 may be implemented using various different mechanisms. Examples of suitable mechanisms include, but are not limited to,

microcode read only memories (ROMs), look-up tables, hardware implementations, programmable logic arrays (PLAs), and the like.

Processor 100 further includes one or more execution units 102, which may include an arithmetic logic unit, or another type of logic unit capable of performing operations based on instruction 106. As a result of instruction decoder 101 decoding instruction 106, execution unit 102 may receive one or more micro-operations, micro-code entry points, microinstructions, other instructions, or other control signals, which reflect, or are derived from, instruction 106.

Execution unit 102 may be operable as a result of instruction 106 indicating one or more source operands (SRC) 108 and to store a result in one or more destination operands (DEST) 109 of register set 105 indicated by instruction 106. Execution unit 102 may include circuitry or other execution logic (e.g., software combined with hardware and/or firmware) operable to execute instructions or other control signals derived from instruction 106 and perform an operation accordingly. Execution unit 102 may represent any kinds of execution units such as logic units, arithmetic logic units (ALUs), arithmetic units, integer units, etc.

In one embodiment, instruction 106 may implicitly indicate and/or explicitly specify (e.g., through one or more dedicated fields or sets of bits) the source and destination operands. Examples of suitable sources and/or destination of the operands include registers, memory, immediate of the instruction, and combinations thereof. In various embodiments, the source and destination operands may be 8-bit, 16-bit, 32-bit, or 64-bit operands, although this is not required.

In one embodiment, some or all of the source and destination operands may be stored in registers of a register set 105 or memory. The register set may be part of a register file, along with potentially other registers, such as status registers, flag registers, etc. A register may be a storage location or device that may be used to store data. The register set may often be physically located on die with the execution unit(s). The registers may be visible from the outside of the processor or from a programmer's perspective. For example, instructions may specify operands stored in the registers. Various different types of registers are suitable, as long as they are capable of storing and providing data as described herein. The registers may or may not be renamed. Examples of suitable registers include, but are not limited to, dedicated physical registers, dynamically allocated physical registers using register renaming, combinations of dedicated and dynamically allocated physical registers, etc. Alternatively, one or more of the source and destination operands may be stored in a storage location other than a register, such as, for example, a location in system memory.

Referring back to Figure 1, in one embodiment, instruction 106 may include at least one prefix and an opcode as shown in Figure 2. Referring to Figures 1 and 2, instruction 200 of Figure 2 may represent instruction 106 of Figure 1. In one embodiment, instruction 200 includes prefix 201 and opcode 202. When instruction 200 is received by instruction decoder 101, according to one embodiment, instruction decoder 101 is configured to generate one or more microcode operations or microinstructions based on opcode 202 to be executed by execution unit 102. In addition, instruction decoder 101 is configured to examine one or more bits of prefix 201 to determine one or more conditions upon which the microcode operations or microinstructions associated with opcode 202 should be executed by execution unit 102. Furthermore, instruction decoder 101 is configured to examine one or more bits of prefix 201 to determine whether execution unit 102 can modify at least one of flags 107 of register 104 as a result of execution (e.g., conditional execution) of the microcode operations or microinstructions .

In one embodiment, when instruction 106 is received by instruction decoder 101, instruction decoder 101 is configured to extract the prefix such as prefix 201 of Figure 2 from instruction 106. Instruction decoder 101 is then to determine a condition upon which the opcode of instruction 106 is to be executed based on the prefix. In one embodiment, the condition is determined by looking up condition encoding table 112, which maps certain prefixes to certain conditions. Different bit patterns of the prefixes may represent different conditions. Condition encoding table 112 may be implemented in hardware, firmware, software, or a combination thereof. Based on the condition or conditions derived from the prefix, instruction decoder 101 is configured to generate one or more conditional executable microcode operations or microinstructions 111 to be conditionally executed by execution unit 102.

In addition, instruction decoder 101 further examines the prefix of instruction 106 to determine as a result of the conditional execution whether execution unit 102 is allowed to modify any of flags 107 of register 104. According to one embodiment, when a bit pattern of one or more bits of prefix 201 matches a predetermined pattern, instruction decoder 101 transmits a flag modification override (FMO) signal 110 to FMO logic 103. FMO logic 103 may prevent the flags from being modified for that operation. If the bit pattern of one or more bits of prefix 201 does not match the predetermined bit pattern, FMO logic 103 may allow execution unit 102 to modify flags 107. Information stored in prefix 201 and/or opcode 202 may be populated by a compiler and/or a programmer. FMO logic 103 may be implemented in hardware, firmware, software, or a combination thereof. Note that a particular bit pattern of prefix 201 may indicate that a particular flag can be modified while other flags may not be

modifiable. Thus, the bit pattern of prefix 201 dictates which of the flags 107 can be modified by execution unit 102 and the condition or conditions which must be true for opcode 202 to be executed. As a result, a single instruction can specify both the condition and the indication whether to override the modification attempt to a flag. Conventional systems typically require multiple instructions or opcodes for conditional executions. Also note that the format of instruction 200 of Figure 2 has been shown and described for the purpose of illustration only; other formats may also be applied.

In one embodiment, prior to generating microcode operations or microinstructions 111 from the opcode, instruction decoder 101 is configured to examine, given an opcode provided in opcode 202, whether prefix 201 having a particular bit pattern is valid. If it is determined that the prefix 201 is invalid based on opcode 202, an exception is generated and the opcode will not be executed. Furthermore, instruction decoder 101 may also examine whether instruction 200 is valid given a capability of processor 100, which may be determined based on a processor identifier such as a CPUID as shown in Figure 3. Referring to Figure 3, based on information such as prefix 201, opcode 202, and/or processor ID 203, instruction decoder 101 is configured to generate exception 204 (e.g., undefined or UD operation) if the instruction is determined to be invalid. Otherwise, instruction decoder 101 is configured to generate FMO signal 110 and conditional executable microcode operations or microinstruction(s) 111 based on prefix 201, opcode 202, and condition encoding table 112. Referring back to Figure 1, based on FMO signal 110, FMO logic 103 is configured to control whether the flags 107 of register 104 or modified for that operation, where FMO logic 103 may or may not allow such a modification dependent upon the specific prefix of instruction 106.

In one embodiment, flags 107 may represent one or more of arithmetic flags. Register 104 may be a status register, a flags register (e.g., EFLAGS), or a condition code register (CCR). An example of register 104 is shown in Figure 4, which represents an example of the EFLAGS register. Flags 107 may include one or more bits, often in a register such as register 104. Flags 107 do not have to be bits in a register. A flag may include a flip-flop circuit. A bit flag may store a binary value, such as a binary one or a binary zero. The bits may store a sequence of binary values or a code. Such binary values, or sequences of binary values, may represent different values of the flag. These different values of the flag may have an assigned or understood meaning to processor 100 or other logic device. Flags 107 may be used for various different purposes. In some cases, flags 107 may be used to record a status, state, condition, or outcome of operations within processor 100 or other logic device. Flags 107 may include a carry flag (CF), an overflow flag (OF), a sign flag (SF), a zero flag (ZF), an auxiliary carry flag

(AF), or a parity flag (PF), or a combination thereof, as shown in Figure 4. Not all of such flags are included in all processors or other logic devices. Additionally, different names are sometimes used for these different flags.

According to one embodiment, instruction 106 is compatible with the advanced vector extension (AVX)-based ISA of Intel Corporation of Santa Clara, California. An embodiment utilizes an AVX prefix of an AVX-based instruction to encode a "flag non-modifying" indication to the ISA. Being done via a prefix allows this extension to be applied broadly to the existing legacy set of x86 GPR instruction opcodes. This can be applied to a variety of modes such as 16-bit, 32-bit, or 64-bit mode. By adding the flag non-modifying indication in the prefix space, embodiments allow a rich set of flag non-modifying instructions to be generated while utilizing mostly pre-existing legacy hardware with only slight modifications. In one embodiment, one or more AVX prefix bits are utilized to denote flag non-modification to any or all legacy integer instruction opcodes that write both an integer GPR or memory destination and update one or more arithmetic flag bits. Such a prefix prevents new opcodes and corresponding decoder hardware from being required whenever a flag non-modifying version of an existing legacy instruction is desired. In addition, according to one embodiment, the AVX prefixes can also be utilized to specify one or more conditions for conditional execution of certain instructions such as x86 integer GPR or memory instructions.

Figure 5 is a block diagram illustrating VEX prefix encoding utilized with AVX-based instructions according to one embodiment. Referring to Figure 5, AVX-based prefixes include a 3-byte C4 VEX encoding 510, which may be implemented as part of prefix 201 of Figure 2. According to one embodiment, a combination of multiple bits in the VEX prefix encoding may be utilized for conditional integer execution and flag non-modification extension. Referring back to Figure 5, certain bits that are not utilized in typical vector instructions such as "mmmmm" bits 502 may be utilized to indicate whether a flag should be modified by an execution unit. In this embodiment, upper three bits 503 of bits 502 are not utilized while bits 504 are utilized in vector instructions. As a result, certain bit patterns of a combination of bits 503 and/or L bit 501 can be utilized to specify one or more conditions for conditional execution and to indicate whether a flag should be modified.

In one embodiment, bits 503 in combination of L bit 501 may also be utilized to specify one or more conditions for conditional execution and to indicate whether a flag should be modified, as shown in Figure 2. The combined bit patterns of bits 503 and L bit 501 are utilized to represent various conditions for conditional executions as shown in Figure 7, which may represent condition encoding table 112 of Figure 1. The condition encoding table as shown in

Figure 7 represents most of the conditions in a legacy condition code table as shown in Figure 6 that typically requires multiple instructions or opcodes. In this embodiment, three currently reserved bits 503 in C4 AVX prefix 510 along with L bit 501 are utilized to enable the existing GPR x86 opcode space with the capability to support a rich set of conditional execution options and flag non-modification, and to take advantage of the existing ternary abilities VEX provides. In one embodiment, the flag non-modification override applies only to instructions that update both a GPR register and a flag register; it may not apply to instructions that modify only the flag register.

X86 instructions that utilize condition codes as shown in Figure 6 consume four bits to specify the conditions that are used to evaluate the x86 arithmetic flags. An embodiment of the invention is to replicate the conditions as shown in Figure 6 for conditional execution behavior by consuming three currently reserved bits in the C4 VEX encoding and repurposing the L bit in the C4 VEX encoding to enable nearly all the same conditions present in the legacy conditional test fields as shown in Figure 6. The "ccc" bits are currently reserved in a conventional system and non-zero values will cause a fault (e.g., #UD). A non-zero value for L in the convention system for the new integer GPR instructions that utilize VEX will cause a fault. With this extension, according to one embodiment, these bits would take on meaning for those integer GPR instructions that can be extended. The encodings can be resolved based on the encoding table as shown in Figure 7.

In one embodiment, x86 legacy opcodes that manipulate only the arithmetic flag register and have no other integer register destination may be treated differently than instructions that write both integer and flag registers while still fitting within the VEX prefix framework. Predication implies flag non-modification for instructions which update integer registers but predication does not need to imply flag non-modification for those x86 instructions that update only arithmetic flags. CMP instruction could allow a conditional execution encoding. For these instructions, the predicate applies to the flag modification itself allowing a predicated TRUE instruction to update the arithmetic flags and a predicated FALSE instruction to not update the flags.

Figure 8 is a flow diagram illustrating a method to process instructions according to one embodiment. For example, method 800 may be performed by instruction decoder 101 of Figure 1. Referring to Figure 8, at block 801, in response to a first instruction received at a processor, prefix and opcode are extracted from the first instructions. At block 802, processing logic is to examine one or more bits of the prefix in view of the opcode and capability of the processor (e.g., CPUID) to determine whether the prefix bits are valid. If not, at block 806, an exception is

generated. If the prefix or the first instruction is valid, at block 803, processing logic generates a second instruction (e.g., one or more microcode operations or microinstructions) to be conditionally executable by an execution unit based on one or more conditions determined based on the prefix. At block 804, processing logic examines one or more bits of the prefix to
5 determine whether the bits match a predetermined bit pattern. If so, at block 805, processing logic overrides an attempt by an execution unit to modify a flag as a result of execution of the microcode operations or microinstructions.

Figure 9 is a block diagram illustrating an example of a data processing system according to one embodiment of the invention. Processor 901 of system 900 may represent processor 100
10 of Figure 1 as described above. System 900 may represent a desktop, a laptop, a tablet, a server, a mobile phone (e.g., Smartphone), a media player, a personal digital assistant (PDA), a personal communicator, a gaming device, a network router or hub, a wireless access point or repeater, a set-top box, or a combination thereof. Note that while Figure 9 illustrates various components of a data processing system, it is not intended to represent any particular architecture or manner of
15 interconnecting the components; as such details are not germane to embodiments of the present invention. It will also be appreciated that network computers, handheld computers, mobile phones, and other data processing systems which have fewer components or perhaps more components may also be used with embodiments of the present invention.

Referring to Figure 9, in one embodiment, system 900 includes processor 901 and
20 chipset 902 to couple various components to processor 901 including memory 905 and devices 903-904 via a bus or an interconnect. Processor 901 may represent a single processor or multiple processors with a single processor core or multiple processor cores 909 included therein. Processor 901 may represent one or more general-purpose processors such as a microprocessor, a central processing unit (CPU), or the like. More particularly, processor 901
25 may be a complex instruction set computing (CISC) microprocessor, reduced instruction set computing (RISC) microprocessor, very long instruction word (VIJW) microprocessor, or processor implementing other instruction sets, or processors implementing a combination of instruction sets. Processor 901 may also be one or more special-purpose processors such as an application specific integrated circuit (ASIC), a field programmable gate array (FPGA), a digital
30 signal processor (DSP), a network processor, a graphics processor, a network processor, a communications processor, a cryptographic processor, a co-processor, an embedded processor, or any other type of logic capable of processing instructions. For example, processor 901 may be a Pentium® 4, Pentium® Dual-Core, Core™ 2 Duo and Quad, Xeon™, Itanium™, XScale™, Core™ i7, Core™ i5, Celeron®, or StrongARM™ microprocessor available from Intel

Corporation of Santa Clara, California. Processor 901 is configured to execute instructions for performing the operations and steps discussed herein.

Processor 901 may include an instruction decoder, which may receive and decode a variety of instructions. The decoder may generate and output one or more micro-operations, micro-code entry points, microinstructions, other instructions, or other control signals, which reflect, or are derived from, an original input instruction. The decoder may be implemented using various different mechanisms. Examples of suitable mechanisms include, but are not limited to, microcode read only memories (ROMs), look-up tables, hardware implementations, programmable logic arrays (PLAs), and the like.

The decoder may not be a required component of processor 901. In one or more other embodiments, processor 901 may instead have an instruction emulator, an instruction translator, an instruction morpher, an instruction interpreter, or other instruction conversion logic. Various different types of instruction emulators, instruction morphers, instruction translators, and the like, are known in the arts. The instruction conversion logic may receive the bit range isolation instruction, emulate, translate, morph, interpret, or otherwise convert the bit range isolation instruction, and output one or more instructions or control signals corresponding to the original bit range isolation instruction. The instruction conversion logic may be implemented in software, hardware, firmware, or a combination thereof. In some cases, some or all of the instruction conversion logic may be located off-die with the rest of the instruction processing apparatus, such as a separate die or in a system memory. In some cases, the instruction processing apparatus may have both the decoder and the instruction conversion logic.

Processor 901 and/or cores 909 may further include one or more execution units coupled with, or otherwise in communication with, an output of the decoder. The term "coupled" may mean that two or more elements are in direct electrical contact or connection. However, "coupled" may also mean that two or more elements are not in direct connection with each other, but yet still co-operate or interact or communicate with each other (e.g., through an intervening component). As one example, the decoder and the execution unit may be coupled with one another through an intervening optional buffer or other component(s) known in the arts to possibly be coupled between a decoder and an execution unit. Processor 901 and/or cores 909 may further include multiple different types of execution units, such as, for example, arithmetic units, arithmetic logic units (ALUs), integer units, etc.

Processor 901 may further include one or more register files including, but are not limited to, integer registers, floating point registers, vector or extended registers, status registers, and an instruction pointer register, etc. The term "registers" is used herein to refer to the on-

board processor storage locations that are used as part of macro-instructions to identify operands. In other words, the registers referred to herein are those that are visible from the outside of the processor (from a programmer's perspective). However, the registers should not be limited in meaning to a particular type of circuit. Rather, a register need only be capable of storing and providing data, and performing the functions described herein. The registers described herein can 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 or 64-bit integer data. A register file may contain extended multimedia SIMD registers (e.g., XMM) for packed data. Such registers may include 128 bits wide XMM registers and 256 bits wide registers (which may incorporate the XMM registers in their low order bits) relating to SSE2, SSE3, SSE4, GSSE, and beyond (referred to generically as "SSEx") technology to hold such packed data operands.

Processor 901 and/or cores 909 may also optionally include one or more other well-known components. For example, processor 901 may optionally include instruction fetch logic, pre-decode logic, scheduling logic, re-order buffers, branch prediction logic, retirement logic, register renaming logic, and the like, or some combination thereof. These components may be implemented conventionally, or with minor adaptations that would be apparent to those skilled in the art based on the present disclosure. Further description of these components is not needed in order to understand the embodiments herein, although further description is readily available, if desired, in the public literature. There are literally numerous different combinations and configurations of such components known in the arts. The scope is not limited to any known such combination or configuration. Embodiments may be implemented either with or without such additional components.

Chipset 902 may include memory control hub (MCH) 910 and input output control hub (ICH) 911. MCH 910 may include a memory controller (not shown) that communicates with a memory 905. MCH 910 may also include a graphics interface that communicates with graphics device 912. In one embodiment of the invention, the graphics interface may communicate with graphics device 912 via an accelerated graphics port (AGP), a peripheral component interconnect (PCI) express bus, or other types of interconnects. ICH 911 may provide an interface to I/O devices such as devices 903-904. Any of devices 903-904 may be a storage device (e.g., a hard drive, flash memory device), universal serial bus (USB) port(s), a keyboard, a mouse, parallel port(s), serial port(s), a printer, a network interface (wired or wireless), a

wireless transceiver (e.g., WiFi, Bluetooth, or cellular transceiver), a media device (e.g., audio/video codec or controller), a bus bridge (e.g., a PCI-PCI bridge), or a combination thereof.

MCH 910 is sometimes referred to as a Northbridge and ICH 911 is sometimes referred to as a Southbridge, although some people make a technical distinction between them. As used
5 herein, the terms MCH, ICH, Northbridge and Southbridge are intended to be interpreted broadly to cover various chips whose functions include passing interrupt signals toward a processor. In some embodiments, MCH 910 may be integrated with processor 901. In such a configuration, chipset 902 operates as an interface chip performing some functions of MCH 910 and ICH 911, as shown in Figure 10. Furthermore, graphics accelerator 912 may be integrated
10 within MCH 910 or processor 901.

Memory 905 may store data including sequences of instructions that are executed by processor 901, or any other device. For example, executable code 913 and/or data 914 of a variety of operating systems, device drivers, firmware (e.g., input output basic system or BIOS), and/or applications can be loaded in memory 905 and executed by processor 901. An operating
15 system can be any kind of operating systems, such as, for example, Windows® operating system from Microsoft®, Mac OS®/iOS® from Apple, Android® from Google®, Linux®, Unix®, or other real-time operating systems. In one embodiment, memory 905 may include one or more volatile storage (or memory) devices such as random access memory (RAM), dynamic RAM (DRAM), synchronous DRAM (SDRAM), static RAM (SRAM), or other types of storage devices.

Nonvolatile memory may also be utilized such as a hard disk or a flash storage device. Front
20 side bus (FSB) 906 may be a multi-drop or point-to-point interconnect. The term FSB is intended to cover various types of interconnects to processor 901. Chipset 902 may communicate with other devices such as devices 903-904 via point-to-point interfaces. Bus 906 may be implemented as a variety of buses or interconnects, such as, for example, a quick path
25 interconnect (QPI), a hyper transport interconnect, or a bus compatible with advanced microcontroller bus architecture (AMBA) such as an AMBA high-performance bus (AHB).

Cache 908 may be any kind of processor cache, such as level-1 (L1) cache, L2 cache, L3
cache, L4 cache, last-level cache (LLC), or a combination thereof. Cache 908 may be shared
30 with processor cores 909 of processor 901. Cache 908 may be embedded within processor 901 and/or external to processor 901. Cache 908 may be shared amongst cores 909. Alternatively, at least one of cores 909 further includes its own local cache (e.g., caches 908A-908N) embedded therein. At least one of cores 909 may utilize both the local cache and the cache shared with another one of cores 909. Processor 901 may further include a direct cache access (DCA) logic to enable other devices such as devices 903-904 to directly access cache 908.

Processor 901 and/or chipset 902 may further include an interrupt controller, such as an advanced programmable interrupt controller (APIC), to handle interrupts such as message signaled interrupts.

Some portions of the preceding detailed descriptions have been presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the ways used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of operations leading to a desired result. The operations are those requiring physical manipulations of physical quantities.

It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the above discussion, it is appreciated that throughout the description, discussions utilizing terms such as those set forth in the claims below, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

The techniques shown in the figures can be implemented using code and data stored and executed on one or more electronic devices. Such electronic devices store and communicate (internally and/or with other electronic devices over a network) code and data using computer-readable media, such as non-transitory computer-readable storage media (e.g., magnetic disks; optical disks; random access memory; read only memory; flash memory devices; phase-change memory) and transitory computer-readable transmission media (e.g., electrical, optical, acoustical or other form of propagated signals - such as carrier waves, infrared signals, digital signals).

The processes or methods depicted in the preceding figures may be performed by processing logic that comprises hardware (e.g. circuitry, dedicated logic, etc.), firmware, software (e.g., embodied on a non-transitory computer readable medium), or a combination of both. Although the processes or methods are described above in terms of some sequential operations, it should be appreciated that some of the operations described may be performed in a different order. Moreover, some operations may be performed in parallel rather than sequentially.

In the foregoing specification, embodiments of the invention have been described with reference to specific exemplary embodiments thereof. It will be evident that various modifications may be made thereto without departing from the broader spirit and scope of the invention as set forth in the following claims. The specification and drawings are, accordingly,

5 to be regarded in an illustrative sense rather than a restrictive sense.

CLAIMS

What is claimed is:

1. A processor, comprising:
an instruction decoder to receive a first instruction having a prefix and an opcode and to
generating, by the instruction decoder of the processor, a second instruction
5 executable based on a condition determined based on the prefix; and
an execution unit to conditionally execute the second instruction based on the condition
determined based on the prefix.
2. The processor of claim 1, further comprising a prefix-condition table to convert a
plurality of prefixes to a plurality of conditions, wherein the instruction decoder is to perform a
lookup operation in the prefix-condition table to translate the prefix to the condition.
3. The processor of claim 1, wherein the execution unit is to examine a flag register of the
processor based on the condition to determine whether one or more flags of the flag register
satisfy the condition derived from the prefix to execute the second instruction.
4. The processor of claim 1, further comprising flag modification override logic to prevent
the execution unit from modifying a flag register of the processor based on the prefix of the first
instruction.
5. The processor of claim 1, wherein the instruction decoder is to:
10 extract the prefix from the first instruction,
determine whether the first instruction is valid based on the prefix in view of a capability
of the processor, wherein the execution unit is to execute the first instruction only
if the first instruction is valid, and
generate an exception indicating that the first instruction is invalid, if one or more bits of
15 the prefix matches a predetermined bit pattern based on the capability of the
processor.

6. The processor of claim 5, wherein in determining whether the first instruction is valid, the instruction decoder is to examine a value of one or more bits of the prefix in view of a processor identifier that identifies a type of the processor.

7. The processor of claim 1, wherein the flag modification logic is to:
prevent the execution unit from modifying the flag register if one or more bits of the
prefix match a first predetermined bit pattern, and
allowing the execution unit to modify the flag register if one or more bits of the prefix
5 match a second predetermined bit pattern.

8. The processor of claim 1, wherein the prefix includes one or more bits representing a vector length when the opcode includes a vector instruction that is supported by the processor.

9. A method, comprising:
receiving, at a processor, first instruction having a prefix and an opcode;
generating, by an instruction decoder of the processor, a second instruction executable
based on a condition determined based on the prefix; and
conditionally executing, by an execution unit of the processor, the second instruction
10 based on the condition determined based on the prefix.

10. The method of claim 9, wherein generating a second instruction comprises performing a lookup operation in a prefix-condition table to translate the prefix to the condition.

11. The method of claim 9, wherein conditionally executing comprises examining a flag register of the processor based on the condition to determine whether one or more flags of the flag register satisfy the condition derived from the prefix to execute the second instruction.

12. The method of claim 9, further comprising preventing the execution unit from modifying a flag register of the processor based on the prefix of the first instruction.

13. The method of claim 9, further comprising:
extracting the prefix from the first instruction;

determining whether the first instruction is valid based on the prefix in view of a capability of the processor, wherein the execution unit is to execute the first instruction only if the first instruction is valid; and
generating an exception indicating that the first instruction is invalid, if one or more bits of the prefix matches a predetermined bit pattern based on the capability of the processor.

14. The method of claim 13, wherein determining whether the first instruction is valid comprises examining a value of one or more bits of the prefix in view of a processor identifier that identifies a type of the processor.

15. The method of claim 9, further comprising:
preventing the execution unit from modifying the flag register if one or more bits of the prefix match a first predetermined bit pattern; and
allowing the execution unit to modify the flag register if one or more bits of the prefix match a second predetermined bit pattern.

16. The method of claim 9, wherein the prefix includes one or more bits representing a vector length when the opcode includes a vector instruction that is supported by the processor.

17. A system, comprising:
an interconnect;
a processor coupled the interconnect to receive a first instruction having a prefix and an opcode, to generating, by an instruction decoder of the processor, a second instruction executable based on a condition determined based on the prefix, and to cause an execution unit to conditionally execute the second instruction based on the condition determined based on the prefix; and
a dynamic random access memory (DRAM) coupled to the interconnect.

18. The system of claim 17, wherein the processor further comprises a prefix-condition table to convert a plurality of prefixes to a plurality of conditions, wherein the instruction decoder is to perform a lookup operation in the prefix-condition table to translate the prefix to the condition.

19. The system of claim 17, wherein the processor is to examine a flag register of the processor based on the condition to determine whether one or more flags of the flag register satisfy the condition derived from the prefix to execute the second instruction.

20. The system of claim 17, wherein the processor further comprises flag modification override logic to prevent the execution unit from modifying a flag register of the processor based on the prefix of the first instruction.

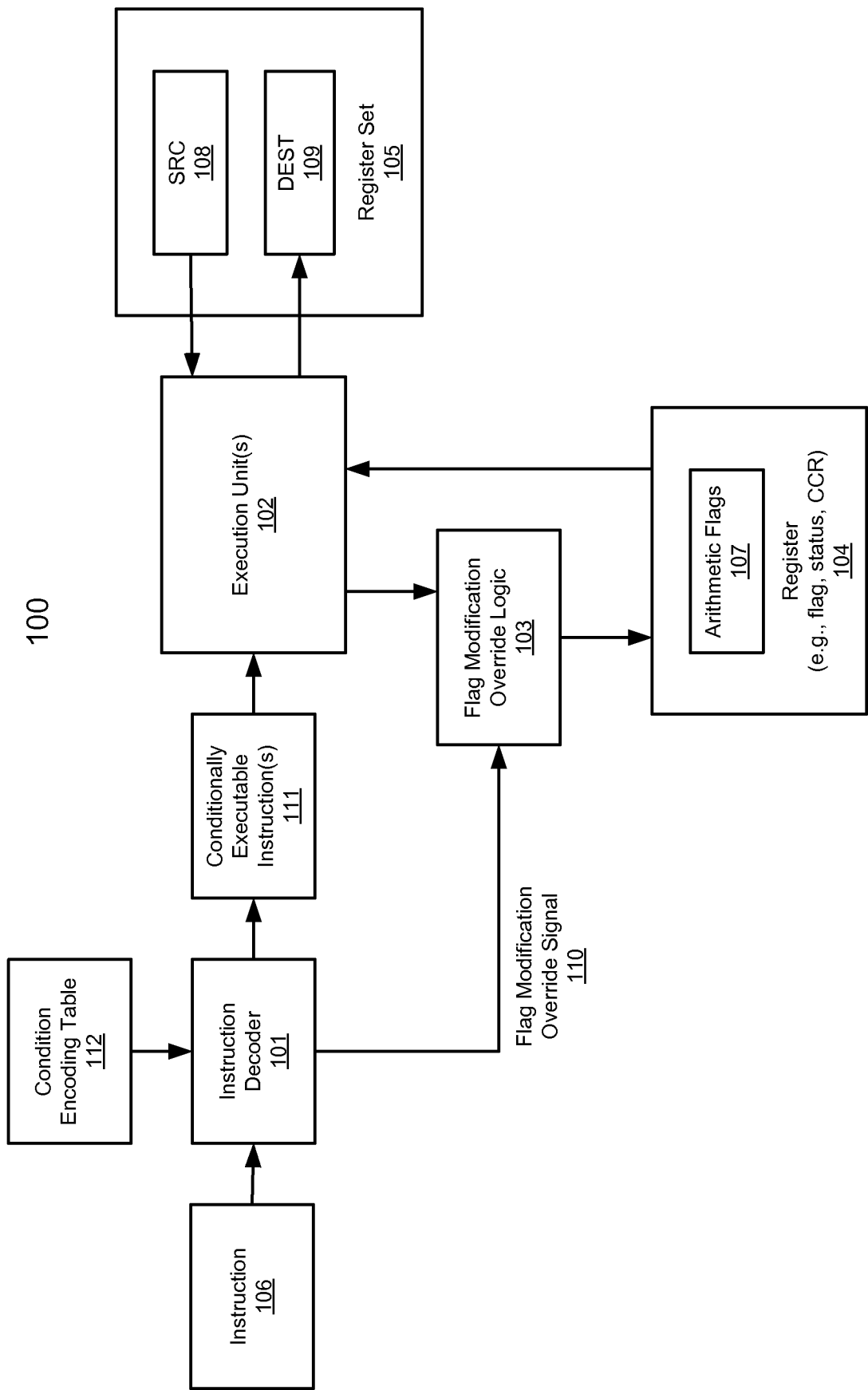


FIG. 1

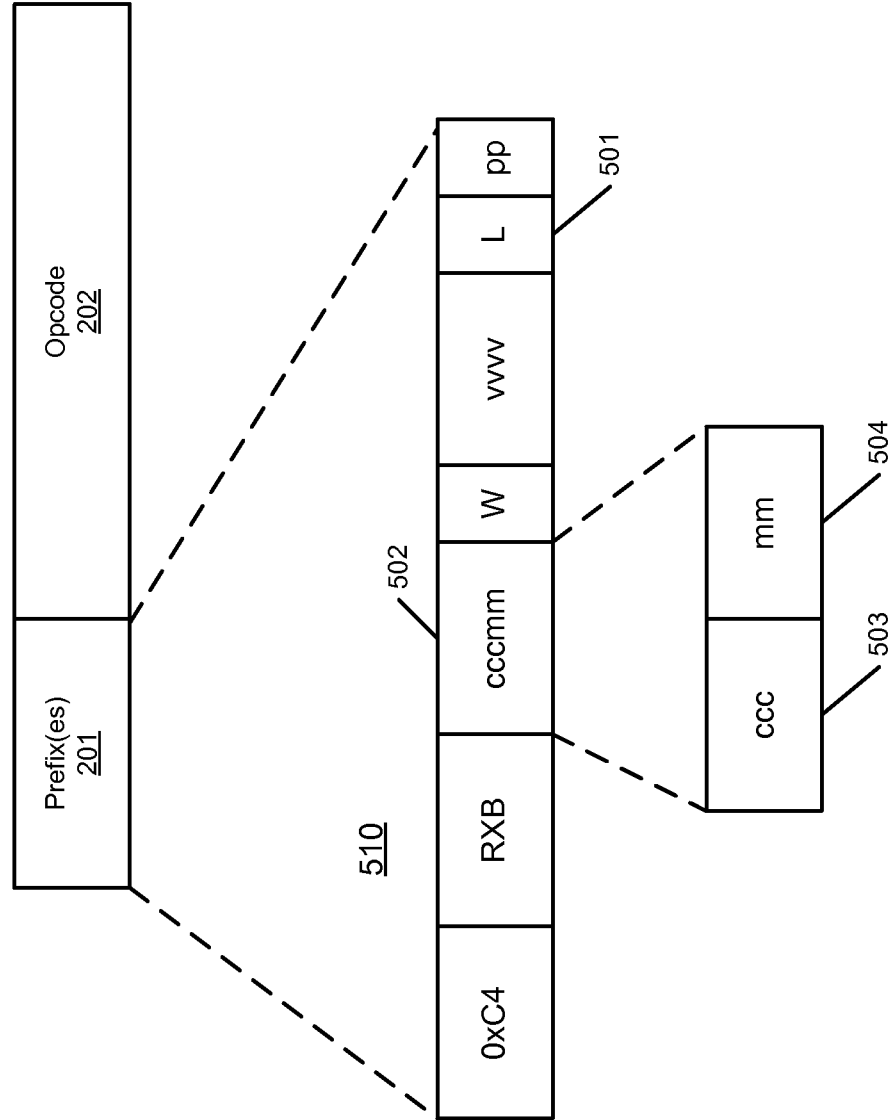


FIG. 2

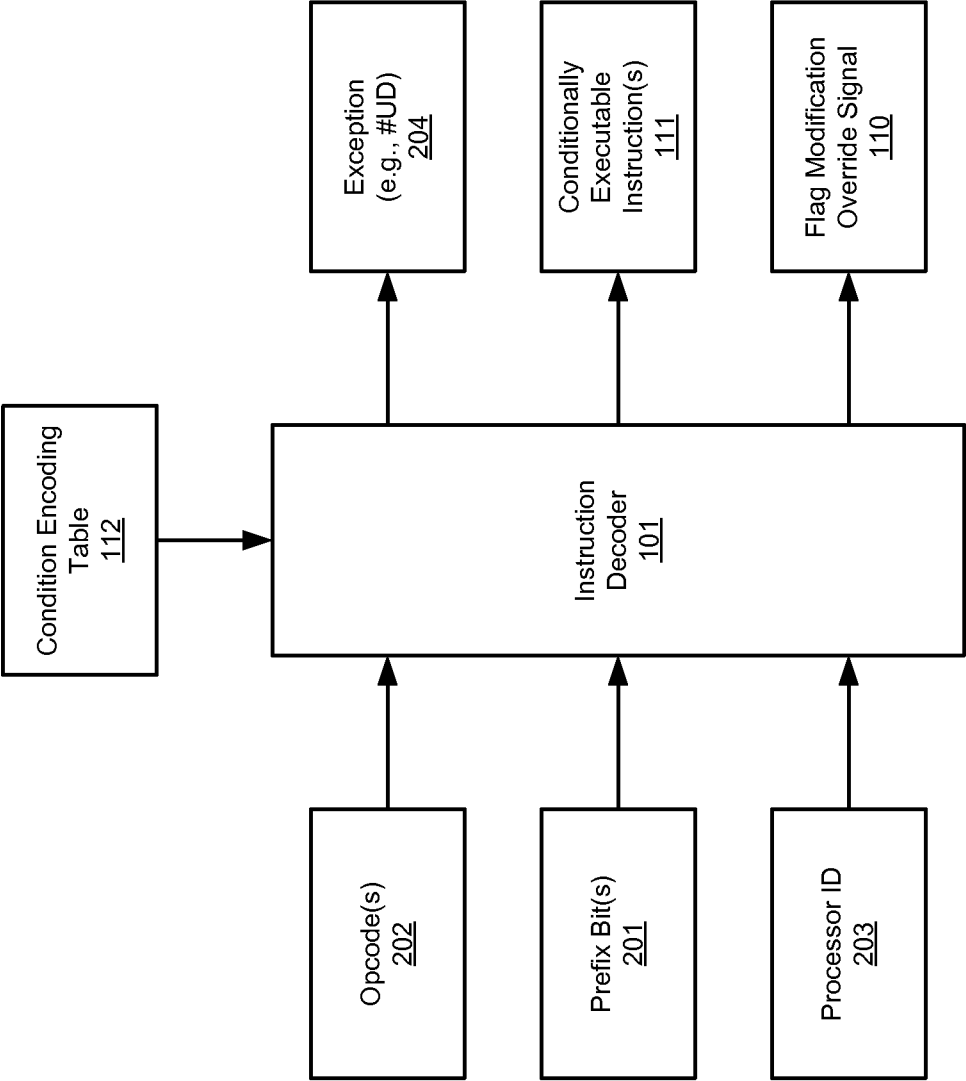
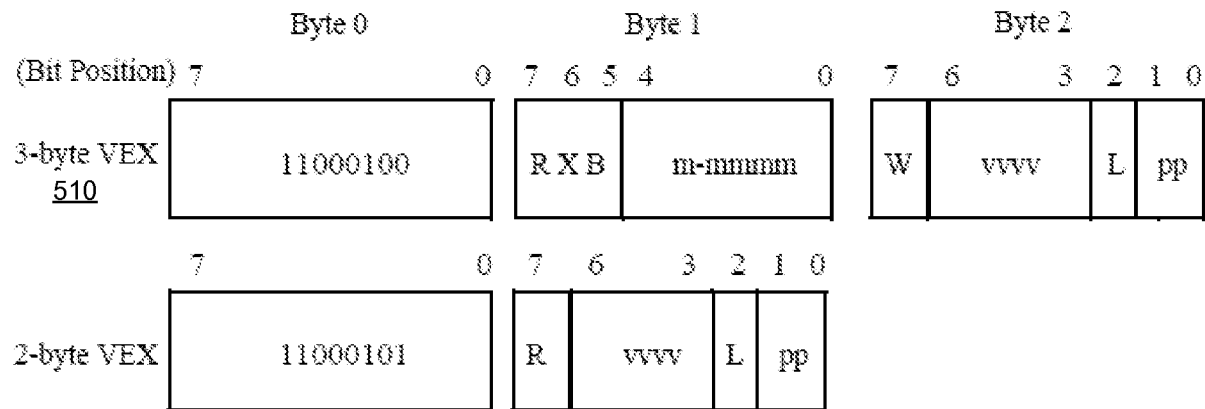


FIG. 3

5/10



R: REX.R in 1's complement (inverted) form

1: Same as REX.R=0 (must be 1 in 32-bit mode)

0: Same as REX.R=1 (64-bit mode only)

X: REX.X in 1's complement (inverted) form

1: Same as REX.X=0 (must be 1 in 32-bit mode)

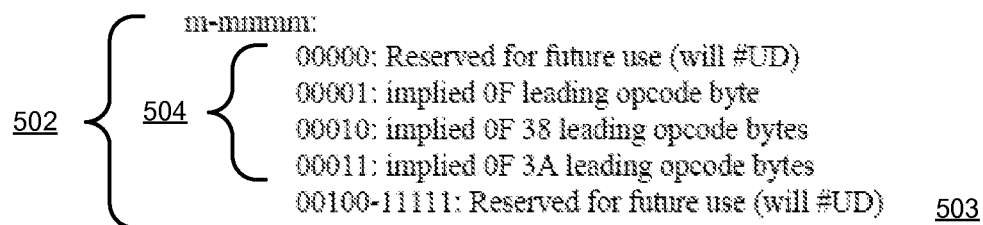
0: Same as REX.X=1 (64-bit mode only)

B: REX.B in 1's complement (inverted) form

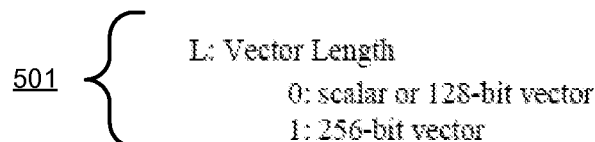
1: Same as REX.B=0 (Ignored in 32-bit mode).

0: Same as REX.B=1 (64-bit mode only)

W: opcode specific (use like REX.W, or used for opcode extension, or ignored, depending on the opcode byte)



vvvv: a register specifier (in 1's complement form) or 1111 if unused.



pp: opcode extension providing equivalent functionality of a SIMD prefix

00: None

01: 66

10: F3

11: F2

FIG. 5

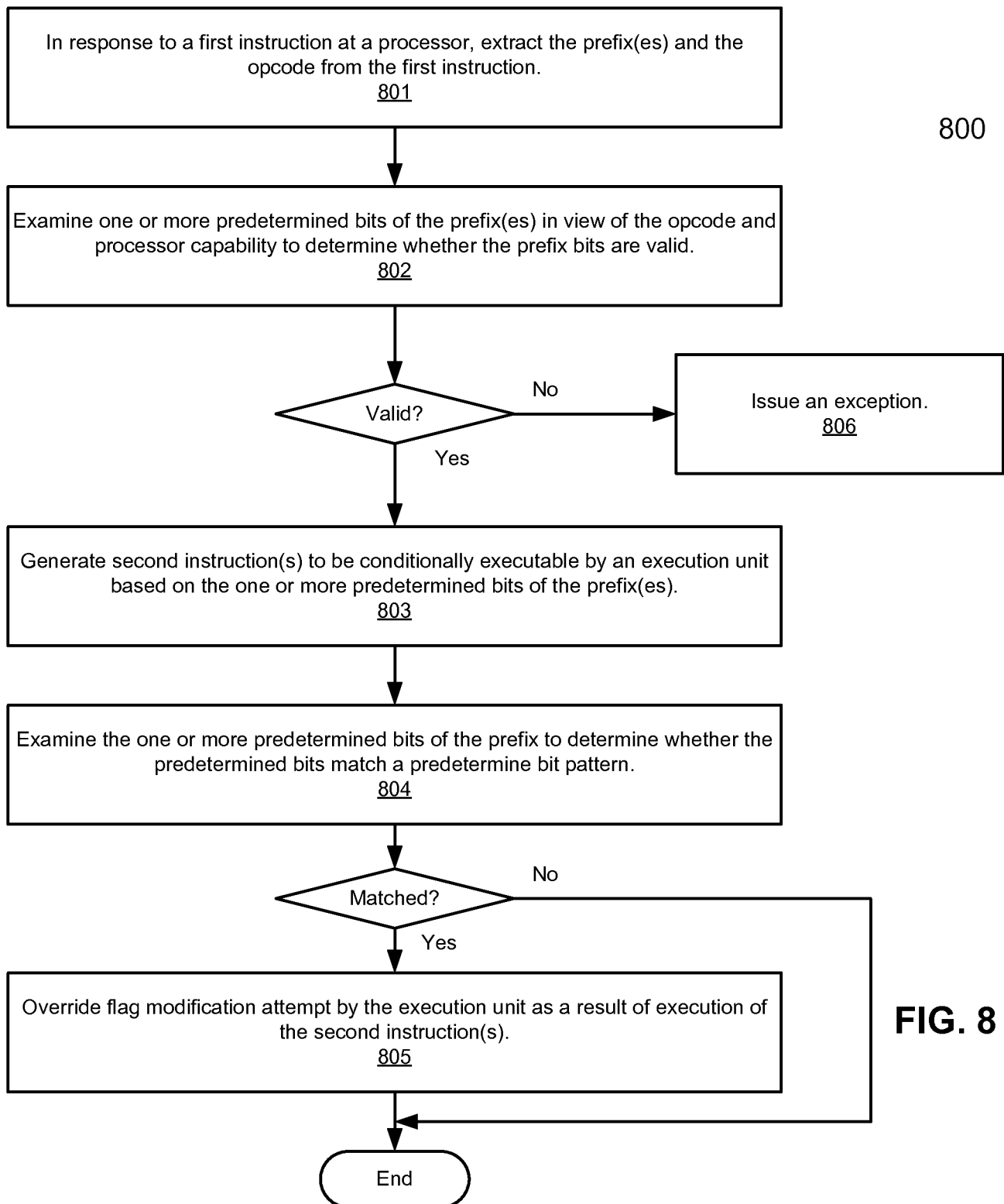
tttn	Mnemonic	Condition
0000	O	Overflow
0001	NO	No overflow
0010	B, NAE	Below, Not above or equal
0011	NB, AE	Not below, Above or equal
0100	E, Z	Equal, Zero
0101	NE, NZ	Not equal, Not zero
0110	BE, NA	Below or equal, Not above
0111	NBE, A	Not below or equal, Above
1000	S	Sign
1001	NS	Not sign
1010	P, PE	Parity, Parity Even
1011	NP, PO	Not Parity, Parity Odd
1100	L, NGE	Less than, Not greater than or equal to
1101	NL, GE	Not less than, Greater than or equal to
1110	LE, NG	Less than or equal to, Not greater than
1111	NLE, G	Not less than or equal to, Greater than

FIG. 6
(Prior Art)

c'cc'L	Mnemonic	Conditional Execution	Flag Non-Modifying
0000	{O}	Overflow	TRUE
0001	{NO}	No overflow	TRUE
0010	{B}, {NAE}	Below, Not above or equal	TRUE
0011	{NB}, {AE}	Not below, Above or equal	TRUE
0100	{E}, {Z}	Equal, Zero	TRUE
0101	{NE}, {NZ}	Not equal, Not zero	TRUE
0110	{BE}, {NA}	Below or equal, Not above	TRUE
0111	{NBE}, {A}	Not below or equal, Above	TRUE
1000	{S}	Sign	TRUE
1001	{NS}	Not sign	TRUE
1010	{UNC} {optional}	Unconditional	FALSE
1011	{UNC} {optional}	Unconditional	TRUE
1100	{L}, {NGE}	Less than, Not greater than or equal to	TRUE
1101	{NL}, {GE}	Not less than, Greater than or equal to	TRUE
1110	{LE}, {NG}	Less than or equal to, Not greater than	TRUE
1111	{NLE}, {G}	Not less than or equal to, Greater than	TRUE

FIG. 7

8/10



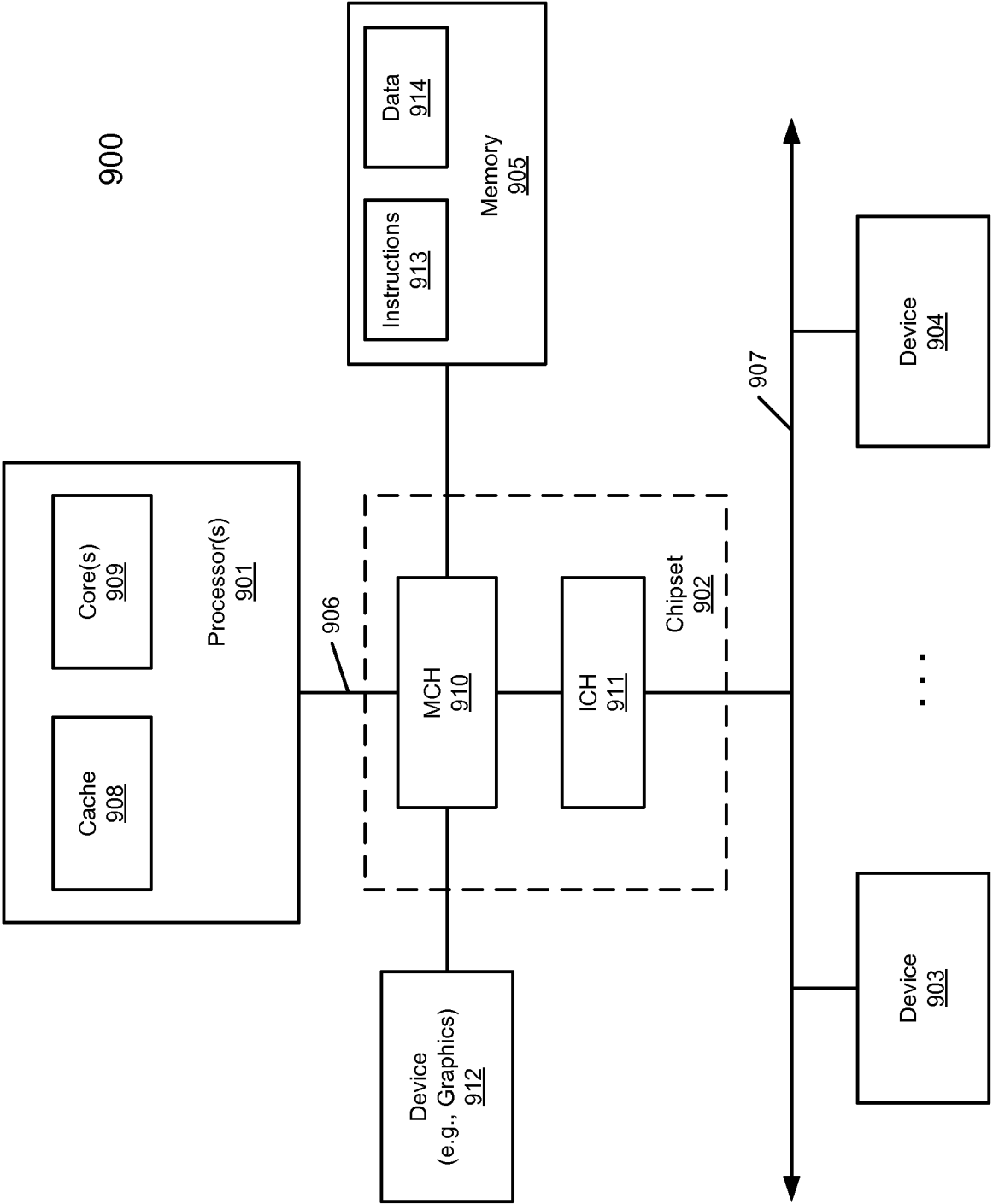


FIG. 9

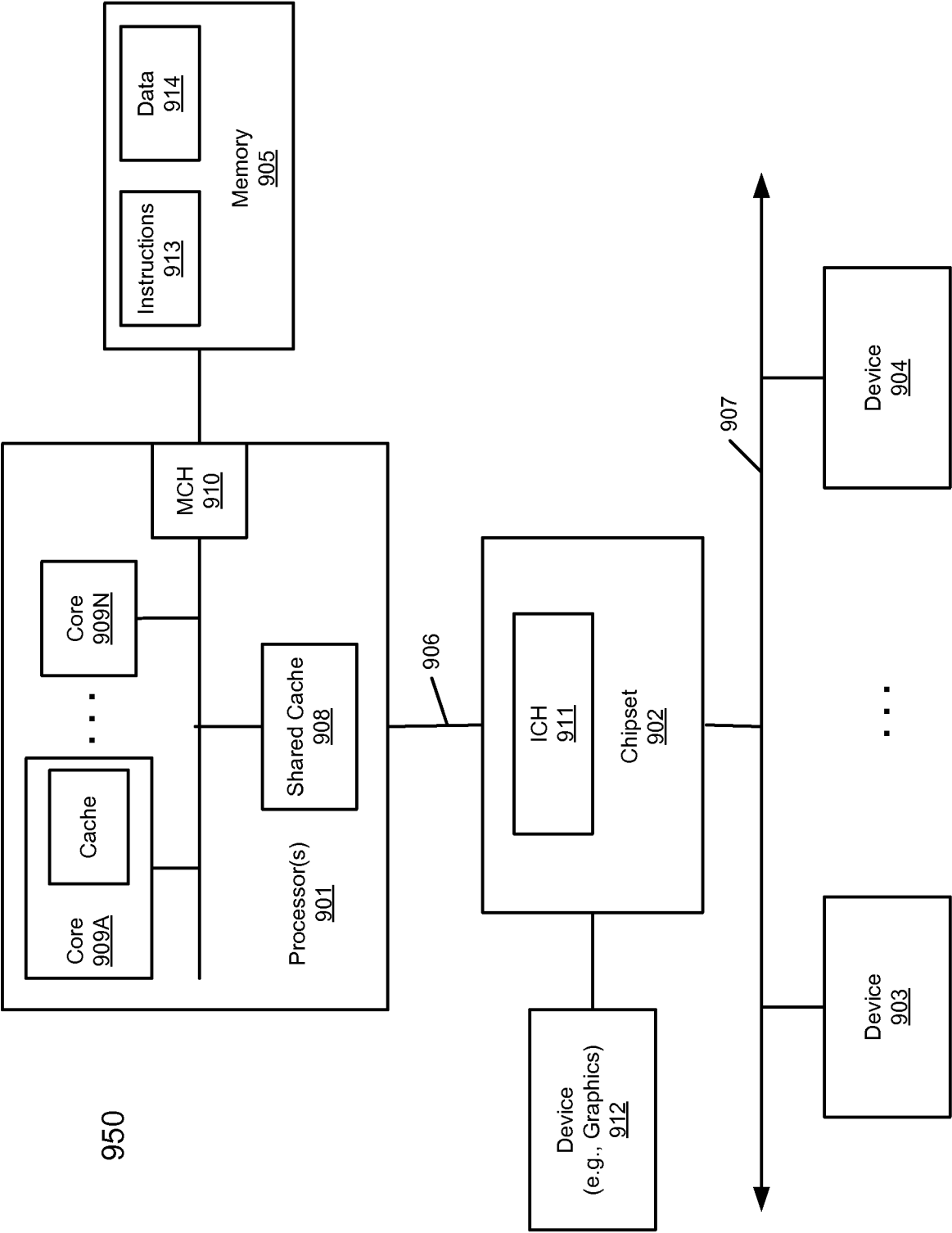


FIG. 10

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2011/062673**A. CLASSIFICATION OF SUBJECT MATTER****G06F 9/30(2006.01)i, G06F 9/06(2006.01)1**

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 9/30; G06F 9/38

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

Korean utility models and applications for utility models

Japanese utility models and applications for utility models

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

eKOMPASS(KIPO internal) & Keywords: instruction, prefix, opcode, generate, second instruction.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 7155598 B2 (HENRY, G.G. et al.) 26 December 2006 See column 5 line 6-38, column 8 line 12-60, column 10 line 10-20, line 39-45, column 11 line 61-67, column 12 line 1-11, column 13 line 13-16, claims 1, 6, 13, 15, 31, and figures 1,3,5,7,9,11.	1-3, 9, 11
X A		17, 19 4-8, 10, 12-16, 18, 20
A	US 2009-0172356 A1 (VALENTINE, R. et al.) 02 July 2009 See paragraphs [0019], [0024], [0066], claims 1-2, and figures 1-3.	1-20
A	US 2010-0332803 A1 (YOSHIDA, T. et al.) 30 December 2010 See paragraphs [0094]-[0095], claims 1,8, and figures 11, 16.	1-20
A	US 06014735A A (CHENNUPATY, S. et al.) 11 January 2000 See column 2 line 49-51, column 3 lines 1-2, column 6 lines 20-28, claim 13, and figures 1-3.	1-20

☐ 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 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

20 JULY 2012 (20.07.2012)

Date of mailing of the international search report

23 JULY 2012 (23.07.2012)

Name and mailing address of the ISA/KR

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

Facsimile No. 82-42-472-7 140

Authorized officer

Hwang, Seung Hee

Telephone No. 82-42-481-5749



INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US201 1/062673

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 7155598 B2	26.12.2006	CN 1212566 C0 CN 1414466 A0 EP 1351130 A2 EP 1351130 A3 TW 569136 A US 2003-0188140 A1	27..07..2005 30..04..2003 08..10..2003 07..09..2005 01..01..2004 02..10..2003
US 2009-0172356 A1	02.07.2009	CN 101533342 A	16..09..2009
US 2010-0332803 A1	30.12.2010	EP 2270650 A1 JP 2011-013865 A	05..01..2011 20..01..2011
US 06014735A A	11.01.2000	EP 0947919 A2 EP 0947919 A3 EP 0947919 B1 TW 445431 A TW 445431 B	06..10..1999 13..09..2000 25..05..2005 11..07..2001 11..07..2001