



- (51) **International Patent Classification:**
G06F 21/50 (2013.01) *G06F 11/30* (2006.01)
- (21) **International Application Number:**
PCT/US2013/048531
- (22) **International Filing Date:**
28 June 2013 (28.06.2013)
- (25) **Filing Language:** English
- (26) **Publication Language:** English
- (30) **Priority Data:**
13/664,532 31 October 2012 (31.10.2012) US
- (71) **Applicant:** INTEL CORPORATION [US/US]; 2200 Mission College Boulevard, Santa Clara, California 95054 (US).
- (72) **Inventors:** FISCHER, Stephen A.; 11357 Tunnel Hill Way, Gold River, California 95670 (US). GOTZE, Kevin C.; 4225 SE Oakhurst St, Hillsboro, Oregon 97123 (US). BULYGIN, Yuriy; 16475 SW Black Bird Dr., Beaverton, Oregon 97007 (US). BRANNOCK, Kirk D.; 18623 Forest Park Rd., Hillsboro, Oregon 97123 (US).
- (74) **Agent:** GARZA, John C.; Trop, Pruner & HU, P.C., 1616 S. Voss Rd., Ste. 750, Houston, Texas 77057-2631 (US).
- (81) **Designated States** (*unless otherwise indicated, for every kind of national protection available*): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY,

BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

- (84) **Designated States** (*unless otherwise indicated, for every kind of regional protection available*): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

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

Published:

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

- (54) **Title:** DETECTION OF RETURN ORIENTED PROGRAMMING ATTACKS

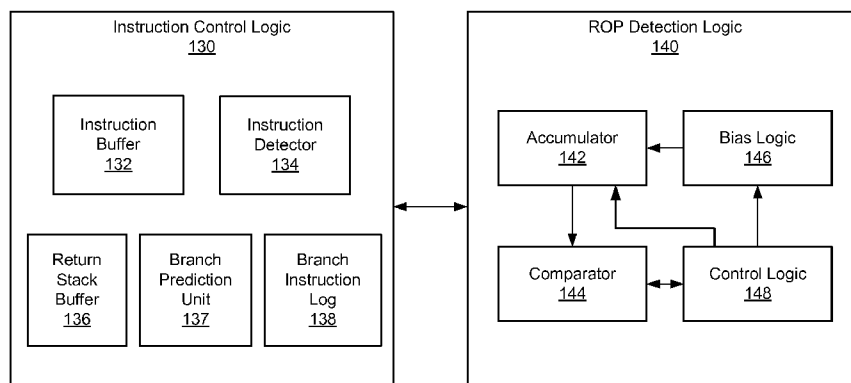


FIG. 1B

- (57) **Abstract:** In one embodiment, a processor includes at least one execution unit and Return Oriented Programming (ROP) detection logic. The ROP detection logic may determine a ROP metric based on a plurality of control transfer events. The ROP detection logic may also determine whether the ROP metric exceeds a threshold. The ROP detection logic may also, in response to a determination that the ROP metric exceeds the threshold, provide a ROP attack notification.

DETECTION OF RETURN ORIENTED PROGRAMMING ATTACKS

Background

[0001] Embodiments relate generally to computer security.

[0002] Computer exploits are techniques which may be used to compromise the security of a computer system or data. Such exploits may take advantage of a vulnerability of a computer system in order to cause unintended or unanticipated behavior to occur on the computer system. For example, Return Oriented Programming (ROP) exploits may involve identifying a series of snippets of code that are already available in executable memory (e.g., portions of existing library code), and which are followed by a return instruction (e.g., a RET instruction). Such snippets may be chained together into a desired execution sequence by pushing a series of pointer values onto the call stack and then tricking the code into execution the first pointer value. This chained execution sequence does not follow the intended program execution order that the original program author intended, but may instead follow an alternative execution sequence. In this manner, an attacker may create a virtual program sequence without requiring injection of external code.

Brief Description Of The Drawings

[0003] FIGs. 1A-1B are block diagrams of systems in accordance with one or more embodiments.

[0004] FIG. 2 is a sequence in accordance with one or more embodiments.

[0005] FIGs. 3A-3E are sequences in accordance with one or more embodiments.

[0006] FIG. 4 is a block diagram of a processor in accordance with an embodiment of the present invention.

[0007] FIG. 5 is a block diagram of a multi-domain processor in accordance with another embodiment of the present invention.

[0008] FIG. 6 is a block diagram of an embodiment of a processor including multiple cores is illustrated.

[0009] FIG. 7 is a block diagram of a system in accordance with an embodiment of the present invention.

Detailed Description

[0010] In accordance with some embodiments, detection of Return Oriented Programming (ROP) attacks may be provided. In one or more embodiments, ROP attacks may be detected based on a ROP metric (e.g., a metric indicating the likelihood that a system is under a current ROP attack). In some embodiments, the ROP metric is generated based at least in part on control transfer events, meaning instances of instructions and/or states that are associated with ROP attacks. For example, in some embodiments, control transfer events may include instances of control transfer instructions such as subroutine call instructions, subroutine return instructions, branch or jump instructions, etc. Further, in some embodiments, control transfer events may include pairs of associated instructions (i.e., specific types of instructions executed within a given range of each other). Furthermore, in some embodiments, control transfer events may include branch or return mispredictions.

[0011] Referring to FIG. 1A, shown is a block diagram of a system 100 in accordance with one or more embodiments. As shown in FIG. 1A, the system 100 may include a processor 110 and a memory 120. In accordance with some embodiments, the system 100 may be all or a portion of any electronic device, such as a cellular telephone, a computer, a server, a media player, a network device, etc.

[0012] In accordance with some embodiments, the memory 120 may include an operating system (OS) 122 and protection software 124. In some embodiments, the operating OS 122 and/or the protection software 124 may include functionality to protect the system 100 against computer exploits and attacks. For example, the protection software 124 may be an anti-virus application, an intrusion detector, a network firewall, etc.

[0013] As shown, in some embodiments, the processor 110 may include instruction control logic 130 and ROP detection logic 140. In one or more embodiments, the instruction control logic 130 may include functionality to manage and/or optimize the performance characteristics of the processor 110. For example, the instruction

control logic 130 may include functionality for performance profiling, logging, pre-fetching, branch prediction, tuning, etc. Further, in some embodiments, the ROP detection logic 140 may include functionality to detect a ROP attack against the system 100. This functionality of the ROP detection logic 140 is described further below with reference to FIGs. 1B, 2, and 3A-3E.

[0014] In one or more embodiments, the ROP detection logic 140 may be implemented in any form of hardware, software, and/or firmware. For example, the ROP detection logic 140 may be implemented in microcode, programmable logic, hard-coded logic, control logic, instruction set architecture, processor abstraction layer, etc. Further, the ROP detection logic 140 may be implemented within the processor 110, and/or any other component accessible or medium readable by processor 110, such as memory 120. While shown as a particular implementation in the embodiment of FIG. 1A, the scope of the various embodiments discussed herein is not limited in this regard.

[0015] Referring now to FIG. 1B, shown are block diagrams of the instruction control logic 130 and the ROP detection logic 140 in accordance with one or more embodiments. As shown, in some embodiments, the instruction control logic 130 may include an instruction buffer 132, an instruction detector 134, a return stack buffer 136, a branch prediction unit 137, and a branch instruction log 138.

[0016] In one or more embodiments, the instruction buffer 132 may be a buffer including entries corresponding to instructions processed in the processor 110. For example, in some embodiments, the instruction buffer 132 may be an instruction retirement buffer. In such embodiments, as each instruction is completed, a corresponding entry is cleared from the instruction buffer 132. Such instructions may include, e.g., subroutine call instructions (e.g., CALL), subroutine return instructions (e.g., RET), branch or jump instructions (e.g., IF-THEN, JMP, GOTO), stack instructions (e.g., PUSH, POP), etc.

[0017] In one or more embodiments, the instruction detector 134 includes functionality to detect entries of the instruction buffer 132 that correspond to control transfer instructions associated with ROP attacks. For example, in some

embodiments, the instruction detector 134 may detect entries corresponding to subroutine call instructions, subroutine return instructions, branch or jump instructions, stack instructions, etc. Further, in response to detecting such entries, the instruction detector 134 may send a detection signal to the ROP detection logic 140. In some embodiments, the detection signal may indicate the type of the detected instruction.

[0018] In one or more embodiments, the return stack buffer 136 stores return pointers for use in performance optimization of the processor 110 (i.e., return prediction). For example, when a call to a subroutine is executed in a program, the instruction control logic 130 may predict that a corresponding return from the subroutine will be subsequently performed. Accordingly, in some embodiments, a return pointer may be pushed onto the return stack buffer 136 in anticipation of executing the predicted return instruction.

[0019] In accordance with some embodiments, the branch prediction unit 137 includes functionality to predict the future direction of a branch executed by the processor 110. In some embodiments, the predictions provided by the branch prediction unit 137 may be used to fetch instructions so that they can be readied for execution and/or speculatively executed, thereby saving some of the time that would be required to fetch the instructions when the branch is taken. In one or more embodiments, the branch prediction unit 137 may include functionality to identify correct and incorrect branch predictions. For example, the branch prediction unit 137 may identify all incorrectly predicted branches, and/or may identify specific types of branch mispredictions, such as mispredicted indirect branches (i.e., branches where the branch target is held in a register or memory), mispredicted far branch instructions (i.e., branches that perform a control transfer that also involves changing the code segment), etc.

[0020] In accordance with some embodiments, the branch instruction log 138 may store address information for a given number of the most recent branch related instructions. For example, in some embodiments, the branch instruction log 138 may store source and destination addresses for the last sixteen branch related instructions (including call and return instructions) processed in the processor 110.

Such functionality of the branch instruction log 138 may be used for debugging purposes, and may be referred to as Last Branch Record (LBR) functionality.

[0021] In some embodiments, the branch instruction log 138 may include functionality to selectively freeze its contents, meaning to stop updating the stored address information at a specific point in time, regardless of whether any new branch related instructions are executed subsequently. In one or more embodiments, the freeze function may be triggered based on a signal received from the ROP detection logic 140.

[0022] As shown, in some embodiments, the ROP detection logic 140 includes an accumulator 142, a comparator 144, bias logic 146, and control logic 148. In one or more embodiments, the accumulator 142 includes functionality to generate a count based on control transfer events occurring during a defined window. Specifically, in some embodiments, the accumulator 142 may increment or decrement a counter in response to detecting instances of control transfer instructions (e.g., call or return instructions, branch or jump instructions, etc.) and/or mispredictions of control transfer instructions. For example, in some embodiments, the accumulator 142 may increment the counter in response to detecting a control transfer instruction associated with popping (i.e., removing) an instruction return pointer value from the call stack (e.g., a return instruction). Further, in some embodiments, the accumulator 142 may decrement the counter in response to detecting a control transfer instruction associated with pushing (i.e., storing) an instruction return pointer value on the call stack (e.g., a call instruction).

[0023] In one or more embodiments, the accumulator 142 may detect instances of control transfer instructions using the instruction detector 134. For example, the accumulator 142 may increment the counter in response to receiving a detection signal indicating the detection of a return instruction. Further, in some embodiments, the accumulator 142 may decrement the counter in response to receiving a detection signal indicating the detection of a call instruction. Note that, in normal operation (i.e., when not under a ROP attack), a call instruction is typically followed some instructions later by a corresponding return instruction. Accordingly, in normal operation, counter increases are generally balanced by counter decreases, and thus

the counter will remain within a specific range around the zero value. However, in the event of a ROP attack, the number of return instructions may substantially exceed the number of call instructions (referred to as a return-call imbalance). Therefore, under a ROP attack, the counter may increase beyond the specific range around the zero value. Thus, in this example, the counter value may be used as a ROP metric.

[0024] In one or more embodiments, the accumulator 142 may interact with the return stack buffer 136 to detect return predictions and/or mispredictions, and to increment/decrement the count based on such detections. For example, in some embodiments, the accumulator 142 may increment a counter in response to detecting a mispredicted return instruction. Further, in some embodiments, the accumulator 142 may decrement the counter in response to detecting a correctly predicted return instruction.

[0025] In one or more embodiments, the accumulator 142 may interact with the branch prediction unit 137 to detect branch predictions and mispredictions, and to increment/decrement the count based on such detections. For example, in some embodiments, the accumulator 142 may increment a counter in response to detecting any mispredicted branch instruction. Further, in some embodiments, the accumulator 142 may decrement the counter in response to detecting any correctly predicted branch instruction. In another example, in some embodiments, the accumulator 142 may increment a counter in response to detecting a particular type of mispredicted branch (e.g., a mispredicted indirect branch, a mispredicted far branch, etc.).

[0026] In one or more embodiments, the accumulator 142 may also detect instances of stack pivots (e.g., a return instruction associated with an instruction moving the stack pointer to a new memory location). Further, in response to detecting a stack pivot, the accumulator 142 may increment the counter by some amount (e.g., 1, 2, 5, etc.).

[0027] In one or more embodiments, the accumulator 142 may adjust a single counter based on multiple types of control transfer events (e.g., call instructions,

return instructions, mispredictions, etc.). Further, in some embodiments, the accumulator 142 may include separate counters, each corresponding to a different type of control transfer event. Alternatively, in some embodiments, the ROP detection logic 140 includes multiple accumulators 142, each corresponding to a different type of control transfer event.

[0028] In one or more embodiments, the accumulator 142 may be limited to a predefined window. For example, the accumulator 142 may reset the count after a specific number of instructions (e.g., 10, 100, 1000, etc.) are processed in the processor 110. In another example, the accumulator 142 may be a circular buffer storing a given number of instructions. In yet another example, the accumulator 142 may reset the count after a given time period (e.g., 1 millisecond, 1 second, 1 minute, etc.) has expired. In such embodiments, the counter may reflect a return-call imbalance occurring within the predefined window (e.g., ten more return instructions than call instructions processed during a window of 1000 instructions). In some embodiments, the accumulator 142 may include a saturating mode to prevent the count from exceeding maximum and/or minimum limits. For example, in some embodiments, the accumulator 142 may clip the count to a maximum count limit (e.g., a hardware buffer capacity) in the case of a count increment, and/or may clip the count to the minimum count limit in the case of a count decrement.

[0029] In accordance with some embodiments, the comparator 144 includes functionality to provide an attack notification when the count of the accumulator 142 exceeds a predefined threshold. In some embodiments, the predefined threshold may be set to a count level or percentage that indicates a high probability that the system 100 is under a ROP attack. For example, assume that the count of the accumulator 142 is to indicate a return-call imbalance during a window of one hundred instructions. Assume further that, before completing the window of one hundred instructions, the comparator 144 determines that the count of the accumulator 142 reaches eleven, and thereby exceeds a predefined threshold of ten (i.e., a count of positive ten). Therefore, in this example, the comparator 144 may trigger an attack notification (e.g., an interrupt, an exception, etc.) to indicate that the system 100 is probably under a ROP attack. Further, in some embodiments, the

attack notification may be sent to the OS 122 and/or the protection software 124. In response, in one or more embodiments, the OS 122 and/or the protection software 124 may undertake actions to prevent and/or interrupt the ROP attack (e.g., system or process stoppage, memory quarantine, event logging, user notification, etc.). For example, the OS 122 and/or protection software 124 may determine that the ROP detection is false based on the system process state, and may thus allow the process to continue execution. In some embodiments, the OS 122 and/or protection software 124 may examine the state of the branch instruction log 138 as part of such a determination.

[0030] Note that, while the functionality of the accumulator 142 and/or the comparator 144 is described above in terms of the example of a return-call imbalance, embodiments are not limited in this regard. In particular, in some embodiments, the accumulator 142 and/or the comparator 144 may use any other type of control transfer event, or any combination of types of control transfer events. For example, in some embodiments, the accumulator 142 may increment the count in response to instances of control transfer instructions, without decrementing the count. In another example, in some embodiments, the accumulator 142 may increment and/or decrement the count in response to detecting control transfer instructions that are associated with stack-related instructions (e.g., a jump instruction using a pointer associated with a pop instruction, a jump instruction using a pointer associated with a push instruction, etc.). In yet another example, the accumulator 142 may increment and/or decrement the count in response to detecting control transfer instructions that are associated with instructions to change the stack pointer value (e.g., a return instruction associated with a move instruction). In still another example, in some embodiments, the accumulator 142 may increment the count in response to return mispredictions and/or branch mispredictions. This functionality of the accumulator 142 and/or the comparator 144 is described further below with reference to FIGs. 3A-3D.

[0031] In one or more embodiments, the ROP detection logic 140 may include multiple sets of components (e.g., accumulator 142, comparator 144, etc.), with each set corresponding to a different type of control transfer event. For example, in some

embodiments, the ROP detection logic 140 may include a first accumulator 142 to generate a first count based on detections of return instructions, and may include a second accumulator 142 to generate a second count based on branch mispredictions. In some embodiments, the counts generated by such sets may be combined to generate a single ROP metric. Alternatively, in some embodiments, the count generated by each set may correspond to a different ROP metric which may be evaluated to detect ROP attacks. In such embodiments, the control logic 148 may evaluate each ROP metric separately using a different threshold and/or window, and may trigger an attack notification if any single ROP metric exceeds its corresponding threshold. Optionally, in some embodiments, the control logic 148 may only trigger an attack notification if at least a predefined number of ROP metrics exceed their associated thresholds. Further, in such embodiments, each ROP metric may be weighted by a respective weight or importance.

[0032] In one or more embodiments, the bias logic 146 includes functionality to bias or adjust the accumulator 142 to reduce any effects due to natural imbalances (i.e., an imbalance or bias that is not caused by ROP attacks). Specifically, in some embodiments, the bias logic 146 may periodically divide or reduce the count of the accumulator 142 over a given period. For example, in one or more embodiments, the bias logic 146 may shift the accumulator 142 to the right by one bit in order to divide the count by two. In such a manner, the bias logic 146 may offset or reduce any natural imbalances that may be inherent in the system 100 (e.g., imbalances due to program exits, error states, etc.).

[0033] In one or more embodiments, the control logic 148 includes functionality to manage the ROP detection logic 140. In some embodiments, such functionality may include adjusting the sensitivity of the ROP detection logic 140 based on an estimated threat level and/or desired level of protection against ROP attacks (e.g., low, medium, high, etc.). For example, the control logic 148 may increase the threshold used by the comparator 144 in response to a lowered threat or protection level, thereby requiring a greater imbalance to occur in the accumulator 142 before triggering an attack notification (i.e., decreasing sensitivity to an ROP attack). Similarly, the control logic 148 may lower the threshold used by the comparator 144

in response to a heightened threat or protection level, thereby requiring a smaller imbalance to occur in the accumulator 142 before triggering an attack notification (i.e., increasing sensitivity to an ROP attack). In another example, the control logic 148 may increase or decrease the length of the window used by the accumulator 142 to adjust sensitivity to an ROP attack. In yet another example, the control logic 148 may adjust the biasing effect of the bias logic 146 to compensate for system changes which may affect any natural imbalances.

[0034] Referring now to FIG. 2, shown is a sequence 200 for detecting a ROP attack, in accordance with one or more embodiments. In one or more embodiments, the sequence 200 may be part of the ROP detection logic 140 shown in FIG. 1A. The sequence 200 may be implemented in hardware, software, and/or firmware. In firmware and software embodiments it may be implemented by computer executed instructions stored in a non-transitory computer readable medium, such as an optical, semiconductor, or magnetic storage device.

[0035] At step 210, control transfer events may be detected. For example, referring to FIG. 1B, the instruction detector 134 may detect instances of control transfer instructions (e.g., call instructions, return instructions, branch or jump instructions, etc). In another example, the branch prediction unit 137 may detect mispredictions of control transfer instructions.

[0036] At step 220, an ROP metric may be determined based on the control transfer events (detected at step 210). In one or more embodiments, the ROP metric may be a count value based on instances of control transfer instructions and/or mispredictions. Further, in some embodiments, the ROP metric may be limited to a predefined window. For example, referring to FIG. 1B, the accumulator 142 may increment and/or decrement a counter in response to a detection of a particular type of instruction (e.g., a return instruction, a call instruction, a branch instruction, a far branch instruction, etc.). In another example, the accumulator 142 may increment and/or decrement the counter in response to associated pairs of instructions (e.g., a push or pop instruction associated with a jump instruction, a return instruction associated with an instruction moving the stack pointer, etc.). In some embodiments, the accumulator 142 may detect the aforementioned

instructions based on a signal from the instruction detector 134. In yet another example, the accumulator 142 may increment and/or decrement the counter in response to a branch misprediction (e.g., determined by interacting with the branch prediction unit 137). In still another example, the accumulator 142 may increment and/or decrement the counter in response to a return misprediction (e.g., determined by interacting with the return stack buffer 136). The steps involved in performing step 220 are discussed in greater detail below with reference to FIGs. 3A-3E.

[0037] At step 230, a determination about whether the metric exceeds a predefined threshold is made. In one or more embodiments, the predefined threshold may correspond to a level of the metric that indicates that a system is probably under ROP attack. For example, referring to FIG. 1B, the comparator 144 may determine whether the count of the accumulator 142 exceeds a predefined threshold.

[0038] If it is determined at step 230 that the metric does not exceed the predefined threshold, then the sequence 200 ends. However, if it is determined at step 230 that the metric exceeds the predefined threshold, then at step 240, protection software (e.g., an anti-malware application) may be notified that a ROP attack has been detected. For example, referring to FIG. 1A, the ROP protection logic 140 may send an ROP attack notification (e.g., an interrupt, an exception, etc.) to the protection software 124 and/or the operating system 122 to indicate that a possible ROP attack has been detected. In some embodiments, the ROP attack notification may trigger the protection software to take one or more actions to address the ROP attack (e.g., monitor suspected code, quarantine suspected code, notify an administrator or a management system, halt execution, shut down a system, etc.).

[0039] At step 250, in some embodiments, a branch instruction log may be frozen in response to determining (at step 230) that the metric exceeds the predefined threshold. For example, referring to FIG. 1B, the contents of the branch instruction log 138 may be frozen (i.e., no longer updated) in response to a signal from the control logic 148. In some embodiments, the contents of the branch instruction log 138 may then be provided to the protection software 124 (shown in FIG. 1A) for use

in analyzing and/or addressing the possible ROP attack. After step 250, the sequence 200 ends.

[0040] Referring now to FIG. 3A, shown is a sequence 310 for determining a metric, in accordance with one or more embodiments. In particular, the sequence 310 illustrates an exemplary expansion of the steps involved in performing step 220 (shown in FIG. 2). In one or more embodiments, the sequence 310 may be part of the ROP detection logic 140 shown in FIG. 1A. The sequence 310 may be implemented in hardware, software, and/or firmware. In firmware and software embodiments it may be implemented by computer executed instructions stored in a non-transitory computer readable medium, such as an optical, semiconductor, or magnetic storage device.

[0041] At step 312, a count may be incremented for each instance of a return instruction. For example, referring to FIG. 1B, the accumulator 142 may receive a signal from the instruction detector 134 indicating the detection of a return instruction. In response to receiving this signal, the accumulator 142 may increment a counter by one.

[0042] At step 314, the count may be decremented for each instance of a call instruction. For example, referring to FIG. 1B, the accumulator 142 may receive a signal from the instruction detector 134 indicating the detection of a call instruction. In response to receiving this signal, the accumulator 142 may decrement the counter by one. In one or more embodiments, the counter value may correspond to a return-call imbalance metric.

[0043] At step 316, the count may optionally be adjusted by a bias factor. For example, referring to FIG. 1B, the bias logic 146 may determine a need to reduce the counter of the accumulator 142 to compensate for natural return-call imbalances in the system 100. Accordingly, the bias logic 146 may shift the accumulator 142 to the right by one bit in order to divide the counter value by two, thereby reducing the effect of the natural imbalances. After step 316, the sequence 310 continues at step 230 (shown in FIG. 2).

[0044] Optionally, in some embodiments, step 314 may be omitted from the sequence 310. For example, referring to FIG. 1B, the accumulator 142 may increment the count in response to instances of return instructions, without decrementing the count in response to instances of call instructions. Accordingly, in such embodiments, the counter value of the accumulator 142 may not correspond to a return-call imbalance metric, but may instead correspond to a metric of the number of return instructions executed during the predefined window of the accumulator 142.

[0045] Referring now to FIG. 3B, shown is a sequence 320 for determining a metric, in accordance with one or more embodiments. In particular, the sequence 320 illustrates an exemplary expansion of the steps involved in performing step 220 (shown in FIG. 2). In one or more embodiments, the sequence 320 may be part of the ROP detection logic 140 shown in FIG. 1A. The sequence 320 may be implemented in hardware, software, and/or firmware. In firmware and software embodiments it may be implemented by computer executed instructions stored in a non-transitory computer readable medium, such as an optical, semiconductor, or magnetic storage device.

[0046] At step 322, a count may be incremented for each instance of a control transfer instruction associated with a stack pop instruction. For example, referring to FIG. 1B, the accumulator 142 may determine whether a jump instruction and a stack pop instruction are executed within a predefined distance (i.e., a number of instructions) of each other. In some embodiments, the association between the jump instruction and a stack pop instruction may be determined based on whether the target value from the stack pop instruction is used by the jump instruction. Further, in some embodiments, such a pair of associated instructions may be functionally similar to a return instruction, and may thus be indicative of a possible ROP attack. Accordingly, if a pair of associated instructions is detected, the accumulator 142 may increment a counter by one.

[0047] At step 324, the count may be decremented for each instance of a control transfer instruction associated with a stack push instruction. In some embodiments, such a pair of associated instructions may be functionally similar to a call instruction. For example, referring to FIG. 1B, the accumulator 142 may determine whether a

jump instruction and a stack push instruction are executed within a predefined distance, and if so, may decrement the counter by one.

[0048] At step 326, the count may optionally be adjusted by a bias factor. For example, referring to FIG. 1B, the bias logic 146 may divide or otherwise reduce the counter value of the accumulator 142 in order to reduce the effect of the natural imbalances. After step 326, the sequence 320 continues at step 230 (shown in FIG. 2).

[0049] Referring now to FIG. 3C, shown is a sequence 330 for determining a metric, in accordance with one or more embodiments. In particular, the sequence 330 illustrates an exemplary expansion of the steps involved in performing step 220 (shown in FIG. 2). In one or more embodiments, the sequence 330 may be part of the ROP detection logic 140 shown in FIG. 1A. The sequence 330 may be implemented in hardware, software, and/or firmware. In firmware and software embodiments it may be implemented by computer executed instructions stored in a non-transitory computer readable medium, such as an optical, semiconductor, or magnetic storage device.

[0050] At step 332, a count may be incremented for each instance of a control transfer instruction associated with an increase in the stack pointer. For example, referring to FIG. 1B, the accumulator 142 may determine whether a jump instruction is associated with an increase in the stack pointer. In response to such a determination, the accumulator 142 may increment a counter by one.

[0051] At step 334, the count may be decremented for each instance of a control transfer instruction associated with a decrease in the stack pointer. For example, referring to FIG. 1B, the accumulator 142 may determine whether a jump instruction is associated with a decrease in the stack pointer, and if so, may decrement the counter by one.

[0052] At step 336, the count may optionally be adjusted by a bias factor. For example, referring to FIG. 1B, the bias logic 146 may divide or otherwise reduce the counter value of the accumulator 142 in order to reduce the effect of the natural

imbalances. After step 336, the sequence 330 continues at step 230 (shown in FIG. 2).

[0053] Referring now to FIG. 3D, shown is a sequence 340 for determining a metric, in accordance with one or more embodiments. In particular, the sequence 340 illustrates an exemplary expansion of the steps involved in performing step 220 (shown in FIG. 2). In one or more embodiments, the sequence 340 may be part of the ROP detection logic 140 shown in FIG. 1A. The sequence 340 may be implemented in hardware, software, and/or firmware. In firmware and software embodiments it may be implemented by computer executed instructions stored in a non-transitory computer readable medium, such as an optical, semiconductor, or magnetic storage device.

[0054] At step 342, a count may be incremented for each return misprediction. For example, referring to FIG. 1B, the accumulator 142 may interact with the return stack buffer 136 to detect a return misprediction. In response to such a determination, the accumulator 142 may increment a counter by one.

[0055] At step 344, the count may optionally be decremented for each correct return prediction. For example, referring to FIG. 1B, the accumulator 142 may interact with the return stack buffer 136 to detect a correct return prediction. In response to such a determination, the accumulator 142 may decrement a counter by one. Alternatively, in some embodiments, the count may not be decremented for each correct return prediction. For example, in some embodiments, the count may be reset to zero for each correct return prediction. In another example, in some embodiments, the count may not be altered in response to a correct return prediction.

[0056] At step 346, the count may optionally be adjusted by a bias factor. For example, referring to FIG. 1B, the bias logic 146 may divide or otherwise reduce the counter value of the accumulator 142 in order to reduce the effect of the natural imbalances. After step 346, the sequence 340 continues at step 230 (shown in FIG. 2).

[0057] Referring now to FIG. 3E, shown is a sequence 350 for determining a metric, in accordance with one or more embodiments. In particular, the sequence 350 illustrates an exemplary expansion of the steps involved in performing step 220 (shown in FIG. 2). In one or more embodiments, the sequence 350 may be part of the ROP detection logic 140 shown in FIG. 1A. The sequence 350 may be implemented in hardware, software, and/or firmware. In firmware and software embodiments it may be implemented by computer executed instructions stored in a non-transitory computer readable medium, such as an optical, semiconductor, or magnetic storage device.

[0058] At step 352, a count may be incremented for each branch misprediction. For example, referring to FIG. 1B, the accumulator 142 may interact with the branch prediction unit 137 to detect any branch misprediction. In response to such a determination, the accumulator 142 may increment a counter by one. Alternatively, in some embodiments, the accumulator 142 may increment a counter only in response to detecting a particular type of branch misprediction (e.g., a mispredicted indirect branch).

[0059] At step 354, the count may optionally be decremented for each correct branch prediction. For example, referring to FIG. 1B, the accumulator 142 may interact with the branch prediction unit 137 to detect a correct branch prediction. In response to such a determination, the accumulator 142 may decrement a counter by one. Alternatively, in some embodiments, the count may not be decremented for each correct branch prediction. For example, in some embodiments, the count may be reset to zero for each correct branch prediction. In another example, in some embodiments, the count may not be altered in response to a correct branch prediction.

[0060] At step 356, the count may optionally be adjusted by a bias factor. For example, referring to FIG. 1B, the bias logic 146 may divide or otherwise reduce the counter value of the accumulator 142 in order to reduce the effect of the natural imbalances. After step 356, the sequence 350 continues at step 230 (shown in FIG. 2).

[0061] Note that the examples shown in FIGs. 1A-1B, 2, and 3A-3E are provided for the sake of illustration, and are not intended to limit any embodiments. For instance, while the above examples describe incrementing or decrementing the accumulator 142 by one, embodiments are not limited in this regard. For example, the accumulator 142 may be incremented by a first amount (e.g., two, four, five, etc.) in response to a first control transfer event, may be decremented by a second amount in response to a second control transfer event, etc. Further, it is contemplated that the ROP detection logic 140 may use any type of control transfer events, and/or any combination thereof.

[0062] Note also that, while embodiments may be shown in simplified form for the sake of clarity, embodiments may include any number and/or arrangement of processors, cores, and/or additional components (e.g., buses, storage media, connectors, power components, buffers, interfaces, etc.). In particular, it is contemplated that some embodiments may include any number of components (e.g., additional accumulators 142 and/or comparators 144) in addition to those shown, and that different arrangement of the components shown may occur in certain implementations. Further, it is contemplated that specifics in the examples shown in FIGs. 1A-1B, 2, and 3A-3E may be used anywhere in one or more embodiments.

[0063] Referring now to FIG. 4, shown is a block diagram of a processor in accordance with an embodiment of the present invention. As shown in FIG. 4, the processor 400 may be a multicore processor including first die 405 having a plurality of cores 410a – 410n of a core domain. The various cores 410a – 410n may be coupled via an interconnect 415 to a system agent or uncore domain 420 that includes various components. As seen, the uncore domain 420 may include a shared cache 430 which may be a last level cache. In addition, the uncore may include an integrated memory controller 440 and various interfaces 450.

[0064] Although not shown for ease of illustration in FIG. 4, in some embodiments, each of the cores 410a – 410n may include the ROP detection logic 140 shown in FIG. 1A-1B. Alternatively, in some embodiments, some or all of the ROP detection logic 140 may be included in the uncore domain 420, and may thus be shared across the cores 410a – 410n.

[0065] With further reference to FIG. 4, the processor 400 may communicate with a system memory 445, e.g., via a memory bus. In addition, by interfaces 450, connection can be made to various off-package components such as peripheral devices, mass storage and so forth. While shown with this particular implementation in the embodiment of FIG. 4, the scope of the present invention is not limited in this regard.

[0066] Referring now to FIG. 5, shown is a block diagram of a multi-domain processor in accordance with another embodiment of the present invention. As shown in the embodiment of FIG. 5, processor 500 includes multiple domains. Specifically, a core domain 510 can include a plurality of cores 510a–510n, a graphics domain 520 can include one or more graphics engines, and a system agent domain 550 may further be present. Each of the cores 510a–510n can include the ROP detection logic 140 described above with reference to FIGs. 1A-1B. Note that while only shown with three domains, understand the scope of the present invention is not limited in this regard and additional domains can be present in other embodiments. For example, multiple core domains may be present each including at least one core.

[0067] In general, each core 510 may further include low level caches in addition to various execution units and additional processing elements. In turn, the various cores may be coupled to each other and to a shared cache memory formed of a plurality of units of a last level cache (LLC) 540a – 540n. In various embodiments, LLC 550 may be shared amongst the cores and the graphics engine, as well as various media processing circuitry. As seen, a ring interconnect 530 thus couples the cores together, and provides interconnection between the cores, graphics domain 520 and system agent circuitry 550.

[0068] In the embodiment of FIG. 5, system agent domain 550 may include display controller 552 which may provide control of and an interface to an associated display. As further seen, system agent domain 550 may also include a power control unit 555 to allocate power to the CPU and non-CPU domains.

[0069] As further seen in FIG. 5, processor 500 can further include an integrated memory controller (IMC) 570 that can provide for an interface to a system memory, such as a dynamic random access memory (DRAM). Multiple interfaces 580a – 580n may be present to enable interconnection between the processor and other circuitry. For example, in one embodiment at least one direct media interface (DMI) interface may be provided as well as one or more Peripheral Component Interconnect Express (PCI Express™ (PCIe™)) interfaces. Still further, to provide for communications between other agents such as additional processors or other circuitry, one or more interfaces in accordance with an Intel® Quick Path Interconnect (QPI) protocol may also be provided. As further seen, a peripheral controller hub (PCH) 590 may also be present within the processor, and can be implemented on a separate die, in some embodiments. Although shown at this high level in the embodiment of FIG. 5, understand the scope of the present invention is not limited in this regard.

[0070] Referring to FIG. 6, an embodiment of a processor including multiple cores is illustrated. Processor 1100 includes any processor or processing device, such as a microprocessor, an embedded processor, a digital signal processor (DSP), a network processor, a handheld processor, an application processor, a co-processor, a system on a chip (SOC), or other device to execute code. Processor 1100, in one embodiment, includes at least two cores—cores 1101 and 1102, which may include asymmetric cores or symmetric cores (the illustrated embodiment). However, processor 1100 may include any number of processing elements that may be symmetric or asymmetric.

[0071] In one embodiment, a processing element refers to hardware or logic to support a software thread. Examples of hardware processing elements include: a thread unit, a thread slot, a thread, a process unit, a context, a context unit, a logical processor, a hardware thread, a core, and/or any other element, which is capable of holding a state for a processor, such as an execution state or architectural state. In other words, a processing element, in one embodiment, refers to any hardware capable of being independently associated with code, such as a software thread, operating system, application, or other code. A physical processor typically refers to

an integrated circuit, which potentially includes any number of other processing elements, such as cores or hardware threads.

[0072] A core often refers to logic located on an integrated circuit capable of maintaining an independent architectural state, wherein each independently maintained architectural state is associated with at least some dedicated execution resources. In contrast to cores, a hardware thread typically refers to any logic located on an integrated circuit capable of maintaining an independent architectural state, wherein the independently maintained architectural states share access to execution resources. As can be seen, when certain resources are shared and others are dedicated to an architectural state, the line between the nomenclature of a hardware thread and core overlaps. Yet often, a core and a hardware thread are viewed by an operating system as individual logical processors, where the operating system is able to individually schedule operations on each logical processor.

[0073] Physical processor 1100, as illustrated in FIG. 6, includes two cores, cores 1101 and 1102. Here, cores 1101 and 1102 are considered symmetric cores, i.e. cores with the same configurations, functional units, and/or logic. In another embodiment, core 1101 includes an out-of-order processor core, while core 1102 includes an in-order processor core. However, cores 1101 and 1102 may be individually selected from any type of core, such as a native core, a software managed core, a core adapted to execute a native instruction set architecture (ISA), a core adapted to execute a translated ISA, a co-designed core, or other known core. Yet to further the discussion, the functional units illustrated in core 1101 are described in further detail below, as the units in core 1102 operate in a similar manner.

[0074] As shown, core 1101 includes two hardware threads 1101a and 1101b, which may also be referred to as hardware thread slots 1101a and 1101b. Therefore, software entities, such as an operating system, in one embodiment potentially view processor 1100 as four separate processors, i.e., four logical processors or processing elements capable of executing four software threads concurrently. As alluded to above, a first thread is associated with architecture state registers 1101a, a second thread is associated with architecture state registers

1101b, a third thread may be associated with architecture state registers 1102a, and a fourth thread may be associated with architecture state registers 1102b. Here, each of the architecture state registers (1101a, 1101b, 1102a, and 1102b) may be referred to as processing elements, thread slots, or thread units, as described above.

[0075] As illustrated, architecture state registers 1101a are replicated in architecture state registers 1101b, so individual architecture states/contexts are capable of being stored for logical processor 1101a and logical processor 1101b. In core 1101, other smaller resources, such as instruction pointers and renaming logic in allocator and renamer block 1130 may also be replicated for threads 1101a and 1101b. Some resources, such as re-order buffers in reorder/retirement unit 1135, ILTB 1120, load/store buffers, and queues may be shared through partitioning. Other resources, such as general purpose internal registers, page-table base register(s), low-level data-cache and data-TLB 1115, execution unit(s) 1140, and portions of out-of-order unit 1135 are potentially fully shared.

[0076] Processor 1100 often includes other resources, which may be fully shared, shared through partitioning, or dedicated by/to processing elements. In FIG. 6, an embodiment of a purely exemplary processor with illustrative logical units/resources of a processor is illustrated. Note that a processor may include, or omit, any of these functional units, as well as include any other known functional units, logic, or firmware not depicted. As illustrated, core 1101 includes a simplified, representative out-of-order (OOO) processor core. But an in-order processor may be utilized in different embodiments. The OOO core includes a branch target buffer 1120 to predict branches to be executed/taken and an instruction-translation buffer (I-TLB) 1120 to store address translation entries for instructions.

[0077] Core 1101 further includes decode module 1125 coupled to fetch unit 1120 to decode fetched elements. Fetch logic, in one embodiment, includes individual sequencers associated with thread slots 1101a, 1101b, respectively. Usually core 1101 is associated with a first ISA, which defines/specifies instructions executable on processor 1100. Often machine code instructions that are part of the first ISA include a portion of the instruction (referred to as an opcode), which references/specifies an instruction or operation to be performed. Decode logic 1125

includes circuitry that recognizes these instructions from their opcodes and passes the decoded instructions on in the pipeline for processing as defined by the first ISA. As a result of the recognition by decoders 1125, the architecture or core 1101 takes specific, predefined actions to perform tasks associated with the appropriate instruction (e.g., the actions shown in FIGs. 2-3E). It is important to note that any of the tasks, blocks, operations, and methods described herein may be performed in response to a single or multiple instructions; some of which may be new or old instructions.

[0078] In one example, allocator and renamer block 1130 includes an allocator to reserve resources, such as register files to store instruction processing results. However, threads 1101a and 1101b are potentially capable of out-of-order execution, where allocator and renamer block 1130 also reserves other resources, such as reorder buffers to track instruction results. Unit 1130 may also include a register renamer to rename program/instruction reference registers to other registers internal to processor 1100. Reorder/retirement unit 1135 includes components, such as the reorder buffers mentioned above, load buffers, and store buffers, to support out-of-order execution and later in-order retirement of instructions executed out-of-order.

[0079] Scheduler and execution unit(s) block 1140, in one embodiment, includes a scheduler unit to schedule instructions/operation on execution units. For example, a floating point instruction is scheduled on a port of an execution unit that has an available floating point execution unit. Register files associated with the execution units are also included to store information instruction processing results. Exemplary execution units include a floating point execution unit, an integer execution unit, a jump execution unit, a load execution unit, a store execution unit, and other known execution units.

[0080] Lower level data cache and data translation buffer (D-TLB) 1150 are coupled to execution unit(s) 1140. The data cache is to store recently used/operated on elements, such as data operands, which are potentially held in memory coherency states. The D-TLB is to store recent virtual/linear to physical address

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

[0081] Here, cores 1101 and 1102 share access to higher-level or further-out cache 1110, which is to cache recently fetched elements. Note that higher-level or further-out refers to cache levels increasing or getting further away from the execution unit(s). In one embodiment, higher-level cache 1110 is a last-level data cache—last cache in the memory hierarchy on processor 1100—such as a second or third level data cache. However, higher level cache 1110 is not so limited, as it may be associated with or includes an instruction cache. A trace cache—a type of instruction cache—instead may be coupled after decoder 1125 to store recently decoded traces.

[0082] In the depicted configuration, processor 1100 also includes bus interface module 1105 and a power controller 1160, which may perform power sharing control in accordance with an embodiment of the present invention. Historically, controller 1170 has been included in a computing system external to processor 1100. In this scenario, bus interface 1105 is to communicate with devices external to processor 1100, such as system memory 1175, a chipset (often including a memory controller hub to connect to memory 1175 and an I/O controller hub to connect peripheral devices), a memory controller hub, a northbridge, or other integrated circuit. And in this scenario, bus 1105 may include any known interconnect, such as multi-drop bus, a point-to-point interconnect, a serial interconnect, a parallel bus, a coherent (e.g. cache coherent) bus, a layered protocol architecture, a differential bus, and a GTL bus.

[0083] Memory 1175 may be dedicated to processor 1100 or shared with other devices in a system. Common examples of types of memory 1175 include DRAM, SRAM, non-volatile memory (NV memory), and other known storage devices. Note that device 1180 may include a graphic accelerator, processor or card coupled to a memory controller hub, data storage coupled to an I/O controller hub, a wireless transceiver, a flash device, an audio controller, a network controller, or other known device.

[0084] Note however, that in the depicted embodiment, the controller 1170 is illustrated as part of processor 1100. Recently, as more logic and devices are being integrated on a single die, such as SOC, each of these devices may be incorporated on processor 1100. For example in one embodiment, memory controller hub 1170 is on the same package and/or die with processor 1100. Here, a portion of the core (an on-core portion) includes one or more controller(s) 1170 for interfacing with other devices such as memory 1175 or a graphics device 1180. The configuration including an interconnect and controllers for interfacing with such devices is often referred to as an on-core (or un-core configuration). As an example, bus interface 1105 includes a ring interconnect with a memory controller for interfacing with memory 1175 and a graphics controller for interfacing with graphics processor 1180. Yet, in the SOC environment, even more devices, such as the network interface, co-processors, memory 1175, graphics processor 1180, and any other known computer devices/interface may be integrated on a single die or integrated circuit to provide small form factor with high functionality and low power consumption.

[0085] Embodiments may be implemented in many different system types. Referring now to FIG. 7, shown is a block diagram of a system in accordance with an embodiment of the present invention. As shown in FIG. 7, multiprocessor system 600 is a point-to-point interconnect system, and includes a first processor 670 and a second processor 680 coupled via a point-to-point interconnect 650. As shown in FIG. 7, each of processors 670 and 680 may be multicore processors, including first and second processor cores (i.e., processor cores 674a and 674b and processor cores 684a and 684b), although potentially many more cores may be present in the processors. Each of the processors can include the ROP detection logic 140 described above with reference to FIGs. 1A-1B.

[0086] Still referring to FIG. 7, first processor 670 further includes a memory controller hub (MCH) 672 and point-to-point (P-P) interfaces 676 and 678. Similarly, second processor 680 includes a MCH 682 and P-P interfaces 686 and 688. As shown in FIG. 7, MCH's 672 and 682 couple the processors to respective memories, namely a memory 632 and a memory 634, which may be portions of system memory (e.g., DRAM) locally attached to the respective processors. First processor 670 and

second processor 680 may be coupled to a chipset 690 via P-P interconnects 652 and 654, respectively. As shown in FIG. 7, chipset 690 includes P-P interfaces 694 and 698.

[0087] Furthermore, chipset 690 includes an interface 692 to couple chipset 690 with a high performance graphics engine 638, by a P-P interconnect 639. In turn, chipset 690 may be coupled to a first bus 616 via an interface 696. As shown in FIG. 7, various input/output (I/O) devices 614 may be coupled to first bus 616, along with a bus bridge 618 which couples first bus 616 to a second bus 620. Various devices may be coupled to second bus 620 including, for example, a keyboard/mouse 622, communication devices 626 and a data storage unit 628 such as a disk drive or other mass storage device which may include code 630, in one embodiment. Further, an audio I/O 624 may be coupled to second bus 620. Embodiments can be incorporated into other types of systems including mobile devices such as a smart cellular telephone, tablet computer, netbook, Ultrabook™, or so forth.

[0088] It should be understood that a processor core may support multithreading (executing two or more parallel sets of operations or threads), and may do so in a variety of ways 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 (e.g., time sliced fetching and decoding and simultaneous multithreading thereafter such as in the Intel® Hyperthreading technology).

[0089] Any processor described herein may be a general-purpose processor, such as a Core™ i3, i5, i7, 2 Duo and Quad, Xeon™, Itanium™, XScale™ or StrongARM™ processor, which are available from Intel Corporation, of Santa Clara, Calif. Alternatively, the processor may be from another company, such as ARM Holdings, Ltd, MIPS, etc.. The processor 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. The processor may be implemented on one or more chips. The processor 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.

[0090] It is contemplated that the processors described herein are not limited to any system or device. Other system designs and configurations known in the arts for laptops, desktops, handheld PCs, personal digital assistants, engineering workstations, servers, network devices, network hubs, switches, embedded processors, digital signal 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, are also suitable. In general, a huge variety of systems or electronic devices capable of incorporating a processor and/or other execution logic as disclosed herein are generally suitable.

[0091] Embodiments may be implemented in code and may be stored on a non-transitory storage medium having stored thereon instructions which can be used to program a system to perform the instructions. The storage medium may include, but is not limited to, any type of disk including floppy disks, optical disks, solid state drives (SSDs), compact disk read-only memories (CD-ROMs), compact disk rewritables (CD-RWs), and magneto-optical disks, semiconductor devices such as read-only memories (ROMs), random access memories (RAMs) such as dynamic random access memories (DRAMs), static random access memories (SRAMs), erasable programmable read-only memories (EPROMs), flash memories, electrically erasable programmable read-only memories (EEPROMs), magnetic or optical cards, or any other type of media suitable for storing electronic instructions.

[0092] The following clauses and/or examples pertain to further embodiments. One example embodiment may be a processor including at least one execution unit, and a Return Oriented Programming (ROP) detection logic. The ROP detection logic may be to determine a ROP metric based on a plurality of control transfer events, determine whether the ROP metric exceeds a threshold, and in response to a determination that the ROP metric exceeds the threshold, provide a ROP attack notification. The ROP detection logic may be to determine the ROP metric based on a count, where the count is based on the plurality of control transfer events. The ROP detection logic may be to increment the count based on an instance of a

subroutine return instruction. The ROP detection logic may be to decrement the count based on an instance of a subroutine call instruction. The ROP detection logic may be to increment the count based on a return misprediction. The ROP detection logic may be to increment the count based on an instance of a control transfer instruction associated with a stack pop instruction. The ROP detection logic may be to increment the count based on an instance of a control transfer instruction associated with an increase in a stack pointer. The ROP attack notification is to trigger a protection application to take one or more actions to address the ROP attack. The ROP detection logic may be to freeze a branch instruction log in response to determining that the ROP metric exceeds the threshold.

[0093] Another example embodiment may be a processor including: an instruction buffer; a branch prediction unit; and a Return Oriented Programming (ROP) detection unit. The ROP detection unit may include an accumulator to generate a count based on one or more control transfer events, and a comparator to provide a notification of a ROP attack when the count exceeds a threshold during a window. The processor may also include an instruction detector to detect the execution of a control transfer instruction, where the one or more control transfer events comprise the control transfer instruction. The processor may also include a return stack buffer to detect a return misprediction, wherein the one or more control transfer events comprise the return misprediction. The one or more control transfer events may include a branch misprediction, where the branch misprediction is detected by the branch prediction unit. The one or more control transfer events may include pairs of associated instructions. The ROP detection unit may also include bias logic to reduce at least one bias effect due to a natural imbalance. The ROP detection unit may also include control logic to adjust the threshold based on a desired level of protection against ROP attacks.

[0094] Yet another example embodiment may be a system including a processor including Return Oriented Programming (ROP) detection logic, where the ROP detection logic is to determine whether a ROP metric exceeds a threshold during a window, and where the ROP metric is based at least on one or more control transfer events. The system may also include a random access memory coupled to the

processor, where the random access memory includes an anti-malware application. The ROP detection logic may be to, in response to a determination that the ROP metric exceeds the threshold during the window, provide a ROP attack notification to the anti-malware application. The anti-malware application may be to, in response to the ROP attack notification, initiate one or more actions to halt the ROP attack. The window may include a number of instructions.

[0095] Still another example embodiment may be a method, including: detecting, by instruction control of a hardware processor, at least one control transfer event; generating, by ROP detection logic of the hardware processor, a ROP metric based on the at least one control transfer event, and upon determining that the ROP metric exceeds a threshold during a window, notifying a protection application of a ROP attack. Generating the ROP metric may include incrementing a counter upon detecting a control transfer instruction. Generating the ROP metric may include incrementing a counter upon detecting a misprediction. The method may also include, upon determining that the ROP metric exceeds the predefined threshold during the window, freezing the contents of a branch instruction log.

[0096] References throughout this specification to “one embodiment” or “an embodiment” mean that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one implementation encompassed within the present invention. Thus, appearances of the phrase “one embodiment” or “in an embodiment” are not necessarily referring to the same embodiment. Furthermore, the particular features, structures, or characteristics may be instituted in other suitable forms other than the particular embodiment illustrated and all such forms may be encompassed within the claims of the present application.

[0097] While the present invention has been described with respect to a limited number of embodiments for the sake of illustration, those skilled in the art will appreciate numerous modifications and variations therefrom. It is intended that the appended claims cover all such modifications and variations as fall within the true spirit and scope of this present invention.

What is claimed is:

- 1 1. A processor comprising:
2 at least one execution unit; and
3 a Return Oriented Programming (ROP) detection logic to:
4 determine a ROP metric based on a plurality of control transfer events,
5 determine whether the ROP metric exceeds a threshold, and in
6 response to a determination that the ROP metric exceeds the threshold, provide a
7 ROP attack notification.
- 1 2. The processor of claim 1, wherein the ROP detection logic is to determine the
2 ROP metric based on a count, wherein the count is based on the plurality of control
3 transfer events.
- 1 3. The processor of claim 2, wherein the ROP detection logic is to increment the
2 count based on an instance of a subroutine return instruction.
- 1 4. The processor of claim 3, wherein the ROP detection logic is to decrement the
2 count based on an instance of a subroutine call instruction.
- 1 5. The processor of claim 2, wherein the ROP detection logic is to increment the
2 count based on a return misprediction.
- 1 6. The processor of claim 2, wherein the ROP detection logic is to increment the
2 count based on an instance of a control transfer instruction associated with a stack
3 pop instruction.
- 1 7. The processor of claim 2, wherein the ROP detection logic is to increment the
2 count based on an instance of a control transfer instruction associated with an
3 increase in a stack pointer.

- 1 8. The processor of claim 1, wherein the ROP attack notification is to trigger a
2 protection application to take one or more actions to address the ROP attack.
- 1 9. The processor of claim 1, wherein the ROP detection logic is further to freeze
2 a branch instruction log in response to determining that the ROP metric exceeds the
3 threshold.
- 1 10. A processor comprising:
2 an instruction buffer;
3 a branch prediction unit;
4 a Return Oriented Programming (ROP) detection unit comprising:
5 an accumulator to generate a count based on one or more control
6 transfer events, and
7 a comparator to provide a notification of a ROP attack when the count
8 exceeds a threshold during a window.
- 1 11. The processor of claim 10, further comprising an instruction detector to detect
2 the execution of a control transfer instruction, wherein the one or more control
3 transfer events comprise the control transfer instruction.
- 1 12. The processor of claim 10, further comprising a return stack buffer to detect a
2 return misprediction, wherein the one or more control transfer events comprise the
3 return misprediction.
- 1 13. The processor of claim 10, wherein the one or more control transfer events
2 comprise a branch misprediction, and wherein the branch misprediction is detected
3 by the branch prediction unit.
- 1 14. The processor of claim 10, wherein the one or more control transfer events
2 comprise pairs of associated instructions.

- 1 15. The processor of claim 10, the ROP detection unit further comprising bias
2 logic to reduce at least one bias effect due to a natural imbalance.
- 1 16. The processor of claim 10, the ROP detection unit further comprising control
2 logic to adjust the threshold based on a desired level of protection against ROP
3 attacks.
- 1 17. A system comprising:
2 a processor including Return Oriented Programming (ROP) detection logic,
3 wherein the ROP detection logic is to determine whether a ROP metric exceeds a
4 threshold during a window, wherein the ROP metric is based at least on one or more
5 control transfer events; and
6 a random access memory coupled to the processor, wherein the random
7 access memory includes an anti-malware application.
- 1 18. The system of claim 17, wherein the ROP detection logic is further to, in
2 response to a determination that the ROP metric exceeds the threshold during
3 the window, provide a ROP attack notification to the anti-malware application.
- 1 19. The system of claim 18, wherein the anti-malware application is to, in
2 response to the ROP attack notification, initiate one or more actions to halt the
3 ROP attack.
- 1 20. The system of claim 17, wherein the window comprises a number of
2 instructions.
- 1 21. A method, comprising:
2 detecting, by instruction control logic of a hardware processor, at least one
3 control transfer event;
4 generating, by ROP detection logic of the hardware processor, a ROP metric
5 based on the at least one control transfer event, and

6 upon determining that the ROP metric exceeds a threshold during a window,
7 notifying a protection application of a ROP attack.

1 22. The method of claim 21, wherein generating the ROP metric comprises
2 incrementing a counter upon detecting a control transfer instruction.

1 23. The method of claim 21, wherein generating the ROP metric comprises
2 incrementing a counter upon detecting a misprediction.

1 24. The method of claim 21, further comprising, upon determining that the ROP
2 metric exceeds the predefined threshold during the window, freezing the contents of
3 a branch instruction log.

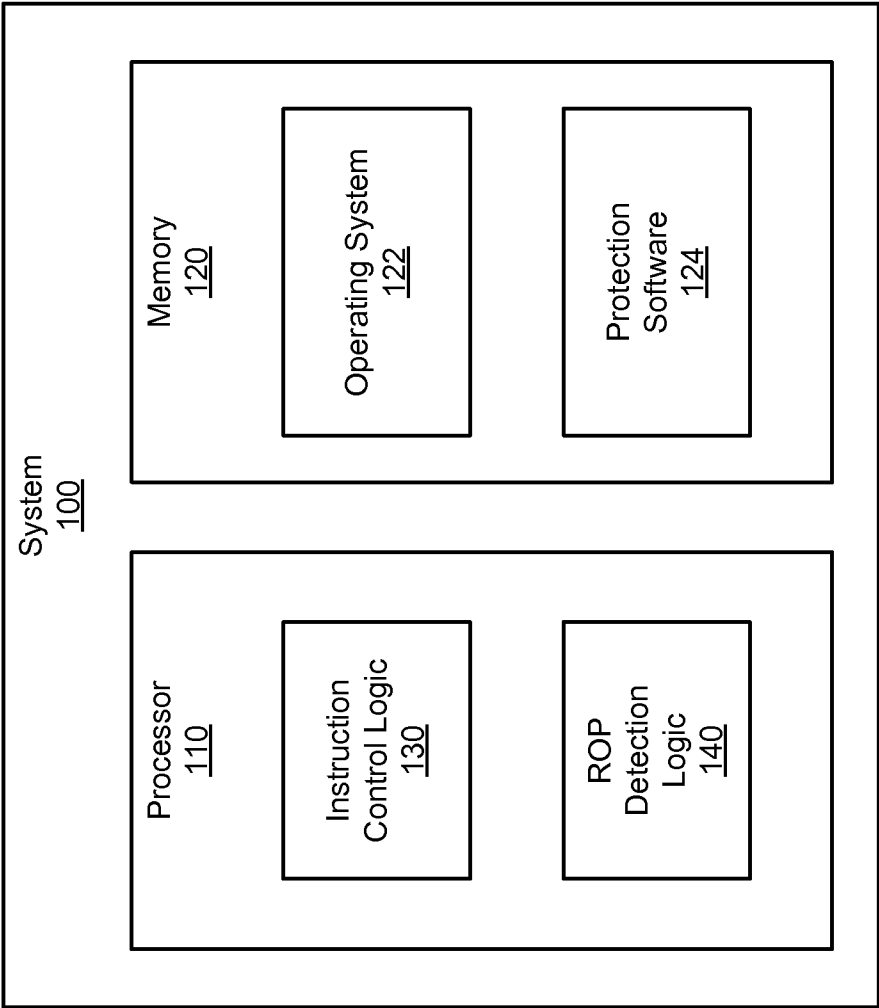


FIG. 1A

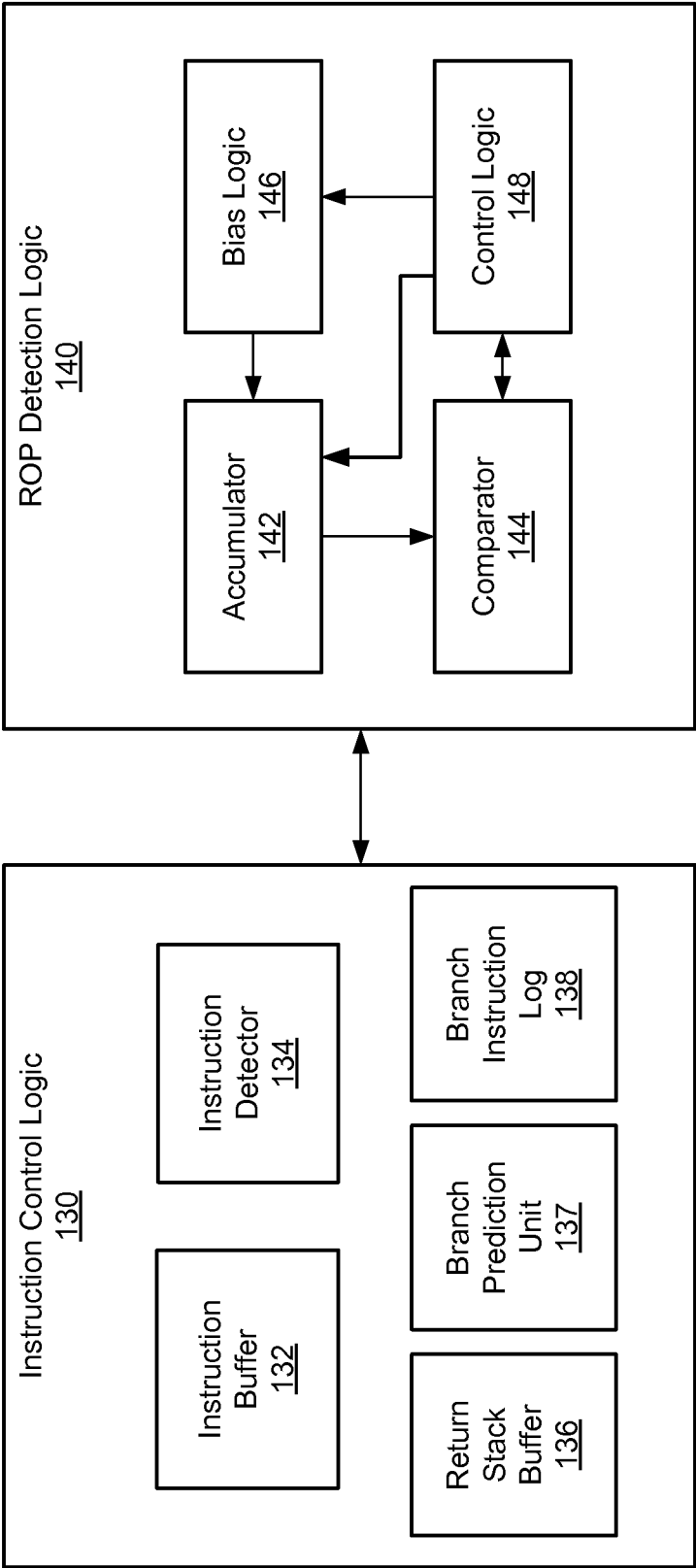


FIG. 1B

3/12

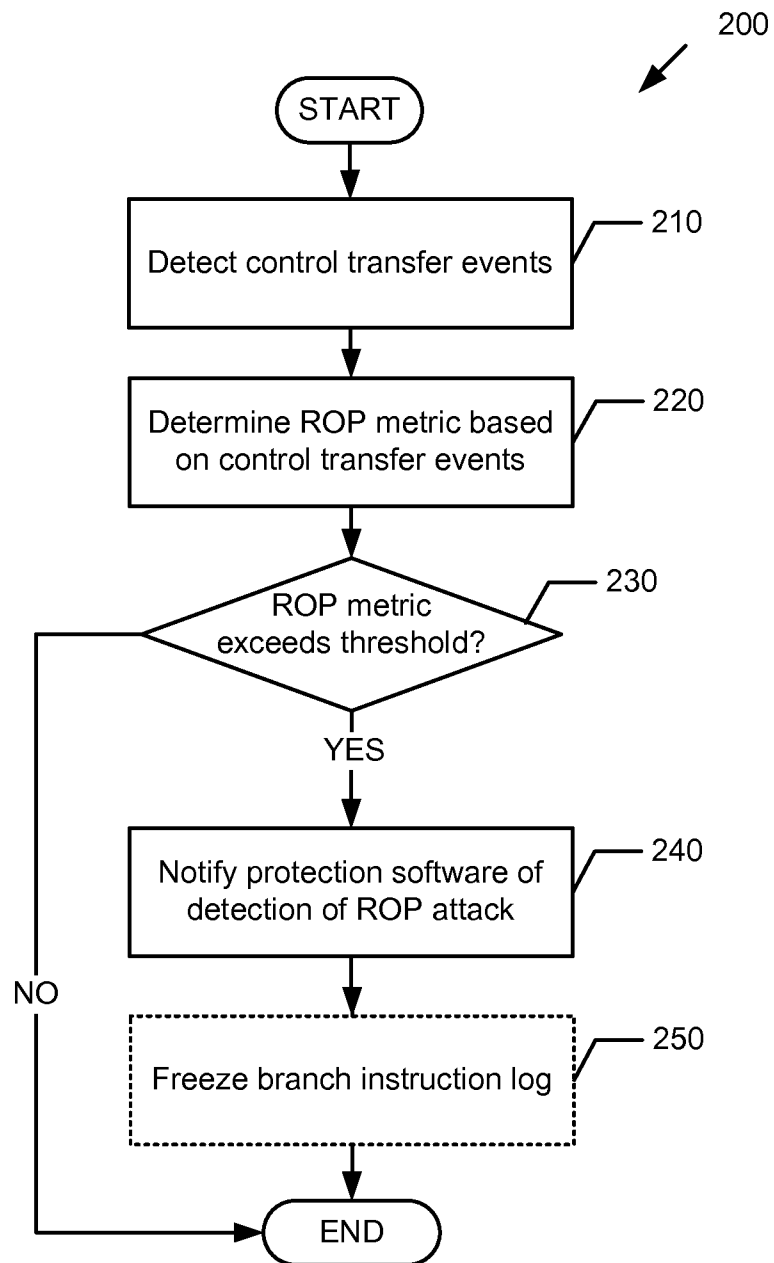


FIG. 2

4/12

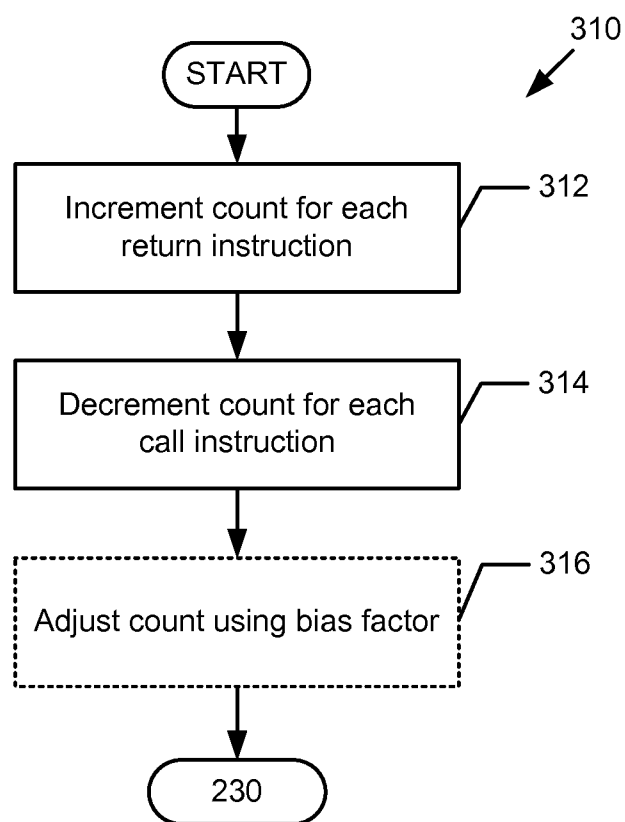


FIG. 3A

5/12

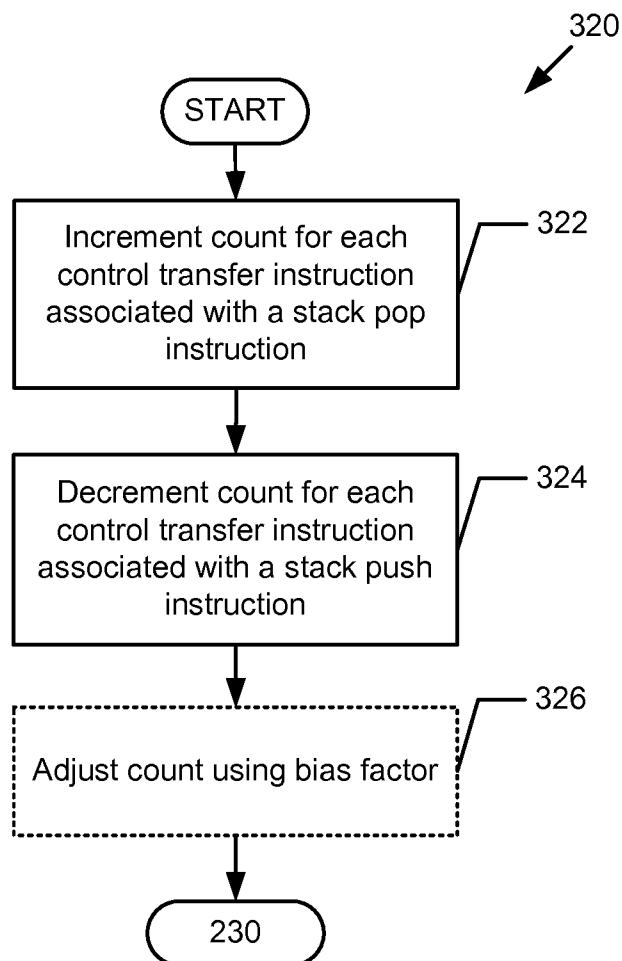


FIG. 3B

6/12

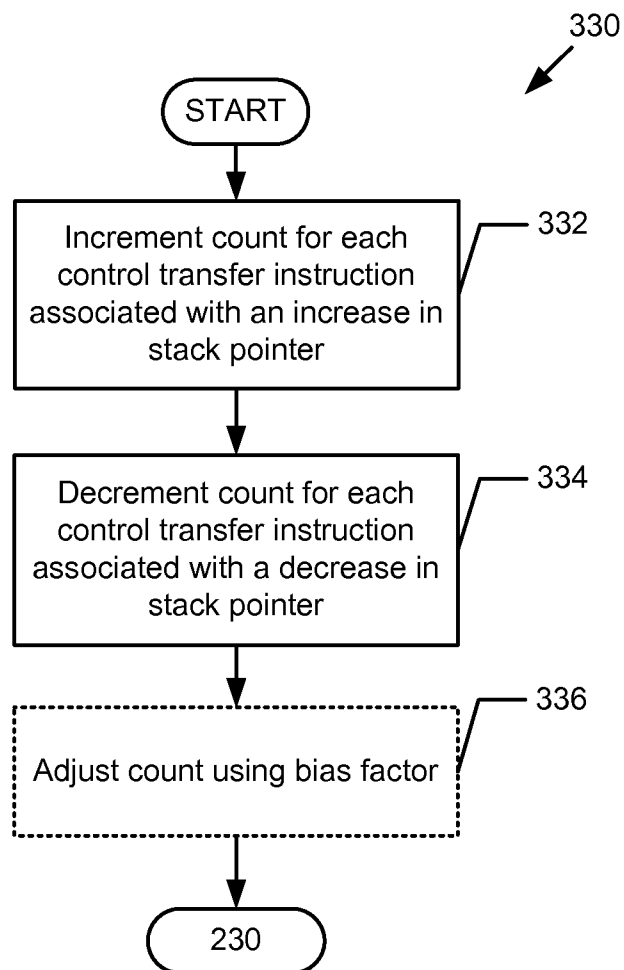


FIG. 3C

7/12

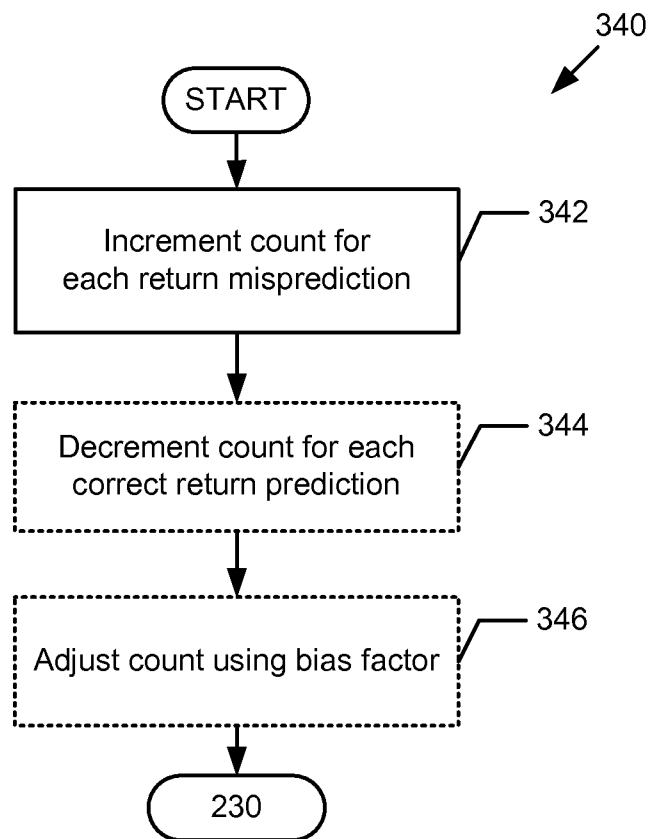


FIG. 3D

8/12

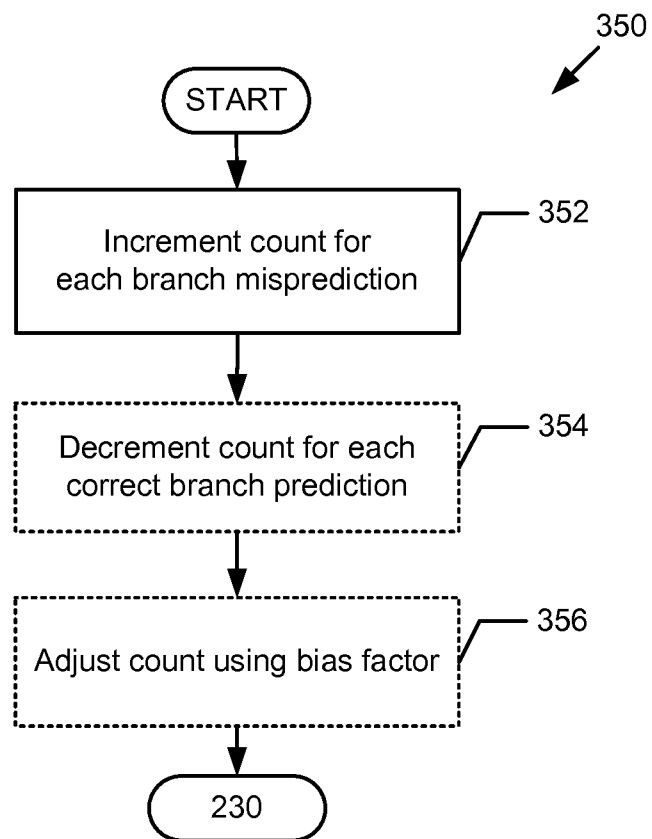


FIG. 3E

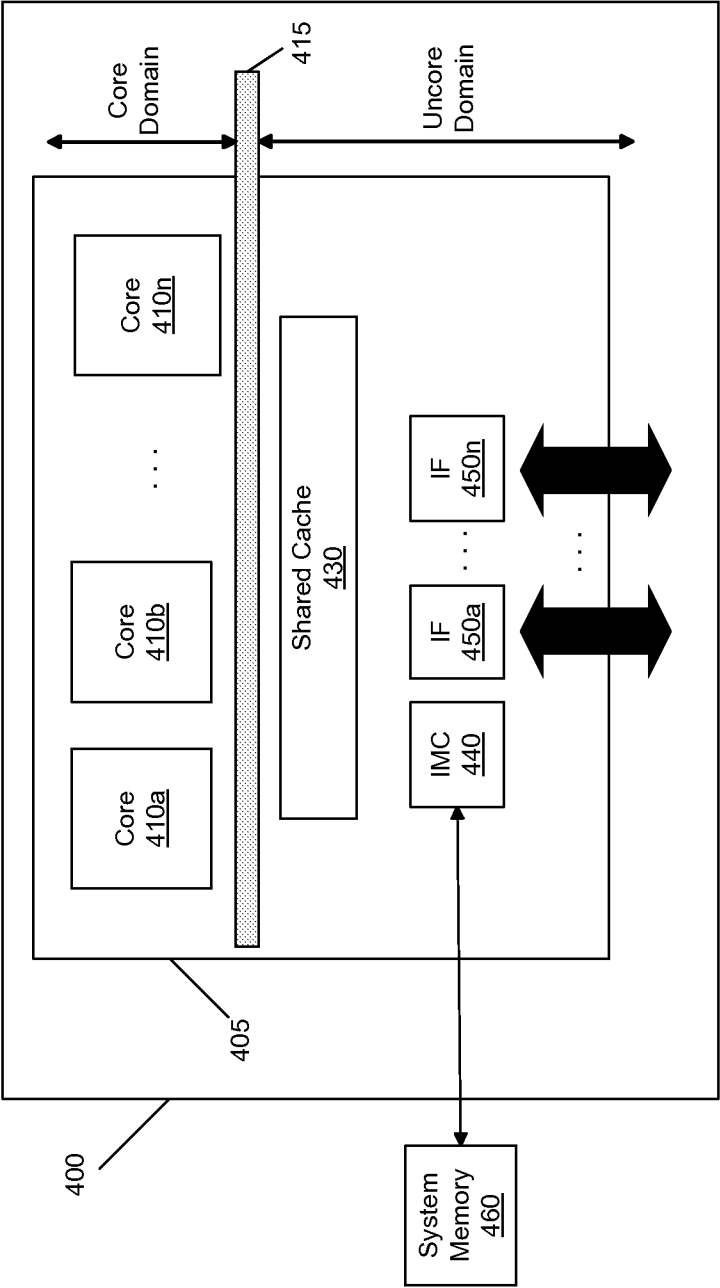


FIG. 4

10/12

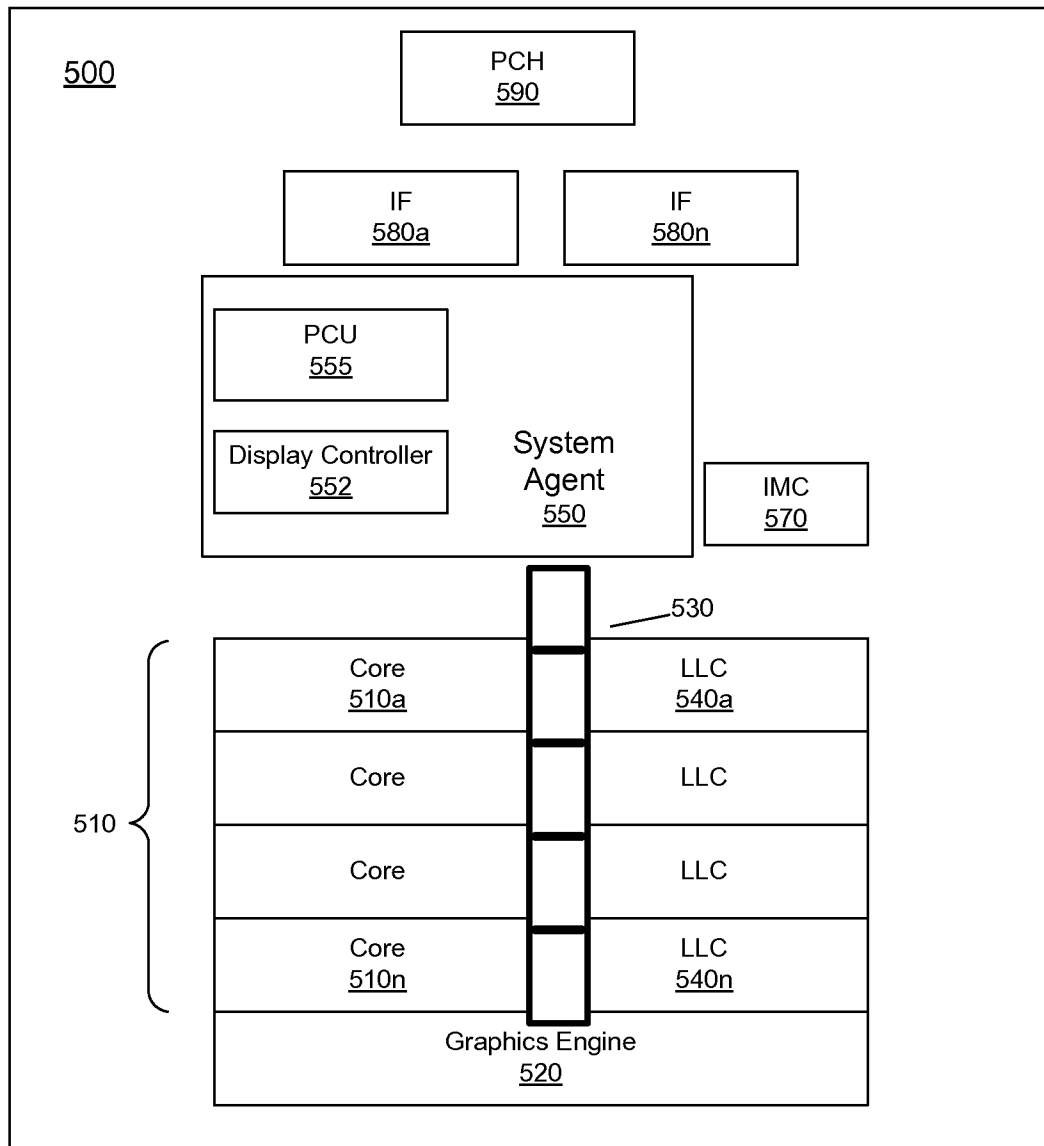


FIG. 5

11/12

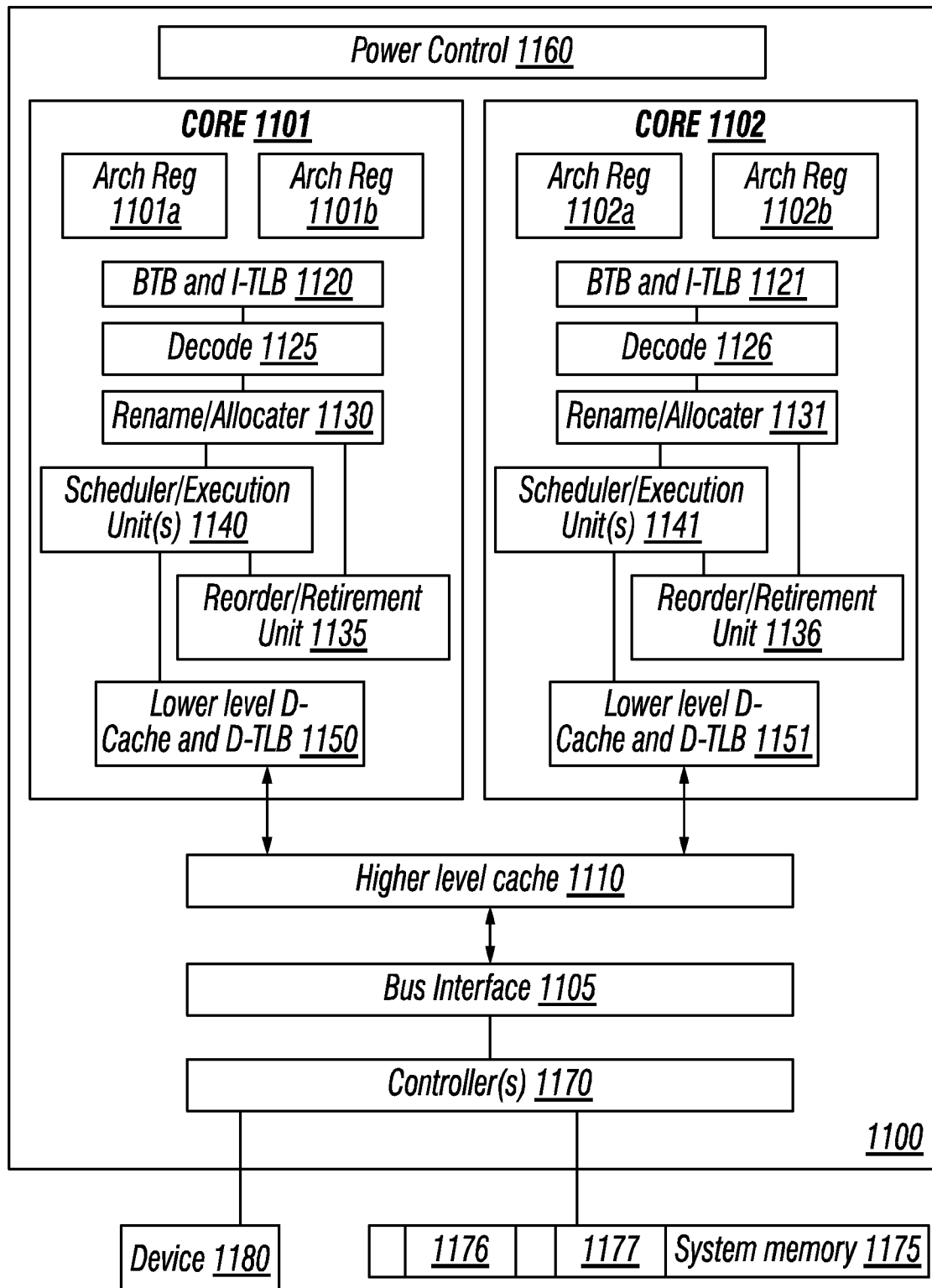


FIG. 6

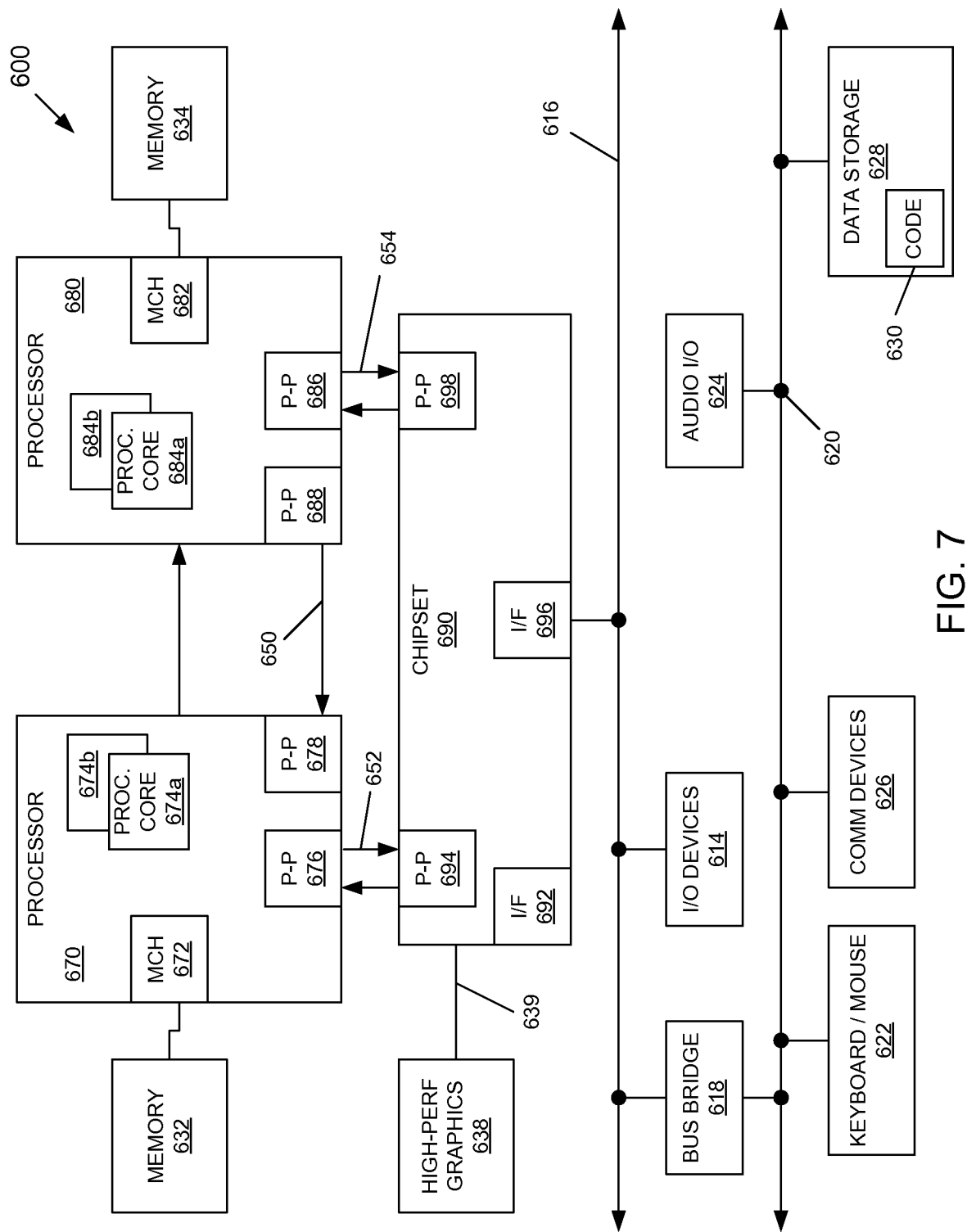


FIG. 7

A. CLASSIFICATION OF SUBJECT MATTER**G06F 21/50(2013.01)i, G06F 11/30(2006.01)i**

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 21/50; G06F 9/54; H04L 9/00; G06F 21/00; G06F 11/30

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: return oriented programming attack, control transfer, instruction buffer, branch prediction, return, call, stack, pointer, pop, push, count, threshold, detect, notify

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	VASILIS PAPPAS, `kBouncer: Efficient and Transparent ROP Mitigation`, 01 April 2012, Retrieved from http://www.cs.columbia.edu/~vpappas/papers/kbouncer.pdf See pages 1-2, 5-6.	1-8,10-14,16-23
A		9,15,24
Y	LIWEI YUAN et al., `Security Breaches as PMU Deviation: Detecting and Identifying Security Attacks Using Performance Counters`, In: Proceedings of the Second Asia-Pacific Workshop on Systems, Shanghai, China, 11-12 July 2011, Article No. 6 See pages 1-3, 5.	1-8,10-14,16-23
A	US 2012-0167120 A1 (DAAVID HENTUNEN) 28 June 2012 See paragraphs [0001]-[0021], [0029]-[0031], [0036]-[0041], [0049]-[0058]; and claims 1, 11, 14.	1-24
A	US 7287283 B1 (PETER SZOR) 23 October 2007 See column 1, lines 6-54; column 2, lines 23-36; column 3, lines 17-36, 47-63; column 4, line 28 - column 6, line 4; column 6, lines 11-27; column 7, line 7 - column 8, line 25; column 8, line 47 - column 9, line 16; claims 1-3, 11; and figures 2, 4-5, 7.	1-24



Further documents are listed in the continuation of Box C.



See patent family annex.

* Special categories of cited documents:

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

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

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of 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 November 2013 (12.11.2013)

Date of mailing of the international search report

14 November 2013 (14.11.2013)

Name and mailing address of the ISA/KR

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

Facsimile No. +82-42-472-7140

Authorized officer

BYUN, Sung Cheal

Telephone No. +82-42-481-8262



INTERNATIONAL SEARCH REPORT

International application No.

PCT/US2013/048531

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2012-0030758 A1 (ERIC VAN DEN BERG et al.) 02 February 2012 See paragraphs [0002]–[0011], [0016]–[0026], [0031]–[0035], [0037]; and claims 1–2.	1–24
A	ERIK BUCHANAN et al., `When Good Instructions Go Bad: Generalizing Return-Oriented Programming to RISC`, In: Proceedings of 15th ACM Conference on Computer and Communications Security 2008, ACM Press, Alexandria, Virginia, USA, 27–31 October 2008, Pages 27–38 See pages 27–30, 32–33, 35.	1–24

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2013/048531

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2012-0167120 A1	28/06/2012	GB 201308692 D0 GB 2499932 A WO 2012-084507 A1	26/06/2013 04/09/2013 28/06/2012
US 7287283 B1	23/10/2007	None	
US 2012-0030758 A1	02/02/2012	None	