(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 July 2013 (04.07.2013)

WIPO | PCT

(10) International Publication Number
WO 2013/101165 Al

(51) International Patent Classification:
G06F 9/30 (2006.01) *G06F 11/14* (2006.01)
G06F 9/305 (2006.01)

(21) International Application Number:
PCT/US201 1/06805 1

(22) International Filing Date:
30 December 201 1 (30. 12.201 1)

(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, CA 95052 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only):* **VERA, Xavier** [ES/ES]; Gran Via Carles III, 87-89B, 2-2, E-08028 Barcelona (ES). **CASADO, Javier, Carretero** [ES/ES]; Calle Gandesa 4,5-4, E-08028 Barcelona (ES). **MONCHIERO, Matteo** [IT/ES]; C De Roger De Flor, 85 3 2B, E-0801 3 Barcelona (ES). **RAMIREZ, Tanausu** [ES/ES]; Jordi Girona 29, 3rd Floor, E-08034 Barcelona (ES). **HERRERO, Enric** [ES/ES]; C/Cucut, 6, E-08440 Cardedeu (ES).

(74) Agents: **VINCENT, Lester, J.** et al; Blakely, Sokoloff, Taylor & Zafhian LLP, 1279 Oakmead Parkway, Sunnyvale, California 94085-4040 (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: REGISTER ERROR PROTECTION THROUGH BINARY TRANSLATION

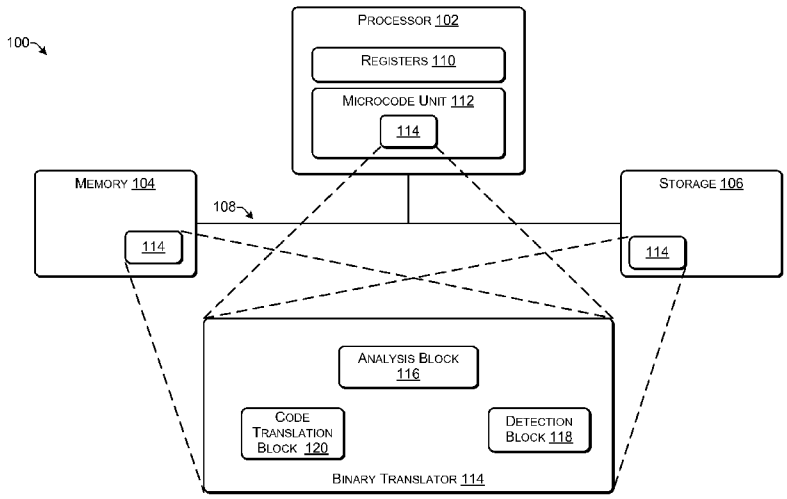

FIG. 1

(57) Abstract: Some implementations provide techniques and arrangements for detecting a register value having a life longer than a threshold period based, at least in part, on at least one code segment of a code being translated by a binary translator. For a register value detected as having a life longer than a threshold period, at least one instruction to cause an access of the detected register value during the life of the register value may be included in at least one translated code segment to be output by the binary translator.

WO 2013/101165 A1

REGISTER ERROR PROTECTION THROUGH BINARY TRANSLATION

TECHNICAL FIELD

[0001] This disclosure relates to the technical field of microprocessors.

BACKGROUND ART

[0002] Soft errors are an increasing problem. Also, the trend toward increasing the number of processor cores creates a need to add reliability features to reduce the per core failure rate, such as the failure in time (FIT) rate. Due to constraints such as latency and overhead, register files are currently protected only by parity, which is a problem when multiple strikes or multiple errors occur. Moreover, parity can only detect an odd number of errors. When registers store register values with long lives (i.e. the values are stored for long periods without being read), multiple errors may occur undetected. This problem is known as Silent Data Corruption.

[0003] One option to avoid Silent Data Corruption is scrubbing the register file. However, scrubbing the register file may have a significant impact on performance. Another option is using stronger error detection and correction codes in register files. However, stronger codes can create significant latency issues and may require greater area on a chip.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] The detailed description is set forth with reference to the accompanying drawing figures. In the figures, the left-most digit(s) of a reference number identifies the figure in which the reference number first appears. The use of the same reference numbers in different figures indicates similar or identical items or features.

[0005] FIG. 1 illustrates an example system according to some implementations.

[0006] FIG. 2 illustrates an example process flow according to some implementations.

[0007] FIG. 3 illustrates select components of an example logic block diagram according to some implementations.

[0008] FIG. 4 illustrates an example translation according to some implementations.

[0009] FIG. 5 illustrates an example translation according to some implementations.

[0010] FIG. 6 illustrates an example translation according to some implementations.

[0011] FIG. 7 illustrates an example system in which some implementations may operate.

DETAILED DESCRIPTION

OVERVIEW

[0012] This disclosure includes techniques and arrangements for detecting long life register values during binary translation and shortening the lives of the detected register values by including instructions to access the register values in the code to be output by the binary translator. Some implementations may use a binary translator to detect register values that have a long life and to include instructions in translated code to shorten the lives of the detected register values. The detection of long life register values may be based on: 1) an analysis of code prior to translation; 2) an analysis of translated code; 3) run-time information regarding the translated code, or 4) any combination thereof.

[0013] Binary translation is a technique that may be used to create a version of a binary program originally executable on one processor-based platform that is, after translation, usable on a different processor-based platform. For example, a binary translator may convert a program intended for an Apple® PowerPC processor based platform into a program executable on an Apple platform based on an Intel® Core Duo® Processor. However, as used herein, binary translation should not be interpreted as limited to converting a program intended for execution on a first platform to a program executable on a second platform. Rather, as used herein, binary translation includes any rewriting of a binary program, such as for dynamic optimization of a program which is already executable on an executing platform. As such, herein, translated code refers to code that has been converted for use on a different platform, code that has been optimized, or code that has been both converted and optimized.

[0014] As alluded to above, programs, or portions of programs, can be optimized when the software is executed (i.e. optimized at "run-time"). This "dynamic optimization" can benefit from profiling information that includes much information

relating the actual execution of the portions of the program executed. Programs can be profiled while operating on test data or while operating on actual end-user data. By profiling software in the end-user environment, the resulting profiling information reflects actual usage patterns, and can aid in the dynamic optimization process.

[0015] In a non-limiting example implementation, dynamic translation and optimization using dynamic profiling can be performed in two phases. The first phase (e.g., a cold translation phase) translates blocks (e.g., a sequence of instructions) of instructions for a first platform to blocks of instructions for a second platform. These cold blocks may or may not be optimized and are instrumented with instructions to measure the different aspects of the execution of the cold blocks. For example, the instrumentation instructions could count a number of times a respective cold block is executed or may compute a number of cycles occurring in the life of a register value. It should be noted that results obtained by instrumentation instructions are not necessarily precise, but rather may be estimates or approximations. After the cold block has been executed (or executed a sufficient number of times to give a reasonably reliable profile), the second phase (e.g., a hot translation phase) begins. In the hot translation phase, the cold blocks are optimized using the profile information generated during execution of the cold blocks. In some implementations, the cold blocks may not all be optimized, but rather only a subset may be optimized. For example, only cold blocks which are found to have long life register values or cold blocks which are frequently executed may be subjected to hot translation phase optimization.

EXAMPLE IMPLEMENTATIONS

[0016] FIG. 1 illustrates an example framework of a processor based system 100 according to some implementations. System 100 is a processor based system including a processor 102, a memory 104 to store data and programs executable by the processor, and a storage unit such as a storage 106 all interconnected by a system bus 108. The processor 102 includes one or more registers 110 and a microcode unit 112.

[0017] In some implementations, a program embodying a binary translator 114 may be stored on the storage 106, loaded into memory 104 via the system bus 108 and executed by the processor 102 on data which may also be stored on the disk system

and optionally in memory 104. In some other implementations, the binary translator 114 may be stored as firmware in the microcode unit 112 on the processor 102 rather than being loaded from memory 104 and storage 106.

[0018] The binary translator 114 includes an analysis block 116, a detection block 118, and a code translation block 120. In general, analysis block 116 analyzes an original code, run-time information and a translated code to generate profiling information. Detection block 118 receives the generated profiling information and a threshold value and, from these, identifies or detects register values which have lives longer than the threshold value. Code translation block 120 uses the profiling information and results generated by detection block 118 to generate translated code. Additional details of the binary translator 114 are described with respect to FIG. 3.

[0019] FIG. 2 illustrates an example process flow 200 according to an implementation in which long life register values are detected and instructions are included in translated code to shorten the lives thereof.

[0020] At 202, code which is to be and/or is being translated by a dynamic binary translator is analyzed and profiled to detect register values with lives longer than a threshold period. As stated above, the analysis and profiling may be performed using the code prior to translation, the translated code, run-time information of the translated code, or any combination thereof.

[0021] As used herein, examples of a life of a register value are 1) a period between the register value being stored in a register R and the next access of register R and 2) the period between an access and a next access of register R while the register value is stored therein. An example of a register value which would be detected as having a life longer than a threshold period often instructions is a register value which is stored in a register at instruction number one (1) and not accessed again until, for example, instruction number eighteen (18). Since the number of instructions in the life of the register value is greater than ten, the register value is detected as having a life longer than the threshold period. As used herein, accessing a register is meant in its broadest sense in that the register value is used in some manner, (i.e. read, moved, copied, used in an operation, etc.) Also, herein, the lives of register values and the threshold period may be discussed in terms of a number of instructions between two points in the

original code. However, this is to simplify discussion herein and should not be interpreted as a limitation. The lives of register values and the threshold period could be measured in terms of any metric which can be used as an indication of a period between two points in the execution of code, such as a number of seconds between the two points, a number of instructions in the translated code between the two points, a number of instructions retired between the two points, or a number of clock cycles that occur between the two points. Further, in some implementations, the threshold period may be a programmable threshold. The programmable threshold may be varied based on various conditions such as voltage levels, an altitude of the executing computer, ambient or internal temperatures, and silent data corruption FIT targets based on a market segment.

[0022] At 204, for a register value detected as having a life longer than the threshold, at least one instruction to cause an access to the register storing the register value is included in the translated code at a point during or at the end of the threshold period. Continuing the above example of a register value which is stored at instruction number one (1) and not accessed again until instruction number eighteen (18), a instruction to copy the register value into a temporary location could be inserted before or after instruction number ten (10) to ensure that the life of the register is limited to ten or less instructions.

[0023] FIG. 3 illustrates a block diagram of an example logical arrangement 300 according to one implementation. The logical arrangement 300 may be implemented as several components of hardware each configured to perform one or more functions, may be implemented in software or firmware where one or more programs are used to perform the different functions, or may be a combination of hardware, firmware, and/or software. For purposes of discussion, the binary translator 114 will be discussed as a set of software routines stored in a computer readable storage medium.

[0024] In operation, analysis block 116 analyzes the original code 302, run-time information 304 and translated code 306 to generate profiling information 308. The profiling information 308 may be generated in the manner discussed above using dynamic profiling. The profiling information is not limited to any particular

measurement or datum but simply provides information needed by detection block 118 and code translation block 120.

[0025] Detection block 118 receives the profiling information 308 and a threshold value 310. From these, detection block 118 identifies or detects register values which have lives longer than the threshold 310. The detection block 118 then outputs detection result 312 including the identification of the detected register values, an indication of locations to insert instructions to access the registers storing the detected register values and/or any information that may be used by the code translation block 120 to include instructions to access the registers storing the detected register values during the threshold period.

[0026] Code translation block 120 uses profiling information 308 and detection result 312 to generate translated code 306. In generating the translated code 306, the code translation block 120 may 1) translate original code 302 from code executable on a first platform to code executable on a second platform; 2) optimize the original code 302 if the original code is already executable by the executing platform; 3) optimize the translated code 306; 4) insert instructions in the translated and/or optimized code to access the registers storing the detected long life register values during the threshold period; or 5) a combination thereof.

[0027] In alternative implementations, the detection block 118 may be integrated into the analysis block 116 or the detection block 118 may receive the original code 302, run-time information 304, translated code 306 and threshold 310 and detect long life register values independent of analysis block 116 and the profiling information 308.

[0028] FIGS. 4, 5, and 6 illustrate example inputs and outputs of binary translator 114 according to some implementations. For simplicity of explanation, in the examples of FIGS. 4, 5, and 6 it is assumed that the original codes have already been, do not need, or will not be translated from one platform to another or optimized. As such, only the detection of long life register values and the inclusion of instructions to shorten the register value lives will be discussed. Further, in the discussion related to FIGS. 4, 5, and 6, the threshold period is a number of instructions between two points in the code.

[0029] FIG. 4 illustrates an example input and output of binary translator 114 in which parameters 400 for the binary translator 114 are a threshold period of two (2) and the access type used to shorten long register value lives is a "read" action (i.e. loading the register value into a temporary location). Specifically, the binary translator receives original code 402 and outputs translated code 404 using parameters 400.

[0030] In operation, binary translator 114 analyzes the original code 402 to determine the lives of register values and to detect register values with lives longer than the threshold period. As illustrated in FIG. 4, the life 406 of register value V is six. Specifically, six instructions (II, 12, 13, 14, 15 and 16) occur between the storage of value V in register r and the next instruction to access register r (the copying of value V into register g. As such, register value V is detected as being a long-life register value.

[0031] Based on this detection, the binary translator 114 includes instructions 408 in translated code 404. Instructions 408 cause the value V stored in register r to be read into register Tmp between instructions 12 and 13 and again between instructions 14 and 15. Thus, the life of register value V is reduced to the threshold period of two instructions between storages and/or accesses. In this way, silent data corruption which could not be detected by parity due to multiple errors occurring over the long life of register value V in the original code 402 may be prevented because multiple errors are less likely to occur during the shortened life of register value V in translated code 404.

[0032] FIG. 5 illustrates an example input and output of binary translator 114 in which parameters 500 for the binary translator 114 are a threshold period of three instructions (3) and the access type used to shorten long register value lives is a "move" action (i.e. moving the register value to a different register). Specifically, the binary translator receives original code 502 and outputs translated code 504 using parameters 500.

[0033] In operation, binary translator 114 analyzes the original code 502 to determine the lives of register values and to detect register values with lives longer than the threshold period. As illustrated in FIG. 5, the life 506 of register value W is

seven. Specifically, seven instructions (II, 12, 13, 14, 15, 16, and 17) occur between the storage of value W in register r and the next instruction to access register r (the copying of value W into register g). As such, register value W is detected as being a long-life register value.

[0034] Based on this detection, the binary translator 114 includes instructions 508 and 510 in the translated code 504. Instruction 508 causes the value W to be moved to register Tmpl between instructions 13 and 14. Next, because the number of instructions between instruction 508 and the original end of the life of register value W is four, which is greater than the threshold of three, binary translator 114 inserts another instruction 510 to move the value W from Tmpl to register Tmp2. Because the four instruction life is not a multiple of the threshold of three, binary translator 114 may vary from inserting the instruction 510 at the end of the threshold period to more evenly split the life of the register value. This is shown in translated code 504 with the placement of instruction 510 between instructions 15 and 16 rather than at the end of the threshold period (between instructions 16 and 17).

[0035] At the original end of the life of register value W, instruction 512 is included in translated code 504. Instruction 512 loads register value W into register g from register Tmp2 rather than register r. Thus, the life of register value W is prevented from exceeding the threshold period of three instructions between accesses. Further, the use of "moves" further prevents any register from even storing a register value for longer than the threshold period. In this way, silent data corruption which could not be detected by parity due to multiple errors occurring over the long life of register value W in the original code 502 may be prevented because multiple errors are less likely to occur during the shortened life of register value W in translated code 504 and because of the reduced period for which any given register stores register value W.

[0036] As discussed above, in the original code 502, the life of register value W was greater than the threshold but not a multiple of the threshold. To handle this, binary translator 114 inserted multiple access instructions and subdivided the last period of the life of register value W in original code 502 into periods smaller than the threshold to evenly distribute the inserted accesses. However, implementations are not limited to this approach. The manner in which long lives of register values are subdivided can

be customized according to the details of the implementation. For example, in an implementations which is less aggressive, the register value lives may be shortened into periods that are substantially longer than the detection threshold value. In short, many variations on the manner in which long register value lives are shortened would be apparent to one of ordinary skill in the art in view of the disclosure provided herein.

[0037] FIG. 6 illustrates an example input and output of binary translator 114 in which parameters 600 for the binary translator 114 are a threshold period of three instructions (3) and the access type used to shorten long register value lives is a "move to memory" action (i.e. moving the register value to cache or system memory). Specifically, the binary translator receives original code 602 and outputs translated code 604 using parameters 600.

[0038] In some situations, moving a register value to cache or to system memory may be preferable to keeping the register value in the register file. For example, in the case of very long life register values, the overhead generated by including many "read" and "move" type accesses over that life may be greater than the overhead generated by moving the value to cache or system memory and then back to the register file. Because cache and system memory use stronger error detection and error correction capabilities than the parity protection typically used for registers, values in cache and system memory can have long lives with less chance of errors such as silent data corruption.

[0039] In operation, binary translator 114 analyzes the original code 502 to determine the lives of register values and to detect register values with lives longer than the threshold period. As illustrated in FIG. 6, the life 606 of register value X is eight. Specifically, eight instructions (II, 12, 13, 14, 15, 16, 17, and 18) occur between the storage of value X in register r and the next instruction to access register r (the copying of value X into register g). As such, register value X is detected as being a long-life register value.

[0040] Based on this detection, the binary translator 114 includes instructions 608 and 610 in translated code 604. Instruction 608 causes the value X to be moved to memory location M[k] before instruction II. Instruction 610 is included at the

original end of the life of register value X and causes register value X to be loaded into register g from memory location M[k]. Thus, the life of register value X is prevented from exceeding the threshold period of three instructions between accesses. In this way, silent data corruption which could not be detected by parity due to multiple errors occurring over the long life of register value X in the original code 502 may be prevented. Further, the use of the "move to memory" access type removes the need for multiple "read" and "move" type accesses which might have a serious impact on the performance of translated code 604.

[0041] In the preceding description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the described implementations, however, one skilled in the art will appreciate that many other implementations may be practiced without these specific details.

[0042] While several examples have been illustrated herein for discussion purposes, numerous other configurations may be used and thus implementations herein are not limited to any particular configuration or arrangement. For example, the discussion herein refers to signals being output and received by particular components of the binary translator. This should not be taken as a limitation as such communication need not be direct and the particular components need not necessarily be separate functional units. These and other variations would be apparent to one of ordinary skill in the art in view of the disclosure provided herein.

[0043] The processes described herein are only examples provided for discussion purposes. Numerous other variations will be apparent to those of skill in the art in light of the disclosure herein. Further, while the disclosure herein sets forth several examples of suitable frameworks, architectures and environments for executing the techniques and processes herein, implementations herein are not limited to the particular examples shown and discussed.

[0044] The processes illustrated herein are shown as a collection of operations in a logical flow graph, which represents a sequence of operations, some or all of which can be implemented in hardware, software or a combination thereof. In the context of software, the blocks represent computer-executable instructions stored on one or more computer-readable media that, when executed by one or more processors, perform the

recited operations. Generally, computer-executable instructions include routines, programs, objects, components, data structures and the like that perform particular functions or implement particular abstract data types. The order in which the operations are described is not intended to be construed as a limitation. Any number of the described blocks can be combined in any order and/or in parallel to implement the process, and not all of the blocks need be executed.

EXAMPLE SYSTEM ARCHITECTURE

[0045] FIG. 7 illustrates select components of an example architecture of a system 700 as an example of the environment in which some implementations may operate. System 700 may encompass any of a variety of devices, such as a desktop computing device, a laptop computing device, tablet computing device, netbook computing device, wireless computing device, or a server computing device, as several nonlimiting examples.

The system 700 may include one or more cores 702 in one or more processors, such as a processor 704, a clock generator 706, a memory 708 (e.g., random access memory (RAM)), an input/output control hub 710, and a power source 712 (e.g., a battery or a power supply). The processor 704 may include a single core 702-1 or multiple cores 702-1, …, 702-N (where N is an integer >1), such as a first core 702-1 and one or more additional cores, up to and including an Nth core 702-N. The processor 704 may include a memory controller 714 to enable access (e.g., reading from or writing) to the memory 708. In some implementations, at least one of the memory 708 and mass storage 716 may include may store instructions, which when executed by the processor 704, cause processor 704 to perform the functions of the binary translator 114.

[0046] The clock generator 706 may generate a clock signal that is the basis for an operating frequency of one or more of the N cores 702 of the processor 704. For example, one or more of the N cores 702 may operate at a multiple of the clock signal generated by the clock generator 706.

[0047] The input/output control hub 710 may be coupled to a mass storage 716. The mass storage 716 may include one or more non-volatile storage devices, such as disk

drives, solid state drives, and the like. An operating system 718 may be stored in the mass storage 716.

[0048] The input/output control hub 710 may be coupled to a network port 720. The network port 720 may enable the system 700 to communicate with other devices via a network 722. The network 722 may include multiple networks, such as wireline networks (e.g., public switched telephone network and the like), wireless networks (e.g., 802.11, code division multiple access (CDMA), global system for mobile (GSM), Long Term Evolution (LTE) and the like), other types of communication networks, or any combination thereof. The input/output control hub 710 may be coupled to a display device 724 that is capable of displaying text, graphics, and the like.

[0049] As described herein, the processor 704 may include multiple computing units or multiple cores. The processor 704 can be implemented as one or more microprocessors, microcomputers, microcontrollers, digital signal processors, central processing units, state machines, logic circuitries, and/or any devices that manipulate signals based on operational instructions. Among other capabilities, the processor 704 can be configured to fetch and execute computer-readable instructions stored in the memory 708 or other computer-readable media.

[0050] The memory 708 is an example of computer storage media for storing instructions which are executed by the processor 704 to perform the various functions described above. The memory 708 may generally include both volatile memory and non-volatile memory (e.g., RAM, ROM, or the like). The memory 708 may be referred to as memory or computer storage media herein, and may be a non-transitory media capable of storing computer-readable, processor-executable program instructions as computer program code that can be executed by the processor 704 as a particular machine configured for carrying out the operations and functions described in the implementations herein.

[0051] Furthermore, this disclosure provides various example implementations, as described and as illustrated in the drawings. However, this disclosure is not limited to the implementations described and illustrated herein, but can extend to other implementations, as would be known or as would become known to those skilled in

the art. Reference in the specification to "one implementation," "this implementation," "these implementations" or "some implementations" means that a particular feature, structure, or characteristic described is included in at least one implementation, and the appearances of these phrases in various places in the specification are not necessarily all referring to the same implementation.

CONCLUSION

[0052] Although the subject matter has been described in language specific to structural features and/or methodological acts, the subject matter defined in the appended claims is not limited to the specific features or acts described above. Rather, the specific features and acts described above are disclosed as example forms of implementing the claims. This disclosure is intended to cover any and all adaptations or variations of the disclosed implementations, and the following claims should not be construed to be limited to the specific implementations disclosed in the specification. Instead, the scope of this document is to be determined entirely by the following claims, along with the full range of equivalents to which such claims are entitled.

CLAIMS

1. A processor comprising:

a plurality of registers; and

an instruction generation logic to generate at least one instruction to periodically read information from at least one register of the plurality of registers in order to detect whether the at least one register of the plurality of registers contains erroneous information.

2. A processor as recited in claim 1, further comprising a detection logic to detect at least one register of the plurality registers that is to store information that will not be accessed for longer than a threshold period based, at least in part, on at least a portion of a code being translated by a binary translator.

3. A processor as recited in claim 2, wherein the instruction generation logic is to generate the at least one instruction to periodically read information from the at least one register detected by the detection logic.

4. A processor as recited in claim 2, further comprising:

an analysis logic to analyze at least a portion of the code being translated by the binary translator; and

wherein the detection logic is to detect the at least one register that is to store the information that will not be accessed for longer than the threshold period based, at least in part, on a result of an analysis performed by the analysis logic.

5. A processor as recited in claim 2, wherein the detection logic is to detect the at least one register that is to store the information that will not be accessed for longer than the threshold period based, at least in part, on dynamic profiling of at least a portion of the code by the binary translator.

6. A processor as recited in claim 2, wherein the threshold period is programmable.

7. A system comprising:

a memory;

a processor coupled to the memory, the processor comprising:

a plurality of registers; and

an instruction generation logic to generate at least one instruction to periodically move information from at least one register of the plurality of registers to at least one other register of the plurality of registers in order to help prevent the information from becoming corrupt.

8. A system as recited in claim 7, the processor further comprising a detection logic to detect at least one register of the plurality registers that is to store information that will not be accessed for longer than a threshold period based, at least in part, on at least a portion of a code being translated by a binary translator.

9. A system as recited in claim 8, wherein the instruction generation logic generates the at least one instruction to periodically move information from the at least one register detected by the detection logic.

10. A system as recited in claim 8, the processor further comprising:

an analysis logic to analyze at least a portion of the code being translated by the binary translator; and

wherein the detection logic is to detect the at least one register that is to store the information that will not be accessed for longer than the threshold period based, at least in part, on a result of an analysis performed by the analysis logic.

11. A system as recited in claim 8, wherein the detection is based, at least in part, on dynamic profiling of at least a portion of the code by the binary translator.

12. A system as recited in claim 8, wherein the threshold period is programmable.

13. A processor comprising:

a plurality of registers; and

a logic to cause information stored in at least one of the plurality of registers to be stored into a memory location based at least in part on a determination that the information will be stored in the at least one of the plurality of registers for at least a threshold period.

14. A processor as recited in claim 13, further comprising a detection logic to detect at least one register of the plurality registers that is to store information that will not be accessed for longer than a threshold period based, at least in part, on at least a portion of a code being translated by a binary translator.

15. A processor as recited in claim 14, wherein the instruction generation logic generates at least one instruction to cause information stored in the at least one register detected by the detection logic to be stored into a memory location.

16. A processor as recited in claim 14, further comprising:

an analysis logic to analyze at least a portion of the code being translated by the binary translator; and

wherein the detection logic detects the at least one register that is to store the information that will not be accessed for longer than the threshold period based, at least in part, on a result of an analysis performed by the analysis logic.

17. A processor as recited in claim 14, wherein the detection is based, at least in part, on dynamic profiling of at least a portion of the code by the binary translator.

18. A processor as recited in claim 14, wherein the threshold period is programmable.

19. A method comprising:

detecting, by a processor comprising a plurality of registers, a register that is to store a register value that will not be accessed for longer than a threshold period based, at least in part, on at least one code segment of a code being translated by a binary translator; and

generating at least one instruction to cause an access of the detected register during a life of the register value.

20. A method as recited in claim 19, wherein the threshold period is programmable.

21. A computer-readable medium maintaining instructions to be executed by a processor to cause the processor to perform operations comprising:

detecting a register that is to store a register value that will not be accessed for longer than a threshold period based, at least in part, on at least one code segment of a code being translated by a binary translator; and

causing the binary translator to include, in at least one translated code segment to be output by the binary translator, at least one instruction to cause an access of the detected register during a life of the register value.

22. A computer-readable medium as recited in claim 21, wherein the threshold period is programmable.

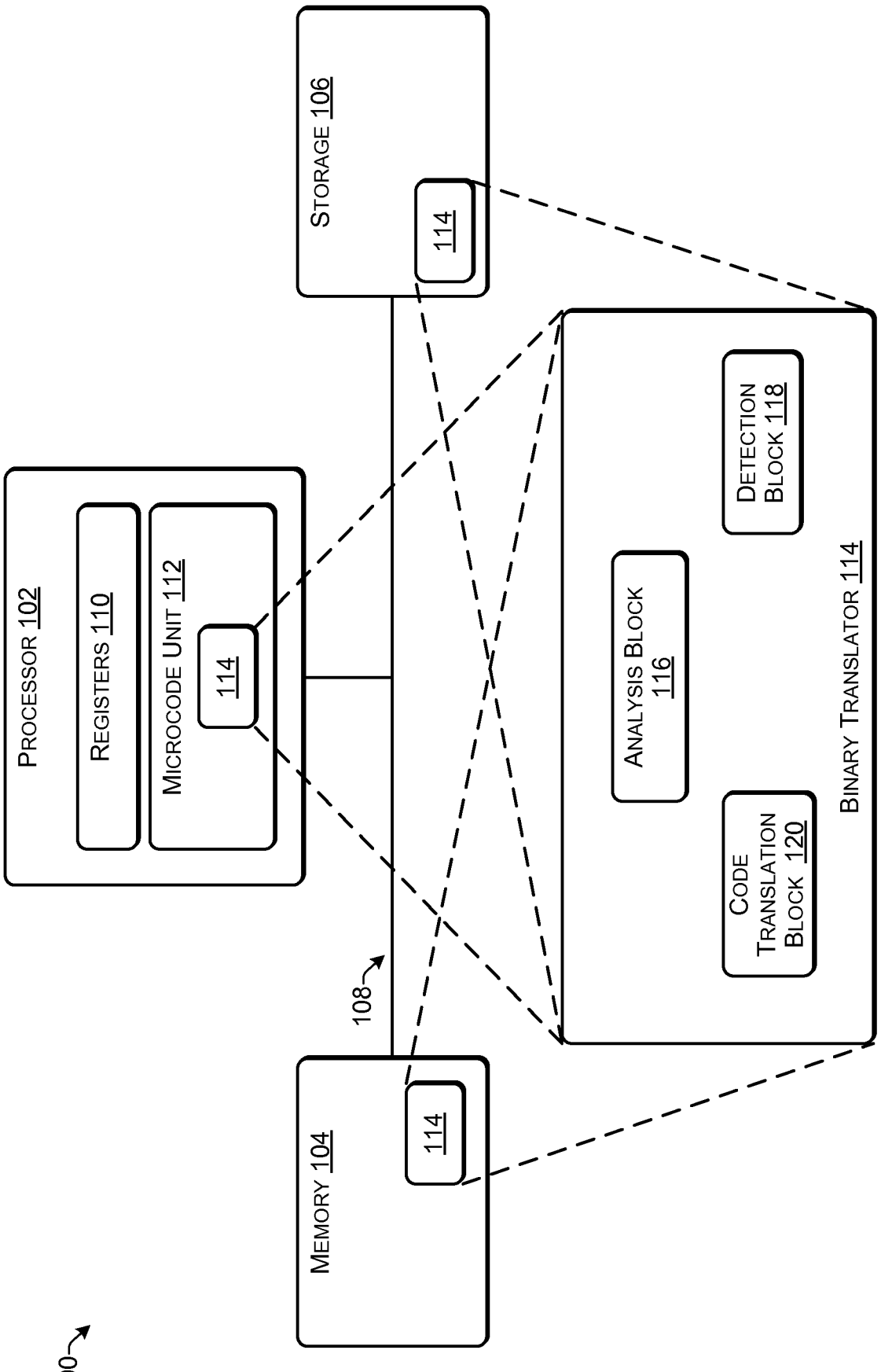
100
PROCESSOR 102
REGISTERS 110
MICROCODE UNIT 112
114
STORAGE 106
114
108
MEMORY 104
114
BINARY TRANSLATOR 114
ANALYSIS BLOCK 116
DETECTION BLOCK 118
CODE TRANSLATION BLOCK 120

FIG. 1

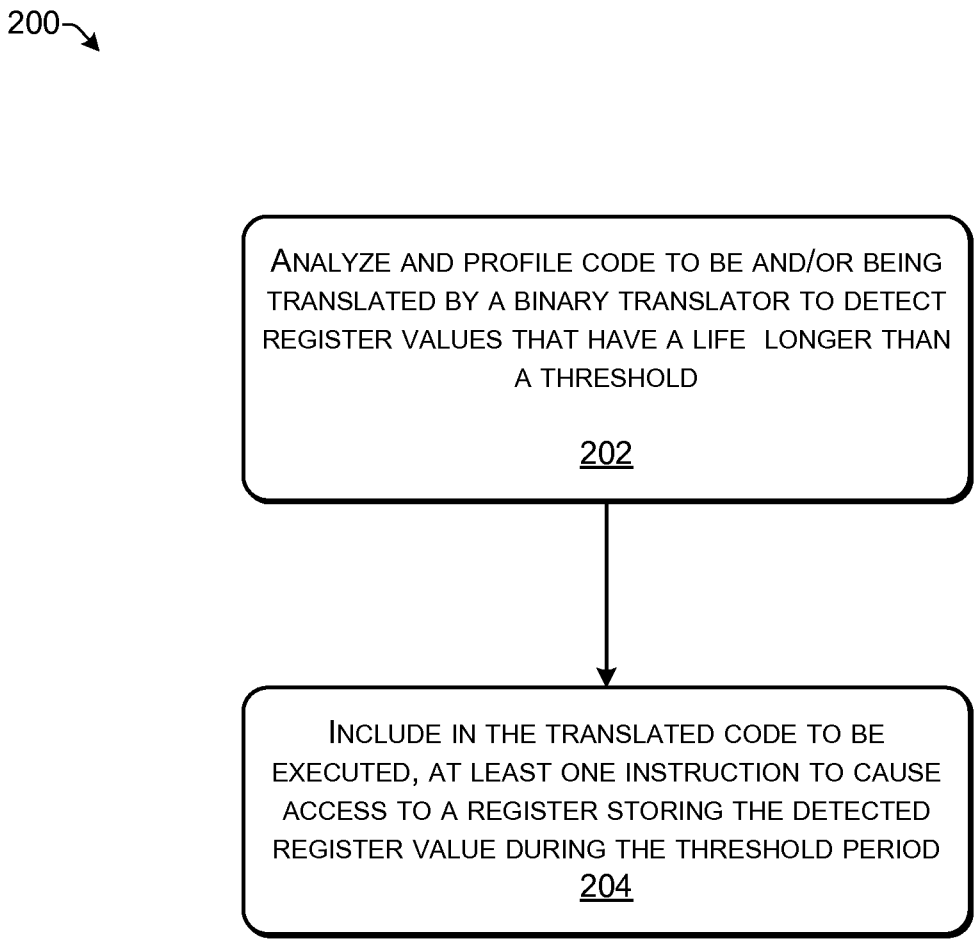
200
ANALYZE AND PROFILE CODE TO BE AND/OR BEING TRANSLATED BY A BINARY TRANSLATOR TO DETECT REGISTER VALUES THAT HAVE A LIFE LONGER THAN A THRESHOLD
202
INCLUDE IN THE TRANSLATED CODE TO BE EXECUTED, AT LEAST ONE INSTRUCTION TO CAUSE ACCESS TO A REGISTER STORING THE DETECTED REGISTER VALUE DURING THE THRESHOLD PERIOD
204

FIG. 2

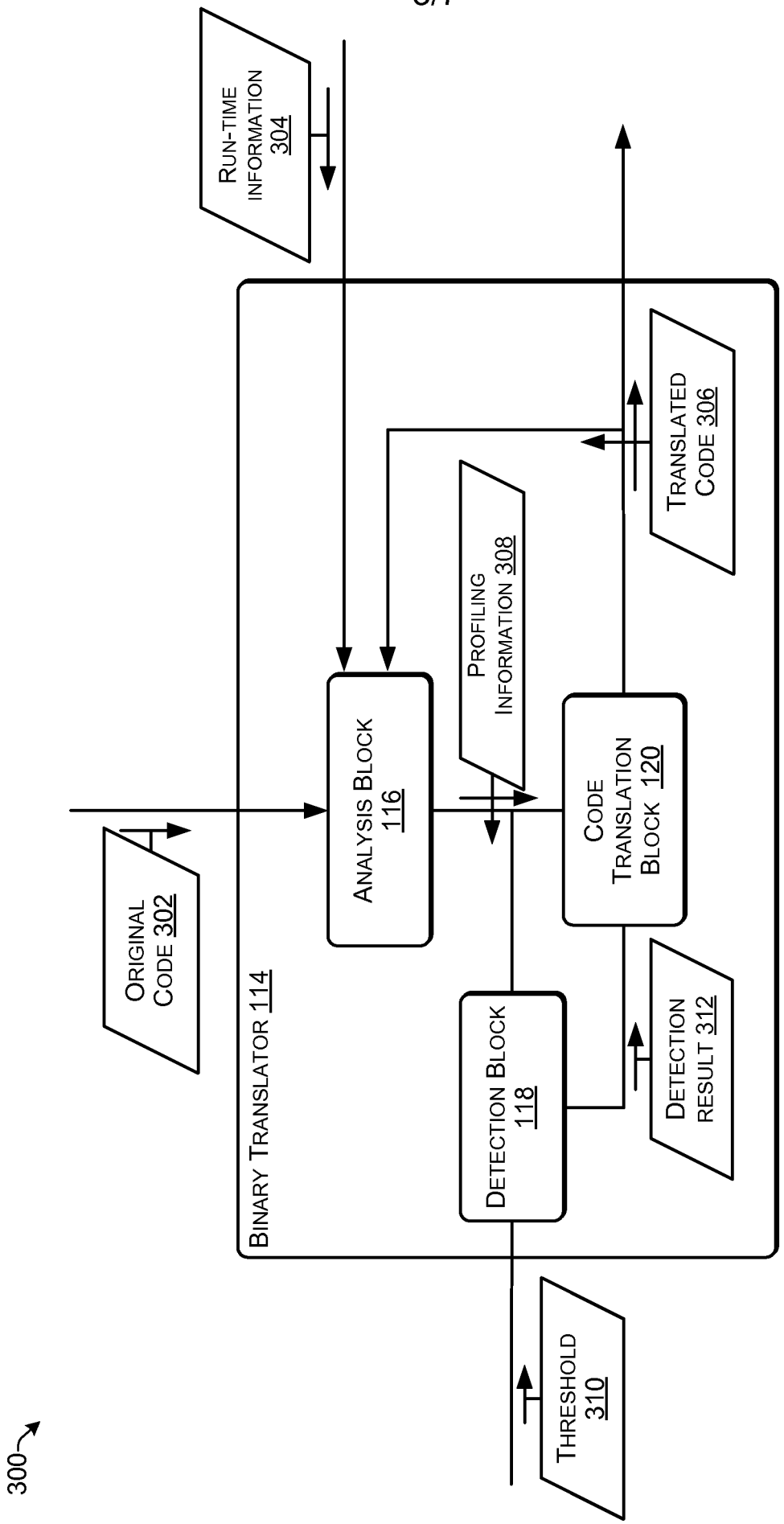
300
RUN-TIME INFORMATION 304
BINARY TRANSLATOR 114
ORIGINAL CODE 302
ANALYSIS BLOCK 116
PROFILING INFORMATION 308
CODE TRANSLATION BLOCK 120
TRANSLATED CODE 306
DETECTION BLOCK 118
DETECTION RESULT 312
THRESHOLD 310

FIG. 3

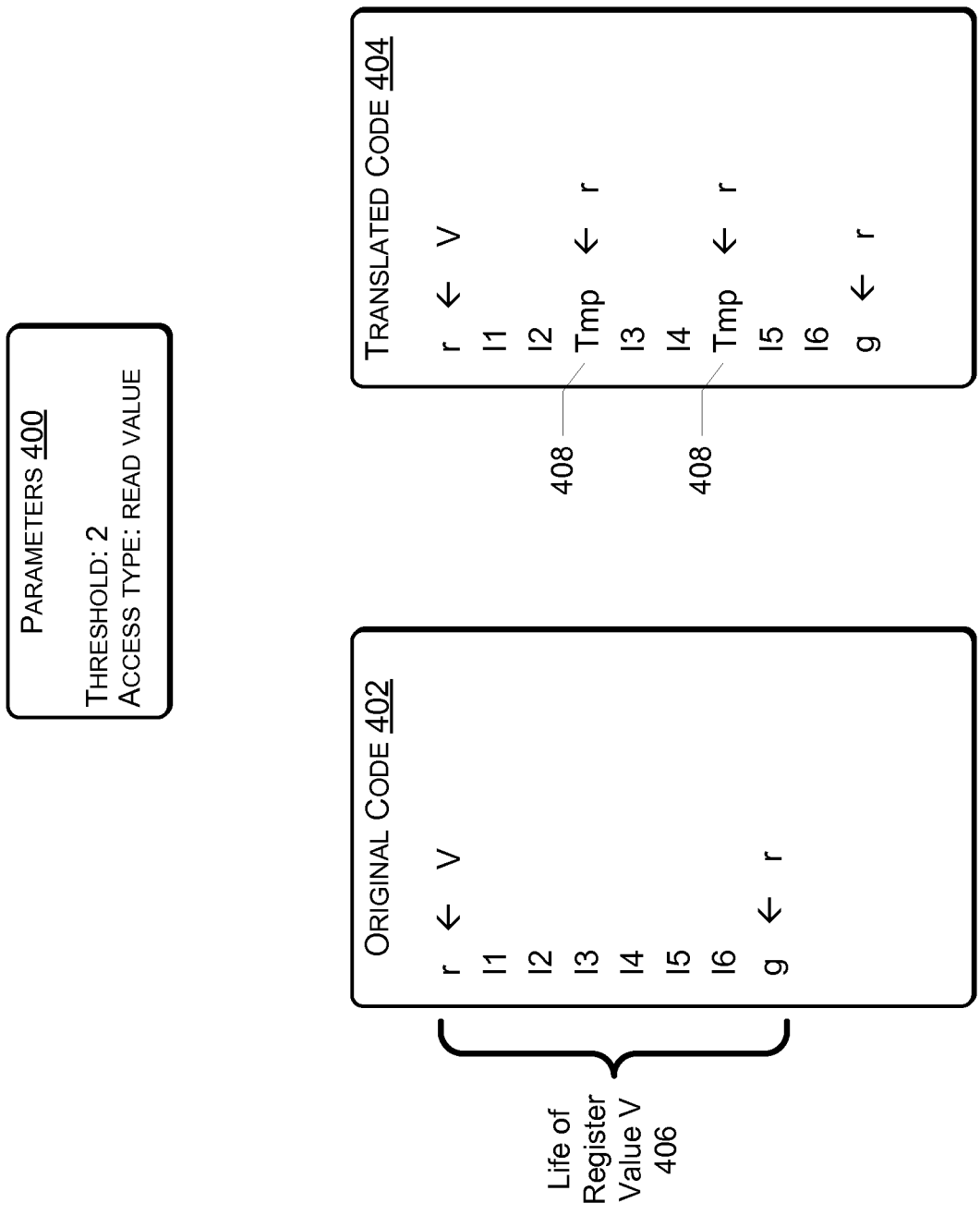
PARAMETERS 400
THRESHOLD: 2
ACCESS TYPE: READ VALUE
ORIGINAL CODE 402
r ← V
I1
I2
I3
I4
I5
I6
g ← r
Life of Register Value V 406
TRANSLATED CODE 404
r ← V
I1
I2
Tmp ← r
I3
I4
Tmp ← r
I5
I6
g ← r
408
408

FIG. 4

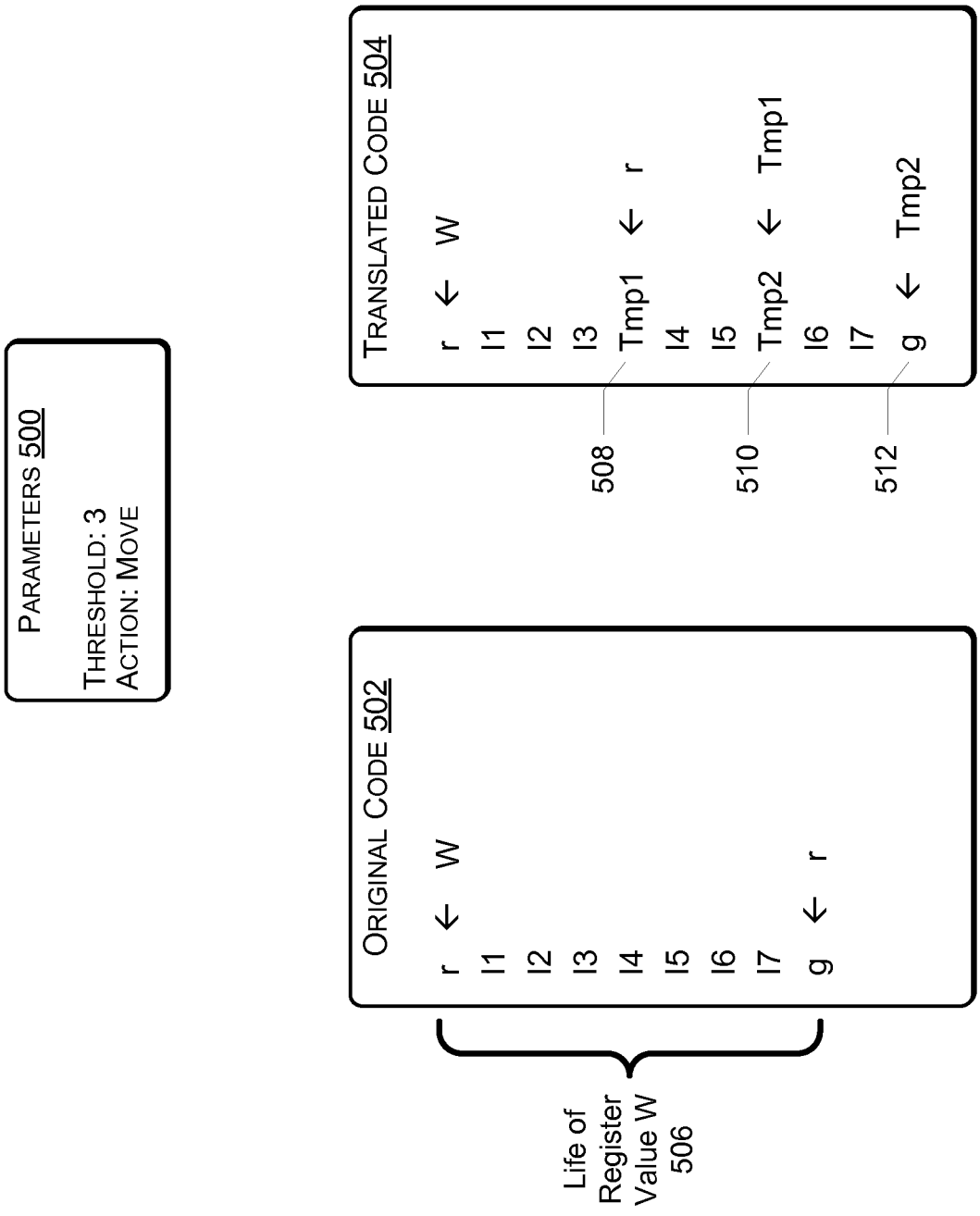
PARAMETERS 500
THRESHOLD: 3
ACTION: MOVE
ORIGINAL CODE 502
r ← W
I1
I2
I3
I4
I5
I6
I7
g ← r
Life of Register Value W 506
TRANSLATED CODE 504
r ← W
I1
I2
I3
Tmp1 ← r
I4
I5
Tmp2 ← Tmp1
I6
I7
g ← Tmp2
508
510
512

FIG. 5

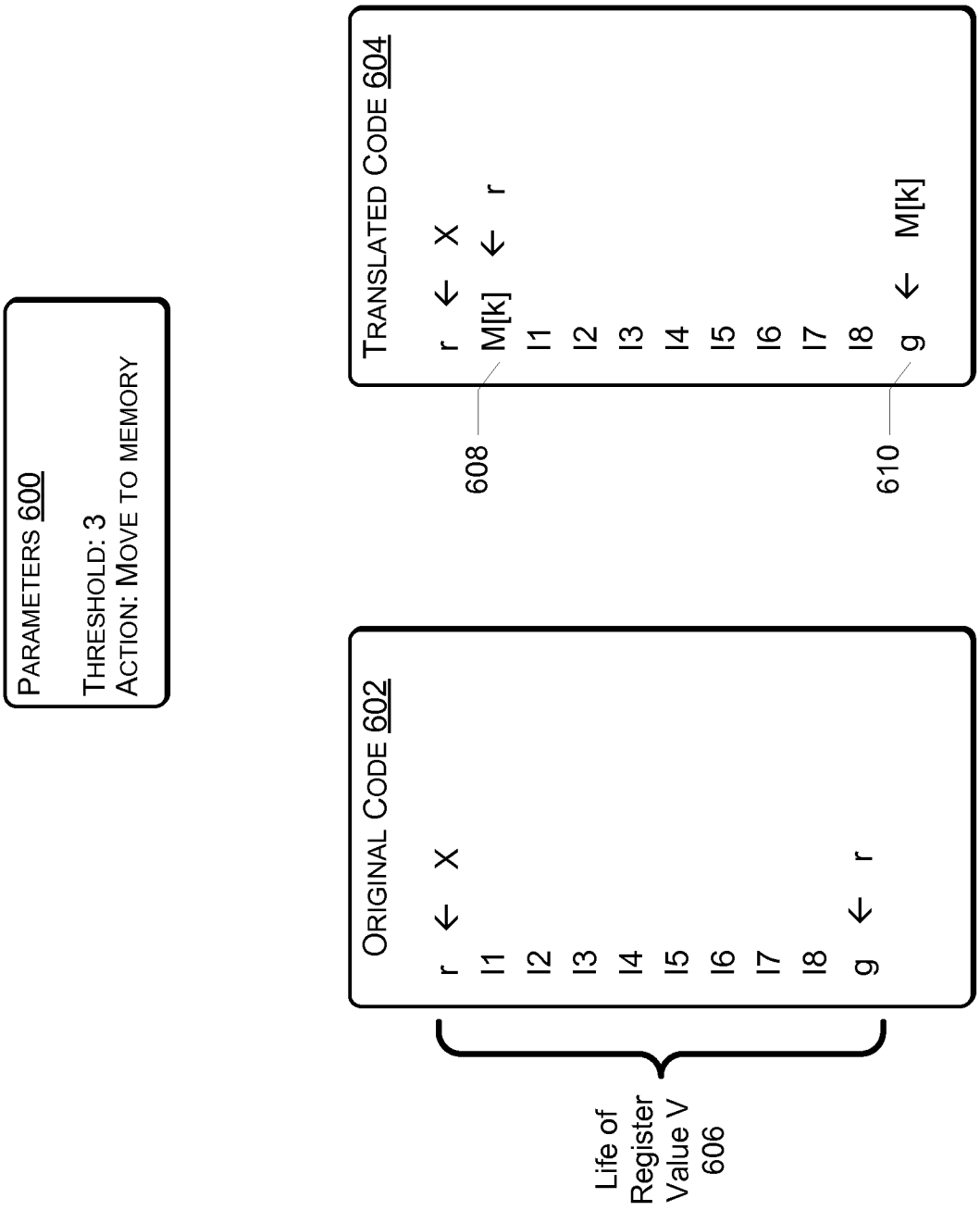
PARAMETERS 600
THRESHOLD: 3
ACTION: MOVE TO MEMORY
ORIGINAL CODE 602
r ← X
I1
I2
I3
I4
I5
I6
I7
I8
g ← r
Life of Register Value V 606
TRANSLATED CODE 604
r ← X
M[k] ← r
I1
I2
I3
I4
I5
I6
I7
I8
g ← M[k]
608
610

FIG. 6

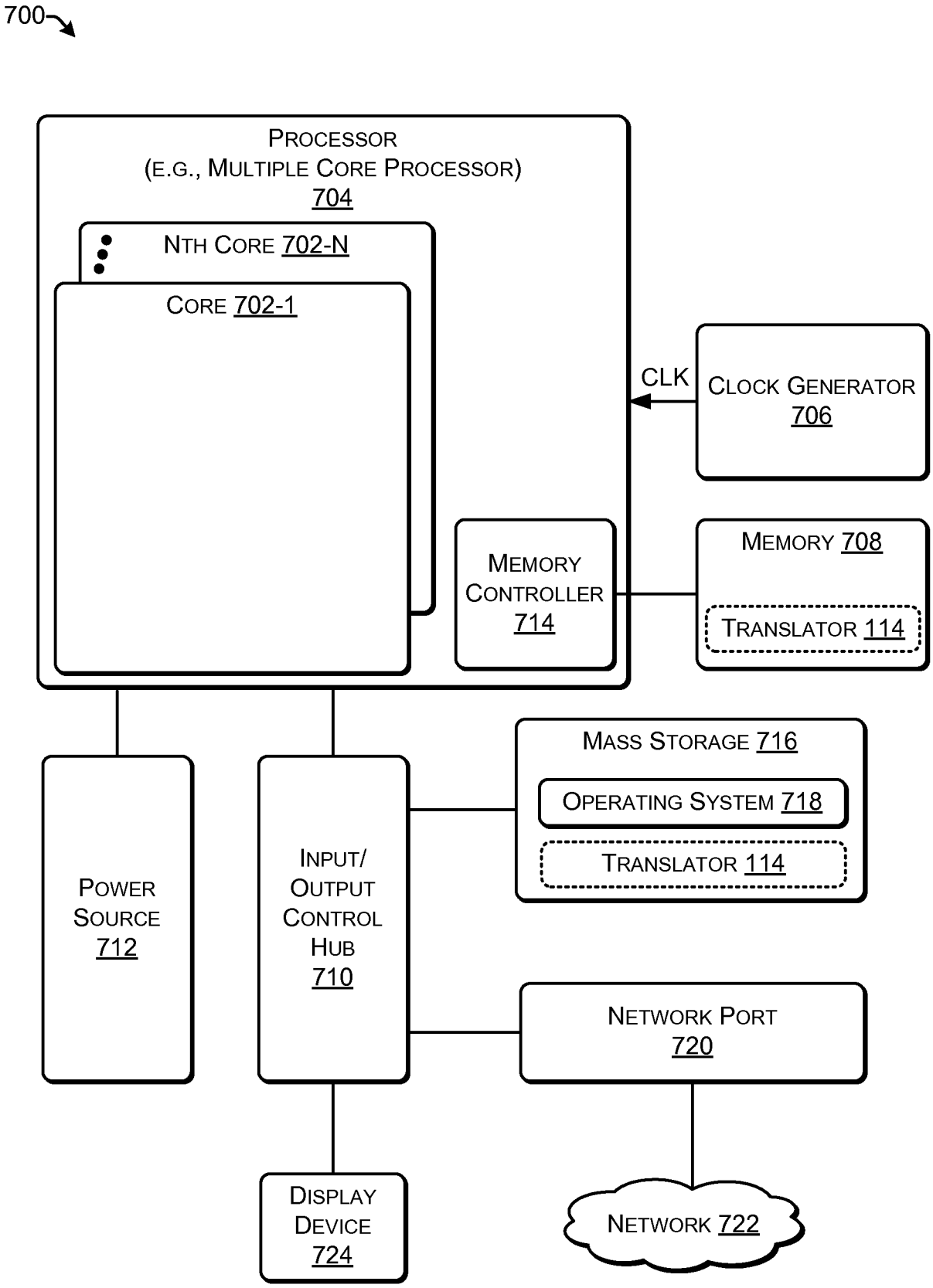
700
PROCESSOR
(E.G., MULTIPLE CORE PROCESSOR)
704
NTH CORE 702-N
CORE 702-1
CLK
CLOCK GENERATOR
706
MEMORY
CONTROLLER
714
MEMORY 708
TRANSLATOR 114
POWER
SOURCE
712
INPUT/
OUTPUT
CONTROL
HUB
710
MASS STORAGE 716
OPERATING SYSTEM 718
TRANSLATOR 114
NETWORK PORT
720
DISPLAY
DEVICE
724
NETWORK 722

FIG. 7

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2011/068051

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/30(2006.01)i, G06F 9/305(2006.01)1, G06F 11/14(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 11/07; H03M 13/00

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: silent data corruption, register file, detection, threshod period, access to long period register value, binary translator,

Google Scholar DB & Keywords: silent data corruption, register file, detection, threshod period, access to long period register value

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	M0NTESIN0S, P. et al. Using register lifetime predictions to protect registe r files against soft errors, 29 August 2008, URL http://iacoma.cs. uiuc.edu/i acoma-papers/ieeeshield.pdf (M0NTESIN0S, P. et al.)	13-15
A	See abstract, page 8 left column 4th paragraph, and figure 14.	1-12 ,16-22
A	US 2008-0155375 Al (XAVIER, V. et al.) 26 June 2008 See paragraphs [0018]-[0020] , claims 1,2,4, and figure 2.	1-22
A	LEE, J. et al. A compiler optimization to reduce soft errors in register fil es, IN: the 2009 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, pages 41-49, ACM, New York, USA, 19-20 June 2008 (LEE, J. et al.) See the pages 41,44,45, and figure 4	1-22
A	US 2011-0167296 Al (FOX, T.W. et al.) 07 July 2011 See paragraphs [0002] ,[0005] , claim 1, and figure 1.	1-22
L	Index of/iacoma-papers , filename=ieeeshield.pdf , URL http://iacoma.cs. uiuc.e du/ iacoma-paper s/ See page 3 1ine 36.	13-15

☐ 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	Date of mailing of the international search report
26 SEPTEMBER 2012 (26.09.2012)	**26 SEPTEMBER 2012 (26.09.2012)**

Name and mailing address of the ISA/KR	Authorized officer
Korean Intellectual Property Office 189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 302-701, Republic of Korea Facsimile No. 82-42-472-7140	Hwang, Seung Hee Telephone No. 82-42-481-5749

Form PCT/ISA/210 (second sheet) **(July** 2009)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US201 1/068051

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2008-0 155375 A1	26 .06 .2008	US 7689804 B2	30 .03 .2010
US 2011- 0 167296 A 1	07 .07 .2011	None	