

Systems and Methods for Policy Execution Processing

RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. Provisional Application No. 62/625,634, filed on February 2, 2018. The entire teachings of the above application are incorporated herein by reference.

BACKGROUND

[0002] The present disclosure relates generally to systems and methods for policy execution processing, for example, to enforce security policies. Computer processors, which include electronic circuits configured to execute one or more computer program instructions, are known. Furthermore, it is known to implement security policies to prevent a computer processor from compromising sensitive information.

SUMMARY

[0003] In one aspect, the disclosure is directed to a processing system comprising a host processing domain comprising a host processor. The host processor may be configured to receive at least one instruction comprising (i) operand information relating to one or more operands, and (ii) operation information indicative of an operation to be performed on the one or more operands. The host processor may be further configured to execute the operation indicated in the operation information on the one or more operands to generate operation output information, and to provide, to a metadata processing domain, instruction information and the operation output information. The metadata processing domain may comprise a write interlock, which may be configured to receive, from the host processing domain, the operation output information, and place the operation output information into a queue. The metadata processing domain may further comprise a tag processing unit configured to receive, from the host processing domain, the instruction information, and use the instruction information to obtain one or more input metadata tags associated with the at least one instruction. The tag processing unit may further determine, in accordance with one or more policies being enforced and in accordance with the one or more input metadata tags associated with the at least one instruction, whether the at least one instruction is allowed. The tag processing unit may, responsive to a determination that the instruction is allowed,

cause the queue of the write interlock to write to memory the operation output information in a manner that associates the operation output information with at least one output metadata tag.

[0004] The tag processing unit may comprise a rule cache configured to store one or more rule entries of at least one policy of the one or more policies enforced by the metadata processing domain.

[0005] The determination that the instruction is allowed may comprise determine that the rule cache stores a rule entry matching the one or more input metadata tags associated with the at least one instruction. The tag processing unit may be configured to use information stored in the rule entry to provide the at least one output metadata tag to be associated with the operation output information.

[0006] The metadata processing domain may comprise a policy execution processor. The determination that the instruction is allowed may comprise (i) determine that the rule cache does not store a rule entry matching the one or more input metadata tags associated with the at least one instruction, (ii) responsive to a determination that the rule cache does not store a rule entry matching the one or more input metadata tags associated with the at least one instruction, provide, to the policy execution processor, the one or more input metadata tags associated the at least one instruction, and (iii) receive, from the policy execution processor, the at least one output metadata tag to be associated with the operation output information.

[0007] The policy execution processor may be configured to receive, from the tag processing unit, the one or more input metadata tags associated with the at least one instruction, and execute policy code against the one or more input metadata tags associated with the at least one instruction to determine whether the at least one instruction is allowed. Responsive to a determination that the at least one instruction is allowed, the policy execution processor may be further configured to install, into the rule cache, a rule entry based on the one or more input metadata tags associated with the at least one instruction and the at least one output metadata tag.

[0008] The policy execution processor may be configured to execute a secure boot operation. The policy execution processor may comprise a boot ROM that stores one or more public keys, and along with code that can (i) read an image from an external memory device, authenticate and decrypt the image using the one or more public keys, and enable the host processor to continue its boot process upon successful authentication and decryption.

[0009] The boot operation may comprise at reset, the host processor remains held in a reset state. The policy execution processor may (i) start execution at its reset vector, (ii) boot the policy software into its own memory space, (iii) configure one or more memory fabric protection configuration registers to define memory regions that each initiator can access, to protect a region of memory to hold a policy data segment, (iv) initialize the policy data segment, (v) copy a boot-loader for the host processor from the external memory device into main memory; and (vii) release the host processor from the reset state.

[0010] The host processor may be further configured to provide, to the metadata processing domain, update information indicative of one or more updates to the host processor's state as a result of executing the at least one instruction. The metadata processing domain may be further configured to, responsive to a determination that the at least one instruction is allowed, use the update information to update a shadow register configured to store a shadow copy of the host processing domain as of a most-recently-allowed instruction. The at least one instruction may comprise a first instruction. the instruction information may comprise first instruction information, and the one or more input metadata tags comprise one or more first input metadata tags. The tag processing unit may be further configured to (i) receive, from the host processing domain, second instruction information relating to a second instruction executed by the host processor, (ii) use the second instruction information to obtain one or more second input metadata tags associated with the second instruction, (iii) determine, in accordance with the one or more policies being enforced and in accordance with the one or more second metadata tags associated with the second instruction, whether the second instruction is allowed, and (iv) responsive to a determination that the second instruction is not allowed, communicate one or more rollback signals to the host processing domain to restore a state of the host processing domain to the shadow copy of the host processing domain.

[0011] The one or more input metadata tags may be inaccessible to the host processor.

[0012] The instruction information may comprise at least one piece of information selected from a group consisting of (i) information indicative of an instruction type of the at least one instruction, (ii) information indicative of a memory address from which the at least one instruction was fetched (iii) information indicative of one or more registers used by the at least one instruction, and (iv) information indicative of a memory address referenced by the at least one instruction.

[0013] In another aspect, the disclosure is directed to a processing system comprising an application processing domain comprising an application processor configured to implement a first instruction set architecture, and receive instructions comprising operand information indicative of one or more operands, and operation information indicative of an operation to be performed on the one or more operands. The instructions may be formatted in accordance with the first instruction set architecture. The application processing domain may further provide, to a metadata processing domain, the instructions in an instruction stream. The metadata processing domain may comprise a tag processing unit that may comprise at least one decode table. The tag processing unit may be configured to extract the operand information and the operation information from the instructions.

[0014] The tag processing unit may comprise at least one look-up table configured to determine, based on the instructions, instruction care bit information indicative of one or more care bits, and operation group information indicative of an operation group.

[0015] The tag processing unit may be further configured to determine, based on the at least one decode table, that the instructions are formatted in accordance with the first instruction set architecture. The determination that the instructions are formatted in accordance with the first instruction set architecture may be further based on the instruction care bit information indicative of the one or more care bits and the operation group information indicative of an operation group. The at least one decode table may comprise a cascade of tables of at least a primary table, a secondary table, and a plurality of address lookup tables.

[0016] The tag processing unit may further comprise a buffered interface configured to store the instructions from the application processor when the tag processing unit is stalled, and to provide stored instructions for use by the tag processing unit when the instruction stream is stalled. The buffered interface may comprise a first-in-first-out (FIFO) component.

[0017] The tag processing unit may further comprise a tag map table (TMT) configured to convert one or both of a physical instruction address and a data memory page address, into one or both of (i) a corresponding associated tag addresses and (ii) directly into a tag. The TMT may be configured to accommodate an immediate value tag, specified for a particular size of a memory region. The particular size may be selected from a range of memory region sizes.

[0018] The metadata processing domain may further comprise a Metadata Tag Interlock Queue configured to (i) perform a first write to change a tag address to a predetermined

constant value that indicates a write transaction is underway, (ii) perform a second write to write new data, and (iii) perform a third write to write a new tag associated with the new data.

[0019] In another aspect, the disclosure is directed to a method of processing instructions, comprising, in a host processing domain, by an application processor, receiving each of the instructions comprising (i) operand information relating to one or more operands, and (ii) operation information indicative of an operation to be performed on the one or more operands. The method may further comprise executing the operation indicated in the operation information on the one or more operands to generate operation output information, and providing, to a metadata processing domain, instruction information and the operation output information. The method may further comprise, in the metadata processing domain, by a write interlock, receiving, from the host processing domain, the operation output information, and placing the operation output information in a queue. The method may further comprise, by a tag processing unit, (i) receiving, from the host processing domain, the instructions information, (ii) using the instruction information to obtain one or more input metadata tags associated with the at least one instruction, (iii) determining, in accordance with one or more policies being enforced and in accordance with the one or more input metadata tags associated with the at least one instruction whether the at least one instruction is allowed, and (iv) responsive to determining that the instruction is allowed, causing the queue of the write interlock to write to memory the operation output information in a manner that associates the operation output information with at least one output metadata tag.

[0020] The tag processing unit may comprise a rule cache configured to store one or more rule entries of at least one policy of the one or more policies enforced by the metadata processing domain. Determining that the instructions are allowed may comprise determining that the rule cache stores a rule entry matching the one or more input metadata tags associated with the at least one instruction.

[0021] The method may further comprise using, by the tag processing unit, information stored in the rule entry to provide the at least one output metadata tag to be associated with the operation output information.

[0022] The metadata processing domain may comprise a policy execution processor. Determining that the instructions are allowed may comprise (i) determining that the rule cache does not store a rule entry matching the one or more input metadata tags associated with the at least one instruction, (ii) responsive to a determination that the rule cache does not store a rule entry matching the one or more input metadata tags associated with the at least

one instruction, providing, to the policy execution processor, the one or more input metadata tags associated the at least one instruction, and (iii) receiving, from the policy execution processor, the at least one output metadata tag to be associated with the operation output information.

[0023] The method may further comprise, by the policy execution processor, (i) receiving, from the tag processing unit, the one or more input metadata tags associated with the at least one instruction, (ii) executing policy code against the one or more input metadata tags associated with the at least one instruction to determine whether the at least one instruction is allowed, and (iii) responsive to a determination that the at least one instruction is allowed, installing into the rule cache, a rule entry based on the one or more input metadata tags associated with the at least one instruction and the at least one output metadata tag.

[0024] The method may further comprise, by the host processor, providing to the metadata processing domain, update information indicative of one or more updates to the host processor's state as a result of executing the at least one instruction;, and by the metadata processing domain, responsive to a determination that the at least one instruction is allowed, using the update information to update a shadow register configured to store a shadow copy of the host processing domain as of a most-recently-allowed instruction.

[0025] The at least one instruction may comprise a first instruction, the instruction information comprises first instruction information, and the one or more input metadata tags comprise one or more first input metadata tags. The method may further comprise, by the tag processing unit, (i) receiving, from the host processing domain, second instruction information relating to a second instruction executed by the host processor, (ii) using the second instruction information to obtain one or more second input metadata tags associated with the second instruction, (iii) determining, in accordance with the one or more policies being enforced and in accordance with the one or more second metadata tags associated with the second instruction, whether the second instructions is allowed, and (iv) responsive to a determination that the second instruction is not allowed, communicating one or more rollback signals to the host processing domain to restore a state of the host processing domain to the shadow copy of the host processing domain.

BRIEF DESCRIPTION OF THE DRAWINGS

[0026] The foregoing will be apparent from the following more particular description of example embodiments, as illustrated in the accompanying drawings in which like reference

characters refer to the same parts throughout the different views. The drawings are not necessarily to scale, emphasis instead being placed upon illustrating embodiments.

[0027] FIG. 1 illustrates a block diagram of a secure processing system according to an embodiment.

[0028] FIG. 2 illustrates a block diagram of a secure processing system according to an embodiment.

[0029] FIG. 3A illustrates a process of processing an instruction according to an embodiment.

[0030] FIG. 3B illustrates a process of processing an instruction according to an embodiment.

[0031] FIG. 4 illustrates a block diagram of a processor interlock for policy enforcement according to an embodiment.

[0032] FIG. 5 illustrates a data flow diagram of tag processing according to an embodiment.

[0033] FIG. 6 illustrates a block diagram of an instruction set architecture decoding system according to an embodiment.

[0034] FIGs. 7A through 7E depict an example set of PIPE Registers as described herein.

[0035] FIG. 8 depicts an example set of care bits as described herein.

[0036] FIGs. 9A through 9D depict an example TPU communication interface as described herein.

[0037] FIG. 10 shows parameters of an example ISA-specific header file as described herein.

[0038] FIG. 11 depicts an example mapping between shadow memory, RF/CSR name, and RF/CSR address, as described herein.

[0039] FIGs. 12A, 12B, and 12C show example communication information used by the MTIQ, as described herein.

DETAILED DESCRIPTION

[0040] A description of example embodiments follows.

[0041] The teachings of all patents, published applications, and references cited herein are incorporated by reference in their entirety.

[0042] Aspects and examples are directed generally to computing systems with improved enforcement of policies (e.g., security policies). In one example, a host processor and a

policy execution processor are connected by a processor interlock for policy enforcement. The processor interlock for policy enforcement determines if instructions committed by the host processor are compliant with a list of policy rules. If the list of rules does not include a desired rule, the policy execution processor may securely determine and provide the desired rule. In some examples, the host processor may be configured according to any instruction set architecture.

[0043] Examples of the methods and systems discussed herein are not limited in application to the details of construction and the arrangement of components set forth in the following description or illustrated in the accompanying drawings. The methods and systems are capable of implementation in other embodiments and of being practiced or of being carried out in various ways. Examples of specific implementations are provided herein for illustrative purposes only and are not intended to be limiting. In particular, acts, components, elements, and features discussed in connection with any one or more examples are not intended to be excluded from a similar role in any other examples.

[0044] Also, the phraseology and terminology used herein is for the purpose of description and should not be regarded as limiting. Any references to examples, embodiments, components, elements or acts of the systems and methods herein referred to in the singular may also embrace embodiments including a plurality, and any references in plural to any embodiment, component, element or act herein may also embrace embodiments including only a singularity. References in the singular or plural form are not intended to limit the presently disclosed systems or methods, their components, acts, or elements. The use herein of “including,” “comprising,” “having,” “containing,” “involving,” and variations thereof is meant to encompass the items listed thereafter and equivalents thereof as well as additional items. References to “or” may be construed as inclusive so that any terms described using “or” may indicate any of a single, more than one, and all the described terms. In addition, in the event of inconsistent usages of terms between this document and documents incorporated herein by reference, the term usage in the incorporated references is supplementary to that of this document; for irreconcilable inconsistencies, the term usage in this document controls.

[0045] Computer processors are designed to receive and execute instructions. Instructions may specify one or more operations to be carried out on one or more specified operands. For example, a computer processor may receive an instruction to perform an addition operation on a first operand and a second operand, and to output the result of the

addition operation to a specific output location (e.g., in memory). The first operand and the second operand may be identified by respective locations (e.g., memory locations) at which the first operand and the second operand are stored. Upon retrieving the first operand and the second operand at the respective locations, the computer processor executes the addition operation and writes the result to the specified output location.

[0046] In some embodiments, instructions may be more fine-grained. As one example, a store instruction may store a value from a data register to a memory location referenced by an address from an address register. As another example, a load instruction may load a value from a memory location referenced by an address from an address register to a data register. As another example, an arithmetic instruction may operate on one or more input values stored in one or more input registers and place an output value into an output register.

[0047] In some embodiments, metadata tags may be associated with entities involved in an instruction (e.g., an instruction type, a program counter register, a memory address from which the instruction is fetched, one or more registers used by the instruction, a memory location referenced by the instruction, etc.). For example, the metadata tags may be used to determine which actions are acceptable/unacceptable, for example, to prevent security compromises. A computer processor executing the instructions may be beholden to one or more security policies which dictate how instructions with certain metadata tags are to be handled. For example, the security policies may disallow instructions with certain metadata tags because execution of the instructions may compromise system security.

[0048] Accordingly, metadata tags and security policies can prevent a computer processor from executing potentially-malicious instructions which may originate from a bad actor. However, if the computer processor executing the instructions is able to access and modify the metadata tags and/or the security policies, then a bad actor may circumvent the security policies by modifying the metadata tags, the security policies, or both, via the computer processor. The bad actor may therefore cause the computer processor to execute otherwise-disallowed instructions by manipulating the metadata tags and/or the security policies governing the computer processor.

[0049] To avoid the aforementioned problem, embodiments disclosed herein provide a processing system which includes two processors. A first processor executes instructions subject to metadata tags and security policies, but, in most circumstances, is not allowed to access the metadata tags or the security policies. A second processor accesses the metadata tags and the security policies, and determines compliance with the security policies, but does

not execute instructions received by the processing system. Separation of the tasks performed by the first processor and the second processor may provide significant advantages to computer processor security and efficiency. Furthermore, systems and methods disclosed herein may allow the first processor to be implemented according to one of several supported Instruction Set Architectures (ISAs), which provides a highly flexible system.

[0050] FIG. 1 illustrates a block diagram of a computer processing system 100 according to an embodiment. The computer processing system 100 includes a host processor, sometimes also referred to as an Application Processor (AP) 102, a Processor Interlock for Policy Enforcement (PIPE) 104, and a System-On-Chip (SOC) bus 106. The AP 102, the PIPE 104, and the SOC bus 106 are communicatively coupled to one another. In some embodiments, the SOC bus 106 is configured to be communicatively coupled to one or more peripherals, one or more memory banks, and a SOC bus configuration system.

[0051] The PIPE 104 includes a Tag Processing Unit (TPU) 108, a Policy EXecution Processor (PEX) 110, and a Metadata Tag Interlock Queue (MTIQ) 112. The TPU 108 is communicatively coupled to the PEX 110 and the MTIQ 112. The PEX 110 is communicatively coupled to the TPU 108. The MTIQ 112 is communicatively coupled to the TPU 108.

[0052] In some embodiments, the AP 102 is generally configured to receive instructions, execute operations specified by the instructions, and write an output to the MTIQ 112. The AP 102 may also provide the instructions to the TPU 108. As discussed in greater detail below, the AP 102 may be unable to access metadata tags or security policies governing the instructions, and therefore unable to validate the executed instructions. In some embodiments, because the AP 102 is unable to validate the instructions, the AP 102 may not include Control and Status Registers (CSRs) which would enable the AP 102 to independently commit non-validated instructions to components external to the AP 102 prior to validation. For instance, the AP 102 may be configured to exclude one or more CSRs, and/or one or more CSRs may be disabled by disconnecting respective ports. Instead, the output write from the AP 102 is stored in the MTIQ 112, where the write is held until the instruction has been validated by the PIPE 104.

[0053] In some embodiments, the PIPE 104 is generally configured to receive retired instructions from the AP 102, and analyze the instructions to determine if the instructions comply with one or more policies (e.g., security policies). The PIPE 104 is configured to access policies and metadata tags associated with retired instructions to determine if the

instructions comply with the policies. The SOC bus 106 is generally configured to route one or more signals between components connected to the SOC bus 106, including the AP 102 and the PIPE 104.

Tag Processing Unit (TPU)

[0054] In some embodiments, the TPU 108 is generally configured to perform one or more of the following functions. A first function is to act as a rule cache to process one or more input metadata tags associated with instructions provided by the AP 102. The TPU 108 writes output metadata tags to the MTIQ 112 upon successful validation of the instructions. A second function is to maintain a shadow copy of register files and CSRs that represent a shadow state of the computing system 100 as of a mostly-recently-allowed instruction. The computing system 100 may be “unwound” to the trusted shadow state if an issue (for example, an instruction which does not comply with the security policies) is detected.

[0055] In some embodiments, the PEX 110 is generally configured to perform one or more of the following functions. A first function is to execute a secure boot operation by booting a desired boot loader into the AP 102. A second function is to evaluate tags to determine and install rules into the TPU 108 when the TPU 108 fast cache does not include desired rule. A third function is to maintain a map of AP 102 memory addresses to metadata tag memory addresses.

[0056] In some embodiments, the MTIQ 112 is generally configured to receive data writes from the AP 102 and corresponding output metadata tags from the TPU 108, and output a combination of a data write and a corresponding output metadata tag. In some embodiments, the MTIQ 112 is configured to store output data received from the AP 102 until an output metadata tag is received from the TPU 108 indicating that the instruction complies with the security policies. Once the output metadata tag is received, the MTIQ 112 writes the output tag-and-data pair to a specified output address. Additionally, or alternatively, the output data may be written to an application memory address, whereas the output tag may be written to a metadata memory address corresponding to the application memory address.

[0057] As discussed in greater detail below, the MTIQ 112 enables a buffered interface to exist between the AP 102 and the PIPE 104. In some conventional systems, a host processor may be disallowed from executing subsequent instructions until a current instruction has been validated. In contrast, because the MTIQ 112 holds all outputs from the AP 102 until

validation is received from the TPU 108, the AP 102 may continue executing instructions and writing outputs to the MTIQ 112 before the instruction is validated by the TPU 108.

Similarly, the PIPE 104 may validate instructions independent of the rate at which the AP 102 executes instructions.

[0058] Accordingly, the AP 102 and the PIPE 104 may not be beholden to one another, and may therefore process instructions independently, yielding significant increases in processing speed. If the AP 102 and the PIPE 104 both successfully process an instruction, then the MTIQ 102 may output the validated result of the instruction to a specified destination. Otherwise, if an output from the AP 102 is disallowed by the PIPE 104, then the computing system 100 may be unwound to the shadow state of the computing system as of a most-recently-allowed instruction. Therefore, the AP 102 and the PIPE 104 may process instructions independently of one another without compromising processor security or efficiency.

[0059] FIG. 2 illustrates a detailed block diagram of a computing system 200 according to an embodiment. The computing system 200 may represent a more detailed block diagram of the computing system 100 according to one implementation. However, alternate implementations of the computing system 100 are intended to be within the scope of this disclosure, and the computing system 200 is provided for explanatory purposes only. Certain elements and connections of the computing system 200 are identified and explained below for explanatory purposes only. For example, certain elements and connections of the computing system 200 are identified for explanation in connection with FIGS. 3A and 3B, below.

[0060] The computing system 200 includes an AP 202, a PIPE 204, and a SOC bus 206. The PIPE 204 includes a TPU 208, a PEX core 210, and an MTIQ 212. The TPU 208 includes an input first-in-first-out component (referred to herein as a “FIFO”) 214, a rule cache 216, and a shadow register 217. The MTIQ 212 includes an operation output FIFO 218 and a tag FIFO 220. In some examples, the AP 202 may be part of a host processing domain, and the PIPE 204 may be part of a metadata processing domain. The processing domains may, in some embodiments, include respective memories. The FIFOs described herein may be implemented in hardware, software, or a combination of both, as is known in the art.

[0061] The AP 202 is configured to receive instructions 222 from the SOC bus 206, provide operation outputs 224 to the operation output FIFO 218, and provide instructions 226 to the input FIFO 214. The TPU 208 is configured to provide a tag output 228 to the tag

FIFO 220, provide an interrupt 230 to the PEX 210, and provide a reset vector 232 to the SOC bus 206 from the shadow register 217. In some embodiments, the TPU 208 may be configured to provide a hardware rollback signal 234 to the AP 202. The PEX 210 is configured to execute policy code according to one or more signals received from the TPU 208. The MTIQ 212 is configured to receive and hold the operation outputs 224 in a queue until a corresponding tag output 228 is received.

[0062] FIGS. 3A and 3B illustrate an example process 300 of handling an instruction according to an embodiment. FIG. 3A depicts the example process from action 302 through action 314, and FIG. 3B continues the example process from action 314 (“action” is also referred to herein as “act”) through action 326. The example process 300 may be executed in connection with the computing system 100 or the computing system 200. For explanatory purposes only, examples are presented in connection with the computing system 200.

[0063] At act 302, the process 300 begins, as shown in FIG. 3A. At act 304, the AP 202 receives the instructions 222. For example, the AP 202 may receive instructions 222 from components external to the computing system 200 via the SOC bus 206. At act 306, the AP 202 executes the instructions. Executing the instructions may include performing an operation specified by the instructions on operands specified by the instructions.

[0064] At act 308, the AP 202 writes an output 224 of the executed operation to the MTIQ 212. For example, the AP 202 may write the output of the executed instructions to the operation output FIFO 218 in the MTIQ 212, where the output is held in a queue until a corresponding output metadata tag is received from the TPU 208 via the tag FIFO 220, indicating that the executed instructions are consistent with one or more policies (e.g., security policies).

[0065] At act 310, the AP 202 provides the instructions 226 to the TPU 208. For example, the AP 202 may provide the instructions 226 to the input FIFO 214. At act 312, the TPU 208 analyzes a received instruction, and determines if the rule cache 216 includes an entry matching metadata tags associated with the received instruction. For example, the rule cache 216 may include a rule cache configured to store frequently-accessed rule entries. If a matching rule entry is available from the rule cache 216 (312 YES), then the process 300 continues to FIG. 3B at act 314.

[0066] Otherwise, if a matching rule entry is not available from the rule cache 216 (312 NO), then the process 300 continues to act 316. At act 316, a request may be sent to the PEX 210 to validate the received instruction. This may include providing the interrupt 230 to the

PEX 210, along with the metadata tags associated with the received instruction. The PEX 210 may evaluate the metadata tags to determine whether the instruction should be allowed. Responsive to determining that the instructions should be allowed, the PEX 210 installs a new entry in the rule cache 216 with the metadata tags associated with the received instruction and/or an output metadata tag provided by the PEX 210. The process 300 continues to FIG. 3B at act 314.

[0067] FIG. 3B begins at act 314 and continues to act 318. At act 318, a determination is made by the TPU 208 as to whether a rule indicated in the matching rule catch entry has been satisfied. For example, act 318 may include applying the rule to one or more metadata tags associated with the received instruction to determine if the one or more metadata tags are in compliance with the rule. If the rule is satisfied (318 YES), then the process 300 continues to act 320. However, it should be appreciated that aspects of the present disclosure are not limited to performing a separate determination of compliance. In some embodiments, a presence of a matching rule entry in the rule cache may indicate the received instruction is allowed. Likewise, an output of the PEX 210 may directly indicate whether the received instruction is allowed. Thus, there may be no act 318 for applying a rule.

[0068] At act 320, the TPU 208 generates a shadow copy of the state of the AP 202 and stores the shadow copy in the shadow register 217. For example, the TPU 208 may write write-back information received in connection with the instruction to the shadow register 217.

[0069] The shadow copy of the AP 202 represents a state of the AP 202 as of execution of a most-recently-allowed instruction. In some embodiments, the shadow copy generated at act 320 enables the AP 202 to continue executing instructions and writing operation outputs 224 to the operation output FIFO 218 before the instructions are determined by the TPU 208 to be compliant with one or more policies. If an instruction is subsequently determined to be non-compliant with any policy, the state of the AP 202 may be unwound to the most-recently-allowed state stored in the shadow register 217 such that no non-validated data is written to components external to the computing system 200.

[0070] At act 322, the output metadata tag 228 associated with the allowed instruction is written to the MTIQ 212 by the TPU 208. For example, the TPU 208 may write the output metadata tag to the tag FIFO 220 in the MTIQ 212. The tag FIFO 220 may be coordinated with the operation output FIFO 218 such that the operation output 224 written to the

operation output FIFO 218 and the metadata tag 228 written to the tag FIFO 220 respectively occupy the same position in the operation output FIFO 218 and the tag FIFO 220.

[0071] At act 324, the operation output 224 stored in the operation output FIFO 218 and the metadata tag 228 stored in the tag FIFO 220 are released. For example, operation output FIFO 218 and the tag FIFO 220 may each release a least-recently-received input representing the operation output 224 and the corresponding output metadata tag 228. The pair may be released to the SOC bus 206 to be routed to an address specified in the instruction.

Additionally, or alternatively, the operation output 224 may be written to an application memory address, whereas the corresponding metadata output tag 228 may be written to a metadata memory address corresponding to the application memory address.

[0072] At act 326, the process 300 ends. Returning to act 318, if the rule is not satisfied (318 NO), then the process 300 continues to act 328. At act 328, a state of the AP 202 is unwound to a state stored in the shadow register 217, which reflects a state of the AP 202 as of a most-recently-allowed instruction. The process 300 ends at act 326.

[0073] Operation of the PIPE 204 will now be described in greater detail with respect to FIG. 4. FIG. 4 illustrates a block diagram of the PIPE 204, in accordance with some embodiments. The PIPE 204 includes the TPU 208, the PEX 210, and the MTIQ 212. The TPU 208 includes the input FIFO 214, the rule cache 216, and the shadow register 217. The MTIQ 212 includes the operation output FIFO 218 and the tag FIFO 220. The TPU 208 further includes a Tag Map Table (TMT) 236, an operation group/care Look-Up Table (LUT) 238, an ISA decoder 240, and Special Function Registers (SFRs) 242. However, it should be appreciated that aspects of the present disclosure are not limited to the particular combination and arrangement of elements in the example of FIG. 4. For instance, aspects of the present disclosure are not limited to using a shadow register, a tag map table, an operation group/care LUT, etc.

[0074] As discussed above, the PIPE 204 may, in some embodiments, be configured to receive input instructions from the AP 202 for each instruction that the AP 202 finishes executing. The PIPE 204 fetches tags associated with the input instructions, and checks the fetched tags against a set of rules stored in the rule cache 216. Because the PIPE 204 is decoupled from the AP 202, the PIPE 204 is able to verify tag sets after the AP 202 has retired an instruction. Due to the post-processing nature of the PIPE 204, it may be advantageous for the PIPE 204 to withhold performing any external effects of the AP 202 instructions until each instruction has been verified by the TPU 208.

[0075] Furthermore, as discussed above, a buffered interface may, in some embodiments, be provided between the AP 202 and the PIPE 204. The buffered interface enables outputs of committed instructions to be queued up from the AP 202 into the operation output FIFO 218, after which point the TPU 208 evaluates a committed instruction to determine if the instruction should be allowed, and what a result tag might be.

[0076] Because the AP 202 is allowed to process and commit instructions before the PIPE 204 verifies the instructions, the AP 202 may commit instructions which are subsequently disallowed by the PIPE 204. For situations in which a policy violation is detected, it may be advantageous to provide a mechanism to unwind the state of the AP 202 to that of a most-recently-allowed instruction once the TPU 208 detects the policy violation. To enable this, the TPU 208 may receive write-back information for each received instruction. The TPU 208 may use the write-back information to maintain a shadow copy of the register file, the floating-point register file, and/or Control/Status Registers (CSRs) in a memory, such as the shadow register 217. The shadow copy represents a trusted “back-up” state of the AP 202 which may be loaded if a policy violation is detected.

[0077] If an unwind event is required, the PEX 210 may use a PIPE register to initiate a reset or interrupt of the AP 202. If a reset is initiated, the PEX 210 may also provide the reset vector 232 pointing to code that will restore the AP 202 state from PIPE shadow state RAM, such as the shadow register 217. In some embodiments, the PIPE 204 may further provide the hardware rollback signal 234 to the AP 202 to roll back the hardware of the AP 202 to a previous trusted state. It should be understood, however, that although the example embodiments describe a example hardware-based roll-back implementation, a software-based roll-back instead of or in addition to the hardware-based implementation may also be used.

[0078] With the unwind mechanism and the MTIQ 212 in place, the AP 202 may commit instructions into register files and CSRs before the instructions are checked, because no external effects are produced and the state may be unwound if needed. Accordingly, in some embodiments, it may be advantageous to not provide the AP 202 with CSRs that may cause effects external to the AP 202, such as General Purpose Input/Output (GPIO) via CSRs. For example, the AP 202 may be configured to exclude the non-standard Berkley extension “mtohost” and “mfromhost” CSRs, and/or one or more ports to such CSRs may be disconnected.

[0079] As discussed above, the PIPE 204 includes SFRs 242. The SFRs 242 are summarized in FIGs. 7A-7E, in accordance with some embodiments. The “Address” column

refers to the lower address bits. The “Bits” column indicates which bit(s) that the field occupies. When a register has more than one field in it, the row with the address will have an empty “Bits” cell, and the proceeding rows will indicate the fields for each corresponding bit location with an empty address cell. In one embodiment, all registers are 32 bits wide, and can be accessed as bytes, half-words, or words. For registers with less than 32 bits indicated, the remaining bits are reserved and may be ignored by software. However, aspects of the present disclosure are not limited to registers of any particular size, or accessing registers at any particular granularity.

[0080] In this example, there are several types of registers in the PIPE 204. Status registers include read-only registers with status and statistics. Control registers include write-only and read-only registers that initiate action from the AP 202 or the PIPE 204. The miss status registers include read-only registers that are populated with values that cause a rule cache miss. For example, the miss status registers include registers to indicate an evaluation of a rule cache ways hash function, and include bits to inform software if a valid rule is present at an indicated location. The software may therefore quickly determine if an ejection of a rule from the rule cache 216 is necessary.

[0081] In some embodiments, write command registers include a set of read/write registers used to install a new rule in the rule cache 216. For example, the new rule may be a desired rule which is not already present in the rule cache 216. The software may specify which rule cache way the rule is being installed into, such that the miss software is responsible for cache management. Key-based query registers include a set of registers for checking if a particular input set, or key, is present in any of the rule cache ways. For example, this may be advantageous where a portion of the rule cache is being used as a per-policy rule cache.

[0082] In some embodiments, address-based inspect registers include a set of registers for looking into a specific rule cache way’s address and fetching the content therein. For example, this may be particularly advantageous for saving a rule before a new insertion such that the software may either re-install the rule to a different cache way, or provide the rule to a software L2 cache. Tag map table control registers include registers for writing the TMT 236, as discussed in greater detail below. The TMT 236 may, in some embodiments, be fully-associative. The operation group table control registers include registers for writing the Operation Group and Cares table, as discussed in greater detail below. The tag RAM access registers include read/write registers that memory map the tag registers maintained by the

PIPE 204. Shadow RAM access registers include read/write registers that provide memory-mapped access to the PIPE's 204 shadow set of the register file and CSR RAMs.

[0083] The tables shown in FIGs. 7A through 7E depict an example set of PIPE Registers as described herein.

[0084] As described herein, the TPU 208 may, in some embodiments, be configured to receive inputs from the AP 202 for each instruction that the AP 202 finishes. The inputs are queued in the input FIFO 214, which holds instructions to be processed by the TPU 208. For example, processing by the TPU 208 may include analyzing an instruction's associated tag set to determine if the policies being enforced by the TPU 208 and the PIPE 204 allow the associated tag set. The FIFO 214 facilitates a buffered interface at the input of the TPU 208. This buffered interface allows the TPU 208 to compensate for events where the TPU 208 must be stalled (e.g., to account for data hazard events) and to compensate for events where the instruction stream 226 is stalled (e.g., due to branch events within the AP).

[0085] FIG. 5 illustrates a data flow diagram 500 of the TPU 208 according to an embodiment. The data flow diagram 500 is nominally divided into seven stages for purposes of explanation only. At a first stage 502, instruction types, execution results, instruction and/or data addresses, and/or write-back values flow into the input FIFO 214. For example, the instruction types, execution results, instruction and/or addresses, and/or write-back values may be received from the AP 202.

[0086] At a second stage 504, the input FIFO 214 is popped, and four events may occur. In a first event, which is described in greater detail below with respect to FIG. 6, one or more bits identifying an instruction type (e.g., opcode) are sent to the operation group/care LUT 238, which translates the instruction operation code into a group ("OPGRP") and a set of input masking bits ("CARES"). For example, the operation group/care LUT 238 may group addition and multiplication operation codes into an arithmetic operation group. Grouping similar operation codes may simplify processing in subsequent stages, because similar or identical rules may apply to a common arithmetic group. The set of masking bits, or CARES, indicate to subsequent logic that a particular instruction, data, or operand tag should be partially or completely masked before evaluation against one or more policies. For example, a jump opcode may have the CARES bits set in such a way that only one operand tag, corresponding to an operand register that holds a destination address of the jump, is used for policy evaluation for the given instruction. Tags for the remaining potential operands are to be masked or otherwise excluded from the policy evaluation.

[0087] In a second event, which may be executed in parallel with the first event, the physical instruction address and the physical data address are translated by the TMT 236, as discussed in greater detail below with respect to FIG. 6. For example, the TMT 236 may maintain a map of physical instruction and data memory addresses to tag addresses and/or values. The TMT 236 may output either a fixed tag value (e.g., for one or more instructions or words, or an entire page of instructions or data words), or the physical address where such a tag value is located.

[0088] In a third event, instructions are provided to a pipeline register 244. The pipeline register 244 is configured to ensure that the instructions are output to a subsequent stage at an appropriate time relative to the other events in the second stage 504.

[0089] In a fourth event, write-back values, types, and addresses are passed to a write-back pipeline register 246. As discussed above, the write-back information may be indicative of a state of the AP 202. The write-back information is stored in the shadow register 217 at a final stage of the data flow diagram 500 to generate a shadow copy of the AP 202 and, if an unwind operation is necessary, the shadow copy is read from the shadow register 217 to unwind the state of the AP 202.

[0090] The write-back pipeline register 246 may ensure that the write-back information is passed along through each stage of the data flow diagram 500 as other operations are being performed on the information received at the first stage 502. Subsequent stages may also include respective write-back pipeline registers that ensure that write-back information is stored in the shadow register 217, which may be at a final stage of the data flow diagram 500, at an appropriate time (for example, when it has been determined that the instructions received in the input FIFO 214 comply with installed policies). For purposes of brevity, subsequent write-back pipeline registers 246 are not specifically discussed in detail, as they function similarly to the pipeline register 246 of the second stage 504. Alternatively, or additionally, the write-back information may be synchronized with the TPU processing pipeline by other mechanisms, such as with a first-in, first-out (FIFO) buffer.

[0091] At a third stage 506, a further four events may occur. A first event occurs when the TMT 236 returns a tag value and/or a tag address for the instruction, and a corresponding care bit for a current instruction is returned from the operation group/care LUT 238. If the TMT 236 returns a tag address, a read of an instruction metadata tag value may be thereafter initiated, and the value returned written to the instruction tag FIFO 250 (ITAG FIFO). If the

TMT 236 returns a tag value, the tag value may be written directly to the ITAG FIFO as the instruction metadata tag value.

[0092] A second event occurs when the TMT 236 returns tag value and/or tag address for memory data, and a corresponding care bit for a current memory data is returned from the operation group/care LUT 238. If the TMT 236 returns a tag address, a read of the data metadata tag may be thereafter initiated, and the value returned written to the data tag FIFO 250 (DTAG FIFO). If the TMT 236 returns a tag value, the tag value provided by the TMT 236 may be written directly to the DTAG FIFO as the data metadata tag value.

[0093] In a third event, the ISA decoder 240 receives care bit information and operation group information from the operation group/care LUT 238, and receives the instructions from the pipeline register 244. A simplified ISA decode operation is executed on the instructions. The simplified ISA decode operation, which is discussed in greater detail below with respect to FIG. 6, includes the determination of which operands and destinations the instructions use, the operand and destination types and addresses, and whether memory is used, or written to, by the instructions.

[0094] In some embodiments, in the ISA decoder 240, the operand and destination addresses provided by the simplified ISA decode operation in the second stage 504 may be condensed. The operand and destination addresses may be passed through an ISA-specific address map block to map the addresses into a condensed single address space. Condensing the address space may be advantageous where addresses of the AP 202 register files and CSRs are sparsely populated and/or aliased, such that the address space may be reduced to a contiguous address space. The mapping may be used both for addressing the operand tag RAM and the shadow state RAM. FIG. 11 illustrates an example address mapping for a RISC-V AP.

[0095] In a fourth event, write-back information is pipelined by a set of pipeline registers.

[0096] At a fourth stage 508, two events may occur. In a first event, a first set of transparent FIFOs 250 receives metadata tags for instructions and/or data. The first set of transparent FIFOs 250 may be coupled to one or more logic gates 252 configured to receive tag mapping information from one of the TMT 236 and the tag lookup blocks 248, and provide the tag mapping information to the first set of transparent FIFOs 250. For example, the one or more logic gates 252 may include logical OR gates coupled to the TMT 236 and the tag lookup tables 250, and configured to output information provided by an active one of the TMT 236 and the tag lookup tables 250. The first set of transparent FIFOs 250 outputs the

tag mapping information to a subsequent stage once the first set of transparent FIFOs 250 are written.

[0097] In a second event, the output of the ISA Decode 240, which includes the operand group and care bit information from the operation group/care LUT 238, is stored in a second transparent FIFO 254, which outputs the decode information to a subsequent stage once written.

[0098] At a fifth stage 510, the head of a second transparent FIFO 254 (S4 FIFO) is analyzed to determine if an instruction tag and/or metadata tag are necessary for a current instruction by analyzing corresponding care bits. The second transparent FIFO 254 is configured to provide care masking information for a current instruction tag and a metadata tag, and is configured to provide addresses for fetching Register Source (RS) tags (e.g., RS1/2/3) from a register file (RF) tags Memory(ies) corresponding to the care bits. Once all necessary inputs are available, the first set of transparent FIFOs 250 and the second transparent FIFO 254 are popped. Previous destination addresses are tracked, and a stall may be implemented to account for a data hazard (i.e., a data dependency event).

[0099] Upon all the availability of the instruction tag and/or data tag indicated as required by the S4 FIFO 254 output, care bit masking may be applied to the instruction tag and/or data tag, and the resultant tag or tags stored in the instruction tag register (ITAG REG) and data tag register (DTAG REG), respectively. Additionally, or alternatively, based on the care bits from the S4 FIFO, operand tags may be looked up by the addresses provided by the S4 FIFO 254 to the Operand TAG register file or RAM. A Program Counter Tag (PCTAG REG) register may provide an output of a program counter tag value to the subsequent state. Further, the content of the S4 FIFO 254 may be registered by a pipeline register to synchronize with the outputs of the instruction tag register.

[00100] In a sixth stage 512, three events may occur. In a first event, care masking may be applied on the RS1/2/3, memory, PC, and CI tags, and any other instruction bits not masked by the ISA decode logic. The tags, operation group, LSBs, and instruction bits may be concatenated to form a rule key (MVEC). The rule key may be hashed (e.g., by orthogonal randomizing hash functions) to produce addresses for the Rule Cache Way RAMs. In a second event, the rule may be used to read the Rule Cache 216.

[00101] At the seventh stage 514, four events may occur. In a first event, the output of each Rule Cache Way is compared against the rule key, MVEC. If no match is found, an interrupt signal, IRQ, is asserted to the PEX 210 to provide a desired rule. For example, the

PEX 210 may either act as a L2 rule cache, or run a full miss handler sequence. The TPU 208 awaits an SFR.WR_RESUME signal to resume operation.

[00102] In a second event, the specified result PC tag is compared against the current PC tag. If a change is being made, a signal is sent to previous stages to invalidate the current instruction, and the fourth stage 508 unwinds the FIFO pointers to the instruction after the specified point. In a third event, any specified destination tags are written to the register file tag memory and/or the MTIQ 212. For example, destination tags may be written to the tag FIFO 220. In a fourth event, the instruction's write-back values are written from a pipeline register (for example, a pipeline register similar to the pipeline register 246) to the shadow register 217 based on the previously-translated addresses. The data flow diagram 500 terminates after completion of the seventh stage 514.

[00103] The simplified ISA decode operation, which may be executed in connection with the TMT 236, the operation output/care LUT 238, will now be described in greater detail. FIG. 6 illustrates a block diagram 600 of configurable instruction decode tables according to an embodiment. The block diagram 600 includes a primary table 602, secondary tables 604, primary logic 606, secondary logic 608, a multiplexer 610, a de-multiplexer 612, and a plurality of address lookup tables 614.

[00104] The primary logic 606 includes a first input configured to receive an instruction, a second input to receive an instruction mask, and an output configured to provide an operation code to the primary table 602. The secondary logic 608 includes, for each secondary table of the plurality of secondary tables 604, a first input configured to receive the instruction, a second input to receive a sub-instruction mask, and an output configured to provide a sub-instruction to a respective secondary table of the plurality of secondary tables 604.

[00105] The primary table 602 includes an input to receive the operation code from the primary logic 606, a first output to provide an input selection signal to the multiplexer 610, and a second output to provide extracted instruction information to the multiplexer 610. The secondary tables 604 each include an input to receive sub-instructions from respective secondary logic of the secondary logic 608, and an output to provide extracted sub-instruction information to the multiplexer 610.

[00106] The multiplexer 610 includes a plurality of inputs coupled to the primary table 602 and the secondary tables 604, an input selection connection coupled to the primary table 602, and an output coupled to the demultiplexer 612. The demultiplexer 612 has an input coupled to the multiplexer 610, and a plurality of outputs coupled to the plurality of address

lookup tables 614 and an output 616. The plurality of address lookup tables 614 each include an input coupled to the demultiplexer 612 and an output coupled to the output 616.

[00107] The primary table 602 and the secondary tables 604 may be configured according to a specific ISA, and may facilitate extracting desired information from a received instruction that has been executed by the AP 202. As discussed above, the AP 202 may execute instructions according to one of several supported ISAs. As used herein, “supported ISAs” may include ISAs for which corresponding tables exist (for example, the primary table 602 or the secondary tables 604) to facilitate extracting desired ISA-specific information from received instructions.

[00108] The primary table 602 and the secondary tables 604 (collectively, the “configurable tables”) may be configured in accordance with a particular ISA to facilitate interpreting a particular instruction received by the PIPE 204. In this manner, the PIPE 204 may be used in connection with performing metadata processing for any suitable and desired ISA of the AP 202 that may be used in such an embodiment. Stated another way, use of such configurable tables provides support for metadata processing and policy enforcement for any instruction set for an ISA corresponding to the AP 202. The configurable tables may be configured in accordance with the particular ISA to facilitate interpreting a particular instruction received by the PIPE 204 having a particular instruction format for the ISA of the AP 202.

[00109] For example, the configurable tables may be used to extract from the received instruction particular bits which form an operation code (“op code”). Because the particular bits which form the operation code typically vary by ISA, the configurable tables may be configured according to a particular ISA and used in connection with extracting the particular bits from the instruction which form the operation code.

[00110] In a similar manner, the configurable tables may be used to examine and extract appropriate bits of the instruction for different operands, as well as any other information to be extracted from the instruction. For example, such configurable tables may be characterized as making the PIPE 204 customizable for use with any ISA of the AP 202 by customizing the tables, along with values for particular PIPE input parameters, in accordance with the ISA and its particular instruction format or layout.

[00111] The PIPE 204 may receive, as input, instruction size information (“INSTR WIDTH”), a number of sub-operand masks (“NLTM SUB MASKS”), a number of location masks (“NUM LOC MASKS”), a maximum number of operands per instruction

("NUM_OPS"), a maximum number of write-back destinations per instruction ("NUM_DESTS"), masks of bits that create the primary operation code ("INSTR_MASK"), an array of sub-operand masks ("SUB_MASK"), and an array of operand and destination location masks ("LOC_MASK").

[00112] To construct an operation code, the primary logic 606 may be implemented as a logical AND gate. The primary logic 606 receives, as inputs, instructions, and an instruction mask of bits that create the primary operation code. The primary logic 606 removes any bits masked by the instruction mask, concatenates the remaining bits to form the operation code, and outputs the operation code. The primary table 602, upon receiving the operation code, maps the operation code and outputs an operand validity check ("op_valid"), an operand type ("op_type"), an operand address location ("op_addr_loc"), a destination validity check ("dest_valid"), a destination type ("dest_type"), a destination address location ("dest_addr_loc"), an instruction mask ("instr_mask"), a memory validity check ("mem_valid"), a memory write ("mem_wr"), and a defer signal ("defer") that provides to an input selection terminal of the multiplexer 610.

[00113] In parallel, the secondary logic 608 may optionally be used to create sub-instructions to address the secondary tables 604. The secondary tables 604, similar to the primary table 602, receive input information, map the input information to output information, and provide the output information to the multiplexer 610.

[00114] The multiplexer 610 is configured to output information corresponding to at least one of the primary table 602 and the secondary tables 604 to the multiplexer 610. In one embodiment, the table selected is determined by an input selection signal received by the multiplexer 610. For example, the defer signal output by the primary table 602 may be utilized to select an input corresponding to one of the tables 602, 604 to provide to the output of the multiplexer 610. The multiplexer 610 outputs selected information to a de-multiplexer 612, which outputs each of the outputs discussed above to either the plurality of address lookup tables 614 or the output 616.

[00115] In some embodiments, the multiplexer 610 is configured to provide the operand address location and destination address location to the plurality of address lookup tables 614. The operand address location and destination address location indicate which of several possible instruction locations contain a necessary operand or destination address. The operand address location and destination address location are used to select a particular location mask result as a corresponding operand or destination address.

[00116] For example, each address lookup table of the plurality of address lookup tables 614 may include a multiplexer having a plurality of inputs and an input selection terminal. Each input of the multiplexer may be coupled to a different operand address. The input selection terminal may be configured to receive, for example, the operand address location signal, which is used to select a desired operand address. The multiplexer outputs the desired operand address to the output 616. Similar principles may apply, for example, to the destination address location information provided to the plurality of address lookup tables 614.

[00117] As discussed above, the secondary tables 604 may be omitted in some embodiments. For example, the secondary tables 604 may be omitted where the AP 202 is configured according to an ISA that does not include sub-instructions. Accordingly, where there are no sub-instructions, the secondary tables 604 may be unnecessary. It is to be appreciated that the number and content of the configurable tables may be customized according to a particular ISA of the AP 202.

[00118] The TMT 236 will now be described in greater detail (see, e.g., FIG. 7C and FIG. 7D). As discussed above, the TMT 236 converts physical instruction and data memory addresses into corresponding tag addresses, or directly into the tag. In some embodiments, the TMT 236 may be a fully-associative array with a parameterized number of entries. Each entry may contain a mask (MapMask) which is applied to an input address, an address which the masked input may be compared against (MapAddr), a bit to indicate if the entry is valid (MapValid), a bit to indicate if the value field is a tag or an address (MapImm), and an output value (MapValue).

[00119] The TMT 236 may be configured to receive several inputs. For example, the TMT 236 may receive an instruction physical address (“InstrPA”), a data physical address (“DataPA”), and several SFR values (“SFR_TMT_Entry,” “SFR_TMT_Type,” “SFR_TMT_Addr,” “SFR_TMT_Mask,” “SFR_TMT_Value,” and “SFR_TMT_Write”). The SFR value SFR_TMT_Type may include valid bits for an instruction tag lookup, a data tag lookup, a flag to indicate when TagMapValue is an immediate tag value instead of an offset, a shift setting to apply on the masked address input, and a field to indicate the size of the tags in the region. The TMT 236 may also receive as an input “SFR_TMT_Write” from writing the SFR value “SFR_TMT_Value.”

[00120] The TMT 236 may be configured to provide several outputs. For example, the TMT 236 may provide an instruction tag address or tag value (“CI_TMT_Value”). A bit

("CI_TMT_Valid") may be asserted to indicate that an instruction tag address is provided to look up an instruction tag from memory. A bit ("CI_TMT_Imm") may be asserted to indicate that an instruction tag value is written directly to the ITag FIFO of the first set of FIFOs 248.

[00121] The TMT 236 may also provide a data tag address or tag value ("Mem_TMT_Value"). A bit ("Mem_TMT_Valid") may be asserted to indicate that a data tag address is provided to look up a data tag from memory. A bit ("Mem_TMT_Imm") may be asserted to indicate that a data tag value is written directly to the DTag FIFO of the first set of FIFOs 248.

[00122] The TMT 236 may alternately provide a data tag address miss signal ("TMT_d_miss") if no data tag address is found. Similarly, the TMT 236 may provide an instruction tag address miss signal ("TMT_i_miss") if no instruction tag address is found.

[00123] When CI_TMT_Valid or Mem_TMT_Valid is asserted (i.e., when CI_TMT_Value or Mem_TMT_Value is used as an address), the inverse of the mask field may indicate how many input bits are OR'd with the output value to create the tag address. Alternatively, when an immediate value tag such as CI_TMT_Imm or Mem_TMT_Imm is asserted (i.e., when an instruction tag or data tag is written directly), the CI_TMT_Value and Mem_TMT_Value field may be used as-is as the tag. The resultant logic may be of the form:

```
Value[n]=TMT_Valid[n]&(InputPA &
TMT_Mask[n])==TMT_Addr[n];
Value[n]=TMT_Imm? TMT_Value[n]:
TMT_Value[n]|
((InputPA&~TMT_Mask[n])>>
TMT_Type.shift[n]),
```

where n ranges from 1 to the number of TMT 236 entries specified. The Value[n] of the highest-numbered entry with a set Valid[n] bit may be selected. If no Valid[n] bit is set, the PIPE 204 may issue an interrupt to the PEX 210 for a tag translation table miss, and may assert either "TMT_IMiss" and/or "TMT_DMiss" in the PipeStatus register, depending on whether the miss occurs with respect to the instruction tag or the data tag.

[00124] In embodiments in which the highest-numbered entry is selected, and in which all the foregoing comparisons are executed in parallel, several advantages may be achieved. For example, an immediate value tag may be specified for a memory region of any size, including

one or more sub-regions using an address map scheme that supersedes the immediate value. This provides a mechanism for sparse tag pages in memory.

[00125] In another advantage, an immediate value tag may be specified for a region of any size, with additional immediate tags for one or more of the sub-regions. This may be useful, for example, for peripheral device tags where a sub-region of the peripheral device has additional constraints. In yet another advantage, an address mapping may be specified globally, with some region(s) of memory using an immediate tag as a means of accelerating performance of the PIPE 204 and reducing memory access power.

[00126] In some embodiments, the TMT 236 contains a single set of map entries, but uses them on two sets of inputs and outputs. A first set may be for the current instruction physical address to generate the current instruction tag physical address. The second set may be for the data physical address to generate the data tag physical address.

[00127] The operation group/care LUT 238 will now be described in greater detail. In some embodiments, the operation group/care LUT 238 may be a simple 1K entry RAM with 10 bits of output for the operation group (“OPGRP”) and 33 care bits, with one for each of the rule cache 216 input fields and 22 for the rule cache instruction input. Example care bits are enumerated in table depicted in FIG. 8.

[00128] Each of the care bits shown in the table of FIG. 8 may be used to mask a corresponding input to the rule cache 216. For CItag and Mtag, no memory fetch occurs unless the care bit is set, and the output from the TMT 236 may not be written to the corresponding ITag FIFO or the DTag FIFO of the first set of FIFOs 248. The logic servicing the read side of the second FIFO 254 inspects the CItag and Mtag care bits to determine if the PIPE 204 needs to wait for the ITag and/or DTag FIFO.

[00129] In addition to saving power, this improves performance characteristics by avoiding unnecessary memory system accesses. Remaining care bits may mask the other inputs to the rule cache 216. OpTag RAM lookups may also only be performed when the corresponding care bit(s) are asserted. When a care bit is not asserted, the corresponding input field of the bit may be masked by being set to “1'b1” (i.e., not masked to zero). Masking to one may be advantageous in that it provides a means of distinguishing between masked fields and uninitialized/default fields.

[00130] In an implementation in which the AP 202 is a RISC-V core, for example, the RISC-V specific configuration of the Otag RAMs (including dual-port for Integer, dual and single for FP, and single for CSR) is a subset of the general configuration possibilities

supported by the PIPE 204. The PIPE 204 utilizes the interface parameters “*RF*” to implement an appropriate number of OPtag RAMs, widths, ports, and address bits to support various different ISAs.

[00131] As discussed above, the PEX core 210 may implement policy software which manages insertion and ejection from the rule cache 216. The TPU 208 may provide data that indicates lines of the rule cache 216 which are flagged as locked, marked invalid, and stamped with an installation count value. The TPU 208 may use this information to populate a “WrWayDest” field with a first invalid way entry, or a Least Recently Installed (LRI) entry that is not locked. The PEX core 210 may then decide to use the value as-is, or change the value to a different way number with a different algorithm utilizing the provided lock, invalid, and installation count information.

[00132] To communicate with the AP 202, the SOC bus 206, and the PEX core 210, the TPU 208 may utilize the communication interface described in the tables depicted in FIGs. 9A, 9B, 9C and 9D.

[00133] In some embodiments, the TPU 208 utilizes an ISA-specific header file to specify ISA-specific parameters about the AP 202 that the TPU 208 may benefit from being aware of. The table shown in FIG. 10 enumerates the parameters specified in this ISA-specific header file, and their corresponding values, using an example RISC-V RV32G implementation of the AP 202. The table depicted in FIG. 11 shows an example mapping between shadow memory, AP RF/CSR name, and AP RF/CSR address.

Metadata Tag Interlock Queue (MTIQ)

[00134] The MTIQ 212 will now be described in detail. As discussed above, the MTIQ 212 is generally configured to receive an operation output from the AP 202, and is configured to await a corresponding metadata tag from the TPU 208, then output the pair of the operation output and the corresponding metadata tag. In some embodiments, the MTIQ 212 may be configured according to several assumptions as set forth in the description below.

[00135] First, if there is an L1 data cache, the cache is a write-through cache that writes each value in the instruction order of the AP 202 as seen at the PIPE 204 input queue within FIFO 214.

[00136] Second, the operation outputs from the AP 202, or the L1 data cache, correspond to each instruction. Stated differently, a sequence of store-byte instructions yields a sequence

of byte writes to the MTIQ 212 layer. They are not queued up into a word write by the AP 202 or L1 data cache, but rather are written by the MTIQ 212.

[00137] Third, in embodiments in which the L1 data cache is present, no writes occur when a cache line is victimized because the cache is write-through.

[00138] Fourth, the PIPE 204 processes instructions in the same order as the AP 202 as it corresponds to memory writes.

[00139] Fifth, both cacheable and non-cacheable writes flow to the MTIQ 212 in the same sequence as the instructions that create the writes.

[00140] In some embodiments, the operation output FIFO 218 has a depth equivalent to a depth of the input FIFO 214. In other embodiments, however, the operation output FIFO 218 may have a smaller depth to take advantage of the likelihood of back-to-back memory writes being low.

[00141] When the TPU 208 writes a tag to the tag FIFO 220, the MTIQ 212 pops data waiting at the top of the queue and writes both the operation output and the metadata tag out to a main memory in an atomic manner. In some embodiments, the operation output and the metadata tag should remain paired together. However, in some embodiments, it may be insufficient to write the tag either before or after the operation output. For example, data governed by a tag may change from non-secret to secret as indicated by the tag. If the tag is written first, the address is marked secret before the secret data is written, thereby ensuring that the secret data is not written without a protecting tag. Accordingly, it is advantageous to write the tag first in such a scenario.

[00142] However, consider an example in which an address with secret data is being made non-secret. It may be advantageous to change the data before changing the tag to a less restrictive tag, thereby suggesting that it may be advantageous to write the tag after the data. Accordingly, it is advantageous to write the tag second in such a scenario.

[00143] Because the MTIQ 212 is incapable of knowing which case may arise ahead of time, applying a static “data-then-tag” or “tag-then-data” scheme may be ineffective. In one embodiment, therefore, three writes are executed. A first write changes the tag address to a predetermined constant value that always indicates a write transaction is underway. A second write writes the new data. A third write writes a new tag associated with the new data. If this sequence is interrupted after the first write, then the policy software may see a tag of a predetermined constant value that indicates the interruption on the data word in question and can take appropriate action.

[00144] As discussed above, the TPU 208 may utilize an immediate tag from the TMT 236 for a store command. When an immediate tag is used, there is no tag address to send a tag update to. Accordingly, a memory tag write may be instead sent to a special address. For example, the special address may include all ones (1's). The MTIQ 212 may interpret this special address to indicate that the pending data write is allowed, and the MTIQ 212 may write the data without any tag write.

[00145] In some embodiments, the MTIQ 212 introduces a window of time in which a memory coherency violation can occur. The window of time may open in a scenario in which a store word instruction causes data to be written to the operation output FIFO 218 to await a tag. However, before the MTIQ 212 writes the word out to a main memory, a load instruction reads the same address as the pending word store.

[00146] In other words, the load instruction is attempting to read "stale" information. The load instruction will therefore read the previous, stale data from memory, instead of the "fresh" value still pending in the MTIQ 212. To prevent this, in some embodiments read requests flow through the MTIQ 212, and the MTIQ 212 stalls reads that match an address pending in the operation output FIFO 218.

[00147] The MTIQ 212 may utilize the communication information listed in the tables shown in FIGs. 12A, 12B, and 12C, with the AP 202, PIPE 204, and AXI fabric.

[00148] As discussed above, the PEX 210 may be configured to execute a secure boot operation. The PEX 210 may include boot ROM which contains code that can read an image from flash memory, authenticate and decrypt the image combined with one of a number of public keys also stored in the boot ROM, write the application to a main memory, and then enable the AP 202 to continue the boot process.

[00149] For example, the secure boot process may begin in the PEX 210. At reset, the AP 202 may stay held in reset until the PEX 210 releases the AP 202 reset via a PIPE SFR. The PEX 210 may start execution at its reset vector, typically in on-chip ROM. The PEX 210 may boot the policy software into its own memory space, configure the memory fabric protection configuration registers to define the memory regions that each initiator can access, protecting a region of memory for itself to hold policy data, initialize that policy data segment, copy a boot-loader for the AP 202 from flash into main memory at an address of the PEX core's 210 choosing, optionally install a set of rules into the PIPE 204, and finally release the AP 202 from reset. At such point, the AP 202 may start executing a boot-loader from main memory to load in an OS and applications from flash memory. Accordingly, from

the very first instruction executed by the AP 202, the PIPE 204 may be checking each executed instruction against the policy rules.

[00150] The PEX 210 may be implemented according to one of several types of CPU cores. For example, the PEX 210 may be implemented as a RISC-V RV32I core. Similarly, the AP 202 may also be implemented as a RISC-V RV32I core. A selection of an implementation for the PEX core 210 and the AP 202 may vary depending on design preferences without adversely impacting operation of the computing system 200.

[00151] Having thus described several aspects of at least one embodiment, it is to be appreciated that various alterations, modifications, and improvements will readily occur to those skilled in the art. Such alterations, modifications, and improvements are intended to be within the scope of the present disclosure. Accordingly, the foregoing description and drawings are by way of example only, and the scope of the present disclosure should be determined from proper construction of the appended claims, and their equivalents.

[00152] It will be apparent that one or more embodiments described herein may be implemented in many different forms of software and/or hardware. Software code and/or specialized hardware (e.g., Application Specific Integrated Circuit, ASIC, and/or Field Programmable Gate Array, FPGA) used to implement embodiments described herein is not limiting of the embodiments of the present disclosure. Thus, to the extent the operation and behavior of embodiments are described without reference to specific software code and/or specialized hardware, it should be understood that one would be able to design software and/or hardware to implement the embodiments based on the description herein.

[00153] Further, certain embodiments of the example embodiments described herein may be implemented as logic that performs one or more functions. This logic may be hardware-based, software-based, or based on a combination of hardware and software. Some or all of the logic may be stored on one or more tangible, non-transitory, computer-readable storage media and may include computer-executable instructions that may be executed by a controller or processor. The computer-executable instructions may include instructions that implement one or more embodiments of the present disclosure. The tangible, non-transitory, computer-readable storage media may be volatile or non-volatile, and may include, for example, flash memories, dynamic memories, removable disks, and non-removable disks.

[00154] While example embodiments have been particularly shown and described, it will be understood by those skilled in the art that various changes in form and details may be

made therein without departing from the scope of the embodiments encompassed by the appended claims.