



- (51) **International Patent Classification:**
G06F 9/30 (2006.01)
- (21) **International Application Number:**
PCT/GB2016/051023
- (22) **International Filing Date:**
12 April 2016 (12.04.2016)
- (25) **Filing Language:** English
- (26) **Publication Language:** English
- (30) **Priority Data:**
1507795.1 7 May 2015 (07.05.2015) GB
- (71) **Applicant:** ARM LIMITED [GB/GB]; 110 Fulbourn Road, Cherry Hinton, Cambridge CB1 9NJ (GB).
- (72) **Inventor:** PARKER, Jason; c/o ARM Limited, 110 Fulbourn Road, Cherry Hinton, Cambridge, Cambridge-shire CB1 9NJ (GB).
- (74) **Agent:** MALDEN, Nicholas; D Young & Co LLP, 120 Holborn, London EC1N 2DY (GB).
- (81) **Designated States** (*unless otherwise indicated, for every kind of national protection available*): AE, AG, AL, AM,

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

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

Published:

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

(54) **Title:** CHECK INSTRUCTION FOR VERIFYING CORRECT CODE EXECUTION CONTEXT

```
IF ((CURRENT_EXCEPTION_LEVEL() & EL. FLAGS) AND
    (CURRENT_SECURITY_STATE() & S. FLAGS) AND
    (CURRENT_DEBUG_STATE() & DEBUG. FLAG))
    {NOP}
ELSE
    {ILLEGAL_INSTRUCTION_EXCEPTION}
```

FIG. 3B

(57) **Abstract:** A data processing apparatus and method of data processing are provided which make use of a processor state check instruction to determine if the data processing apparatus is currently operating in a processor state, defined by at least one runtime processor state configuration value, which matches a processor state check value defined by the processor state check instruction. Dependent on the required runtime processor state configuration value(s) matching the processor state check value, the processor state check instruction is treated as an ineffective instruction. When the at least one runtime processor state configuration value does not match the processor state check value an exception is generated. Improved security of the data processing apparatus is thus provided.



CHECK INSTRUCTION FOR VERIFYING CORRECT CODE EXECUTION CONTEXT

TECHNICAL FIELD

The present disclosure relates to a data processing apparatus. More particularly
5 it relates to checking that the context in which program code is executed on the data
processing apparatus is correct.

BACKGROUND

A data processing apparatus may operate in more than one state or mode,
10 which has various effects on the manner in which the data processing apparatus carries
out its data processing apparatus, but in particular may define which data stored in the
data processing apparatus is currently allowed to be accessed. For example, a memory
management unit (MMU) may be interposed between a processor of the data
processing apparatus and the memory in order to police access to the memory. The
15 configuration of the MMU can be such that when the processor is operating in a less
privileged mode or a non-secure state, it is not allowed to access regions of the
memory which have been defined only to be accessible to the processor when
operating in a privileged or secure state. This may for example be done by means of
the MMU accessing page tables stored in the memory, entries of which define which
20 regions of memory are accessible to which modes or states. The provision of such
techniques have previously been seen as sufficient to ensure that the desired security of
data items stored in the memory is respected during operation of the data processing
apparatus.

25 SUMMARY

In one example embodiment there is a data processing apparatus comprising:
data processing circuitry to perform data processing operations in response to a
sequence of data processing instructions, wherein the data processing circuitry is
responsive to at least one runtime processor state configuration value stored in the data
30 processing apparatus to perform the data processing operations in a processor state
defined by the at least one runtime processor state configuration value, wherein the
data processing circuitry is responsive to a processor state check instruction to

establish a no-operation condition dependent on the at least one runtime processor state configuration value matching a processor state check value defined by the processor state check instruction, and to treat the processor state check instruction as an ineffective instruction when the no-operation condition is established, and to
5 generate an illegal instruction exception when the no-operation condition is not established, wherein in response to said illegal instruction exception further execution of said sequence of data processing instructions is prevented.

In another example embodiment there is a computer program product in which
10 a computer program is stored in a non-transient fashion, which when executed on a computing device causes the computing device to operate as the data processing apparatus of the above example embodiment.

In another example embodiment there is a computer program product in which
15 a computer program is stored in a non-transient fashion comprising at least one processor state check instruction as defined in accordance with the above example embodiment.

In another example embodiment there is a dynamic code generator capable of
20 generating data processing instructions for a data processing apparatus according to above example embodiment, wherein the data processing instructions comprise the processor state check instruction.

In another example embodiment there is a virtual machine provided by a
25 computer program executing upon a data processing apparatus, said virtual machine providing an instruction execution environment corresponding to the apparatus of the above example embodiment.

In another example embodiment there is a data processing apparatus
30 comprising: means for performing data processing operations in response to a sequence of data processing instructions, wherein the data processing operations are performed in a processor state defined by at least one runtime processor state

configuration value stored in the data processing apparatus; means for establishing a no-operation condition dependent on the runtime processor state configuration value matching a processor state check value defined by a processor state check instruction; means for treating the processor state check instruction as an ineffective instruction
5 when the no-operation condition is established; and means for generating an illegal instruction exception when the no-operation condition is not established, wherein in response to said illegal instruction exception further execution of said sequence of data processing instructions is prevented.

10 In another example embodiment there is a method of data processing comprising: performing data processing operations in response to a sequence of data processing instructions, wherein the data processing operations are performed in a processor state defined by at least one runtime processor state configuration value; in response to a processor state check instruction, establishing a no-operation condition
15 dependent on the at least one runtime processor state configuration value matching a processor state check value defined by the processor state check instruction; treating the processor state check instruction as an ineffective instruction when the no-operation condition is established; and generating an illegal instruction exception when the no-operation condition is not established, wherein in response to said illegal
20 instruction exception further execution of said sequence of data processing instructions is prevented.

In another example embodiment there is a computer program product in which a computer program is stored in a non-transient fashion, which when executed on a
25 computing device causes the computing device to carry out the method of the above example embodiment.

BRIEF DESCRIPTION OF THE DRAWINGS

The present techniques will be described further, by way of example only, with
30 reference to embodiments thereof as illustrated in the accompanying drawings, in which:

Figure 1 schematically illustrates a data processing apparatus in one embodiment;

Figure 2A schematically illustrates the configuration of execution circuitry and registers in a data processing apparatus in one embodiment;

5 Figure 2B schematically illustrates privilege levels and/or exception levels supported in a data processing apparatus of one embodiment;

Figure 2C schematically illustrates a memory management unit interposed between a processor core and memory in one embodiment;

10 Figure 3A shows two example processor state check instructions in one embodiment;

Figure 3B shows a pseudo-code representation of the manner in which the data processing circuitry responds to the first processor state check instruction of Figure 3A in one embodiment;

Figure 4A shows a processor state check instruction in one embodiment;

15 Figure 4B schematically illustrates execution circuitry referencing a permission look-up table when executing the processor state check instruction of Figure 4A in one embodiment;

Figure 4C shows two processor state specific check instructions in one embodiment;

20 Figure 5 shows a sequence of steps which are carried out in the method of one embodiment;

Figure 6 schematically illustrates a general purpose computing device which is used to provide or support some embodiments; and

25 Figure 7 schematically illustrates a virtual machine environment according to one embodiment.

DESCRIPTION OF EMBODIMENTS

At least some embodiments provide a data processing apparatus comprising: data processing circuitry to perform data processing operations in response to a
30 sequence of data processing instructions, wherein the data processing circuitry is responsive to at least one runtime processor state configuration value stored in the data processing apparatus to perform the data processing operations in a processor state

defined by the at least one runtime processor state configuration value, wherein the data processing circuitry is responsive to a processor state check instruction to establish a no-operation condition dependent on the at least one runtime processor state configuration value matching a processor state check value defined by the processor state check instruction, and to treat the processor state check instruction as an ineffective instruction when the no-operation condition is established, and to generate an illegal instruction exception when the no-operation condition is not established, wherein in response to said illegal instruction exception further execution of said sequence of data processing instructions is prevented.

10

The present techniques recognise that although a data processing apparatus may be provided, the configuration of which is intended to ensure that data processing circuitry of the data processing apparatus cannot access particular regions of memory unless it is operating in a defined state which is allowed to do so, situations may nevertheless arise (whether due to inadvertent bugs in program code or an intended malicious attack) in which reliable security with respect to the data processing operations is not entirely guaranteed. For example, the present techniques recognise that it may be possible for bugs or attacks to lead to the “out of context” use of sensitive code, such as private algorithms that should not be exposed (i.e. their security should be maintained). Alternatively, situations may arise (whether due to bugs or an attack) in which the inappropriate use of code in a sensitive context may occur, such as pure user code from a kernel context. The present techniques recognise that such situations could lead to security vulnerabilities and exploits, despite correct configuration of, for example, memory protection attributes handled by a memory management unit.

25

To address this problem, the present techniques provide a processor state check instruction in the defined set of data processing instructions which the data processing apparatus is able to interpret and execute. This processor state check instruction causes data processing circuitry (e.g. a processor core, or more particularly an execution pipeline) to compare a processor state check value defined by the processor state check instruction with the current (runtime) processor state, in particular by

30

comparing to a runtime processor state configuration value stored in the data processing apparatus. The programmer of the data processing apparatus can therefore appropriately set a processor state check value defined by the processor state check instruction for the desired context in which the instruction should be executed. In
5 other words, this processor state check instruction can be inserted by the programmer of the data processing apparatus into one or more positions in a particular sequence of data processing instructions, such that when executing that sequence of data processing instructions the data processing circuitry will perform a check (in response to the processor state check instruction) to ensure that the data processing apparatus is
10 currently operating in a state which is appropriate (i.e. allowed) to execute that sequence of data processing instructions. If this check is passed then the data processing circuitry treats the processor state check instruction as a no-operation (NOP) instruction and execution continues without any side effects. However, if the check fails then the data processing circuit responds by generating an exception, which
15 can be appropriately defined to ensure that further execution of this sequence of data processing instructions is not possible. Accordingly, the provision of this processor state check instruction ensures that the system programmer has a further mechanism (to the known technique of correctly configuring memory protection attributes) to ensure that it can be checked (as often as the programmer cares to insert this processor
20 state check instruction into code) that the current context (mode or state) is appropriate for the data processing operations being carried out.

The nature of the processor state which is defined by the runtime processor state configuration value and which is checked by the use of the processor state check
25 instruction may take a variety of forms. In some embodiments, the processor state defined by the at least one runtime processor state configuration value comprises operation at an exception level of multiple exception levels at which the data processing apparatus is capable of performing the data processing operations. These multiple exception levels (which may in some contexts be referred to instead as
30 “privilege levels” or “rings”) may for example comprise four exception levels each with an increasing level of privilege and allowed accessibility to data in the data processing apparatus with respect to a lower exception level. In some embodiments,

the processor state defined by the at least one runtime processor state configuration value comprises a security state of multiple security states in which the data processing apparatus is capable of performing the data processing operations. For example, the data processing apparatus may be configured to operate in either a “secure” mode or a “non-secure” mode, wherein certain data stored in the data processing apparatus or particular sequences of program code held in the apparatus may only be accessed or executed respectively when the data processing apparatus is currently in its “secure” mode.

10 In some embodiments the processor state defined by the at least one runtime processor state configuration value comprises a debug state of multiple debug states in which the data processing apparatus is capable of performing the data processing operations. The ability to operate in a debug state may be provided for the data processing apparatus in order to allow the programmer greater visibility of the internal operation of the data processing apparatus as it carries out its data processing operations in response to a sequence of data processing instructions. However it is recognised by the present techniques that this additional visibility, which is provided to the programmer by operation in the debug state, and which may for example include the ability to cause the data processing apparatus to “single step” through data processing instructions, may only be appropriate to be provided in certain contexts, and for example, not in contexts where greater protection of data items or sequences of program instructions is to be provided.

25 In some embodiments the processor state defined by the at least one runtime processor state configuration value comprises a privilege mode of multiple privilege modes in which the data processing apparatus is capable of performing the data processing operations. Similarly, the data processing apparatus may be arranged to be able to operate in more than one privilege mode, wherein access to certain data items, or the execution of particular sequences of program code, should only be carried out when the data processing apparatus is in a “privileged” mode and not when operating in “non-privileged” mode and the present techniques support the enforcement of such constraints.

The above-mentioned examples of runtime processor state configuration values may individually define a processor mode or state, but in some embodiments the data processing circuitry is responsive to multiple runtime processor state configuration values stored in the data processing apparatus to perform the data processing operations in respective processor states defined by the multiple runtime processor state configuration values, and wherein the data processing circuitry is responsive to the processor state check instruction to require each of the multiple runtime processor state configuration values to match a corresponding processor state check value of multiple processor state check values defined by the processor state check instruction to establish the no-operation condition. In such embodiments the processor state check instruction can define values that each of the multiple runtime processor state configuration values must match in order for the no-operation condition to be established. In other words, one processor state check instruction may define required values for each of several different processor state configuration values.

Alternatively, multiple types of processor state check instruction may be provided, for example one type for each type of runtime processor state configuration value, and in such embodiments the data processing circuitry is responsive to multiple runtime processor state configuration values stored in the data processing apparatus to perform the data processing operations in respective processor states defined by the multiple runtime processor state configuration values, and wherein the data processing circuitry is responsive to each of multiple types of processor state check instruction to require a corresponding runtime processor state configuration value of the multiple runtime processor state configuration values to match a corresponding processor state check value defined by that processor state check instruction to establish the no-operation condition.

In some embodiments the processor state check instruction defines a particular value of the processor state check value for a given type of processor state which must be matched in order for the no-operation condition to be established. However, in some embodiments the processor state check instruction defines multiple processor

state check values for one type of processor state, and wherein the data processing circuitry is responsive to the processor state check instruction to require at least one of the multiple processor state check values defined for one type of processor state to match the corresponding runtime processor state configuration value stored in the data processing apparatus for the one type of processor state to establish the no-operation condition. In other words, the processor state check instruction can define more than one acceptable value for the runtime processor state configuration value in order to establish the no-operation condition. For example, in the above examples where several operational levels are defined for the operation of the data processing apparatus (e.g. exception levels or privilege levels), the processor state check instruction can define that more than one of these levels (e.g. the uppermost two) are acceptable states in which the data processing apparatus may be operating and thus for the no-operation condition to be allowed to be established.

15 The runtime processor state configuration value may be stored in a variety of locations or formats in the data processing apparatus, but in some embodiments the data processing apparatus comprises a state register to store the runtime processor state configuration value.

20 The processor state check instruction may define the corresponding processor state check value in a variety of ways. In some embodiments the processor state check value is comprised in the processor state check instruction. In other words, the corresponding processor state check value is then defined as a flag or immediate value given in the processor state instruction itself.

25 In some embodiments, the data processing apparatus further comprises check value storage to store the corresponding processor state check value and to which the processor state check instruction causes reference to be made by the data processing circuitry. This check value storage may for example take the form of a lookup table in the data processing apparatus for which the processor state check instruction provides an index, such that corresponding required processor state check values may be retrieved to be compared against.

The manner in which the processor state check instruction is treated as an ineffective instruction when the no-operation condition is established can take a number of forms. In some embodiments it comprises allowing the processor state
5 check instruction to proceed through an execution pipeline of the data processing circuitry without affecting an outcome of the data processing operations. Thus to the external observer the presence of the processor state check instruction in the sequence of instructions executed will have had no perceptible effect in terms of data processing outcome, although of course resources will have been consumed by virtue of the
10 passage of the processor state check instruction through the execution pipeline. In some embodiments treating the processor state check instruction as an ineffective instruction comprises treating the processor state check instruction as a no-operation instruction. As such the processor state check instruction may be discarded at the first opportunity.

15

At least some embodiments provide a computer program product in which a computer program is stored in a non-transient fashion, which when executed on a computing device causes the computing device to operate as the data processing apparatus of any of the above-discussed embodiments.

20

At least some embodiments provide a computer program product in which a computer program is stored in a non-transient fashion comprising at least one processor state check instruction as discussed above with reference to the above-discussed embodiments.

25

At least some embodiments provide a dynamic code generator capable of generating data processing instructions for a data processing apparatus of any of the above-described embodiments, wherein the data processing instructions comprise a processor state check instruction in any of the forms thereof mentioned above. For
30 example, this dynamic code generator may take the form of a just-in-time (JIT) engine.

At least some embodiments provide a virtual machine provided by a computer program executing upon a data processing apparatus, the virtual machine providing an instruction execution environment corresponding to the apparatus of any one of the above-described embodiments.

5

At least some embodiments provide a data processing apparatus comprising: means for performing data processing operations in response to a sequence of data processing instructions, wherein the data processing operations are performed in a processor state defined by at least one runtime processor state configuration value stored in the data processing apparatus; means for establishing a no-operation condition dependent on the runtime processor state configuration value matching a processor state check value defined by a processor state check instruction; means for treating the processor state check instruction as an ineffective instruction when the no-operation condition is established; and means for generating an illegal instruction exception when the no-operation condition is not established, wherein in response to said illegal instruction exception further execution of said sequence of data processing instructions is prevented.

At least some embodiments provide a method of data processing comprising: performing data processing operations in response to a sequence of data processing instructions, wherein the data processing operations are performed in a processor state defined by at least one runtime processor state configuration value; in response to a processor state check instruction, establishing a no-operation condition dependent on the at least one runtime processor state configuration value matching a processor state check value defined by the processor state check instruction; treating the processor state check instruction as an ineffective instruction when the no-operation condition is established; and generating an illegal instruction exception when the no-operation condition is not established, wherein in response to said illegal instruction exception further execution of said sequence of data processing instructions is prevented.

30

At least some embodiments provide a computer program product in which a computer program is stored in a non-transient fashion, which when executed on a

computing device causes the computing device to carry out the method of the above-mentioned embodiments.

Some particular embodiments are now described with reference to the figures.

5

Figure 1 schematically illustrates a data processing apparatus in one embodiment. The data processing apparatus 10 comprises an execution pipeline 12 and associated registers 14. Data processing instructions to be executed by the execution pipeline 12 are fetched by the instruction fetch unit 16, which on the basis of each instruction to be
10 issued to the execution pipeline 12 transmits various control signals to the execution pipeline 12 and the registers 14, so that the desired data processing takes place. The instruction fetch unit 16 retrieves the sequence of data processing instructions to be executed from memory 20, and the data processing apparatus 10 further comprises a local cache 18 in which copies of recently accessed instructions and data are
15 temporarily stored. This general arrangement of a data processing apparatus is one with which one of ordinary skill in the art will be familiar, and further detailed description thereof is omitted here for brevity. However, according to the present techniques, the data processing apparatus 10 is configured to be responsive to a “processor state check instruction”, which the programmer can use to ensure that the
20 context in which program code is executed on the data processing apparatus is correct. This is described in more detail below. Note that Figure 1 also schematically illustrates further data processing apparatus 21, which is running software comprising a just-in-time (JIT) dynamic code generator 22, which can generate and store a sequence of data processing instructions for execution by the data processing apparatus
25 10 in the memory 20, and these data processing instructions may include at least one of the processor state check instructions.

The registers 14 of the data processing apparatus 10 may be used in a variety of ways by the execution pipeline 12 shown in Figure 1 in order to support the data
30 processing operations instructed by the data processing instructions. However, one particular aspect thereof which is of relevance to the present techniques is the use of specific registers to hold one or more values which define a runtime processor state for

the processor core (i.e. execution pipeline or circuitry). This is schematically illustrated in Figure 2A. In this figure, four of this type of state register are illustrated each holding a runtime processor state configuration value, namely registers for the exception level 23, security state 24, debug state 26 and privilege mode 28. It should be noted that all four of these registers are illustrated in Figure 2A for the sake of providing a range of examples, but there is no need for all four such registers to be present in order to define the runtime processor state, and other embodiments may only have one or more of these registers. Indeed, it may for example be the case that there is no need to have both the exception level register 23 and the privilege mode register 28, since in some embodiments exception level and privilege level are essentially the same thing. For example in the context of the ARM architectures, 32-bit ARM is defined in terms of privilege levels, whilst 64-bit ARM is defined in terms of exception levels. Figure 2A further shows a debug enable value 30 held within the execution circuitry 12 which is configured in dependence on the value set in the debug state register 26.

Figure 2B schematically illustrates one example usage of the exception levels / privilege modes, in the context of a virtual machine environment being provided by the execution circuitry 12. Figure 2B shows guest operating system 34 and an application 32 which operate within this virtualised operating environment. Overall control of this guest operating system and the application it executes is maintained by the hypervisor 36, which further maintains overall control of the wider virtualised operating environment and dictates what a particular virtual machine (i.e. guest operating system and the applications it is running) can see and interact with in the system. Figure 2B further schematically illustrates the fact that different components of the system shown operate at different “exception levels” (EL0-3) or “privilege levels” (PL0-2), which define the level of control and visibility of other components on the system that a given component has. Thus, a privilege hierarchy is supported, with the application 32 operating at a lowest level of that hierarchy, the guest operating system 34 operating the next level of that hierarchy, and the hypervisor 36 operating at a still higher level. Note also that in Figure 2B a further still higher level of this privilege hierarchy is shown, namely the secure monitor 38, which operates at the

highest level of privilege supported in the system (“Secure-PL1” / EL3) in order to maintain strict control over the sub-division of the system into a “secure world” and a “non-secure world” to provide a reliable security division between trusted applications/operating systems which may handle security sensitive data and other applications/operating systems to which such trust is not afforded.

With reference to the above mentioned “secure world” and “non-secure world”, Figure 2C schematically illustrates one way in which access to defined secure and non-secure regions of the memory is implemented in one embodiment. Here, a processor core 40 seeks access to content of a memory 42 by issuing a memory access request. This memory access request is received by a memory management unit (MMU) 44 which polices such memory access requests and ensures in particular that access to a secure region of memory 46 is only afforded to the processor core 40 when it is operating in an appropriate context, i.e. one in which the relevant component issuing the memory access request is defined to be “secure”. Access to the non-secure region of memory 48 may for example be more freely provided, such that components operating in either the secure world or the non-secure world may access such data. The MMU 44 determines whether a given memory access request is allowed to proceed to the memory 42 in dependence on the attributes of a corresponding page table entry. The MMU 44 stores a number of page table entries 50 locally in order to expedite this process, but these page table entries 50 are originally retrieved, in a manner with which one of ordinary skill in the art will be familiar, from a full page table in memory as appropriate.

Figure 3A shows two example processor state check instructions (“MODE_ASSERT”) in one embodiment. It can be seen that the first example of the instruction defines fields which relate to the exception level, the security state, and the debug state. Each of these fields of the instruction can be set to have a value indicating the corresponding run processor state configuration value which is required for the MODE_ASSERT instruction to be treated as an ineffective instruction. In one embodiment of the data processing apparatus 10 of Figure 1, the MODE_ASSERT instruction is allowed to progress through the execution pipeline 12, but the control

signals from the instruction fetch unit (circuitry) 16 prevent this instruction from affecting the outcome of the data processing operations, for example by restricting the effect that the instruction can have on the content of the registers 14. In another embodiment of the data processing apparatus 10 of Figure 1, the control signals from the instruction fetch unit 16 cause this instruction `MODE_ASSERT` instruction to be treated as a no-operation (NOP) instruction and hence to immediately be discarded. Note again (similarly to the comment made with respect to Figure 2A) that there may be embodiments of the data processing apparatus which executes these instructions which only has one or other of the exception level and privilege mode runtime processor state configuration values, and correspondingly the second example `MODE_ASSERT` instruction of Figure 3A defines fields which relate to the privilege level, the security state, and the debug state. The configuration of the data processing apparatus is such that, for either example instruction, if at least one value defined in the `MODE_ASSERT` instruction in each set of fields matches the corresponding value currently defining the runtime processor state configuration (e.g. as stored in a corresponding register 14 in the example of Figures 1 and 2A) then the `MODE_ASSERT` instruction is treated as ineffective (e.g. as a NOP instruction). Otherwise an illegal instruction exception is generated.

Figure 3B shows example pseudo-code for implementing the response of the data processing apparatus to an example `MODE_ASSERT` instruction. Note that in the example of Figure 3B there is no dependency on a “privilege mode” value, and only a dependency on an “exception level” value, since the data processing apparatus in this example is arranged to have exception levels, but not privilege levels. It can be seen that at least one of the flags set for each of the exception level, the security state and the debug state in the instruction must match the current (runtime) level or state of each for the no-operation response to follow. Otherwise an illegal instruction exception is generated. Referring back to Figure 2B, note that some combinations of the values of the fields of the `MODE_ASSERT` instruction will always be invalid, due to the system level definition of the privilege hierarchy (in terms of exception/privilege levels and security), namely: PL2 and Secure; EL2 and Secure; and EL3 and Non-Secure.

Whilst the `MODE_ASSERT` instruction may comprise immediate values defining the run time processor state configuration value(s) which will allow the `MODE_ASSERT` instruction to be treated as a NOP instruction, the example

5 `MODE_ASSERT` instruction shown in Figure 4A is structured such that two indices into one or more permission lookup table are provided, and the `MODE_ASSERT` instruction is therefore configured to cause the data processing apparatus to reference the permission lookup table to determine the relevant runtime processor state configuration values. In the example of Figure 4A an exception level and security

10 state are required to match for the NOP response to the `MODE_ASSERT` instruction to be triggered. Figure 4B schematically illustrates an example data processing apparatus 60, which has a configuration to execute a `MODE_ASSERT` instruction such as that shown in Figure 4A. The execution circuitry 62 which executes the instruction uses each index to reference the permission lookup table 64, which returns

15 the relevant processor state check value (the allowed value for the corresponding run time processor state configuration value).

The example `MODE_ASSERT` instructions shown in Figure 3A and 4A define allowed runtime processor state configuration values for more than one type of

20 runtime processor state configuration value (e.g. for both exception level and security state in the example of Figure 4A). Figure 4C shows two further example `MODE_ASSERT` instructions which are each arranged to be runtime processor state configuration specific, in that each only defines allowed runtime processor state configuration values for one type of the value, namely the exception level in the first

25 example of Figure 4C and the security state in the second example of Figure 4C. Accordingly, each of the example instructions shown in Figure 4C will be treated as NOP instructions if at least one of the value flags set in that instruction matches the corresponding runtime processor state configuration value and will generate an illegal instruction exception otherwise.

30

Figure 5 shows a sequence of steps which are taken in the method of one embodiment. The flow can be considered to begin at step 70 at which instructions are

5 fetched from the memory and issued to the execution pipeline. Thereafter, at step 72, each instruction is examined and it is determined if a MODE_ASSERT instruction has been encountered. Whilst this is not true the flow returns to step 70 and the data processing apparatus continues fetching instructions from memory and issuing them to its execution pipeline. When a MODE_ASSERT instruction is encountered the flow proceeds to step 74, where it is determined if the current exception level at which the data processing apparatus (processor) is operating matches at least one exception level (EL) flag in the MODE_ASSERT instruction. If it does, then flow proceeds to step 76, where it is determined if the current security state matches at least one security flag defined in the MODE_ASSERT instruction. If it does, then the flow proceeds to step 10 78, where this instruction is treated as a NOP instruction and the flow returns to step 70. If however at either step 74 or step 76 a negative outcome to the determination occurs then the flow proceeds to step 80 where an illegal instruction exception is generated. It will be understood therefore that the MODE_ASSERT instruction discussed in the context of this example method is one which defines one or more 15 allowed exception levels and one or more allowed security state but does not for example define a constraint on debug state or privilege level as in the example of the generic MODE_ASSERT instruction shown in Figure 3A.

20 Figure 6 schematically illustrates a general purpose computing device 100 of the type that may be used to implement the above described techniques. The general purpose computing device 100 includes a central processing unit 102, a random access memory 104 and a read only memory 106, connected together via bus 122. It also further comprises a network interface card 108, a hard disk drive 110, a display driver 25 112 and monitor 114, as well as a user input/output circuit 116 with a keyboard 118 and mouse 120 all connected via the common bus 122. In operation the central processing unit 102 will execute computer program instructions that may for example be stored in the random access memory 104 and/or the read only memory 106. Program instructions could be additionally be retrieved from the hard disk drive 110 or 30 dynamically downloaded via the network interface card 108. The results of the processing performed may be displayed to a user via a connected display driver 112 and monitor 114. User inputs for controlling the operation of the general purpose

computing device 100 may be received via a connected user input output circuit 116 from the keyboard 118 or the mouse 120. It will be appreciated that the computer program could be written in a variety of different computer languages. The computer program may be stored locally on a recording medium or dynamically downloaded to
5 the general purpose computing device 100. According to the present techniques, the general purpose computing device 100 has a configuration such that it is responsive to one or more type of processor state check instruction as discussed above. The general purpose computing device 100 can also have a configuration which allows it to provide an instruction execution environment (i.e. a virtual machine) such as that
10 discussed above with reference to Figure 2B. The architecture of the general purpose computing device 100 could vary considerably and Figure 9 is only one example.

Figure 7 illustrates such a virtual machine implementation that may be used. Whilst the above described embodiments generally implement the present techniques
15 in terms of apparatus and methods for operating specific processing hardware supporting the techniques concerned, it is also possible to provide so-called virtual machine implementations of hardware devices. These virtual machine implementations run on a host processor 530 typically running a host operating system 520 supporting a virtual machine program 510. This may require a more powerful
20 processor to be provides in order to support a virtual machine implementation which executes at a reasonable speed, but such an approach may be justified in certain circumstances, such as when there is a desire to run code native to another processor for compatibility or re-use reasons. The virtual machine program 510 provides an application program interface to an application program 500 which is the same as the
25 application program interface which would be provided by the real hardware which is the device being modelled by the virtual machine program 510. Thus, program instructions including one or more examples of the above-discussed processor state check instruction may be executed from within the application program 500 using the virtual machine program 510 to model their interaction with the virtual machine
30 hardware.

Some examples of the present techniques are also be described in the following example configurations:

Example 1) A data processing apparatus comprising: data processing circuitry to
5 perform data processing operations in response to data processing instructions,
wherein the data processing circuitry is responsive to at least one runtime processor
state configuration value stored in the data processing apparatus to perform the data
processing operations in a processor state defined by the at least one runtime processor
state configuration value, wherein the data processing circuitry is responsive to a
10 processor state check instruction to establish a no-operation condition dependent on
the at least one runtime processor state configuration value matching a processor state
check value defined by the processor state check instruction, and to treat the processor
state check instruction as an ineffective instruction when the no-operation condition is
established, and to generate an exception when the no-operation condition is not
15 established.

Example 2) A data processing apparatus comprising: means for performing data
processing operations in response to data processing instructions, wherein the data
processing operations are performed in a processor state defined by at least one
20 runtime processor state configuration value stored in the data processing apparatus;
means for establishing a no-operation condition dependent on the runtime processor
state configuration value matching a processor state check value defined by a
processor state check instruction; means for treating the processor state check
instruction as an ineffective instruction when the no-operation condition is established;
25 and means for generating an exception when the no-operation condition is not
established.

Example 3) A method of data processing comprising: performing data processing
operations in response to data processing instructions, wherein the data processing
30 operations are performed in a processor state defined by at least one runtime processor
state configuration value; in response to a processor state check instruction,
establishing a no-operation condition dependent on the at least one runtime processor

state configuration value matching a processor state check value defined by the processor state check instruction; treating the processor state check instruction as an ineffective instruction when the no-operation condition is established; and generating an exception when the no-operation condition is not established.

5

In the present application, the words “configured to...” or “arranged to...” are used to mean that an element of an apparatus has a configuration able to carry out the defined operation. In this context, a “configuration” means an arrangement or manner of interconnection of hardware or software. For example, the apparatus may have
10 dedicated hardware which provides the defined operation, or a processor or other processing device may be programmed to perform the function. “Configured to” or “arranged to...” does not imply that the apparatus element needs to be changed in any way in order to provide the defined operation.

15 Although illustrative embodiments have been described in detail herein with reference to the accompanying drawings, it is to be understood that the invention is not limited to those precise embodiments, and that various changes, additions and modifications can be effected therein by one skilled in the art without departing from the scope and spirit of the invention as defined by the appended claims. For example,
20 various combinations of the features of the dependent claims could be made with the features of the independent claims without departing from the scope of the present invention.

25

CLAIMS

1. A data processing apparatus comprising:
data processing circuitry to perform data processing operations in response to a
5 sequence of data processing instructions, wherein the data processing circuitry is
responsive to at least one runtime processor state configuration value stored in the data
processing apparatus to perform the data processing operations in a processor state
defined by the at least one runtime processor state configuration value,
wherein the data processing circuitry is responsive to a processor state check
10 instruction to establish a no-operation condition dependent on the at least one runtime
processor state configuration value matching a processor state check value defined by
the processor state check instruction,
and to treat the processor state check instruction as an ineffective instruction
when the no-operation condition is established, and to generate an illegal instruction
15 exception when the no-operation condition is not established, wherein in response to
said illegal instruction exception further execution of said sequence of data processing
instructions is prevented.
2. The data processing apparatus as claimed in claim 1, wherein the processor
20 state defined by the at least one runtime processor state configuration value comprises
operation at an exception level of multiple exception levels at which the data
processing apparatus is capable of performing the data processing operations.
3. The data processing apparatus as claimed in claim 1 or claim 2, wherein the
25 processor state defined by the at least one runtime processor state configuration value
comprises a security state of multiple security states in which the data processing
apparatus is capable of performing the data processing operations.
4. The data processing apparatus as claimed in any preceding claim, wherein the
30 processor state defined by the at least one runtime processor state configuration value
comprises a debug state of multiple debug states in which the data processing
apparatus is capable of performing the data processing operations.

5. The data processing apparatus as claimed in any preceding claim, wherein the processor state defined by the at least one runtime processor state configuration value comprises a privilege mode of multiple privilege modes in which the data processing apparatus is capable of performing the data processing operations.

6. The data processing apparatus as claimed in any preceding claim, wherein the data processing circuitry is responsive to multiple runtime processor state configuration values stored in the data processing apparatus to perform the data processing operations in respective processor states defined by the multiple runtime processor state configuration values,

and wherein the data processing circuitry is responsive to the processor state check instruction to require each of the multiple runtime processor state configuration values to match a corresponding processor state check value of multiple processor state check values defined by the processor state check instruction to establish the no-operation condition.

7. The data processing apparatus as claimed in any of claims 1-5, wherein the data processing circuitry is responsive to multiple runtime processor state configuration values stored in the data processing apparatus to perform the data processing operations in respective processor states defined by the multiple runtime processor state configuration values,

and wherein the data processing circuitry is responsive to each of multiple types of processor state check instruction to require a corresponding runtime processor state configuration value of the multiple runtime processor state configuration values to match a corresponding processor state check value defined by that processor state check instruction to establish the no-operation condition.

8. The data processing apparatus as claimed in any preceding claim, wherein the processor state check instruction defines multiple processor state check values for one type of processor state,

and wherein the data processing circuitry is responsive to the processor state check instruction to require at least one of the multiple processor state check values defined for one type of processor state to match the runtime processor state configuration value stored in the data processing apparatus for the one type of processor state to establish the no-operation condition.

9. The data processing apparatus as claimed in any preceding claim, wherein the data processing apparatus comprises a state register to store the runtime processor state configuration value.

10

10. The data processing apparatus as claimed in any preceding claim, wherein the processor state check value is comprised in the processor state check instruction.

11. The data processing apparatus as claimed in any of claims 1-9, further comprising check value storage to store the processor state check value and to which the processor state check instruction causes reference to be made by the data processing circuitry.

12. The data processing apparatus as claimed in any preceding claim, wherein treating the processor state check instruction as an ineffective instruction comprises allowing the processor state check instruction to proceed through an execution pipeline of the data processing circuitry without affecting an outcome of the data processing operations.

13. The data processing apparatus as claimed in any preceding claim, wherein treating the processor state check instruction as an ineffective instruction comprises treating the processor state check instruction as a no-operation instruction.

14. A computer program product in which a computer program is stored in a non-transient fashion, which when executed on a computing device causes the computing device to operate as the data processing apparatus of any preceding claim.

15. A computer program product in which a computer program is stored in a non-transient fashion comprising at least one processor state check instruction as defined in any of claims 1-13.
- 5 16. A dynamic code generator capable of generating data processing instructions for a data processing apparatus as claimed in any of claims 1-13, wherein the data processing instructions comprise the processor state check instruction.
- 10 17. A virtual machine provided by a computer program executing upon a data processing apparatus, said virtual machine providing an instruction execution environment corresponding to the apparatus of any of claims 1-13.
18. A data processing apparatus comprising:
- 15 means for performing data processing operations in response to a sequence of data processing instructions, wherein the data processing operations are performed in a processor state defined by at least one runtime processor state configuration value stored in the data processing apparatus;
- 20 means for establishing a no-operation condition dependent on the runtime processor state configuration value matching a processor state check value defined by a processor state check instruction;
- means for treating the processor state check instruction as an ineffective instruction when the no-operation condition is established; and
- 25 means for generating an illegal instruction exception when the no-operation condition is not established, wherein in response to said illegal instruction exception further execution of said sequence of data processing instructions is prevented.
19. A method of data processing comprising:
- performing data processing operations in response to a sequence of data processing instructions, wherein the data processing operations are performed in a processor state defined by at least one runtime processor state configuration value;
- 30 in response to a processor state check instruction, establishing a no-operation condition dependent on the at least one runtime processor state configuration value

matching a processor state check value defined by the processor state check instruction;

treating the processor state check instruction as an ineffective instruction when the no-operation condition is established; and

- 5 generating an illegal instruction exception when the no-operation condition is not established, wherein in response to said illegal instruction exception further execution of said sequence of data processing instructions is prevented.

20. A computer program product in which a computer program is stored in a non-
10 transient fashion, which when executed on a computing device causes the computing device to carry out the method of claim 19.

1/8

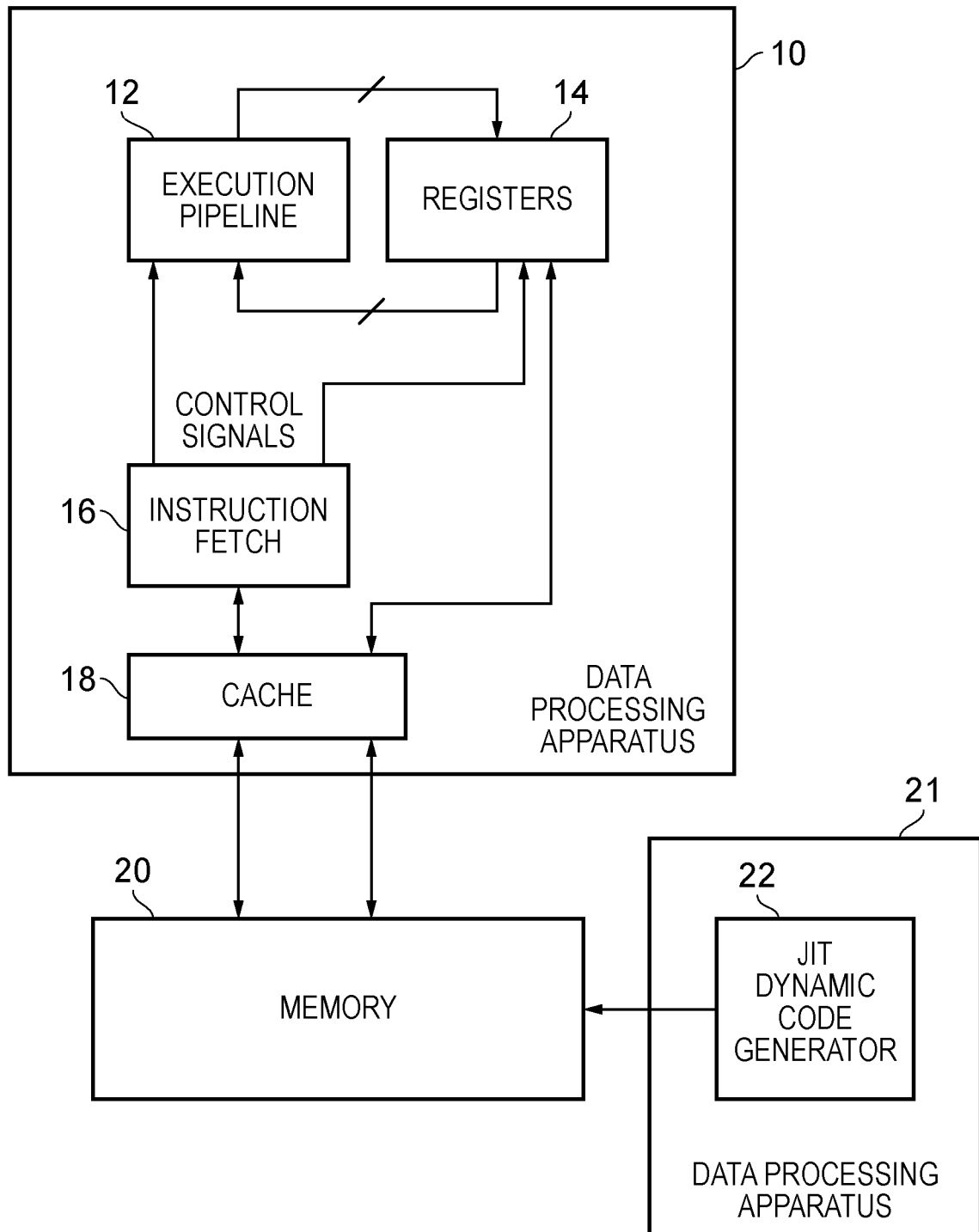


FIG. 1

2/8

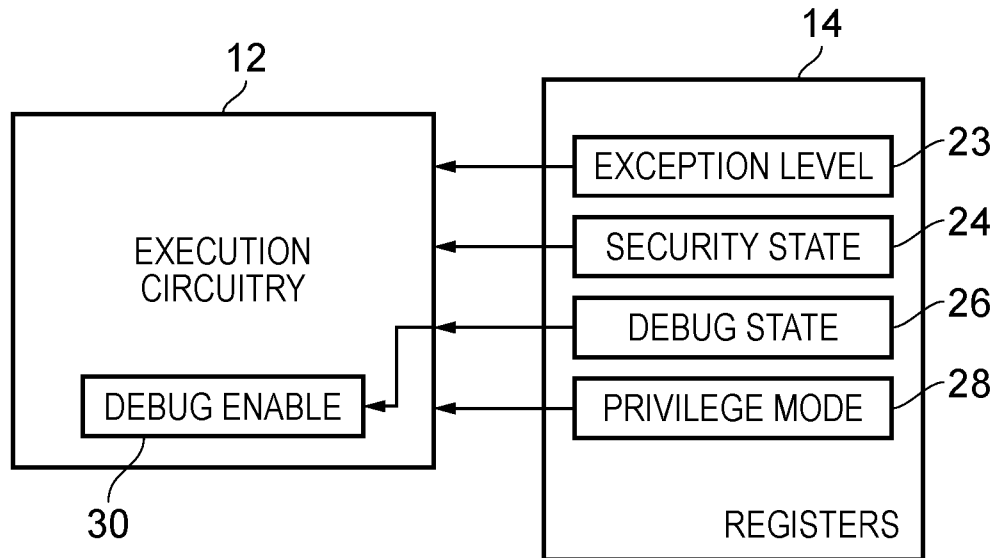


FIG. 2A

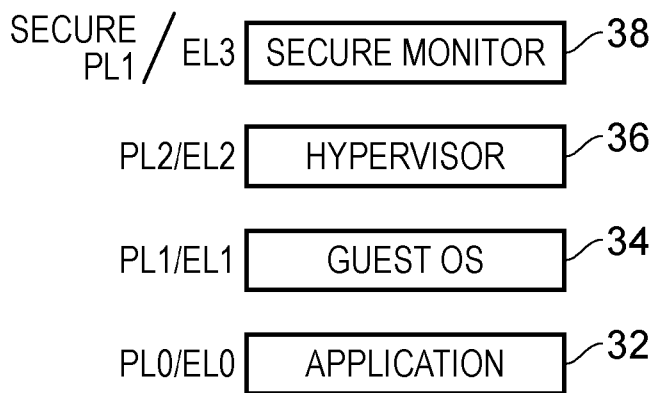


FIG. 2B

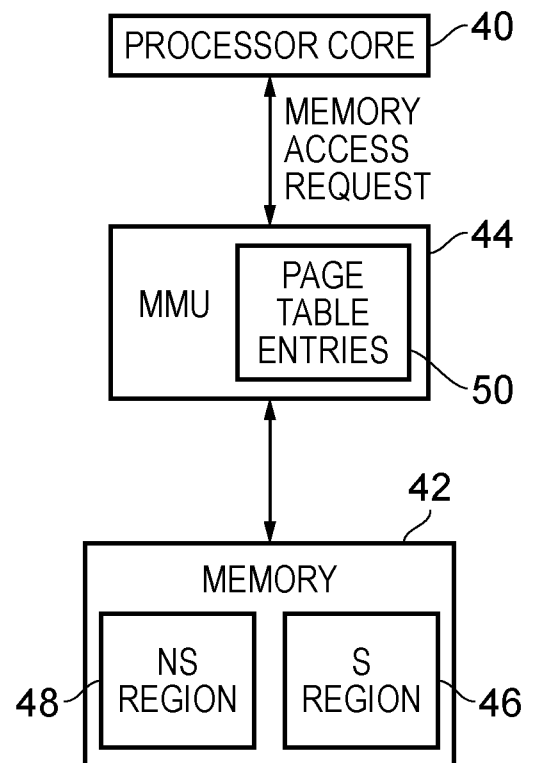


FIG. 2C

3/8

MODE_ASSERT <EL3, EL2, EL1, EL0>, <S, NS>,
<DEBUG, nDEBUG>

MODE_ASSERT <PL2, PL1, PL0>, <S, NS>,
<DEBUG, nDEBUG>

FIG. 3A

4/8

```
IF ((CURRENT_EXCEPTION_LEVEL() & EL. FLAGS) AND  
    (CURRENT_SECURITY_STATE() & S. FLAGS) AND  
    (CURRENT_DEBUG_STATE() & DEBUG. FLAG))  
    {NOP}  
ELSE  
    {ILLEGAL_INSTRUCTION_EXCEPTION}
```

FIG. 3B

5/8

MODE_ASSERT <EL_INDEX>, <S_INDEX>

FIG. 4A

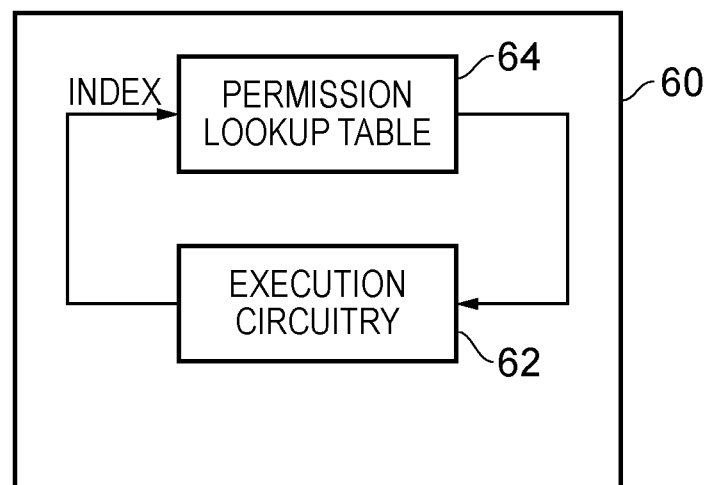


FIG. 4B

MODE_ASSERT_EXCEPTION_LEVEL <EL0, EL1, E2, EL3>

MODE_ASSERT_SECURE <S, NS>

FIG. 4C

6/8

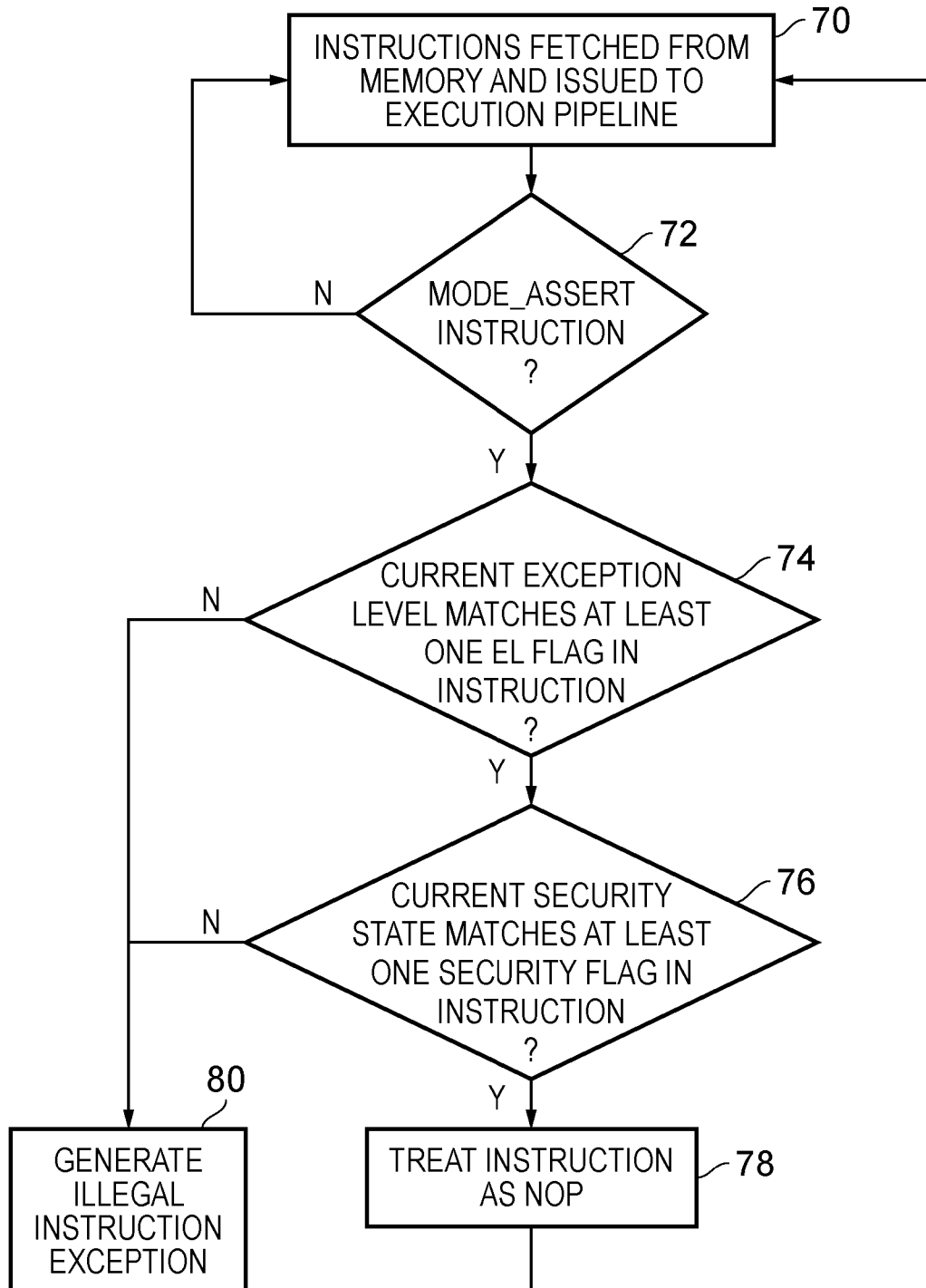


FIG. 5

7/8

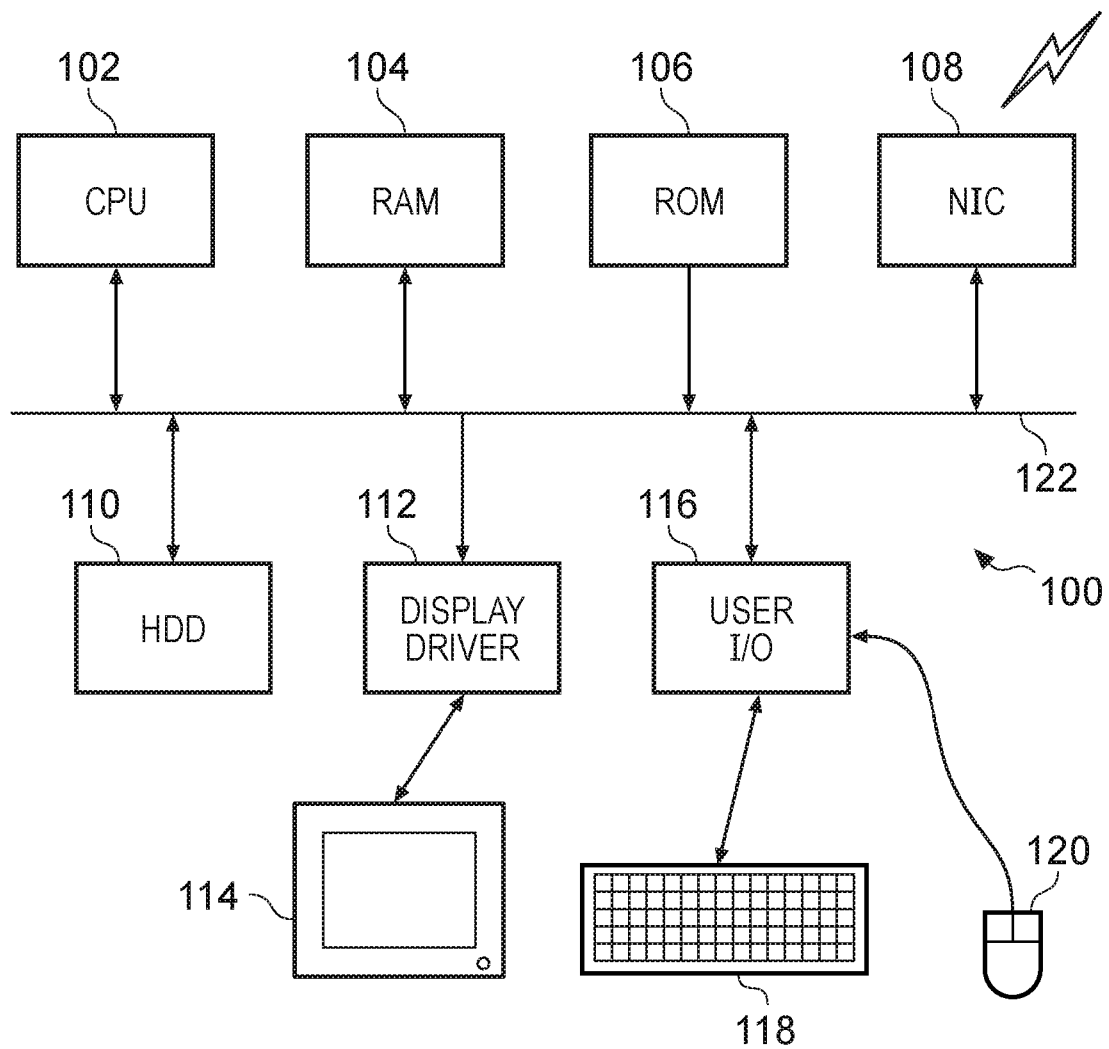


FIG. 6

8/8

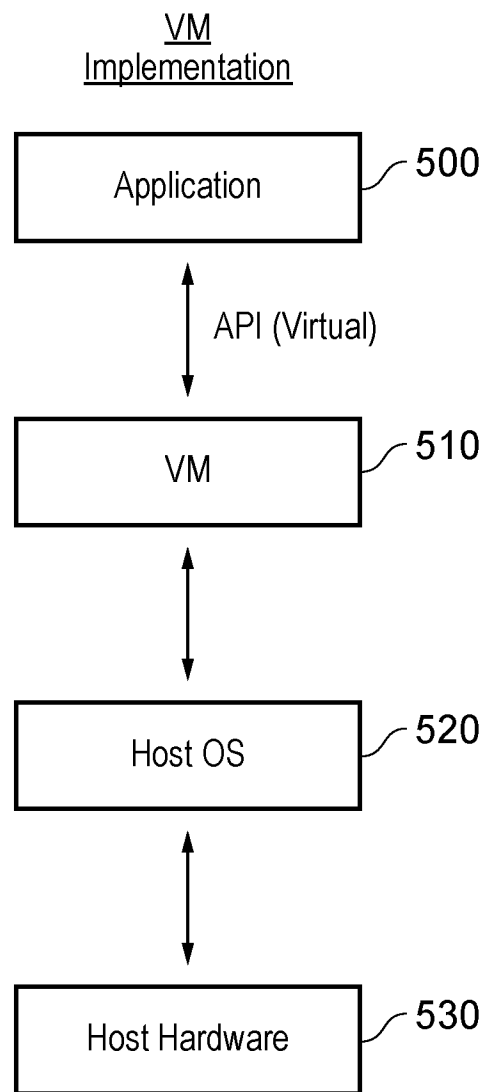


FIG. 7

INTERNATIONAL SEARCH REPORT

International application No

PCT/GB2016/051023

A. CLASSIFICATION OF SUBJECT MATTER

INV. G06F9/30
ADD.

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F

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

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

EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 6 513 110 B1 (KEPPEL DAVID [US] ET AL) 28 January 2003 (2003-01-28) the whole document -----	1-13, 17-20
A	EP 1 124 184 A2 (INST THE DEV OF EMERGING ARCHI [US]) 16 August 2001 (2001-08-16) the whole document -----	1-13, 17-20
A	GB 2 482 701 A (ADVANCED RISC MACH LTD [GB]) 15 February 2012 (2012-02-15) the whole document -----	1,9,18

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents :

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

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

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"Z" document member of the same patent family

Date of the actual completion of the international search

22 June 2016

Date of mailing of the international search report

04/07/2016

Name and mailing address of the ISA/

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

Authorized officer

Klocke, Lynn

INTERNATIONAL SEARCH REPORT

International application No.
PCT/GB2016/051023

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:
2. ☒ Claims Nos.: 14-16
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:
see FURTHER INFORMATION sheet PCT/ISA/210
3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.
2. ☐ As all searchable claims could be searched without effort justifying an additional fees, this Authority did not invite payment of additional fees.
3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:
4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

- ☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.
- ☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.
- ☐ No protest accompanied the payment of additional search fees.

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

Continuation of Box II.2

Claims Nos.: 14-16

The subject-matter of claims 14-16 cannot be searched, as these claims relate to programs for computers, which are not patentable under Rule 39.1(vi) PCT. The subject-matter of these claims does not achieve any further technical effect, as the program is not running on a computer.

The applicant's attention is drawn to the fact that claims relating to inventions in respect of which no international search report has been established need not be the subject of an international preliminary examination (Rule 66.1(e) PCT). The applicant is advised that the EPO policy when acting as an International Preliminary Examining Authority is normally not to carry out a preliminary examination on matter which has not been searched. This is the case irrespective of whether or not the claims are amended following receipt of the search report or during any Chapter II procedure. If the application proceeds into the regional phase before the EPO, the applicant is reminded that a search may be carried out during examination before the EPO (see EPO Guidelines C-IV, 7.2), should the problems which led to the Article 17(2) declaration be overcome.

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/GB2016/051023

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 6513110	B1	28-01-2003	NONE
EP 1124184	A2	16-08-2001	EP 1124184 A2 16-08-2001
		JP 2001249848 A	14-09-2001
		KR 20010078371 A	20-08-2001
		US 7680999 B1	16-03-2010
GB 2482701	A	15-02-2012	CN 103052941 A 17-04-2013
			EP 2572274 A1 27-03-2013
			GB 2482701 A 15-02-2012
			JP 5699213 B2 08-04-2015
			JP 2013539574 A 24-10-2013
			KR 20130137591 A 17-12-2013
			TW 201229809 A 16-07-2012
			US 2012042154 A1 16-02-2012
			WO 2012020238 A1 16-02-2012