

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
31 December 2008 (31.12.2008)

PCT

(10) International Publication Number
WO 2009/001038 A2

(51) International Patent Classification:
G06F 11/36 (2006.01)

(21) International Application Number:
PCT/GB2008/002090

(22) International Filing Date: 19 June 2008 (19.06.2008)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
0712407.6 26 June 2007 (26.06.2007) GB

(71) Applicant (for all designated States except US): **SYMBIAN SOFTWARE LIMITED** [GB/GB]; 2-6 Boundary Row, Southwark, London SE1 8HP (GB).

(72) Inventor; and

(75) Inventor/Applicant (for US only): **HAYUN, Roy** [GB/GB]; Symbian Software Limited, 2-6 Boundary Row, Southwark, London SE1 8HP (GB).

(74) Agent: **KOEN, Sophia**; Symbian Software Limited, 2-6 Boundary Row, Southwark, London SE1 8HP (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, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, 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, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, NO, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— without international search report and to be republished upon receipt of that report

(54) Title: **AUTHORISED DEBUGGING**

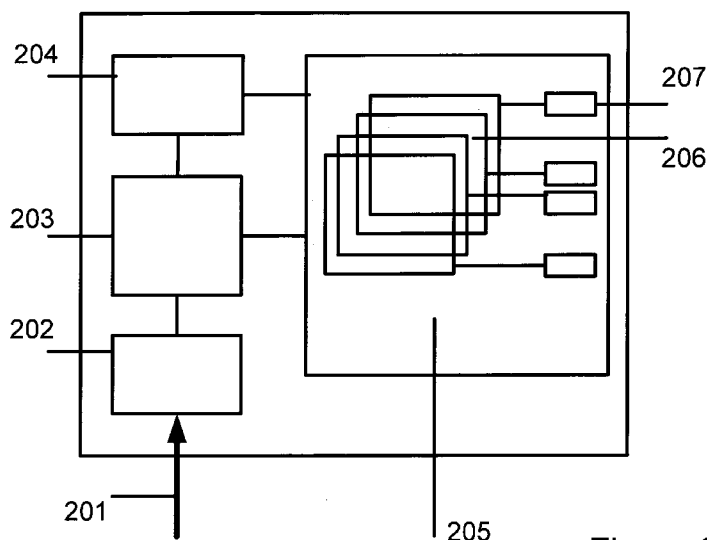


Figure 2

(57) Abstract: An access controller for controlling access by a debugger to a set of program instructions stored on a computing device, the access controller being responsive to a control signal indicative of a request by the debugger to access at least part of the set of program instructions to determine that the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted and inhibit the debugger from causing the computing device to suspend execution of the requested set of program instructions at an instruction contained in the subset of instructions when the computing device is executing the requested set of program instructions responsive to the debugger's request.

WO 2009/001038 A2

AUTHORISED DEBUGGING

The invention relates to an access controller for controlling access by a debugger to a set of program instructions.

A debugger is a program that may be used to run another program, sometimes on behalf of a user. Typically a debugger is used to test and debug other programs. While the target program (or debuggee) is running, the debugger may be able to pause, resume or restart the target, display or change values and watch for calls to particular functions, changes to particular variables etc.

Four key principles of debugger design are as follows:

1. The debugger should not affect the normal operation of the system;
2. The debugger should always provide truthful information during debugging;
3. The debugger should have the ability to provide program context information; and
4. System development occurs before debugging support is available.

Different debuggers have been developed for use with different programming languages. Different debuggers may also be used for debugging source code from those used to debug machine code. Debuggers may be implemented as stand-alone debuggers or as part of an integrated development environment (IDE).

A debugger may be used as a tool for assisting a programmer to identify problems with a computer program. When a program crashes (e.g. when the program cannot continue because of a programming bug, such as the program trying to use an instruction not available on the current version of the CPU or attempting to access protected memory), a debugger will attempt to identify the code position at which the problem occurred. If the debugger is a source-level debugger, the identified position will be in the original code whereas for a machine-level debugger the identified position will be in the disassembly. A debugger may also provide the user with information about the program state that existed when the crash occurred.

Typically, debuggers also provide more sophisticated analysis tools such as single-stepping (which enables a program to be run step-by-step) and breaking (pausing the

program to examine the current state). These tools are generally implemented by means of breakpoints. The debugger uses breakpoints to mark the points in the program at which it should pause. After having inserted a breakpoint at the appropriate point in the program, the debugger allows the program to start running. When the program reaches the breakpoint, it stops until the debugger removes the breakpoint and allows the program to start running again. The debugger may insert a new breakpoint before the program starts running. Typically, when the target program is paused, the debugger displays the contents of the memory and other information.

If the instruction stream includes a branch instruction, the debugger typically sets a breakpoint at the destination of that branch too. If the debugger is implementing a single-step procedure and there's a branch instruction in the next line, the debugger will place the breakpoint at the start of the function called by the branch instruction rather than the "next line" of the instructions currently being executed.

Breakpoints can be made conditional so that the breakpoint is only executed if some predefined condition is met. For example, the breakpoint may be made conditional on a specific variable having a particular value, on the number of times a loop has been executed or on the name of a file.

To set a breakpoint, a debugger usually replaces the instruction at the target address with an instruction that causes the debugger to become active. When the debugger becomes active, it replaces its instruction with the original instruction. The instruction that the debugger uses for the breakpoint may be a software interrupt, an opcode defined for the purpose, an invalid instruction or a jump instruction that leads back into the debugger's code. Some processors have registers that can be used to implement breakpoints without having to manipulate the instruction stream. Alternatively, the processor may have a special operation mode that causes a software interrupt to fire automatically after each instruction is executed.

When a debugger causes a program to pause by means of a breakpoint the program is temporarily suspended. Entering break mode does not terminate or end execution

of the program. Execution can be resumed at any time. Also, entering break mode does not change the program state. For example, the functions, variables and objects remain in memory, it is only their movements and activities that are suspended.

A debugger therefore allows a programmer to inspect a computer program. For example, an on-target debugger allows the user to see how values are changing as the program is executed, which branches the program is taking and which functions are actually being called. Furthermore, some debuggers allow the user to make adjustments to the program while in break mode. For example, the user may change the value of a variable, move the execution point or even make changes to the code itself.

The same functionality which makes a debugger useful for eliminating bugs in a computer program also allows it to be used as a tool for reverse engineering the code and for maliciously changing the code. For example, a debugger could be used to learn how to evade copy protection or other programmed limitations. Therefore, there is a need for an improved computer program that incorporates a degree of protection against a debugger.

The features of a debugger given above are for the purposes of example only. The access controller and computer program according to the invention are not limited to use with any specific type of debugger and are not limited to use with a debugger having the specific features described above.

According to a first embodiment of the invention, there is provided an access controller for controlling access by a debugger to a set of program instructions stored on a computing device, the access controller being responsive to a control signal indicative of a request by the debugger to access at least part of the set of program instructions to determine that the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted and inhibit the debugger from causing the computing device to suspend execution of the requested set of program instructions at an instruction contained in the subset of

instructions when the computing device is executing the requested set of program instructions responsive to the debugger's request.

The access controller may be arranged to determine whether the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted in dependence on an area of memory in which one or more of the requested set of program instructions are stored.

The access controller may be arranged to determine that the subset of instructions is a set of instructions to which access by the debugger is to be restricted in dependence on the provision of an authorisation associated with that subset of instructions. The authorisation may be a key and the access controller may comprise a copy of the key. A copy of the key may also be stored in a memory of the computing device and the access controller may be arranged to access the memory of the computing device so as to obtain that copy of the key.

The access controller may be arranged to access the memory of the computing device responsive to a control signal indicative of a direction from the debugger to the copy of the key stored in the memory of the computing device.

The access controller may be arranged to inhibit the debugger from causing the computing device to suspend execution of the requested set of instructions at an instruction contained in the subset of instructions if the copy of the key comprised in the computing program does not correspond to the copy of the key stored in the memory of the computing device. The access controller may be arranged to determine that the copy of the key comprised in the computing program does not correspond to the copy of the key stored in the memory of the computing device if the two keys are not identical.

The access controller is arranged to inhibit the debugger from changing the instructions contained in the subset of instructions and/or from changing a program state associated with the execution of the subset of instructions by the computing device.

According to a second aspect of the invention, there is provided a computer program comprising program instructions for controlling access by a debugger to a set of program instructions stored on a computing device, the program controller being responsive to a control signal indicative of a request by the debugger to access at least part of the set of program instructions to determine that the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted and inhibit the debugger from causing the computing device to suspend execution of the requested set of program instructions at an instruction contained in the subset of instructions when the computing device is executing the requested set of program instructions responsive to the debugger's request.

According to a third aspect of the invention, there is provided a computer program comprising program instructions that are accessible by a debugger, the computer program being arranged such that one or more of the instructions form a subset of instructions to which access by a debugger is to be restricted by inhibiting the debugger from causing a computing device to suspend execution of the requested set of program instructions at an instruction contained in the subset of instructions when the computing device is executing the requested set of program instructions responsive to a request by the debugger.

The computer program may be arranged such that the subset of instructions to which access by a debugger is to be restricted is associated with an authorisation requirement, access to the subset of instructions being unrestricted if said authorisation requirement is fulfilled.

The computer program may be arranged such that the instructions form a plurality of subsets to which access by a debugger is to be restricted, each subset of instructions being associated with a respective authorisation requirement.

The computer program is preferably stored in a memory of a computing device, the subset of instructions to which access by the debugger is to be restricted being

formed by the way in which the program instructions of the computer program are arranged in the memory.

According to a fourth aspect of the invention, there is provided a method for controlling access by a debugger to a set of program instructions stored on a computing device, the method comprising receiving a control signal indicative of a request by the debugger to access at least part of the set of program instructions, determining that the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted and inhibiting the debugger from causing the computing device to suspend execution of the requested set of program instructions at an instruction contained in the subset of instructions when the computing device is executing the requested set of program instructions responsive to the debugger's request.

According to a fifth aspect of the invention, there is provided an access controller for inhibiting a debugger from accessing memory locations used by a set of program instructions stored on a computing device, the access controller being responsive to a control signal indicative of a request by the debugger to execute at least part of the set of program instructions to determine whether the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted, cause the requested set of program instructions to be executed by a processor, the processor being capable of accessing registers and other memory and inhibit the debugger from accessing memory locations contained in the other memory that are used during execution of the requested set of program instructions by the restricted set of program instructions.

The access controller may be arranged to inhibit the debugger from accessing the memory locations used by the restricted set of program instructions while the requested set of program instructions is being executed.

The access controller may be arranged to inhibit the debugger from accessing the memory locations used by the restricted set of program instructions following a

suspension in the execution of the requested set of program instructions by the processor.

The access controller may be arranged to inhibit the debugger from accessing the memory locations used by the restricted set of program instructions following a suspension in the execution of the requested set of program instructions caused by an error in the execution of those program instructions.

The access controller may be arranged to inhibit the debugger from accessing memory locations in the other memory that are storing instructions contained in the restricted set of program instructions and/or memory locations in the other memory that are storing program data associated with the restricted set of program instructions.

The other memory is preferably external to the processor and may be a random access memory.

According to a sixth aspect of the invention, there is provided a computer program comprising program instructions for inhibiting a debugger from accessing memory locations used by a set of program instructions stored on a computing device, the computer program being responsive to a control signal indicative of a request by the debugger to execute at least part of the set of program instructions to determine whether the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted, cause the requested set of program instructions to be executed by a processor, the processor being capable of accessing registers and other memory and inhibit the debugger from accessing memory locations contained in the other memory that are used during execution of the requested set of program instructions by the restricted set of program instructions.

According to a seventh aspect of the invention, there is provided a method for inhibiting a debugger from accessing memory locations used by a set of program instructions stored on a computing device, the method comprising receiving a control signal indicative of a request by the debugger to access at least part of the set of

program instructions, determining whether the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted, executing the requested set of program instructions by means of a processor, the processor being capable of accessing registers and other memory and inhibiting the debugger from accessing memory locations contained in the other memory that are used during execution of the requested set of program instructions by the restricted set of program instructions.

A computer program according to embodiments of the invention is suitably an operating system.

According to an eighth aspect of the invention, there is provided a computing device comprising an access controller as claimed in any of claims 1 to 11 or claims 18 to 25.

According to a ninth aspect of the invention, there is provided a computing device programmed with a computer program as claimed in any of claims 12 to 16 or claim 26.

The computing device may comprise a memory that stores a key for authorising a debugger to access a subset of program instructions that are contained in the computer program and to which access by the debugger is to be restricted.

The invention will now be described by way of a specific example and with reference to the accompanying drawings in which:

Figures 1a to d show a debugger accessing a computer program that has been arranged into modular levels;

Figure 2 shows the functional components of a computer program according to certain embodiments of the invention;

Figure 3 shows a process performed by a device running a computer program according to certain embodiments of the invention;

Figure 4 shows a process performed by a device running a computer program according to certain embodiments of the invention;

Figure 5 shows an operating system running on a mobile phone and on an emulator;

Figure 6 shows an emulator debug architecture;

Figure 7 shows a run-mode debug architecture;

Figure 8 shows a host debugger and a debug agent on a target platform;

Figure 9 shows a stop-mode debug architecture;

Figure 10 shows the interactions of a stop-mode debugger; and

Figure 11 shows the functional components of a mobile phone on which a computer program according to certain embodiments of the invention may suitably be run.

A computer program according to certain embodiments of the invention is suitably arranged in modular levels so that it contains at least one subset of program instructions, which contains one or more of the instructions that make up the program. Access to this subset of instructions is restricted so that although a debugger can cause that subset of instructions to be run by a processor along with the rest of the computer program, the operations that the debugger is permitted to perform in respect of that subset of instructions is limited to causing those instructions to be executed. In particular, the debugger is inhibited from causing the program to stop running the set of instructions at an instruction that is contained in the subset (e.g. the debugger is inhibited from placing a breakpoint at one of those instructions).

Certain embodiments of the invention also provide an access controller that is capable of controlling access by a debugger to a set of program instructions stored on a computing device. The access controller may suitably be implemented as a computer program. If the access controller is implemented as a computer program, the program instructions to which the access controller is arranged to control access may form part of the same computer program as the access controller or may be part of a different computer program.

The access controller is preferably capable of receiving a control signal from a debugger that is indicative of a request by the debugger to access at least part of a set of program instructions. The access controller is arranged to control access to those instructions by the debugger in dependence on the request. For example, the access controller may determine that the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted. The access controller may then inhibit the debugger from causing the computing device to suspend execution of the program instructions at an instruction contained in that subset of instructions.

According to certain embodiments of the invention, the access controller may be arranged to control access by a debugger to a requested set of program instructions by inhibiting the debugger from accessing memory used by a subset of instructions to which access by the debugger is to be restricted. Under certain circumstances the debugger may be able to access registers used during execution of the restricted set of program instructions. For example, if there is an error that occurs during execution of the restricted set of program instructions, the debugger may be able to inspect the contents of the registers after the crash. However, the access controller is preferably able to stop the debugger from accessing other memory used by the restricted set of program instructions. This memory could include memory external to the processor, such as random access memory, and may encompass memory locations used to store the restricted set of program instructions and other data used by the restricted set of program instructions (e.g. data used or generated during execution of the restricted set of instructions).

The access controllers and computer program described above are advantageous because they enable certain sensitive parts of a computer program to be protected from reverse engineering or alteration by means of a debugger without excluding the debugger from other, less sensitive parts of the program. Therefore, the computer program is still able to be debugged by third parties trying to further develop the program but certain important parts of the program are protected.

A computer program may suitably be used to implement an access controller according to certain embodiments of the invention. Such a computer program is capable of interacting with another computer program (the debugger) to control what the debugger is permitted to do in terms of analysing a set of program instructions. Therefore, the computer program can act as an access controller between a set of program instructions and a user trying to analyse those program instructions using an analysis tool (the debugger). The computer program may therefore limit how a user can cause a set of program instructions to execute.

Preferably the restricted access that is permitted the debugger does not prevent the debugger from accessing information contained in the registers in the conventional way if the restricted part of the program should crash during operation. In this way, third parties will still be able to access at least some of the information that is normally presented to third party programmers in the case of the program independently crashing, which should assist them to discover the cause of the crash. However, third parties will be prevented from accessing memory locations other than the registers that are used by the restricted part of the program and from pausing or single-stepping through the restricted part of the program, which helps to prevent third parties from reverse engineering sensitive parts of the program. Third parties are also preferably prevented from altering code contained in restricted sections of the program or from altering a program state resulting from running the code in restricted sections of the program.

Because third party developers may genuinely need to step-through or pause even sensitive areas of the program, or access memory locations used by sensitive parts of the program, a computer program according to certain embodiments of the

invention may be arranged such that a restriction that applies to a certain section of code can be overcome by provision of an appropriate authorisation. This is illustrated in figures 1a to d.

Figure 1a shows a computer program that has been divided into three different modular levels, each of which is associated with a different restriction level. Level 3 is subject to unrestricted access and so the debugger is free to access the program instructions contained in this level and to analyse these instructions in accordance with its full range of analysis tools (as shown in figure 1b). Level 2 is subject to restricted access. However, this restriction may be overcome by provision of an appropriate authorisation, which in this case is a key. The key shown in figure 1c is the correct key for level 2 and so the debugger is able to overcome the restriction and freely access level 2. Level 1 is either subject to a restriction that cannot be overcome at all or that cannot be overcome by the particular key shown in the figure. The debugger is therefore only able to cause instructions that are contained in level 1 to execute normally. The debugger is not able to pause or step-through the instructions contained in level 1.

The authorisation key may suitably be provided by the computing device on which the computer program is loaded, e.g. by being stored in the memory of that device. For example, the computer program may be loaded on a mobile phone and the mobile phone manufacturer may burn a key onto the read-only memory (ROM) of selected phones. Storing the key in a protected part of ROM is advantageous because it prevents any external interrogator from being able to access the key. Storing the key on ROM also has the advantage of keeping the key secret from any third parties, as they do not need to know the key to be able to debug sensitive areas of the program.

When a third party uses a debugger to access the computer program stored on the phone, the debugger directs the computer program to the key stored on the ROM of the computing device on which the computer program is being run. For example, the debugger may pass an operand to the computer program that specifies the area of ROM in which the key is stored. The computer program determines if the debugger is

authorised to overcome a restriction associated with a particular subset of program instructions in dependence on this key.

The computer program has its own copy of the key, and if this copy of the key matches that which is stored on the memory of the computing device then the debugger is considered to be authorised to step-through restricted parts of the code. If the keys do not match, or if no key is stored at that memory location, then the computer program determines that the debugger is not authorised to access the restricted parts of the code.

In this way, the manufacturer can control which third parties are able to debug sensitive parts of the computer program, simply by burning this authorisation onto the ROM of the phone provided to that third party. If a third party is not authorised to access the restricted parts of the code, the key will either not be present in the ROM of the computing device or the wrong key will be present (which could simply be other data stored at the location to which the computer program was directed by the debugger). The computer program will therefore not authorise the debugger to access the restricted parts of the code. This is equivalent to providing special “development” phones (or other computing devices) to third parties trusted by the manufacturer to carry out further development work. Also, only trusted third parties will be told where on the ROM the appropriate keys are stored (so that the debugger provides the computer program with the correct direction), thus providing a further layer of security.

The authorisation procedure described above provides two assurances: first, the computer program knows that it is running on an appropriate “development device” if the device has the correct key stored on its ROM; second, the computer program knows that the debugger is being operated by an authorised third party if the debugger directs it to the correct part of the ROM to find the key.

Instead of a direct comparison between a key stored on the computing device and a key that forms part of the computer program, the authentication process might be slightly more complex, e.g. a public-private key authentication process in which the

computer program would be required to perform some function on the key stored by the computing device in order to verify that the key is correct.

As an alternative authorisation mechanism, a copy of the key might be in the possession of the debugger and provided to the computer program either on request or sent automatically (e.g. with the initial debug request). The computer program can then check that the provided key matches a key that it has or that is stored on the computing device (for example, a key burned onto the ROM of the computing device, in a similar way described above). If the two keys are identical, then the debugger is authorised to overcome the restriction and debug the sensitive areas freely.

The key may be any suitable data string, e.g. an alphanumeric string, a random number etc.

As shown in figures 1a to 1d, different levels within the program may require different levels of authorisation. For example, the key illustrated in figures 1c and 1d is sufficient to “unlock” program instructions contained in level 2 but is not sufficient to “unlock” the program instructions contained in level 1. By providing the third parties with appropriate keys to unlock some restricted areas of code but not others, manufacturers can vary the amount of debugging access that is permitted to a third party in dependence on how far they trust that third party and in dependence the development purposes of the third party, i.e. what parts of the computer program the third party reasonably needs to debug for its development work.

If the program is arranged into modular levels that are associated with different authorisation levels, the computer program may inform the debugger what level of authorisation is required, so that the debugger can point the computer program to the section of the computing device’s memory that is storing the appropriate key. Alternatively, the debugger may always point the computer program to a default block of memory that stores a group of keys together with their associated authorisation levels, so that the computing program knows which of the stored keys it should compare against its own key to achieve the appropriate level of authorisation.

Figure 2 illustrates the functional components of a computer program according to certain embodiments of the invention. This is for the purposes of example only and it should be understood that the invention encompasses any structure of computer program that is capable of performing as described herein.

The computer program is arranged so that a set of program instructions 205 are accessible by a debugger for the purposes of debugging. Some of these instructions are contained in one or more subsets of instructions to which access by a debugger is subject to restrictions. In figure 2, four of these subsets are shown (206). Each subset may be associated with a particular key (207). If the debugger is able to provide the correct key or if the correct key has been pre-stored in the memory of the computing device on which the computer program is loaded, then the debugger will be able to “unlock” the restriction and step-through even restricted parts of the code.

The computer program includes means for receiving 202 an indication that a debugger has requested to debug at least part of the set of program instructions 205. In figure 2, this request is shown as a control signal 201. This “request processing” component of the program determines that the control signal is indicative of a request from a debugger and passes this request to an “access checking” part of the program 203. This part of the program includes means for checking whether the debugger’s request includes a request to access a restricted subset of instructions and means for determining whether any restriction can be overcome.

Preferably the program instructions are stored in memory such that any program instructions to which access by a debugger is to be restricted are stored in sections of memory that are considered by the computer program to be “restricted” areas. The “access checking” part of the program can therefore determine that the debugger’s request includes instructions that are contained in a restricted subset in dependence on the area of memory in which the instructions are stored.

Each “restricted” area of memory may also be associated with a respective authorisation level, so that the “access checking” part of the program can also determine if the debugger is authorised to “unlock” the restricted sections of code in

dependence on the area of memory in which those particular restricted sections of code are stored. For example, code stored in some parts of memory may be associated with different authorisation keys from code stored in other parts of the memory.

Alternatively, the “access checking” part of the program may determine that the debugger’s request includes a restricted subset of instructions by accessing a list of “restricted” instructions stored in memory. For example, the computer program may be stored in memory together with a list of memory locations at which instructions contained in a restricted subset are stored. If instructions contained in a restricted subset are stored in a contiguous block of memory, then the restricted list may simply include the start and end locations of that contiguous block of memory to represent the restricted subset of instructions.

If the list of restricted subsets is stored as a list of memory locations then the “access checking” part of the program may be arranged to perform mapping operation to map the program instructions requested by the debugger (which may be in the form of instruction line numbers) to the memory locations at which those instructions are stored. Alternatively, the “access checking” part of the program may conveniently have access to a record of the restricted subsets that is in the form in which those instructions will be requested by the debugger (e.g. in the form of line numbers) so that no conversion or mapping operation need be performed.

If the computer program is arranged to include a list of programs that are subject to restricted access, that list may incorporate the idea of authorisation levels by including a key or a pointer to a key associated with the restricted subsets that can be “unlocked”. The pointer could just be a record of an authorisation level associated with each subset, with a respective key associated with each authorisation level being contained elsewhere in the program.

As a further alternative, rather than the computer program being arranged to include a record of those instructions that are part of a restricted subset as a separate list, the record may be stored together with each individual instruction, e.g. as a

“restricted” flag. Different flags may be used to represent different levels of authorisation.

In order to check whether the debugger is authorised to “unlock” the restricted parts of the code that it wants to access, the “access checking” part of the program is preferably arranged to access a part of ROM that stores a key. The “access checking” part of the program may be arranged to access the key responsive to a direction from the debugger (e.g. an opcode or memory address). The “access checking” part of the program may be arranged to request this direction from the debugger when it determines that some of the instructions that the debugger wants to access are contained in a restricted subset. Alternatively, the computer program may just inform the debugger that some of the instructions are subject to restricted access, to which the debugger automatically in responds with the direction to the appropriate key. A further option is that the debugger provides the direction with its original request to access the computer program.

The “access checking” part of the program is preferably arranged to check the section of ROM to which the debugger has referred it and, if it finds something stored at that memory location, to check the stored data against a key that forms part of the computer program. If the keys are identical, then the debugger is considered to be authorised to “unlock” the restricted parts of the program. If not, the debugger is still only permitted restricted access to the restricted parts of the program.

Preferably the “access checking” part of the computer program is capable of considering different levels of authorisation that may be required for different restricted subsets, e.g. by comparing the appropriate keys stored on ROM and contained in the computer program.

If the debugger is authorised to access the restricted subset, the debugger can be permitted to freely debug the program instructions it has requested via the “debug access” part of the program 204. If not, the “access checking” part of the program informs the “debug access” part of the program that the debugger should not be

permitted to pause the program at an instruction that is contained in the restricted subset.

The “debug access” part of the program is preferably arranged to act as an intermediary between the set of program instructions that the debugger wants to access and the debugger. So, for example, if the debugger wants to access a certain section of code the access controller calls that section of code from memory and causes it to be executed by the processor. Similarly, if the debugger wants to step-through a certain section of code, the “debug access” part of the program inserts breakpoints at relevant sections of the code (e.g. by replacing an instruction with a software interrupt, an opcode, an invalid instruction or a jump instruction back into the access controller’s code) and replaces the original instruction when execution of the program instructions is resumed.

The “debug access” part of the program may be arranged to inhibit the debugger from causing a suspension in the execution of a subset of instructions to which access by the debugger is restricted by not inserting breakpoints at instructions that are contained in the subset of instructions when requested to do so by the debugger. The debugger will therefore be able to have the restricted subset executed by the processor, but the debugger will be prevented from causing the processor to suspend execution of the requested program instructions while the restricted subset is being executed.

In certain other embodiments of the invention, the “debug access” part of the program is also arranged to act as an intermediary between the set of program instructions that the debugger wants to access and the debugger. However, in these embodiments the “debug access” part of the program is arranged to limit the memory locations which the debugger is permitted to access. This may be additional to or an alternative to inhibiting the debugger from causing the computer device to suspend execution of the set or program instructions at an instruction contained in the restricted set of program instructions.

Typically the processor of a computing includes one or more registers. Registers are generally storage locations internal to the processor that hold data on which the processor can operate directly. Some registers may be set aside for certain tasks, e.g. a program counter, stack and flags. The processor is also typically capable of accessing other memory, such as random access memory (RAM) located external to the processor. The “debug access” part of the program according to certain embodiments of the invention is preferably arranged to inhibit the debugger from accessing memory locations in this “other memory” that are used by the restricted set of program instructions. The “debug access” part of the program may be arranged to inhibit the debugger from accessing these restricted memory locations while the requested set of program instructions is being executed and/or if the execution of the restricted set of program instructions is stopped due to some error or fault (e.g. if the program crashes).

The “debug access” part of the program may inhibit the debugger from accessing memory locations used by the restricted set of programs by preventing the debugger from reading data contained in those memory locations or changing data already stored in those memory locations.

The process of controlling access by a debugger to a set of program instructions according to certain embodiments of the invention need not be performed in software. The invention could also be implemented in hardware. For example, certain embodiments of the invention may provide an access controller that is implemented as a dedicated hardware unit and which is adapted to control access by a debugger to a set of program instructions. The access controller could be a switching unit that restricts the debugger's access to memory or that connects/disconnects the debugger from a debug agent that will insert breakpoints in the program instructions responsive to a request from the debugger.

A computer program or access controller according to certain embodiments of the invention may suitably be implemented in a computing device, resulting in the process shown in figure 3.

In step 302 the program receives a control signal indicative of a request by a debugger to access a set of program instructions. In step 304 it is checked whether the requested instructions are in a restricted subset. If not, the debugger is allowed unrestricted access to those instructions (step 306). If the requested set of instructions does include instructions that are contained in a restricted subset, the process proceeds to step 308 where it is checked whether the restriction can be overcome with an appropriate authorisation. If not, the debugger is inhibited from causing execution of the requested set of instructions to pause at an instruction contained in a restricted subset (step 310). In this example, the debugger itself has been provided with a key, and this key is obtained by the computer program in step 312. In step 314 it is checked whether the authorisation is correct. If the authorisation is correct, the debugger is allowed unrestricted access to all the requested instructions (step 318). If not, the debugger is inhibited from causing execution of the requested set of instructions to pause at an instruction contained in a restricted subset (step 316).

A computer program or access controller according to certain other embodiments of the invention may also suitably be implemented in a computing device, resulting in the process shown in figure 4.

In step 402 the program receives a control signal indicative of a request by a debugger to access a set of program instructions. In step 404 it is checked whether the requested instructions are in a restricted subset. If not, the debugger is allowed unrestricted access to those instructions (step 406). If the requested set of instructions does include instructions that are contained in a restricted subset, the process proceeds to step 408 where it is checked whether the restriction can be overcome with an appropriate authorisation. If not, the debugger is inhibited from accessing memory locations used by the restricted set of program instructions (step 410). In this example, the debugger itself has been provided with a key, and this key is obtained by the computer program in step 412. In step 414 it is checked whether the authorisation is correct. If the authorisation is correct, the debugger is allowed unrestricted access to all the requested instructions and the memory locations used

by those instructions (step 418). If not, the debugger is inhibited from accessing memory locations used by the restricted set of program instructions (step 416).

A computer program according to certain embodiments of the invention may be an operating system, e.g. an operating system for a mobile phone. Different architectures for enabling such an operating system to be accessed by a debugger include an emulator debug architecture, a run-mode debug architecture and a stop-mode debug architecture.

The emulator has two main uses: developing the operating system and demonstrating the operating system. The code that runs on an emulator has a great deal in common with that which runs on a real device, but at the architecture-specific levels a "WIN32" CPU replaces the processor of the real device. This means that the emulator is effectively a port to a different processor. The emulator has processes and scheduling that are almost identical to those on a real device. Figure 5 illustrates the differences between the operating system running on a real mobile phone and on an emulator.

An emulator debug architecture is illustrated in figure 6. In this architecture, both the debugger integrated development environment (IDE) and the mobile phone operating system are hosted by a PC operating system. The IDE treats the mobile phone operating system like any other application of the host PCT operating system. While it is attached, the emulator debugger has complete control over the execution of the mobile phone operating system, other than those parts of the code that are subject to restricted access.

Although the emulator provides a powerful debugging environment, it is necessary to move to a hardware environment for development of hardware-specific device drivers or for software that is timing sensitive.

One common type of target debugger is the run-mode debugger illustrated in figure 7. In this implementation, the debugger user interface runs on a host PC and controls threads running on the target operating system through a proxy running remotely on

the target mobile phone. The host PC runs a host debugger, which connects to the mobile phone running the target operating system. The host debugger talks to a remote debug agent running on the mobile phone operating system, which performs actions on its behalf. Figure 8 shows the components of the run-mode debug architecture in more detail.

Another type of debugger architecture is the hardware-assisted debugger, such as the stop-mode debug architecture illustrated in figures 9 and 10. Figures 9 and 10 show an in-circuit emulator (ICE), which allows the target processor to be halted and subsequently controlled by the host at the instruction level. This provides support for features such as instruction level step, hardware breakpoint support, remote memory reads and writes and CPU register manipulation.

The computer program according to certain embodiments of the invention, which controls access by a debugger to a set of program instructions, may be implemented as part of the operating system itself or as part of the debug agent of figures 7 and 8 or the stop-mode debug API of figures 9 and 10.

Figure 11 shows a mobile phone in which the computer program according to certain embodiments of the invention may suitably be run. The mobile phone, shown generally at 1, includes a non-volatile memory 2 that stores instructions defining application programs (shown schematically at 3) and an operating system (shown schematically at 4). The mobile phone may be configured in such a way that the operating system controls the memory. The mobile phone has a CPU 5 which can execute the instructions stored in memory 2. The non-volatile memory also stores data (shown schematically at 6) defining a series of resource usage profiles. The mobile phone has a keypad 7 by which a user can control the operation of the phone. The mobile phone has an RF transceiver 8 coupled to an antenna 9, by means of which it can transmit and receive data according to a mobile phone radio protocol. The transceiver is coupled to the CPU. Data received by the transceiver is passed to the CPU and data can be passed from the CPU to the transceiver for transmission. The mobile phone has a display 10 for displaying data to a user, a loudspeaker 11 for producing sound (e.g. to reproduce audio data received through the transceiver 8).

and a microphone 12 for receiving sound (e.g. to capture audio data that is subsequently to be transmitted by the transceiver 7). The mobile phone is powered by a battery 13. The mobile phone may be provided with a working memory, which may be on the CPU or in RAM (random access memory) 15 coupled to the CPU. The mobile phone is also provided with a ROM (read only memory) 16 coupled to the CPU.

The embodiments of the invention described with reference to the drawings comprise computing devices and processes performed in computing devices. However, the invention also extends to computer programs, as is evident from the above description. The program may be in the form of source code, object code, a code intermediate source and object code such as code in partially compiled form, or in any other form suitable for use in the implementation of processes according to the invention. The computer program may be on or in a carrier. The carrier may be any entity or device capable of carrying the program.

For example, the carrier may comprise a storage medium, such as a ROM (e.g. a CD ROM or semi-conductor ROM), or a magnetic recording medium (e.g. a floppy disk or hard disk). Further, the carrier may be a transmissible carrier such as an electrical or optical signal which may be conveyed by electrical or optical cable or by radio or other means. When the program is embodied in a signal which may be conveyed by a cable or other device or means, the carrier may be constituted by such cable or other device or means.

Alternatively, the carrier may be an integrated circuit in which the program is embedded, the integrated circuit being adapted for performing, or for use in the performance of, the relevant processes.

Computing devices include desktop and laptop computers, personal digital assistants (PDAs), mobile telephones, smartphones, digital cameras and digital music players. They also include converged devices incorporating the functionality of one or more of the classes of device already mentioned, together with many other industrial and domestic electronic appliances.

The applicant hereby discloses in isolation each individual feature described herein and any combination of two or more such features, to the extent that such features or combinations are capable of being carried out based on the present specification as a whole in light of the common general knowledge of a person skilled in the art, irrespective of whether such features or combinations of features solve any problems disclosed herein, and without limitation to the scope of the claims. The applicant indicates that aspects of the present invention may consist of any such feature or combination of features. In view of the foregoing description it will be evident to a person skilled in the art that various modifications may be made within the scope of the invention.

Claims

1. An access controller for controlling access by a debugger to a set of program instructions stored on a computing device, the access controller being responsive to a control signal indicative of a request by the debugger to access at least part of the set of program instructions to:

determine that the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted; and

inhibit the debugger from causing the computing device to suspend execution of the requested set of program instructions at an instruction contained in the subset of instructions when the computing device is executing the requested set of program instructions responsive to the debugger's request.

2. An access controller as claimed in claim 1, wherein the access controller is arranged to determine whether the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted in dependence on an area of memory in which one or more of the requested set of program instructions are stored. ⊗

3. An access controller as claimed in claim 1 or 2, wherein the access controller is arranged to determine that the subset of instructions is a set of instructions to which access by the debugger is to be restricted in dependence on the provision of an authorisation associated with that subset of instructions.

4. An access controller as claimed in claim 3, wherein the authorisation is a key.

5. An access controller as claimed in claim 4, wherein the access controller comprises a copy of the key.

6. An access controller as claimed in claim 4 or 5, wherein a copy of the key is stored in a memory of the computing device and the access controller is arranged to access the memory of the computing device so as to obtain that copy of the key.

7. An access controller as claimed in claim 6, wherein the access controller is arranged to access the memory of the computing device responsive to a control signal indicative of a direction from the debugger to the copy of the key stored in the memory of the computing device.
8. An access controller as claimed in claim 6 or claim 7 as dependent on claim 5, wherein the access controller is arranged to inhibit the debugger from causing the computing device to suspend execution of the requested set of instructions at an instruction contained in the subset of instructions if the copy of the key comprised in the computing program does not correspond to the copy of the key stored in the memory of the computing device.
9. An access controller as claimed in claim 8, wherein the access controller is arranged to determine that the copy of the key comprised in the computing program does not correspond to the copy of the key stored in the memory of the computing device if the two keys are not identical.
10. An access controller as claimed in any preceding claim, wherein the access controller is arranged to inhibit the debugger from changing the instructions contained in the subset of instructions.
11. An access controller as claimed in any preceding claim, wherein the access controller is arranged to inhibit the debugger from changing a program state associated with the execution of the subset of instructions by the computing device.
12. A computer program comprising program instructions for controlling access by a debugger to a set of program instructions stored on a computing device, the program controller being responsive to a control signal indicative of a request by the debugger to access at least part of the set of program instructions to:
 - determine that the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted; and
 - inhibit the debugger from causing the computing device to suspend execution of the requested set of program instructions at an instruction contained in the subset

of instructions when the computing device is executing the requested set of program instructions responsive to the debugger's request.

13. A computer program comprising program instructions that are accessible by a debugger, the computer program being arranged such that one or more of the instructions form a subset of instructions to which access by a debugger is to be restricted by inhibiting the debugger from causing a computing device to suspend execution of the requested set of program instructions at an instruction contained in the subset of instructions when the computing device is executing the requested set of program instructions responsive to a request by the debugger.

14. A computer program as claimed in claim 13, wherein the computer program is arranged such that the subset of instructions to which access by a debugger is to be restricted is associated with an authorisation requirement, access to the subset of instructions being unrestricted if said authorisation requirement is fulfilled.

15. A computer program as claimed in claim 14, wherein the computer program being arranged such that the instructions form a plurality of subsets to which access by a debugger is to be restricted, each subset of instructions being associated with a respective authorisation requirement.

16. A computer program as claimed in any of claims 13 to 15, wherein the computer program is stored in a memory of a computing device, the subset of instructions to which access by the debugger is to be restricted being formed by the way in which the program instructions of the computer program are arranged in the memory.

17. A method for controlling access by a debugger to a set of program instructions stored on a computing device, the method comprising:

receiving a control signal indicative of a request by the debugger to access at least part of the set of program instructions;

determining that the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted; and

inhibiting the debugger from causing the computing device to suspend execution of the requested set of program instructions at an instruction contained in the subset of instructions when the computing device is executing the requested set of program instructions responsive to the debugger's request.

18. An access controller for inhibiting a debugger from accessing memory locations used by a set of program instructions stored on a computing device, the access controller being responsive to a control signal indicative of a request by the debugger to execute at least part of the set of program instructions to:

determine whether the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted;

cause the requested set of program instructions to be executed by a processor, the processor being capable of accessing registers and other memory; and

inhibit the debugger from accessing memory locations contained in the other memory that are used during execution of the requested set of program instructions by the restricted set of program instructions.

19. An access controller as claimed in claim 18, wherein the access controller is arranged to inhibit the debugger from accessing the memory locations used by the restricted set of program instructions while the requested set of program instructions is being executed.

20. An access controller as claimed in claim 18 or 19, wherein the access controller is arranged to inhibit the debugger from accessing the memory locations used by the restricted set of program instructions following a suspension in the execution of the requested set of program instructions by the processor.

21. An access controller as claimed in claim 20, wherein the access controller is arranged to inhibit the debugger from accessing the memory locations used by the restricted set of program instructions following a suspension in the execution of the requested set of program instructions caused by an error in the execution of those program instructions.

22. An access controller as claimed in any of claims 18 to 21, wherein the access controller is arranged to inhibit the debugger from accessing memory locations in the other memory that are storing instructions contained in the restricted set of program instructions.

23. An access controller as claimed in any of claims 18 to 22, wherein the access controller is arranged to inhibit the debugger from accessing memory locations in the other memory that are storing program data associated with the restricted set of program instructions.

24. An access controller as claimed in any of claims 18 to 23, wherein the other memory is external to the processor.

25. An access controller as claimed in any of claims 18 to 23, wherein the other memory is a random access memory.

26. A computer program comprising program instructions for inhibiting a debugger from accessing memory locations used by a set of program instructions stored on a computing device, the computer program being responsive to a control signal indicative of a request by the debugger to execute at least part of the set of program instructions to:

determine whether the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted;

cause the requested set of program instructions to be executed by a processor, the processor being capable of accessing registers and other memory; and

inhibit the debugger from accessing memory locations contained in the other memory that are used during execution of the requested set of program instructions by the restricted set of program instructions.

27. A method for inhibiting a debugger from accessing memory locations used by a set of program instructions stored on a computing device, the method comprising:

receiving a control signal indicative of a request by the debugger to access at least part of the set of program instructions;

determining whether the requested set of program instructions includes a subset of instructions to which access by the debugger is to be restricted;

executing the requested set of program instructions by means of a processor, the processor being capable of accessing registers and other memory; and

inhibiting the debugger from accessing memory locations contained in the other memory that are used during execution of the requested set of program instructions by the restricted set of program instructions.

28. A computer program as claimed in any of claims 12 to 16 or claim 26, wherein the computer program is an operating system.

29. A computing device comprising an access controller as claimed in any of claims 1 to 11 or claims 18 to 25.

30. A computing device programmed with a computer program as claimed in any of claims 12 to 16 or claim 26.

31. A computing device as claimed in claim 29 or 30, wherein the computing device comprises a memory that stores a key for authorising a debugger to access a subset of program instructions that are contained in the computer program and to which access by the debugger is to be restricted.

32. An access controller substantially as herein described with reference to the accompanying drawings.

33. A computer program substantially as described herein with reference to the accompanying drawings.

34. A method substantially as described herein with reference to the accompanying drawings.

35. A computing device substantially as described herein with reference to the accompanying drawings.

1/8

Figure 1a

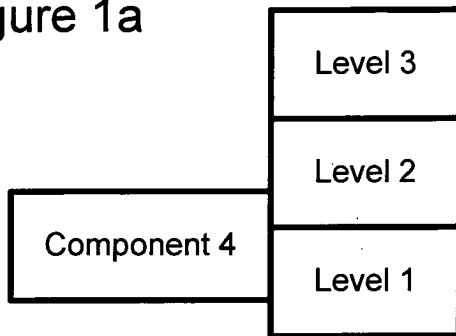


Figure 1b

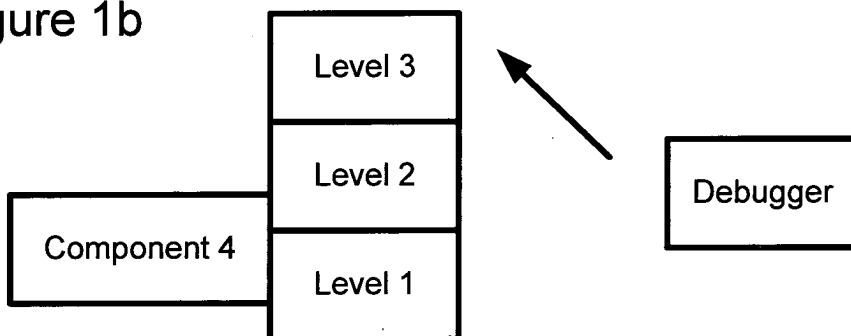


Figure 1c

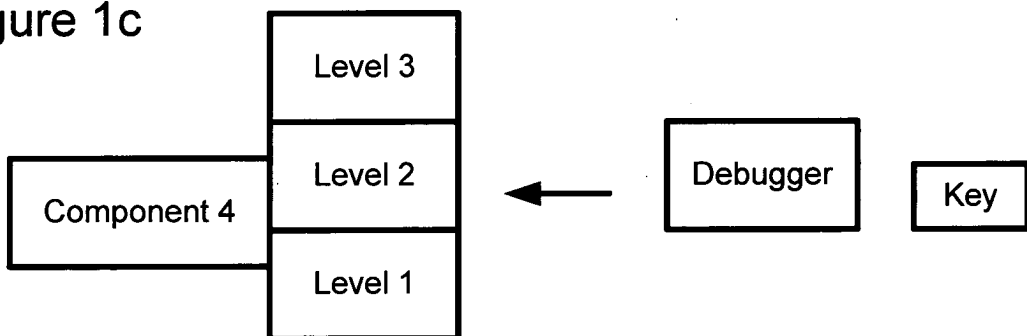
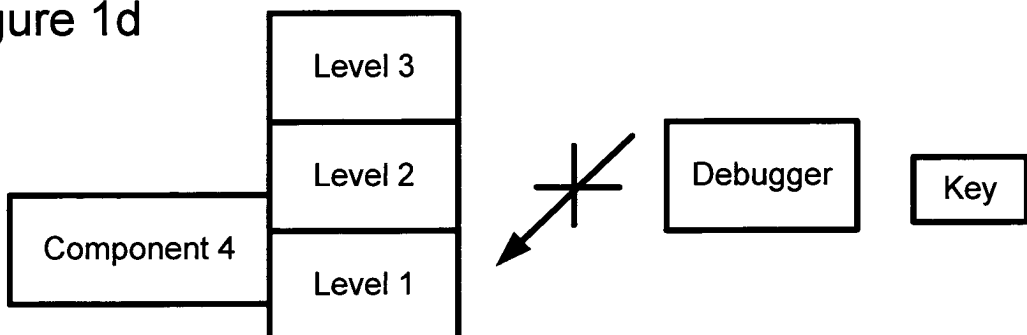


Figure 1d



2/8

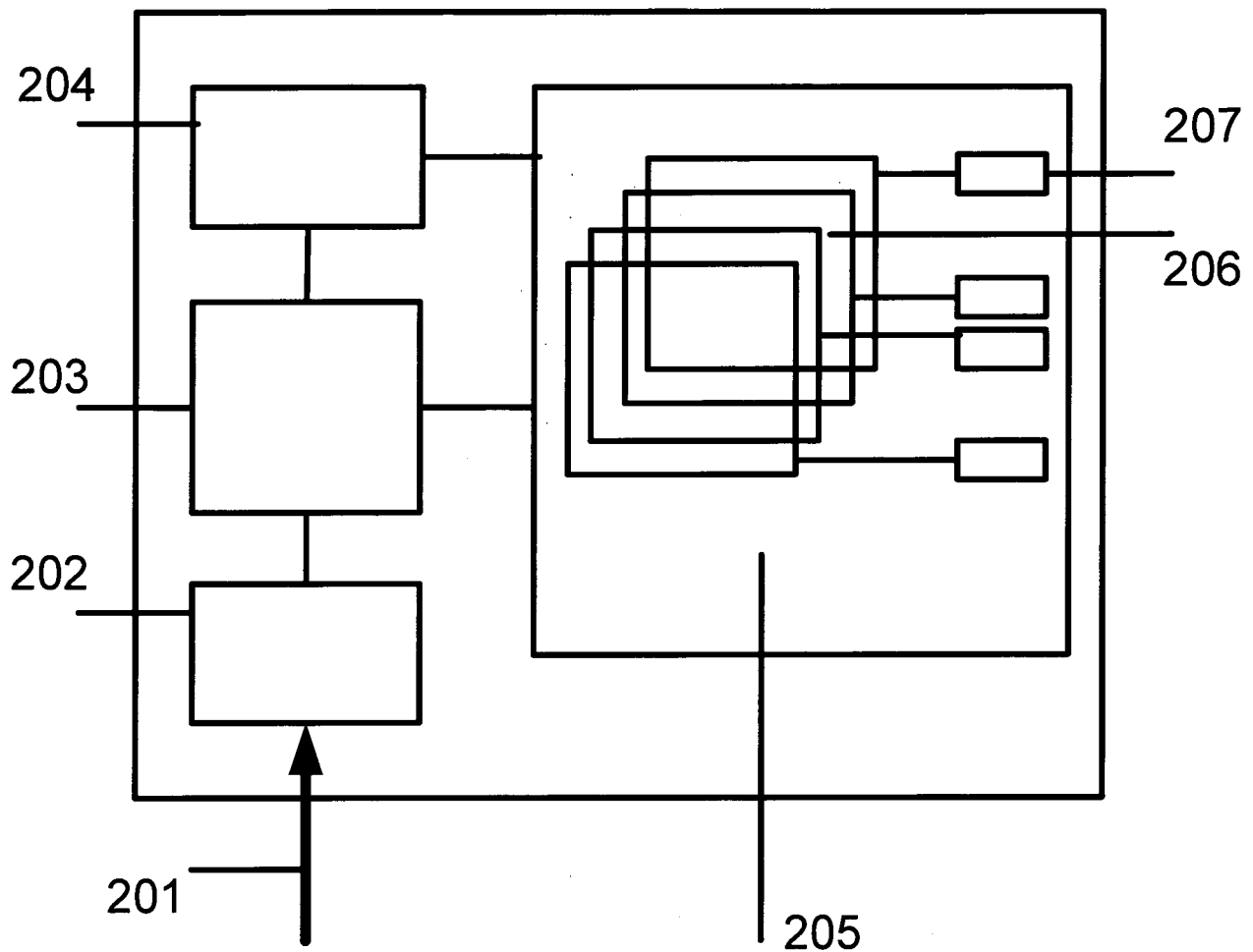


Figure 2

3/8

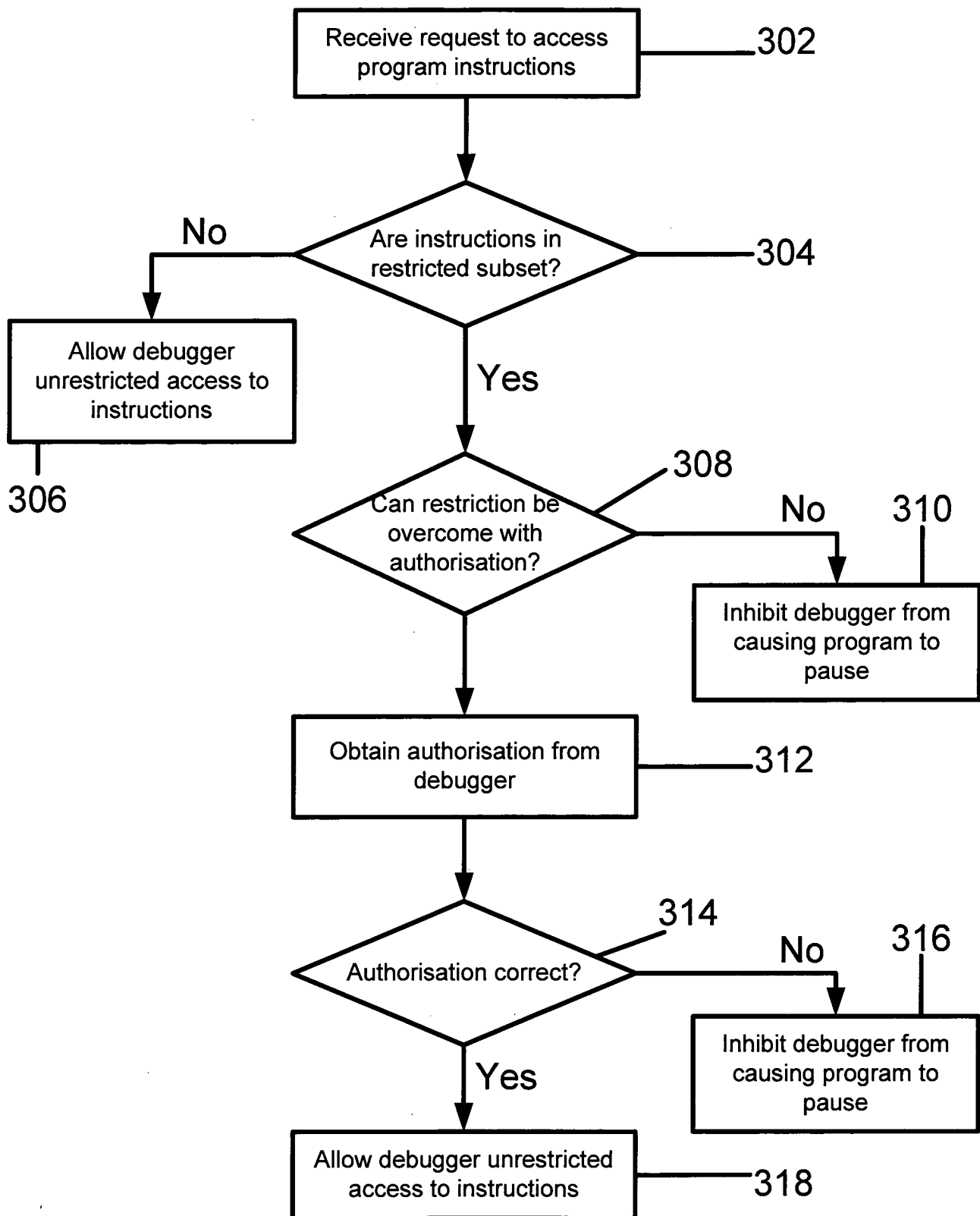


Figure 3

4/8

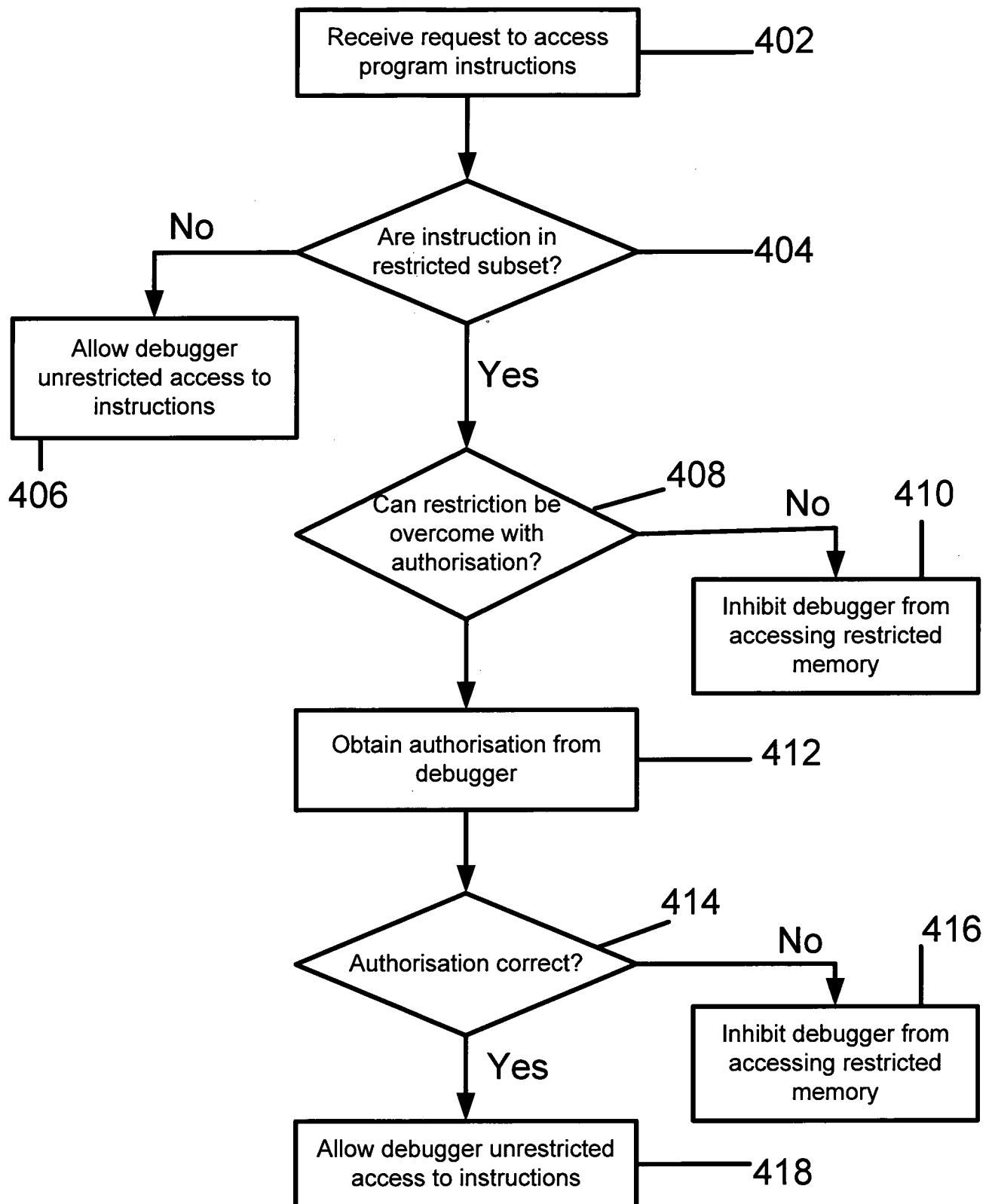


Figure 4

5/8

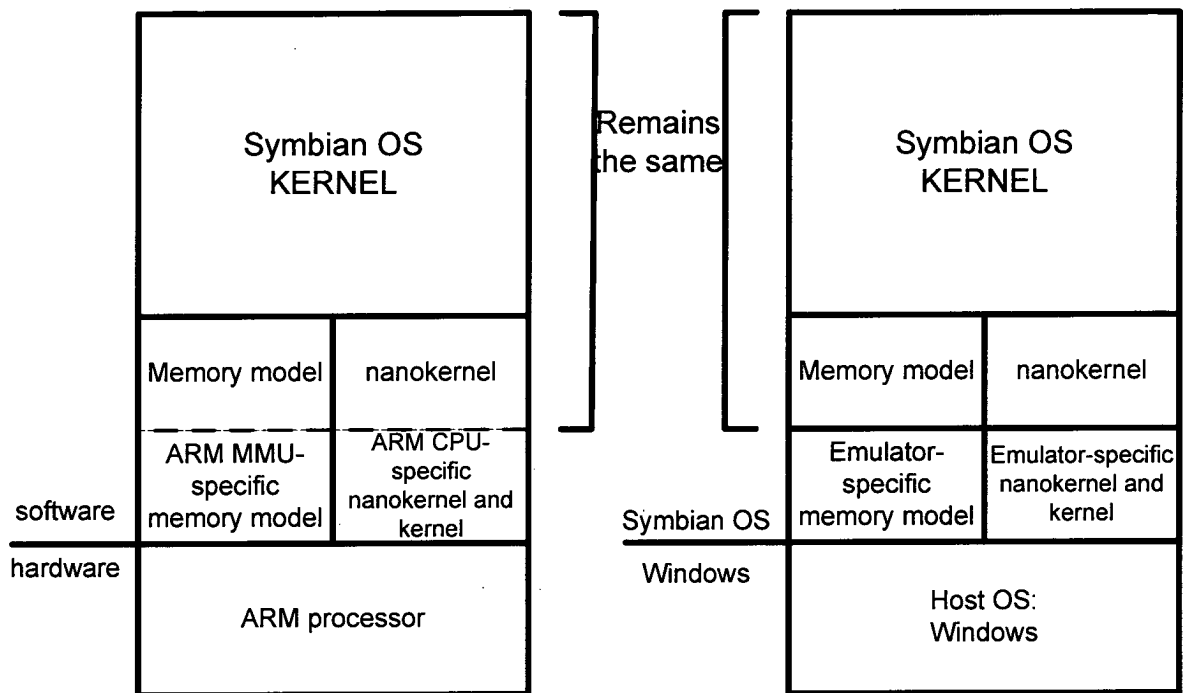


Figure 5

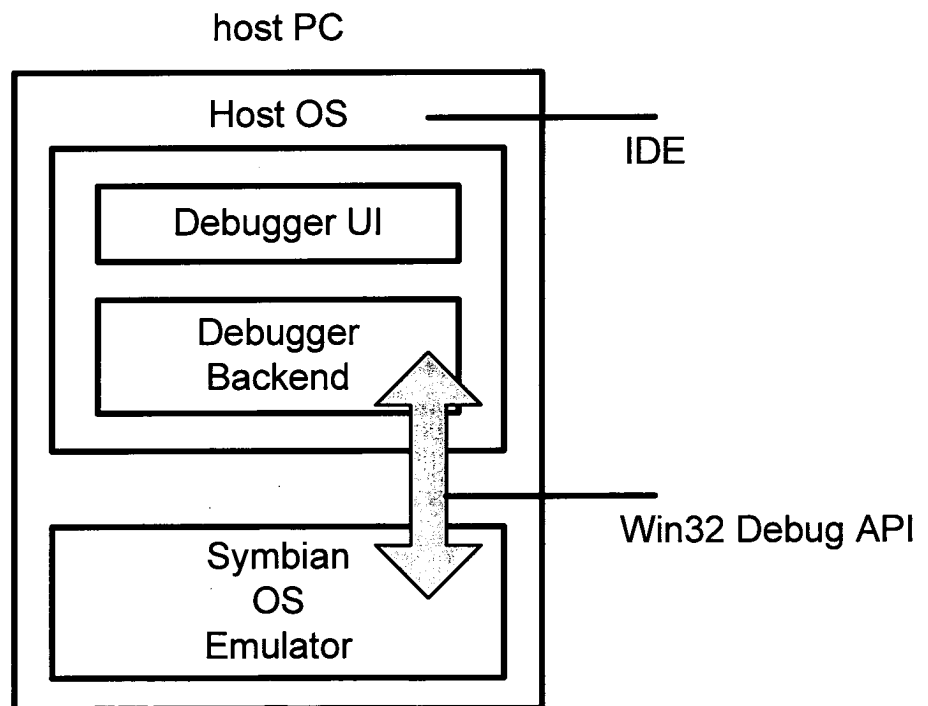


Figure 6

6/8

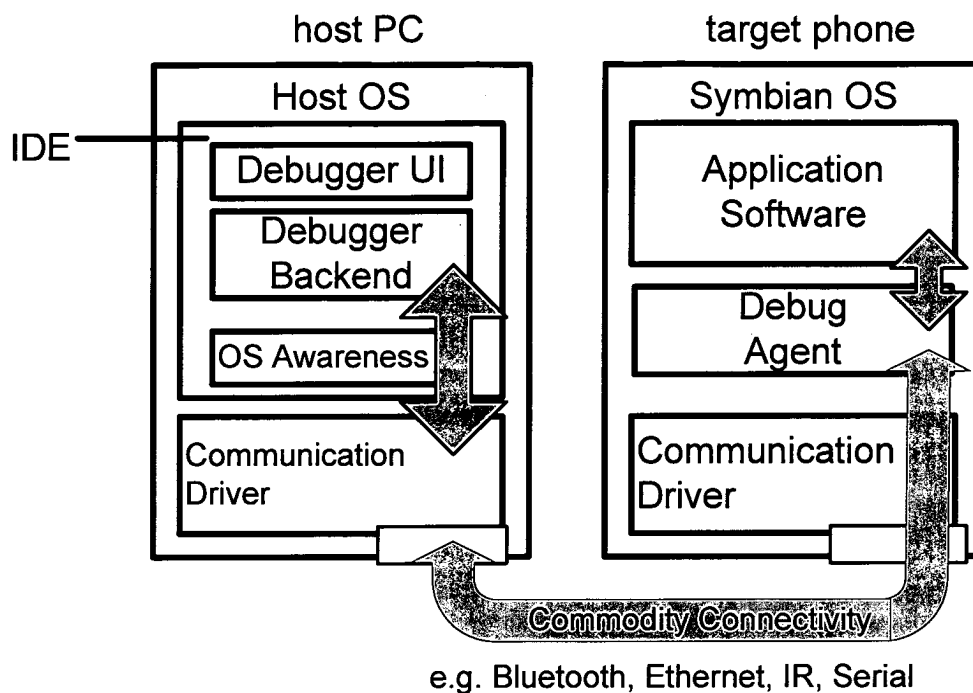


Figure 7

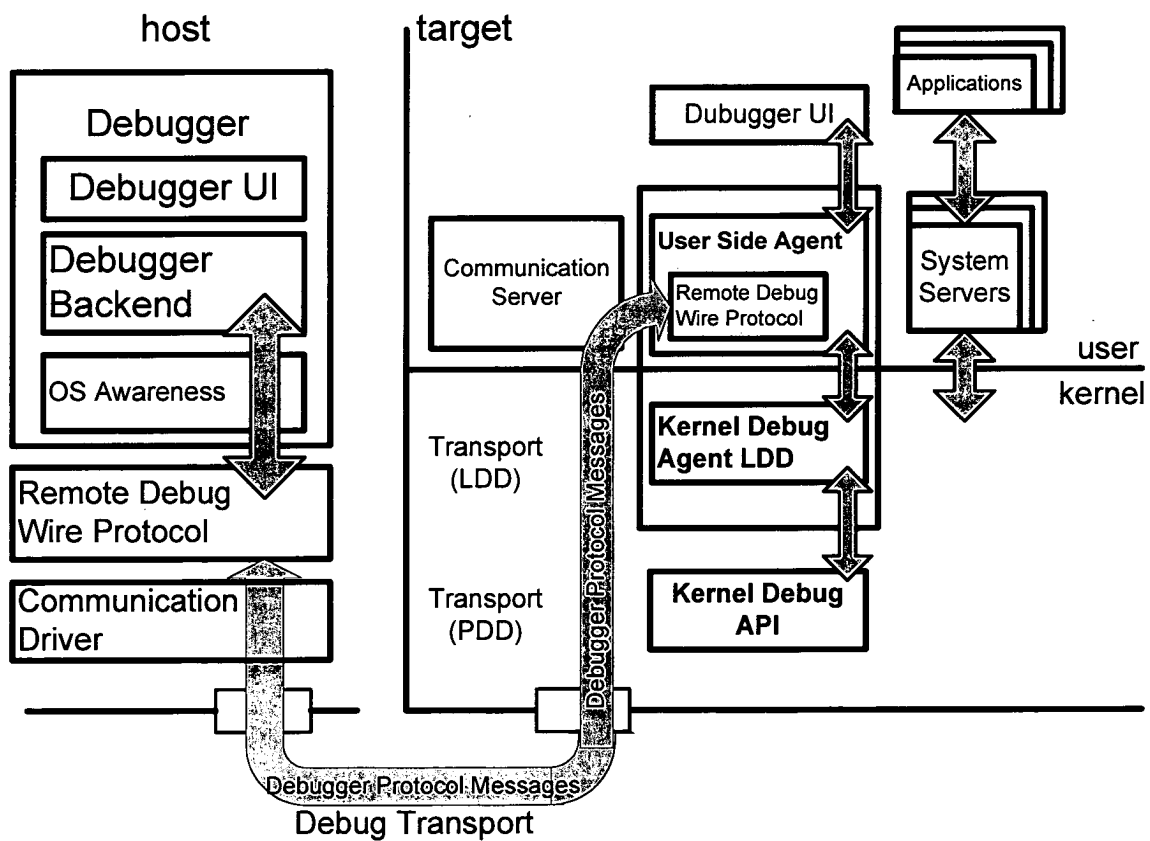


Figure 8

7/8

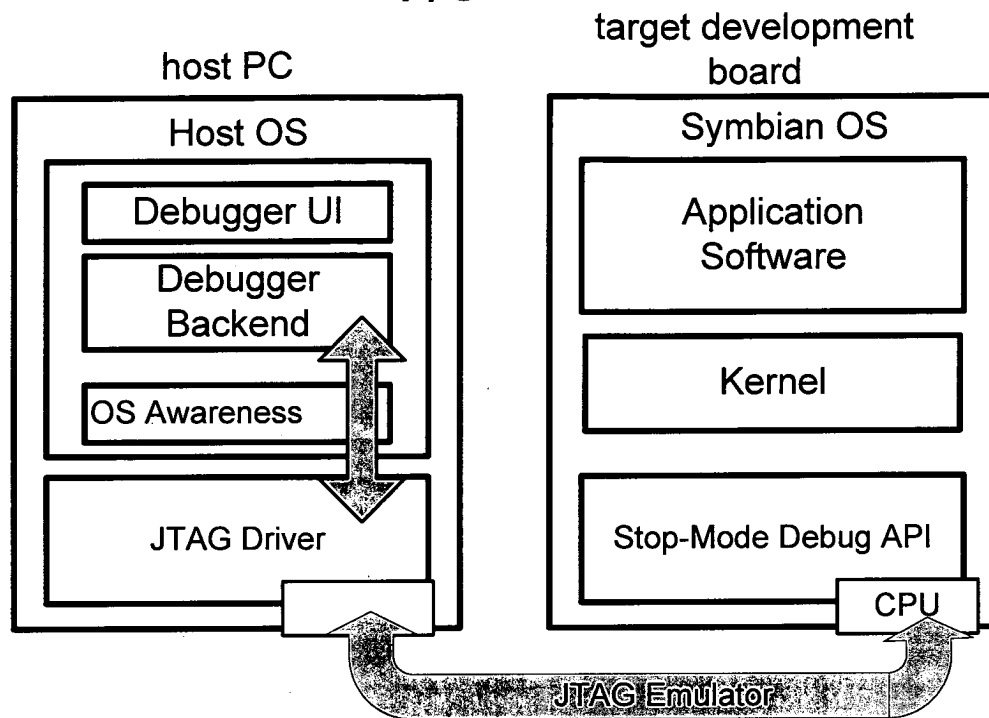


Figure 9

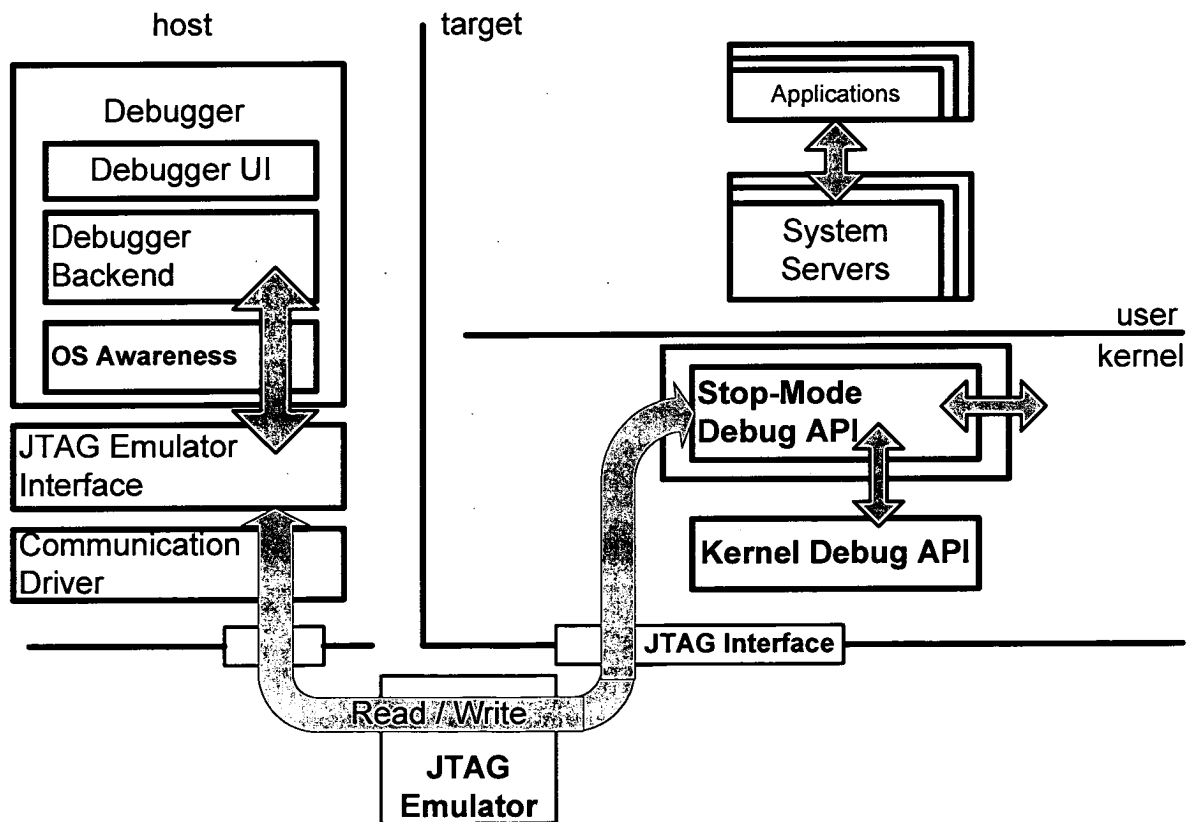


Figure 10

8/8

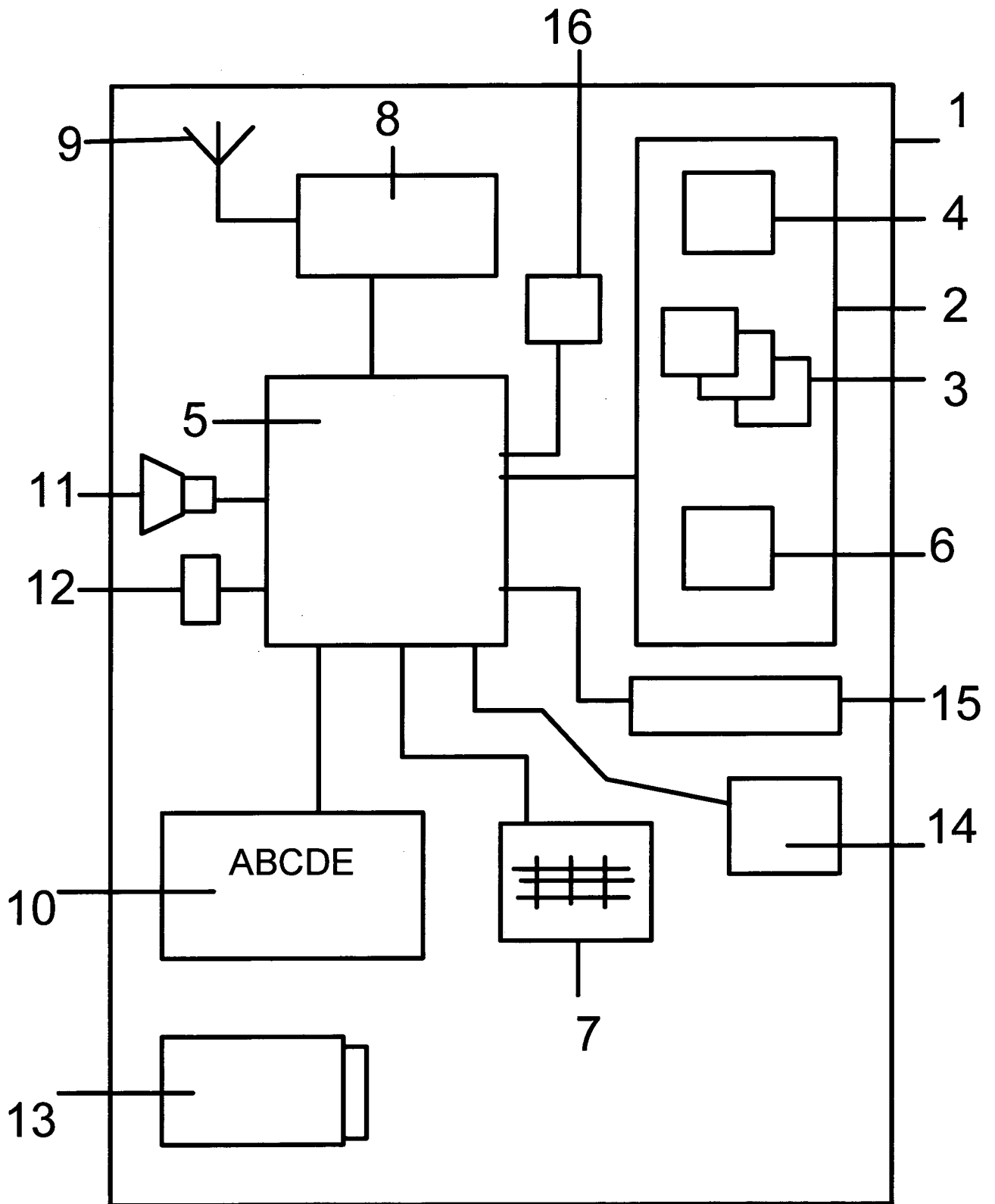


Figure 11