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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
12 August 2021 (12.08.2021)

WIPO | PCT

(10) International Publication Number
WO 2021/156596 A1

(51) International Patent Classification:
G05B 19/05 (2006.01)

(21) International Application Number:
PCT/GB2021/000010

(22) International Filing Date:
02 February 2021 (02.02.2021)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
2001687.9 07 February 2020 (07.02.2020) GB

(71) Applicant: **THE SECRETARY OF STATE FOR DEFENCE** [GB/GB]; DSTL, Porton Down, Salisbury, Wiltshire SP4 0JQ (GB).

(72) Inventor: **HOWE, Andrew, George**; DSTL IP Group, G02 Building 005, DSTL Porton Down, Salisbury. Wiltshire SP4 0JQ (GB).

(74) Agent: **FARNSWORTH, Alastair Graham**; DIPR Formalities Section, Poplar 2#2214, MOD Abbey Wood (South), Bristol BS34 8JH (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, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, IT, JO, JP, KE, KG, KH, KN, KP, KR, KW, 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, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

WO 2021/156596 A1

(54) **Title:** AN INDUSTRIAL PROGRAMMABLE LOGIC CONTROLLER

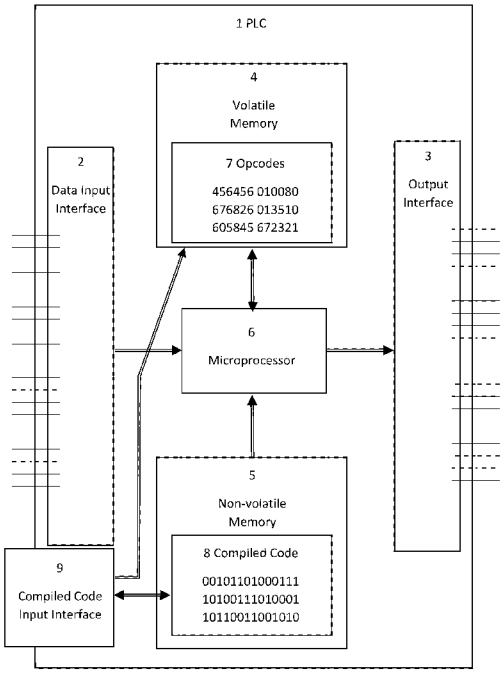


Figure 1

(57) **Abstract:** An industrial programmable logic controller (PLC) (1), comprises physical input interfaces (2, 9) and a physical output interface (3), volatile memory (4) for working data and non-volatile memory (5) for statically compiled code (8) and a microprocessor (6), wherein: the compiled code defines, or calls, an instruction set of an interpreted language, expressed in the compiled language, with opcodes defining operations, and the volatile memory comprises opcodes (7) defining software patching code to be executed, including to read input data via one of the physical input interfaces, to authenticate the input data as a software patch and to modify the opcodes such as to patch the PLC. This enables the PLC software to be patched, without needing to pause the operation of the PLC, and without enabling modification by an unauthorised person.

[Continued on next page]

(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))*

An industrial programmable logic controller

Programmable logic controllers (PLCs) are widely used across industry, and are a type of operational technology (OT). The term operational technology is intended to distinguish such infrastructure control systems from information technology (IT) such as laptops and smart phones and consumer internet connected devices. Whilst IT devices do control some types of hardware, such as office printers, IT generally is not focussed on the control of hardware and does not generally control industrial hardware, whereas OT is focussed on the control of industrial hardware and is used in an industrial environment for example on a factory floor to control industrial valves, heaters, conveyor belts, presses and pumps. Typical characteristics of a programmable logic controller are having a rugged housing, being non-portable in use, and being mounted in use on a suitable support such as a wall, near to industrial equipment or in a control room.

PLCs are generally programmed using a dialect or combination of a set of statically-compiled graphical and text-based programming languages that are defined by the International Electro-technical Commission (IEC) 61131-3 standard. The reverse is also possible in that logic may be uploaded to a host computer from the PLC for inspection.

Generally the program logic ("subroutines") to be executed by the PLC is created and cross-compiled on a host computer before being downloaded to the target PLC's non-volatile (flash) memory. Importantly, this process means that the pathways through the logic are defined at compile time (I.e. the program executes in a predictable manner, and does not change (unless re-programmed).

One problem with PLCs are that having been installed in an industrial environment, programmed appropriately and turned on, they are generally left and it is generally assumed that they are functioning optimally. Often PLCs are left to operate for several years without ever being turned off. This becomes a problem when after a period of time a problem is identified in how the industrial facility is functioning, and it is necessary to test whether changes to the PLC configuration could provide improvements to the operation of the industrial facility, since then it is necessary to stop all the machinery (E.g. the whole factory)

whilst testing of new software on the PLC is performed and whilst the performance of the upgraded PLC is checked and certified. A great deal of productivity would be lost and as a result it is common that users of industrial facilities are greatly deterred from attempting to implement and test improvements to the PLC functionality, or to apply security patches.

It is an object of the present invention to provide an industrial PLC in which its software can be reliably patched without needing to pause its operation.

According to a first aspect of the present invention there is provided a PLC as set out in claim 1. Further aspects of the invention are set out in the penultimate and final claims respectively.

This has the advantage that the software of the PLC can be reliably patched by a user, without needing to pause its operation, but without causing it to be it readily modifiable by an unauthorised person.

Regarding the term 'comprising a loop' generally the PLC is hard wired to perform the loop, so this generally does not need specified in the software to be uploaded onto the PLC, but instead the software may be configured based on the fact that when uploaded onto the PLC it will be run iteratively in a loop (barring an initial setup function which is usually a required function, although it can be empty).

The engine (the definition of an instruction set of an interpreted language) can be included in-line in the main function that is iteratively executed as a loop, or may be in a separate function which is called from that main function – the choice is generally immaterial and a matter of style.

The override instruction is an instruction to use the engine to read the opcodes (if present). Normally these would be located in predetermined location(s) for the engine to retrieve, but alternatively the engine could be adapted to scan the volatile memory for anything matching a predetermined format in order to find the opcodes. Opcodes can also be stored in non-volatile memory, but generally there is little point to doing this except perhaps to store a library of functions. Clearly at some opcodes would have to be in volatile memory once a software patch has been installed, since it is typically not possible for the engine to add them to the non-volatile memory (the non-volatile memory is changed only when software is uploaded onto it, which generally requires stopping the PLC from operating).

Typically there will be no opcodes present in the volatile memory until a software patch is installed, however as an alternative approach, some, or indeed substantially all of the software (i.e. apart from the engine and the instruction to run it to implement opcodes located in volatile memory) of the software could be arranged as opcodes in volatile memory (and/or conceivably in non-volatile memory).

The term software as separate from the software patching code and any software patch, means that software which is used to control the PLC to perform operations or functions required of the PLC e.g. to perform actions needed in the industrial environment of the PLC. The software is generally the software excluding the engine, opcodes and instruction to apply the engine to implement the opcodes. Industrial includes military and relates to environments where machinery is being controlled or monitored, i.e. as compared to consumer and home environments. The different usually is found in the degree of ruggedness of the controller, which generally has a strong casing typically fixed securely to equipment or to a wall.

The term, 'override instruction is positioned within the loop, such as to be able to override the operation of or, at least one output of, the software code' typically involves placing the override instruction at the end, so as to be substantially the last instruction executed, however the software may function in a manner where information from the previous iteration of the loop is further operated on rather than merely being used to output data to the output interface or output buffer, and in this case it may be adequate to position the override instruction arbitrarily within the loop. For example if the software maintains a log of recent temperatures to identify a running average of the last 10 measurements, and only takes action if the average exceeds a threshold, and it was desired to implement a software patch to disable this action, then in principle an instruction to execute the opcodes via the engine could be effective even if placed at the beginning of the loop, provided that the software patch modified the data from previous loops for example by replacing those temperature values with absolute zero, such that in practice it would be impossible for the average of the previous value, combined with 9 such modified values, to exceed the threshold. However in practice, to maximise the ease with which software patches could be implemented, generally the instruction will be proximal to the end, substantially at the end, or indeed at the end of the loop.

The term "compiled code" which describes what is added by the user initially, is code that '"has been" compiled', as opposed to 'source code that "can be" compiled'. By contrast the term "interpreted code" (which is what the user adds to the volatile memory), neither has been compiled, nor will be compiled in the use of the invention, but rather is code that "will be" interpreted in the course of using the device. Interpreted code is type of programming language for which most of its implementations execute instructions directly and freely, without any requirement to be compiled into machine-language instructions. Instead the interpreter (E.g. the engine) executes the program directly, translating each statement into a sequence of one or more subroutines, and then into another language (often machine code). Note that the term 'opcode' is a term of art. An opcode (abbreviated from operation code, also known as instruction syllable, instruction parcel or opstring) is the portion of a machine language instruction that specifies the operation to be performed.

The code of the engine may be within the loop block of code, or maybe external and called by it, i.e. as a function call, there is no real difference in terms of how the code functions. The engine, or its call, is arranged so that when executed it can override the functionality of the software (the software, or firmware, which controls the PLC to perform it's main, useful functions), and normally the engine or it's call, will be arranged to be at the end of the loop block of code. However since the loop block of code is executed repeatedly as a loop, depending on the design of the software, the engine may be able to override some or all of the functionality of the software even if it is positioned in the middle or beginning of the loop block of code. As an example if the software is adapted to check temperature values and turn on air conditioning if the air temperature has been measured three times sequentially as being 'high' and on the latest occasion as being 'very high'. If the user wanted to adjust this to ignore previous values, this could be patched even if the engine (or its call) executes at the beginning of the loop block of code, by implementing a software patch that adjusts all previous values of temperature to 'high'. However in general, to have the engine executed or called in the middle or beginning of the loop block of code requires careful design of the software, and this may limit the range of software patches that could be implemented. Therefore, in general it will usually provide for the greatest range of software patching to be possible, by positioning the engine or its call, at substantially the end of the loop block of code, i.e. following substantially all the important code of the software, or indeed preferably

at the end of the loop block of code (i.e. the last executed or called function of the loop block of code, prior to the loop repeating to the beginning of the loop block of code).

A preferred embodiment of the present invention will now be described by way of example only, with reference to the figures in which:

Figure 1 is a block diagram of a PLC according to an embodiment of the present invention.

With reference to figure 1, according to a preferred embodiment of the invention, a PLC 1 is provided with data input interface 2 to receive data, and compiled code input interface 9 which is generally a communications port, typically a USB port. This is used to load the PLC with compiled software, usually via an input buffer (not shown), which then resides as compiled code 8 in non-volatile memory 5. A microprocessor 6 executes the compiled code 8 to operate the PLC, which generally involves using volatile memory 4 as working memory for variables being worked on, and involves receiving data from the data input interface 2 often on an effectively continuous basis, processing it and providing outputs via the output interface 3, generally to control machinery. A typical example might be that the data inputs are from temperature and pressure sensors, and the outputs are to control a valve, for example in a pharmaceutical or food production warehouse or factory.

Whilst some PLCs are sold without any software loaded onto them, others are sold with compiled software pre-loaded onto the non-volatile memory. This is often the case when the PLC is installed by a contractor as part of the installation of various other equipment into the industrial facility.

Normally the compiled code 8 in the non-volatile memory contains, typically controls the microprocessor to read inputs from input 2 process it and output data to control industrial machinery via output 3, however in the present case the compiled code 8 in the non-volatile memory 5 defines an engine for interpreting code of an interpreted programming language, and the volatile memory 4 contains opcodes, for example starting at a predetermined location in the volatile memory, and arranged in a predetermined sequence or arrangement, such as for example being in a series forward from that position. Other arrangements are possible, for example being arranged starting at that position and extending backwards, or even perhaps arranged in predetermined non-contiguous locations, as long as the microprocessor is able to retrieve them based on the compiled code this is sufficient.

The engine defines how opcodes are related to functions. So for example opcodes maybe defined as numbers between 1 and 1 million, and the numbers 000123 through to 000127 correspond to the operation "while(", "for(", ")", "sum(", "product(" and so on.

Accordingly additional functionality can be encoded in opcodes defining operations in an interpreted language, and the compiled code includes instructions to get the opcodes, and execute their operations. Accordingly in principle any functionality that could have been achieved using the compiled code alone, can be implemented by making the microprocessor retrieve and interpret an equivalent series of opcodes.

At point of sale, the PLC as a product would contain an instruction in the compiled code in the non-volatile memory to execute the operations defined in the opcodes in the volatile memory, and the instruction would instruct the microprocessor to look for data in an input interface such as the input interface 2 which would constitute a software patch. The microprocessor would then process the software patch to modify the opcodes, thereby patching the PLC.

As part of this step the opcodes as interpreted via the engine in the compiled code, are executed by the microprocessor to verify the data as being a valid software patch. It is important that noise or arbitrary and irrelevant input signals are not interpreted as a software patch since this would lead to the opcodes being modified in random ways which clearly would not lead to improved performance. A verification check is implemented, which can be any type of verification desired, however as a minimum it involves verifying that the input data is either of a required format and or contains required data such that modification of the opcodes will not occur accidentally nor as a result of malicious hacking. The only way to initiate installation of the software patch by modification of the opcodes will be to use the format or data required, and this provides the verification effect of a password. A malicious 3rd party cannot obtain the password data by reading the compiled code from the PLC because the required password data/format is only identifiable from the opcodes. A malicious 3rd party cannot obtain the opcodes because the compiled code does not instruct the microprocessor to output them and because if the PLC is turned off to stop its operation the opcodes are lost because they are only contained in the volatile memory.

Accordingly, a legitimate user can implement a software patch without turning the PLC off, by inputting data in through a physical input interface with the required password data which the opcodes instruct the microprocessor to check for. A valid software patch will be effective because the compiled code contains the interpreting engine and instructs the microprocessor to check the volatile memory for opcodes and to execute them, and the opcodes instruct the microprocessor to check for and implement software updates but only if they have the required predetermined password data.

In order to set up the PLC in accordance with an embodiment of the invention, the user loads the PLC with compiled code which contains the interpreting engine and the instruction for the microprocessor to obtain and execute operations defined by any opcodes in the volatile memory. Whilst doing so, the user loads the PLC volatile memory with opcodes.

The operations defined by the opcodes of the software patching code, define instructions to:

- Read input data via one of the physical input interfaces; and to
- Authenticate the input data as being a software patch, by applying a verification criterion, and if input data passes the verification criterion; to
- Modify the opcodes in the volatile memory based on the software patch, such as to patch the PLC.

Any verification check may be used, and will be readily implemented by the skilled person. A typical example might be that the input data must contain a particular password, indeed typically must start with the password. By password is meant any predetermined identifiable input such as a number, text string, sequence of opcodes, or a sequence matching any predetermined and checkable criterion, which the software patching code is arranged to check.

Preferably the operations defined by the opcodes of the software patching code, define instructions to read input data via the second physical input interface. This has the advantage that the PLC can be patched more easily.

Optionally more than one verification criterion is provided, with a first verification criterion being provided to permit the user to modify the opcodes, and a second verification criterion being provided to permit the user to activate or control functionality provided by the existing opcodes.

In general terms, there is provided an industrial programmable logic controller (PLC) comprises first and second physical input interfaces and a physical output interface, volatile memory for working data and non-volatile memory for statically compiled code and a microprocessor, wherein: the compiled code defines an instruction set of an interpreted language, expressed in the compiled language, with opcodes defining operations, and the volatile memory comprises opcodes defining software patching code to be executed, including to read input data via one of the physical input interfaces, to authenticate the input data as a software patch and to modify the opcodes such as to patch the PLC.

Optionally the instruction, or call thereto, is positioned at substantially the end of the loop block of code. Further optionally the instruction, or call thereto, is the last instruction in the loop block of code.

Optionally the opcodes define instructions to control the PLC to inspect the values of at least one section of volatile memory, and to output that data or data derived therefrom.

The advantage of this is that one could determine whether the volatile memory has changed in ways which would not be expected during normal use, thereby identifying whether the PLC has been tampered with. This would be particularly easy if the section of volatile memory to be inspected was one that is ordinarily unchanged during normal use, or alternatively many sections, or indeed the entirety of the volatile memory could be outputted and checked remotely to identify any concerns.

Optionally the opcodes define instructions to control the PLC to output at least one predetermined variable used by the software.

This has the advantage that a user can be provided with a report on the functioning or operational conditions of the PLC, and for example might obtain such reporting data from each of many PLCs in a factory or other environment, thus enabling the user to identify any problems or concerns, however since the functionality is controlled by the opcodes, which

can be readily patched, the user is able to adjust what information gets reported. For example in summer the user might issue a software patch such that the opcodes control the PLC(s) to report if certain equipment is above a certain temperature, however in winter the user might issue a software patch such that the opcodes control the PLC(s) to report if the equipment is below a certain temperature. Optionally the predetermined criterion is stored, at least partly in volatile memory. Optionally the predetermined criterion is a password or passkey. Optionally the predetermined criterion is a cryptographic key or hash.

Storing the criterion as volatile memory, as a password/passkey or cryptographic key/hash has the benefit that is it more difficult for an unauthorised party to identify how to make the PLC authenticate a software patch.

According to a further aspect there is provided an industrial programmable logic controller (PLC), suitable for use in an industrial environment, suitable for controlling industrial machinery, and suitable for being programmed by an external computer using statically compiled code, comprising:

- A first physical input interface for receiving statically compiled code of a compiled language; A second physical input interface for reading bit level electrical signals from electromechanical device(s); A physical output interface for outputting bit level electrical signals to electromechanical device(s); and Volatile memory for being loaded with data, comprising an input buffer and an output buffer;
- Non-volatile memory loaded with statically compiled code in the compiled language, the statically compiled code comprising; - A setup block of code comprising an instruction to read input data from the second physical input interface; - A loop block of code comprising an instruction to operate on the input data by sequentially executing one or more compiled subroutine(s), and to write an output to the physical output interface;
- A microprocessor, arranged to execute the statically compiled code, using the volatile memory as working storage;

Characterised in that:

- The loop block of code comprises a definition of an instruction set of an interpreted language, expressed in the compiled language (an engine), comprising a list of opcodes defining respective operations;
- The volatile memory comprises opcodes having respective operations defined by the engine, and jointly defining software patching code;
- The loop block of code comprises an instruction to read the opcodes from the volatile memory, and, via the engine, to execute their operations; and
- The operations defined by the opcodes of the software patching code, define instructions to: - Read input data via one of the physical input interfaces; and to Authenticate the input data as being a software patch, by applying a verification criterion, and if input data passes the verification criterion; to - Modify the opcodes in the volatile memory based on the software patch, such as to patch the PLC.

CLAIMS

Claim 1. An industrial programmable logic controller (hereafter referred to as a PLC), suitable for use in an industrial environment for controlling industrial machinery, and suitable for being programmed by an external computer using statically compiled code, comprising:

- a first physical input interface for statically compiled code of a compiled language;
- a second physical input interface for electrical signals from electromechanical device(s);
- a physical output interface for electrical signals to electromechanical device(s);
- volatile memory for being loaded with data;
- non-volatile memory loaded with statically compiled code in a compiled language, comprising a loop arranged to be executed repeatedly that comprises or calls instructions, to:
  - input data from a physical input interface or an input buffer thereof;
  - perform an operation on stored data; and
  - output data to a physical output interface or an output buffer thereof;
- a microprocessor, arranged to execute the statically compiled code, using the volatile memory as working storage;

characterised in that:

- the statically compiled code comprises, or calls:
  - a definition of an instruction set of an interpreted language, expressed in the compiled language, comprising a list of opcodes defining respective operations (hereafter referred to as an engine); and
  - an override instruction to control the microprocessor to read opcodes (if present) from the volatile memory, and, via the engine, to execute their operations; and
- the PLC comprises software, arranged to control the microprocessor to measure via the input data, and/or influence via the output data, the physical environment of the PLC, either as:
  - opcodes to be executed by the override instruction via the engine; and/or
  - statically compiled code in non-volatile memory, wherein the override instruction is positioned within the loop, such as to be able to override the operation of or, at least one output of, the software code;
- the PLC comprises software patching code, in the form of either compiled code in non-volatile memory, and/or in the form of opcodes, arranged to control the microprocessor to:
  - read input data via a physical input interface or an associated input buffer;
  - authenticate the input data as being a software patch, by applying a predetermined verification criterion, and, if the input data passes the verification criterion; to
  - add opcodes, or modify existing opcodes, in the volatile memory based on the software patch, so as to patch the software.

Claim 2. An industrial programmable logic controller (PLC) according to claim 1, wherein the operations defined by the opcodes of the software patching code, define instructions to read input data via the second physical input interface.

Claim 3. An industrial programmable logic controller (PLC) according to claim 1 or claim 2, wherein more than one verification criterion is provided, with a first verification criterion being provided to permit the user to modify the opcodes, and a second verification criterion being provided to permit the user to activate or control functionality provided by the existing opcodes.

Claim 4. An industrial programmable logic controller (PLC) according to claim 1, 2 or 3, wherein the instruction, or call thereto, is positioned at substantially the end of the loop block of code.

Claim 5. An industrial programmable logic controller (PLC) according to claim 4, wherein the instruction, or call thereto, is the last instruction in the loop block of code.

Claim 6. An industrial programmable logic controller (PLC) according to any one of the preceding claims, wherein the opcodes define instructions to control the PLC to inspect the values of at least one section of volatile memory, and to output that data or data derived therefrom.

Claim 7. An industrial programmable logic controller (PLC) according to any one of the preceding claims, wherein the opcodes define instructions to control the PLC to output at least one predetermined variable used by the software.

Claim 8. An industrial programmable logic controller (PLC) according to any one of the preceding claims, wherein the predetermined criterion is stored, at least partly in volatile memory.

Claim 9. An industrial programmable logic controller (PLC) according to any one of the preceding claims, wherein the predetermined criterion is a password or passkey.

Claim 10. An industrial programmable logic controller (PLC) according to any one of the preceding claims, wherein the predetermined criterion is a cryptographic key or hash.

Claim 11. A computer program suitable for an industrial programmable logic controller (PLC) of the type as set out in any one of claims 1 to 10, comprising:

- statically compiled code in a compiled language, for being loaded onto non-volatile memory, comprising a loop arranged to be executed repeatedly that comprises or calls instructions, to:
  - input data from a physical input interface or an associated input buffer;
  - perform an operation on stored data; and
  - output data to the physical output interface or an associated output buffer;
- statically compiled code that comprises, or calls:
  - a definition of an instruction set of an interpreted language, expressed in a compiled language, comprising a list of opcodes defining respective operations (hereafter referred to as an engine); and
  - an override instruction to control the microprocessor to read opcodes from the volatile memory, and, via the engine, to execute their operations; and
- software, arranged to control a microprocessor to measure via input data, and/or influence via output data, a physical environment of a PLC, either as:
  - opcodes arranged to be executed by the override instruction via the engine; and/or
  - statically compiled code for non-volatile memory, wherein the override instruction is positioned within the loop, such as to be able to override the operation of or, at least one output of, the software code;
- software patching code, in the form of either compiled code for non-volatile memory, and/or in the form of opcodes, arranged to control the microprocessor to:
  - read input data via a physical input interface or an associated input buffer;
  - authenticate the input data as being a software patch, by applying a predetermined verification criterion, and, if the input data passes the verification criterion; to
  - add opcodes, or modify existing opcodes, in the volatile memory based on the software patch, so as to patch the software.

Claim 12. A method of patching a computer program of an industrial programmable logic controller (PLC), comprising the steps of:

- Providing a PLC as set out in any one of claims 1 to 10;
- Providing the PLC with data via a physical input interface of the PLC, wherein the data comprises opcodes defining a software patch for the PLC, and wherein data is adapted to pass the verification criterion of the PLC.

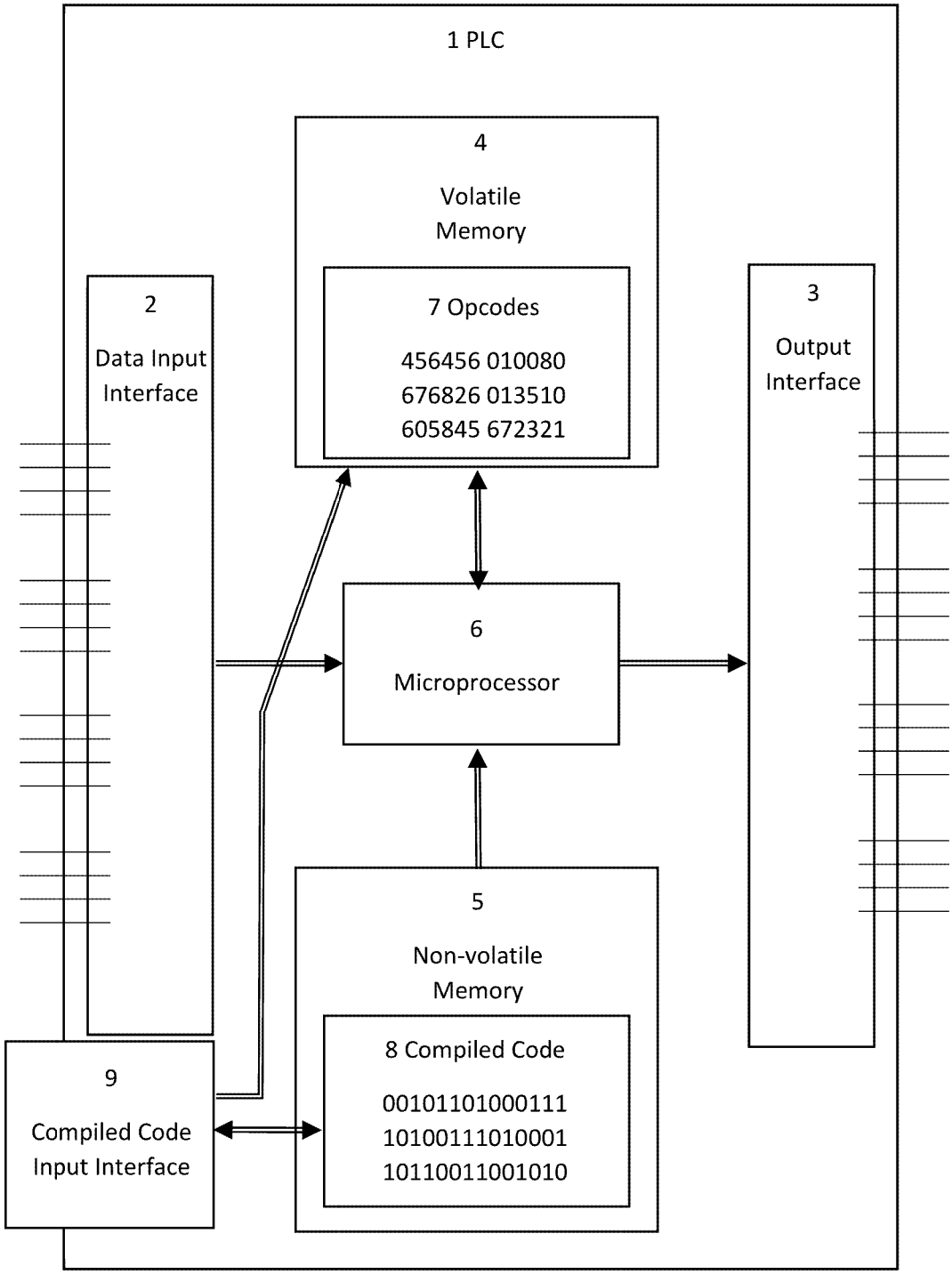

1 PLC
4
Volatile
Memory
7 Opcodes
456456 010080
676826 013510
605845 672321
2
Data Input
Interface
3
Output
Interface
6
Microprocessor
5
Non-volatile
Memory
8 Compiled Code
00101101000111
10100111010001
10110011001010
9
Compiled Code
Input Interface

Figure 1

INTERNATIONAL SEARCH REPORT

International application No
PCT/GB2021/000010

A. CLASSIFICATION OF SUBJECT MATTER
INV. G05B19/05
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)
G05B

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 8 286 238 B2 (DURHAM DAVID M [US]; KHOSRAVI HORMUZD M [US] ET AL.) 9 October 2012 (2012-10-09) the whole document	1-12
A	ANASTASIS KELIRIS ET AL: "ICSREF: A Framework for Automated Reverse Engineering of Industrial Control Systems Binaries", ARXIV.ORG, CORNELL UNIVERSITY LIBRARY, 201 OLIN LIBRARY CORNELL UNIVERSITY ITHACA, NY 14853, 9 December 2018 (2018-12-09), XP080990831, the whole document	1-12
A	US 2006/277539 A1 (AMARASINGHE SAMAN P [US] ET AL) 7 December 2006 (2006-12-07) the whole document	1-12

☐ 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

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
2 April 2021	14/04/2021

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Marrone, Fabrizio

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/GB2021/000010

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 8286238 B2	09-10-2012	CN 101154187 A	02-04-2008
		EP 1909171 A1	09-04-2008
		EP 2287724 A2	23-02-2011
		KR 20080029949 A	03-04-2008
		US 2008083030 A1	03-04-2008
US 2006277539 A1	07-12-2006	CN 101278260 A	01-10-2008
		EP 1891520 A2	27-02-2008
		US 2006277539 A1	07-12-2006
		US 2011185433 A1	28-07-2011
		WO 2006133222 A2	14-12-2006