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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
5 January 2017 (05.01.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/001967 A1

(51) International Patent Classification:
G06F 12/02 (2006.01)

(21) International Application Number:
PCT/IB2016/053675

(22) International Filing Date:
21 June 2016 (21.06.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
14/755,748 30 June 2015 (30.06.2015) US
14/941,558 14 November 2015 (14.11.2015) US

(71) Applicant: **INTERNATIONAL BUSINESS MACHINES CORPORATION** [US/US]; New Orchard Road, Armonk, NY 10504 (US).

(71) Applicants *(for MG only)*: **IBM UNITED KINGDOM LIMITED** [GB/GB]; PO Box 41, North Harbour, Portsmouth, Hampshire PO6 3AU (GB). **IBM (CHINA) INVESTMENT COMPANY LIMITED** [CN/CN]; 25/F, Pangu Plaza, No.27, Central North 4th Ring Road, Chaoyang District, Beijing 100101 (CN).

(72) Inventors: **FRAZIER, Giles, Roger**; IBM Corporation, Mail Drop 9024G015, 11400 Burnet Rd, Austin, TX 78758-3493 (US). **GSCHWIND, Michael Karl**; IBM Corporation, 1101 Kitchawan Rd, Route 134 / PO Box 218, Yorktown Heights, NY 10598 (US).

(74) Agent: **PYECROFT, Justine**; IBM United Kingdom Limited, Intellectual Property Law, Hursley Park, Winchester, Hampshire SO21 2JN (GB).

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

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

Published:

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

(54) Title: GARBAGE COLLECTION ABSENT USE OF SPECIAL INSTRUCTIONS

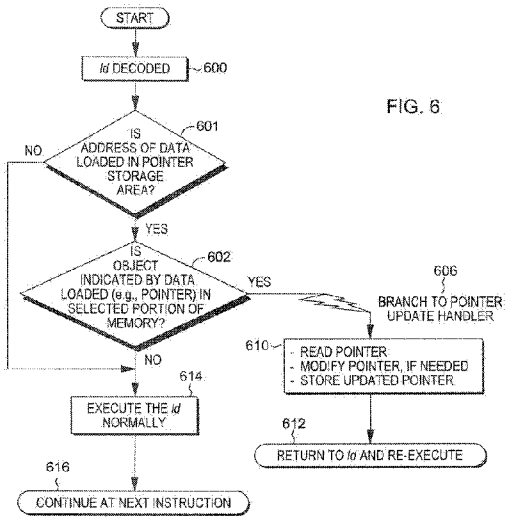

FIG. 6

(57) Abstract: Garbage collection processing is facilitated. Based on execution of a load instruction and determining that an address of an object pointer to be loaded is located in a pointer storage area and the object pointer indicates a location within a selected portion of memory undergoing garbage collection, processing control is obtained by a handler executing within a processor of the computing environment. The handler obtains the object pointer from the pointer storage area, and determines whether the object pointer is to be modified. If the object pointer is to be modified, the handler modifies the object pointer. The handler may then store the modified object pointer in a selected location.

WO 2017/001967 A1

GARBAGE COLLECTION ABSENT USE OF SPECIAL INSTRUCTIONS

BACKGROUND

[0001] One or more aspects relate, in general, to processing within a computing environment, and in particular, to garbage collection processing within the computing environment.

[0002] Garbage collection is an automatic memory management process that identifies objects in memory that are no longer being referenced and frees those objects. As memory objects of varying sizes are allocated and later freed, the memory in which they are stored becomes increasingly fragmented. Eventually, very few large free areas of memory exist, and it becomes difficult to store additional objects without increasing the memory size. When this occurs, a process within garbage collection, referred to as compaction, is employed in order to consolidate the allocated objects into one large area, leaving another large area of free space available for new objects. During consolidation, the memory objects that are still being referenced are moved from one area of memory to another area of memory.

[0003] Conventionally, when garbage collection is performed on an object storage area, applications using the object storage area are required to pause execution. One reason for this is to determine whether the pointers to the objects used by the applications to access the objects are still valid, since the objects may have moved. These pauses, occasionally several seconds long, prevent the applications from being used for time-sensitive tasks, such as transaction processing, real-time games, or mechanical control. Thus, a need exists for an optimized garbage collection process.

SUMMARY

[0004] In accordance with one or more embodiments of the present invention, an optimized garbage collection process is provided that enables applications to continue executing during the garbage collection process (without being paused due to garbage collection) when those applications are not accessing objects in an area of memory undergoing garbage collection, and

enables applications accessingobjects in an area of memory undergoing garbage collection to immediately resume processing after a very brief delay. This is enabled by providing an efficient mechanism to recognize when a pointer to the object storage area that is being collected is accessed, to obtain the pointer to an object in the area of memory being garbage collected, and to modify the pointer. Advantageously, this is performed without using special instructions and/or requiring program modifications to execute special instructions. Further, performance within a computing environment, including application performance and/or processor performance is improved.

[0005] According to one, there is provided a computer program product for facilitating garbage collection within a computing environment. The computer program product comprises a storage medium readable by a processing circuit and storing instructions for execution by the processing circuit for performing a method. The method includes, for instance, obtaining processing control by a handler executing within a processor of the computing environment. The obtaining processing control being based on execution of a load instruction and a determination that an address of an object pointer to be loaded is located in a pointer storage area and the object pointer indicates a location within a selected portion of memory undergoing garbage collection. Based on obtaining processing control by the handler, obtaining by the handler from the pointer storage area the object pointer, the object pointer indicating a location of an object pointed to by the object pointer. Determining by the handler whether the object pointer is to be modified, and modifying by the handler, based on determining the object pointer is to be modified, the object pointer to provide a modified object pointer. Based on modifying the object pointer, the modified object pointer is stored in a selected location.

[0006] Advantageously, this allows applications using objects in an area of memory not undergoing garbage collection to continue processing during garbage collection without interruption, and allows applications using objects in an area of memory undergoing garbage collection to continue processing after a very short unnoticeable delay, thus improving performance. Further, it does not require the use of special instructions or modifications of applications to use the special instructions.

[0007] In one embodiment, the obtaining processing control is via an interrupt issued by processor hardware. The interrupt is issued based on execution of the load instruction and the determination that the address of the object pointer to be loaded is located in the pointer storage area and the object pointer indicates the location within the selected portion of memory undergoing garbage collection. In one particular example, the interrupt is a lightweight interrupt (i.e., not involving the operating system) that gives control directly to an application-level handler. Advantageously, one or more of these embodiments enable immediate processing by the handler, and enable the application that accessed the pointer to continue processing immediately after the interrupt is processed without incurring the delay of a supervisor-level interrupt handler.

[0008] In one further embodiment, the selected portion of memory undergoing garbage collection is part of an object area that also includes one or more other objects not undergoing garbage collection, and advantageously, one or more applications accessing the object area not undergoing garbage collection continue to process during garbage collection. For instance, they continue to process without interruption. Further, the application that accessed the object pointer that indicates an object in the selected portion of memory undergoing garbage collection immediately resumes processing after a very brief delay during the time the handler (e.g., lightweight, application-level handler) processes the pointer. This enables applications to be used for time-sensitive processing because no application is delayed for a time period that is significant enough to be noticeable.

[0009] In one embodiment, the address of the object pointer (i.e., the address specifying a location of the object pointer) is determined from execution of the load instruction, and the address of the object pointer is compared with data related to the pointer storage area to determine whether the address of the object pointer is located within the pointer storage area. Based on the comparing determining that the address of the object pointer is located within the pointer storage area, the object pointer (i.e., an address specifying a location of an object) is compared with information relating to the selected portion of memory to determine that the object pointer indicates the location within the selected portion of memory. Control to the handler is provided based on determining the address of the object pointer is located within the pointer storage area and the object pointer indicates the location within the selected portion of

memory. Obtaining the object pointer from the pointer storage area advantageously eliminates the need for using special instructions to obtain the object pointer; thus, eliminating the need to define special instructions dedicated to garbage collection and to modify existing programs to use the instructions to obtain object pointers.

[0010] In embodiments, the pointer storage area includes a contiguous area of memory that includes a plurality of object pointers. As one example, the pointer storage area is indicated by one of a register or a location within memory.

[0011] Further, in embodiments, the selected portion of memory is indicated by one of a register or a location within memory, and in one embodiment, the register includes a base address of the selected portion of memory and a size of the selected portion of memory.

[0012] Yet further, in one example, the selected location in which the modified object pointer is stored is a location within the pointer storage area or a location specified by the load instruction.

[0013] As one example, the load instruction includes one or more operation code fields to specify a load operation, a result field to be used to obtain the object pointer, and one or more other fields to be used in the load operation.

[0014] In yet a further aspect, a computer program product for facilitating processing within a computing environment is provided. The computer program product includes a computer readable storage medium readable by a processing circuit and storing instructions for execution by the processing circuit for performing a method. The method includes, for instance, obtaining processing control by a handler executing within a processor of the computing environment. The obtaining processing control being based on a determination that an object pointer to be accessed indicates a location within a specified memory area. Based on obtaining processing control by the handler, the handler takes action. The taking action includes at least one of: providing an alert; preventing access to the specified memory area; or modifying the object pointer and storing the modified object pointer. This advantageously allows the handler

to take action whenever a pointer is to access a specified memory area regardless of the reason for the access.

[0015] According to another aspect, there is provided a computer system for facilitating garbage collection within a computing environment, said computer system comprising: a memory; and a processor in communications with the memory, wherein the computer system is configured to perform a method, said method comprising: obtaining processing control by a handler executing within a processor of the computing environment, the obtaining processing control being based on execution of a load instruction and a determination that an address of an object pointer to be loaded is located in a pointer storage area and the object pointer indicates a location within a selected portion of memory undergoing garbage collection; based on obtaining processing control by the handler, obtaining by the handler from the pointer storage area the object pointer, the object pointer indicating a location of an object pointed to by the object pointer; determining by the handler whether the object pointer is to be modified; modifying by the handler, based on determining the object pointer is to be modified, the object pointer to provide a modified object pointer; and storing, based on modifying the object pointer, the modified object pointer in a selected location.

[0016] According to another aspect, there is provided a computer-implemented method of facilitating garbage collection within a computing environment, said computer-implemented method comprising: obtaining processing control by a handler executing within a processor of the computing environment, the obtaining processing control being based on execution of a load instruction and a determination that an address of an object pointer to be loaded is located in a pointer storage area and the object pointer indicates a location within a selected portion of memory undergoing garbage collection; based on obtaining processing control by the handler, obtaining by the handler from the pointer storage area the object pointer, the object pointer indicating a location of an object pointed to by the object pointer; determining by the handler whether the object pointer is to be modified; modifying by the handler, based on determining the object pointer is to be modified, the object pointer to provide a modified object pointer; and storing, based on modifying the object pointer, the modified object pointer in a selected location.

[0017] Computer-implemented methods and systems relating to one or more aspects, as well as other computer program products, may also be described and claimed herein. Further, services relating to one or more aspects are also described and may be claimed herein.

[0018] Additional features and advantages are realized through the techniques described herein. Other embodiments and aspects are described in detail herein and may be considered a part of the claimed aspects.

BRIEF DESCRIPTION OF THE DRAWINGS

[0019] Preferred embodiments of the present invention, will now be described, by way of example only, and with reference to the following drawings:

Figure 1 depicts one example of a computing environment to incorporate and use one or more embodiments;

Figure 2A depicts another example of a computing environment to incorporate and use one or more embodiments;

Figure 2B depicts further details of the memory of FIG. 2A in accordance with an embodiment of the present invention;

Figure 3 depicts one example of a load doubleword instruction, in accordance with one or more embodiments;

Figure 4 depicts further details of memory for which garbage collection is to be performed, in accordance with one or more embodiments;

Figure 5A depicts one example of a pointer storage area register, in accordance with one or more embodiments;

Figure 5B depicts one example of a load monitored region register, in accordance with one or more embodiments;

Figure 6 depicts one embodiment of logic to perform garbage collection using the load doubleword instruction, in accordance with one or more embodiments;

Figure 7 depicts one embodiment of logic to perform optimized garbage collection;

Figure 8 depicts one embodiment of logic to take action by a handler based on a specified condition;

Figure 9 depicts one example of a cloud computing node, in accordance with one or more embodiments;

Figure 10 depicts one embodiment of a cloud computing environment, in accordance with one or more embodiments; and

Figure 11 depicts one example of abstraction model layers, in accordance with one or more embodiments.

DETAILED DESCRIPTION

[0020] In accordance with one or more embodiments, a capability is provided for an optimized garbage collection process that advantageously improves application performance, improves performance of the processor executing the application, and/or improves performance of the computing environment in which the processor executes.

[0021] The optimized garbage collection process allows applications (also referred to as programs) to perform garbage collection absent requiring program modification to make use of special instructions or hardware to execute special instructions. Further, in one or more embodiments, it allows applications that are accessing objects in an area of memory not undergoing garbage collection to continue processing during garbage collection without interruption, allows applications accessing objects in an area of memory being garbage collected to continue processing after a very short unnoticeable delay, and further improves the handling of the object pointers (also referred to as pointers).

[0022] In one embodiment, an instruction, referred to as a load doubleword (ld) instruction, is used to load data, including object pointers (e.g., addresses of objects). When the instruction is executed, the address of the data (e.g., object pointer) to be loaded is compared to a pointer storage area, and if the data is not in the pointer storage area, then the load instruction is executed as conventional. However, if the address of the data to be loaded is in the pointer storage area, then a check is made to determine if the data to be loaded points to an object in a given address range (e.g., a memory area undergoing garbage collection). If it does not point to an object in the address range, then the load instruction is executed as conventional. However, if the data to be loaded does point to the given address range, the processor causes an

asynchronous branch (referred to as an Event-Based Branch (EBB)) to a pointer update handler (also referred to as a garbage collection handler, an EBB handler, or handler). This enables the pointer update handler to update the pointer (e.g., the address of the object) if the object pointed to has been moved during an ongoing garbage collection process or is moved by the handler. The updated pointer is then stored in a selected location, such as a location within the pointer storage area, in one example.

[0023] In one embodiment, to determine the address of the pointer, the pointer update handler examines the ld instruction to determine the source registers, reads the source registers, and calculates the address of the pointer based on contents of the source registers. In yet a further embodiment, instead of the pointer update handler calculating the pointer address as indicated above, the pointer update handler is provided or has direct access to the pointer address. In this embodiment, the processor hardware obtains the address of the pointer from, for instance, the instruction (e.g., EA indicated by the ld instruction described below), and stores the address of the pointer (also referred to as a pointer address) in a pre-defined location, such as a specified memory location or register directly accessible to the pointer update handler.

[0024] As one example, the pointer address may be stored in a specified location in storage. For instance, the storage address at which the pointer address is to be stored may be specified to immediately precede the EBB handler address stored in an Event Based Branch Handler Register (EBBHR), or at some specified offset from the address. An example offset might be to store the pointer address at a storage location that immediately precedes the start of the pointer update handler itself. If this is done, the pointer update handler simply reads that storage location in order to determine the pointer address. As a further example, the pointer address is stored in an existing register that is not needed for another purpose. For instance, the register might be part of another facility, such as a performance monitor facility. Yet further, the address may be stored in the register that is used to store the address of the pointer update handler, the EBBHR. The EBBHR could be loaded with the pointer address by the hardware immediately after the EBB occurred, and then, the pointer update handler would read that register when the pointer address was needed, and would restore it to contain the start address of the pointer update handler prior to returning control back to the application. Any other register may be used similarly, provided that the register was not needed during execution of

the pointer update handler, other than to indicate the pointer address, and that register could be restored by the pointer update handler prior to returning to the application.

[0025] One embodiment of a computing environment to incorporate and use one or more embodiments is described with reference to FIG. 1. A computing environment 100 includes, for instance, a processor 102 (e.g., a central processing unit), a memory 104 (e.g., main memory), and one or more input/output (I/O) devices and/or interfaces 106 coupled to one another via, for example, one or more buses 108 and/or other connections.

[0026] In one embodiment, processor 102 is based on the Power Architecture offered by International Business Machines Corporation. One embodiment of the Power Architecture is described in "Power ISA™ Version 2.07B," International Business Machines Corporation, April 9, 2015, which is hereby incorporated herein by reference in its entirety. POWER ARCHITECTURE® is a registered trademark of International Business Machines Corporation, Armonk, New York, USA. Other names used herein may be registered trademarks, trademarks, or product names of International Business Machines Corporation or other companies.

[0027] In another example, processor 102 is based on the z/Architecture offered by International Business Machines Corporation, and is part of a server, such as the System z server, which implements the z/Architecture and is also offered by International Business Machines Corporation. One embodiment of the z/Architecture is described in an IBM® publication entitled, "z/Architecture Principles of Operation," IBM® Publication No. SA22-7832-10, Eleventh Edition, March 2015, which is hereby incorporated herein by reference in its entirety. In one example, the processor executes an operating system, such as z/OS, also offered by International Business Machines Corporation. IBM®, Z/ARCHITECTURE® and Z/OS® are registered trademarks of International Business Machines Corporation.

[0028] In yet a further embodiment, processor 102 is based on an Intel architecture offered by Intel Corporation. Intel® is a registered trademark of Intel Corporation, Santa Clara, California. Yet further, processor 102 may be based on other architectures. The architectures mentioned herein are merely provided as examples.

[0029] Another embodiment of a computing environment to incorporate and use one or more embodiments is described with reference to FIG. 2A. In this example, a computing environment 200 includes, for instance, a native central processing unit 202, a memory 204, and one or more input/output devices and/or interfaces 206 coupled to one another via, for example, one or more buses 208 and/or other connections. As examples, computing environment 200 may include a PowerPC® processor, a zSeries server, or a pSeries server offered by International Business Machines Corporation, Armonk, New York; an HP Superdome with Intel Itanium II processors offered by Hewlett Packard Co., Palo Alto, California; and/or other machines based on architectures offered by International Business Machines Corporation, Hewlett Packard, Intel, Oracle, or others.

[0030] Native central processing unit 202 includes one or more native registers 210, such as one or more general purpose registers and/or one or more special purpose registers used during processing within the environment. These registers include information that represent the state of the environment at any particular point in time.

[0031] Moreover, native central processing unit 202 executes instructions and code that are stored in memory 204. In one particular example, the central processing unit executes emulator code 212 stored in memory 204. This code enables the processing environment configured in one architecture to emulate another architecture. For instance, emulator code 212 allows machines based on architectures other than the Power® architecture, such as zSeries servers, pSeries servers, HP Superdome servers or others, to emulate the Power architecture and to execute software and instructions developed based on the Power architecture. In a further example, emulator code 212 allows machines based on architectures other than the z/Architecture, such as PowerPC processors, pSeries servers, HP Superdome servers or others, to emulate the z/Architecture and to execute software and instructions developed based on the z/Architecture. Other architectures may also be emulated. Power and PowerPC are trademarks of International Business Machines Corporation.

[0032] Further details relating to emulator code 212 are described with reference to FIG. 2B. Guest instructions 250 stored in memory 204 comprise software instructions (e.g., correlating to machine instructions) that were developed to be executed in an architecture other

than that of native CPU 202. For example, guest instructions 250 may have been designed to execute on a PowerPC processor or a z/Architecture processor 102, but instead, are being emulated on native CPU 202, which may be, for example, an Intel Itanium II processor. In one example, emulator code 212 includes an instruction fetching routine 252 to obtain one or more guest instructions 250 from memory 204, and to optionally provide local buffering for the instructions obtained. It also includes an instruction translation routine 254 to determine the type of guest instruction that has been obtained and to translate the guest instruction into one or more corresponding native instructions 256. This translation includes, for instance, identifying the function to be performed by the guest instruction and choosing the native instruction(s) to perform that function.

[0033] Further, emulator code 212 includes an emulation control routine 260 to cause the native instructions to be executed. Emulation control routine 260 may cause native CPU 202 to execute a routine of native instructions that emulate one or more previously obtained guest instructions and, at the conclusion of such execution, return control to the instruction fetch routine to emulate the obtaining of the next guest instruction or a group of guest instructions. Execution of the native instructions 256 may include loading data into a register from memory 204; storing data back to memory from a register; or performing some type of arithmetic or logic operation, as determined by the translation routine.

[0034] Each routine is, for instance, implemented in software, which is stored in memory and executed by native central processing unit 202. In other examples, one or more of the routines or operations are implemented in firmware, hardware, software or some combination thereof. The registers of the emulated processor may be emulated using registers 210 of the native CPU or by using locations in memory 204. In embodiments, guest instructions 250, native instructions 256 and emulator code 212 may reside in the same memory or may be disbursed among different memory devices.

[0035] As used herein, firmware includes, e.g., the microcode, millicode and/or macrocode of the processor. It includes, for instance, the hardware-level instructions and/or data structures used in implementation of higher level machine code. In one embodiment, it includes, for instance, proprietary code that is typically delivered as microcode that includes trusted software

or microcode specific to the underlying hardware and controls operating system access to the system hardware.

[0036] In one example, a guest instruction 250 that is obtained, translated and executed is an instruction described herein. The instruction, which is of one architecture (e.g., the Power architecture or z/Architecture) is fetched from memory, translated and represented as a sequence of native instructions 256 of another architecture (e.g., the z/Architecture, Power architecture, Intel architecture, etc.). These native instructions are then executed.

[0037] One instruction used in accordance with one or more embodiments is the load doubleword instruction used to load data, including object pointers. One particular implementation of the load doubleword instruction in the Power Architecture is described with reference to FIG. 3. In one example, a load doubleword (ld) instruction 300 includes operation code (opcode) fields 302a (e.g., bits 0-5), 302b (e.g., bit 30) indicating a load operation; a result field (RT) 304 (e.g., bits 6-10) used to indicate a register to store a result of the load operation; a register field (RA) 306 (e.g., bits 11-15) used to specify a register to be used by the load operation; and another field (DS) (e.g., bits 16-29) used by the load operation. Each of the fields 304-308, in one example, is separate and independent from one another; however, in other embodiments, more than one field may be combined. Further information on the use of the fields is described below.

[0038] In operation of the ld instruction, a check is made as to whether the address of data (e.g., an object pointer) to be loaded is in the pointer storage area. If it is in the pointer storage area, then a further check is made as to whether the data to be loaded points to an object located in a selected portion of memory, referred to herein as a load monitored region. If the address of the data to be loaded is not in the pointer storage area or the data to be loaded does not point to an object located in the selected portion of memory, then a conventional load is performed. For instance, in one example, a conventional load doubleword is executed.

[0039] In operation of the conventional ld instruction: Let an effective address (EA) be the sum (RA|0) + (DS||0b00). The doubleword in storage addressed by EA is loaded into RT.

[0040] One example of pseudo-code for the conventional ld instruction is as follows:

If RA=0 then b ← 0

else b ← (RA)

EA ← b + EXTS(DS || 0b00)

RT ← MEM(EA, 8)

wherein EA is an address of the object pointer; and MEM(EA,8) is the object pointer.

[0041] If, however, the address of the data to be loaded is in the pointer storage area and the object pointer does point to an object located in the selected portion of memory undergoing garbage collection, then processing is interrupted causing an Event Based Branch to an update pointer handler that performs one or more tasks related to garbage collection, including updating the pointer, as described herein further below.

[0042] One example of pseudo-code for the ld instruction, in accordance with one or more embodiments, is as follows:

If RA=0 then b ← 0

else b ← (RA)

EA ← b + EXTS(DS||0b00)

loaded_ea ← MEM(EA, 8)

if ¬((EA in pointer storage & loaded_ea is in enabled section of load-monitored region) & $BESCR_{GE\ LME}$=0b11)

RT ← loaded_ea

wherein loaded_ea is the object pointer; EA is an address of the object pointer; BESCR refers to branch event status-control register; GE refers to general enable; and LME = load monitored enabled.

[0043] Although, in the examples herein, the instruction format is for the Power Architecture, similar formats may be used for other architectures.

[0044] Further, in another example, a load doubleword indexed (ldx) instruction may be used, in which instead of DS 308, there is another register field (RB) 308 in bits 16-20, and bits 21-30 include a portion of the opcode 302b. With this format, EA ← b+(RB).

[0045] In one embodiment, as indicated previously, the selected portion of memory undergoing garbage collection is referred to herein as the load monitored region. As shown in FIG. 4, memory 400 includes a load monitored region 402, as well as a plurality of objects 404. One of the objects, Object B 406, is in load monitored region 402 meaning that the object is in a portion of memory in which garbage collection is being performed. Therefore, the current pointer may need to be updated, if the object to which the pointer points has been moved due to, for instance, the garbage collection process.

[0046] Further, as used herein, an object area includes the load monitored region and the area of memory including objects that are not undergoing garbage collection.

[0047] Additionally, in one embodiment, memory 400 includes a pointer storage area 410 including a plurality of pointers.

[0048] In this figure, it is further shown that an application program 420 executes an ld instruction 422, which attempts to load pointer B which is in the pointer storage area. Pointer B points to an object 406 in the load monitored region, and thus, an interrupt is performed giving control to EBB handler 424 (also known as the update pointer handler). Handler 424 modifies pointer B, if necessary, and stores the modified pointer in the location from which it was obtained, 426, in one example. Processing then returns to the instruction, which is re-executed.

[0049] In a further embodiment, the handler modifies the pointer, stores the modified pointer in the target register of the instruction, and processing continues at the instruction after the ld instruction, thereby emulating the load of the pointer. In one or more embodiments, the application is unaware of the EBB processing, and simply receives the pointer, as before.

[0050] As indicated above, interrupt processing is performed when the address of the data (e.g., the object pointer) to be loaded is in a pointer storage area and the object pointer points to an object that is in the load monitored region of memory. The pointer storage area may be

identified in many ways, including, for instance, specifying its size and base address in a register or a specified location in memory. For instance, as shown in FIG. 5A, a pointer storage area (PSA) register 500 includes, for instance, a field 502 including a pointer area base address, and a field 504 including a size of the area. In one example, bits 0:53 identify the starting address of the pointer storage area. In this example, the base address is aligned on an address boundary that is the same as its size in order to facilitate implementation. For example, a 256K pointer storage area would be aligned on a 256K boundary, and a 2K pointer storage area is aligned on a 2K boundary. This alignment is a means to allow the pointer address and its size to fit within a 64-bit register and is not an absolute requirement. Any alignment may be possible using a larger register or multiple registers.

[0051] Bits 61:63, in this embodiment, determine the size of the pointer storage area, as follows:

Bit 61:63	**Size**
000	256K
001	128K
010	64K
011	32K
100	16K
101	8K
110	4K
111	2K

[0052] The load monitored region may also be identified in alternative ways. For instance, in one implementation, its size and base address are stored in a register, such as depicted in FIG. 5B. As shown, a load monitored region register (LMRR) 550 includes, for instance, a field 552 including a load monitored region base address, and a field 554 including a size of the region.

[0053] In one example, the load monitored region base address includes the high-order bits of the load monitored region. In this embodiment, it is assumed that the load monitored region is aligned on a granularity of its size. The size field is encoded such that each value

corresponds to a specified size. For example, if 16 possible sizes are needed, the size field has 4 bits. Typical sizes are in the order of 10's of MBs (megabytes) to over a GB (gigabyte). The number of bits in the load monitored region base address field can be derived from the minimum size supported. For example, if the minimum size supported is 16 MB, then the load monitored region base address field is 40 bits, since 64-40=24, and 24 bits of address are sufficient to identify any 16 MB memory region aligned on a 16 MB address boundary. When the size field indicates smaller sizes, then fewer bits are required to specify the base address, and low-order bits in the load monitored region base address field are set to 0s.

[0054] In other examples, the size and base address for the load monitored region and/or the pointer storage area may be specified in a memory location, or a register used for another purpose, etc. Additionally, other information may be used to specify the address range of the load monitored region and/or the pointer storage area.

[0055] One embodiment of logic associated with executing the ld instruction, in accordance with an embodiments of the present invention, is described with reference to FIG. 6. In one implementation, hardware of a processor executes an application that issues the ld instruction, and the processor hardware decodes ld, STEP 600. During execution of ld, the processor hardware compares the address of the data (e.g., the pointer) to be loaded to the pointer storage area indicated, for instance, by the PSA register, INQUIRY 601. If the address of the object pointer is in the pointer storage area, then the processor hardware further compares the data read with the load monitored region register (or other register or memory location) that specifies the selected portion of memory undergoing garbage collection. If the pointer that was read points to a location within the load monitored region, INQUIRY 602, then the hardware causes an interrupt (e.g., an Event Based Branch) to the pointer update handler, STEP 606. The pointer update handler then determines the address of the pointer either by calculating its address from the load instruction registers or by reading it from a pre-defined location; reads the pointer from the pointer storage area; modifies the pointer, if needed (i.e., if the object to which it points was moved during garbage collection or by the handler) and performs other garbage collection tasks as needed and as time permits; and stores the modified pointer back into the location it was read from, STEP 610. The handler can then return control to the application at the ld instruction and re-execute the instruction, STEP 612.

[0056] In a further example, the update handler reads the target register of the ld instruction and stores the updated pointer in the target register. This has the effect of emulating a load of the pointer, and therefore, the handler can then return control to the application at the instruction after the ld instruction.

[0057] Returning to INQUIRY 601, if the pointer is not in the pointer storage area or the pointer does not point to a location within the load monitored region, INQUIRY 602, then the processor hardware does not cause the Event Based Branch, but instead, executes the ld instruction without the interrupt, STEP 614. Thereafter, processing continues to the next instruction, STEP 616.

[0058] As described herein, garbage collection is optimized by allowing applications that are accessing objects in an area of memory not being garbage collected to continue processing (without being paused) during garbage collection, and allowing applications that are accessing objects in an area of memory being garbage collected to resume processing after a short, unnoticeable delay. This is enabled by determining during the load of the pointer that the address of the object pointer is located in a pointer storage area and the object being pointed to is in the selected portion of memory undergoing garbage collection, and thus, an interrupt is issued to a handler that manages the pointer. In one advantageous implementation, the pointer is available to the handler, and therefore, the handler does not have to calculate the address of the pointer, thereby, reducing the number of instructions needed to update the pointer, and improving computer processing.

[0059] One embodiment of the logic associated with facilitating garbage collection, in accordance with one or more embodimentss is described with reference to FIG. 7. Initially, a handler (e.g., an application level handler) executing within a processor obtains processing control, STEP 700, via, for instance, an interrupt (e.g., a lightweight interrupt that does not involve the operating system) issued by processor hardware, STEP 702. Processing control is obtained by the handler based on execution of a load instruction (e.g., ld or ldx) and a determination that an address of an object pointer to be loaded is located in a pointer storage area and the object pointer indicates a location within a selected portion of memory undergoing garbage collection, STEP 704. In one example, the address of the object pointer is obtained by

the load instruction (e.g., EA) and is compared with data related to the pointer storage area to determine whether the object pointer is located within the pointer storage area, STEP 706. Based on the comparing indicating the address of the object pointer is located within the pointer storage area, the object pointer is compared with information related to the selected portion of memory to determine that the object pointer indicates the location within the selected portion of memory, STEP 708. In one embodiment, the pointer storage area is indicated by a register that may include a base address and a size, or a memory location as examples, STEP 710. Further, the selected portion of memory is indicated by a register that may include a base address and a size, or a memory location, as examples, STEP 712.

[0060] Based on obtaining processing control, the handler obtains an address of the object pointer, STEP 720. In one example, the address is determined from execution of the ld instruction, STEP 722, or read from the pre-defined location, STEP 724. The handler uses the address to obtain (e.g., read) the object pointer from the pointer storage area indicated in, e.g., a register or memory location, STEP 726.

[0061] The handler then determines if the object pointer is to be modified, INQUIRY 730. For instance, the handler determines whether the object pointed to by the object pointer has been moved. If the object has been moved, then the object pointer is modified, STEP 735. Thereafter, the modified object pointer is stored, STEP 740, in a location it was read from (e.g., in a location within the pointer storage area) and processing continues to the ld instruction, which is re-executed; or it may be stored in a location specified by the load instruction, such as, e.g., in the target register specified by the load instruction, and processing continues to the instruction after the ld instruction.

[0062] Returning to INQUIRY 730, if the object pointer is not modified, then it is stored, for instance, in the target register of the load instruction, and the handler returns control to the application at the instruction after the ld, STEP 732.

[0063] Advantageously, in one or more embodiments, garbage collection is facilitated by allowing applications that are not accessing objects in the selected portion of memory to continue processing (that is, no need to pause) during garbage collection, 750. Also,

applications that are accessing objects in the selected portion of memory are only delayed briefly.

[0064] As described herein, one or more embodiments provide the address of the object pointer directly to the handler without requiring the handler to calculate the address of the object pointer. One or more embodiments eliminate the decoding by the handler of the ld instruction to determine the source and destination addresses, the reading of the contents of the source general purpose registers, and the calculating the effective address of the pointer—again, instead, the handler reads the address from a pre-defined location and does not need to calculate it. One or more embodiments advantageously enable applications requiring garbage collection to enter time-sensitive environments, such as high-speed trading, real-time mechanical control, time-sensitive games, etc. One or more embodiments do not increase the footprint of the application, since no additional instructions need to be inserted into the application, and do not require any modifications to the operating system. Further, the applications do not need to be modified.

[0065] In accordance with one or more embodiments, upon each access to an object pointer, processing may be diverted to a real-time garbage collection handler (e.g., pointer update handler) if the pointer is located in a pointer storage area and points to an object in a region of memory being garbage collected.

[0066] As described herein, garbage collection is facilitated. In one embodiment, processing control is obtained by a handler executing within a processor of the computing environment, the obtaining processing control being based on execution of a load instruction and a determination that an address of an object pointer to be loaded is located in a pointer storage area and the object pointer indicates a location within a selected portion of memory undergoing garbage collection. Based on obtaining processing control by the handler, the handler obtains from the pointer storage area the object pointer, the object pointer indicating a location of an object pointed to by the object pointer. The handler determines whether the object pointer is to be modified, and based on determining the object pointer is to be modified, the object pointer is modified to provide a modified object pointer. Based on modifying the

object pointer, the modified object pointer is stored in a selected location, such as a location within the pointer storage area or a location specified by the load instruction.

[0067] Advantageously, this allows applications using objects in an area of memory not undergoing garbage collection to continue processing during garbage collection without interruption, allows an application using an object in an area of memory undergoing garbage collection to continue processing after a very brief, unnoticeable delay, and does not require the use of special instructions or program modifications, thereby improving performance.

[0068] In one further embodiment, the selected portion of memory undergoing garbage collection is part of an object area that also includes one or more other objects not undergoing garbage collection, and advantageously, one or more applications accessing the object area not undergoing garbage collection continue process during garbage collection. For instance, they continue executing without interruption. Further, in one embodiment, the application that accessed the object pointer that indicates an object in the selected portion of memory undergoing garbage collection immediately resumes processing after a very brief delay during the time the handler (e.g., application-level handler) processes the pointer. This enables applications to be used for time-sensitive processing because no application is delayed for a time period that is significant enough to be noticeable.

[0069] Additionally, one or more embodiments may be used for other than garbage collection. For example, since one or more embodiments described herein may be used to detect when a pointer to a specified storage address range is loaded, it may be used to provide an advance warning about imminent access into a restricted memory space. In this case, a memory region is initialized to be the restricted memory region, and the pointer storage area is set to indicate the location of the memory pointers. Subsequently, when a pointer in the pointer storage area is read that points to a restricted area, an EBB occurs. The EBB handler can then either prevent the access entirely, provide an advance warning signal to a security monitor that an access is about to be made into a restricted area of memory or perform some other selected action. Other related applications are possible in situations in which there is a desire for an alert about an expected access that is about to be made to a specified memory area.

[0070] One embodiment of logic to take action by a handler based on a specified condition is described with reference to FIG. 8. In one example, a load instruction is decoded, STEP 800. (In another embodiment, it may be another type of instruction.) The load instruction may be one of various load instructions, including the ld instruction, the ldx instruction or another load instruction, as examples. The load instruction is decoded and based on the decoding, the object pointer is determined. A determination is made as to whether the object pointer indicates an object in a specified memory area, INQUIRY 802. This memory area is, for instance, a specified storage address range that is to be restricted for one reason or the other. If the pointer does not indicate an object in the specified memory area, then the load (or other instruction) is executed as conventional, STEP 804. Processing then continues at the next instruction, STEP 806.

[0071] However, returning to INQUIRY 802, if the pointer does indicate an object in a specified memory area, then control is obtained by a handler (e.g., an application-level handler), STEP 810. For instance, the processor hardware performs an interrupt (e.g., a lightweight interrupt that does not involve the operating system) to the handler. The handler may then take one or more actions, STEP 812. For instance, in one embodiment, the handler provides an alert, optionally prevents access to the specified memory area, and then continues processing at the next instruction, STEP 814. As a further example, the handler obtains the pointer address (e.g., from a pre-defined location, or calculates it from the instruction), reads the pointer, modifies the pointer, stores the modified pointer back in the location from which it was read, returns control to the instruction and re-executes the instruction, such that the specified memory area is not accessed, STEP 816. Other possibilities also exist.

[0072] As used herein, storage, central storage, main storage, memory and main memory are used interchangeably, unless otherwise noted, implicitly by usage or explicitly.

[0073] One or more embodiments may relate to cloud computing.

[0074] It is understood in advance that although this disclosure includes a detailed description on cloud computing, implementation of the teachings recited herein are not limited to a cloud computing environment. Rather, embodiments of the present invention are capable

of being implemented in conjunction with any other type of computing environment now known or later developed.

[0075] Cloud computing is a model of service delivery for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, network bandwidth, servers, processing, memory, storage, applications, virtual machines, and services) that can be rapidly provisioned and released with minimal management effort or interaction with a provider of the service. This cloud model may include at least five characteristics, at least three service models, and at least four deployment models.

[0076] Characteristics are as follows:

On-demand self-service: a cloud consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with the service's provider.

Broad network access: capabilities are available over a network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and PDAs).

Resource pooling: the provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to demand. There is a sense of location independence in that the consumer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter).

Rapid elasticity: capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

Measured service: cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported providing transparency for both the provider and consumer of the utilized service.

[0077] Service Models are as follows:

Software as a Service (SaaS): the capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based email). The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

Platform as a Service (PaaS): the capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including networks, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.

Infrastructure as a Service (IaaS): the capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).

[0078] Deployment Models are as follows:

Private cloud: the cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on-premises or off-premises.

Community cloud: the cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on-premises or off-premises.

Public cloud: the cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.

Hybrid cloud: the cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by

standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for loadbalancing between clouds).

[0079] A cloud computing environment is service oriented with a focus on statelessness, low coupling, modularity, and semantic interoperability. At the heart of cloud computing is an infrastructure comprising a network of interconnected nodes.

[0080] Referring now to FIG. 9, a schematic of an example of a cloud computing node is shown. Cloud computing node 10 is only one example of a suitable cloud computing node and is not intended to suggest any limitation as to the scope of use or functionality of embodiments of the invention described herein. Regardless, cloud computing node 10 is capable of being implemented and/or performing any of the functionality set forth hereinabove.

[0081] In cloud computing node 10 there is a computer system/server 12, which is operational with numerous other general purpose or special purpose computing system environments or configurations. Examples of well-known computing systems, environments, and/or configurations that may be suitable for use with computer system/server 12 include, but are not limited to, personal computer systems, server computer systems, thin clients, thick clients, handheld or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputer systems, mainframe computer systems, and distributed cloud computing environments that include any of the above systems or devices, and the like.

[0082] Computer system/server 12 may be described in the general context of computer system executable instructions, such as program modules, being executed by a computer system. Generally, program modules may include routines, programs, objects, components, logic, data structures, and so on that perform particular tasks or implement particular abstract data types. Computer system/server 12 may be practiced in distributed cloud computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed cloud computing environment, program modules may be located in both local and remote computer system storage media including memory storage devices.

[0083] As shown in FIG. 9, computer system/server 12 in cloud computing node 10 is shown in the form of a general-purpose computing device. The components of computer system/server 12 may include, but are not limited to, one or more processors or processing units 16, a system memory 28, and a bus 18 that couples various system components including system memory 28 to processor 16.

[0084] Bus 18 represents one or more of any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus.

[0085] Computer system/server 12 typically includes a variety of computer system readable media. Such media may be any available media that is accessible by computer system/server 12, and it includes both volatile and non-volatile media, removable and non-removable media.

[0086] System memory 28 can include computer system readable media in the form of volatile memory, such as random access memory (RAM) 30 and/or cache memory 32. Computer system/server 12 may further include other removable/non-removable, volatile/non-volatile computer system storage media. By way of example only, storage system 34 can be provided for reading from and writing to a non-removable, non-volatile magnetic media (not shown and typically called a "hard drive"). Although not shown, a magnetic disk drive for reading from and writing to a removable, non-volatile magnetic disk (e.g., a "floppy disk"), and an optical disk drive for reading from or writing to a removable, non-volatile optical disk such as a CD-ROM, DVD-ROM or other optical media can be provided. In such instances, each can be connected to bus 18 by one or more data media interfaces. As will be further depicted and described below, memory 28 may include at least one program product having a set (e.g., at least one) of program modules that are configured to carry out the functions of embodiments of the invention.

[0087] Program/utility 40, having a set (at least one) of program modules 42, may be stored in memory 28 by way of example, and not limitation, as well as an operating system, one or more application programs, other program modules, and program data. Each of the operating system, one or more application programs, other program modules, and program data or some combination thereof, may include an implementation of a networking environment. Program modules 42 generally carry out the functions and/or methodologies of embodiments of the invention as described herein.

[0088] Computer system/server 12 may also communicate with one or more external devices 14 such as a keyboard, a pointing device, a display 24, etc.; one or more devices that enable a user to interact with computer system/server 12; and/or any devices (e.g., network card, modem, etc.) that enable computer system/server 12 to communicate with one or more other computing devices. Such communication can occur via Input/Output (I/O) interfaces 22. Still yet, computer system/server 12 can communicate with one or more networks such as a local area network (LAN), a general wide area network (WAN), and/or a public network (e.g., the Internet) via network adapter 20. As depicted, network adapter 20 communicates with the other components of computer system/server 12 via bus 18. It should be understood that although not shown, other hardware and/or software components could be used in conjunction with computer system/server 12. Examples, include, but are not limited to: microcode, device drivers, redundant processing units, external disk drive arrays, RAID systems, tape drives, and data archival storage systems, etc.

[0089] Referring now to FIG. 10, illustrative cloud computing environment 50 is depicted. As shown, cloud computing environment 50 comprises one or more cloud computing nodes 10 with which local computing devices used by cloud consumers, such as, for example, personal digital assistant (PDA) or cellular telephone 54A, desktop computer 54B, laptop computer 54C, and/or automobile computer system 54N may communicate. Nodes 10 may communicate with one another. They may be grouped (not shown) physically or virtually, in one or more networks, such as Private, Community, Public, or Hybrid clouds as described hereinabove, or a combination thereof. This allows cloud computing environment 50 to offer infrastructure, platforms and/or software as services for which a cloud consumer does not need to maintain resources on a local computing device. It is understood that the types of computing devices

54A-N shown in FIG. 10 are intended to be illustrative only and that computing nodes 10 and cloud computing environment 50 can communicate with any type of computerized device over any type of network and/or network addressable connection (e.g., using a web browser).

[0090] Referring now to FIG. 11, a set of functional abstraction layers provided by cloud computing environment 50 (FIG. 10) is shown. It should be understood in advance that the components, layers, and functions shown in FIG. 11 are intended to be illustrative only and embodiments of the invention are not limited thereto. As depicted, the following layers and corresponding functions are provided:

Hardware and software layer 60 includes hardware and software components. Examples of hardware components include mainframes 61; RISC (Reduced Instruction Set Computer) architecture based servers 62; servers 63; blade servers 64; storage devices 65; networks and networking components 66. In some embodiments, software components include network application server software 67 and database software 68.

Virtualization layer 70 provides an abstraction layer from which the following examples of virtual entities may be provided: virtual servers 71; virtual storage 72; virtual networks 73, including virtual private networks; virtual applications and operating systems 74; and virtual clients 75.

[0091] In one example, management layer 80 may provide the functions described below. Resource provisioning 81 provides dynamic procurement of computing resources and other resources that are utilized to perform tasks within the cloud computing environment. Metering and Pricing 82 provide cost tracking as resources are utilized within the cloud computing environment, and billing or invoicing for consumption of these resources. In one example, these resources may comprise application software licenses. Security provides identity verification for cloud consumers and tasks, as well as protection for data and other resources. User portal 83 provides access to the cloud computing environment for consumers and system administrators. Service level management 84 provides cloud computing resource allocation and management such that required service levels are met. Service Level Agreement (SLA) planning and fulfillment 85 provide pre-arrangement for, and procurement of, cloud computing resources for which a future requirement is anticipated in accordance with an SLA.

[0092] Workloads layer 90 provides examples of functionality for which the cloud computing environment may be utilized. Examples of workloads and functions which may be provided from this layer include: mapping and navigation 91; software development and lifecycle management 92; virtual classroom education delivery 93; data analytics processing 94; transaction processing 95; and garbage collection processing of one or more embodiments of the present invention 96.

[0093] Aspects of the present invention are described herein with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems), and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer readable program instructions.

[0094] These computer readable program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks. These computer readable program instructions may also be stored in a computer readable storage medium that can direct a computer, a programmable data processing apparatus, and/or other devices to function in a particular manner, such that the computer readable storage medium having instructions stored therein comprises an article of manufacture including instructions which implement aspects of the function/act specified in the flowchart and/or block diagram block or blocks.

[0095] The computer readable program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other device to cause a series of operational steps to be performed on the computer, other programmable apparatus or other device to produce a computer implemented process, such that the instructions which execute on the computer, other programmable apparatus, or other device implement the functions/acts specified in the flowchart and/or block diagram block or blocks.

[0096] The flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods, and computer program products according to various embodiments of the present invention. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of instructions, which comprises one or more executable instructions for implementing the specified logical function(s). In some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts or carry out combinations of special purpose hardware and computer instructions.

[0097] In addition to the above, one or more aspects may be provided, offered, deployed, managed, serviced, etc. by a service provider who offers management of customer environments. For instance, the service provider can create, maintain, support, etc. computer code and/or a computer infrastructure that performs one or more aspects for one or more customers. In return, the service provider may receive payment from the customer under a subscription and/or fee agreement, as examples. Additionally or alternatively, the service provider may receive payment from the sale of advertising content to one or more third parties.

[0098] In one aspect, an application may be deployed for performing one or more embodiments. As one example, the deploying of an application comprises providing computer infrastructure operable to perform one or more embodiments.

[0099] As a further aspect, a computing infrastructure may be deployed comprising integrating computer readable code into a computing system, in which the code in combination with the computing system is capable of performing one or more embodiments.

[00100] As yet a further aspect, a process for integrating computing infrastructure comprising integrating computer readable code into a computer system may be provided. The

computer system comprises a computer readable medium, in which the computer medium comprises one or more embodiments. The code in combination with the computer system is capable of performing one or more embodiments.

[00101] Although various embodiments are described above, these are only examples. For example, computing environments of other architectures can be used to incorporate and use one or more embodiments. Further, different instructions, instruction formats, instruction fields and/or instruction values may be used. Many variations are possible.

[0102] Further, other types of computing environments can benefit and be used. As an example, a data processing system suitable for storing and/or executing program code is usable that includes at least two processors coupled directly or indirectly to memory elements through a system bus. The memory elements include, for instance, local memory employed during actual execution of the program code, bulk storage, and cache memory which provide temporary storage of at least some program code in order to reduce the number of times code must be retrieved from bulk storage during execution.

[0103] Input/Output or I/O devices (including, but not limited to, keyboards, displays, pointing devices, DASD, tape, CDs, DVDs, thumb drives and other memory media, etc.) can be coupled to the system either directly or through intervening I/O controllers. Network adapters may also be coupled to the system to enable the data processing system to become coupled to other data processing systems or remote printers or storage devices through intervening private or public networks. Modems, cable modems, and Ethernet cards are just a few of the available types of network adapters.

[0104] The terminology used herein is for the purpose of describing particular embodiments only and is not intended to be limiting. As used herein, the singular forms “a”, “an” and “the” are intended to include the plural forms as well, unless the context clearly indicates otherwise. It will be further understood that the terms “comprises” and/or “comprising”, when used in this specification, specify the presence of stated features, integers, steps, operations, elements, and/or components, but do not preclude the presence or addition of one or more other features, integers, steps, operations, elements, components and/or groups thereof.

[0105] The corresponding structures, materials, acts, and equivalents of all means or step plus function elements in the claims below, if any, are intended to include any structure, material, or act for performing the function in combination with other claimed elements as specifically claimed. The description of one or more embodiments has been presented for purposes of illustration and description, but is not intended to be exhaustive or limited to in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in order to best explain various aspects and the practical application, and to enable others of ordinary skill in the art to understand various embodiments with various modifications as are suited to the particular use contemplated.

CLAIMS

1. A computer program product for facilitating garbage collection within a computing environment, said computer program product comprising:

a computer readable storage medium readable by a processing circuit and storing instructions for execution by the processing circuit for performing a method comprising:

obtaining processing control by a handler executing within a processor of the computing environment, the obtaining processing control being based on execution of a load instruction and a determination that an address of an object pointer to be loaded is located in a pointer storage area and the object pointer indicates a location within a selected portion of memory undergoing garbage collection;

based on obtaining processing control by the handler, obtaining by the handler from the pointer storage area the object pointer, the object pointer indicating a location of an object pointed to by the object pointer;

determining by the handler whether the object pointer is to be modified;

modifying by the handler, based on determining the object pointer is to be modified, the object pointer to provide a modified object pointer; and

storing, based on modifying the object pointer, the modified object pointer in a selected location.

2. The computer program product of claim 1, wherein the selected location is a location within the pointer storage area or a location specified by the load instruction.

3. The computer program product of claim 1, wherein the obtaining processing control is via an interrupt issued by processor hardware, the interrupt issued based on execution of the load instruction and determination that the address of the object pointer to be loaded is located in the pointer storage area and the object pointer indicates the location within the selected portion of memory undergoing garbage collection.

4. The computer program product of claim 1, wherein the selected portion of memory undergoing garbage collection is part of an object area that includes one or more other objects

not undergoing garbage collection, and wherein one or more applications accessing the object area not undergoing garbage collection continue to process during garbage collection.

5. The computer program product of claim 1, wherein the method further comprises: determining from execution of the load instruction the address of the object pointer;

comparing the address of the object pointer with data related to the pointer storage area to determine whether the address of the object pointer is located within the pointer storage area;

based on the comparing determining that the address of the object pointer is located within the pointer storage area, comparing the object pointer with information relating to the selected portion of memory to determine that the object pointer indicates the location within the selected portion of memory; and

providing control to the handler, based on determining the address of the object pointer is located within the pointer storage area and the object pointer indicates the location within the selected portion of memory.

6. The computer program product of claim 1, wherein the pointer storage area comprises a contiguous area of memory that includes a plurality of object pointer addresses.

7. The computer program product of claim 1, wherein the pointer storage area is indicated by one of a register or a location within memory.

8. The computer program product of claim 1, wherein the selected portion of memory is indicated by one of a register or a location within memory.

9. The computer program product of claim 8, wherein the register includes a base address of the selected portion of memory and a size of the selected portion of memory.

10. The computer program product of claim 1, wherein the load instruction includes one or more operation code fields to specify a load operation, a result field to be used to obtain the object pointer, and one or more other fields to be used in the load operation.

11. A computer system for facilitating garbage collection within a computing environment, said computer system comprising:

a memory; and

a processor in communications with the memory, wherein the computer system is configured to perform a method, said method comprising:

obtaining processing control by a handler executing within a processor of the computing environment, the obtaining processing control being based on execution of a load instruction and a determination that an address of an object pointer to be loaded is located in a pointer storage area and the object pointer indicates a location within a selected portion of memory undergoing garbage collection;

based on obtaining processing control by the handler, obtaining by the handler from the pointer storage area the object pointer, the object pointer indicating a location of an object pointed to by the object pointer;

determining by the handler whether the object pointer is to be modified;

modifying by the handler, based on determining the object pointer is to be modified, the object pointer to provide a modified object pointer; and

storing, based on modifying the object pointer, the modified object pointer in a selected location.

12. The computer system of claim 11, wherein the obtaining processing control is via an interrupt issued by processor hardware, the interrupt issued based on execution of the load instruction and determination that the address of the object pointer to be loaded is located in the pointer storage area and the object pointer indicates the location within the selected portion of memory undergoing garbage collection.

13. The computer system of claim 11, wherein the selected portion of memory undergoing garbage collection is part of an object area that includes one or more other objects not undergoing garbage collection, and wherein one or more applications accessing the object area not undergoing garbage collection continue to process during garbage collection.

14. The computer system of claim 11, wherein the method further comprises:

determining from execution of the load instruction the address of the object pointer;

comparing the address of the object pointer with data related to the pointer storage area to determine whether the address of the object pointer is located within the pointer storage area;

based on the comparing determining that the address of the object pointer is located within the pointer storage area, comparing the object pointer with information relating to the selected portion of memory to determine that the object pointer indicates the location within the selected portion of memory; and

providing control to the handler, based on determining the address of the object pointer is located within the pointer storage area and the object pointer indicates the location within the selected portion of memory.

15. The computer system of claim 11, wherein the selected location is a location within the pointer storage area or a location specified by the load instruction.

16. A computer-implemented method of facilitating garbage collection within a computing environment, said computer-implemented method comprising:

obtaining processing control by a handler executing within a processor of the computing environment, the obtaining processing control being based on execution of a load instruction and a determination that an address of an object pointer to be loaded is located in a pointer storage area and the object pointer indicates a location within a selected portion of memory undergoing garbage collection;

based on obtaining processing control by the handler, obtaining by the handler from the pointer storage area the object pointer, the object pointer indicating a location of an object pointed to by the object pointer;

determining by the handler whether the object pointer is to be modified;

modifying by the handler, based on determining the object pointer is to be modified, the object pointer to provide a modified object pointer; and

storing, based on modifying the object pointer, the modified object pointer in a selected location.

17. The computer-implemented method of claim 16, wherein the obtaining processing control is via an interrupt issued by processor hardware, the interrupt issued based on execution of the load instruction and determination that the object pointer to be loaded is located in the

pointer storage area and indicates the location within the selected portion of memory undergoing garbage collection.

18. The computer-implemented method of claim 16, wherein the selected portion of memory undergoing garbage collection is part of an object area that includes one or more other objects not undergoing garbage collection, and wherein one or more applications accessing the object area not undergoing garbage collection continue to process during garbage collection.

19. The computer-implemented method of claim 16, further comprising:

determining from execution of the load instruction the address of the object pointer;

comparing the address of the object pointer with data related to the pointer storage area to determine whether the address of the object pointer is located within the pointer storage area;

based on the comparing determining that the address of the object pointer is located within the pointer storage area, comparing the object pointer with information relating to the selected portion of memory to determine that the object pointer indicates the location within the selected portion of memory; and

providing control to the handler, based on determining the address of the object pointer is located within the pointer storage area and the object pointer indicates the location within the selected portion of memory.

20. A computer program product for facilitating processing within a computing environment, said computer program product comprising:

a computer readable storage medium readable by a processing circuit and storing instructions for execution by the processing circuit for performing a method comprising:

obtaining processing control by a handler executing within a processor of the computing environment, the obtaining processing control being based on a determination that an object pointer to be accessed indicates a location within a specified memory area; and

based on obtaining processing control by the handler, taking action by the handler, the taking action comprising at least one of: providing an alert; preventing access to the specified memory area; or modifying the object pointer and storing the modified object pointer.

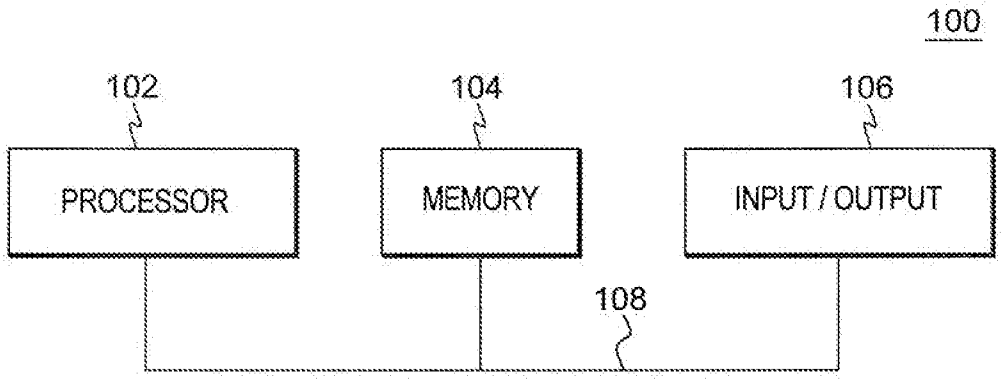

FIG. 1

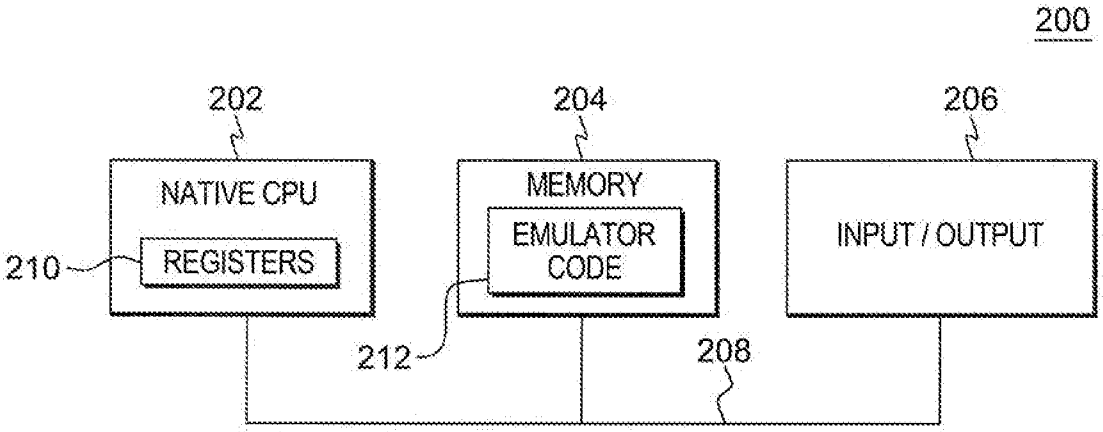

FIG. 2A

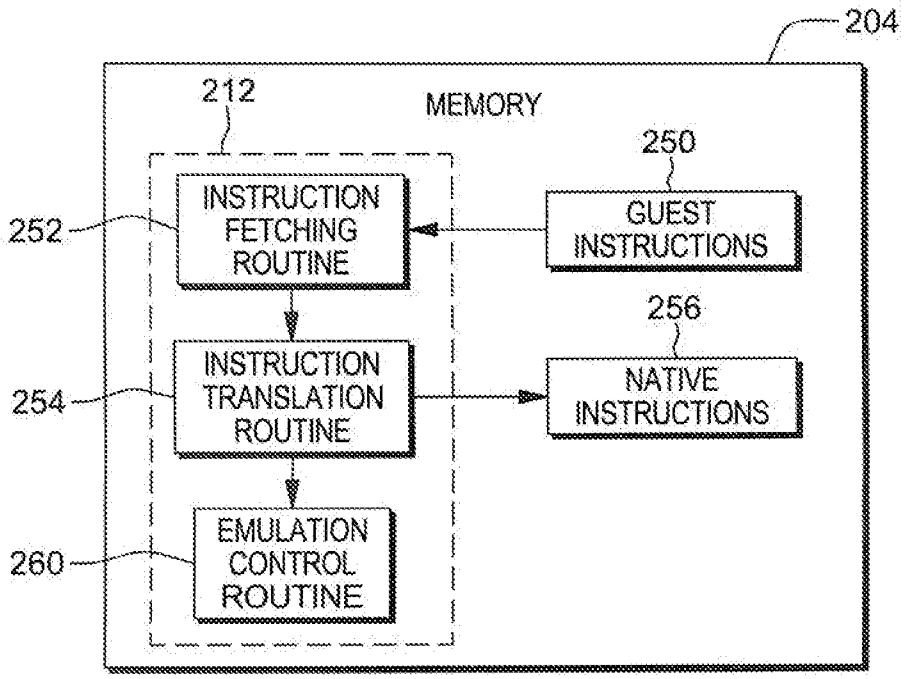

FIG. 2B

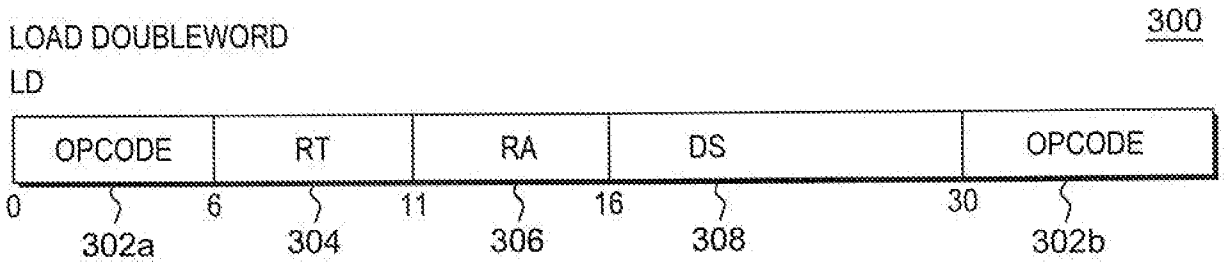

FIG. 3

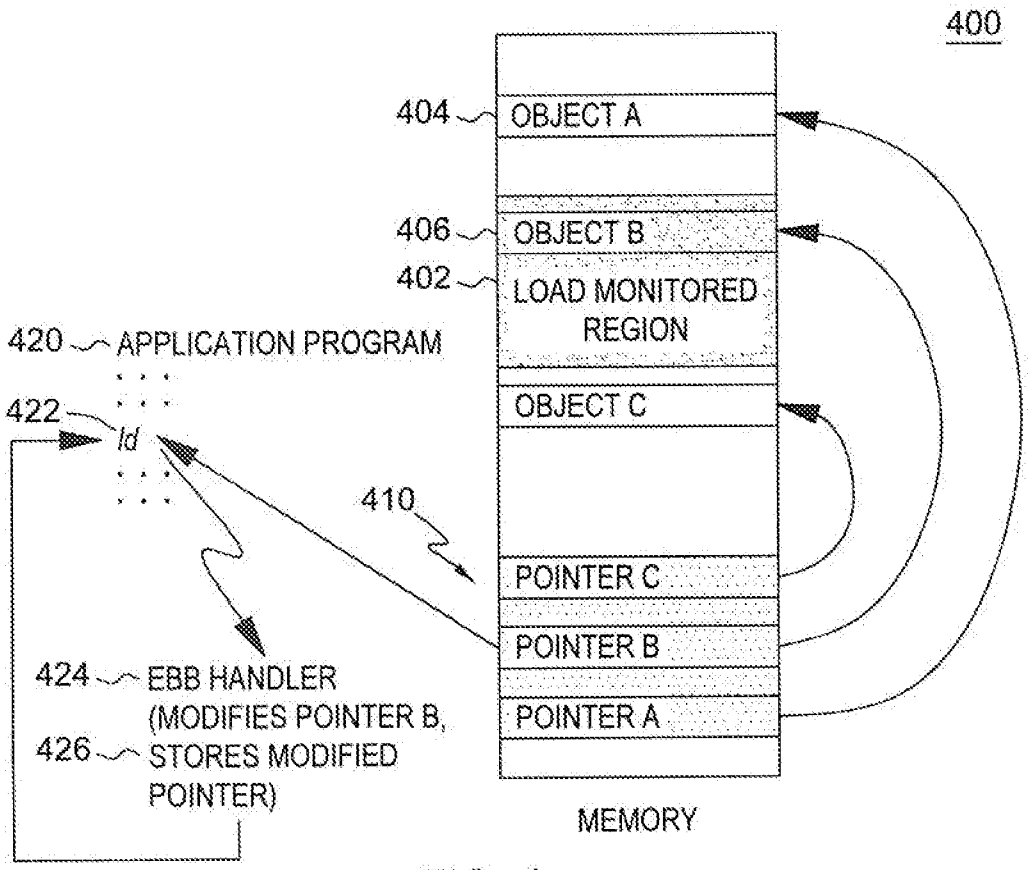

FIG. 4

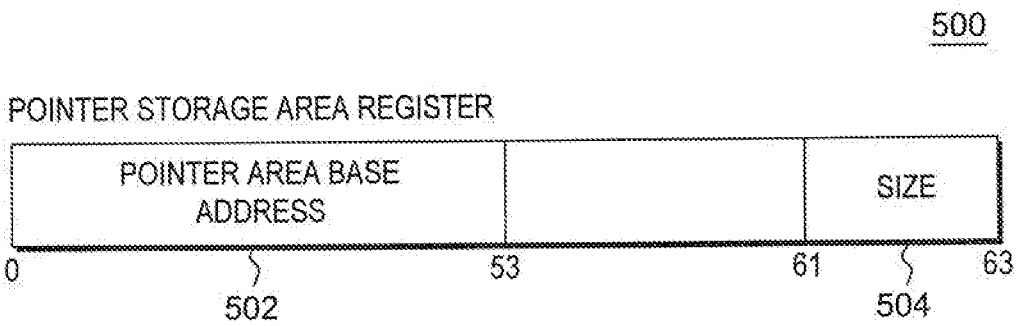

FIG. 5A

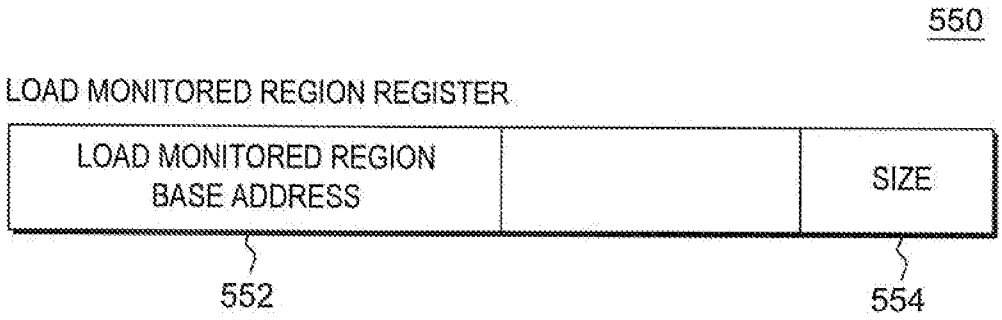
550
LOAD MONITORED REGION REGISTER
LOAD MONITORED REGION BASE ADDRESS
SIZE
552
554

FIG. 5B

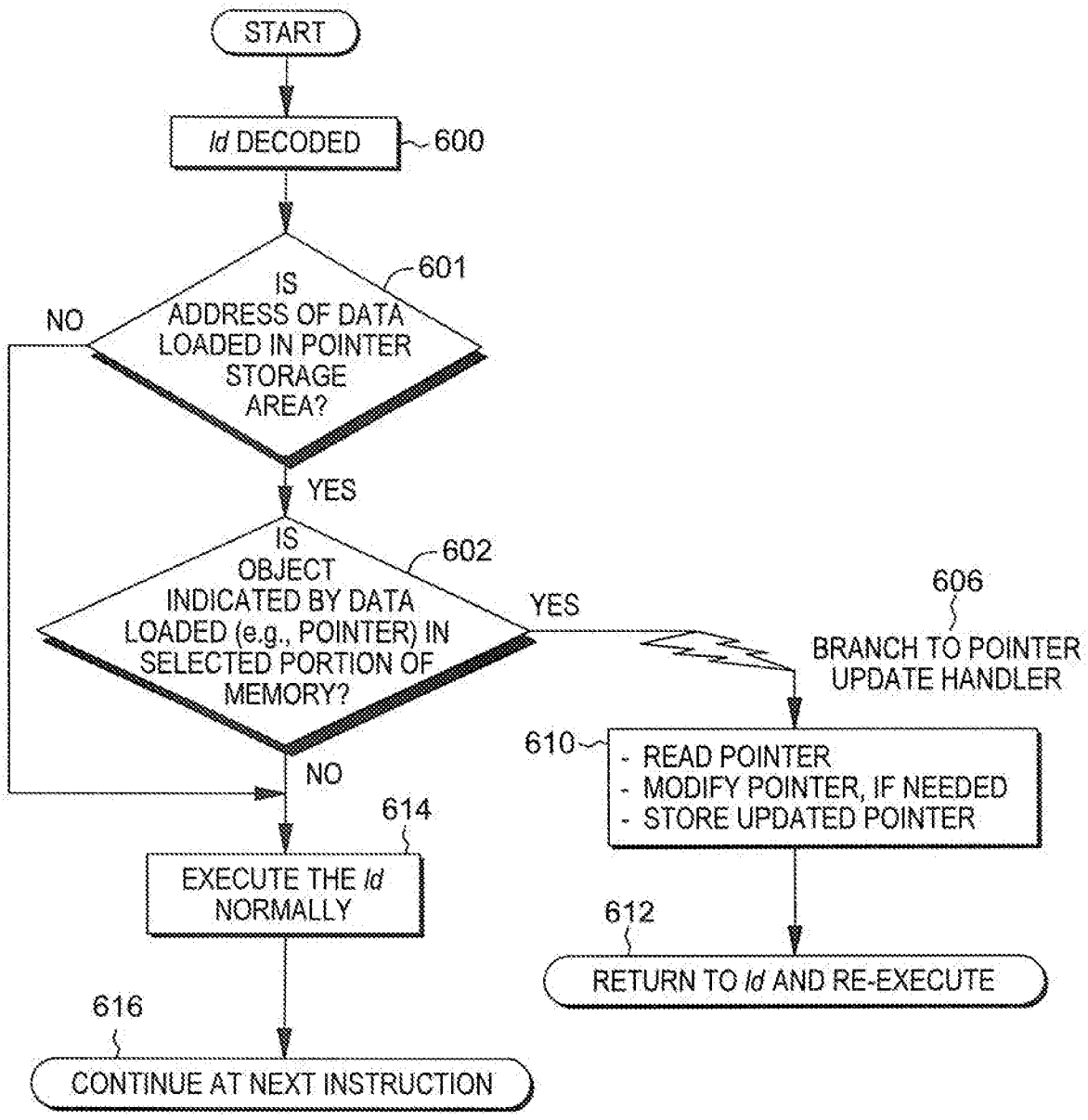
START
ld DECODED
600
IS ADDRESS OF DATA LOADED IN POINTER STORAGE AREA?
601
NO
YES
IS OBJECT INDICATED BY DATA LOADED (e.g., POINTER) IN SELECTED PORTION OF MEMORY?
602
YES
606
BRANCH TO POINTER UPDATE HANDLER
610
- READ POINTER
- MODIFY POINTER, IF NEEDED
- STORE UPDATED POINTER
NO
614
EXECUTE THE ld NORMALLY
612
RETURN TO ld AND RE-EXECUTE
616
CONTINUE AT NEXT INSTRUCTION

FIG. 6

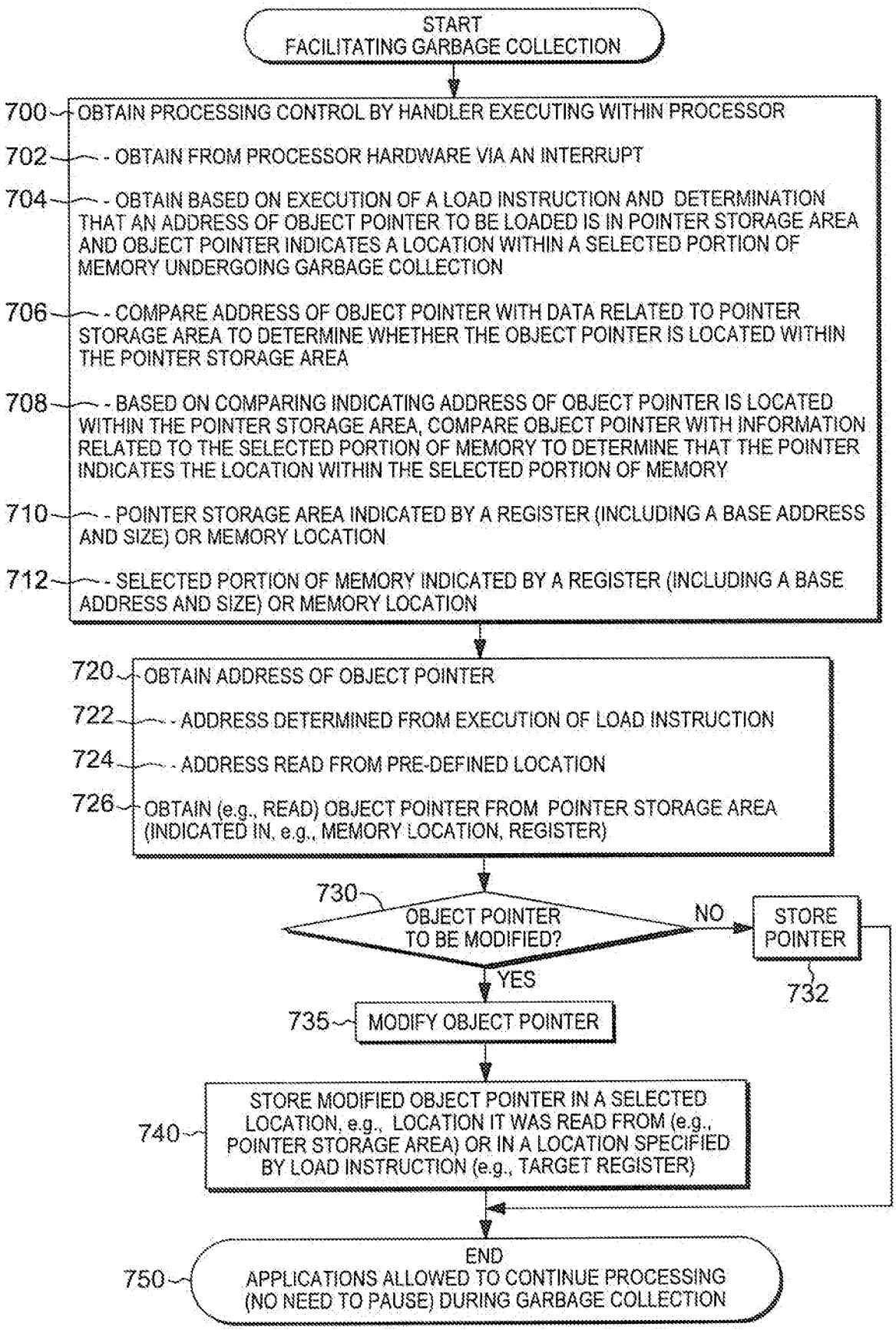
START
FACILITATING GARBAGE COLLECTION
700
OBTAIN PROCESSING CONTROL BY HANDLER EXECUTING WITHIN PROCESSOR
702
- OBTAIN FROM PROCESSOR HARDWARE VIA AN INTERRUPT
704
- OBTAIN BASED ON EXECUTION OF A LOAD INSTRUCTION AND DETERMINATION THAT AN ADDRESS OF OBJECT POINTER TO BE LOADED IS IN POINTER STORAGE AREA AND OBJECT POINTER INDICATES A LOCATION WITHIN A SELECTED PORTION OF MEMORY UNDERGOING GARBAGE COLLECTION
706
- COMPARE ADDRESS OF OBJECT POINTER WITH DATA RELATED TO POINTER STORAGE AREA TO DETERMINE WHETHER THE OBJECT POINTER IS LOCATED WITHIN THE POINTER STORAGE AREA
708
- BASED ON COMPARING INDICATING ADDRESS OF OBJECT POINTER IS LOCATED WITHIN THE POINTER STORAGE AREA, COMPARE OBJECT POINTER WITH INFORMATION RELATED TO THE SELECTED PORTION OF MEMORY TO DETERMINE THAT THE POINTER INDICATES THE LOCATION WITHIN THE SELECTED PORTION OF MEMORY
710
- POINTER STORAGE AREA INDICATED BY A REGISTER (INCLUDING A BASE ADDRESS AND SIZE) OR MEMORY LOCATION
712
- SELECTED PORTION OF MEMORY INDICATED BY A REGISTER (INCLUDING A BASE ADDRESS AND SIZE) OR MEMORY LOCATION
720
OBTAIN ADDRESS OF OBJECT POINTER
722
- ADDRESS DETERMINED FROM EXECUTION OF LOAD INSTRUCTION
724
- ADDRESS READ FROM PRE-DEFINED LOCATION
726
OBTAIN (e.g., READ) OBJECT POINTER FROM POINTER STORAGE AREA (INDICATED IN, e.g., MEMORY LOCATION, REGISTER)
730
OBJECT POINTER TO BE MODIFIED?
NO
STORE POINTER
732
YES
735
MODIFY OBJECT POINTER
740
STORE MODIFIED OBJECT POINTER IN A SELECTED LOCATION, e.g., LOCATION IT WAS READ FROM (e.g., POINTER STORAGE AREA) OR IN A LOCATION SPECIFIED BY LOAD INSTRUCTION (e.g., TARGET REGISTER)
750
END
APPLICATIONS ALLOWED TO CONTINUE PROCESSING (NO NEED TO PAUSE) DURING GARBAGE COLLECTION

FIG. 7

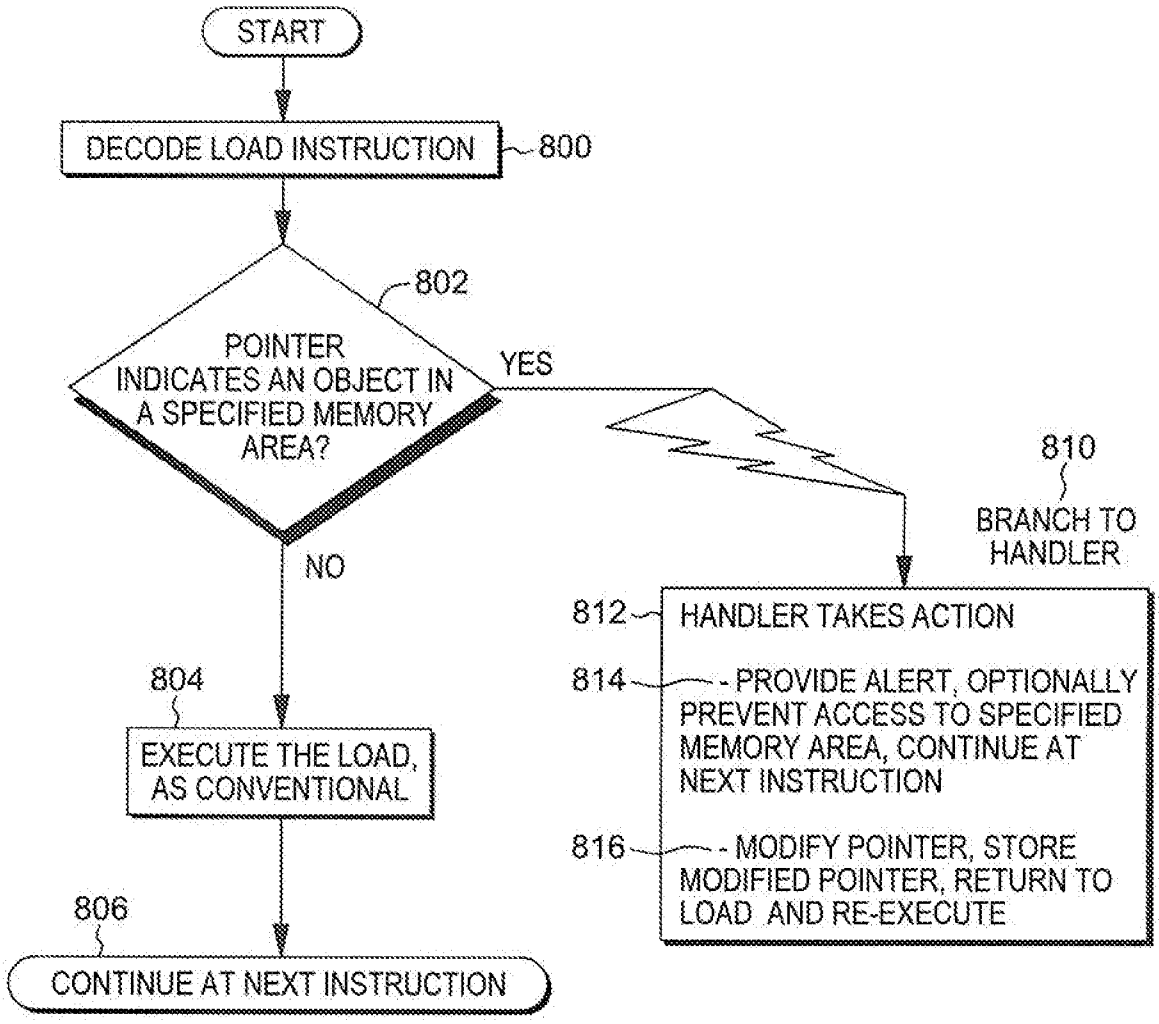
START
DECODE LOAD INSTRUCTION
800
802
POINTER INDICATES AN OBJECT IN A SPECIFIED MEMORY AREA?
YES
NO
810
BRANCH TO HANDLER
812
HANDLER TAKES ACTION
814
- PROVIDE ALERT, OPTIONALLY PREVENT ACCESS TO SPECIFIED MEMORY AREA, CONTINUE AT NEXT INSTRUCTION
816
- MODIFY POINTER, STORE MODIFIED POINTER, RETURN TO LOAD AND RE-EXECUTE
804
EXECUTE THE LOAD, AS CONVENTIONAL
806
CONTINUE AT NEXT INSTRUCTION

FIG. 8

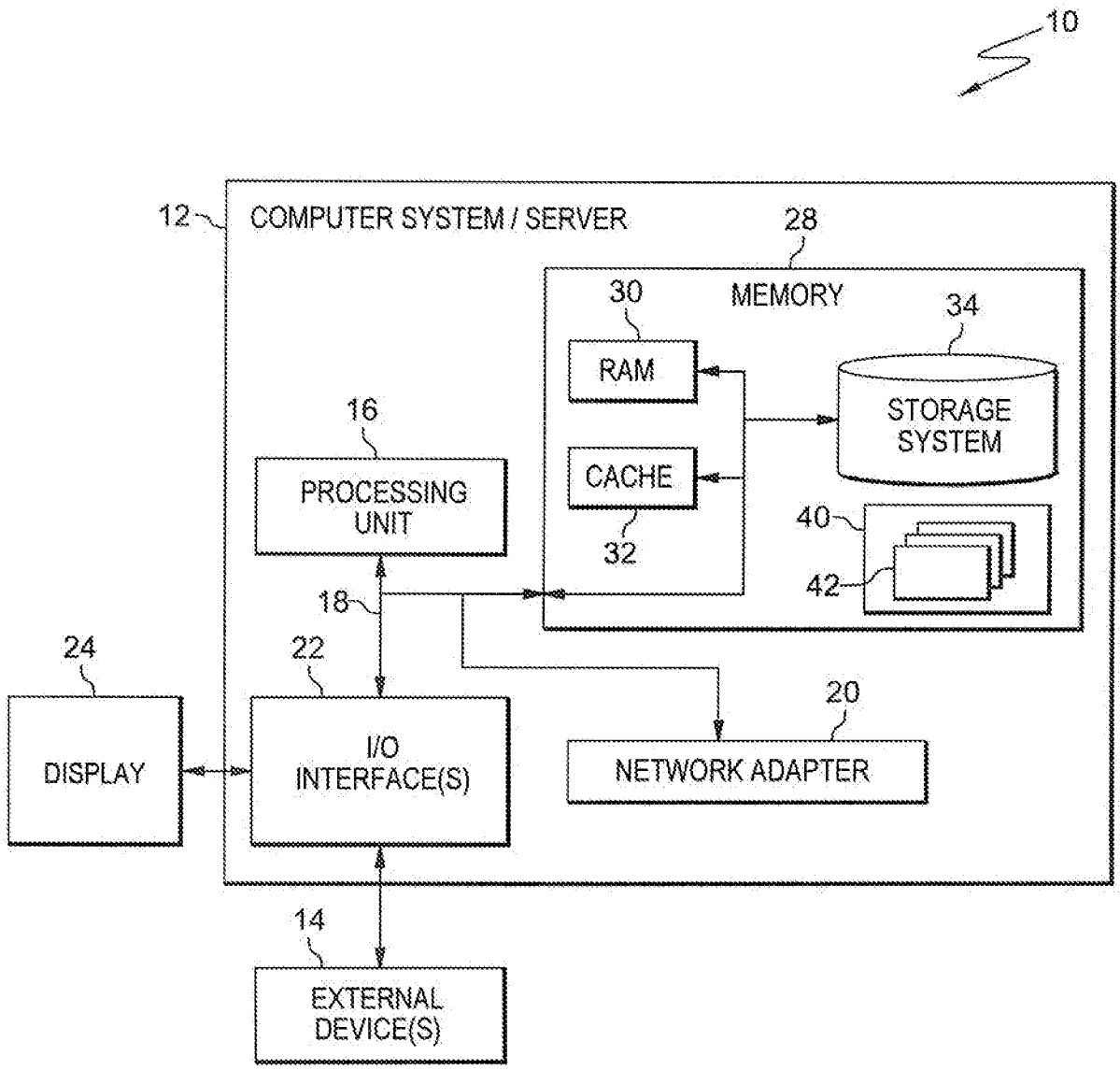
10
12
COMPUTER SYSTEM / SERVER
28
MEMORY
30
RAM
34
STORAGE SYSTEM
16
PROCESSING UNIT
CACHE
32
40
42
18
24
22
20
DISPLAY
I/O INTERFACE(S)
NETWORK ADAPTER
14
EXTERNAL DEVICE(S)

FIG. 9

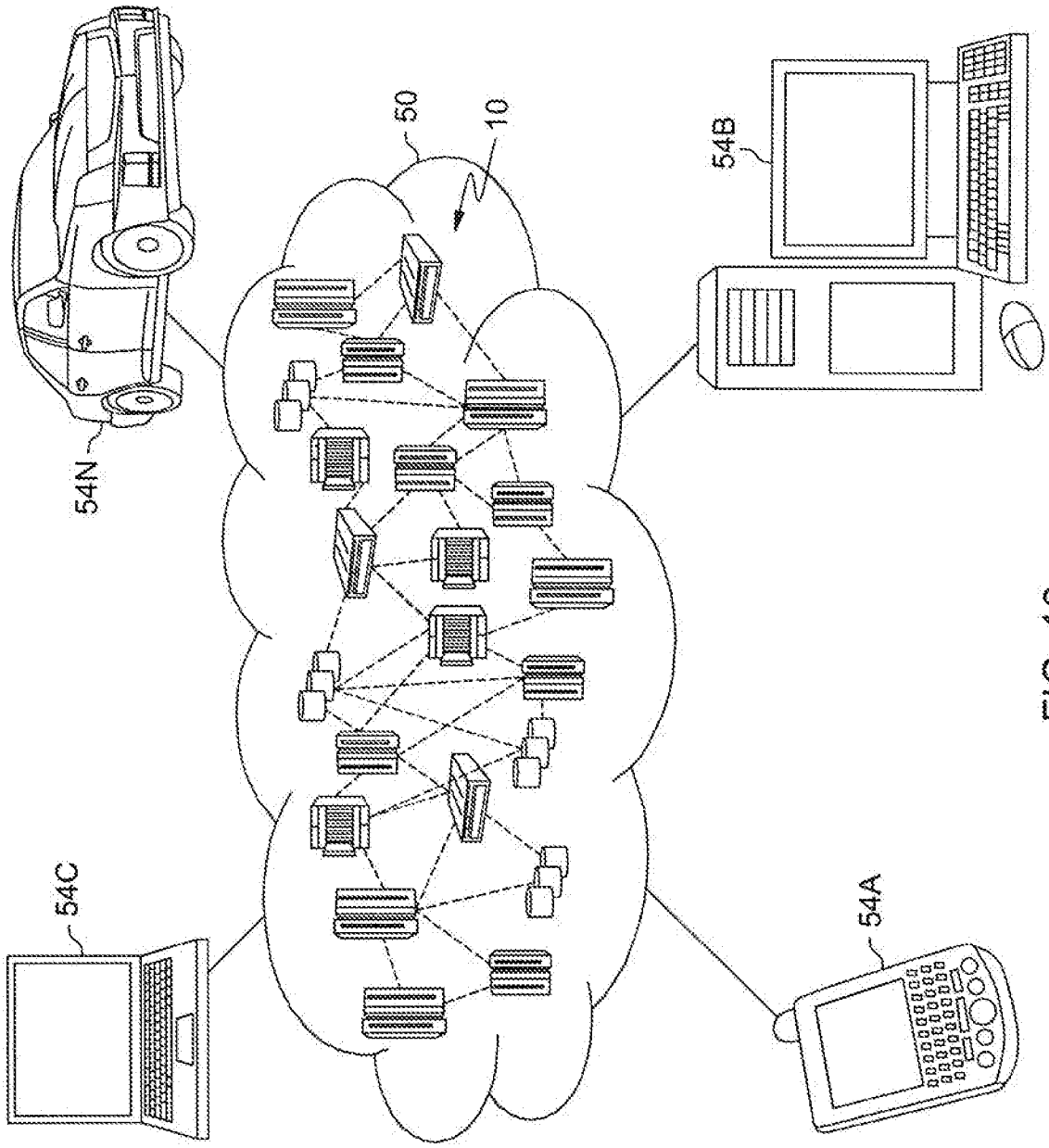

FIG. 10

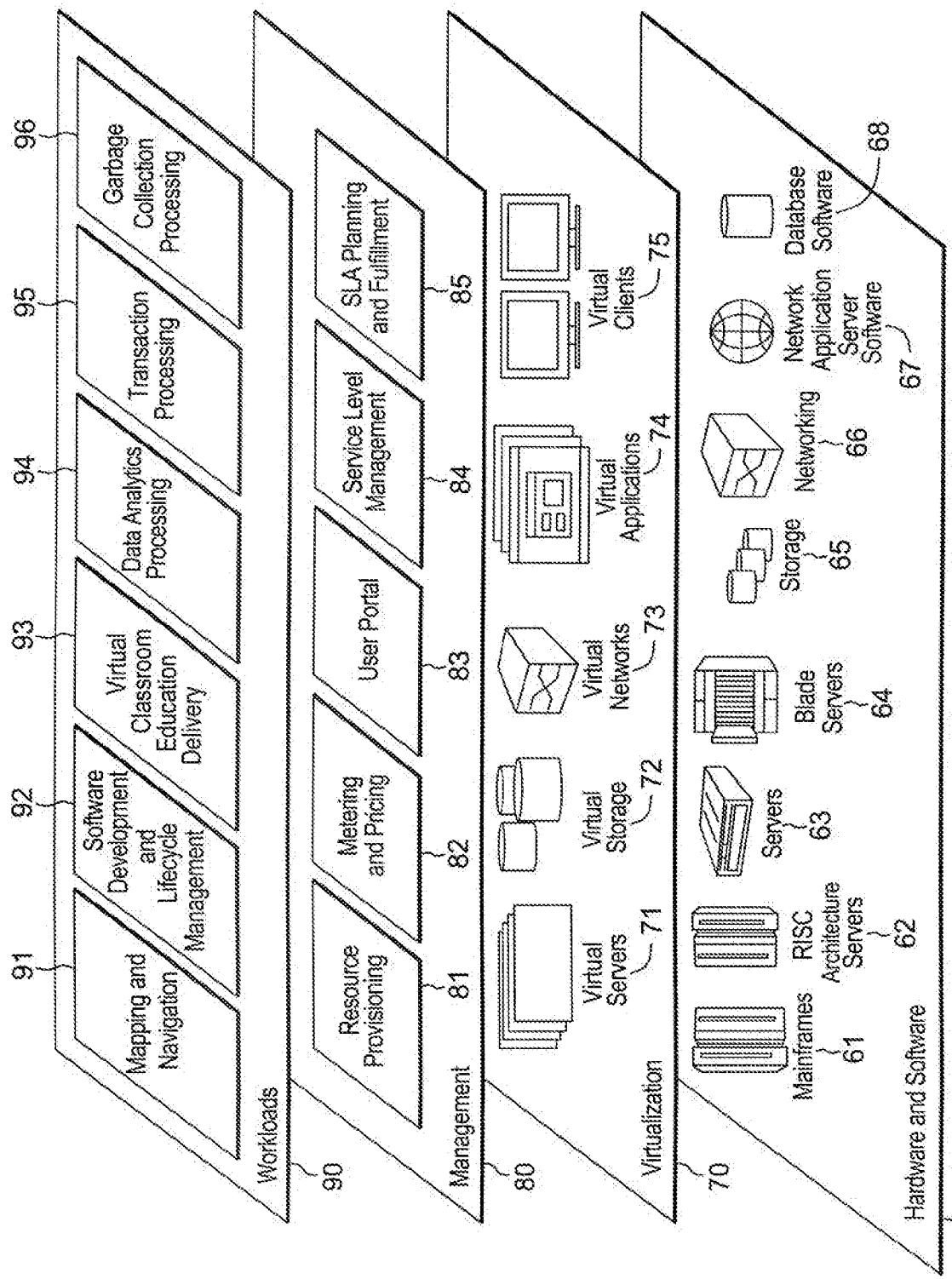
Workloads
90
Mapping and Navigation
91
Software Development and Lifecycle Management
92
Virtual Classroom Education Delivery
93
Data Analytics Processing
94
Transaction Processing
95
Garbage Collection Processing
96
Management
80
Resource Provisioning
81
Metering and Pricing
82
User Portal
83
Service Level Management
84
SLA Planning and Fulfillment
85
Virtualization
70
Virtual Servers
71
Virtual Storage
72
Virtual Networks
73
Virtual Applications
74
Virtual Clients
75
Hardware and Software
60
Mainframes
61
RISC Architecture Servers
62
Servers
63
Blade Servers
64
Storage
65
Networking
66
Network Application Server Software
67
Database Software
68

FIG. 11

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IB2016/053675

A. CLASSIFICATION OF SUBJECT MATTER

G06F 12/02(2006.01)i

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 12/-

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)

CNKI, CNTXT, WPI, EPODOC:garbage, rubbish, collection, modify, update, change, pointer,address

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2014059093 A1 (FUJITSU LTD) 27 February 2014 (2014-02-27) claims 1-15, description, paragraphs [0009]-[0010], [0040]-[0147], figure 7	1-20
A	US 2009063595 A1 (STOODLEY MARK GRAHAM) 05 March 2009 (2009-03-05) the whole document	1-20

☐ 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
27 September 2016	**11 October 2016**
Name and mailing address of the ISA/CN **STATE INTELLECTUAL PROPERTY OFFICE OF THE P.R.CHINA** **6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088** **China** Facsimile No. **(86-10)62019451**	Authorized officer **SONG,Yunyun** Telephone No. **(86-10)62089120**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/IB2016/053675

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
US 2014059093 A1	27 February 2014	JP 2014044472 A	13 March 2014
US 2009063595 A1	05 March 2009	None	