DECIMAL LOAD IMMEDIATE INSTRUCTION

BACKGROUND

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

[0002] Applications executing within a processor of a computing environment control the behavior of the processor. The applications are created using programming languages which are designed to communicate instructions to the processor.

[0003] There are various types of programming languages, and each language may use one or more types of encodings to represent data. For example, COBOL and PL/I programming languages use binary coded decimal (BCD) to encode decimal numbers. With BCD encodings, each decimal digit is represented by a fixed number of bits, such as 4 bits, in one example.

[0004] Decimal numbers may contain up to 31 digits, as well as a sign digit. This represents a 16-byte value in memory. Many computations are performed using small constants. To load a constant into a register, it is loaded from memory. Because of this, instruction latency is incurred, memory bandwidth is used, and data sizes within binary executables are increased.

SUMMARY

[0005] Shortcomings of the prior art are overcome and additional advantages are provided through the provision of a computer program product for facilitating processing in 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 an instruction for execution, the instruction to generate a value for use in processing within the computing environment, the instruction having associated therewith a sign control to be used to provide a sign of the value to be generated by the instruction. The instruction is executed, and the

executing includes obtaining the sign control; shifting an input value provided by the instruction in a specified direction by a selected amount to provide a result; placing the result in a first designated location in a register; and including the sign, based on the sign control, in a second designated location of the register, the result and the sign providing a signed value to be used in processing within the computing environment.

[0006] By using the instruction to generate values, such as constants, values are generated without loading the values from memory. Thus, processing is facilitated and performance is improved within a computing environment. Instruction latency is reduced, less memory bandwidth is used and/or data sizes within binary executables are decreased.

[0007] In one embodiment, the obtaining the sign control includes obtaining the sign control from a field of the instruction, and wherein based on the sign control being a first value, the sign is positive, and based on the sign control being a second value, the sign is negative.

[0008] As a further embodiment, the executing includes obtaining the input value from a field of the instruction, checking validity of the input value, and performing the shifting, based on the checking indicating the input value is valid.

[0009] Further, in one embodiment, the executing includes obtaining the selected amount from a field of the instruction.

[0010] In one example, the specified direction is left. Additionally, the first designated location is a location of a magnitude of a signed packed decimal format in the register.

[0011] Further, the register may be specified using at least one field of the instruction. As an example, the at least one field includes a register field specifying a register number and an extension field specifying an extension value to be appended to the register number.

[0012] As one embodiment, the placing of the result further includes placing zeros in a third designated location of the register. The third designated location includes, e.g., one or more leftmost positions of the register.

[0013] Methods and systems relating to one or more aspects are also described and claimed herein. Further, services relating to one or more aspects are also described and may be claimed herein.

[0014] Additional features and advantages are realized through the techniques described herein. Other embodiments and aspects are described in detail herein and are considered a part of the claimed aspects.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] Preferred embodiments of the present invention will now be disclosed, by way of example only, and with reference to the following drawings:

FIG. 1A depicts one example of a computing environment to incorporate and use one or more embodiments of the present invention;

FIG. 1B depicts further details of the processor of FIG. 1A, in accordance with an embodiment of the present invention;

FIG. 2A depicts another example of a computing environment to incorporate and use one or more embodiments of the present invention;

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

FIG. 3A depicts one example of a Vector Load Immediate Decimal instruction, in accordance with an embodiment of the present invention;

FIG. 3B depicts controls of an immediate field of the Vector Load Immediate Decimal instruction of FIG. 3A, in accordance with an embodiment of the present invention;

FIG. 4 depicts one example of a block diagram of execution of the Vector Load Immediate Decimal instruction, in accordance with an embodiment of the present invention;

FIGS. 5A-5B depict one example of processing associated with the Vector Load Immediate Decimal instruction, in accordance with an embodiment of the present invention;

FIG. 6 depicts one embodiment of a cloud computing environment; and

FIG. 7 depicts one example of abstraction model layers.

DETAILED DESCRIPTION

[0016] One or more embodiments relate to improving processing within a computing environment by providing a capability for generating values (e.g., constants, such as signed packed decimals) without loading the values from memory. In one example, the capability includes an instruction (e.g., an architected machine instruction), referred to herein as a Vector Load Immediate Decimal instruction, used to generate the values. The instruction includes, in one example, a sign control used to provide a sign for a value generated by the instruction. The instruction generates values (e.g., constants, such as signed packed decimals) without loading the values from memory, thereby reducing instruction latency and improving computer performance, reducing memory bandwidth and/or decreasing data sizes within binary executables.

[0017] One example of a computing environment to incorporate and use one or more embodiments of the present invention is described with reference to FIG. 1A. In one example, the computing environment is based on the z/Architecture, offered by International Business Machines Corporation, Armonk, New York. One embodiment of the z/Architecture is described in "z/Architecture Principles of Operation," IBM Publication No. SA22-7832-10, March 2015, which is hereby incorporated herein by reference in its entirety. Z/ARCHITECTURE is a registered trademark of International Business Machines Corporation, Armonk, New York, USA.

[0018] In another example, the computing environment is based on the Power Architecture, offered by International Business Machines Corporation, Armonk, New York. 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.

[0019] The computing environment may also be based on other architectures, including, but not limited to, the Intel x86 architectures. Other examples also exist.

[0020] As shown in FIG. 1A, a computing environment 100 includes, for instance, a node 10 having, e.g., 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 (PC) 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.

[0021] 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 many computing environments, including but not limited to, 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.

[0022] As shown in FIG. 1A, computer system/server 12 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.

[0023] 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.

[0024] 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.

[0025] 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.

[0026] 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.

[0027] 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.

[0028] In one example, processor 16 includes a plurality of functional components used to execute instructions. As depicted in FIG. 1B, these functional components include, for instance, an instruction fetch component 120 to fetch instructions to be executed; an instruction decode unit 122 to decode the fetched instructions and to obtain operands of the decoded instructions; an instruction execute component 124 to execute the decoded instructions; a memory access component 126 to access memory for instruction execution, if necessary; and a write back component 130 to provide the results of the executed instructions. One or more of these components may, in accordance with an embodiment of the present invention, be used to generate values (e.g., constants) 136, as described further below.

[0029] Processor 16 also includes, in one embodiment, one or more registers 140 to be used by one or more of the functional components.

[0030] Another example 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 (CPU) 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 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.

[0031] 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.

[0032] 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 computing environment configured in one architecture to emulate another architecture. For instance, 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.

[0033] 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 z/Architecture processor, 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.

[0034] Further, emulator 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.

[0035] 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.

[0036] 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.

[0037] A guest instruction 250 that is obtained, translated and executed is, for instance, a Vector Load Immediate Decimal instruction described herein. The instruction, which is of one architecture (e.g., the z/Architecture), is fetched from memory, translated and represented as a sequence of native instructions 256 of another architecture (e.g., PowerPC, pSeries, Intel, etc.). These native instructions are then executed.

[0038] Details relating to one embodiment of a Vector Load Immediate Decimal instruction, including fields of the instruction and execution by a processor (either in a native or emulated system) are described herein. In accordance with an embodiment of the present invention, the Vector Load Immediate Decimal instruction is used to generate values (e.g., constants) absent loading the values from memory. This improves computer performance by reducing instruction latency by eliminating memory accesses to obtain the constants, reduces use of memory bandwidth, and decreases the data sizes within binary executables. Thus,

embodiments of the invention are inextricably tied to computer technology, and to the improvement of computer processing.

[0039] In one embodiment, the Vector Load Immediate Decimal instruction is part of a vector facility, which provides, for instance, fixed sized vectors ranging from one to sixteen elements. Each vector includes data which is operated on by vector instructions defined in the facility. In one embodiment, if a vector is made up of multiple elements, then each element is processed in parallel with the other elements. Instruction completion does not occur until processing of all the elements is complete. In other embodiments, the elements are processed partially in parallel and/or sequentially.

[0040] Vector instructions can be implemented as part of various architectures, including, but not limited to, the z/Architecture, the Power Architecture, x86, IA-32, IA-64, etc. Although embodiments described herein are for the z/Architecture, the vector instruction described herein and one or more other aspects may be based on many other architectures. The z/Architecture is only one example.

[0041] In one embodiment in which the vector facility is implemented as part of the z/Architecture, to use the vector registers and instructions, a vector enablement control and a register control in a specified control register (e.g., control register 0) are set to, for instance, one. If the vector facility is installed and a vector instruction is executed without the enablement controls set, a data exception is recognized. If the vector facility is not installed and a vector instruction is executed, an operation exception is recognized.

[0042] In one embodiment, there are 32 vector registers and other types of registers can map to a quadrant of the vector registers. For instance, a register file may include 32 vector registers and each register is 128 bits in length. Sixteen floating point registers, which are 64 bits in length, can overlay the vector registers. Thus, as an example, when floating point register 2 is modified, then vector register 2 is also modified. Other mappings for other types of registers are also possible.

[0043] Vector data appears in storage, for instance, in the same left-to-right sequence as other data formats. Bits of a data format that are numbered 0-7 constitute the byte in the leftmost (lowest-numbered) byte location in storage, bits 8-15 form the byte in the next sequential location, and so on. In a further example, the vector data may appear in storage in another sequence, such as right-to-left.

[0044] One example of a Vector Load Immediate Decimal instruction is described with reference to FIGS. 3A-3B. As shown, the instruction has a plurality of fields, and a field may have a subscript number associated therewith. The subscript number associated with a field of the instruction denotes the operand to which the field applies. For instance, the subscript number 1 associated with vector register V_1 denotes that the register in V_1 includes the first operand, and so forth. A register operand is one register in length, which is, for instance, 128 bits.

[0045] Referring to FIG. 3A, in one embodiment, a Vector Load Immediate Decimal instruction 300 includes opcode fields 302a, 302b indicating a Vector Load Immediate Decimal operation; a vector register field 304 used to designate a vector register (V_1); a first immediate field (I_2) 306; a second immediate field (I_3) 308; and a register extension bit (RXB) field 310, each of which is described below. In one embodiment, the fields are separate and independent from one another; however, in other embodiments, more than one field may be combined. Further information regarding these fields is described below.

[0046] Vector register field 304 is used to indicate a vector register that is to store the first operand, the first operand being a value, such as a constant in a signed packed decimal format. In one example, vector register field 304 is used with RXB field 310 to designate the vector register.

[0047] For instance, RXB field 310 includes the most significant bit for a vector register designated operand. Bits for register designations not specified by the instruction are to be reserved and set to zero. The most significant bit is concatenated, for instance, to the left of the four-bit register designation of the vector register field to create a five-bit vector register designation.

[0048] In one example, the RXB field includes four bits (e.g., bits 0-3), and the bits are defined, as follows:

0 - Most significant bit for the first vector register designation (e.g., in bits 8-11) of the instruction.

1 - Most significant bit for the second vector register designation (e.g., in bits 12-15) of the instruction, if any.

2 - Most significant bit for the third vector register designation (e.g., in bits 16-19) of the instruction, if any.

3 - Most significant bit for the fourth vector register designation (e.g., in bits 32-35) of the instruction, if any.

[0049] Each bit is set to zero or one by, for instance, the assembler depending on the register number. For instance, for registers 0-15, the bit is set to 0; for registers 16-31, the bit is set to 1, etc.

[0050] In one embodiment, each RXB bit is an extension bit for a particular location in an instruction that includes one or more vector registers. For instance, bit 0 of RXB is an extension bit for location 8-11, which is assigned to, e.g., V_1 and so forth. In particular, for vector registers, the register containing the operand is specified using, for instance, a four-bit field of the register field with the addition of its corresponding register extension bit (RXB) as the most significant bit. For instance, if the four bit field is 0110 and the extension bit is 0, then the five bit field 00110 indicates register number 6. In a further embodiment, the RXB field includes additional bits, and more than one bit is used as an extension for each vector or location.

[0051] I_2 field 306 includes the second operand, having, for instance, a number of decimal digits (e.g., 4). Additionally, I_3 field 308 includes a third operand that includes a plurality of controls. For instance, I_3 field 308 includes the following, as depicted in FIG. 3B:

- Sign Control (SC) 320: When bit 0 of I_3 field 308 is zero, the result is positive with a sign code of 1100. When bit 0 of I_3 field 308 is one, the result is negative with a sign code of 1101.

- Shift Amount (SHAMT) 330: Bits 1-3 of I_3 field 308 specify a three bit unsigned binary number specifying the number of digits to shift the second operand in a specified direction (e.g., left).

[0052] Although various fields and registers are described, one or more embodiments of the present invention may use other, additional or less fields or registers, or other sizes of fields or registers, etc. Many variations are possible. For instance, implied registers may be used instead of explicitly specified registers or fields of the instruction. Further, registers other than vector registers may be used. Again, other variations are also possible.

[0053] In operation of the instruction, the four decimal digits (e.g., binary coded decimal (BCD) digits) of the second operand, shifted left by the specified number of digits, and concatenated on the right with the specified sign, form a signed-packed-decimal format number placed at the first operand location.

[0054] In one example, in the signed-packed-decimal format, each byte contains two decimal digits (D), except for the rightmost byte, which contains a sign (S) to the right of a decimal digit. Decimal arithmetic is performed with operands and generates results in the signed-packed-decimal format.

[0055] The signed-packed-decimal operands and results of decimal-arithmetic instructions may be up to, e.g., 16 bytes (31 digits and sign). The editing instructions can fetch as many as 256 decimal digits from one or more decimal numbers of variable length, each in the signed-packed-decimal format.

[0056] Further details regarding operation of the instruction are described with reference to FIG. 4. In one example, the logic of FIG. 4 is performed by a processor.

[0057] Referring to FIG. 4, initially, the four decimal digits (e.g., BCD digits) of the second operand, referred to herein as an input value, are obtained, STEP 400. In one example, the digits of the second operand are checked for validity, INQUIRY 402. If one or more of the digits is invalid, then processing is complete, and an error may be indicated, in one example.

However, if the digits of the second operand are valid, then processing continues with obtaining shift amount 330 and sign control 320 provided by the third operand of the instruction (I_3 308), STEP 404. The four digit second operand is shifted in a specified direction (e.g., left) by the number of digit positions specified by shift amount (SHAMT) control 330 specified in I_3 308 to obtain a result, STEP 406. Zeros are supplied for vacated digit positions.

[0058] In a further embodiment, the shift may be to the right and/or the direction of the shift may be selectable by a control of the instruction. Other variations are possible.

[0059] The result (e.g., the shifted value) is placed in the first operand at a first designated location, STEP 408. For example, the result is placed in the magnitude of the signed packed decimal format of the first operand. Additionally, zeros are placed in selected remaining locations of the first operand, such as the remaining leftmost digits, STEP 410. Further, a sign code of the result is placed in a second designated location of the first operand, STEP 412. As an example, the sign code of the result is determined by sign control 320 of the I_3 field and is placed in the first operand at the location on the right. Thus, the first operand includes the four decimal digits of the second operand shifted left by the specified number of digits and with a sign concatenated on the right.

[0060] As described herein, an instruction is provided which takes a number of BCD digits (e.g., 4) and shifts them to create up to a 31-digit number. The sign digit is controlled by a single bit indicating positive or negative. This allows for a wide range of constants (e.g., in a signed packed decimal format) to be created without requiring loading them from memory. Since the fixed point in BCD numbers can be at any point, the constant is shifted to the proper location. The resulting signed value (e.g., constant) may be used in processing within a computing environment including in computations used for many applications.

[0061] Further details relating to executing an instruction to generate a value to be used in processing within a computing environment are described with reference to FIGS. 5A-5B.

[0062] Referring to FIG. 5A, initially, an instruction is obtained for execution, STEP 500. The instruction is to generate a value (e.g., a constant in a signed packed decimal format) for

use in processing within the computing environment, and it has associated therewith a sign control to be used to provide a sign of the value to be generated by the instruction. The instruction is executed, STEP 502. The executing includes, e.g., obtaining the sign control, STEP 504, and shifting an input value provided by the instruction in a specified direction (e.g., left) by a selected amount to provide a result, STEP 506. Further, the executing includes placing the result in a first designated location in a register, STEP 508, and including the sign, based on the sign control, in a second designated location of the register, the result and the sign providing a signed value to be used in processing within the computing environment, STEP 510.

[0063] As one example, the obtaining the sign control includes obtaining the sign control from a field of the instruction, STEP 512, and wherein based on the sign control being a first value, the sign is positive, and based on the sign control being a second value, the sign is negative.

[0064] In one embodiment, the executing further includes obtaining the input value from a field of the instruction, STEP 514, checking validity of the input value, STEP 516, and performing the shifting, based on the checking indicating the input value is valid, STEP 518. In yet a further embodiment, the executing further includes obtaining the selected amount from a field of the instruction, STEP 520.

[0065] As examples, the first designated location is a location of a magnitude of a signed packed decimal format in the register 530 (FIG. 5B); the register is specified using at least one field of the instruction 532; and the at least one field includes a register field specifying a register number and an extension field specifying an extension value to be appended to the register number 534.

[0066] In one embodiment, the placing of the result further includes placing zeros in a third designated location of the register; the third designated location includes one or more leftmost positions of the register, STEP 536.

[0067] Described herein is a facility for using, e.g., an architected instruction to generate values (e.g., constants) without loading the values from memory, in order to improve computer processing and performance. Although various examples are provided, variations are possible without departing from a spirit of the claimed aspects.

[0068] One or more aspects may relate to cloud computing.

[0069] 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.

[0070] 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.

[0071] 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.

[0072] 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).

[0073] 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).

[0074] 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. One such node is node 10 depicted in FIG. 1A.

[0075] 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.

[0076] Referring now to FIG. 6, illustrative cloud computing environment 50 is depicted. As shown, cloud computing environment 50 comprises one or more 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. 6 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).

[0077] Referring now to FIG. 7, a set of functional abstraction layers provided by cloud computing environment 50 (FIG. 6) is shown. It should be understood in advance that the components, layers, and functions shown in FIG. 7 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; and 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.

[0078] 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.

[0079] 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 instruction processing 96.

[0080] The present invention may be a system, a method, and/or a computer program product at any possible technical detail level of integration. The computer program product may include a computer readable storage medium (or media) having computer readable program instructions thereon for causing a processor to carry out aspects of the present invention.

[0081] The computer readable storage medium can be a tangible device that can retain and store instructions for use by an instruction execution device. The computer readable storage medium may be, for example, but is not limited to, an electronic storage device, a magnetic storage device, an optical storage device, an electromagnetic storage device, a semiconductor storage device, or any suitable combination of the foregoing. A non-exhaustive list of more

specific examples of the computer readable storage medium includes the following: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), a static random access memory (SRAM), a portable compact disc read-only memory (CD-ROM), a digital versatile disk (DVD), a memory stick, a floppy disk, a mechanically encoded device such as punch-cards or raised structures in a groove having instructions recorded thereon, and any suitable combination of the foregoing. A computer readable storage medium, as used herein, is not to be construed as being transitory signals per se, such as radio waves or other freely propagating electromagnetic waves, electromagnetic waves propagating through a waveguide or other transmission media (e.g., light pulses passing through a fiber-optic cable), or electrical signals transmitted through a wire.

[0082] Computer readable program instructions described herein can be downloaded to respective computing/processing devices from a computer readable storage medium or to an external computer or external storage device via a network, for example, the Internet, a local area network, a wide area network and/or a wireless network. The network may comprise copper transmission cables, optical transmission fibers, wireless transmission, routers, firewalls, switches, gateway computers and/or edge servers. A network adapter card or network interface in each computing/processing device receives computer readable program instructions from the network and forwards the computer readable program instructions for storage in a computer readable storage medium within the respective computing/processing device.

[0083] Computer readable program instructions for carrying out operations of the present invention may be assembler instructions, instruction-set-architecture (ISA) instructions, machine instructions, machine dependent instructions, microcode, firmware instructions, state-setting data, configuration data for integrated circuitry, or either source code or object code written in any combination of one or more programming languages, including an object oriented programming language such as Smalltalk, C++, or the like, and procedural programming languages, such as the "C" programming language or similar programming languages. The computer readable program instructions may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the

latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider). In some embodiments, electronic circuitry including, for example, programmable logic circuitry, field-programmable gate arrays (FPGA), or programmable logic arrays (PLA) may execute the computer readable program instructions by utilizing state information of the computer readable program instructions to personalize the electronic circuitry, in order to perform aspects of the present invention.

[0084] 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.

[0085] 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.

[0086] 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.

[0087] 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.

[0088] 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.

[0089] 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.

[0090] 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.

[0091] 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.

[0092] 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.

[0093] 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.

[0094] 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.

[0095] 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.

[0096] 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 processing within a computing environment, the 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 an instruction for execution, the instruction to generate a value for use in processing within the computing environment, the instruction having associated therewith a sign control to be used to provide a sign of the value to be generated by the instruction;

executing the instruction, the executing comprising:

obtaining the sign control;

shifting an input value provided by the instruction in a specified direction by a selected amount to provide a result;

placing the result in a first designated location in a register; and

including the sign, based on the sign control, in a second designated location of the register, the result and the sign providing a signed value to be used in processing within the computing environment.

2. The computer program product of claim 1, wherein the obtaining the sign control comprises obtaining the sign control from a field of the instruction, and wherein based on the sign control being a first value, the sign is positive, and based on the sign control being a second value, the sign is negative.

3. The computer program product of claim 1, wherein the executing further comprises:

obtaining the input value from a field of the instruction;

checking validity of the input value; and

performing the shifting, based on the checking indicating the input value is valid.

4. The computer program product of claim 1, wherein the specified direction is left.

5. The computer program product of claim 1, wherein the executing further comprises obtaining the selected amount from a field of the instruction.

6. The computer program product of claim 1, wherein the first designated location is a location of a magnitude of a signed packed decimal format in the register.

7. The computer program product of claim 1, wherein the register is specified using at least one field of the instruction.

8. The computer program product of claim 7, wherein the at least one field comprises a register field specifying a register number and an extension field specifying an extension value to be appended to the register number.

9. The computer program product of claim 1, wherein the placing of the result further comprises placing zeros in a third designated location of the register.

10. The computer program product of claim 9, wherein the third designated location includes one or more leftmost positions of the register.

11. A computer system for facilitating processing within a computing environment, the computer system comprising:

a memory; and

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

obtaining an instruction for execution, the instruction to generate a value for use in processing within the computing environment, the instruction having associated therewith a sign control to be used to provide a sign of the value to be generated by the instruction;

executing the instruction, the executing comprising:

obtaining the sign control;

shifting an input value provided by the instruction in a specified direction by a selected amount to provide a result;

placing the result in a first designated location in a register; and

including the sign, based on the sign control, in a second designated location of the register, the result and the sign providing a signed value to be used in processing within the computing environment.

12. The computer system of claim 11, wherein the obtaining the sign control comprises obtaining the sign control from a field of the instruction, wherein based on the sign control being a first value, the sign is positive, and based on the sign control being a second value, the sign is negative; wherein the register is specified using at least one field of the instruction; wherein the specified direction is left; and wherein the executing further comprises obtaining the selected amount from a field of the instruction.

13. The computer system of claim 11, wherein the executing further comprises:
obtaining the input value from a field of the instruction;

checking validity of the input value; and

performing the shifting, based on the checking indicating the input value is valid.

14. The computer system of claim 11, wherein the first designated location is a location of a magnitude of a signed packed decimal format in the register.

15. The computer system of claim 11, wherein the placing of the result further comprises placing zeros in a third designated location of the register.

16. A computer-implemented method of facilitating processing within a computing environment, the computer-implemented method comprising:

obtaining, by a processor, an instruction for execution, the instruction to generate a value for use in processing within the computing environment, the instruction having associated therewith a sign control to be used to provide a sign of the value to be generated by the instruction;

executing the instruction, the executing comprising:

obtaining the sign control;

shifting an input value provided by the instruction in a specified direction by a selected amount to provide a result;

placing the result in a first designated location in a register; and

including the sign, based on the sign control, in a second designated location of the register, the result and the sign providing a signed value to be used in processing within the computing environment.

17. The computer-implemented method of claim 16, wherein the obtaining the sign control comprises obtaining the sign control from a field of the instruction, wherein based on the sign control being a first value, the sign is positive, and based on the sign control being a second value, the sign is negative; wherein the register is specified using at least one field of the instruction; wherein the specified direction is left; and wherein the executing further comprises obtaining the selected amount from a field of the instruction.

18. The computer-implemented method of claim 16, wherein the executing further comprises:

obtaining the input value from a field of the instruction;

checking validity of the input value; and

performing the shifting, based on the checking indicating the input value is valid.

19. The computer-implemented method of claim 16, wherein the first designated location is a location of a magnitude of a signed packed decimal format in the register.

20. The computer-implemented method of claim 16, wherein the placing of the result further comprises placing zeros in a third designated location of the register.