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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 March 2017 (30.03.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/052917 Al

(51) International Patent Classification:
G06F 9/30 (2006.01) *G06F 9/38* (2006.01)
G06F 12/08 (2006.01)

(21) International Application Number:
PCT/US2016/048370

(22) International Filing Date:
24 August 2016 (24.08.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
14/866,921 26 September 2015 (26.09.2015) US

(71) Applicant: **INTEL CORPORATION** [US/US]; 2200 Mission College Boulevard, Santa Clara, California 95054 (US).

(72) Inventors: **MISHRA, Asit K.;** 2080 NW Thorncroft DR, APT 1126, Hillsboro, Oregon 97124 (US). **GROCHOWSKI, Edward T.;** 5565 Yale Dr, San Jose, California 95118 (US). **PEARCE, Jonathan D.;** 2111 Northeast 25th Avenue, JF4-350, Hillsboro, Oregon 97214 (US). **MARR, Deborah T.;** 2564 NW Pettygrove St, Portland, Oregon 97210 (US). **COHEN, Ehud;** Dakar St.74, 26430 Kiryat Motskin (IL). **OULD-AHMED-VALL, Elmoustapha;** 5000 W Chandler Blvd, M/S: CH7-401, Chandler, Arizona 85226 (US). **CORBAL SAN ADRIAN, Jesus;** Jordi Girona 1-3 Intel Labs, Barcelona, p3, Nexus II, 08034, 08034 Barcelona (ES). **VALENTINE, Robert;** Rechov Hadganiot 33-5, 36054 Kiryat Tivon (IL). **CHARNEY, Mark J.;** 610 Waltham St, Lexington, Massachusetts 02421 (US). **HUGHES, Christopher J.;** 3543 Druffel PI, Santa Clara, California 95051 (US). **GIRKAR, Milind B.;** 1049 W Olive Ave #3, Sunnyvale, California 94086 (US).

(74) Agent: **VECCHIA, Brent E.;** Vecchia Patent Agent, c/o CPA Global, 900 Second Avenue South, Suite 600, Minneapolis, Minnesota 55402 (US).

(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).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(H))*

Published:

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

WO 2017/052917 A1

(54) Title: DATA ELEMENT COMPARISON PROCESSORS, METHODS, SYSTEMS, AND INSTRUCTIONS

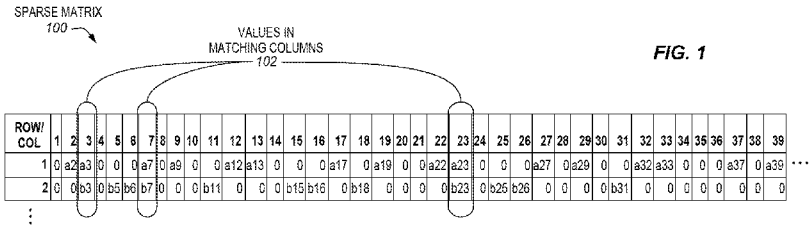


FIG. 1

(57) Abstract: A processor includes a decode unit to decode an instruction that is to indicate a first source packed data operand that is to include at least four data elements, to indicate a second source packed data operand that is to include at least four data elements, and to indicate one or more destination storage locations. The execution unit, in response to the instruction, is to store at least one result mask operand in the destination storage location(s). The at least one result mask operand is to include a different mask element for each corresponding data element in one of the first and second source packed data operands in a same relative position. Each mask element is to indicate whether the corresponding data element in said one of the source packed data operands equals any of the data elements in the other of the source packed data operands.

DATA ELEMENT COMPARISON PROCESSORS, METHODS, SYSTEMS, AND INSTRUCTIONS

BACKGROUND

Technical Field

Embodiments described herein generally relate to processors. In particular, embodiments described herein generally relate to processors to process packed data operands.

Background Information

Many processors have Single Instruction, Multiple Data (SIMD) architectures. In SIMD architectures, a packed data instruction, vector instruction, or SIMD instruction may operate on multiple data elements or multiple pairs of data elements simultaneously or in parallel. The processor may have parallel execution hardware responsive to the packed data instruction to perform the multiple operations simultaneously or in parallel.

Multiple data elements may be packed within one register or memory location as packed data or vector data. In packed data, the bits of the register or other storage location may be logically divided into a sequence of data elements. For example, a 256-bit wide packed data register may have four 64-bit wide data elements, eight 32-bit data elements, sixteen 16-bit data elements, etc. Each of the data elements may represent a separate individual piece of data (e.g., a pixel color, a component of a complex number, etc.), which may be operated upon separately and/or independently of the others.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention may best be understood by referring to the following description and accompanying drawings that are used to illustrate embodiments. In the drawings:

Figure 1 is a block diagram of a portion of an example sparse matrix.

Figure 2 illustrates a compressed sparse row representation of a subset of the columns of rows 1 and 2 of the sparse matrix of **Figure 1.**

Figure 3 is a block diagram of an embodiment of a processor that is operative to perform an embodiment of a data element comparison instruction.

Figure 4 is a block flow diagram of an embodiment of a method of performing an embodiment of a data element comparison instruction.

Figure 5 is a block diagram of a first example embodiment of a data element comparison operation.

Figure 6 is a block diagram of a second example embodiment of a data element comparison operation.

Figure 7 is a block diagram of a third example embodiment of a data element comparison operation.

Figure 8 is a block diagram of a fourth example embodiment of a data element comparison operation.

Figure 9 is a block diagram of an example of a masked data element consolidate operation.

Figure 10 is a block diagram of an example embodiment of a suitable set of packed data operation mask registers.

Figure 11 is a block diagram of an example embodiment of a suitable set of packed data registers.

Figures 12A-C are block diagrams illustrating a generic vector friendly instruction format and instruction templates thereof, according to embodiments of the invention.

Figure 13A-B is a block diagram illustrating an exemplary specific vector friendly instruction format and opcode field, according to embodiments of the invention.

Figure 14A-D is a block diagram illustrating an exemplary specific vector friendly instruction format and fields thereof, according to embodiments of the invention.

Figure 15 is a block diagram of an embodiment of a register architecture.

Figure 16A is a block diagram illustrating an embodiment of an in-order pipeline and an embodiment of a register renaming out-of-order issue/execution pipeline.

Figure 16B is a block diagram of an embodiment of processor core including a front end unit coupled to an execution engine unit and both coupled to a memory unit.

Figure 17A is a block diagram of an embodiment of a single processor core, along with its connection to the on-die interconnect network, and with its local subset of the Level 2 (L2) cache.

Figure 17B is a block diagram of an embodiment of an expanded view of part of the processor core of Figure 17A.

Figure 18 is a block diagram of an embodiment of a processor that may have more than one core, may have an integrated memory controller, and may have integrated graphics.

Figure 19 is a block diagram of a first embodiment of a computer architecture.

Figure 20 is a block diagram of a second embodiment of a computer architecture.

Figure 21 is a block diagram of a third embodiment of a computer architecture.

Figure 22 is a block diagram of a fourth embodiment of a computer architecture.

Figure 23 is a block diagram of use of a software instruction converter to convert binary instructions in a source instruction set to binary instructions in a target instruction set, according to embodiments of the invention.

DETAILED DESCRIPTION OF EMBODIMENTS

Disclosed herein are data element comparison instructions, processors to execute the instructions, methods performed by the processors when processing or executing the instructions,

and systems incorporating one or more processors to process or execute the instructions. In the following description, numerous specific details are set forth (e.g., specific instruction operations, data formats, processor configurations, microarchitectural details, sequences of operations, etc.). However, embodiments may be practiced without these specific details. In other instances, well-known circuits, structures and techniques have not been shown in detail to avoid obscuring the understanding of the description.

The data element comparison instructions disclosed herein are general-purpose instructions and are not limited to any known use. Rather, these instructions may be used for different purposes and/or in different ways based on the creativity of the programmer, compiler, or the like. In some embodiments, these instructions may be used to process data that is associated with sparse matrices, although the scope of the invention is not so limited. In some embodiments, these instructions may be used to process data associated with a compressed sparse row (CSR) representation, although the scope of the invention is not so limited. To further illustrate certain concepts, specific uses of these instructions to process indices in a CSR format, which may be used to represent indices and values of a sparse matrix, will be described, although it is to be appreciated that this is just one possible use of these instructions. Representatively, this may be useful in data analytics, high performance computing, machine learning, sparse linear algebra problems, and the like. In other embodiments, these instructions may be used to process other types of data besides sparse matrices and/or CSR format data. For example, these instructions may be used to process various different types of data, such as, for example, multimedia data, graphics data, sound data, video data, pixels, text string data, character string data, financial data, other types of integer data, or the like. Moreover, such processing of data may be used for different purposes, such as, for example, to identify duplicate data elements, select duplicate data elements, consolidate duplicate data elements, remove duplicate data elements, alter duplicate data elements, or for various other purposes.

Figure 1 is a block diagram of a portion of an example sparse matrix 100. The matrix generally represents a two-dimensional data structure in which data values are arranged into rows and columns. The data values may also be referred to herein simply as values or data elements. The illustrated example sparse matrix is shown to have at least thirty nine columns and at least two rows, and optionally more. Alternatively, other sparse matrices may have more rows and/or fewer or more columns. The values of the first row are shown as a* values, where the asterisk (*) represents the column number having the value. Similarly, the values of the second row are shown as b* values, where the asterisk (*) represents the column number having the value. For example, the value in row 1 column 7 is a7, the value in row 1 column 23 is a23, the value in row 2 column 15 is bl5, and so

In many different applications it may be desirable to operate on two vectors, such as, for example, two rows of the sparse matrix. For example, this may be performed for sparse vector dot product calculations. Such sparse vector dot product calculations are commonly used in machine learning applications, for example. Examples of such machine learning applications are the kernelized support vector machine (SVM), the open source libSVM, kernelized principal component analysis, and the like. A frequently used kernel in such applications is the squared distance computation pattern, which is also known as the L2-norm between two vectors. The squared distance function, f , ($\| f \|$) between two vectors a and β is represented by Equation 1:

$$\| \alpha - \beta \|^2 = \alpha^2 + \beta^2 - 2\alpha \bullet \beta \qquad \text{Equation 1}$$

The inner-product (·) between the two vectors a and β, which may be sparse vectors, is represented as a dot product calculation as shown in Equation 2:

$$\alpha \bullet \beta = \Sigma \alpha(i) * \beta(i), \ 0 \leq i \leq \min(\text{length}(\alpha), \text{length}(\beta)) \qquad \text{Equation 2}$$

Such sparse vector dot product calculations tend to significantly contribute to the overall computational time of machine learning and other applications. Accordingly, increasing the performance of performing such sparse vector dot product calculations may tend to help improve the performance of machine learning as well as other applications.

Referring again to **Figure 1,** the sparse matrix 100 may be referred to as sparse when a significant number or proportion of the values of the matrix are zero values. Often, such zero values have special mathematical properties, such as, for example, multiplication by zero generates a product of zero, or the like. For example, in the case of multiplication of values in different rows of the same column, such zero values may generate zero valued products, whereas multiplication of two non-zero values may generate non-zero values. By way of example, multiplication of the data elements in rows 1 and 2 of column 2 (i.e., a2*0) generates a product of zero, whereas multiplication of the data elements in rows 1 and 2 of column 3 (i.e., a3*b3) generates a non-zero product. Further, in the specific case of a multiply accumulate or dot product type of calculation, often such zero values may not contribute to the overall accumulation value or dot product.

Accordingly, in these and certain other uses, it may be desired to ignore these zero values of the sparse matrix. In the sparse matrix of this particular example, as shown by reference number 102 there are only three pairs of values from rows 1 and 2 occupying a common column which both include non-zero values. Specifically, this is true for a3 and b3, for a7 and b7, and for a23 and b23. In some embodiments, it may be beneficial to be able to efficiently identify and/or isolate such pairs of values. As will be explained further below, the data element comparison instructions disclosed herein are useful for this purpose, although they are not limited to just this purpose.

source

data

and the second result mask operand 330 may instead optionally be stored in a packed data register in the set of packed data registers 320. For example, the result mask operands may be stored in a different packed data register than those used to store the first and second source packed data operands. Alternatively, a packed data register used for either the first source packed data operand or the second source packed data operand may optionally be reused to store the first and second result mask operands. For example, the instruction may indicate a source/destination packed data register that may implicitly or impliedly be understood by the processor to be used both initially for a source packed data operand and subsequently to store the result mask operands.

Referring again to **Figure** 3, the processor includes a decode unit or decoder 314. The decode unit may receive and decode the data element comparison instruction. The decode unit may output one or more relatively lower-level instructions or control signals 316 (e.g., one or more microinstructions, micro-operations, micro-code entry points, decoded instructions or control signals, etc.), which reflect, represent, and/or are derived from the relatively higher-level data element comparison instruction. In some embodiments, the decode unit may include one or more input structures (e.g., port(s), interconnect(s), an interface) to receive the data element comparison instruction, an instruction recognition and decode logic coupled therewith to recognize and decode the data element comparison instruction, and one or more output structures (e.g., port(s), interconnect(s), an interface) coupled therewith to output the lower-level instructions) or control signal(s). The decode unit may be implemented using various different mechanisms including, but not limited to, microcode read only memories (ROMs), look-up tables, hardware implementations, programmable logic arrays (PLAs), and other mechanisms suitable to implement decode units.

In some embodiments, instead of the data element comparison instruction being provided directly to the decode unit, an instruction emulator, translator, morpher, interpreter, or other instruction conversion module may optionally be used. Various types of instruction conversion modules may be implemented in software, hardware, firmware, or a combination thereof. In some embodiments, the instruction conversion module may be located outside the processor, such as, for example, on a separate die and/or in a memory (e.g., as a static, dynamic, or runtime emulation module). By way of example, the instruction conversion module may receive the data element comparison instruction, which may be of a first instruction set, and may emulate, translate, morph, interpret, or otherwise convert the data element comparison instruction into one or more corresponding intermediate instructions or control signals, which may be of a second different instruction set. The one or more intermediate instructions or control signals of the second instruction set may be provided to a decode unit (e.g., decode unit 314), which may

decode them into one or more lower-level instructions or control signals executable by native hardware of the processor (e.g., one or more execution units).

Referring again to **Figure** 3, the execution unit 318 is coupled with the decode unit 314, the packed data registers 320, and optionally the packed data operation mask registers 332 (e.g., when the result mask operands 328, 330 are to be stored therein). The execution unit may receive the one or more decoded or otherwise converted instructions or control signals 316 that represent and/or are derived from the data element comparison instruction. The execution unit may also receive the first source packed data operand 322 and the second source packed data operand 324. The execution unit may be operative in response to and/or as a result of the data element comparison instruction (e.g., in response to the one or more instructions or control signals decoded from the instruction) to store the first result mask operand 328 and the optional second result mask operand 330 in the one or more destination storage locations 326 indicated by the instruction. In some embodiments, at least one result mask operand (e.g., the first result mask operand 328) may include a different mask element for each corresponding data element in one of the first and second source packed data operands (e.g., the first source packed data operand 322) in a same relative position within the operands. In some embodiments, each mask element may indicate whether the corresponding data element in the aforementioned one of the first and second source packed data operands (e.g., the first result mask operand 328) equals any of the data elements in the other of the first and second source packed data operands (e.g., the second result mask operand 330).

In some embodiments, the first result mask operand 328 may include a different mask element for each corresponding data element in the first source packed data operand 322 in a same relative position within the operands, and each mask element of the first result mask operand 328 may indicate whether the corresponding data element in the first source packed data operand 322 equals any of the data elements in the second source packed data operand 324. In some embodiments, the second result mask operand 330 may include a different mask element for each corresponding data element in the second source packed data operand 330 in a same relative position within the operands, and each mask element of the second result mask operand 330 may indicate whether the corresponding data element in the second source packed data operand 324 equals any of the data elements in the first source packed data operand 322. In some embodiments, each mask element may be a single mask bit. In some embodiments, the result may be any of those shown and described for **Figures** 5-8, although the scope of the invention is not so limited.

The execution unit and/or the processor may include specific or particular logic (e.g., transistors, integrated circuitry, or other hardware potentially combined with firmware

(e.g.,

eight

Figures 15-18.

or more data elements, indicate a second source packed data operand including at least four data elements, or in some cases at least eight or more data elements, and indicate one or more destination storage locations. In some embodiments, the data elements may represent indices corresponding to a CSR representation, although the scope of the invention is not so limited.

At least one result mask operand may be stored in the one or more destination storage locations in response to and/or as a result of the data element comparison instruction, at block 438. The at least one result mask operand may include a different mask element for each corresponding data element in one of the first and second source packed data operands in a same relative position within the operands. Each mask element may indicate whether the corresponding data element in the aforementioned one of the first and second source packed data operands equals any of the data elements in the other of the first and second source packed data operands. In some embodiments, at least two result mask operands are stored. In some embodiments, the two result mask operands may be stored in a single mask register. In other embodiments, the two result mask operands may be stored in two different mask registers. In still other embodiments, the two result mask operands may be stored in a packed data operand, such as, for example, by storing a bit of each of the first and second result mask operands in each data element of the packed data operand.

The illustrated method involves architectural operations (e.g., those visible from a software perspective). In other embodiments, the method may optionally include one or more microarchitectural operations. By way of example, the instruction may be fetched, decoded, scheduled out-of-order, source operands may be accessed, an execution unit may perform microarchitectural operations to implement the instruction, etc. In some embodiments, the microarchitectural operations to implement the instruction may optionally include comparing each data element of the first source packed data operand with each data element of the second source packed data operand. In some embodiments, a crossbar based hardware comparison logic may be used to perform these comparisons.

In some embodiments, the method may optionally be performed during or as part of an algorithm to accelerate sparse vector-sparse vector arithmetic (e.g., a sparse vector-sparse vector dot product calculation), although the scope of the invention is not so limited. In some embodiments, the result mask operands stored in response to the instruction may be used to consolidate or collect together data elements that the result mask operands indicate are matching in the source packed data operands. For example, in some embodiments, the result mask operands may be indicated as a source operand of, and used by, a masked data element consolidation instruction. In other embodiments, the result mask operand(s) may be minimally processed and then resulting result mask operand(s) may be indicated as source operand(s)

‘TO

instructions).

Figure

second

mask

Figure

However, 640 operation relative details and characteristics common similar optionally the all repeating without described, primarily will 640 operation the for characteristic additional and/or different the description, obscuring avoid To 5 Figure of 540 operation the to similarities certain has 640 operation The instruction. comparison element data of embodiment example second a response in performed may that 640 operation comparison element data of embodiment example second a illustrating diagram block 6 Figure below. further discussed will way this in used be may instructions consolidation element data predicated masked such how of details Further tuples. CSR these values corresponding the together consolidate elements mask bits unmasked these may instruction consolidation element data predicated masked the and compared be that tuples CSR for indices matching correspond may register mask masks result the of elements mask bits unmasked the example, For operation. consolidation element data for operand control conditional predicate data, mask, source a use needed, being operand mask result second and first the of processing additional without substantially operands, mask result second and first the of one different each may instruction consolidation element data predicated or masked the of instances two example, way By use. such to limited not invention the of scope the although instruction), VPCOMPRESS examples, (such instruction predicated consolidation element data predicated masked such instruction data, packed predicated or masked for operand mask operation data packed source a use for suitable directly may register mask and/or operand mask result each that embodiments, some In advantage, possible One registers). mask different (e.g., registers mask different stored are operand mask result second and first the 5 Figure in illustrated embodiment the possible. also is convention opposite The stored.

is it result corresponding performed operation corresponding which for operation unmasked represent to (i.e., one binary set bit mask whereas stored, isn't result corresponding and/or performed isn't operation corresponding which for operation out masked represent to (i.e., zero binary cleared mask convention, possible one According destination. stored be not to element data result corresponding the allow and/or performed operation allow not value different second a have may or destination, the in stored be element data result corresponding the allow and performed operation allow to value first a have may bit mask each example, For others the of independently and/or separately controlled conditionally predicated, masked, be may elements data, corresponding of pairs elements data, different on operations granularity element per-data provided be may control conditional predication, masking, embodiments, some In operation. data packed control conditionally to predicate data, mask, used be to select source operand registers

additional and/or different description, obscuring To Figure 5 540 operation the certain similarities has 740 The operation instruction. comparison data element of embodiment example third a to response in performed may that 740 operation comparison element data of embodiment example third a illustrating diagram block a 7 Figure use. such to limited not is invention the of scope the although instruction), VPCOMPRESS for examples, such (e.g., instruction consolidation element data predicated or masked such instruction, data packed masked for operand mask operation data packed source a as use for suitable directly be may operand mask result second the that so [0:15] bits into register mask the of [16:31] bits shift to used be may shift simple a Moreover, such to limited not is invention the of scope the although instruction), VPCOMPRESS example, as, such (e.g., instruction consolidation element data predicated masked such instruction, data packed masked for operand mask operation data packed source a as use for suitable directly be may operand mask result first the significant least the embodiments, some In 64-bits. example, for as, such more, or fewer have may embodiments other although 32-bits, only having shown is register mask the illustration, operand). data packed source second the of case this in (e.g., operand data packed source the of one another to corresponding 16-bits) significant most next the (e.g., register mask single the of portion significant more and operand), data packed source first the of case this in (e.g., operand data packed source the of one to corresponding 16-bits) significant least the (e.g., register mask single the of portion significant least a in case, this In swapped. optionally be may operand mask second and first the of positions the Alternatively, register, mask single the of 16-bits adjacent in stored is 630 operand mask result second the and register, mask single the of 16-bits significant least in stored is 628 operand mask result first the Specifically, 532-2). register mask second the and 532-1 register mask first the (e.g., register mask different a in stored being each of instead 632, register mask single a in stored are operands mask result second and first the that is 5, Figure of embodiment the to relative 9 Figure of embodiment the of difference One 630. operand mask result second a and 628 operand mask result first store and generate to may instruction the (from), decoded 616 signals control or instructions one by controlled (e.g., instruction the to responsive, execution The 618. unit execution an to input be may operand data packed source second and first The 624. operand data packed source second a indicate otherwise or specify may and 622, operand data packed source first a indicate otherwise or specify may instruction The 9. Figure of embodiment the At apparent. clearly otherwise or stated otherwise unless 640, operation the to apply optionally also may thereof, embodiments alternate and variations including 540 operation the of details and characteristics described previously the that appreciated be to

characteristics for the operation 740 will primarily be described, without repeating all the optionally similar or common characteristics and details relative to the operation 540. However, it is to be appreciated that the previously described characteristics and details of the operation 540, including the variations and alternate embodiments thereof, may also optionally apply to the operation 740, unless stated otherwise or otherwise clearly apparent.

As in the embodiment of **Figure 7,** the instruction may specify or otherwise indicate a first source packed data operand 722, and may specify or otherwise indicate a second source packed data operand 724. The first and second source packed data operands may be input to an execution unit 718. The execution unit, responsive to the instruction (e.g., as controlled by one or more instructions or control signals 716 decoded from the instruction), may generate and store a result.

One difference of the embodiment of **Figure 7,** relative to the embodiment of **Figure 5,** is that the execution unit 718 may only generate and store a single result mask operand 728. In some embodiments, the single result mask operand may be stored in a mask register (e.g., a packed data operation mask register). In some embodiments, the single result mask operand may correspond to one of the first and second source packed data operands (e.g., in the illustrated example the first source packed data operand). In some embodiments, the result mask operand 728 and/or mask register 732 may be directly suitable for use as a source packed data operation mask operand for a masked packed data instruction, such as a masked or predicated data element consolidation instruction (e.g., such as, for example, a VPCOMPRESS instruction), although the scope of the invention is not limited to such a use. A different instance of the instruction (with the same opcode) may be performed again to generate the result mask operand for the other source packed data operand.

Figure 8 is a block diagram illustrating a fourth example embodiment of a data element comparison operation 840 that may be performed in response to a fourth example embodiment of a data element comparison instruction. The operation 840 has certain similarities to the operation 540 of **Figure 5.** To avoid obscuring the description, the different and/or additional characteristics for the operation 840 will primarily be described, without repeating all the optionally similar or common characteristics and details relative to the operation 540. However, it is to be appreciated that the previously described characteristics and details of the operation 540, including the variations and alternate embodiments thereof, may also optionally apply to the operation 840, unless stated otherwise or otherwise clearly apparent.

As in the embodiment of **Figure** 8, the instruction may specify or otherwise indicate a first source packed data operand 822, and may specify or otherwise indicate a second source packed data operand 824. The first and second source packed data operands may be input to

Figure 8.

Figures 1-8.

Figures

source

packed

Figure 10

data operation mask registers k1 through k7 (but not kO) may be addressed as a predicate operand to predicate a masked packed data operation. The register kO may be used as a regular source or destination, but may not be encoded as a predicate operand (e.g., if kO is specified it has a "no mask" encoding), although mis is not required.

Figure 11 is a block diagram of an example embodiment of a suitable set of packed data registers 1120. The packed data registers include thirty-two 512-bit packed data registers labeled ZMMO through ZMM31. In the illustrated embodiment, the lower order 256-bits of the lower sixteen registers, namely ZMM0-ZMM15, are aliased or overlaid on respective 256-bit packed data registers labeled YMM0-YMM15, although this is not required. Likewise, in the illustrated embodiment, the lower order 128-bits of the registers YMM0-YMM15 are aliased or overlaid on respective 128-bit packed data registers labeled XMMO-XMM15, although this also is not required. The 512-bit registers ZMMO through ZMM31 are operative to hold 512-bit packed data, 256-bit packed data, or 128-bit packed data. The 256-bit registers YMM0-YMM15 are operative to hold 256-bit packed data or 128-bit packed data. The 128-bit registers XMM0-XMM15 are operative to hold 128-bit packed data. In some embodiments, each of the registers may be used to store either packed floating-point data or packed integer data. Different data element sizes are supported including at least 8-bit byte data, 16-bit word data, 32-bit doubleword, 32-bit single-precision floating point data, 64-bit quadword, and 64-bit double-precision floating point data. In alternate embodiments, different numbers of registers and/or different sizes of registers may be used. In still other embodiments, registers may or may not use aliasing of larger registers on smaller registers and/or may or may not be used to store floating point data.

An instruction set includes one or more instruction formats. A given instruction format defines various fields (number of bits, location of bits) to specify, among other things, the operation to be performed (opcode) and the operand(s) on which that operation is to be performed. Some instruction formats are further broken down though the definition of instruction templates (or subformats). For example, the instruction templates of a given instruction format may be defined to have different subsets of the instruction format's fields (the included fields are typically in the same order, but at least some have different bit positions because there are less fields included) and/or defined to have a given field interpreted differently. Thus, each instruction of an ISA is expressed using a given instruction format (and, if defined, in a given one of the instruction templates of that instruction format) and includes fields for specifying the operation and the operands. For example, an exemplary ADD instruction has a specific opcode and an instruction format that includes an opcode field to specify that opcode and operand fields to select operands (source 1/destination and source2); and an occurrence

Exemplary Instruction Formats

VEX Instruction Format

Figures 13A-13B

Figure 13A

Figure 13B

Figure 13A

Figure 13B

different

Figures 13A-B).

Instruction Templates of Class A

Figures 13A-B.

Figures 13A-B.

augmentation operation types are to be performed (e.g., round 13S2A.1 and data transform 1352A.2 are respectively specified for the no memory access, round type operation 1310 and the no memory access, data transform type operation 1315 instruction templates), while the beta field 13S4 distinguishes which of the operations of the specified type is to be performed. In the no memory access 1305 instruction templates, the scale field 1360, the displacement field 1362A, and the displacement scale filed 1362B are not present.

No-Memory Access Instruction Templates - Full Round Control Type Operation

In the no memory access full round control type operation 1310 instruction template, the beta field 1354 is interpreted as a round control field 1354A, whose contends) provide static rounding. While in the described embodiments of the invention the round control field 1354A includes a suppress all floating point exceptions (SAE) field 1356 and a round operation control field 1358, alternative embodiments may support may encode both these concepts into the same field or only have one or the other of these concepts/fields (e.g., may have only the round operation control field 1358).

SAE field 1356 - its content distinguishes whether or not to disable the exception event reporting; when the SAE field's 1356 content indicates suppression is enabled, a given instruction does not report any kind of floating-point exception flag and does not raise any floating point exception handler.

Round operation control field 1358 - its content distinguishes which one of a group of rounding operations to perform (e.g., Round-up, Round-down, Round-towards-zero and Round-to-nearest). Thus, the round operation control field 1358 allows for the changing of the rounding mode on a per instruction basis. In one embodiment of the invention where a processor includes a control register for specifying rounding modes, the round operation control field's 1350 content overrides that register value.

No Memory Access Instruction Templates - Data Transform Type Operation

In the no memory access data transform type operation 1315 instruction template, the beta field 1354 is interpreted as a data transform field 1354B, whose content distinguishes which one of a number of data transforms is to be performed (e.g., no data transform, swizzle, broadcast).

In the case of a memory access 1320 instruction template of class A, the alpha field 1352 is interpreted as an eviction hint field 1352B, whose content distinguishes which one of the eviction hints is to be used (in Figure 13A, temporal 1352B.1 and non-temporal 1352B.2 are respectively specified for the memory access, temporal 1325 instruction template and the memory access, non-temporal 1330 instruction template), while the beta field 1354 is interpreted as a data manipulation field 1354C, whose content distinguishes which one of a number of data manipulation operations (also known as primitives) is to be performed (e.g., no manipulation;

control

field 1359A allows for the changing of the rounding mode on a per instruction basis. In one embodiment of the invention where a processor includes a control register for specifying rounding modes, the round operation control field's 13S0 content overrides that register value.

In the no memory access, write mask control, VSIZE type operation 1317 instruction template, the rest of the beta field 1354 is interpreted as a vector length field 13S9B, whose content distinguishes which one of a number of data vector lengths is to be performed on (e.g., 128, 256, or 512 byte).

In the case of a memory access 1320 instruction template of class B, part of the beta field 1354 is interpreted as a broadcast field 1357B, whose content distinguishes whether or not the broadcast type data manipulation operation is to be performed, while the rest of the beta field 1354 is interpreted the vector length field 1359B. The memory access 1320 instruction templates include the scale field 1360, and optionally the displacement field 1362A or the displacement scale field 1362B.

With regard to the generic vector friendly instruction format 1300, a full opcode field 1374 is shown including the format field 1340, the base operation field 1342, and the data element width field 1364. While one embodiment is shown where the full opcode field 1374 includes all of these fields, the full opcode field 1374 includes less than all of these fields in embodiments that do not support all of them. The full opcode field 1374 provides the operation code (opcode).

The augmentation operation field 1350, the data element width field 1364, and the write mask field 1370 allow these features to be specified on a per instruction basis in the generic vector friendly instruction format.

The combination of write mask field and data element width field create typed instructions in that they allow the mask to be applied based on different data element widths.

The various instruction templates found within class A and class B are beneficial in different situations. In some embodiments of the invention, different processors or different cores within a processor may support only class A, only class B, or both classes. For instance, a high performance general purpose out-of-order core intended for general-purpose computing may support only class B, a core intended primarily for graphics and/or scientific (throughput) computing may support only class A, and a core intended for both may support both (of course, a core that has some mix of templates and instructions from both classes but not all templates and instructions from both classes is within the purview of the invention). Also, a single processor may include multiple cores, all of which support the same class or in which different cores support different class. For instance, in a processor with separate graphics and general purpose cores, one of the graphics cores intended primarily for graphics and/or scientific computing may support only class A, while one or more of the general purpose cores may be high performance

general purpose cores with out-of-order execution and register renaming intended for general-purpose computing that support only class B. Another processor that does not have a separate graphics core, may include one more general purpose in-order or out-of-order cores that support both class A and class B. Of course, features from one class may also be implemented in the other class in different embodiments of the invention. Programs written in a high level language would be put (e.g., just in time compiled or statically compiled) into a variety of different executable forms, including: 1) a form having only instructions of the class(es) supported by the target processor for execution; or 2) a form having alternative routines written using different combinations of the instructions of all classes and having control flow code that selects the routines to execute based on the instructions supported by the processor which is currently executing the code.

Exemplary Specific Vector Friendly Instruction Format

Figure 14 is a block diagram illustrating an exemplary specific vector friendly instruction format according to embodiments of the invention. **Figure 14** shows a specific vector friendly instruction format 1400 that is specific in the sense that it specifies the location, size, interpretation, and order of the fields, as well as values for some of those fields. The specific vector friendly instruction format 1400 may be used to extend the x86 instruction set, and thus some of the fields are similar or the same as those used in the existing x86 instruction set and extension thereof (e.g., AVX). This format remains consistent with the prefix encoding field, real opcode byte field, MOD R/M field, SIB field, displacement field, and immediate fields of the existing x86 instruction set with extensions. The fields from **Figure 13** into which the fields from **Figure 14** map are illustrated.

It should be understood that, although embodiments of the invention are described with reference to the specific vector friendly instruction format 1400 in the context of the generic vector friendly instruction format 1300 for illustrative purposes, the invention is not limited to the specific vector friendly instruction format 1400 except where claimed. For example, the generic vector friendly instruction format 1300 contemplates a variety of possible sizes for the various fields, while the specific vector friendly instruction format 1400 is shown as having fields of specific sizes. By way of specific example, while the data element width field 1364 is illustrated as a one bit field in the specific vector friendly instruction format 1400, the invention is not so limited (that is, the generic vector friendly instruction format 1300 contemplates other sizes of the data element width field 1364).

The generic vector friendly instruction format 1300 includes the following fields listed below in the order illustrated in **Figure 14A.**

- EVEX Prefix (Bytes 0-3) 1402 - is encoded in a four-byte form.

Format Field 1340 (EVEX Byte 0, bits [7:0]) - the first byte (EVEX Byte 0) is the format field 1340 and it contains 0x62 (the unique value used for distinguishing the vector friendly instruction format in one embodiment of the invention).

The second-fourth bytes (EVEX Bytes 1-3) include a number of bit fields providing specific capability.

REX field 1405 (EVEX Byte 1, bits [7-5]) - consists of a EVEX.R bit field (EVEX Byte 1, bit [7] - R), EVEX.X bit field (EVEX byte 1, bit [6] - X), and 1357BEX byte 1, bit[5] - B). The EVEX.R, EVEX.X, and EVEX.B bit fields provide the same functionality as the corresponding VEX bit fields, and are encoded using Is complement form, i.e. ZMM0 is encoded as 1111B, ZMM15 is encoded as 0000B. Other fields of the instructions encode the lower three bits of the register indexes as is known in the art (rrr, xxx, and bbb), so that Rrrr, Xxxx, and Bbbb may be formed by adding EVEX.R, EVEX.X, and EVEX.B.

REX' field 1310 - this is the first part of the REX' field 1310 and is the EVEX.R' bit field (EVEX Byte 1, bit [4] - R') that is used to encode either the upper 16 or lower 16 of the extended 32 register set. In one embodiment of the invention, this bit, along with others as indicated below, is stored in bit inverted format to distinguish (in the well-known x86 32-bit mode) from the BOUND instruction, whose real opcode byte is 62, but does not accept in the MOD R/M field (described below) the value of 11 in the MOD field; alternative embodiments of the invention do not store this and the other indicated bits below in the inverted format. A value of 1 is used to encode the lower 16 registers. In other words, R'Rrrr is formed by combining EVEX.R', EVEX.R, and the other RRR from other fields.

Opcode map field 1415 (EVEX byte 1, bits [3:0] - mmmm) - its content encodes an implied leading opcode byte (OF, OF 38, or OF 3).

Data element width field 1364 (EVEX byte 2, bit [7] - W) - is represented by the notation EVEX.W. EVEX.W is used to define the granularity (size) of the datatype (either 32-bit data elements or 64-bit data elements).

EVEX.vvvv 1420 (EVEX Byte 2, bits [6:3]-ww> the role of EVEX.ww may include the following: 1) EVEX.vvvv encodes the first source register operand, specified in inverted (Is complement) form and is valid for instructions with 2 or more source operands; 2) EVEX.vvvv encodes the destination register operand, specified in Is complement form for certain vector shifts; or 3) EVEX.vvvv does not encode any operand, the field is reserved and should contain 1111b. Thus, EVEX.vvvv field 1420 encodes the 4 low-order bits of the first source register specifier stored in inverted (Is complement) form. Depending on the instruction, an extra different EVEX bit field is used to extend the specifier size to 32 registers.

EVEX.U 1368 Class field (EVEX byte 2, bit [2]-U) - If EVEX.U = 0, it indicates class A

or EVEX.UO; if EVEX.U = 1, it indicates class B or EVEX.U1 .

Prefix encoding field 1425 (EVEX byte 2, bits [l:0]-pp) - provides additional bits for the base operation field. In addition to providing support for the legacy SSE instructions in the EVEX prefix format, this also has the benefit of compacting the SIMD prefix (rather than requiring a byte to express the SIMD prefix, the EVEX prefix requires only 2 bits). In one embodiment, to support legacy SSE instructions that use a SIMD prefix (66H, F2H, F3H) in both the legacy format and in the EVEX prefix format, these legacy SIMD prefixes are encoded into the SIMD prefix encoding field; and at runtime are expanded into the legacy SIMD prefix prior to being provided to the decoder's PLA (so the PLA can execute both the legacy and EVEX format of these legacy instructions without modification). Although newer instructions could use the EVEX prefix encoding field's content directly as an opcode extension, certain embodiments expand in a similar fashion for consistency but allow for different meanings to be specified by these legacy SIMD prefixes. An alternative embodiment may redesign the PLA to support the 2 bit SIMD prefix encodings, and thus not require the expansion.

Alpha field 1352 (EVEX byte 3, bit [7] - EH; also known as EVEX.EH, EVEX.rs, EVEX.RL, EVEX.write mask control, and EVEX.N; also illustrated with a) - as previously described, this field is context specific.

Beta field 1354 (EVEX byte 3, bits [6:4]-SSS, also known as EVEX.s_{2-o}, EVEX.r_{2-o}, EVEX.rrl, EVEX.LL0, EVEX.LLB; also illustrated with βββ) - as previously described, this field is context specific.

REX' field 1310 - this is the remainder of the REX' field and is the EVEX.V bit field (EVEX Byte 3, bit [3] - V) that may be used to encode either the upper 16 or lower 16 of the extended 32 register set. This bit is stored in bit inverted format. A value of 1 is used to encode the lower 16 registers. In other words, V'VVW is formed by combining EVEX.V, EVEX.vvw.

Write mask field 1370 (EVEX byte 3, bits [2:0]-kkk) - its content specifies the index of a register in the write mask registers as previously described. In one embodiment of the invention, the specific value EVEX.kkk=000 has a special behavior implying no write mask is used for the particular instruction (this may be implemented in a variety of ways including the use of a write mask hardwired to all ones or hardware that bypasses the masking hardware).

Real Opcode Field 1430 (Byte 4) is also known as the opcode byte. Part of the opcode is specified in this field.

MOD R/M Field 1440 (Byte 5) includes MOD field 1442, Reg field 1444, and R/M field 1446. As previously described, the MOD field's 1442 content distinguishes between memory access and non-memory access operations. The role of Reg field 1444 can be summarized

two situations: encoding either the destination register operand or a source register operand, or be treated as an opcode extension and not used to encode any instruction operand. The role of R/M field 1446 may include the following: encoding the instruction operand that references a memory address, or encoding either the destination register operand or a source register operand.

Scale, Index, Base (SIB) Byte (Byte 6) - As previously described, the scale field's 13S0 content is used for memory address generation. SIB.xxx 1454 and SIB.bbb 14S6 - the contents of these fields have been previously referred to with regard to the register indexes Xxxx and Bbbb.

Displacement field 1362A (Bytes 7-10) - when MOD field 1442 contains 10, bytes 7-10 are the displacement field 1362A, and it works the same as the legacy 32-bit displacement (disp32) and works at byte granularity.

Displacement factor field 1362B (Byte 7) - when MOD field 1442 contains 01, byte 7 is the displacement factor field 1362B. The location of this field is that same as that of the legacy x86 instruction set 8-bit displacement (disp8), which works at byte granularity. Since disp8 is sign extended, it can only address between -128 and 137 bytes offsets; in terms of 64 byte cache lines, disp8 uses 8 bits that can be set to only four really useful values -128, -64, 0, and 64; since a greater range is often needed, disp32 is used; however, disp32 requires 4 bytes. In contrast to disp8 and disp32, the displacement factor field 1362B is a reinterpretation of disp8; when using displacement factor field 1362B, the actual displacement is determined by the content of the displacement factor field multiplied by the size of the memory operand access (N). This type of displacement is referred to as disp8*N. This reduces the average instruction length (a single byte of used for the displacement but with a much greater range). Such compressed displacement is based on the assumption that the effective displacement is multiple of the granularity of the memory access, and hence, the redundant low-order bits of the address offset do not need to be encoded. In other words, the displacement factor field 1362B substitutes the legacy x86 instruction set 8-bit displacement. Thus, the displacement factor field 1362B is encoded the same way as an x86 instruction set 8-bit displacement (so no changes in the ModRM/SIB encoding rules) with the only exception that disp8 is overloaded to disp8*N. In other words, there are no changes in the encoding rules or encoding lengths but only in the interpretation of the displacement value by hardware (which needs to scale the displacement by the size of the memory operand to obtain a byte-wise address offset).

Immediate field 1372 operates as previously described.

Full Opcode Field

Figure 14B is a block diagram illustrating the fields of the specific vector friendly instruction format 1400 that make up the full opcode field 1374 according to one embodiment

the invention. Specifically, the full opcode field 1374 includes the format field 1340, the base operation field 1342, and the data element width (W) field 1364. The base operation field 1342 includes the prefix encoding field 142S, the opcode map field 141 5, and the real opcode field 1430.

Register Index Field

Figure 14C is a block diagram illustrating the fields of the specific vector friendly instruction format 1400 that make up the register index field 1344 according to one embodiment of the invention. Specifically, the register index field 1344 includes the REX field 140S, the REX' field 1410, the MODR/M.reg field 1444, the MODR/M.r/m field 1446, the W W field 1420, xxx field 1454, and the bbb field 1456.

Augmentation Operation Field

Figure 14D is a block diagram illustrating the fields of the specific vector friendly instruction format 1400 that make up the augmentation operation field 1350 according to one embodiment of the invention. When the class (U) field 1368 contains 0, it signifies EVEX.U0 (class A 1368A); when it contains 1, it signifies EVEX.U1 (class B 1368B). When U=0 and the MOD field 1442 contains 11 (signifying a no memory access operation), the alpha field 1352 (EVEX byte 3, bit [7] - EH) is interpreted as the rs field 1352A. When the rs field 1352A contains a 1 (round 1352A.1), the beta field 1354 (EVEX byte 3, bits [6:4]- SSS) is interpreted as the round control field 1354A. The round control field 1354A includes a one bit SAE field 1356 and a two bit round operation field 1358. When the rs field 1352A contains a 0 (data transform 1352A.2), the beta field 1354 (EVEX byte 3, bits [6:4]- SSS) is interpreted as a three bit data transform field 1354B. When U=0 and the MOD field 1442 contains 00, 01, or 10 (signifying a memory access operation), the alpha field 1352 (EVEX byte 3, bit [7] - EH) is interpreted as the eviction hint (EH) field 1352B and the beta field 1354 (EVEX byte 3, bits [6:4]- SSS) is interpreted as a three bit data manipulation field 1354C.

When U=l, the alpha field 1352 (EVEX byte 3, bit [7] - EH) is interpreted as the write mask control (Z) field 1352C. When U=l and the MOD field 1442 contains 11 (signifying a no memory access operation), part of the beta field 1354 (EVEX byte 3, bit [4]- S_0) is interpreted as the RL field 1357A; when it contains a 1 (round 1357A.1) the rest of the beta field 1354 (EVEX byte 3, bit [6-5]- S_{2-1}) is interpreted as the round operation field 1359A, while when the RL field 1357A contains a 0 (VSIZE 1357.A2) the rest of the beta field 1354 (EVEX byte 3, bit [6-5]- S_2. 1) is interpreted as the vector length field 1359B (EVEX byte 3, bit [6-5]- Li^). When U=l and the MOD field 1442 contains 00, 01, or 10 (signifying a memory access operation), the beta field 1354 (EVEX byte 3, bits [6:4]- SSS) is interpreted as the vector length field 1359B (EVEX byte 3, bit [6-5]- L_{i}.o) and the broadcast field 1357B (EVEX byte 3, bit [4]-

Exemplary Register Architecture

Figure 15 is a block diagram of a register architecture 1500 according to one embodiment of the invention. In the embodiment illustrated, there are 32 vector registers IS10 that are 512 bits wide; these registers are referenced as zmmO through zmm31. The lower order 256 bits of the lower 16 zmm registers are overlaid on registers ymmO-16. The lower order 128 bits of the lower 16 zmm registers (the lower order 128 bits of the ymm registers) are overlaid on registers xmm0-15. The specific vector friendly instruction format 1400 operates on these overlaid register file as illustrated in the below tables.

Adjustable Vector Length	Class	Operations	Registers
Instruction Templates that do not include the vector length field 1359B	A (Figure 13A; U=0)	1310, 1315, 1325, 1330	zmm registers (the vector length is 64 byte)
	B (Figure 13B; U=1)	1312	zmm registers (the vector length is 64 byte)
Instruction templates that do include the vector length field 1359B	B (Figure 13B; U=1)	1317, 1327	zmm, ymm, or xmm registers (the vector length is 64 byte, 32 byte, or 16 byte) depending on the vector length field 1359B

In other words, the vector length field 1359B selects between a maximum length and one or more other shorter lengths, where each such shorter length is half the length of the preceding length; and instructions templates without the vector length field 1359B operate on the maximum vector length. Further, in one embodiment, the class B instruction templates of the specific vector friendly instruction format 1400 operate on packed or scalar single/double-precision floating point data and packed or scalar integer data. Scalar operations are operations performed on the lowest order data element position in an zmm/ymm/xmm register; the higher order data element positions are either left the same as they were prior to the instruction or zeroed depending on the embodiment.

Write mask registers 1515 - in the embodiment illustrated, there are 8 write mask registers (kO through k7), each 64 bits in size. In an alternate embodiment, the write mask registers 1515 are 16 bits in size. As previously described, in one embodiment of the invention, the vector mask register kO cannot be used as a write mask; when the encoding that would normally indicate kO is used for a write mask, it selects a hardwired write mask of OxFFFF, effectively disabling write masking for that instruction.

General-purpose registers 1S2S - in the embodiment illustrated, there are sixteen 64-bit general-purpose registers that are used along with the existing x86 addressing modes to address memory operands. These registers are referenced by the names RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, and R8 through R15.

Scalar floating point stack register file (x87 stack) 1545, on which is aliased the MMX packed integer flat register file 1550 - in the embodiment illustrated, the x87 stack is an eight-element stack used to perform scalar floating-point operations on 32/64/80-bit floating point data using the x87 instruction set extension; while the MMX registers are used to perform operations on 64-bit packed integer data, as well as to hold operands for some operations performed between the MMX and XMM registers.

Alternative embodiments of the invention may use wider or narrower registers. Additionally, alternative embodiments of the invention may use more, less, or different register files and registers.

Exemplary Core Architectures, Processors, and Computer Architectures

Processor cores may be implemented in different ways, for different purposes, and in different processors. For instance, implementations of such cores may include: 1) a general purpose in-order core intended for general-purpose computing; 2) a high performance general purpose out-of-order core intended for general-purpose computing; 3) a special purpose core intended primarily for graphics and/or scientific (throughput) computing. Implementations of different processors may include: 1) a CPU including one or more general purpose in-order cores intended for general-purpose computing and/or one or more general purpose out-of-order cores intended for general-purpose computing; and 2) a coprocessor including one or more special purpose cores intended primarily for graphics and/or scientific (throughput). Such different processors lead to different computer system architectures, which may include: 1) the coprocessor on a separate chip from the CPU; 2) the coprocessor on a separate die in the same package as a CPU; 3) the coprocessor on the same die as a CPU (in which case, such a coprocessor is sometimes referred to as special purpose logic, such as integrated graphics and/or scientific (throughput) logic, or as special purpose cores); and 4) a system on a chip that may include on the same die the described CPU (sometimes referred to as the application core(s) or application processors)), the above described coprocessor, and additional functionality. Exemplary core architectures are described next, followed by descriptions of exemplary processors and computer architectures.

Exemplary Core Architectures

In-order and out-of-order core block diagram

Figure 16A is a block diagram illustrating both an exemplary in-order pipeline and an

1656

cache

Specific Exemplary In Order Core Architecture

Figures 17A

architecture, which core would be one of several logic blocks (including other cores of the same type and/or different types) in a chip. The logic blocks communicate through a high-bandwidth interconnect network (e.g., a ring network) with some fixed function logic, memory I/O interfaces, and other necessary I/O logic, depending on the application.

Figure 17A is a block diagram of a single processor core, along with its connection to the on-die interconnect network 1702 and with its local subset of the Level 2 (L2) cache 1704, according to embodiments of the invention. In one embodiment, an instruction decoder 1700 supports the x86 instruction set with a packed data instruction set extension. An L I cache 1706 allows low-latency accesses to cache memory into the scalar and vector units. While in one embodiment (to simplify the design), a scalar unit 1708 and a vector unit 1710 use separate register sets (respectively, scalar registers 11712 and vector registers 1714) and data transferred between them is written to memory and then read back in from a level 1 (LI) cache 1706, alternative embodiments of the invention may use a different approach (e.g., use a single register set or include a communication path that allow data to be transferred between the two register files without being written and read back).

The local subset of the L2 cache 1704 is part of a global L2 cache that is divided into separate local subsets, one per processor core. Each processor core has a direct access path to its own local subset of the L2 cache 1704. Data read by a processor core is stored in its L2 cache subset 1704 and can be accessed quickly, in parallel with other processor cores accessing their own local L2 cache subsets. Data written by a processor core is stored in its own L2 cache subset 1704 and is flushed from other subsets, if necessary. The ring network ensures coherency for shared data. The ring network is bi-directional to allow agents such as processor cores, L2 caches and other logic blocks to communicate with each other within the chip. Each ring data¬ path is 1012-bits wide per direction.

Figure 17B is an expanded view of part of the processor core in **Figure 17A** according to embodiments of the invention. **Figure 17B** includes an L I data cache 1706A part of the L I cache 1704, as well as more detail regarding the vector unit 1710 and the vector registers 1714. Specifically, the vector unit 1710 is a 16-wide vector processing unit (VPU) (see the 16-wide ALU 1728), which executes one or more of integer, single-precision float, and double-precision float instructions. The VPU supports swizzling the register inputs with swizzle unit 1720, numeric conversion with numeric convert units 1722A-B, and replication with replication unit 1724 on the memory input. Write mask registers 1726 allow predicating resulting vector writes.

Processor with integrated memory controller and graphics

Figure 18 is a block diagram of a processor 1800 that may have more than one core, may have an integrated memory controller, and may have integrated graphics according to

embodiments of the invention. The solid lined boxes in **Figure 18** illustrate a processor 1800 with a single core 1802A, a system agent 1810, a set of one or more bus controller units 1816, while the optional addition of the dashed lined boxes illustrates an alternative processor 1800 with multiple cores 1802A-N, a set of one or more integrated memory controller unit(s) 1814 in the system agent unit 1810, and special purpose logic 1808.

Thus, different implementations of the processor 1800 may include: 1) a CPU with the special purpose logic 1808 being integrated graphics and/or scientific (throughput) logic (which may include one or more cores), and the cores 1802A-N being one or more general purpose cores (e.g., general purpose in-order cores, general purpose out-of-order cores, a combination of the two); 2) a coprocessor with the cores 1802A-N being a large number of special purpose cores intended primarily for graphics and/or scientific (throughput); and 3) a coprocessor with the cores 1802A-N being a large number of general purpose in-order cores. Thus, the processor 1800 may be a general-purpose processor, coprocessor or special-purpose processor, such as, for example, a network or communication processor, compression engine, graphics processor, GPGPU (general purpose graphics processing unit), a high-throughput many integrated core (MIC) coprocessor (including 30 or more cores), embedded processor, or the like. The processor may be implemented on one or more chips. The processor 1800 may be a part of and/or may be implemented on one or more substrates using any of a number of process technologies, such as, for example, BiCMOS, CMOS, or NMOS.

The memory hierarchy includes one or more levels of cache within the cores, a set or one or more shared cache units 1806, and external memory (not shown) coupled to the set of integrated memory controller units 1814. The set of shared cache units 1806 may include one or more mid-level caches, such as level 2 (L2), level 3 (L3), level 4 (L4), or other levels of cache, a last level cache (LLC), and/or combinations thereof. While in one embodiment a ring based interconnect unit 1812 interconnects the integrated graphics logic 1808, the set of shared cache units 1806, and the system agent unit 1810/integrated memory controller unites) 1814, alternative embodiments may use any number of well-known techniques for interconnecting such units. In one embodiment, coherency is maintained between one or more cache units 1806 and cores 1802-A-N.

In some embodiments, one or more of the cores 1802A-N are capable of multi-threading. The system agent 1810 includes those components coordinating and operating cores 1802A-N. The system agent unit 1810 may include for example a power control unit (PCU) and a display unit. The PCU may be or include logic and components needed for regulating the power state of the cores 1802A-N and the integrated graphics logic 1808. The display unit is for driving one or more externally connected displays.

The cores 1802A-N may be homogeneous or heterogeneous in terms of architecture instruction set; that is, two or more of the cores 1802A-N may be capable of execution the same instruction set, while others may be capable of executing only a subset of that instruction set or a different instruction set.

Exemplary Computer Architectures

Figures 19-21 are block diagrams of exemplary computer architectures. Other system designs and configurations known in the arts for laptops, desktops, handheld PCs, personal digital assistants, engineering workstations, servers, network devices, network hubs, switches, embedded processors, digital signal processors (DSPs), graphics devices, video game devices, set-top boxes, micro controllers, cell phones, portable media players, hand held devices, and various other electronic devices, are also suitable. In general, a huge variety of systems or electronic devices capable of incorporating a processor and/or other execution logic as disclosed herein are generally suitable.

Referring now to **Figure 19,** shown is a block diagram of a system 1900 in accordance with one embodiment of the present invention. The system 1900 may include one or more processors 1910, 1915, which are coupled to a controller hub 1920. In one embodiment the controller hub 1920 includes a graphics memory controller hub (GMCH) 1990 and an Input/Output Hub (IOH) 1950 (which may be on separate chips); the GMCH 1990 includes memory and graphics controllers to which are coupled memory 1940 and a coprocessor 1945; the IOH 1950 is couples input/output (I/O) devices 1960 to the GMCH 1990. Alternatively, one or both of the memory and graphics controllers are integrated within the processor (as described herein), the memory 1940 and the coprocessor 1945 are coupled directly to the processor 1910, and the controller hub 1920 in a single chip with the IOH 1950.

The optional nature of additional processors 1915 is denoted in **Figure 19** with broken lines. Each processor 1910, 1915 may include one or more of the processing cores described herein and may be some version of the processor 1800.

The memory 1940 may be, for example, dynamic random access memory (DRAM), phase change memory (PCM), or a combination of the two. For at least one embodiment, the controller hub 1920 communicates with the processor(s) 1910, 1915 via a multi-drop bus, such as a frontside bus (FSB), point-to-point interface such as QuickPath Interconnect (QPI), or similar connection 1995.

In one embodiment, the coprocessor 1945 is a special-purpose processor, such as, for example, a high-throughput MIC processor, a network or communication processor, compression engine, graphics processor, GPGPU, embedded processor, or the like. In one embodiment, controller hub 1920 may include an integrated accelerator.

placed

direct

change

memory (PCM), magnetic or optical cards, or any other type of media suitable for storing electronic instructions.

Accordingly, embodiments of the invention also include non-transitory, tangible machine-readable media containing instructions or containing design data, such as Hardware Description Language (HDL), which defines structures, circuits, apparatuses, processors and/or system features described herein. Such embodiments may also be referred to as program products.

Emulation (including binary translation, code morphing, etc)

In some cases, an instruction converter may be used to convert an instruction from a source instruction set to a target instruction set. For example, the instruction converter may translate (e.g., using static binary translation, dynamic binary translation including dynamic compilation), morph, emulate, or otherwise convert an instruction to one or more other instructions to be processed by the core. The instruction converter may be implemented in software, hardware, firmware, or a combination thereof. The instruction converter may be on processor, off processor, or part on and part off processor.

Figure 23 is a block diagram contrasting the use of a software instruction converter to convert binary instructions in a source instruction set to binary instructions in a target instruction set according to embodiments of the invention. In the illustrated embodiment, the instruction converter is a software instruction converter, although alternatively the instruction converter may be implemented in software, firmware, hardware, or various combinations thereof. **Figure 23** shows a program in a high level language 2302 may be compiled using an x86 compiler 2304 to generate x86 binary code 2306 that may be natively executed by a processor with at least one x86 instruction set core 2316. The processor with at least one x86 instruction set core 2316 represents any processor that can perform substantially the same functions as an Intel processor with at least one x86 instruction set core by compatibly executing or otherwise processing (1) a substantial portion of the instruction set of the Intel x86 instruction set core or (2) object code versions of applications or other software targeted to run on an Intel processor with at least one x86 instruction set core, in order to achieve substantially the same result as an Intel processor with at least one x86 instruction set core. The x86 compiler 2304 represents a compiler that is operable to generate x86 binary code 2306 (e.g., object code) that can, with or without additional linkage processing, be executed on the processor with at least one x86 instruction set core 2316. Similarly, **Figure 23** shows the program in the high level language 2302 may be compiled using an alternative instruction set compiler 2308 to generate alternative instruction set binary code 2310 that may be natively executed by a processor without at least one x86 instruction set core 2314 (e.g., a processor with cores that execute the MIPS instruction set of MIPS Technologies of Sunnyvale, CA and/or that execute the ARM instruction set of ARM Holdings of Sunnyvale,

CA). The instruction converter 2312 is used to convert the x86 binary code 2306 into code that may be natively executed by the processor without an x86 instruction set core 2314. This converted code is not likely to be the same as the alternative instruction set binary code 2310 because an instruction converter capable of this is difficult to make; however, the converted code will accomplish the general operation and be made up of instructions from the alternative instruction set. Thus, the instruction converter 2312 represents software, firmware, hardware, or a combination thereof that, through emulation, simulation or any other process, allows a processor or other electronic device that does not have an x86 instruction set processor or core to execute the x86 binary code 2306.

Components, features, and details described for any of **Figures 1-2** and **5-11** may also optionally apply to any of **Figures 3-4.** Moreover, components, features, and details described for any of the apparatus may also optionally apply to any of the methods, which in embodiments may be performed by and/or with such apparatus. Any of the processors described herein may be included in any of the computer systems disclosed herein (e.g., **Figures 19-23).** In some embodiments, the computer system may include a dynamic random access memory (DRAM). Alternatively, the computer system may include a type of volatile memory that does not need to be refreshed or flash memory. The instructions disclosed herein may be performed with any of the processors shown herein, having any of the microarchitectures shown herein, on any of the systems shown herein. The instructions disclosed herein may have any of the features of the instruction formats shown herein (e.g., in **Figures 12-14).**

In the description and claims, the terms "coupled" and/or "connected," along with their derivatives, may have be used. These terms are not intended as synonyms for each other. Rather, in embodiments, "connected" may be used to indicate that two or more elements are in direct physical and/or electrical contact with each other. "Coupled" may mean mat two or more elements are in direct physical and/or electrical contact with each other. However, "coupled" may also mean that two or more elements are not in direct contact with each other, but yet still co-operate or interact with each other. For example, an execution unit may be coupled with a register and/or a decode unit through one or more intervening components. In the figures, arrows are used to show connections and couplings.

In the description and/or claims, the terms "logic," "unit," "module," or "component," may have been used. Each of these terms may be used to refer to hardware, firmware, software, or various combinations thereof. In example embodiments, each of these terms may refer to integrated circuitry, application specific integrated circuits, analog circuits, digital circuits, programmed logic devices, memory devices including instructions, and the like, and various combinations thereof. In some embodiments, these may include at least some hardware

(e.g.,

transistors, gates, other circuitry components, etc.).

The term "and/or" may have been used. As used herein, the term "and/or" means one or the other or both (e.g., A and/or B means A or B or both A and B).

In the description above, specific details have been set forth in order to provide a thorough understanding of the embodiments. However, other embodiments may be practiced without some of these specific details. The scope of the invention is not to be determined by the specific examples provided above, but only by the claims below. In other instances, well-known circuits, structures, devices, and operations have been shown in block diagram form and/or without detail in order to avoid obscuring the understanding of the description. Where considered appropriate, reference numerals, or terminal portions of reference numerals, have been repeated among the figures to indicate corresponding or analogous elements, which may optionally have similar or the same characteristics, unless specified or clearly apparent otherwise.

Certain operations may be performed by hardware components, or may be embodied in machine-executable or circuit-executable instructions, that may be used to cause and/or result in a machine, circuit, or hardware component (e.g., a processor, potion of a processor, circuit, etc.) programmed with the instructions performing the operations. The operations may also optionally be performed by a combination of hardware and software. A processor, machine, circuit, or hardware may include specific or particular circuitry or other logic (e.g., hardware potentially combined with firmware and/or software) is operative to execute and/or process the instruction and store a result in response to the instruction.

Some embodiments include an article of manufacture (e.g., a computer program product) that includes a machine-readable medium. The medium may include a mechanism that provides, for example stores, information in a form that is readable by the machine. The machine-readable medium may provide, or have stored thereon, an instruction or sequence of instructions, that if and/or when executed by a machine is operative to cause the machine to perform and/or result in the machine performing one or operations, methods, or techniques disclosed herein.

In some embodiments, the machine-readable medium may include a non-transitory machine-readable storage medium. For example, the non-transitory machine-readable storage medium may include a floppy diskette, an optical storage medium, an optical disk, an optical data storage device, a CD-ROM, a magnetic disk, a magneto-optical disk, a read only memory (ROM), a programmable ROM (PROM), an erasable-and-programmable ROM (EPROM), an electrically-erasable-and-programmable ROM (EEPROM), a random access memory (RAM), a static-RAM (SRAM), a dynamic-RAM (DRAM), a Flash memory, a phase-change memory, a phase-change data storage material, a non-volatile memory, a non-volatile data storage device, a non-transitory memory, a non-transitory data storage device, or the like. The non-transitory

non-transitory

second

source packed data operands.

Example 2 includes the processor of Example 1, in which the execution unit, in response to the instruction, is to store two result mask operands in the one or more destination storage locations. The two result mask operands are to include a first result mask operand that is to include a different mask element for each corresponding data element in the first source packed data operand in a same relative position. Each mask element of the first result mask operand to indicate whether the corresponding data element in the first source packed data operand equals any of the data elements in the second source packed data operand. A second result mask operand is to include a different mask element for each corresponding data element in the second source packed data operand in a same relative position. Each mask element of the second result mask operand is to indicate whether the corresponding data element in the second source packed data operand equals any of the data elements in the first source packed data operand.

Example 3 includes the processor of Example 2, in which the one or more destination storage locations comprise a first mask register and a second mask register, and in which the execution unit, in response to the instruction, is to store the first result mask operand in the first mask register and is to store the second result mask operand in the second mask register.

Example 4 includes the processor of Example 2, in which the one or more destination storage locations comprise a single mask register, and in which the execution unit, in response to the instruction, is to store the first result mask operand and the second result mask operand in the single mask register.

Example 5 includes the processor of Example 4, in which the execution unit, in response to the instruction, is to store the first result mask operand in a least significant portion of the single mask register and is to store the second result mask operand in a portion of the single mask register more significant than the least significant portion.

Example 6 includes the processor of Example 1, in which the execution unit, in response to the instruction, is to store both a first result mask operand and a second result mask operand in a packed data register, and in which each data element in the packed data register is to have both a mask element of the first result mask operand and a mask element of the second result mask operand.

Example 7 includes the processor of Example 1, in which the execution unit, in response to the instruction, is to store a single result mask operand in a single mask register.

Example 8 includes the processor of Example 1, in which the execution unit, in response to the instruction, is to store the at least one result mask operand in at least one mask register, and in which an instruction set of the processor includes masked packed data instructions that are operative to indicate the at least one mask register as a storage location for a source mask

each

least

signal

processor (DSP) coupled with the at least one interconnect, an optional display controller coupled with the at least one interconnect, an optional memory controller coupled with the at least one interconnect, an optional wireless modem coupled with the at least one interconnect, an optional image signal processor coupled with the at least one interconnect, an optional Universal Serial Bus (USB) 3.0 compatible controller coupled with the at least one interconnect, an optional Bluetooth 4.1 compatible controller coupled with the at least one interconnect, and an optional wireless transceiver controller coupled with the at least one interconnect.

Example 25 is a processor or other apparatus to perform or operative to perform the method of any one of Examples 13 to 18.

Example 26 is a processor or other apparatus that includes means for performing the method of any one of Examples 13 to 18.

Example 27 is an article of manufacture that includes an optionally non-transitory machine-readable medium, which optionally stores or otherwise provides an instruction, which if and/or when executed by a processor, computer system, electronic device, or other machine, is operative to cause the machine to perform the method of any one of Examples 13 to 18.

Example 28 is a processor or other apparatus substantially as described herein.

Example 29 is a processor or other apparatus that is operative to perform any method substantially as described herein.

Example 30 is a processor or other apparatus to perform (e.g., that has components to perform or that is operative to perform) any data element comparison instruction substantially as described herein.

Example 31 is a computer system or other electronic device that includes a processor having a decode unit to decode instructions of a first instruction set. The processor also has one or more execution units. The electronic device also includes a storage device coupled with the processor. The storage device is to store a first instruction, which may be any of the data element comparison instructions substantially as disclosed herein, and which is to be of a second instruction set. The storage device is also to store instructions to convert the first instruction into one or more instructions of the first instruction set. The one or more instructions of the first instruction set, when performed by the processor, are to cause the processor to store any of the results of the first instruction disclosed herein.

CLAIMS

What is claimed is:

1. A processor comprising:

a decode unit to decode a data element comparison instruction, the data element comparison instruction to indicate a first source packed data operand that is to include at least four data elements, to indicate a second source packed data operand that is to include at least four data elements, and to indicate one or more destination storage locations; and

an execution unit coupled with the decode unit, the execution unit, in response to the data element comparison instruction, to store at least one result mask operand in the one or more destination storage locations, the at least one result mask operand to include a different mask element for each corresponding data element in one of the first and second source packed data operands in a same relative position, each mask element to indicate whether the corresponding data element in said one of the first and second source packed data operands equals any of the data elements in the other of the first and second source packed data operands.

2. The processor of claim 1, wherein the execution unit, in response to the instruction, is to store two result mask operands in the one or more destination storage locations, the two result mask operands to include:

a first result mask operand that is to include a different mask element for each corresponding data element in the first source packed data operand in a same relative position, each mask element of the first result mask operand to indicate whether the corresponding data element in the first source packed data operand equals any of the data elements in the second source packed data operand; and

a second result mask operand that is to include a different mask element for each corresponding data element in the second source packed data operand in a same relative position, each mask element of the second result mask operand to indicate whether the corresponding data element in the second source packed data operand equals any of the data elements in the first source packed data operand.

3. The processor of claim 2, wherein the one or more destination storage locations comprise a first mask register and a second mask register, and wherein the execution unit, in response to the instruction, is to store the first result mask operand in the first mask register and is to store the second result mask operand in the second mask register.

4. The processor of claim 2, wherein the one or more destination storage locations comprise a single mask register, and wherein the execution unit, in response to the instruction, is to store the first result mask operand and the second result mask operand in the single mask register.

more

destination storage locations; and

storing at least one result mask operand in the one or more destination storage locations in response to the data element comparison instruction, the at least one result mask operand including a different mask element for each corresponding data element in one of the first and second source packed data operands in a same relative position, each mask element indicating whether the corresponding data element in said one of the first and second source packed data operands equals any of the data elements in the other of the first and second source packed data operands.

14. The method of claim 13, wherein storing comprises:

storing a first result mask operand in the one or more destination storage locations, the first result mask operand including a different mask element for each corresponding data element in the first source packed data operand in a same relative position, each mask element of the first result mask operand indicating whether the corresponding data element in the first source packed data operand equals any of the data elements in the second source packed data operand; and

storing a second result mask operand in the one or more destination storage locations, the second result mask operand including a different mask element for each corresponding data element in the second source packed data operand in a same relative position, each mask element of the second result mask operand indicating whether the corresponding data element in the second source packed data operand equals any of the data elements in the first source packed data operand.

15. The method of claim 14, wherein storing the first result mask operand comprises storing the first result mask operand in a first mask register, and wherein storing the second result mask operand comprises storing the second result mask operand in a second mask register.

16. The method of claim 14, wherein storing the first result mask operand and storing the second result mask operand comprises storing both the first and second result mask operands in a single mask register.

17. The method of claim 13, wherein storing the at least one result mask operand in the one or more destination storage locations comprises storing both a first result mask operand and a second result mask operand in a result packed data operand.

18. The method of claim 13, further comprising receiving a masked packed data instruction indicating the at least one result mask operand as a predicate operand.

19. A system to process instructions comprising:

an interconnect;

a processor coupled with the interconnect, the processor to receive a data element comparison instruction, the instruction indicate a first source packed data operand that is

include at least four data elements, to indicate a second source packed data operand that is to include at least four data elements, and to indicate one or more destination storage locations, the processor, in response to the instruction, to store at least one result mask operand in the one or more destination storage locations, the at least one result mask operand to include a different mask bit for each corresponding data element in one of the first and second source packed data operands in a same relative position, each mask bit to indicate whether the corresponding data element in said one of the first and second source packed data operands equals any of the data elements in the other of the first and second source packed data operands; and

a dynamic random access memory (DRAM) coupled with the interconnect, the DRAM storing a sparse vector-sparse vector arithmetic algorithm, the sparse vector-sparse vector arithmetic algorithm to include a masked data element consolidation instruction that is to indicate the at least one result mask operand as a source operand to mask a data element consolidation operation.

20. The system of claim 19, wherein the execution unit in response to the instruction is to store two result mask operands each corresponding to a different one of the source packed data operands, wherein the two result mask operands are to be stored in at least one mask register.

21. An article of manufacture comprising a non-transitory machine-readable storage medium, the non-transitory machine-readable storage medium storing a data element comparison instruction,

the instruction to indicate a first source packed data operand that is to include at least four data elements, to indicate a second source packed data operand that is to include at least four data elements, and to indicate one or more destination storage locations, and the instruction if executed by a machine is to cause the machine to perform operations comprising:

store a first result mask operand in the one or more destination storage locations, the first result mask operand to include a different mask bit for each corresponding data element in the first source packed data operand in a same relative position, each mask bit to indicating whether the corresponding data element in the first source packed data operand equals any of the data elements in the second source packed data operand.

22. The article of manufacture of claim 21, wherein the instruction if executed by a machine is to cause the machine to perform operations comprising store a second result mask operand in the one or more destination storage locations, and wherein the one or more destination storage locations comprise at least one mask register, and wherein the first and second result mask operands together have no more mask bits than a number of data elements in the first and second source packed data operands.

23. An apparatus comprising means for performing the method of any one of claims 13 to 18.

24. An article of manufacture comprising a machine-readable medium, which stores an instruction, which if executed by a machine, is operative to cause the machine to perform the method of any one of claims 13 to 18.

25. An electronic device comprising an interconnect, the processor of any one of claims 1 to 8 coupled with the interconnect, and a dynamic random access memory (DRAM) coupled with the interconnect.

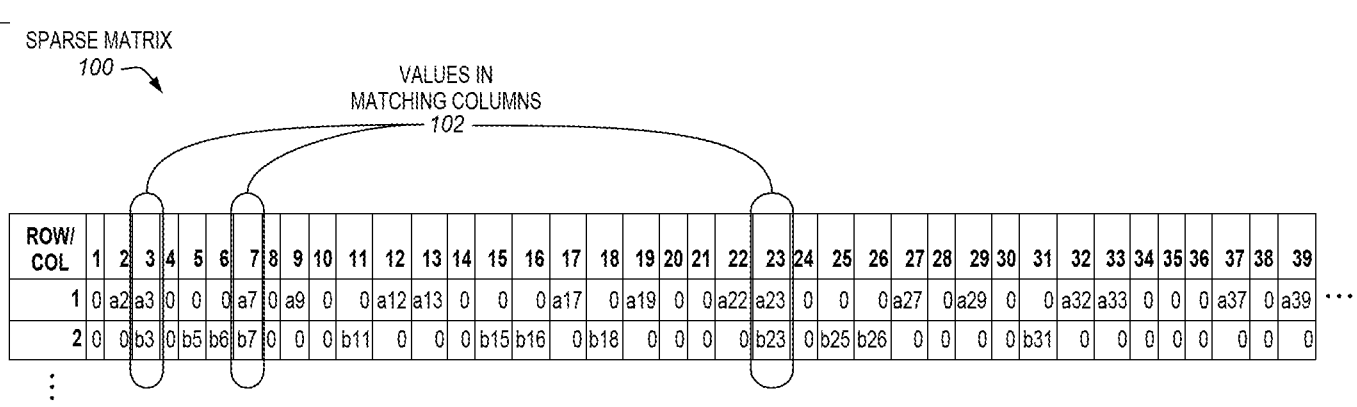

SPARSE MATRIX
100
VALUES IN
MATCHING COLUMNS
102
ROW/
COL
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
1 0 a2 a3 0 0 0 a7 0 a9 0 0 a12 a13 0 0 0 a17 0 a19 0 0 a22 a23 0 0 0 a27 0 a29 0 0 a32 a33 0 0 0 a37 0 a39 ...
2 0 0 b3 0 b5 b6 b7 0 0 0 b11 0 0 0 b15 b16 0 b18 0 0 0 0 b23 0 b25 b26 0 0 0 0 b31 0 0 0 0 0 0 0 0

FIG. 1

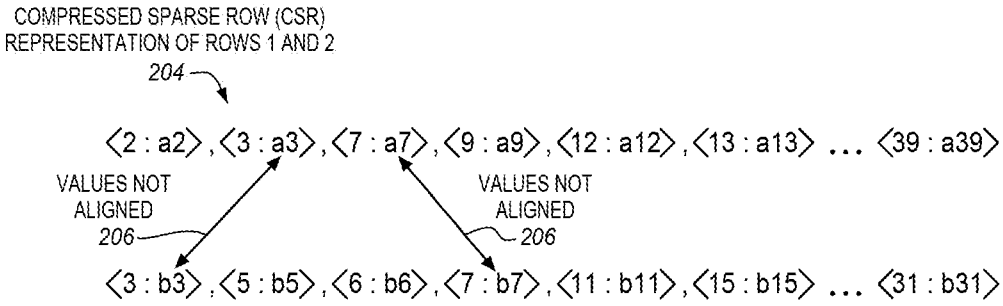

COMPRESSED SPARSE ROW (CSR)
REPRESENTATION OF ROWS 1 AND 2
204
⟨2 : a2⟩, ⟨3 : a3⟩, ⟨7 : a7⟩, ⟨9 : a9⟩, ⟨12 : a12⟩, ⟨13 : a13⟩ ... ⟨39 : a39⟩
VALUES NOT
ALIGNED
206
VALUES NOT
ALIGNED
206
⟨3 : b3⟩, ⟨5 : b5⟩, ⟨6 : b6⟩, ⟨7 : b7⟩, ⟨11 : b11⟩, ⟨15 : b15⟩ ... ⟨31 : b31⟩

FIG. 2

FIG. 3

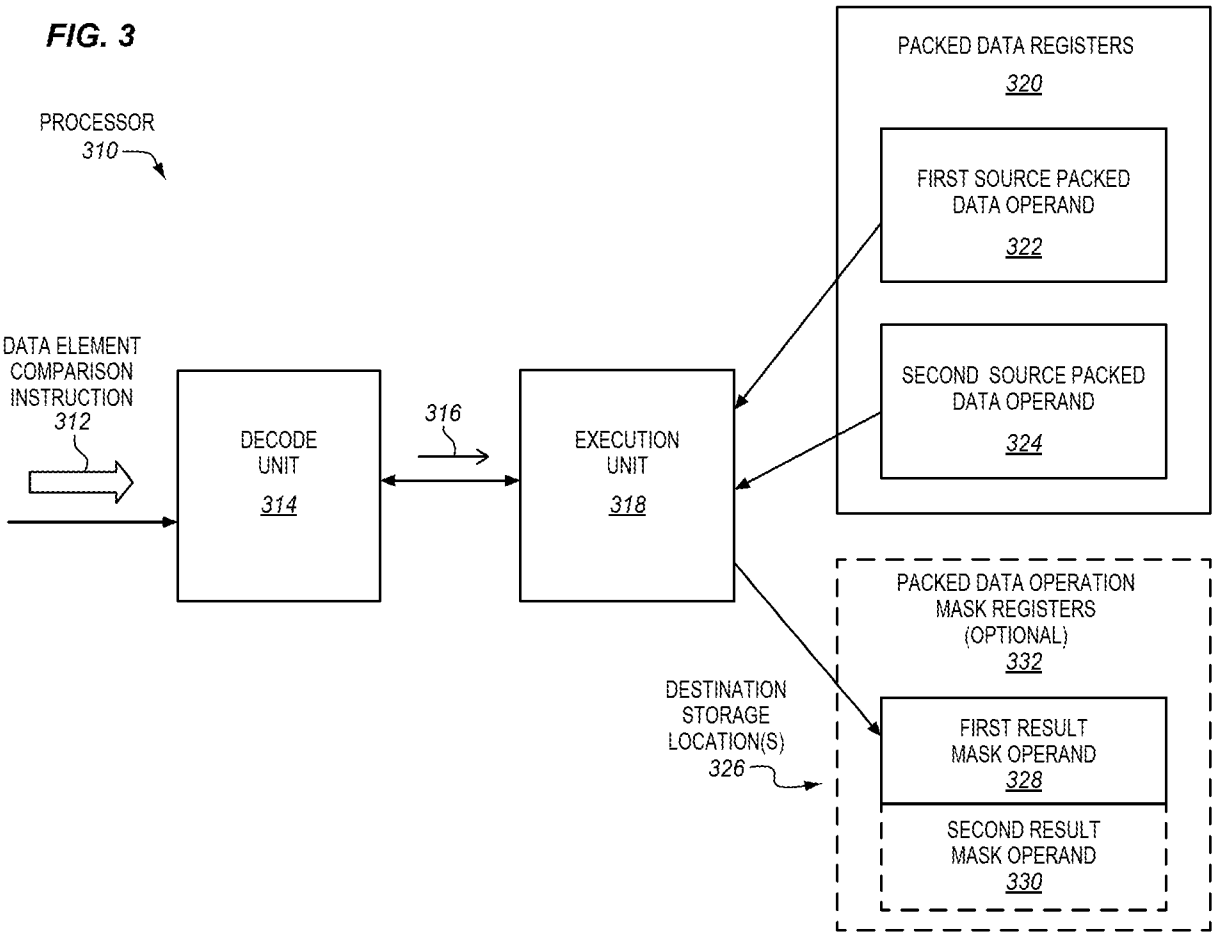

PROCESSOR
310
DATA ELEMENT COMPARISON INSTRUCTION
312
DECODE UNIT
314
316
EXECUTION UNIT
318
PACKED DATA REGISTERS
320
FIRST SOURCE PACKED DATA OPERAND
322
SECOND SOURCE PACKED DATA OPERAND
324
PACKED DATA OPERATION MASK REGISTERS (OPTIONAL)
332
DESTINATION STORAGE LOCATION(S)
326
FIRST RESULT MASK OPERAND
328
SECOND RESULT MASK OPERAND
330

FIG. 4

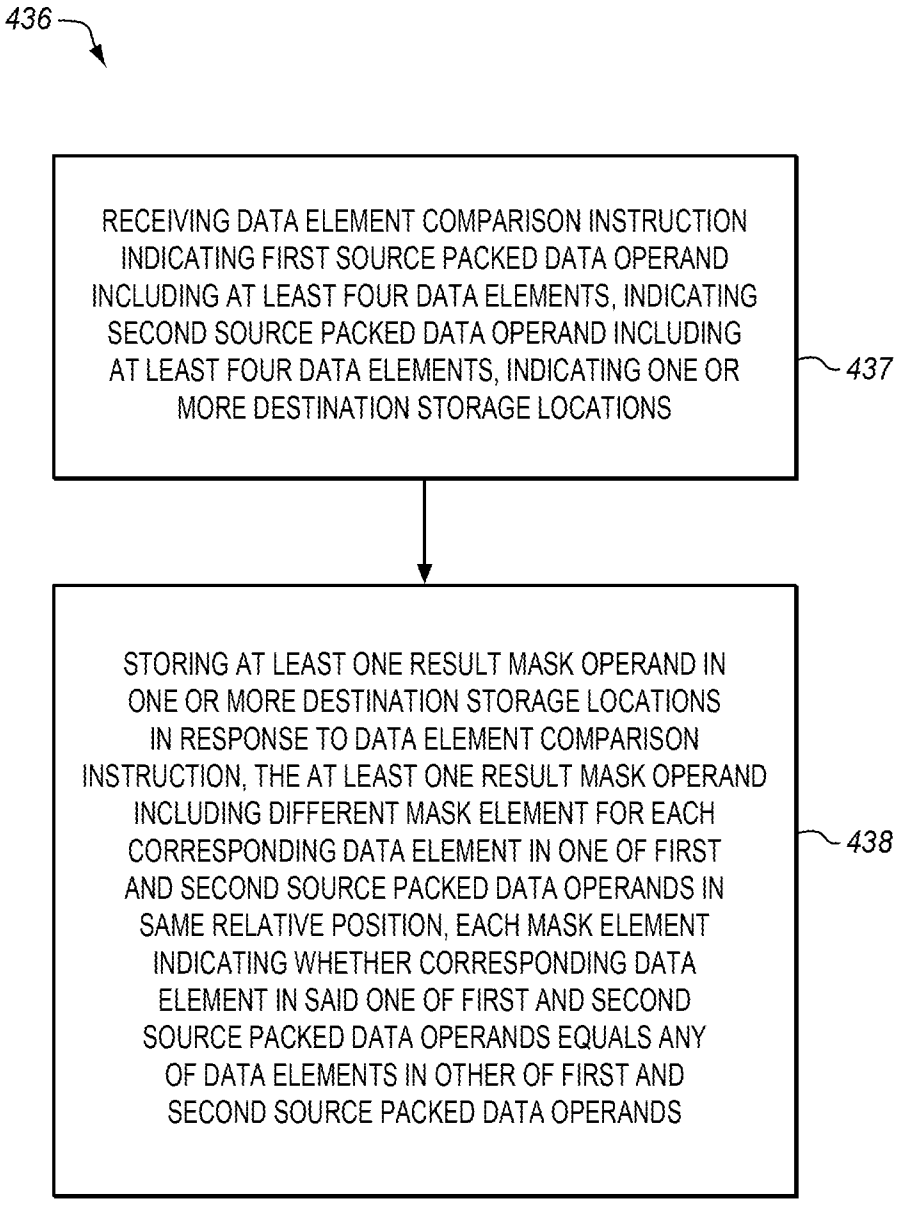

436
RECEIVING DATA ELEMENT COMPARISON INSTRUCTION INDICATING FIRST SOURCE PACKED DATA OPERAND INCLUDING AT LEAST FOUR DATA ELEMENTS, INDICATING SECOND SOURCE PACKED DATA OPERAND INCLUDING AT LEAST FOUR DATA ELEMENTS, INDICATING ONE OR MORE DESTINATION STORAGE LOCATIONS
437
STORING AT LEAST ONE RESULT MASK OPERAND IN ONE OR MORE DESTINATION STORAGE LOCATIONS IN RESPONSE TO DATA ELEMENT COMPARISON INSTRUCTION, THE AT LEAST ONE RESULT MASK OPERAND INCLUDING DIFFERENT MASK ELEMENT FOR EACH CORRESPONDING DATA ELEMENT IN ONE OF FIRST AND SECOND SOURCE PACKED DATA OPERANDS IN SAME RELATIVE POSITION, EACH MASK ELEMENT INDICATING WHETHER CORRESPONDING DATA ELEMENT IN SAID ONE OF FIRST AND SECOND SOURCE PACKED DATA OPERANDS EQUALS ANY OF DATA ELEMENTS IN OTHER OF FIRST AND SECOND SOURCE PACKED DATA OPERANDS
438

FIG. 5

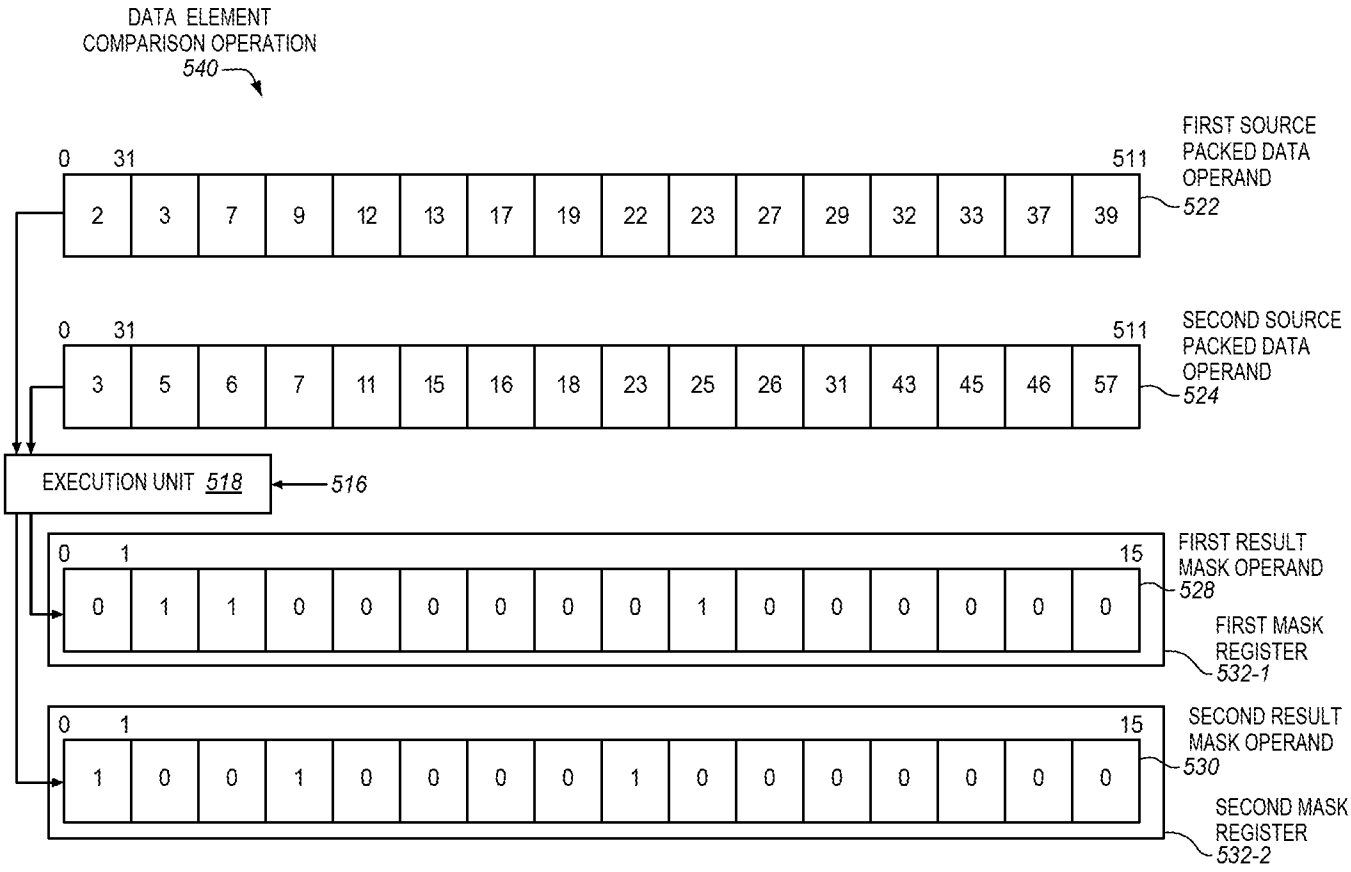

DATA ELEMENT COMPARISON OPERATION
540
0
31
511
2 3 7 9 12 13 17 19 22 23 27 29 32 33 37 39
FIRST SOURCE PACKED DATA OPERAND
522
0
31
511
3 5 6 7 11 15 16 18 23 25 26 31 43 45 46 57
SECOND SOURCE PACKED DATA OPERAND
524
EXECUTION UNIT 518
516
0
1
15
0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0
FIRST RESULT MASK OPERAND
528
FIRST MASK REGISTER
532-1
0
1
15
1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0
SECOND RESULT MASK OPERAND
530
SECOND MASK REGISTER
532-2

FIG. 6

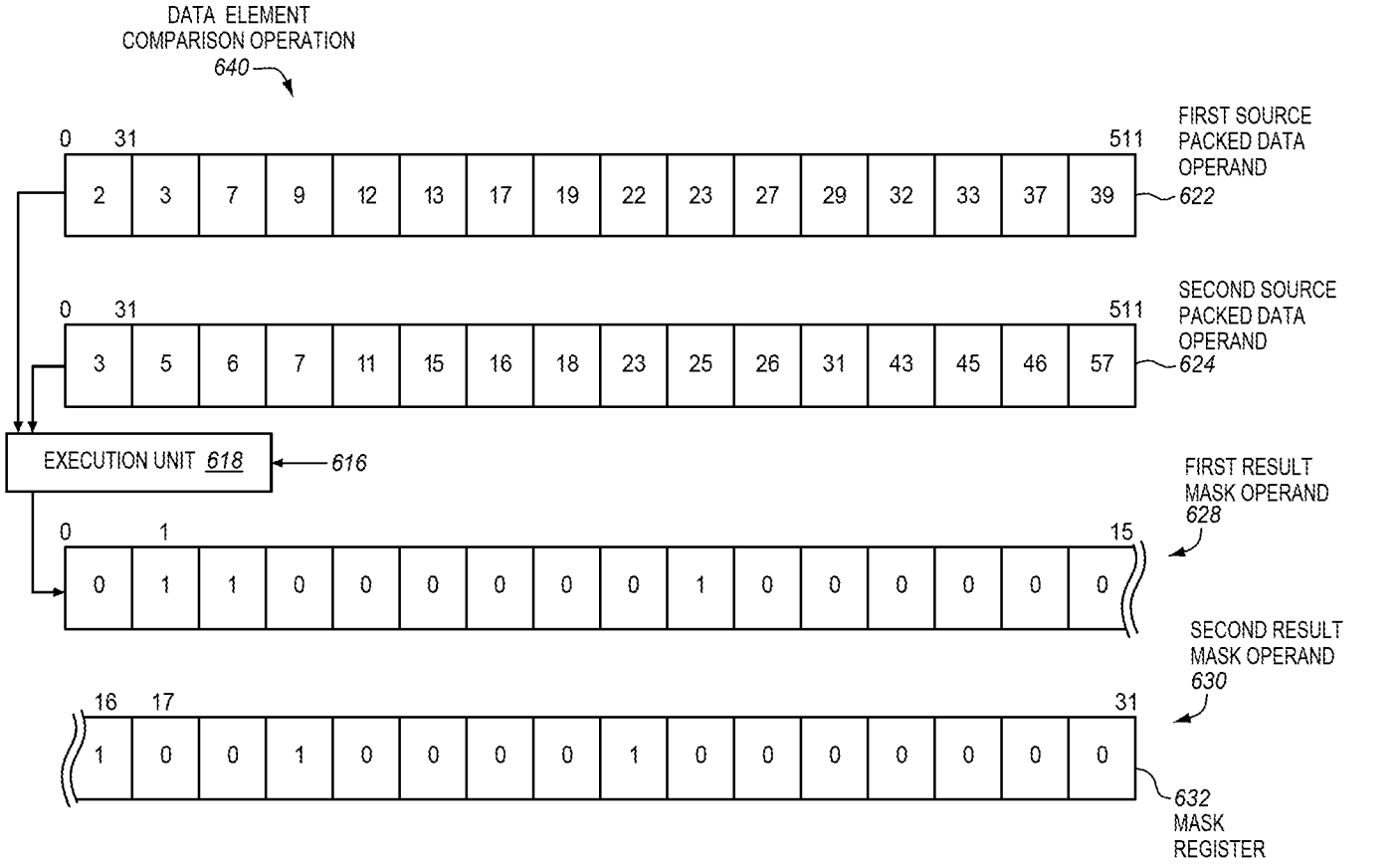

DATA ELEMENT COMPARISON OPERATION
640
0
31
511
2 3 7 9 12 13 17 19 22 23 27 29 32 33 37 39
FIRST SOURCE PACKED DATA OPERAND
622
0
31
511
3 5 6 7 11 15 16 18 23 25 26 31 43 45 46 57
SECOND SOURCE PACKED DATA OPERAND
624
EXECUTION UNIT 618
616
FIRST RESULT MASK OPERAND
628
0
1
15
0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0
SECOND RESULT MASK OPERAND
630
16
17
31
1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0
632
MASK REGISTER

FIG. 7

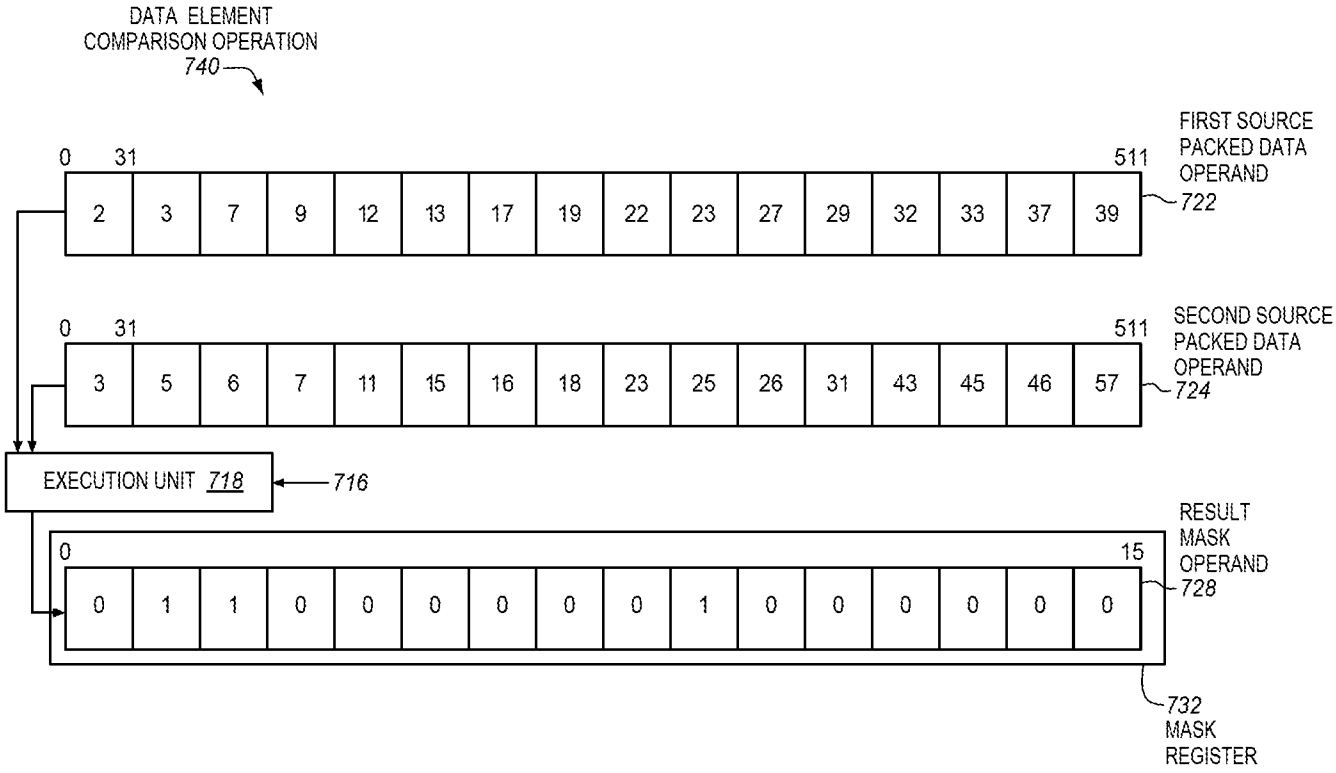

DATA ELEMENT COMPARISON OPERATION
740
0
31
511
2 3 7 9 12 13 17 19 22 23 27 29 32 33 37 39
FIRST SOURCE PACKED DATA OPERAND
722
0
31
511
3 5 6 7 11 15 16 18 23 25 26 31 43 45 46 57
SECOND SOURCE PACKED DATA OPERAND
724
EXECUTION UNIT 718
716
0
15
0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0
RESULT MASK OPERAND
728
732
MASK REGISTER

FIG. 8

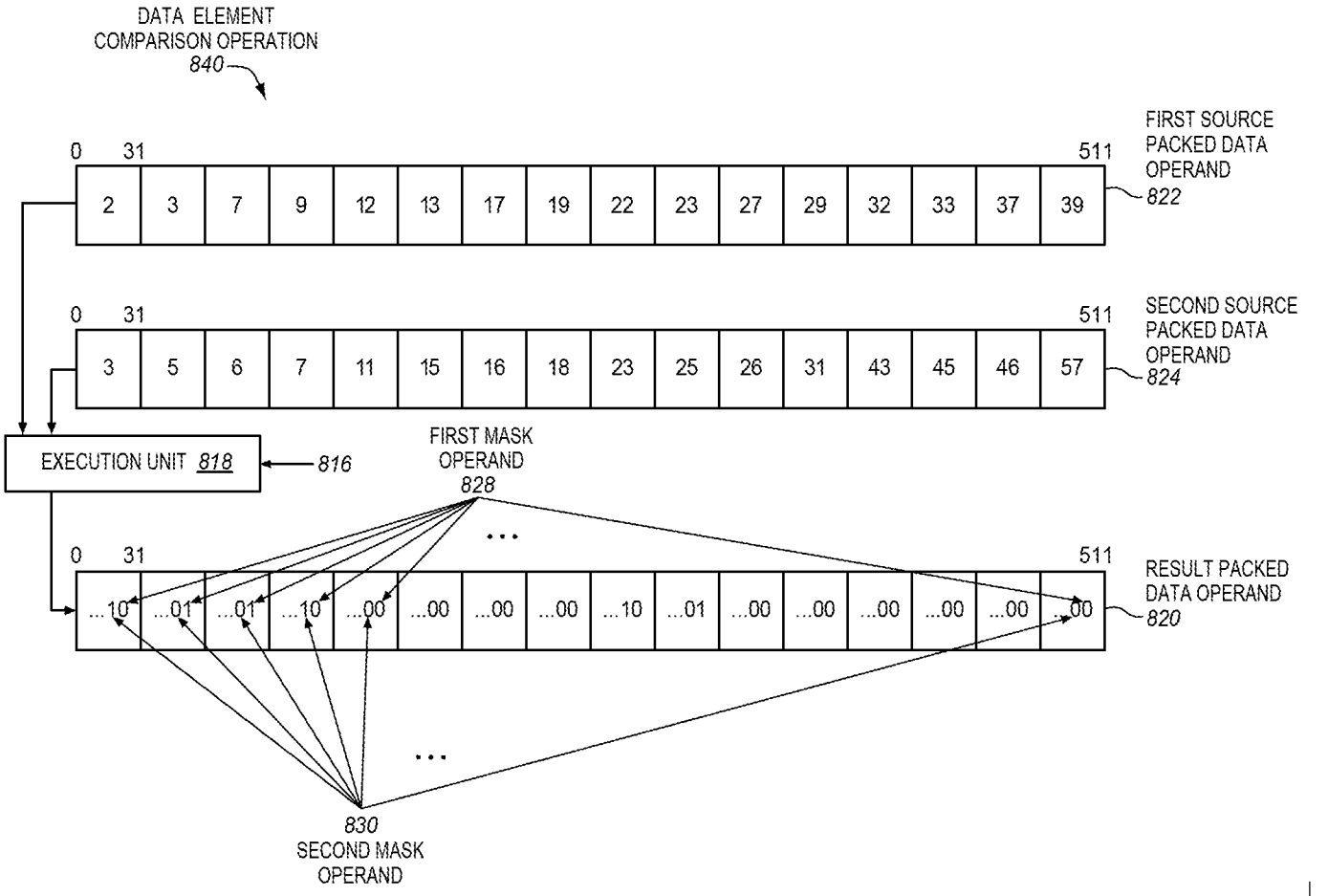
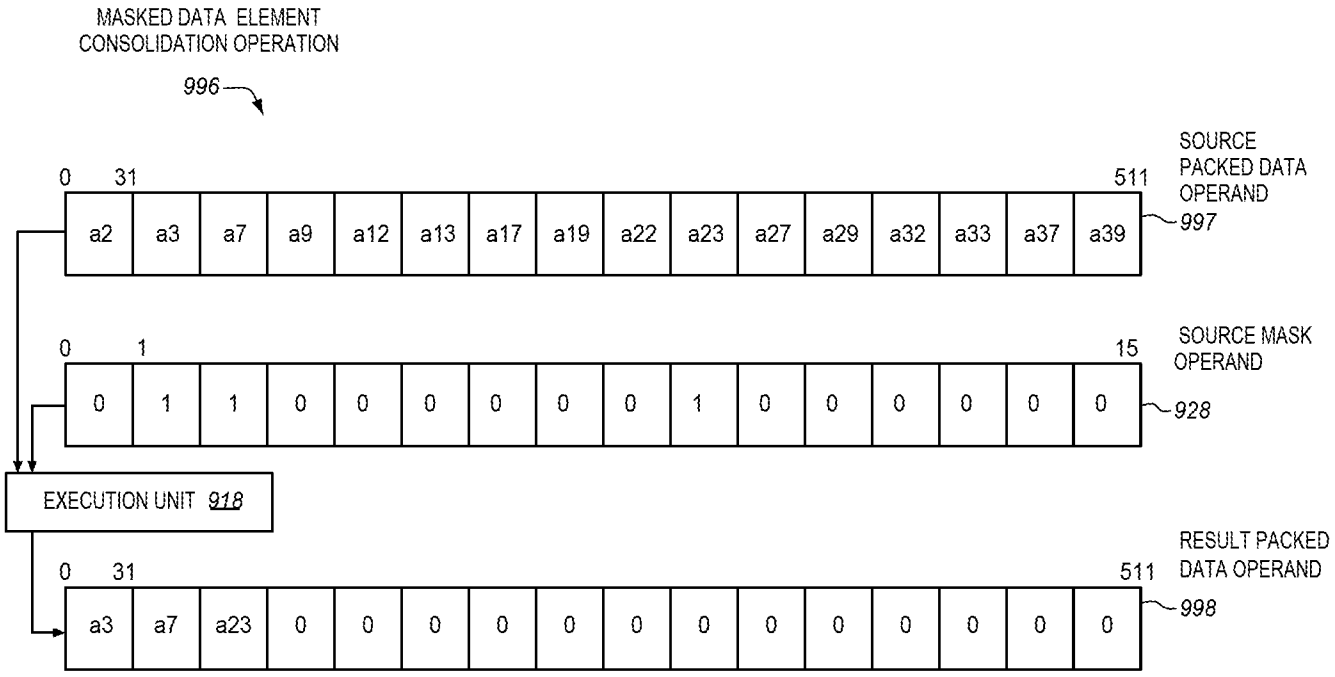

DATA ELEMENT COMPARISON OPERATION
840
0
31
511
2 3 7 9 12 13 17 19 22 23 27 29 32 33 37 39
FIRST SOURCE PACKED DATA OPERAND
822
3 5 6 7 11 15 16 18 23 25 26 31 43 45 46 57
SECOND SOURCE PACKED DATA OPERAND
824
EXECUTION UNIT 818
816
FIRST MASK OPERAND
828
...10 ...01 ...01 ...10 ...00 ...00 ...00 ...00 ...10 ...01 ...00 ...00 ...00 ...00 ...00 ...00
RESULT PACKED DATA OPERAND
820
830
SECOND MASK OPERAND

FIG. 9


MASKED DATA ELEMENT CONSOLIDATION OPERATION
996
0
31
511
a2 a3 a7 a9 a12 a13 a17 a19 a22 a23 a27 a29 a32 a33 a37 a39
SOURCE PACKED DATA OPERAND
997
0
1
15
0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0
SOURCE MASK OPERAND
928
EXECUTION UNIT 918
0
31
511
a3 a7 a23 0 0 0 0 0 0 0 0 0 0 0 0 0
RESULT PACKED DATA OPERAND
998

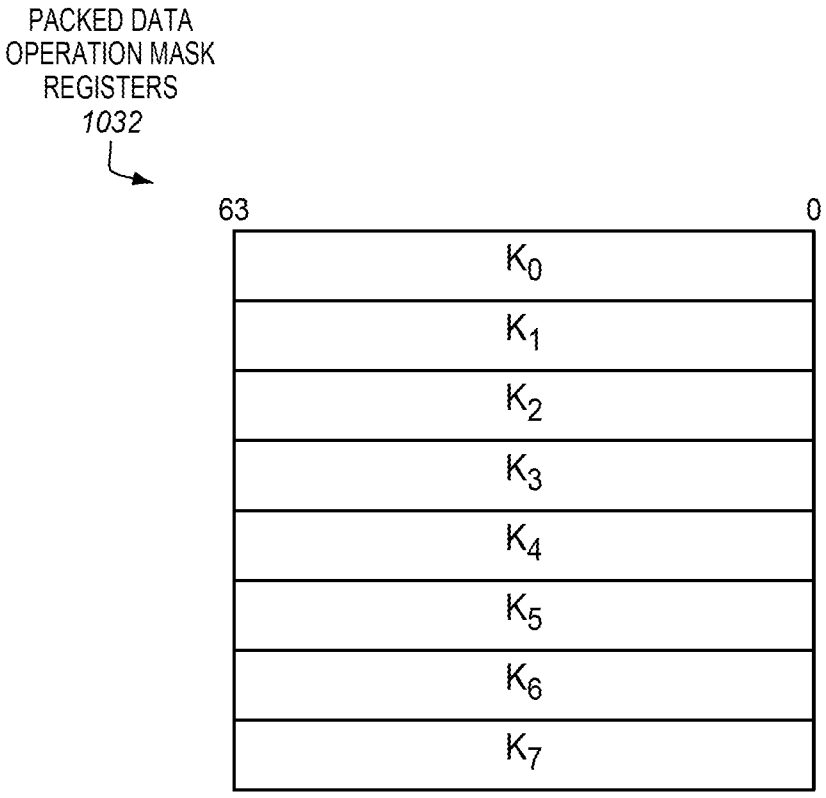

PACKED DATA
OPERATION MASK
REGISTERS
1032
63
0
K0
K1
K2
K3
K4
K5
K6
K7

FIG. 10

FIG. 11

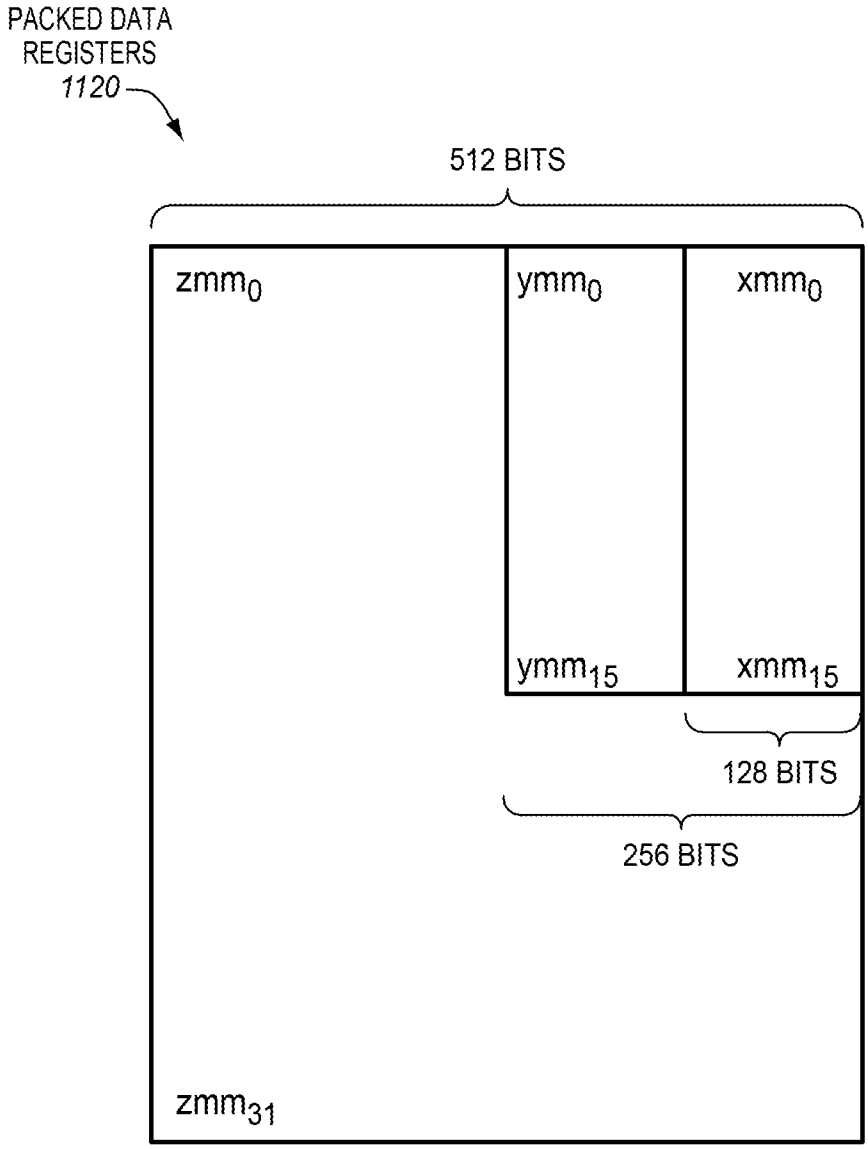
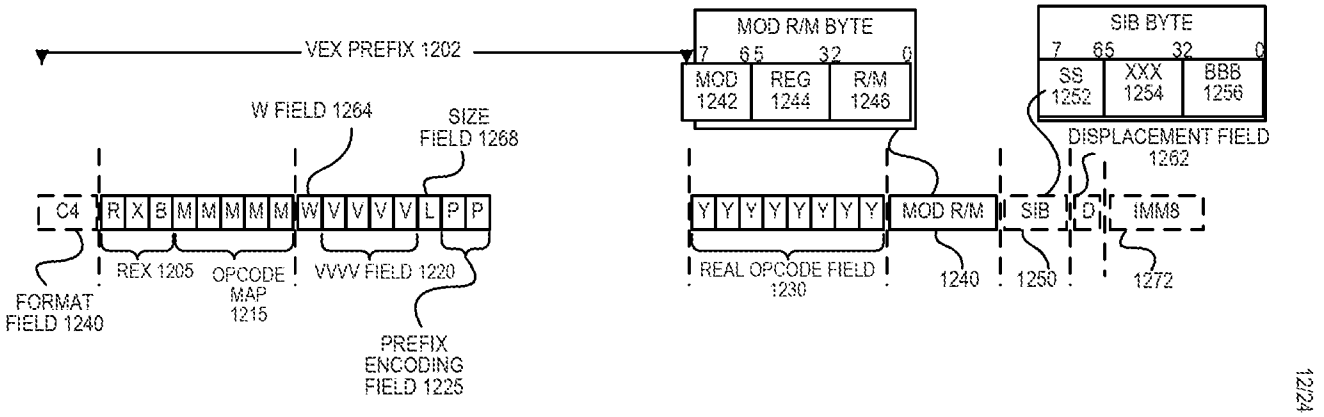

PACKED DATA REGISTERS
1120
512 BITS
zmm0
ymm0
xmm0
ymm15
xmm15
128 BITS
256 BITS
zmm31

FIG. 12A



12/24

FIG. 12B

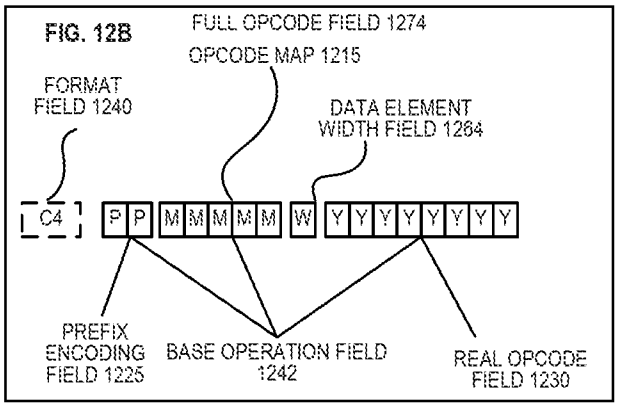
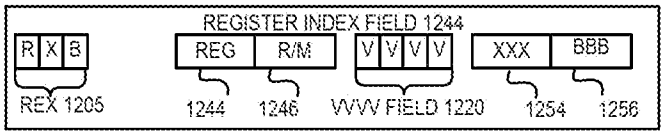


FIG. 12C



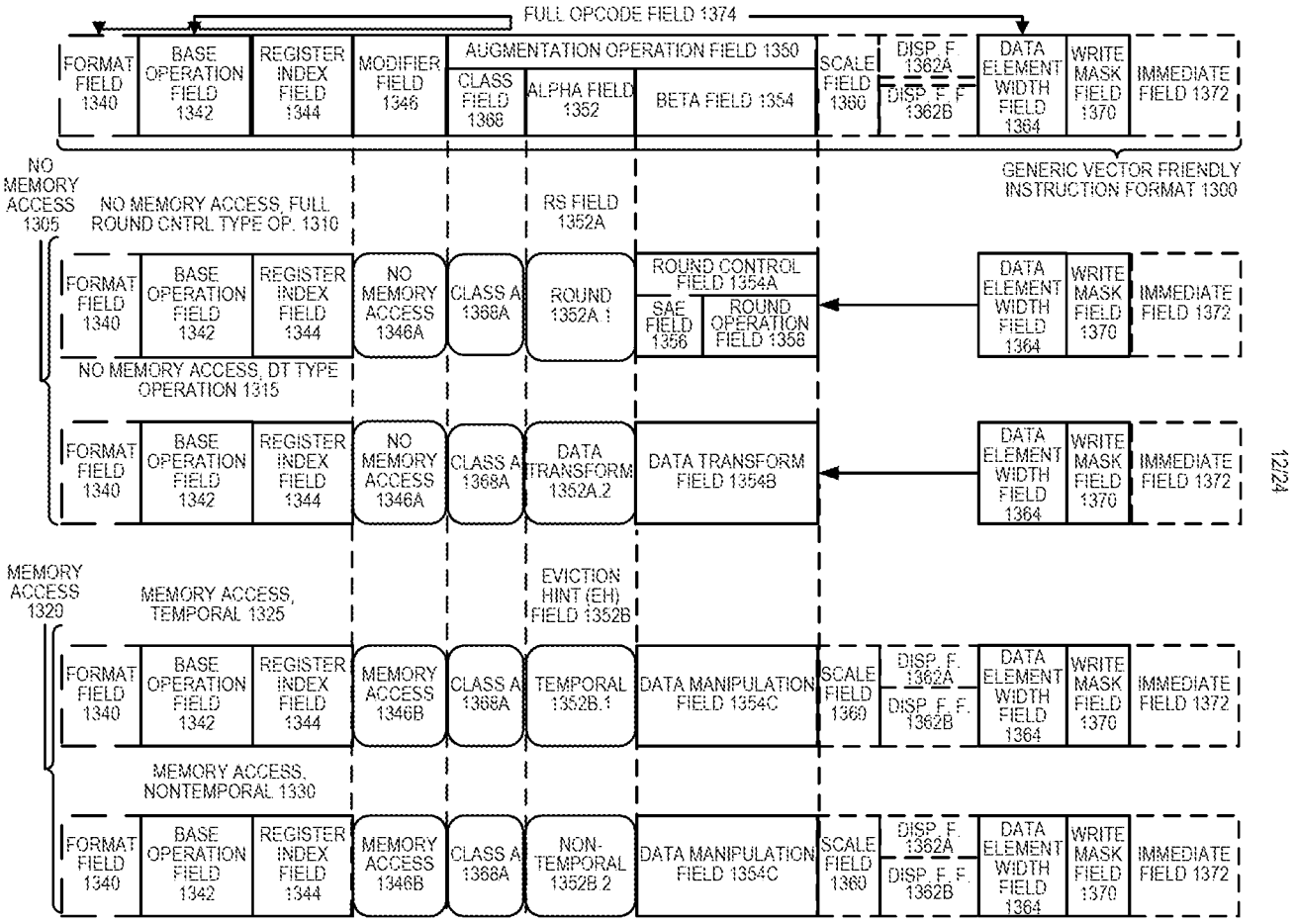
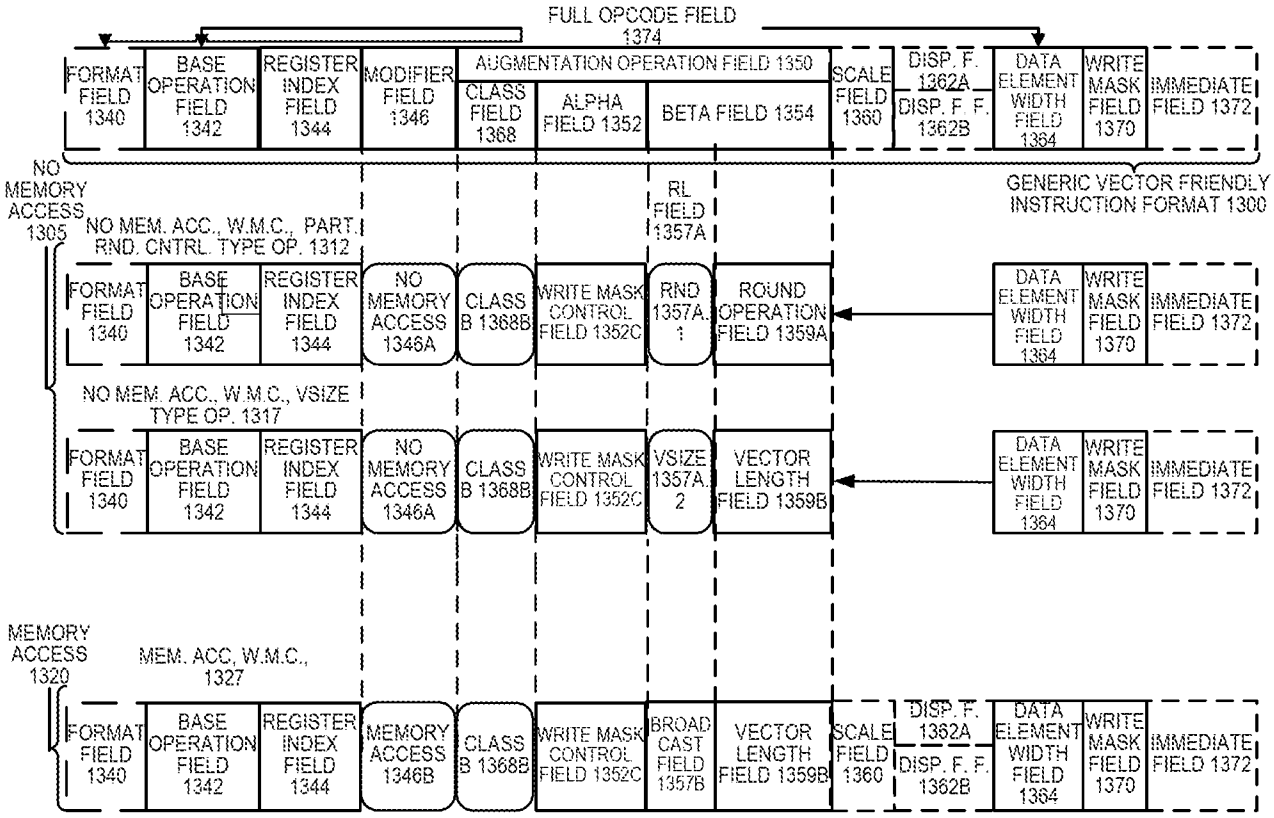

FULL OPCODE FIELD 1374
FORMAT FIELD 1340
BASE OPERATION FIELD 1342
REGISTER INDEX FIELD 1344
MODIFIER FIELD 1346
AUGMENTATION OPERATION FIELD 1350
CLASS FIELD 1368
ALPHA FIELD 1352
BETA FIELD 1354
SCALE FIELD 1360
DISP. F. 1362A
DISP. F. F. 1362B
DATA ELEMENT WIDTH FIELD 1364
WRITE MASK FIELD 1370
IMMEDIATE FIELD 1372
GENERIC VECTOR FRIENDLY INSTRUCTION FORMAT 1300
NO MEMORY ACCESS 1305
NO MEMORY ACCESS, FULL ROUND CNTRL TYPE OP. 1310
RS FIELD 1352A
NO MEMORY ACCESS 1346A
CLASS A 1368A
ROUND 1352A.1
ROUND CONTROL FIELD 1354A
SAE FIELD 1356
ROUND OPERATION FIELD 1358
NO MEMORY ACCESS, DT TYPE OPERATION 1315
DATA TRANSFORM 1352A.2
DATA TRANSFORM FIELD 1354B
MEMORY ACCESS 1320
MEMORY ACCESS, TEMPORAL 1325
EVICTION HINT (EH) FIELD 1352B
MEMORY ACCESS 1346B
TEMPORAL 1352B.1
DATA MANIPULATION FIELD 1354C
MEMORY ACCESS, NONTEMPORAL 1330
NON-TEMPORAL 1352B.2
12/24

FIG.13A

FIG.13B


FULL OPCODE FIELD 1374
FORMAT FIELD 1340
BASE OPERATION FIELD 1342
REGISTER INDEX FIELD 1344
MODIFIER FIELD 1346
AUGMENTATION OPERATION FIELD 1350
CLASS FIELD 1368
ALPHA FIELD 1352
BETA FIELD 1354
SCALE FIELD 1360
DISP. F. 1362A
DISP. F. F. 1362B
DATA ELEMENT WIDTH FIELD 1364
WRITE MASK FIELD 1370
IMMEDIATE FIELD 1372
GENERIC VECTOR FRIENDLY INSTRUCTION FORMAT 1300
NO MEMORY ACCESS 1305
RL FIELD 1357A
NO MEM. ACC., W.M.C., PART. RND. CNTRL. TYPE OP. 1312
NO MEMORY ACCESS 1346A
CLASS B 1368B
WRITE MASK CONTROL FIELD 1352C
RND 1357A. 1
ROUND OPERATION FIELD 1359A
NO MEM. ACC., W.M.C., VSIZE TYPE OP. 1317
VSIZE 1357A. 2
VECTOR LENGTH FIELD 1359B
MEMORY ACCESS 1320
MEM. ACC, W.M.C., 1327
MEMORY ACCESS 1346B
BROAD CAST FIELD 1357B

13/24

FIG. 14A

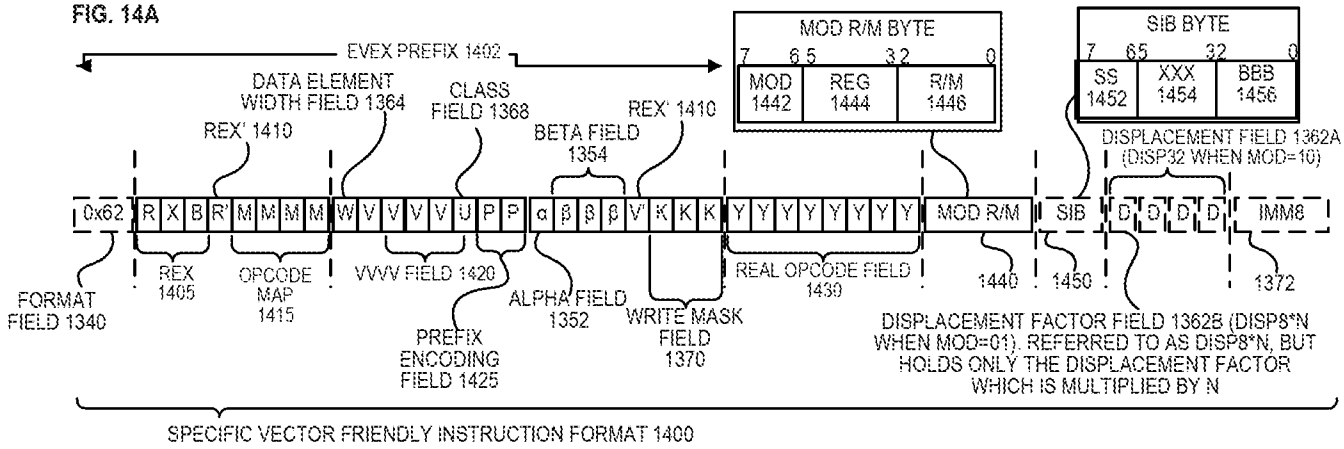


FIG. 14B

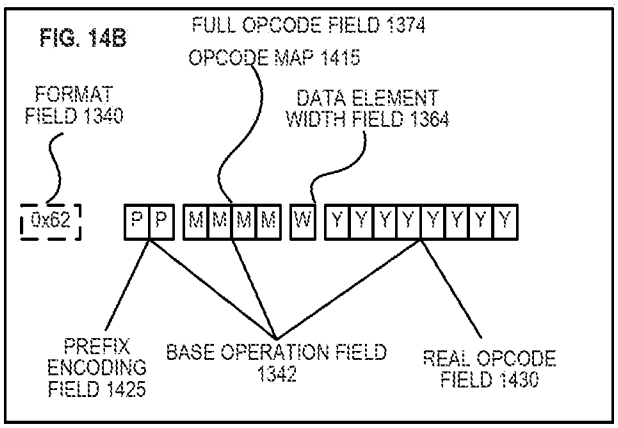
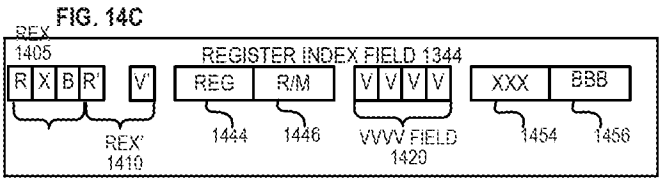


FIG. 14C



14/24

15/24

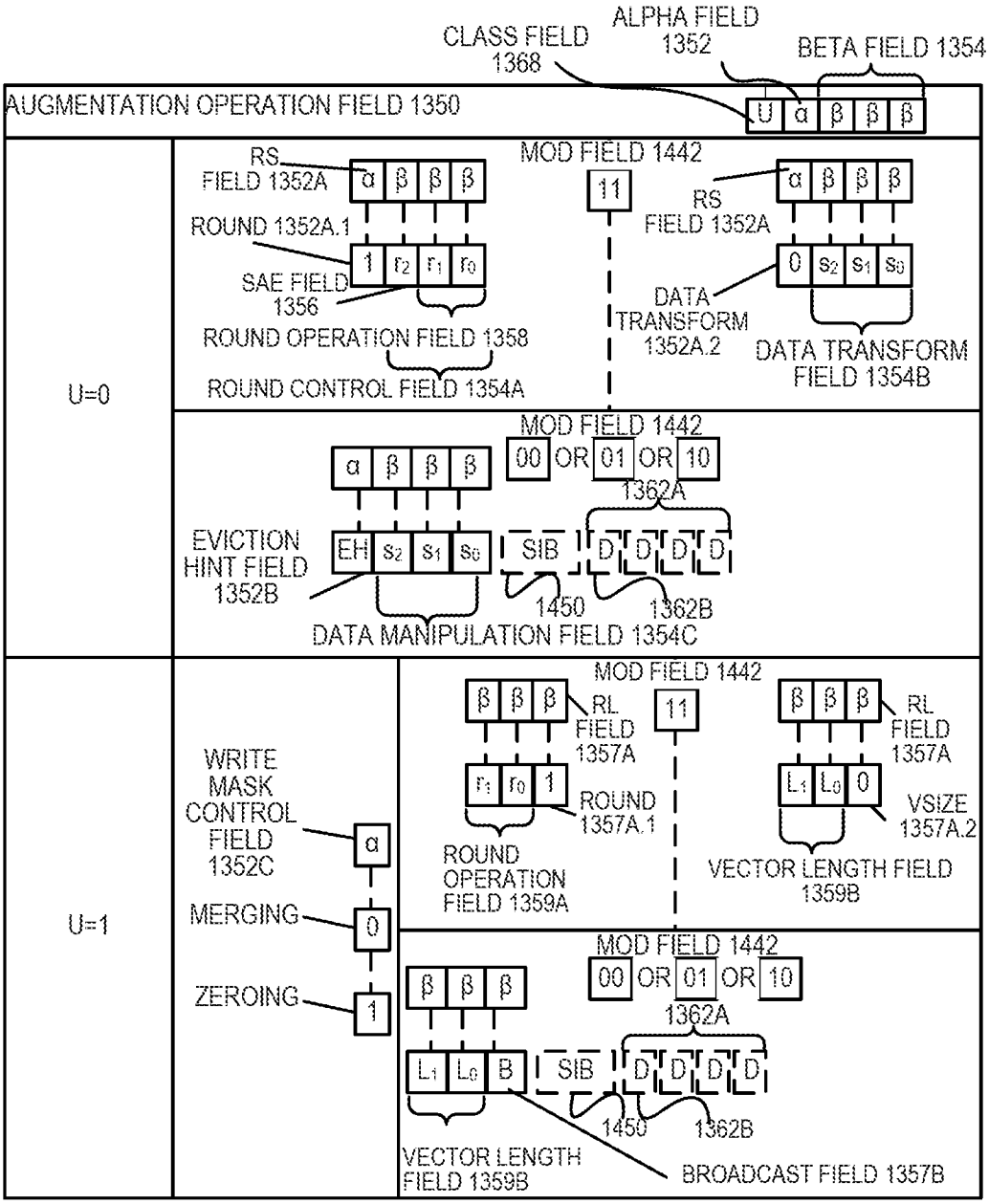

CLASS FIELD 1368
ALPHA FIELD 1352
BETA FIELD 1354
AUGMENTATION OPERATION FIELD 1350
U α β β β
U=0
RS FIELD 1352A
MOD FIELD 1442
11
ROUND 1352A.1
SAE FIELD 1356
ROUND OPERATION FIELD 1358
ROUND CONTROL FIELD 1354A
DATA TRANSFORM 1352A.2
DATA TRANSFORM FIELD 1354B
00 OR 01 OR 10
1362A
EVICTION HINT FIELD 1352B
SIB
1450
1362B
DATA MANIPULATION FIELD 1354C
U=1
WRITE MASK CONTROL FIELD 1352C
MERGING
ZEROING
RL FIELD 1357A
ROUND 1357A.1
ROUND OPERATION FIELD 1359A
VSIZE 1357A.2
VECTOR LENGTH FIELD 1359B
VECTOR LENGTH FIELD 1359B
BROADCAST FIELD 1357B

FIG. 14D

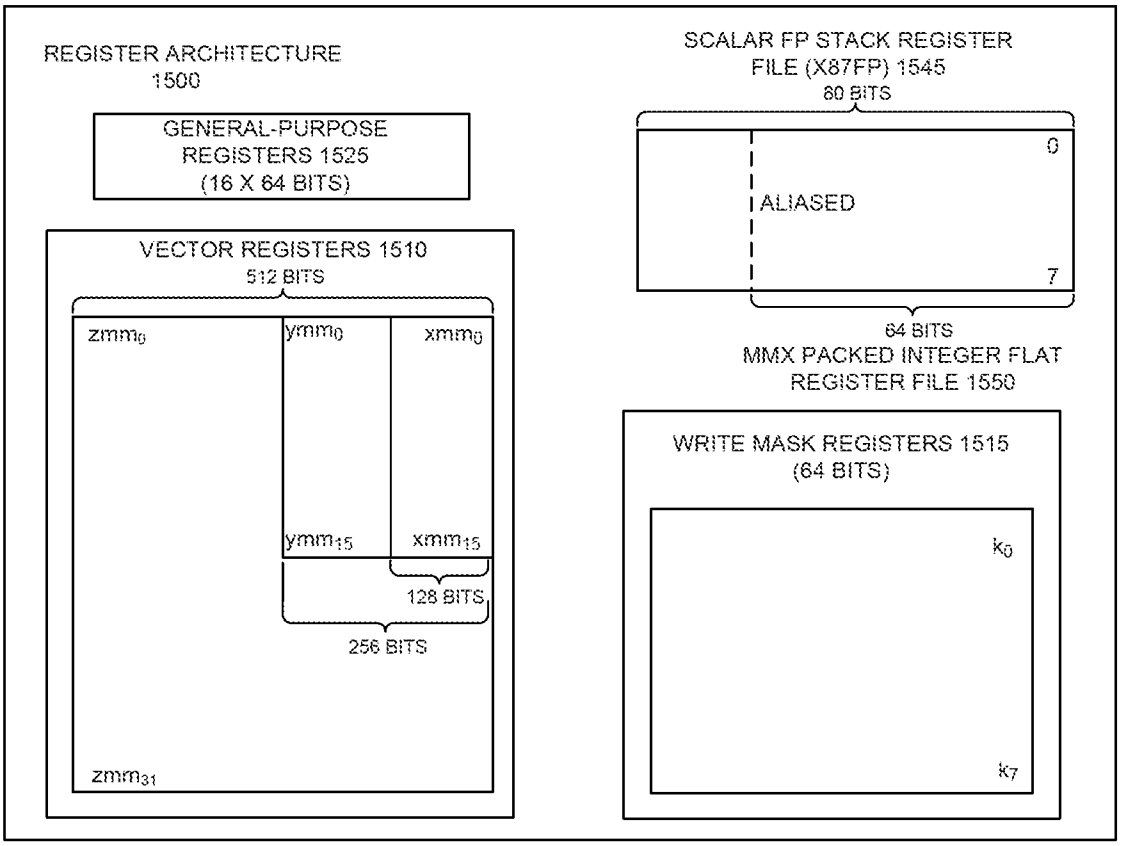

REGISTER ARCHITECTURE
1500
GENERAL-PURPOSE
REGISTERS 1525
(16 X 64 BITS)
VECTOR REGISTERS 1510
512 BITS
zmm0
ymm0
xmm0
ymm15
xmm15
128 BITS
256 BITS
zmm31
SCALAR FP STACK REGISTER
FILE (X87FP) 1545
80 BITS
0
ALIASED
7
64 BITS
MMX PACKED INTEGER FLAT
REGISTER FILE 1550
WRITE MASK REGISTERS 1515
(64 BITS)
k0
k7

FIG. 15

16/24

FIG. 16A

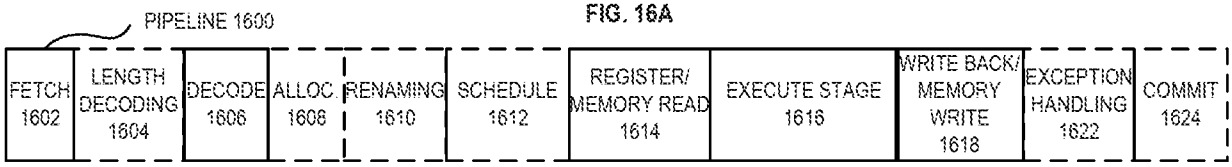
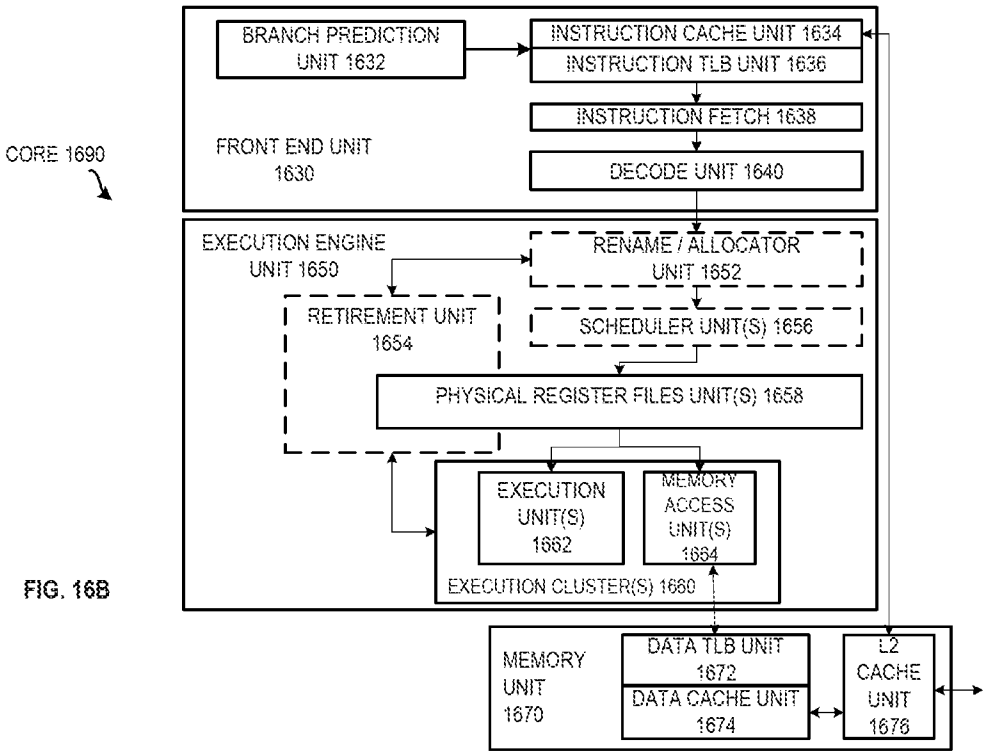


FIG. 16B



17/24

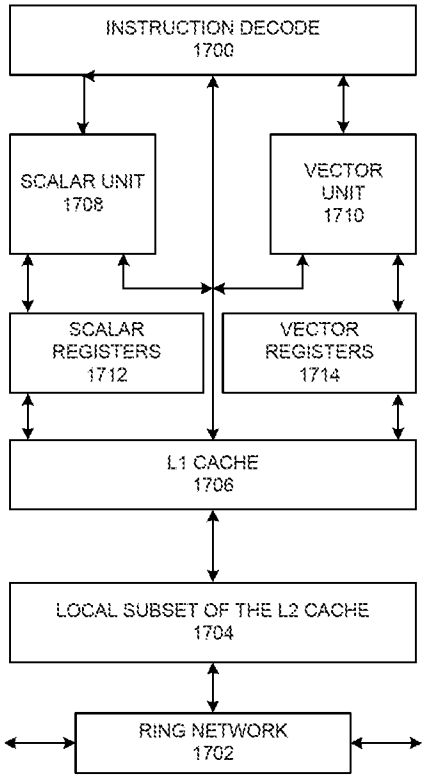


FIG. 17A

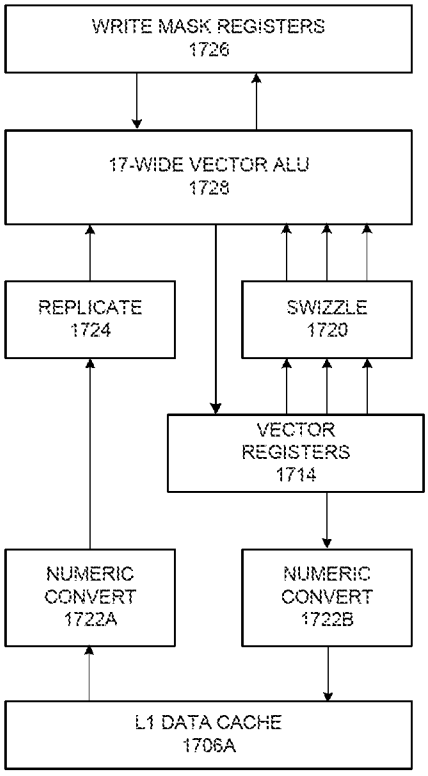


FIG. 17B

18/24

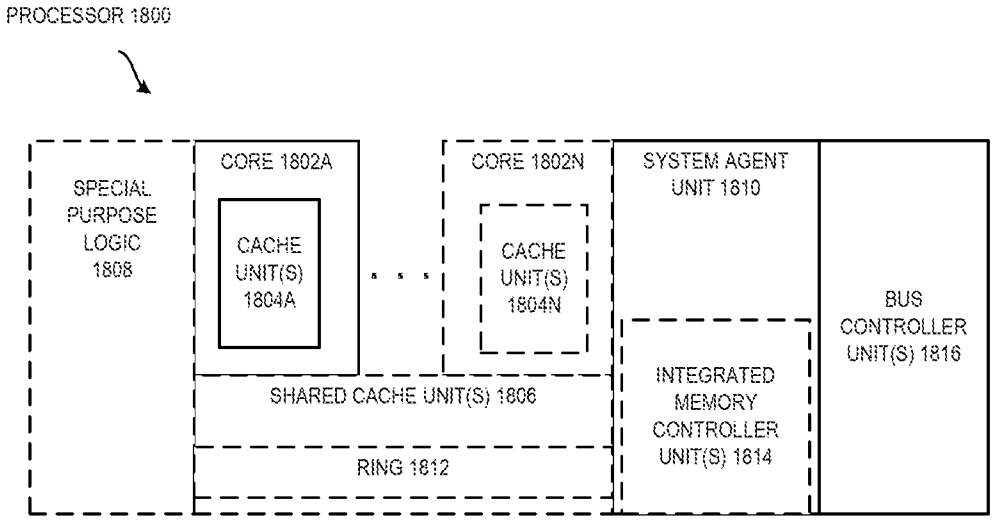

PROCESSOR 1800
SPECIAL PURPOSE LOGIC 1808
CORE 1802A
CACHE UNIT(S) 1804A
CORE 1802N
CACHE UNIT(S) 1804N
SYSTEM AGENT UNIT 1810
BUS CONTROLLER UNIT(S) 1816
INTEGRATED MEMORY CONTROLLER UNIT(S) 1814
SHARED CACHE UNIT(S) 1806
RING 1812

FIG. 18

20/24

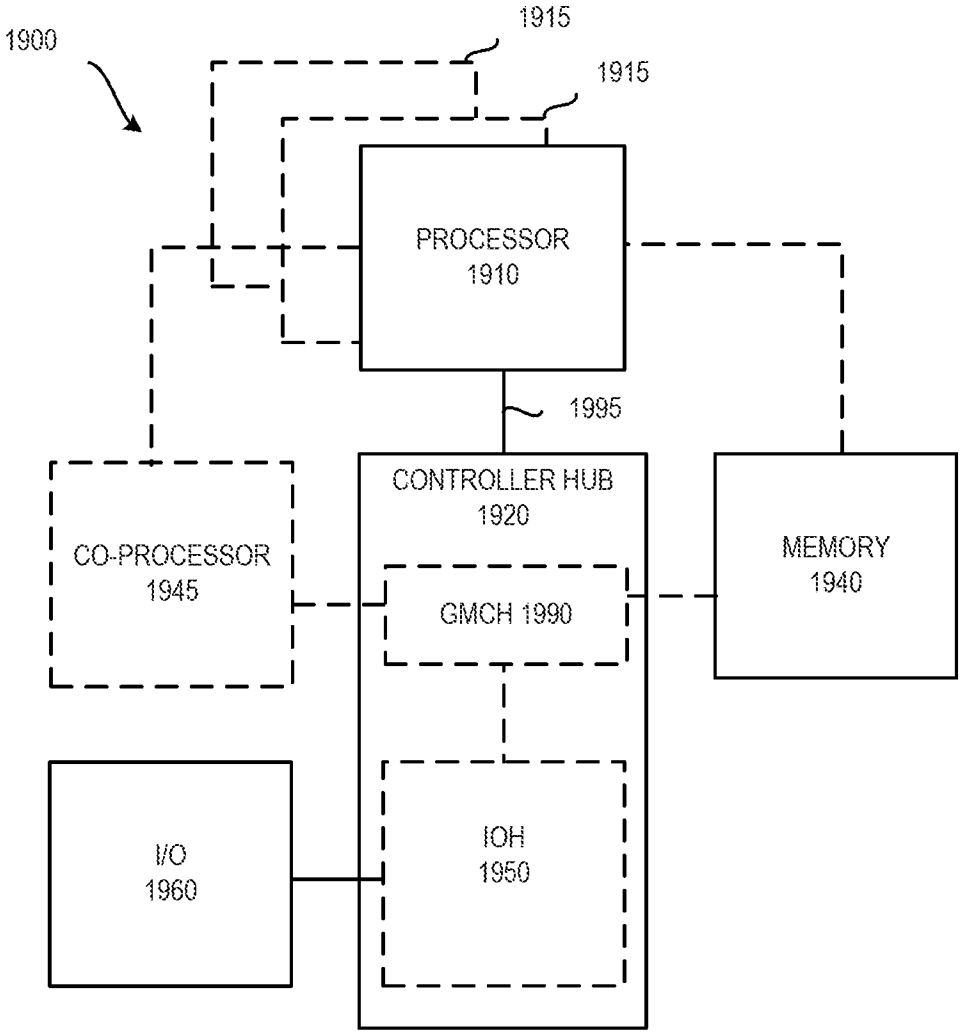

1900
1915
1915
PROCESSOR
1910
1995
CONTROLLER HUB
1920
CO-PROCESSOR
1945
GMCH 1990
MEMORY
1940
I/O
1960
IOH
1950

FIG. 19

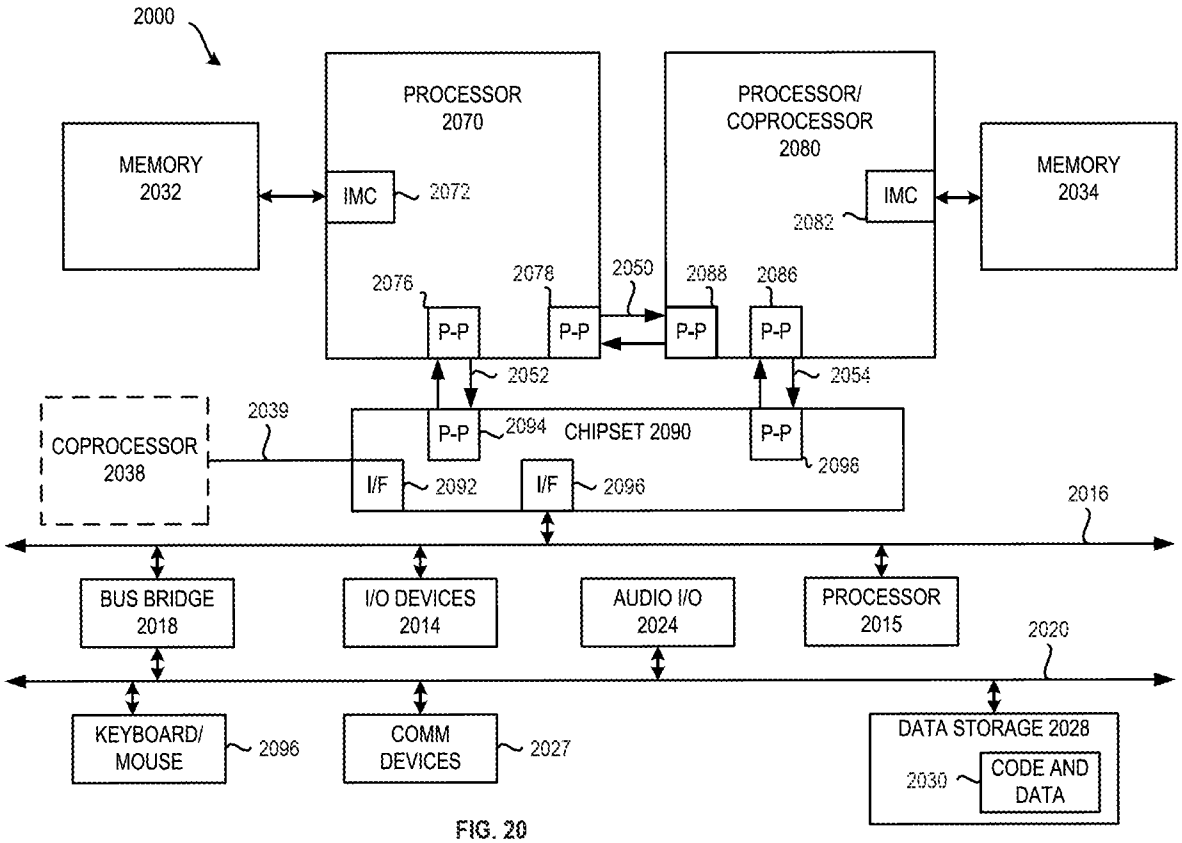

2000
PROCESSOR 2070
PROCESSOR/ COPROCESSOR 2080
MEMORY 2032
MEMORY 2034
IMC
2072
IMC
2082
2076
2078
2050
2088
2086
P-P
P-P
P-P
P-P
2052
2054
COPROCESSOR 2038
2039
P-P
2094
CHIPSET 2090
P-P
2096
I/F
2092
I/F
2096
2016
BUS BRIDGE 2018
I/O DEVICES 2014
AUDIO I/O 2024
PROCESSOR 2015
2020
KEYBOARD/ MOUSE
2096
COMM DEVICES
2027
DATA STORAGE 2028
2030
CODE AND DATA

FIG. 20

21/24

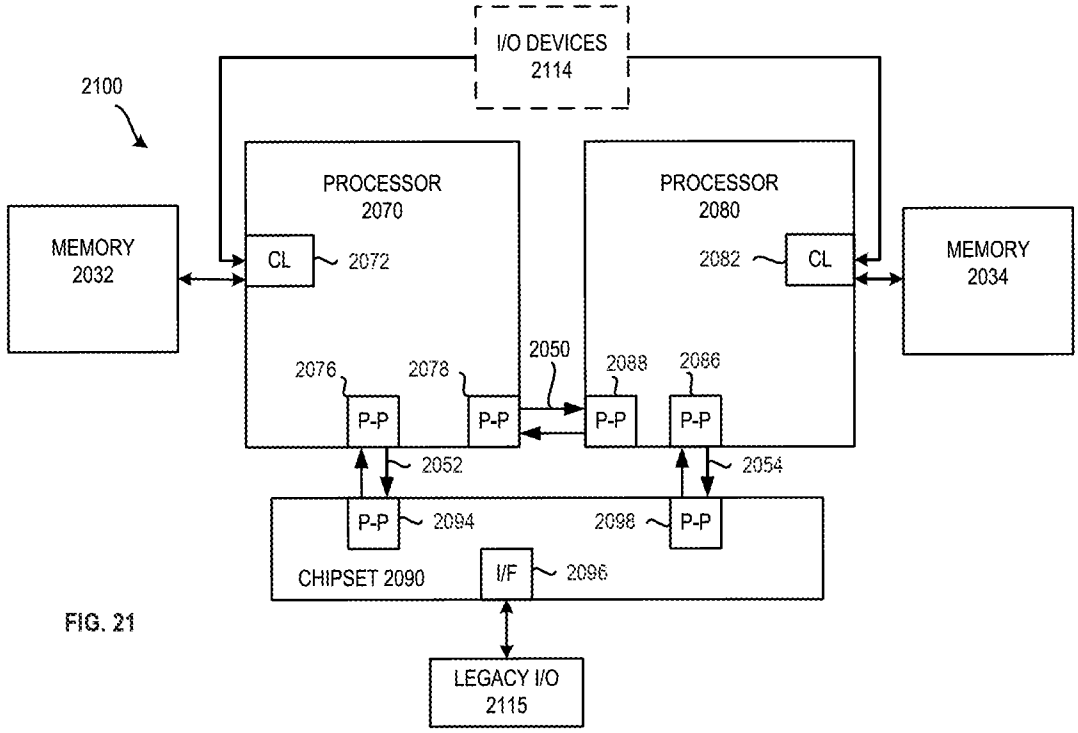

I/O DEVICES
2114
2100
PROCESSOR
2070
PROCESSOR
2080
MEMORY
2032
CL
2072
2082
CL
MEMORY
2034
2076
2078
2050
2088
2086
P-P
P-P
P-P
P-P
2052
2054
P-P
2094
2098
P-P
CHIPSET 2090
I/F
2096
LEGACY I/O
2115

FIG. 21

22/24

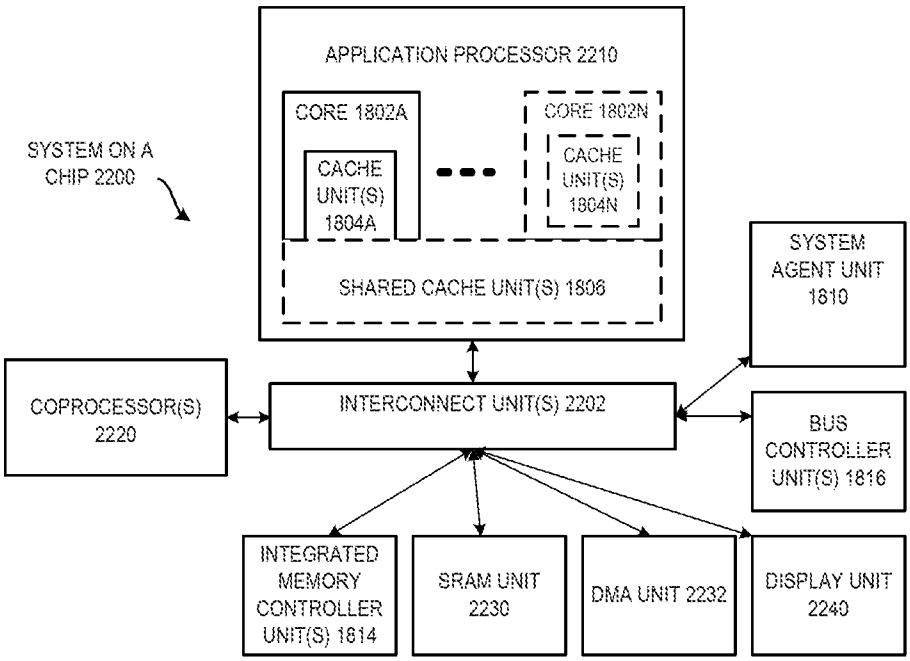

APPLICATION PROCESSOR 2210
CORE 1802A
CACHE UNIT(S) 1804A
CORE 1802N
CACHE UNIT(S) 1804N
SHARED CACHE UNIT(S) 1806
SYSTEM ON A CHIP 2200
SYSTEM AGENT UNIT 1810
COPROCESSOR(S) 2220
INTERCONNECT UNIT(S) 2202
BUS CONTROLLER UNIT(S) 1816
INTEGRATED MEMORY CONTROLLER UNIT(S) 1814
SRAM UNIT 2230
DMA UNIT 2232
DISPLAY UNIT 2240

FIG. 22

23/24

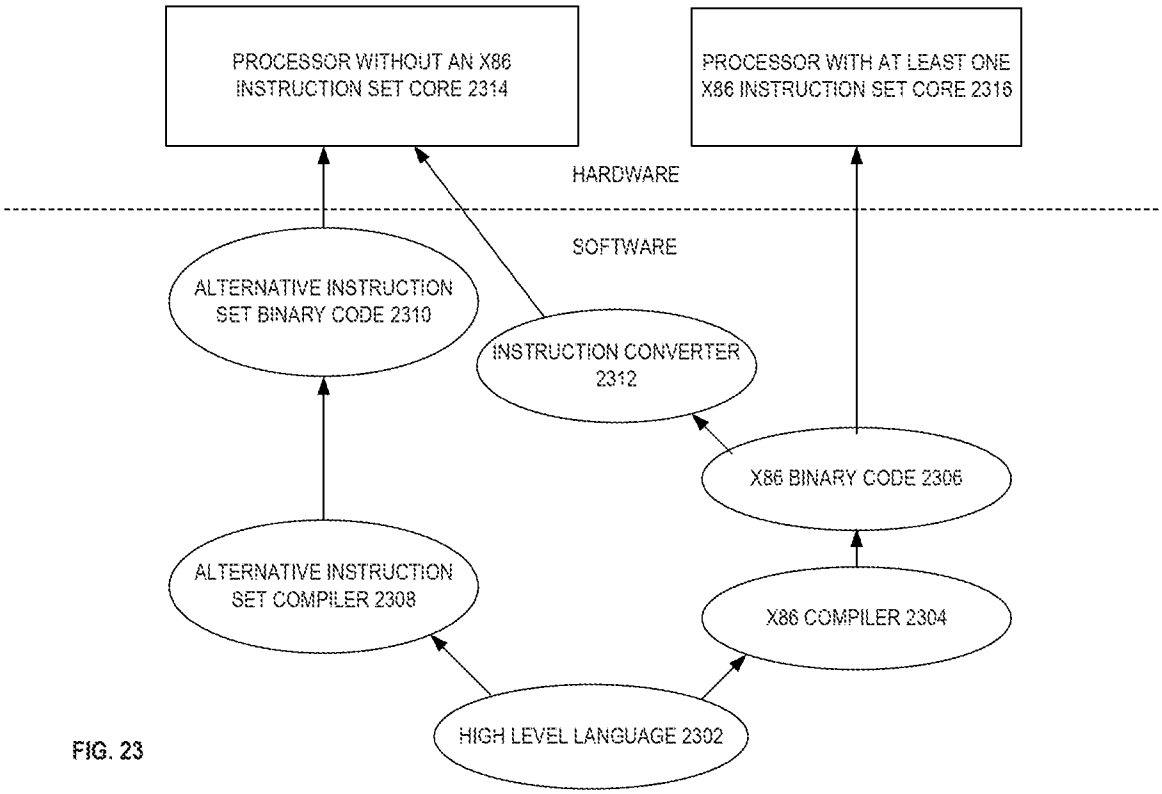

PROCESSOR WITHOUT AN X86 INSTRUCTION SET CORE 2314
PROCESSOR WITH AT LEAST ONE X86 INSTRUCTION SET CORE 2316
HARDWARE
SOFTWARE
ALTERNATIVE INSTRUCTION SET BINARY CODE 2310
INSTRUCTION CONVERTER 2312
X86 BINARY CODE 2306
ALTERNATIVE INSTRUCTION SET COMPILER 2308
X86 COMPILER 2304
HIGH LEVEL LANGUAGE 2302

FIG. 23

24/24

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2016/048370

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/30(2006.01)i, G06F 12/08(2006.01)i, G06F 9/38(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 9/30; G06F 9/308; G06F 11/34; G06F 7/00; G06F 7/02; G06F 9/00; G06F 7/32; G06F 12/08; G06F 9/38

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & Keywords: mask operand, second source, element, second result, equal, match

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2014-0289503 A1 (BRET L. TOLL et al.) 25 September 2014 See paragraphs [0035]-[0067]; claim 12; and figure 3.	1-25
A	US 2002-0002666 A1 (CAROLE DULONG et al.) 03 January 2002 See paragraphs [0023]-[0032]; and figures 1-2.	1-25
A	US 5651121 A (DANIEL DAVIES) 22 July 1997 See column 10, line 9 - column 11, line 25; and figures 1-2.	1-25
A	US 5341500 A (WILLIAM C. MOYER et al.) 23 August 1994 See column 3, line 35 - column 7, line 42; and figures 1-2.	1-25
A	US 6036350 A (LARRY MENNEMEIER et al.) 14 March 2000 See column 3, line 54 - column 8, line 23; and figures 1-3B.	1-25

☐ 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
25 November 2016 (25.11.2016)	**25 November 2016 (25.11.2016)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea
Facsimile No. +82-42-481-8578

Authorized officer
CHIN, Sang Bum
Telephone No. +82-42-481-8398

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2016/048370

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2014-0289503 Al	25/09/2014	CN 104011653 A	27/08/2014
		EP 2798458 Al	05/11/2014
		TW 201344562 A	01/11/2013
		TW 1496077 B	11/08/2015
		US 2016-154652 Al	02/06/2016
		US 9244687 B2	26/01/2016
		WO 2013-101124 Al	04/07/2013
US 2002-0002666 Al	03/01/2002	None	
US 05651121 A	22/07/1997	DE 69328070 Dl	20/04/2000
		DE 69328070 T2	13/07/2000
		EP 0602886 A2	22/06/1994
		EP 0602886 A3	07/09/1994
		EP 0602886 Bl	15/03/2000
		JP 03584053 B2	04/11/2004
		JP 06-222918 A	12/08/1994
US 05341500 A	23/08/1994	EP 0507208 A2	07/10/1992
		EP 0507208 A3	20/04/1994
		JP 02776132 B2	16/07/1998
		JP 05-173837 A	13/07/1993
US 06036350 A	14/03/2000	US 06036350 A	14/03/2000