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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
12 July 2007 (12.07.2007)

PCT

(10) International Publication Number
WO 2007/078830 A2

(51) International Patent Classification:
G11C 29/00 (2006.01)

(21) International Application Number:
PCT/US2006/047717

(22) International Filing Date:
14 December 2006 (14.12.2006)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/322,988 30 December 2005 (30.12.2005) US

(71) Applicant *(for all designated States except US)*: **INTEL CORPORATION** [US/US]; 2200 Mission College Boulevard, Santa Clara, CA 95052 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **DEMPSEY, Morgan, J.** [US/US]; 15251 S. 50th Street, #3092, Phoenix, AZ 85044 (US). **MAIZ, Jose, A.** [US/US]; 1344 Nw Benfield Drive, Portland, OR 97229 (US).

(74) Agent: **VINCENT, Lester, J.**; BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP, 12400 Wilshire Boulevard, 7th Floor, Los Angeles, CA 90025 (US).

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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

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

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 2007/078830 A2

(54) Title: REPAIR BITS FOR LOW VOLTAGE CACHE

(57) Abstract: A method and apparatus for repairing cache memories/arrays is described herein. A cache includes a plurality of lines and logically viewable in columns. A repair cache coupled to the cache includes a repair bit mapped to each logically viewable column. A repair module determines a bad bit to be repaired within a column based on any individual or combination of factors, such as the number of errors per line of the cache, the number of errors correctable per line of the cache due to error correction code (ECC), the failure rate of bits, or other considerations. The bad bit is transparently repaired by the repair bit mapped to the column including the bad bit, upon an access to a cache line including the bad bit.

REPAIR BITS FOR A LOW VOLTAGE CACHE

FIELD

[0001] This invention relates to the field of cache memories and, in particular, to repairing locations in a cache array.

BACKGROUND

[0002] Providing design flexibility in a cache by allowing a variety of size and associativity choices, while maintaining the speed of the cache in locating/storing a requested element, may be highly advantageous for architectures that utilize a cache. However, when dealing with semiconductor technology, power savings become an evermore prevalent concern, which often leads to limiting performance or sacrificing reliability.

[0003] A typical method to save power, when dealing with cache arrays, includes running the cache arrays at lower voltages than the rest of the device, such as a processor, chipset, or other integrated circuit. Yet, lower voltages may make locations within a cache array more susceptible to soft errors, i.e. flipping of bits that result in an error. Furthermore, locations within the cache are often permanently damaged due to design, manufacture, or after manufacture event, which is often referred to as a hard error.

[0004] Traditionally, to correct hard errors, a line of cache including a hard error is replaced by a spare line. In contrast, a soft error is usually correctable by error correction code (ECC) employed by the cache array. ECC typically refers to logic that detects and may potentially locate, as well as fix errors. As an example, many cache arrays use 1-bit ECC to correct single bit errors per word or cache line.

[0005] Referring to **Figure 1**, a prior art cache ***100*** is illustrated. Often a cache memory is simply a memory array; however, it may also be physically organized or logically viewed as having a plurality of lines/words, such as lines ***106*** through ***113***. In addition, each bit or group of bits in every line of the cache are logically viewable to form a column, such as columns, ***115, 116, 117***, and ***118***. Assuming cache ***100*** includes 1-bit ECC, then single errors per line, such as bit-error ***130*** in cache line ***106***, bit error ***131*** in cache line ***109***, and bit error ***132*** in cache line ***111*** may be detected and fixed.

[0006] However, as voltages to cache *100* are decreased below a critical voltage certain bits begin to fail. Therefore, to ensure reliability in cache *100* the voltage supplied to cache *100* may only be lowered to a critical voltage before lines, such as cache line *113*, begin to have multiple bit errors, such as bit errors *120* and *125*. As a consequence, voltage is not decreased any further to ensure reliability; however, this is at the expense of sacrificing power savings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] The present invention is illustrated by way of example and not intended to be limited by the figures of the accompanying drawings.

[0008] Figure 1 illustrates an embodiment of a prior art cache array.

[0009] Figure 2 illustrates an embodiment of a processor including a cache, a repair cache, and a repair module.

[0010] Figure 3 illustrates embodiments of physical and logically viewable organizations of a cache array.

[0011] Figure 4a illustrates an embodiment of a repair cache to repair a plurality of errors in a cache.

[0012] Figure 4b illustrates an embodiment of a look-up table.

[0013] Figure 5 illustrates an embodiment of a repair cache to repair errors in a cache array with a plurality of sets and ways.

[0014] Figure 6a illustrates an embodiment of flow diagram for repairing a cache location.

[0015] Figure 6b illustrates a specific embodiment of the flow diagram shown in Figure 6a for repairing a cache location.

[0016] Figure 7 illustrates an embodiment of a flow diagram for repairing a bit in a cache based on an optimal configuration of a plurality of bits to be repaired in the cache.

DETAILED DESCRIPTION

[0017] In the following description, numerous specific details are set forth such as examples of specific cache organizations, cache placements, cache sizes, etc. in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that these specific details need not be employed to practice the present invention. In other instances, well known

components or methods, such as error correction, cache design, and cache interfaces, have not been described in detail in order to avoid unnecessarily obscuring the present invention.

[0018] The method and apparatus described herein are for repairing cache memories/arrays. In one embodiment, a cache is implemented on a processing device, such as a microprocessor, an embedded processor, a cell processor, or other integer or floating point execution device. However, the methods and apparatus for repairing a cache are not so limited, as they may be implemented on or in association with any integrated circuit device.

AN EMBODIMENT OF A PROCESSOR

[0019] **Figure 2** illustrates a processor ***200*** including processor logic ***220***, cache ***205***, repair cache ***210***, and repair module ***215***. Processor logic ***220*** may include any logic for execution of instructions or operation on data. In one embodiment, processor logic ***220*** includes logic to execute integer and floating point data out of order and in parallel. As another illustrative example, processor logic ***220*** includes interface logic to interface with external devices, front-end logic to fetch and decode data or instructions, out-of-order logic to assist in out-of-order execution of instructions, and at least an execution unit to execute instructions and/or operate on data.

[0020] Cache ***205*** includes any cache memory array associated with processor ***200*** including a low-level cache, such as a level-one cache, a mid-level cache, such as a trace cache or level-two cache, or a higher level cache, such as a level-three cache. Although, cache ***205***, repair cache ***210***, and repair module ***215*** are illustrated in processor ***200***, each may be integrated within one another, as well as located off processor ***200***. As an example, repair module ***215*** is implemented in repair cache ***210***. Any module, such as module ***215***, may be implemented in hardware, software, firmware, or any combination thereof. Commonly, module boundaries vary and functions are implemented together, as well as separately in different embodiments. Repair cache ***210*** is to repair cache locations in cache ***205*** as determined by repair module ***215***. Repair cache ***210*** and module ***215*** are discussed in more detail in reference to **Figures 4a, 4b and 5.**

PHYSICAL AND LOGICAL ORGANIZATION OF A CACHE

[0021] Turning to **Figure 3**, the organization of a cache array ***302*** is illustrated. Often, a cache is physically organized as an array. Cache array ***302*** is illustrated as a one-dimensional array; however, caches may be multi-dimensional and physically organized in any manner. Yet, logically caches are viewable as having a plurality of sets, ways, lines/words, columns, and cells/bits. Furthermore, a cache location refers to any granularity of breaking down a cache word/line. For example, a cache location includes a single bit, or in the alternative, refers to a grouping of bits. Cache ***302*** includes any type of memory, such as a random access memory (RAM) device or static RAM (SRAM device, used as an intermediary storage for data or instructions.

[0022] Traditionally, there have been three types of cache organizations that have been used: the fully associative, the k-way set associative; and the direct mapped cache organizations. In a fully associative cache, any line of main memory is stored in any cache entry, making cache comparison complex. In a set associative cache, the cache is typically logically broken into ways, where locations from main memory are stored in specific predetermined locations within each way, which simplifies the cache lookup process. The locations within each of the ways that a main memory location may be stored are often logically grouped together and referred to as a set. A direct mapped cache, which is effectively a one way set associative cache, has one location where a plurality of memory locations with common addressing attributes are potentially stored.

[0023] To simplify the discussion of rows and columns within a cache, it is assumed that array ***302*** is logically viewable as a single way with cache lines/words ***305***, ***310***, and ***315***. Each of cache lines, ***305***, ***310***, and ***315*** having a plurality of bits, such as bits ***306***, ***311***, and ***316***. Physical view ***300*** illustrates the cache lines physically organized as a one-dimensional array. Logical view ***350*** illustrates a row and column configuration, where cache lines ***305***, ***310***, and ***315*** are the rows and each of the bits in the same position of the cache lines form a column. As examples, column ***320*** includes first bits ***306***, ***311***, and ***316*** from cache lines ***305***, ***310***, and ***315***.

[0024] As another example, array ***302*** is logically organized into M rows and N columns. Typically, a cache has any integer number of M rows and has a multiple of 16 for N bits in each row. However, a cache is not so limited, as M and N may be any positive integer. As noted above, a cache may be both physically and

logically organized in a different manner than illustrated in **Figure 3.** In fact, an example of a set associative cache is illustrated in and discussed in reference to **Figure 5.**

AN EMBODIMENT OF A REPAIR CACHE

[0025] Turning next to **Figure 4a**, a cache ***405*** and a repair cache ***430*** is illustrated. Cache ***405*** has a plurality of words ***406-413***, which are also referred to as lines, rows, or elements. As stated above, in reference to **Figure 3**, each bit of words ***406-413*** that have the same position or offset within their respective word is logically viewable or logically organized as a column. For example, column ***415*** includes the first bit within each of rows ***406*** through ***413***, while column ***416*** includes the second bit of lines ***406-413***.

[0026] As illustrated, cache ***405*** includes some errors, such as at locations ***421-423*** and ***426-429***. These errors may be either hard errors, soft errors, or a combination of hard and soft errors. A hard error is where a bit or cache location fails due to some manufacturing, hardware, or design defect. A soft error may also include predictable errors due to manufacturing defects, as well as random errors less likely to affect the same location twice, such as the cosmic flipping of a bit due to a low voltage supply. Cache ***405*** also depicts repaired locations ***426***, ***427***, ***428***, and ***429***. Repaired locations ***426-429*** are discussed in more detail in reference to repair cache ***430***.

[0027] Repair cache ***430***, although illustrated separate from and coupled to cache ***405***, is not so limited. In fact, in one embodiment repair cache ***430*** is implemented in cache ***405***. Repair cache ***430*** includes a plurality of "ways." Similar to a cache organization scheme broken into sets and ways, a way in repair cache ***430*** includes at least one bit or location mapped to a column in cache ***405***. As a simplified illustrative example, assume that each line of cache is 128-bits wide, then repair cache ***430*** includes 128 ways. It is also worth noting that repair cache ***430*** is not required to have the same number of ways as bits in cache ***405***. As an example, a repair cache includes 64 ways, even though, a cache associated with the repair cache has a line width of 128-bits. In that example, each way of the repair cache is associated with or mapped to two columns of the cache. In the example in **Figure 4a**, each way in repair cache ***430*** has 1-bit; however, each way in repair cache 430 may instead include a plurality of bits, as shown later in **Figure 5.**

ASSOCIATING REPAIR WAYS/BITS WITH COLUMNS OF A CACHE

[0028] Ways within repair cache ***430*** are associated with the logically viewable columns/vertical stripes in cache ***405***. For simplicity, **Figure 4a** illustrates each way in repair cache ***430*** only having one bit, such as the first way including bit ***438*** and the second way including bit ***436***. Bit ***438*** is associated with column ***415*** of cache ***405***. In one embodiment, column ***415*** is directly mapped to bit ***438***. As a consequence, errors within column ***415*** are mapped to and repaired by bit ***438***.

[0029] In an embodiment where a different number of ways or a plurality of bits in each way are used, other well-known mapping techniques and replacement algorithms may be used to associate a way with a column. In one example, a last used or time replacement algorithm is used to select a repair bit to repair an error in a cache. Other examples are discussed below in reference to **Figure 5**.

[0030] Association of a repair bit or repair way to a logically viewable column of a cache may be implemented in any module. In one embodiment, repair bit ***438*** is associated with column ***415*** through hardware, where repair bit ***438*** is physically associated with the addresses or portion of addresses referencing column ***415***. In another embodiment, repair bit ***438*** is associated with column ***415*** through a combination of hardware and software. As an example, specific bits of a physical or a virtual address that reference the locations in column ***415*** are used to associate bit ***438*** with column ***415***. Therefore, when there is an access that references those specific bits the correct repair bit ***438*** is used to repair the location. As another example, when repair bit ***438*** is used to repair a location in column ***415***, such as repaired location ***428***, a lookup table associates repair bit ***438*** with location column ***415*** and location ***428***.

DETERMINING A LOCATION TO REPAIR

[0031] Determining or targeting a location or bit to repair within a cache, such as cache ***405*** may be based on any number of factors or considerations. Often, in determining a location or bit to repair, it is first determined whether there is an error at a bit or location. Cache ***405***, as discussed above, illustrates a number of cells with errors, such as bit ***421***, ***422***, and ***423***, as well as bits ***426-429***. As stated above, errors ***421-423*** and ***426-429*** include hard errors, soft errors, or a combination of both hard and soft errors. A hard error is relatively easy to detect, as some hardware or manufacturing defect, tends to induce an error consistently. However,

a soft error is usually more difficult to detect, as one time the bit or location is valid and the next time it is accessed there is an error. This is especially true when the voltage supplied to a cache is lowered near a cache location's critical voltage, where certain locations begin to fail.

[0032] As a consequence, determining if a location or bit has an error initially may also be based on a number of factors. In a first embodiment, any cell that produces a flipped bit or an error value is determined to be a bad bit or erred bit. In another embodiment, a less rigid policy is used to detect and determined failed bits. As an example, predictive failure analysis is used to determine if a bit is in error. Here, the number of times a bit or location fails is tracked, i.e. the failure rate of the bit. After a predetermined number of failures or a high enough failure rate, the bit is determined to be a bad or failed bit to be repaired. This potentially avoids labeling a bit as a failed bit or error bit upon too few or too many failures. Other well-known error detection methods, not specifically described herein as to not obscure the invention, may be used to detect and/or identify bad cache locations.

[0033] When a plurality of bits, such as bits ***421-423*** and ***426-429***, are considered erred or failed bits, then a bit or plurality of bits is/are targeted to be repaired. Repairing of bits in a cache is discussed in more detail in reference to **Figure 5**. In one embodiment, an optimal configuration of bits to be repaired is determined. As stated above, a number of factors, such as the location of a bit in a word and column, the failure rate of the bit, the predictive analysis of the failure rate of the bit, the number of errors per word, the number of errors correctable per word, optimizing error correction per word, and other well-known considerations for correcting or repairing a cache location are used to determine an optimal configuration of bits to be repaired. In determining a bit to repair each of these considerations may be taken into account individually or in combination with each other.

[0034] As a specific illustrative example, assume cache ***405*** implements single-bit error correction code (ECC). Therefore, 1-bit per word or row is correctable by ECC. Previously, a part, such as a cache, with two bad bits in a single cache line would make the part unusable, as single bit correct ECC could not recover the correct data. However, utilizing repair cache ***430*** the voltage may be lowered and more errors per word are repaired. In this example, cache line ***409*** and

413 each have three bit-errors. Therefore, to reduce the number of errors per cache line to a correctable amount, which is 1-bit from the implemented 1-bit ECC in this example, two bits in both lines *409* and *413* of the cache are repaired. Note that in column *416*, bit *426* is repaired by repair bit *436*, instead of repairing bit *421*. Bit *426* is targeted to be repaired based on the number of errors per word and the number of errors correctable per word. Specifically, if repair bit *436* was used to repair bit *421* instead, then word *409* would have three errors, only one of which, bit *427*, is repaired. Consequently, 1-bit ECC would not be able to correct both errors and there would be a failure in an access to word *409*.

[0035] In the alternative, if cache *405* implemented multi-bit correcting ECC, such as X-bit error correction, then an optimal configuration of bits to repair is determined to reduce the number of errors per word of cache *405* to X errors. Specifically, if 2-bit correcting ECC is implemented, then optimizing error correction per word includes reducing the number of errors per word to 2 or less. Yet, determining a bit to repair may be based on the aforementioned considerations in a hierarchal manner, as well. In one embodiment, the first level consideration is to reduce the number of errors per word of the cache to two errors. At a second level, where there is an equal choice between repairing two bits, the choice is based on other secondary factors.

[0036] To illustrate, notice that either bit *423* or bit *429* in column *417* are candidates to be repaired by repair bit *439*. In the alternative to the embodiment illustrated, where repair bit *439* repairs bit *429*, repair bit *439* instead repairs bit *423*. To compensate for this change, repair bit *440* repairs bit *441* to keep only 1-bit error in line *413*. Therefore, to choose between bits *423* and *429*, other secondary considerations, such as failure rate, are used to determine which bit to repair. To further the example, assume that bit *429* failed 20 times (75% of the time) and bit *423* failed 5 times (20% of the time), then either bit is selected to be repaired based on the number of failures or their failure rates. Note that not every error or bit needs be repaired, as a number of errors correctable by ECC may still remain.

[0037] These aforementioned embodiments and examples are purely exemplarily, as demonstrated by the fact that cache *405* may implement a multi-bit ECC or need not implement any level of ECC, as well as determine bits to be

bad/failed bits by any known method or target bits to be repaired based on any number of factors.

[0038] Module ***450***, which is used to determine bits to repair within cache ***405***, although depicted in repair cache ***430***, may be implemented in cache ***405***, repair cache ***430***, a processor including cache ***405*** and repair cache ***430***, or any combination thereof. Module ***450*** includes any hardware, software, firmware, code, circuits or combination thereof to determine bits to repair within cache ***405***. Any module, such as module ***450***, may be implemented in hardware, software, firmware, or any combination thereof. Commonly, module boundaries vary and functions are implemented together, as well as separately in different embodiments. Firmware often refers to a combination of hardware and software/microcode routines to perform a function.

[0039] In one embodiment of module ***450***, logic in cache ***405*** including ECC is used to determine which bits have failed, while firmware in repair cache ***430*** tracks the number of failures and/or failure rate of the bits and targets which bits to repair based on the number of errors per word and the failure rate. In another embodiment, firmware purely associated with repair cache ***430*** solely determines bits in cache ***405*** to repair.

[0040] As another simplified illustrative example of module ***450***, firmware tracks failures of bits in cache ***405***. A firmware routine determines bits to repair from an algorithm, based on any individual or combination of the aforementioned factors, such as number of errors per word, number of errors correctable per word, failure rate, and other considerations. In addition, the firmware stores a lookup table to associate any combination of the following with each other: a cache location/bit to be repaired, a cache line/word including the cache location/bit to be repaired, a repair location/bit, or a column of the cache including the cache location/bit to be repaired. As discussed below, a lookup table potentially assists in the repairing of cache locations upon accesses to the cache.

REPAIRING CACHE LOCATIONS

[0041] As noted above in reference to **Figure 4a**, a repair cache or repair module, such as repair cache ***430*** and repair module ***450***, repairs bits in a cache, such as cache ***405***, with repair bits, such as repair bits ***436-440***, associated with the columns in cache ***405*** that include the bits to be repaired. Repairing a bit in cache

405 with a repair bit in repair cache *430* includes replacing the bit to be repaired with the repair bit, storing values to be stored in the bit to be repaired in the repair bit, providing the contents of the repair bit in place of the bit to be repaired, or otherwise assisting in the repair of the bit to be repaired in the cache.

[0042] In one embodiment, repairing a bit in a cache using a repair bit includes determining if a request to a cache references a word in the cache including a bit to be repaired and replacing logical information to be read from the bit in the cache with logical information stored in the repair bit. As an example, assume repair bit *428* is targeted to be repaired. Then either the contents of bit *428* are copied to bit *438* or upon a previous write to line *413* the bit to be stored in bit *428* was stored in bit *438*. Therefore, if a read to line *413* is requested, then bit *428* is repaired by reading the value the logical value stored in repair bit *438* upon fulfilling the read. From a different perspective, the device requesting the read receives cache line *413* from cache *405* with the bit from repair location *438*, instead of the bit stored in location *428*, as location *428* is targeted as bad location with potentially incorrect information.

[0043] In another embodiment, repairing a bit in a cache using a repair bit includes determining if a request to a cache references a word in the cache including a bit to be repaired and writing/storing logical information to be written to the bit in the cache to the repair bit. As an example, assume repair bit *426* is targeted to be repaired. Therefore, if there is a write to word *409*, then bit *426* is repaired by writing the value to be written to bit *426* to repair bit *436*. From a different perspective, the device writing writes to word *409* in cache *405*, and the bit to be written to location *426* is written to repair location *436*.

[0044] As can be seen from above, repair cache *430* may operate transparently as viewed from cache *405's* perspective, as well as operate non-transparently as an intermediary.

[0045] In one embodiment, repair cache *430* and associated logic act as an intermediary between a device accessing cache *405* and cache *405*. In this embodiment, repair cache *430* would receive all requests, such as reads and writes, and repair the cache location during the read and write operations. For example, if a device requests a read from line *413* of cache *405*, then repair cache *430* would intercept the request and upon fulfilling the request would provide the contents of

cache line *413* to the requested device with bit *428* replaced by repair bit *438* and bit *429* replaced by bit *439*.

[0046] However, acting as an intermediary may potentially slow down certain cache access times. For example, upon an access to cache line *406*, where no bits are repaired by repair cache *430*, there would be no reason for repair cache to intercept and forward the request.

[0047] Consequently, in another embodiment, repair cache *430* repairs bits in cache *405* transparently. Repair cache *430* operating transparently is illustrated by contrasting from the example above. Upon an access to cache line *413*, repair cache *430* looks at the location referenced by the request, instead of intercepting the request and forwarding it on. If the request is a read and references cache line *413*, then upon fulfilling the request, cache line *413* is sent out of cache *405* and repair cache *430* replaces cache location *428* with repair bit *438* and cache location *429* with repair bit *439*. As can be seen, the operation of repair cache *430* is essentially transparent to cache *405*, as cache *405* received the request and fulfilled the request unaffected by repair cache's replacement of bits *428* and *429*. In addition, upon a write to cache line *413*, the write continues as it would before, with respect to cache *405*, as all values are written into cache *405*. However, the values for cache location *428* and *429* are also written into repair locations *438* and *439*.

[0048] As mentioned above, in one embodiment, module *450* may track which bit within cache *405* a repair bit in repair cache *430* is to repair. For example, in a lookup table, such as lookup table *460* illustrated in **Figure 4b**, the repair bits, listed in column *465*, are associated with the bits they repair in cache *405*, which are listed as corresponding entries in column *470*. Therefore, when an access references cache line *413* including locations *428* and *429*, by checking lookup table *460* it is able to be determined that bits *428* and *429* are targeted to be repaired. In addition, table 460 includes the information that they are to be repaired by bits *438* and *439*.

[0049] However, module *450* is not limited to the specific implementation of lookup table *460* as shown in **Figure 4b**. For example, the lookup table may include references to addresses of cache lines either in addition to the bits listed in column *470* or instead of bits listed in column *470*. In another embodiment, the table includes all of the repair bits in repair cache *430* and either sets a flag bit or

writes a corresponding bit location when a bit is to be repaired. In the alternative, as illustrated in **Figure 4b**, entries are only created in table 460, when a repair bit is to repair a bad location.

[0050] As another illustrative example, assume that bit ***426*** in cache ***405*** is determined by module ***450*** to be a bit in need of repair. Assume further that module ***450*** determined to repair bit ***426***, because ECC has detected and firmware has tracked bit ***426*** having a high failure rate. Additionally, line ***409***, which includes bit ***426***, has two other errors and only 1-bit ECC is implemented. As a result, module ***450*** stores in a table a starting address and a size of cache line ***409***, the address of bit ***426***, and the address of repair bit ***436***, since repair bit ***436*** is associated with column ***416***. Therefore, upon an access to cache line ***409***, the table is checked and it is determined that cache line ***409*** is referenced. Upon fulfilling the request, module ***405*** repairs bit ***426*** with repair bit ***436***, since they are associated with the cache line in the table. As a modification, multiple cache locations and repair bits are associated with cache lines.

[0051] As an illustrative example, the example above demonstrates that vast possibilities of repair module ***450*** and lookup table ***460***. For example, table ***460*** through hardware, software, firmware, or a combination thereof may associate each of the following with each other to aide in repair of cache locations: a cache location/bit to be repaired, a cache line/word including the cache location/bit to be repaired, a repair location/bit, or a column of the cache including the cache location/bit to be repaired. Also note that addresses, reference to addresses, portions of addresses, or other representations of the foregoing items to be associated may be stored.

AN EMBODIMENT OF A REPAIR CACHE IN A SYSTEM

[0052] Turning to **Figure 5** a system having a microprocessor ***505*** coupled to a memory ***561*** is illustrated. Although not illustrated, a controller hub or other integrated circuit/device is potentially coupled between memory ***561*** and microprocessor ***505***. Microprocessor ***505*** includes any logic or modules for operating on data and executing instructions. As illustrated microprocessor ***505*** includes cache ***510*** and repair cache ***550***. Microprocessor ***505*** may also include other processing logic to execute instructions, operate on data, or communicate with external devices.

[0053] In the example shown, cache ***510*** is illustrated as a two-way set associative cache; however, cache ***510*** may be organized as a direct mapped cache, an associative cache, a fully associative cache, a set-associative cache, or a way-associative cache. Often main memory array ***561*** is logically broken up into pages, such as pages ***565-580***. Main memory ***561*** includes any memory bank or array, such as a random access memory (RAM), a static random access memory (SRAM), a dynamic RAM (DRAM), or other main memory device.

[0054] A location within page ***561*** of memory, such as location ***566*** within page ***565***, is referenced by a main memory address ***566***. Within address ***566*** there are certain bits that are the same for each location within page ***565***, as well as some bits that designate the offset of location ***566*** within page ***565***, i.e. the page offset value. A section ***567*** of address ***566***, which is also referred to as a tag value, is commonly used to map location 566 to a set within cache ***510***, such as set ***540***. In fact, either page ***565***, or alternatively, the locations with the same offset in each of pages ***565-580*** may be mapped to a set by section 567. Set ***540*** includes a cache line in first way ***511*** and a cache line in second way ***512*** that address location ***566*** is mapped to. Therefore, upon a request to main memory location ***566***, the cache need only check set ***540*** to see if a copy of memory location ***566*** exists. Each way in cache ***510*** includes M lines, where each line includes N bits. In **Figure 5**, M and N are illustrated as 8 lines per way and 7 bits per line, respectively.

[0055] Repair cache ***550*** is illustrated coupled to cache ***510***. Repair cache ***550*** includes 14 ways, which is 1-way for each of the N bits of the M lines of cache ***510***, where each one of the ways is associated with at least one of the N bits. As stated above, in another embodiment a repair cache is implemented using 7-ways, where each way has two bits and is mapped to two columns of cache ***510***. However, as shown, repair cache ***550*** includes 14 ways, each way having two bits and being associated with a column of cache ***510***. For example, way ***555*** includes bit ***556*** and bit ***557*** and is associated with column ***541***.

[0056] In one embodiment, cache ***510*** implements ECC. As an illustrative example, cache 510 implements 2-bit ECC with the capability of correcting two bits per line of cache ***510***. Line ***521*** includes three failed bits, bits ***545-547***, which all cannot be corrected by two-bit ECC. Therefore, repair bit ***556*** in way ***555*** associated with column ***541***, is used to repair bit ***545***. Furthermore, line ***522***

includes four bad bits. As a result, the second bit in way *555*, repair bit *557*, is used to repair cache location *548*.

[0057] Repair cache *550* also includes repair module *560*. As stated above, repair module *560* may be implemented in or across repair cache *550*, cache *510*, microprocessor *505*, or any combination thereof. Repair module *560* associates each way and repair bits with a column of the cache. In one embodiment, way *555* is directly mapped to column *541*. In this example, direct mapping of way *555* to column *541* includes repairing only bits in column *545* with repair bits in way *555*. An example of mapping includes: associating a portion of addresses identifying the bits in column *541* with the way *555*, which may be done through comparison logic, a mapping table, lookup logic, or other common association techniques.

[0058] Repair module *560* also determines bits in columns to repair. As described above, repair module *560* selects bits to repair in cache *510* based on an optimal configuration. In determining an optimal configuration and bits to repair any number of factors through an algorithm, software, hardware, or other mechanism may be taken into account, such as the failure rate of bits, the number of errors per cache line, the number of errors correctable per cache line, the type of ECC implemented, the number of repair bits available, the effect of repairing one bit in a column and not repairing another on other cache lines or columns, as well as other considerations. In one embodiment, repair logic *560* dynamically analyzes and re-determines the bits to be corrected based on the aforementioned factors.

[0059] In addition, repair module *560* repairs bits determined to be repaired. As stated above, upon a read to a cache line including a bad bit, a repair bit associated with the column of the bad bit is read from the repair bit and replaces the bad bit upon fulfilling the read request. As an example, assume that processor *505* makes a read request referencing main memory address *566*, which is mapped to set *540*. Cache *510* includes a copy of main memory location *566* in line *521*. Repair module *560* determines that bit *545* is to be repaired. As an illustrative example, it is assumed that repair logic *560* had previously determined that bit *545* was a failed bit and created an entry in a table to associate that bad bit *545* is to be repaired by repair bit *557*. Cache line *521* is then read out of cache *510* and the value from bad bit *5451* is replaced with the value stored in repair bit *557*. In the end, microprocessor *505* receives a valid copy of main memory location *566* from cache

510, as bad bit *545* was replaced with valid information from repair bit *557*. Similarly, in a write to cache *510*, the value to be written to bad location *545* is written to repair bit *557*.

EMBODIMENTS FOR REPAIRING A CACHE LOCATION

[0060] Referring next to **Figure 6a**, an embodiment of a flow diagram for repairing a cache location is illustrated. In block *605*, a request to access a cache is received, where the request references a first cache line. As stated above, a request may reference a cache line in a number of ways, including a tag value associated with the cache line, a physical address associated with the cache line, a virtual address associated with the cache line, a physical/virtual main memory address that is copied into the cache line, or any other common method of referencing a word/line of cache.

[0061] Next, in block *610*, it is determined if the cache line includes a first cache location to be repaired. In one embodiment, a lookup module is used to determine if the cache line includes a first cache location to be repaired. When it is determined that a first cache location is to be repaired, an entry in the lookup module is created or modified to associate a repair location with a cache location and/or a cache line including the cache location. Consequently, upon an access, the lookup module is checked to see if the first cache line includes a first location to be repaired.

[0062] In another embodiment, a multiple tier lookup module is used to first determine if the cache line includes any locations to repair. This may be done by listing references to cache lines that include locations to repair in the lookup module. If there are no matching cache lines then there is no need for further analysis. If a cache line is listed/matched then it is determined if the cache location is within the first cache line or which cache locations to repair. This may be done in a number of ways, such as entering the information in the lookup module or using address comparison to determine the cache location is within the address bounds of the cache line.

[0063] Other mechanisms and modules may be used to determine if the cache line includes a first cache location to be repaired. For example, firmware may store a reference to cache lines with locations to be repaired. Refer to the repairing

cache locations section above for more discussion on determining if the cache line includes a first location to be repaired.

[0064] In block ***615***, if the first cache line includes the first cache location to be repaired, then the first cache location is repaired from the repair cache location associated with the first cache location, upon fulfilling the request to access the cache. As stated above, the first cache location may be associated through a module, which includes a lookup table. In one embodiment, the first cache location is associated with the repair cache location through a mapping of the repair cache location to a logically viewable column that includes the first cache location. Here, the repair cache location is directly mapped to a column of the cache, and upon determining the first cache location is to be repaired, the repair cache location directly mapped to the column including the first cache location is used.

[0065] Turning to **Figure 6b**, a specific embodiment of a flow diagram for repairing the first cache location with a repair cache location is illustrated. In block ***616***, upon fulfilling a write request to the cache, a logic value to be written to the first cache location is transparently written to the repair cache location associated with the first cache location. An example includes, writing to a line of cache, wherein the line of cache has a first bad bit. The write to the cache lines occurs, from the cache's perspective, as normal. However, when it is determined that the write is to a line of cache which includes a bad bit, as in block ***610***, then the value to be written to the first bad bit is also written to a repair bit mapped to a logical column of the cache including the first bad bit location.

[0066] In block ***617***, upon fulfilling a read request from the cache, a logic value to be read from the first cache location is read from the repair cache location associated with the first cache location. Continuing the example from above, a request for the line of the cache is determined to include the first bad bit location, in block ***610***. The line of the cache is read and provided by the cache, from the cache's perspective, as fulfilling a normal request. However, the value stored in the repair bit associated with the first bad bit is read from the repair bit location and replaces the value from the first bad bit location. As a result, the cache operates normally, while the data stored and read out for the bad bit is in the repair bit. This perspective from the cache's point of view refers to the transparency of the write/store and the read/replace, as the cache operates normally.

[0067] Turning to **Figure 7**, an embodiment of a flow diagram for repairing a cache based on an optimal configuration of a plurality of bits to be repaired is illustrated. In block *705*, an optimal configuration of a plurality of bits in a cache to be repaired with a repair cache is determined. The repair cache includes a repair bit directly mapped to each logical vertical stripe/column of the cache.

[0068] As stated above, the cache may have any number of bits determined to be bad bits. As a consequence, a choice of which bits to repair may be made based on any number of factors. In one embodiment, to choose the optimal configuration an algorithm is used that selects a configuration of bits to be repaired based on any one or combination of the following: the failure rate of bits, the number of bad bits per cache line, the number of bad bits correctable per cache line, the type of ECC implemented, the number of repair bits available, and the effect of repairing one bit in a vertical stripe and not repairing another within other cache lines or columns.

[0069] In one embodiment, the determination of the optimal configuration of bits is dynamically re-evaluated or dynamically determined. For example, assume it is determined that a first bit is to be repaired based on its failure rate and location within a line with other bad bits. However, if at a different time conditions change, another bit within the same vertical stripe as the first bit may be selected to be repaired instead of the first bit. This allows the repair cache ultimate flexibility in ensuring the most optimal configuration of bits to repair is determined under changing conditions.

[0070] Next, in block *710*, upon an access to the cache referencing a word of the cache including a first bit, the first bit of the plurality of bits in the cache determined to be repaired with a repair bit directly mapped to a logical vertical stripe including the first bit is repaired. In one embodiment, repairing the first bit of the plurality of bits includes writing a value to be stored in the first bit to the repair bit, if the access to the cache referencing the word including the first bit is a write operation. In another example, repairing the first bit of the plurality of bits includes reading a value stored in the repair bit, if the access to the cache referencing a word of the cache including the first bit is a read operation.

[0071] As illustrated above, a repair cache may be used to increases the efficiency and accuracy of a cache, which is run at lower voltages. Previously, a cache would have a lower voltage limit that it could not decrease supply voltage under, as it would cause too many errors per cache line to be corrected by ECC. However, by including a repair cache, errors per logical column of the cache may be repaired allowing the voltage supplied to be decreased without sacrificing the number of errors that are corrected by ECC. Additionally, not every error within a cache is required to be repaired. Potentially, only enough errors to ensure valid data, through usage in combination with correcting ECC, are repaired. Moreover, the bits chosen to be corrected/repaired may dynamically change over time to select a more optimal configuration of repair bits. Consequently, power is saved without sacrificing accuracy of a cache.

[0072] In the foregoing specification, a detailed description has been given with reference to specific exemplary embodiments. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative sense rather than a restrictive sense. Furthermore, the foregoing use of embodiment and other exemplarily language does not necessarily refer to the same embodiment or the same example, but may refer to different and distinct embodiments, as well as potentially the same embodiment.

CLAIMS

What is claimed is:

1. An apparatus comprising:
   a cache having a plurality of words logically viewable in a plurality of columns;
   a repair cache having a plurality of repair ways, each of the plurality of repair ways associated with one of the plurality of columns;
   a repair module to repair a first bit with a repair bit in a first way of the plurality of repair ways associated with a first column including the first bit.

2. The apparatus of claim 1, wherein the repair module dynamically determines the first bit to repair, and wherein determining a first bit to repair is based on optimizing error correction per word of the plurality of words.

3. The apparatus of claim 2, wherein the cache implements one-bit error correction code (ECC).

4. The apparatus of claim 2, wherein the cache implements multi-bit error correction code (ECC) per word of the plurality of words.

5. The apparatus of claim 2, wherein determining a first bit to repair is further based on the failure rate of the first bit.

6. The apparatus of claim 1, wherein each of the plurality of repair ways includes one repair bit, and wherein the one repair bit of the plurality of repair ways is directly mapped to one of the plurality of columns.

7. The apparatus of claim 1, wherein each of the plurality of repair ways includes a plurality of repair bits, and wherein a replacement algorithm is used to select the repair bit in the first way to repair the first bit.

8. The apparatus of claim 1, wherein repairing the first bit using a repair bit in a first way of the plurality of repair ways associated with the first column comprises:

determining if a cache write references a cache word of the plurality of words including the first bit;

writing logical information to be written to the first bit by the cache write to the repair bit in the repair cache.

9. The apparatus of claim 1, wherein repairing the first bit using a repair bit in a first way of the plurality of repair ways associated with the first column comprises:

determining if a cache read references a cache word of the plurality of words including the first bit;

replacing logical information to be read from the first bit with logical information from the repair bit in the repair cache.

10. The apparatus of claim 9, wherein replacing logical information with logical information from the repair bit in the repair cache is transparent to the cache.

11. An apparatus comprising:

a memory array logically organized into a M rows and N columns;

a repair array having N replacement bits, each one of the N replacement bits being directly mapped to one of the N columns; and

a module to target a first bit in a first column of the N columns to repair and repair the first bit with a replacement bit of the N replacement bits directly mapped to the first column, upon an access to a row including the first bit.

12. The apparatus of claim 11, wherein M is a positive integer and N is an integer multiple of 16.

13. The apparatus of claim 11, wherein the memory array is a cache memory, and wherein the repair array is a repair cache.

14. The apparatus of claim 13, further comprising X bit error correction code (ECC) to fix X bits per row of the M rows.

15. The apparatus of claim 14, wherein the module targets the first bit based on an algorithm to reduce a number of errors per row of the M rows to X errors.

16. The apparatus of claim 15, wherein the module dynamically re-targets the first bit based on the algorithm to reduce a number of errors per row of the M rows to X errors.

17. A system comprising:
a microprocessor including
a cache having a plurality of lines, wherein the cache is logically viewable in a plurality of columns;
a repair module to
associate a repair bit with each one of the plurality of columns,
determine a first bit in a first column of the plurality of columns to repair, and
repair the first bit with the repair bit associated with the first column, upon an access to a line including the first bit;
a system memory including a plurality of elements coupled to the microprocessor, wherein the cache is to store local copies of the contents of the plurality elements.

18. The system of claim 17, wherein the microprocessor is capable of parallel out-of-order integer and floating point execution.

19. The system of claim 17, wherein the cache has an organization selected from a group consisting of a direct mapped cache, a set associative cache, and a fully associative cache.

20. The system of claim 17, wherein the repair logic comprises firmware, and wherein a repair bit is associated with each of the plurality of columns through direct mapping of a repair bit to each of the plurality of columns.

21. The system of claim 17, wherein determining a first bit in a first column of the plurality of columns to repair comprises
tracking a failure rate of the first bit;
determining a number of errors in a line including the first bit;
determining the first bit to repair based on the failure rate of the first bit and the number of errors in the line including the first bit.

22. The system of claim 17, wherein repairing the first bit with the repair bit associated with the first column, upon an access to a line including the first bit comprises
storing a value to be stored in the first bit to the repair bit associated with the first column, if the access to the line including the first bit is a write to the line including the first bit;
replacing a value to be written out from the first bit with a value stored in the repair bit associated with the first column, if the access to the line including the first bit is a read from the line including the first bit.

23. The system of claim 17, wherein the system memory is a memory device selected from a group consisting of a random access memory (RAM), double data rate (DDR) memory device, and a static RAM (SRAM) memory device.

24. A method comprising:
receiving a request to access a cache, the request referencing a first cache line;
determining if the first cache line includes a first cache location to be repaired; and
repairing the first cache location from a repair cache location associated with the first cache location, upon fulfilling the request to access the

cache, if the first cache line includes the first cache location to be repaired.

25. The method of claim 24, wherein the request to access the cache is a write to the cache, and wherein repairing the first cache location comprises transparently writing a logic value to be written to the first cache location to the repair cache location associated with the first cache location, upon fulfilling the write to the cache.

26. The method of claim 24, wherein the request to access the cache is a read from the cache, and wherein repairing the first cache location comprises: transparently reading a logic value to be read from the first cache location from the repair cache location associated with the first cache location, upon fulfilling the read from the cache.

27. The method of claim 24, wherein determining if the first cache line includes a first cache location to be repaired comprises:

determining the first cache line does not include the first cache location to be repaired, if a lookup module does not associate the first cache location with the first cache line;

determining the first cache line does include the first cache location to be repaired, if the lookup module associates the first cache location with the first cache line.

28. The method of claim 24, wherein associating the repair cache location with the first cache location includes directly mapping the repair cache location to a column of the cache including the first cache location.

29. The method of claim 28, wherein the first cache location is a first bit and the repair cache location is a repair bit.

30. The method of claim 29, wherein reference to a first cache line includes a tag value to identify the first line of the cache.

31. The method of claim 24, wherein the request to access the cache is a write to the cache, and wherein repairing the first cache location comprises transparently writing a logic value to be written to the first cache location to the repair cache location associated with the first cache location, upon fulfilling the write to the cache.

32. A method comprising:

determining an optimal configuration of a plurality of bits in a cache to be repaired with a repair cache, the repair cache including a repair bit directly mapped to each logical vertical stripe of the cache; and

repairing a first bit of the plurality of bits in the cache determined to be repaired with a repair bit directly mapped to a logical vertical stripe including the first bit, upon an access to the cache referencing a word of the cache including the first bit.

33. The method of claim 32, wherein determining an optimal configuration of a plurality of bits in a cache to be repaired is based on reducing a first number of errors per word of the cache to a correctable number of errors per word of the cache.

34. The method of claim 33, wherein the cache includes 2-bit error correction code (ECC), and wherein the correctable number of errors per word of the cache is two.

35. The method of claim 32, wherein determining an optimal configuration of a plurality of bits in a cache to be repaired is based on a failure rate of the plurality of bits.

36. The method of claim 32, wherein repairing the first bit of the plurality of bits comprises:

writing a value to be stored in the first bit to the repair bit directly mapped to the logical vertical stripe including the first bit, if the access to the cache referencing the word including the first bit is a write operation;

reading a value stored in the repair bit directly mapped to the logical vertical stripe including the first bit, if the access to the cache referencing a word of the cache including the first bit is a read operation.

37. The method of claim 32, wherein determining an optimal configuration of the plurality of bits to be repaired is dynamically re-evaluated.

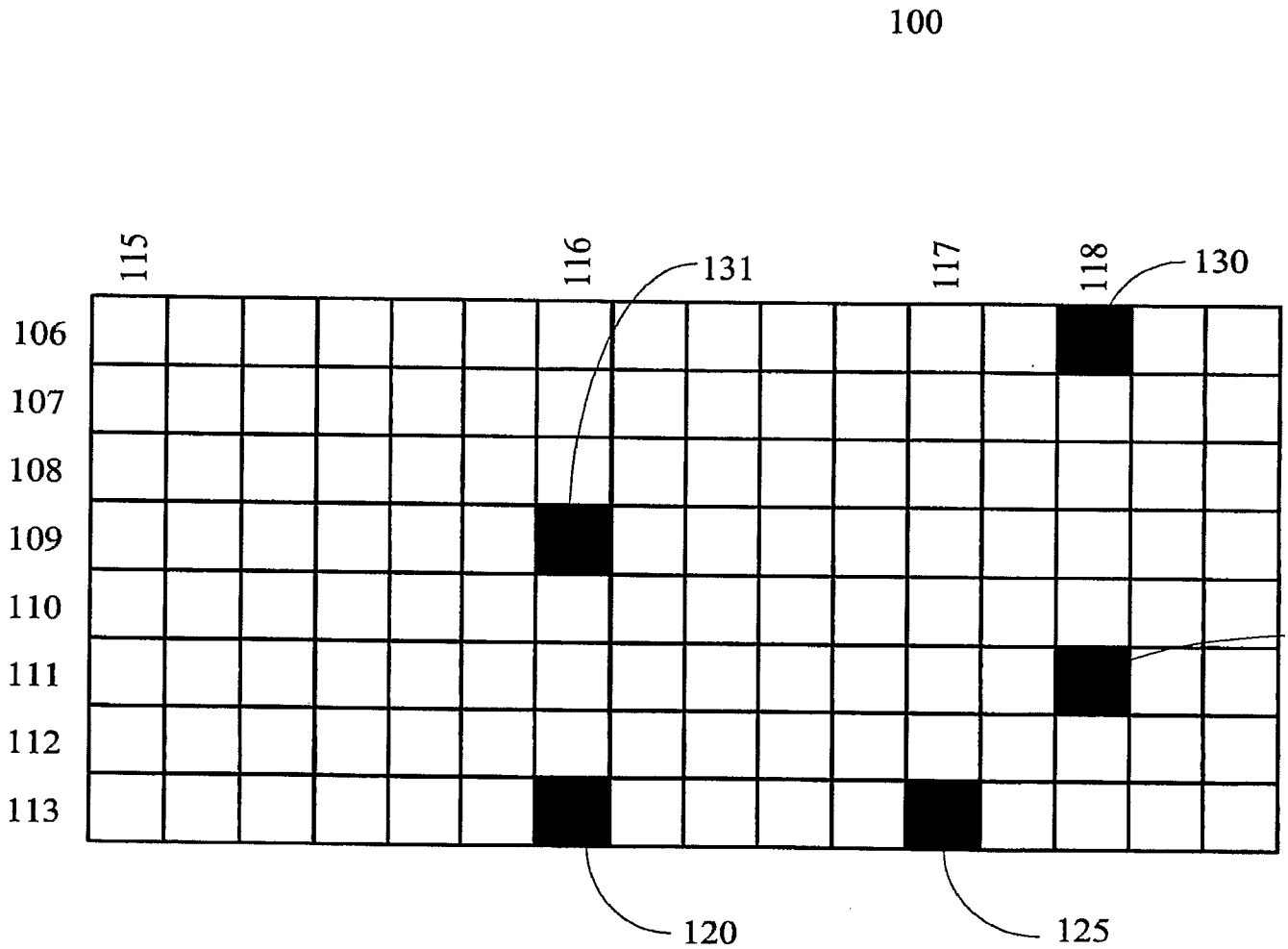

FIG. 1
Prior Art

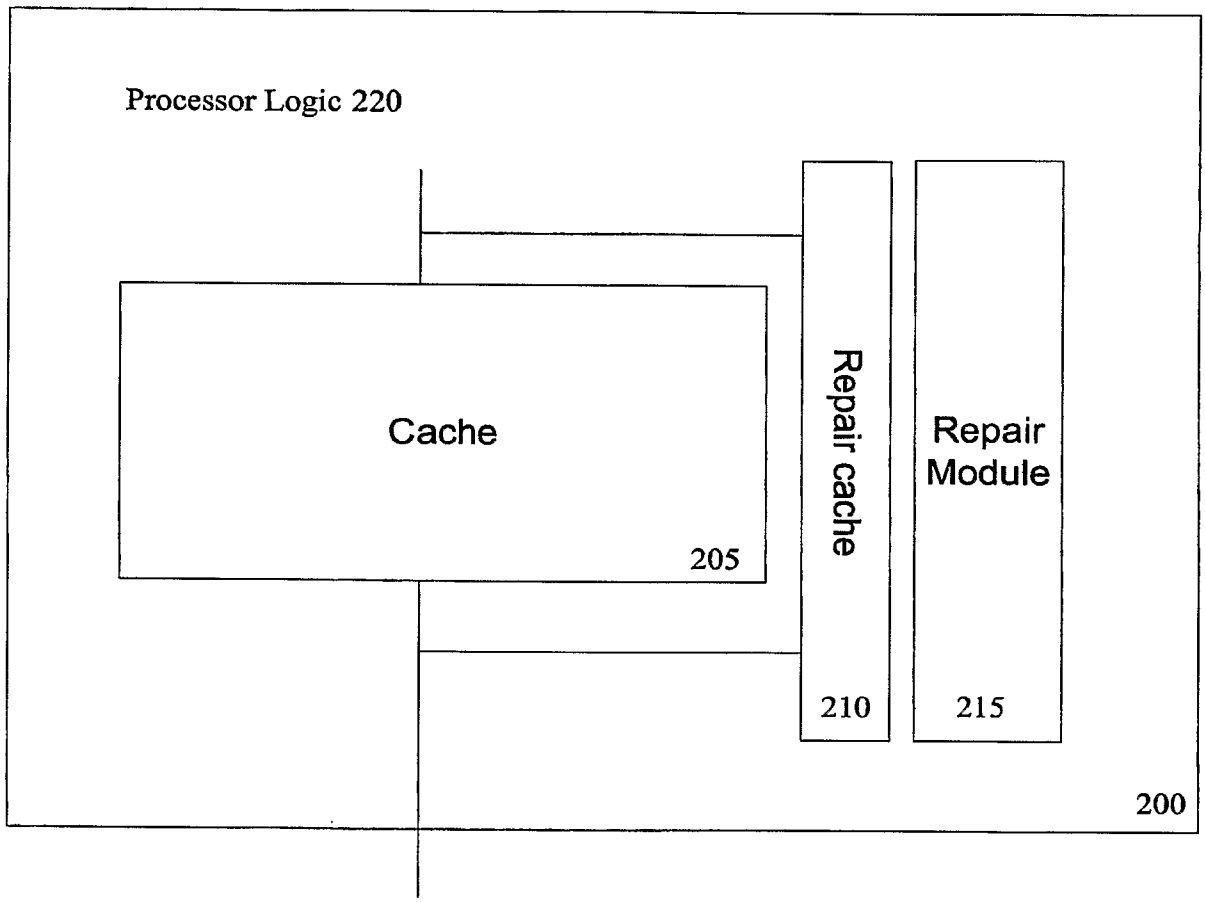
Processor Logic 220
Cache
205
Repair cache
210
Repair Module
215
200

FIG. 2

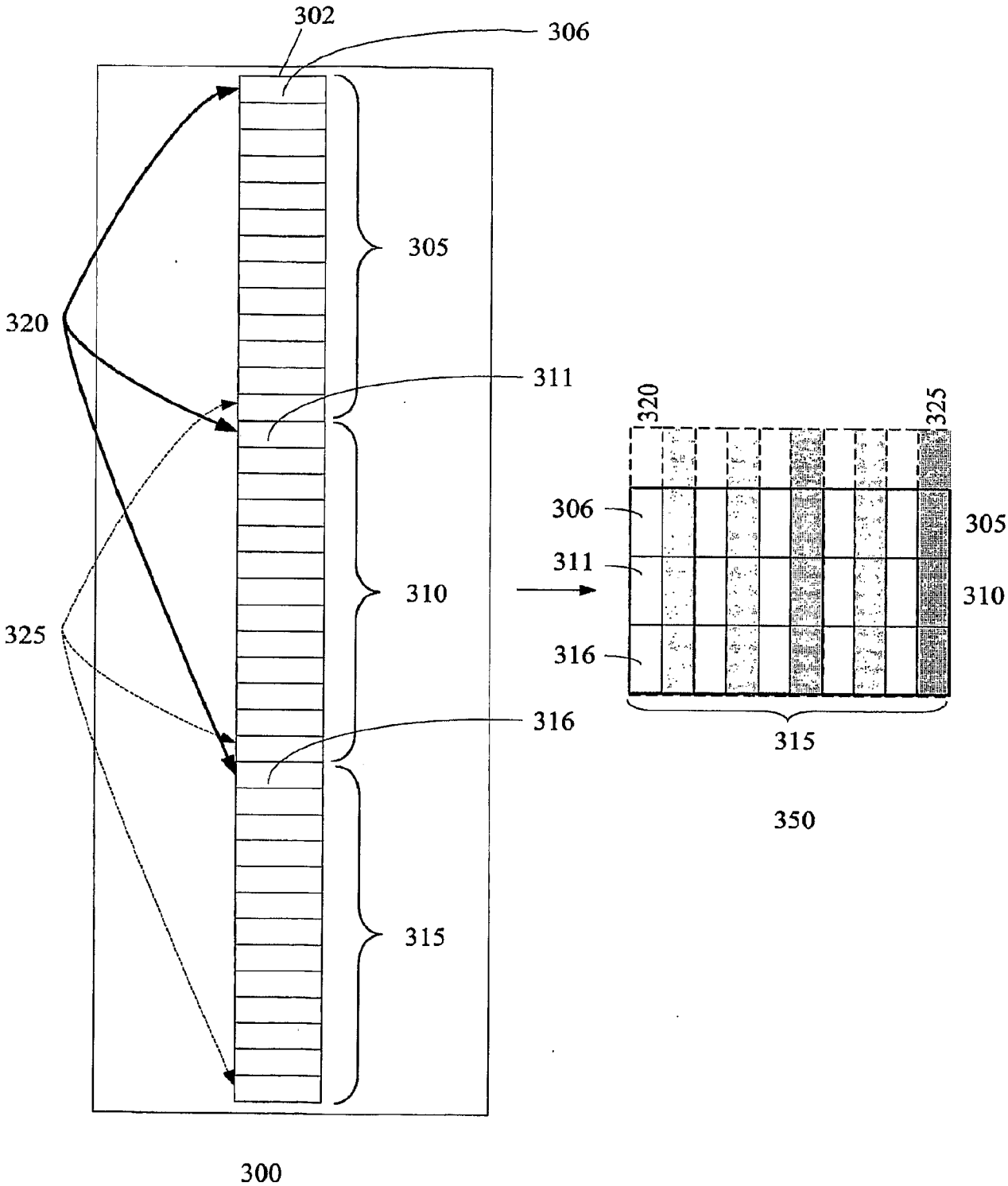
302
306
305
320
311
310
325
316
315
300
320
325
306
311
316
305
310
315
350

FIG. 3

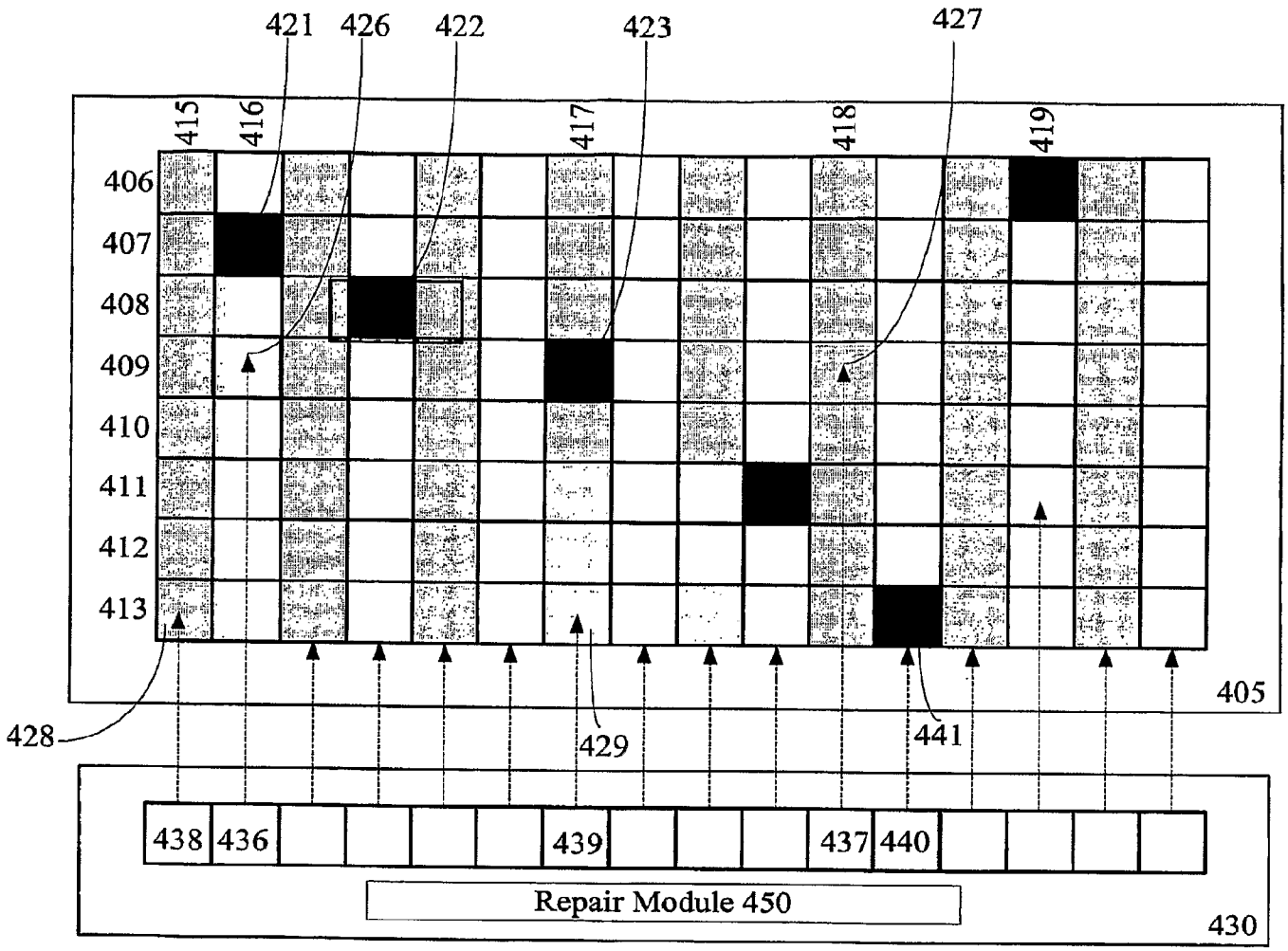

FIG. 4a

460

465 Repair Bits	470 Bad Bits
438	428
436	426
439	429
437	427
440	441

FIG. 4b

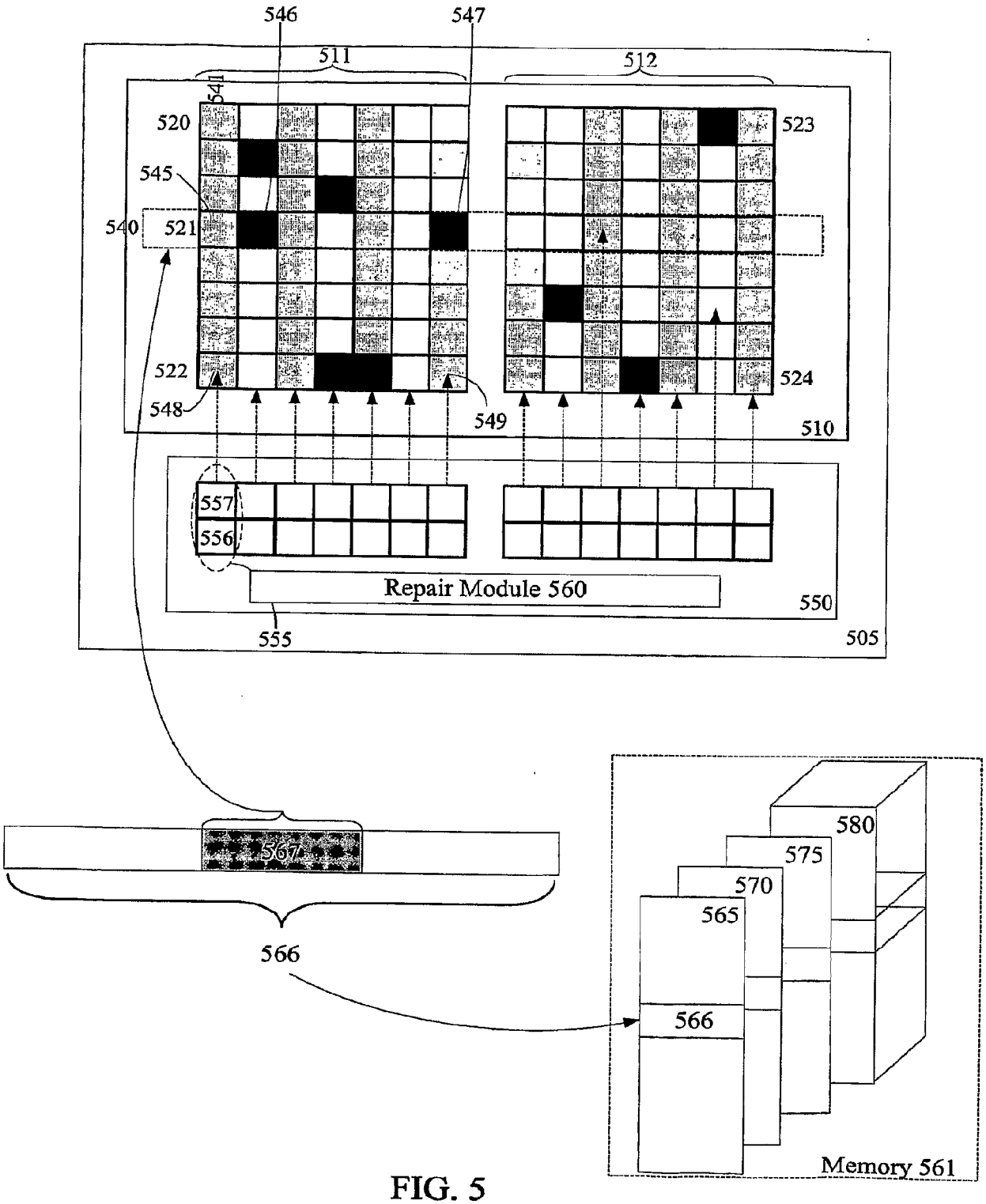
546
547
511
512
541
520
523
545
540
521
522
524
548
549
510
557
556
Repair Module 560
550
555
505
567
566
580
575
570
565
566
Memory 561

FIG. 5

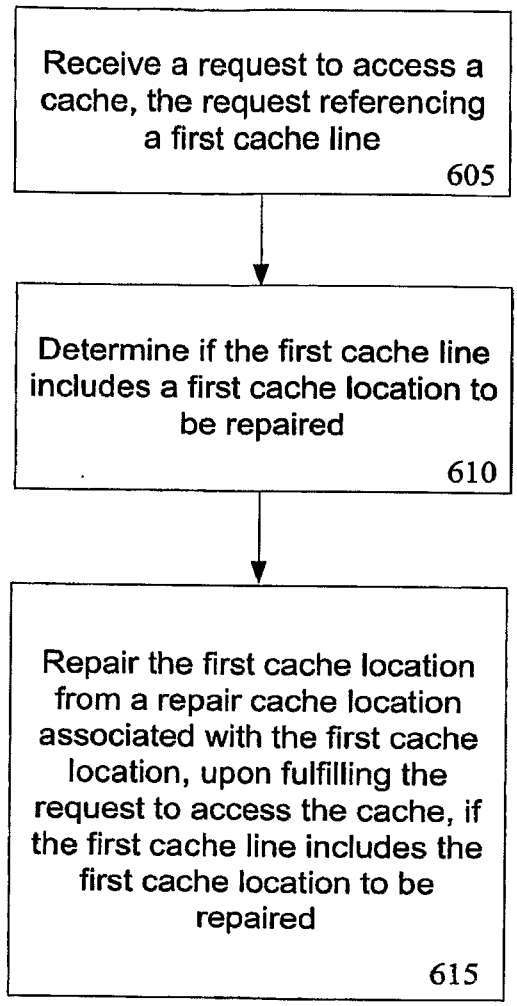
Receive a request to access a cache, the request referencing a first cache line
605
Determine if the first cache line includes a first cache location to be repaired
610
Repair the first cache location from a repair cache location associated with the first cache location, upon fulfilling the request to access the cache, if the first cache line includes the first cache location to be repaired
615

FIG. 6a

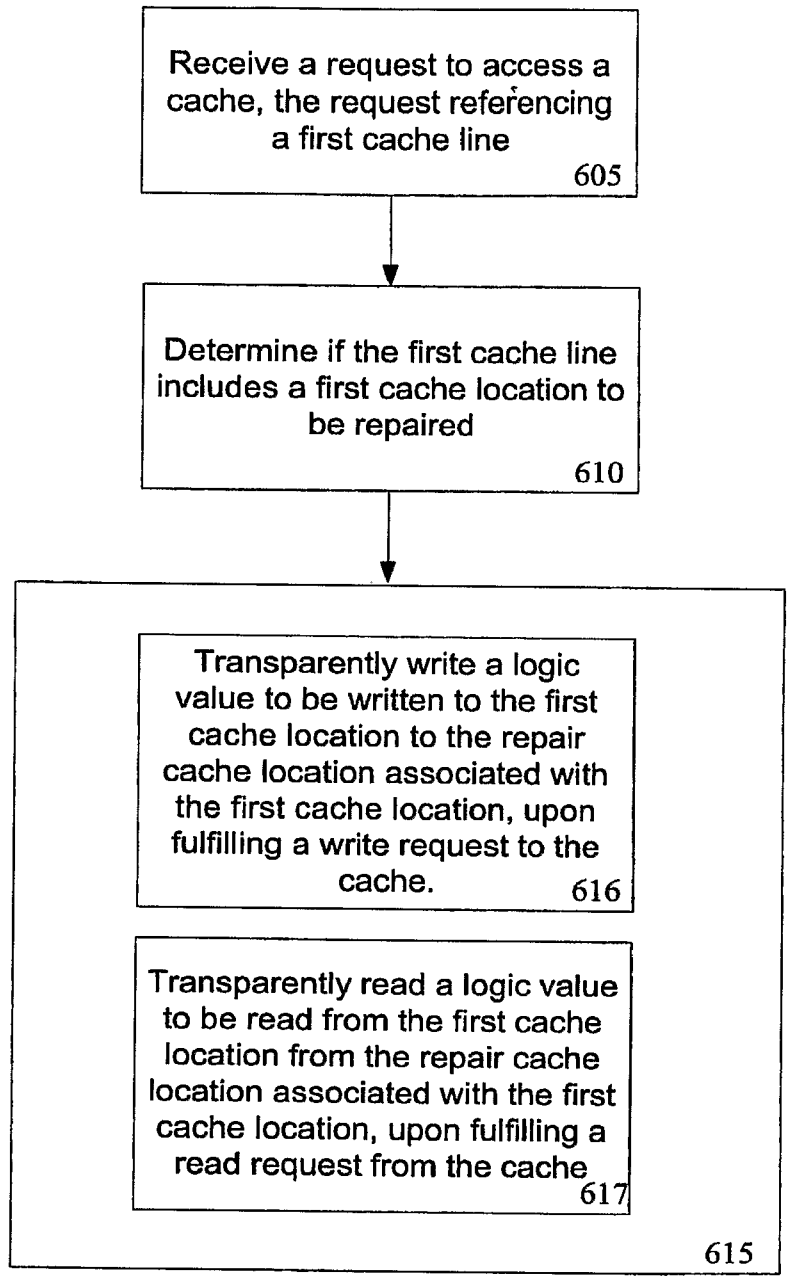
Receive a request to access a cache, the request referencing a first cache line
605
Determine if the first cache line includes a first cache location to be repaired
610
Transparently write a logic value to be written to the first cache location to the repair cache location associated with the first cache location, upon fulfilling a write request to the cache.
616
Transparently read a logic value to be read from the first cache location from the repair cache location associated with the first cache location, upon fulfilling a read request from the cache
617
615

FIG. 6b

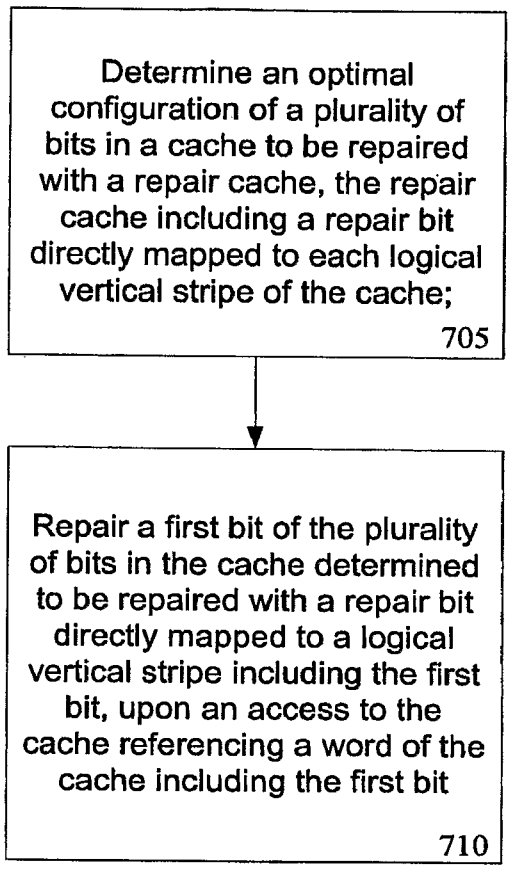
Determine an optimal configuration of a plurality of bits in a cache to be repaired with a repair cache, the repair cache including a repair bit directly mapped to each logical vertical stripe of the cache;
705
Repair a first bit of the plurality of bits in the cache determined to be repaired with a repair bit directly mapped to a logical vertical stripe including the first bit, upon an access to the cache referencing a word of the cache including the first bit
710

FIG. 7