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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 April 2006 (13.04.2006)

PCT

(10) International Publication Number
WO 2006/038991 A2

(51) International Patent Classification:
G06F 9/26 (2006.01)

(21) International Application Number:
PCT/US2005/029135

(22) International Filing Date: 16 August 2005 (16.08.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

10/920,610	17 August 2004 (17.08.2004)	US
10/920,682	17 August 2004 (17.08.2004)	US
10/920,995	17 August 2004 (17.08.2004)	US
10/921,026	17 August 2004 (17.08.2004)	US

(71) Applicant *(for all designated States except US)*: **NVIDIA CORPORATION** [US/US]; 2701 San Tomas Expressway, Santa Clara, California 95050 (US).

(72) Inventors; and

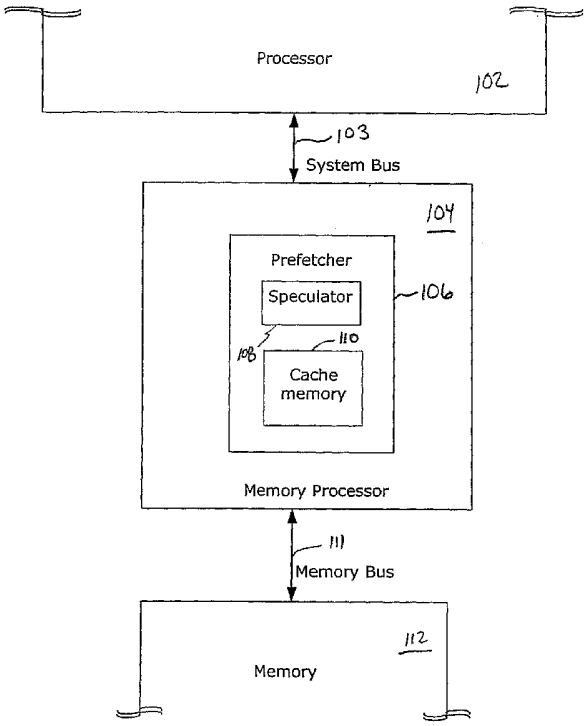
(75) Inventors/Applicants *(for US only)*: **HAKURA, Ziyad, S.** [JO/US]; 2400 W. El Camino Real, #401, Mountain View, California 94040 (US). **DANILAK, Radoslav** [SK/US]; 1850 Pruneridge Ave., Apt. #25, Santa Clara, California 95050 (US). **SIMERAL, Brad, W.** [US/US]; 1049 Dolores Street, Condo #3, San Francisco, California 94110 (US). **LANGENDORF, Brian, Keith** [US/US]; 272 Semples Crossing, Benicia, California 94510 (US). **PESCADOR, Stefano, A.** [CA/US]; 268 Lakemuir Dr., Sunnyvale, California 94086 (US). **VYSHETSKY, Dmitry** [IL/US]; 11627 Cedar Spring Ct., Cupertino, California 95014 (US).

(74) Agents: **BACKUS, Kenneth, R.** et al.; c/o Cooley Godward, LLP, Five Palo Alto Square, 3000 El. Camino Road, Palo Alto, CA 94306-2155 (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, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL,

[Continued on next page]

(54) Title: SYSTEM, APPARATUS AND METHOD FOR PREDICTING VARIOUS TYPES OF ACCESSES TO A MEMORY AND FOR MANAGING PREDICTIONS ASSOCIATED WITH A CACHE MEMORY



(57) Abstract: A system, apparatus, and method are disclosed for predicting accesses to memory. In one embodiment, an exemplary apparatus comprises a processor configured to execute program instructions and process program data, a memory including the program instructions and the program data, and a memory processor. The memory processor can include a speculator configured to receive an address containing the program instructions or the program data. Such a speculator can comprise a sequential predictor and a nonsequential predictor for generating a configurable number of sequential and nonsequential addresses respectively. In one embodiment, a prefetcher implements the apparatus. In various embodiments, the speculator can also include any of the following: an expediter, a suppressor, an inventory, an inventory filter, a post-inventory filter and a data return cache memory, which can include a short term and a long term cache, among other things.

WO 2006/038991 A2

SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, 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.

SYSTEM, APPARATUS AND METHOD FOR PREDICTING VARIOUS TYPES OF ACCESSES TO A MEMORY AND FOR MANAGING PREDICTIONS ASSOCIATED WITH A CACHE MEMORY

BRIEF DESCRIPTION OF THE INVENTION

[0001] This invention relates generally to computing systems, and more particularly, to predicting sequential and nonsequential accesses to a memory, for example, by generating a configurable amount of predictions as well as by suppressing and filtering predictions against, for example, a prediction inventory and/or a multi-level cache.

BACKGROUND OF THE INVENTION

[0002] Prefetchers are used to fetch program instructions and program data so that a processor can readily avail itself of the retrieved information as it is needed. The prefetcher predicts which instructions and data the processor might use in the future so that the processor need not wait for the instructions or data to be accessed from system memory, which typically operates at a slower rate than the processor. With a prefetcher implemented between a processor and system memory, the processor is less likely to remain idle as it waits for requested data from memory. As such, prefetchers generally improve processor performance.

[0003] Generally, the more predictions generated by a prefetcher, the more likely that the prefetcher can arrange to have the necessary instructions and data available for a processor, thereby decreasing the latency of a processor. But conventional prefetchers typically lack sufficient management of the prediction process. Without such management, these prefetchers are prone to overload memory resources when the amount of predicted addresses exceeds what the prefetchers can handle. So to prevent resource overload, traditional prefetchers tend to be conservative in generating predictions so as not to generate an amount of predictions that could overload either the prefetcher or the memory resource. In addition, conventional prefetchers typically generate predictions without regard to the costs of implementing such prediction processes, thereby failing to realize the benefits from streamlining the prediction process and the amount of resources necessary for supporting it. In particular, traditional types of prefetchers primarily rely on standard techniques to produce predictions that are sequential in nature and do not store predictions in a manner that conserves resources, whether computational or otherwise. Also, conventional prefetchers

usually lack sufficient management of the prediction process, and therefore are prone to overload computational and memory resources when the amount of predicted addresses exceeds what the prefetchers can handle. So to prevent resource overload, these prefetchers tend to be conservative in generating predictions so as not to generate an amount of predictions that could overload the prefetcher. Further, many conventional prefetchers lack capabilities to manage predictions after they are generated and before a processor requests those predictions. Usually these prefetchers store prefetch data in a single cache memory, which is typically lacking in functionality to limit predictions that are superfluous with respect to those already stored in the cache. The cache memories used in traditional prefetchers are merely for storing data and are not sufficiently designed for effectively managing predicted addresses stored therein as they.

[0004] In view of the foregoing, it would be desirable to provide a system, an apparatus and a method for effectively predicting accesses to memory. Ideally, an exemplary system, apparatus or method would minimize or eliminate at least the above-described drawback.

SUMMARY OF THE INVENTION

[0005] A system, apparatus, and method are disclosed for predicting accesses to memory. In one embodiment, an exemplary apparatus comprises a processor configured to execute program instructions and process program data, a memory including the program instructions and the program data, and a memory processor. The memory processor can include a speculator configured to receive an address containing the program instructions or the program data. Such a speculator can comprise a sequential predictor for generating a configurable number of sequential addresses. The speculator can also include a nonsequential predictor configured to associate a subset of addresses to the address. The nonsequential predictor can also be configured to predict a group of addresses based on at least one address of the subset, wherein at least one address of the subset is unpatternable to the address. In one embodiment, an exemplary nonsequential predictor anticipates accesses to a memory. The nonsequential predictor includes a prediction generator configured to generate indexes and tags from addresses. Also, the nonsequential predictor includes a target cache coupled to the prediction generator. The target cache includes a number of portions of memory each having memory locations for storing trigger-target associations. A trigger-

target association stored in a first portion of memory is associated with a higher priority than another trigger-target association stored in a second portion of memory

[0006] In an embodiment of the present invention, the apparatus includes a prediction inventory, which includes queues each configured to maintain a group of items. The group of items typically includes a triggering address that corresponds to the group of items. Each item of the group is of one type of prediction. Also, the apparatus includes an inventory filter configured to compare the number of predictions against at least one of the queues having the same prediction type as the number of predictions. In some cases, the inventory filter is configured to compare the number of predictions against at least one other of the queues having a different prediction type. For example, a number of forward sequential predictions can be filtered against a back queue, or the like. In at least one embodiment, an apparatus includes a data return cache memory to manage predictive accesses to a memory. The data return cache memory can include a short term cache memory configured to store predictions having an age, for example, less than a threshold and a long term cache memory configured to store the predictions having an age, for example, greater than or equal to the threshold. The long term cache memory typically has more memory capacity than the short term cache. The prefetcher also can include an interface configured to detect in parallel, such as during one cycle of operation or over two cycles, whether multiple predictions are stored in either the short term cache memory or the long term cache memory, or both, wherein the interface uses at least two representations of each of the multiple predictions when examining the short term cache and the long term cache memories.

BRIEF DESCRIPTION OF THE FIGURES

[0007] The invention is more fully appreciated in connection with the following detailed description taken in conjunction with the accompanying drawings, in which:

[0008] FIG. 1 is a block diagram illustrating an exemplary speculator implemented with a memory processor, according to a specific embodiment of the present invention;

[0009] FIG. 2 depicts an exemplary speculator according to one embodiment of the present invention;

[0010] FIG. 3A depicts an exemplary forward sequential predictor in accordance with a specific embodiment of the present invention;

[0011] FIG. 3B depicts an exemplary blind back sequential predictor in accordance with a specific embodiment of the present invention;

[0012] FIG. 3C depicts an exemplary back sector sequential predictor in accordance with a specific embodiment of the present invention;

[0013] FIG. 3D depicts the behavior of an exemplary reverse sequential predictor in accordance with a specific embodiment of the present invention;

[0014] FIG. 4 illustrates an exemplary nonsequential predictor, according to one embodiment of the present invention;

[0015] FIG. 5 illustrates an exemplary technique of suppressing nonsequential predictions for a stream of interleaved sequential addresses, according to one embodiment of the present invention;

[0016] FIG. 6 illustrates an exemplary technique of suppressing nonsequential predictions for interleaved sequential addresses over multiple threads, according to one embodiment of the present invention;

[0017] FIG. 7 illustrates another technique for suppressing nonsequential predictions based on the arrival times of the base address and a nonsequential address, according to a specific embodiment of the present invention;

[0018] FIG. 8 depicts an exemplary technique for expediting generation of predictions, according to a specific embodiment of the present invention;

[0019] FIG. 9 shows another exemplary speculator including a prediction filter, according to one embodiment of the present invention;

[0020] FIG. 10 is a block diagram illustrating a prefetcher implementing an exemplary nonsequential predictor, according to a specific embodiment of the present invention;

[0021] FIG. 11 depicts an exemplary nonsequential predictor in accordance with one embodiment of the present invention;

[0022] FIG. 12 illustrates an exemplary prediction generator, according to an embodiment of the present invention;

[0023] FIG. 13 illustrates an exemplary priority adjuster, according to a specific embodiment of the present invention;

[0024] FIG. 14 depicts an exemplary pipeline for operating a nonsequential predictor generator when forming nonsequential predictions, according to a specific embodiment of the present invention;

[0025] FIG. 15 depicts an exemplary pipeline for operating a priority adjuster to prioritize nonsequential predictions, according to a specific embodiment of the present invention;

[0026] FIG. 16 is a block diagram illustrating an exemplary prediction inventory within a memory processor, according to a specific embodiment of the present invention;

[0027] FIG. 17 depicts an exemplary prediction inventory in accordance with one embodiment of the present invention;

[0028] FIG. 18 illustrates an example of an inventory filter in accordance with a specific embodiment of the present invention;

[0029] FIGs. 19A and 19B are diagrams illustrating exemplary techniques of filtering out redundancies, according to a specific embodiment of the present invention;

[0030] FIG. 20 shows another exemplary prediction inventory disposed within a prefetcher, according to one embodiment of the present invention;

[0031] FIG. 21 is a block diagram illustrating a prefetcher that includes an exemplary cache memory, according to a specific embodiment of the present invention;

[0032] FIG. 22 illustrates an exemplary multi-level cache, according to one embodiment of the present invention;

[0033] FIG. 23A illustrates an exemplary first query interface for a first address store in accordance with a specific embodiment of the present invention;

[0034] FIG. 23B shows that any number of input addresses can be examined in parallel using the first query interface of FIG. 23A;

[0035] FIG. 24 illustrates an exemplary second query interface for a second address store in accordance with a specific embodiment of the present invention;

[0036] FIG. 25A depicts possible arrangements of exemplary addresses (or representations thereof) as stored in a second address store, according to one embodiment of the present invention;

[0037] FIG. 25B depicts an exemplary hit generator that generates results based on unordered addresses and ordered valid bits, according to an embodiment of the present invention;

[0038] FIG. 26 is a schematic representation of a component for generating one result, R, of the hit generator of FIG. 25B, according to an embodiment of the present invention;

[0039] FIG. 27 depicts one example of a hit generator, according to a specific embodiment of the present invention; and

[0040] FIG. 28 depicts another example of a hit generator, according to another embodiment of the present invention.

Like reference numerals refer to corresponding parts throughout the several views of the drawings.

DETAILED DESCRIPTION OF EXEMPLARY EMBODIMENTS

[0041] The present invention provides a system, an apparatus and a method for effectively predicting accesses to memory for retrieving program instructions and program data that a processor might be expected to need. By effectively predicting accesses to memory, the latency of providing necessary data to one or more processors can be minimized. According to a specific embodiment of the present invention, an apparatus includes a speculator configured to predict memory accesses. An exemplary speculator can be configured to generate a configurable amount of predictions to vary the prediction generation rate. In another embodiment, a speculator can suppress the generation of certain predictions to limit quantities of unnecessary predictions, such as redundant predictions, that a prefetcher otherwise might be required to manage. In a specific embodiment, a speculator can also filter unnecessary predictions by probing whether a cache memory or an inventory containing predictions include a more suitable prediction for presentation to a processor. In one embodiment, a cache memory store predictions in a short term cache and a long term cache, both of which are examined in parallel to filter out redundant predictions.

Exemplary Embodiments for a Prefetcher and Speculator for Generating Sequential and Nonsequential Predictions

[0042] FIG. 1 is a block diagram illustrating an exemplary speculator, according to a specific embodiment of the present invention. In this example, speculator 108 is shown to reside within a prefetcher 106. Further, prefetcher 106 is shown to reside in a memory processor 104, which is designed to at least control memory accesses by one or more processors. Prefetcher 106 operates to "fetch" both program instructions and program data

from a memory 112 before being required, and then provide the fetched program instructions and program data to a processor 102 upon request by that processor. By fetching them prior to use (i.e., "prefetching"), processor idle time (e.g., the time during which processor 102 is starved of data) is minimized. Prefetcher 106 also includes a cache memory 110 for storing and managing the presentation of prefetched data to processor 102. Cache memory 110 serves as a data store for speeding-up instruction execution and data retrieval. Notably, cache memory 110 resides in prefetcher 106 and operates to supplement other memory caches, such as "L1" and "L2" caches, which are generally employed to decrease some latency apart from memory controller 104.

[0043] In operation, speculator 108 monitors system bus 103 for requests ("read requests") by processor 102 to access memory 112. In particular, as processor 102 executes program instructions, speculator 108 detects read requests for addresses that contain program instructions and program data yet to be used by processor 102. For purposes of discussion, an "address" is associated with a cache line or unit of memory that is generally transferred between memory 112 and cache memory 110. An "address" of a cache line can refer to a memory location, and the cache line can contain data from more than one address of memory 112. The term "data" refers to a unit of information that can be prefetched, whereas the terms "program instructions" and "program data" respectively refer to instructions and data used by processor 102 in its processing. So, data (e.g., any number of bits) can represent predictive information constituting program instructions and/or program data. Also, the term "prediction" can be used interchangeably with the term "predicted address." When a predicted address is used to access memory 112, one or more cache lines containing that predicted address, as well as other addresses (predicted or otherwise), is typically fetched.

[0044] Based on detected read requests, speculator 108 can generate a configurable number of predicted addresses that might likely be requested next by processor 102. Speculator 108 does so by using one or more speculation techniques in accordance with at least one embodiment of the present invention. Speculator 108 implements these speculation techniques as predictors, the implementations of which are described below. Moreover, speculator 108 suppresses the generation of some predictions and filters other predictions. By either suppressing or filtering certain predictions, or by doing both, the number of redundant predictions is decreased, thereby preserving resources. Examples of preserved resources include memory resources, such as cache memory 110, and bus resources (e.g., in terms of bandwidth), such as memory bus 111.

[0045] After the predictions of speculator 108 undergo optional filtering, memory processor 104 transports surviving predictions (i.e., not filtered out) via memory bus 111 to memory 112. In response, memory 112 returns the prefetched data with the predicted addresses. Cache memory 110 temporarily stores the returned data until such time that memory processor 104 sends that data to processor 102. At an appropriate point in time, memory processor 104 transports prefetched data via system bus 103 to processor 102 to ensure latency is minimized, among other things.

[0046] FIG. 2 depicts an exemplary speculator in accordance with one embodiment of the present invention. Speculator 108 is configured to receive read requests 201 from which it generates predictions 203. As shown, speculator 108 includes a prediction controller 202 configured to provide control information and address information to sequential predictor ("SEQ. Predictor") 206 and to nonsequential predictor ("NONSEQ. Predictor") 216, both of which generate predictions 203. Prediction controller 202 serves, in whole or in part, to govern the prediction generation process in a manner that provides an optimal amount and type of predictions. For example, prediction controller 202 can vary the number and the types of predictions generated for a particular cache line, or group of cache lines, specified in read request 201. As another example, prediction controller 202 includes a suppressor 204 to suppress the generation of certain predictions so as to preserve resources, such as available memory in target cache 218, or to minimize unnecessary accesses to memory 112 due to redundantly predicted addresses. Prediction controller 202 can optionally include expediter 205 to hasten the generation of nonsequential predictions. Expediter 208 operates, as described in FIG. 8, to trigger the generation of a nonsequential prediction prior to the detection of an address that immediately precedes the nonlinear address stream to which the nonsequential prediction relates. A more detailed discussion of prediction controller 202 is subsequent to the following descriptions of sequential predictor 206 and nonsequential predictor 216.

[0047] Sequential predictor 206 is configured to generate predictions (i.e., predicted addresses) having a degree of expectancy. That is, sequential predictor 206 generates predictions that might be expected to follow one or more patterns of regular read requests 201 over time. These patterns arise from the fact that memory references have spatial locality among themselves. For example, as processor 102 executes program instructions, a stream of read requests 201 can be sequential in nature as they traverse system bus 103. To predict addresses following a sequential pattern, a type of speculation technique described below as

"forward sequential prediction" can predict sequential addresses. This type of speculation technique is described next.

[0048] Forward sequential predictor 208 is configured to generate a number of sequential addresses, ascending in order. So, if processor 102 transmits a series of read requests 201 onto system bus 103 that include a stream of ascending addresses, then forward sequential predictor 208 will generate a number of predictions for prefetching additional ascending addresses. An example of forward sequential predictor ("FSP") 208 is depicted in FIG. 3A. As is shown in FIG. 3A, FSP 208 receives addresses, such as address A0, and generates one or more addresses in a forward (i.e., ascending) sequence from the A0 address. The notation of A0 identifies a base address (i.e., A+0) from which one or more predictions are formed. So, the notations A1, A2, A3, etc. represent addresses of A+1, A+2, A+3, etc., whereas the notations A(-1), A(-2), A(-3), etc. represent addresses of A-1, A-2, A-3, etc. Although these notations represent a series of addresses either ascending or descending by one address, any patternable set of addresses can be referred to as sequential. As used throughout, sequential addresses can be represented by and referred to as single letter. For example, "A" represents A0, A1, A2, A3, etc., and "B" represents B0, B1, B2, B3, etc. As such, "A" and "B" each represent sequential address streams, but address streams of "B" are nonsequential to those of "A."

[0049] Further to FIG. 3A, FSP 208 is shown to receive at least an enable signal and a batch signal, both of which are provided by prediction controller 202. The enable signal controls whether forward sequential predictions are to be generated, and if so, the batch signal controls the number of sequential addresses that FSP 208 generates. In this example, the batch signal indicates that "seven" addresses beyond the base address are to be predicted. And as such, FSP 208 generates forward-sequenced addresses A1 to A7. So, when speculator 108 receives an address as part of a read request 201, such as A0, sequential predictor 206 can provide addresses A1, A2, A3, . . . , Ab, as a portion of predictions 203, where b is the number "batch."

[0050] Blind back sequential predictor 210 of FIG. 2 is configured to generate one sequential address, but descending in order from the base address. An example of blind back sequential predictor ("blind back") 210 is depicted in FIG. 3B, which shows blind back sequential predictor 210 receiving one or more addresses, such as address A0, and generating only one prediction, such as address A(-1), in a backward (i.e., descending) sequence from

the A0 address. As is the case with FSP 208, blind back sequential predictor 210 also receives an enable signal to control whether it generates a backward prediction.

[0051] Back sector sequential predictor 214 of FIG. 2 is configured to generate a specific cache line as a prediction after it detects another specific cache line from system bus 103. In particular, if back sector sequential predictor 214 detects that a certain read request 201 is for a high-order cache line, then an associated low-order cache line is generated as a prediction. A high-order cache line can be referred to as an upper ("front") sector that includes an odd address, whereas a low-order cache line can be referred to as a lower ("back") sector that includes an even address. To illustrate, consider that a cache line contains 128 bytes and is composed of a high-order cache line of 64 bytes (i.e., upper half of 128 bytes) and a low-order cache line of 64 bytes (i.e., lower half of 128 bytes).

[0052] An example of back sector sequential predictor 214 is depicted in FIG. 3C, which shows back sector sequential predictor ("back sector") 214 receiving one or more addresses. Upon receiving read request 201 for an upper or front sector of a cache line, such as address AU, back sector sequential predictor 214 generates only one prediction: address AL. This type of speculation technique leverages the phenomenon that processor 102 typically requests a lower or back sector sometime after requesting the upper or front sector of the cache line. Also, back sector sequential predictor 214 receives an enable signal to control whether it generates a back sector prediction.

[0053] Reverse sequential predictor 212 of FIG. 2 is configured to generate a number of sequential addresses, descending in order. So if processor 102 transmits a series of read requests onto system bus 103 that include a stream of descending addresses, then reverse sequential predictor 212 will generate a number of predictions for additional descending addresses. An example of reverse sequential predictor ("RSP") 212 is depicted in FIG. 3D. As is shown in FIG. 3D, RSP 212 detects a stream of addresses, such as addresses A0, A(-1), and A(-2), and in response, generates one or more addresses in a reverse (i.e., descending) sequence from base address A0. FIG. 3D also shows that RSP 212 receives at least an enable signal, a batch signal, and a confidence level ("Conf.") signal, all of which are provided by prediction controller 202. Although the enable signal and batch signal operate in a similar manner as used with FSP 208, the confidence level ("Conf.") signal controls a threshold that defines when to trigger the generation of reversely-sequenced predictions.

[0054] FIG. 3D further shows a chart 310 that depicts the behavior of an exemplary RSP 212, in accordance with a specific embodiment of the present invention. Here, a

confidence level of "two" sets trigger level 312 and a batch signal indicates that "five" addresses beyond the trigger address to be predicted. A trigger address is an address that causes a predictor to generate predictions. Consider that after detecting A(0) during interval I1, RSP 212 also detects address A(-1) during a following interval, I2. Next, upon detecting address A(-2) during interval I3, a certain level of confidence is reached that the detected stream is a series of descending addresses. This level of confidence is reached when trigger level 312 is surpassed, which causes RSP 212 to generate reversely-sequenced addresses A(-3) to A(-7). So, when speculator 108 receives a certain number of addresses, such as A0, A(-1) and A(-2), as a series of a read requests 201, then sequential predictor 206 can provide addresses A(-3), A(-4), A(-5), . . . , Ab, as a portion of predictions 203, where b is the number "batch." Note that in some embodiments, RSP 212 does not employ a confidence level, but rather generates predictions beginning after the base address. In other embodiments of the present invention, the concept of a confidence level is employed in other predictors described herein. The control of RSP 212 and other constituent predictors of sequential predictor 206 are discussed further below; nonsequential predictor 216 of FIG. 2 is described next.

[0055] Nonsequential predictor 216 is configured to generate one or more predictions (i.e., predicted addresses) subsequent to an address detected by speculator 108, even when the address is within a nonlinear stream of read requests 201. Typically, when there is no observable pattern of requested addresses upon which to predict a next address, predictions based on the preceding address alone is difficult. But in accordance with an embodiment of the present invention, nonsequential predictor 216 generates nonsequential predictions, which include predicted addresses that are unpatternable from one or more preceding addresses. An "unpatternable" prediction is a prediction that cannot be patterned with or is irregular to a preceding address. One type of unpatternable prediction is the nonsequential prediction. A preceding address upon which a nonsequential prediction is based can be either an immediate address or any address configured as a trigger address. Notably, a lack of one or more patterns over two or more addresses in a stream of read requests 201 is indicative of processor 102 executing program instructions in a somewhat scattershot fashion in terms of fetching instructions and data from various spatial locations of memory locations.

[0056] Nonsequential predictor 216 includes a target cache 218 as a repository for storing associations from a preceding address to one or more possible nonsequential addresses that can qualify as a nonsequential prediction. Target cache 218 is designed to readily compare its contents against incoming, detected addresses for generating

nonsequential predictions in a timely fashion. A detected address from which to generate a nonsequential prediction is referred to as a "trigger" address and the resulting prediction is a "target" of the unpatternable association between the two. An exemplary nonsequential predictor 216 is described next.

[0057] FIG. 4 illustrates an exemplary nonsequential predictor 216, according to one embodiment of the present invention. Nonsequential predictor 216 includes a nonsequential prediction engine ("NonSeq. Prediction Engine") 420 operably coupled to a repository, which is target cache 422. Target cache 422 maintains associations between each trigger address and one or more corresponding target addresses. Note that FIG. 4 shows one of many ways with which to associate nonsequential addresses. Here, a tree structure relates a specific trigger address to its corresponding target addresses. In this example, target cache 422 includes address "A" as a trigger address from which to form associations to addresses of possible nonsequential predictions, such as addresses "B," "X," and "L." These three target addresses are also trigger addresses for respective addresses "C" and "G," "Y," and "M." The formation and operation of target cache 422 is discussed in more detail below. Note that address "A" can also be a target address for a trigger address that is not shown in FIG. 4. Moreover, many other associations are also possible among addresses that are not shown.

[0058] Nonsequential prediction engine 420 is configured to receive at least four signals and any number of addresses 402. To control operation of nonsequential prediction engine 420, prediction controller 202 provides a "batch" signal and an "enable" signal, both of which are similar in nature to those previously described. Prediction controller 202 also provides two other signals: a width ("W") signal and a depth ("D") signal. These signals control the formation of target cache 422; the width signal, W, sets the number of possible targets from which a trigger address can predict, and the depth signal, D, sets the number of levels associated with a trigger address. An example of the latter is when D indicates a depth of "four." This means that address A is at a first level, address B is at a second level, addresses C and G are at a third level, and address D is at fourth level. An example of the former is when W is set to "two." This means only two of the three addresses "B," "X," and "L" are used for nonsequential prediction.

[0059] FIG. 4 also shows nonsequential prediction engine 420 configured to receive exemplary addresses 402 from prediction controller 202, such as addresses conceptually depicted in nonsequential address streams 404, 406, 408, 410 and 412, each of which includes an address that is unpatternable to a previously detected address. For example,

stream 404 includes address "A" followed by address "B," which in turn is followed by address "C." As is the case with nonsequential addresses, detecting a pattern to predict "B" from "A," and to predict "C" from "B" is a difficult proposition without more than just monitoring read requests 201 from processor 102. To this end, nonsequential predictor 216 forms target cache 422 to enable the prediction of unpatternable associations between a specific trigger address and its target addresses. Once nonsequential prediction engine 420 forms a nonsequential prediction, it generates a group of predictions from the associated target address. So if trigger address "A" leads to a nonsequential prediction of address "B" (i.e., B0 as base address), then the predicted addresses would include B0, B1, B2, . . . Bb, where b is a number set by the batch signal.

[0060] In one embodiment of the present invention, nonsequential prediction engine 420 forms target cache 422 as it stores an association from each of addresses 402 to a subsequent address. For example, upon detecting address A of stream 404, nonsequential prediction engine 420 populates target cache 422 with associations, such as an association from A to B, an association from B to C, an association from C to D, etc. Nonsequential prediction engine 420 does the same when it detects addresses of other streams 406, 408, etc.

[0061] In accordance with a specific embodiment, target cache 422 stores these associations in tabular form, such as tables 430, 440 and 450. These tables include a trigger column 426 and a target column 428 for respectively storing associations between a trigger address and a target address. Next consider that addresses 402 of all the streams are stored in tables 430, 440, and 450 of target cache 422. As shown in table 430, trigger-target associations 432, 434, and 436 describe associations from A to B, from B to C, and from G to Q, respectively. Other trigger-target associations 438 include associations from C to D, and so on. Likewise, table 440 includes trigger-target association 442 to describe an association from A to X, and table 450 includes trigger-target association 452 to describe an association from A to L.

[0062] FIG. 4 shows that tables 430, 440, and 450 are respectively identified as "Way 0," "Way 1," and "Way 2," which describes the relative priority of multiple trigger-target associations for the same trigger address. In this case, Way 0 is associated with the highest priority, Way 1 with the second highest, and so on. In this example, trigger-target association 432 of table 430 indicates that the association from A to B is a higher priority than the association from A to X, which is trigger-target association 442 of table 440. So after target cache 422 includes these associations, the next time nonsequential prediction engine 420

detects address A (so long as prediction controller 202 enables nonsequential prediction engine 420 to operate), then address B will be predicted as highest priority, followed by address X as second-highest priority, etc. due to the relative priorities of the tables.

[0063] In accordance with one embodiment of the present invention, the relative priorities are determined in at least two ways. First, a trigger-target association is assigned with the highest priority when it is first detected and placed into target cache 422. Second, a trigger-target association is assigned with the highest priority when nonsequential prediction engine 420 determines that that trigger-target association is successful (e.g., there has been a most-recent cache hit resulting from the nonsequential prediction based on that particular association). A "most-recent" cache hit is a recent cache hit of at least one of the target addresses associated to a specific trigger address. Further, the previous "highest priority" (also designated as leg 0) is shuffled to the second highest priority (also designated as leg 1) by moving the corresponding association to the way 1 table. As an example, consider that at a first point in time that the association from A to X is introduced into target cache 422 as the first trigger-target association. As a result, it will be assigned the highest priority (i.e., initially at leg 0) by being placed into table 430 (i.e., way 0). At some later point in time, target cache 422 inserts the association from A to B into table 430 (highest priority, leg 0). Also, the association from A to X is moved to table 440 (second highest priority, leg 1). In a specific embodiment of the present invention, the table to which a trigger-target association is stored depends on a portion of the address bits that constitute an index.

[0064] Referring back to FIG. 2, prediction controller 202 is configured to control both sequential predictor 206 and nonsequential predictor 216. Prediction controller 202 controls the amount as well as the types of predictions generated by either sequential predictor 206 or nonsequential predictor 216, or both. Also, prediction controller 202 suppresses the generation of predictions 203 that otherwise are unnecessary, such as redundant or duplicative predictions. As each of the predictors 208, 210, 212, 214 and 216 can be operational at the same time, the number of predictions 203 should be managed so as to not overload prefetcher resources. Prediction controller 202 employs suppressor 204 to perform this and other similar operations.

[0065] In one embodiment of the present invention, suppressor 204 controls the amount of predictions generated. It does so by first ascertaining certain attributes of read request 201. In particular, suppressor 204 determines whether read request 201 pertains to either program instructions (i.e., "code") or program data (i.e., "not code"). Typically, read

requests 201 for retrieving code rather than program data tend to be more likely sequential in a nature, or at least patternable. This is because processor 102 generally executes instructions in a more linear fashion than its requests for program data. As such, suppressor 204 can instruct sequential predictor 206 or nonsequential predictor 216 to suppress prediction generation when read requests 201 relate to program data. This helps prevent generating spurious predictions.

[0066] Suppressor 204 can also adjust the amount of predictions that sequential predictor 206 and nonsequential predictor 216 generate by ascertaining whether read request 201 is a non-prefetch "demand" or a prefetch. Processor 102 typically will demand (as a non-prefetch demand) a program instruction or program data be retrieved from memory 112 in some cases where it is absolutely necessary, whereas processor 102 may only request to prefetch a program instruction or program data to anticipate a later need. Since an absolute need can be more important to service than an anticipated need, suppressor 204 can instruct specific predictors to suppress predictions based on prefetch read requests 201 in favor of predictions based on demand read requests 201.

[0067] Table I illustrates an exemplary technique for suppressing the number of predictions generated. That is, when read request 201 pertains to both code and to a demand, suppressor 204 will be least suppressive. That is, prediction controller 202 will set "batch" at a large size, which is denoted as Batch Size (4) in Table I. In a particular example, Batch Size (4) can be set to seven. But if for the reasons given above, when read request 201 relates to both program data (i.e., not code) and to a processor-generated prefetch, suppressor 204 will be most suppressive. As such, prediction controller 202 will set "batch" at a small size, which is denoted as Batch Size (1) in Table I. As an example, Batch Size (1) can be set to one. In other cases, prediction controller 202 can vary the level of prediction suppression by using other batch sizes, such as Batch Size (2) and Batch Size (3). Although a suppressor in accordance with one embodiment of the present invention is configured to suppress the generation of at least one predicted address by decreasing the "batch" quantity if a processor request is for data or a prefetch request, or both, Table I is not limiting. For example, a processor request for code or instructions could decrease the "batch" size rather increasing it. As another example, requests for a demand could also decrease the "batch" size rather increasing it. One having ordinary skill in the art should appreciate that the many variations are within the scope of the present invention.

Code or Data	Demand or Prefetch	Batch
Not Code (i.e., Data)	Prefetch	Batch Size (1)
Not Code (i.e., Data)	Demand	Batch Size (2)
Code	Prefetch	Batch Size (3)
Code	Demand	Batch Size (4)

Read Request Types
Table I.

[0068] Suppressor 204 can also adjust the type of predictions that sequential predictor 206 and nonsequential predictor 216 generate. First, consider that prediction controller 202 can simultaneously enable both forward sequential predictor 208 and reverse sequential predictor 212. As such, suppressor 204 instructs prediction controller 202 to disable at least forward sequential predictor 208 when reverse sequential predictor 212 triggers (i.e., the confidence level is surpassed) so as to minimize predicting addresses in an ascending order when processor 102 is requesting read addresses in a descending order.

[0069] Second, consider that a certain address triggers back prediction (i.e., either blind back sequential predictor 210 or back sector sequential predictor 214) when prediction controller 202 enables sequential prediction (i.e., either forward sequential predictor 208 or reverse sequential predictor 212) to operate. In this case, suppressor 204 suppresses the batch by one from its initial amount for either forward sequential predictor 208 or reverse sequential predictor 212. That is, if "batch" was initially set to seven, then "batch" will be decreased by one upon triggering or activation of either blind back sequential predictor 210 or back sector sequential predictor 214. For example, if the batch for forward sequential predictor 208 is set to generate address A0, A1, A2, . . . , A7, and if blind back sequential predictor 210 is enabled for one or more read requests 201, then forward sequential predictor 208 generates only predictions A1, A2, . . . , A6. The final result is a set of predictions A(-1), A(0), A1, A2, . . . , A6 for those read requests 201, where back prediction provides prediction A(-1).

[0070] Third, prediction controller 202 can optionally disable either blind back sequential predictor 210 or back sector sequential predictor 214 to suppress their predictions

after the first generated prediction in a sequential stream of addresses 201 from the processor. This is because after a base address of a sequence has been established, subsequent forward or reverse sequential predictions also predict backward-type speculations (albeit one address behind). For example, forward sequential predictions A2, A3, and A4 also cover backward-type predictions A1, A2, and A3, all of which have already been predicted (if the base address is A0). Suppressor 204 can be configured to suppress other types of predictions, examples of which follow.

[0071] FIG. 5 illustrates an exemplary technique of suppressing nonsequential predictions, according to one embodiment of the present invention. According to this technique, suppressor 204 detects interleaved sequential streams that otherwise could be considered nonsequential, which requires storage of trigger-target associations in target cache 422. To preserve resources, especially useable memory in target cache 422, suppressor 204 parses nonsequential addresses, such as in stream 502, and models those nonsequential addresses as interleaved sequential streams. As is shown, stream 502 is composed of addresses A0, B0, C0, A1, B1, C1, A2, B2, and C2, each detected during respective intervals I1, I2, I3, I4, I5, I6, I8, and I9. Suppressor 204 includes a data structure, such as table 504, to model the nonsequential addresses as sequential. Table 504 can contain any number of stream trackers for deconstructing stream 502. In particular, stream trackers 520, 522, and 524 are designed to model sequential streams B0, B1, and B2, A0, A1, and A2, and C0 and C1, respectively. Later-detected read addresses from stream 502, such as A7 (not shown), are compared against these streams to see whether nonsequential predictions still can be suppressed for those streams being tracked.

[0072] In operation, suppressor 204 tracks sequential streams by storing a base address 510, such as the first address of a sequence. Thereafter, suppressor 204 maintains a last-detected address 514. For each new last-detected address (e.g., B2 of stream tracker 520), the previous last-detected address (e.g., B1 of stream tracker 520) is voided ("void') by being placed in column 512, which is an optional column. With this exemplary technique, suppressor 204 suppresses the generation of unnecessary nonsequential predictions when other types of predictions can be used. So for the example shown in FIG. 5, forward sequential predictor 208 can adequately generate predictions for stream 502.

[0073] FIG. 6 illustrates another exemplary technique of suppressing nonsequential predictions, according to one embodiment of the present invention. According to this technique, suppressor 204 models nonsequential addresses as interleaved sequential streams

similar to the process described in FIG. 5. But the technique of FIG. 6 implements multiple data structures each used to detect sequential streams over any number of threads. In this example, tables 604, 606, and 608 include stream trackers for thread (0) ("T"), thread (1) ("T′") and thread (2) ("T″"), respectively. So with this technique, nonsequential addresses of stream 602 can be modeled as multiple sequential streams over multiple threads so as to suppress nonsequential predictions. Note that this technique can apply to reverse sequential streams or other types of predictions.

[0074] FIG. 7 illustrates another technique for suppressing nonsequential predictions, according to a specific embodiment of the present invention. For a stream of addresses 702, a nonsequentiality exists between address A4 and B0. But in some cases, if the difference in time between these requested read addresses is too short, then there will not be enough time to employ nonsequential prediction. Matcher 706 of suppressor 204 operates to compare the difference in time, d, between address A4 and B0. If d is equal to or greater than a threshold, TH, then matcher 706 signals to enable (i.e., "not suppress") nonsequential predictor 216 to operate. But if d is less than TH, then matcher 706 signals to disable nonsequential predictor 216, thereby suppressing predictions.

[0075] Another suppression mechanism that can be implemented by suppressor 204 is as follows. Generally there is a finite amount of time that elapses before a request for a back sector address is made by processor 102, after requesting a front sector address. If the amount of time is long enough, then the back sector address read request may appear to be an irregularity (i.e., unpatternable to the front sector). To prevent this, suppressor 204 is configured to maintain a list of front sector reads by processor 102. Subsequent to detecting the front sector address, addresses are compared against that front sector address. When the corresponding back sector arrives, then it will be so recognized. Therefore, an otherwise nonsequentiality as well as its predictions can be suppressed.

[0076] FIG. 8 depicts an exemplary technique for expediting generation of predictions, according to a specific embodiment of the present invention. Specifically, expediter 205 (FIG. 2) operates to in accordance with this technique to hasten the generation of nonsequential predictions. In this example, stream 802 includes two abutting sequential streams A0 to A4 and B0 to B3. Nonsequential predictor 216 typically designates address A4 as trigger address 808 with address B0 as target address 810. But to decrease the time to generate nonsequential predictions, trigger address 808 can be changed to new trigger address 804 (i.e., A0). So with the designation of a new trigger address for the target address, the

next time processor 102 requests addresses of stream 802, nonsequential predictor 216 can immediately generate its predictions upon detecting an earlier address rather than a later address (i.e., generate predictions when A0 is detected as the "new" trigger address rather than A4). This ensures that the nonsequential predictions are generated at the most opportune time.

[0077] FIG. 9 shows another exemplary speculator, according to one embodiment of the present invention. In this example, prefetcher 900 includes a speculator 908 with a filter 914 for filtering redundant addresses so as to keep unnecessary prediction generation to a minimum. Prefetcher 900 of FIG. 9 also includes a multi-level cache 920 and a prediction inventory 916. Here, multi-level cache 920 is composed of a first level data return cache ("DRC1") 922 and a second level data return cache ("DRC2") 924. First level data return cache 922 can generally be described as a short-term data store and second level data return cache 924 can generally be described as a long-term data store. Multi-level cache 920 stores prefetched program instructions and program data from memory 112 until processor 102 requires them. Similarly, prediction inventory 916 provides temporary storage for generated predictions until selected by arbiter 918 to access memory 112. Arbiter 918 is configured to determine, in accordance with arbitration rules, which of the generated predictions are to be issued for accessing memory 112 to prefetch instructions and data.

[0078] Filter 914 includes at least two filters: cache filter 910 and inventory filter 912. Cache filter 910 is configured to compare newly-generated predictions to those previous predictions that prefetched instructions and data already stored in multi-level cache 920. So if one or more of the newly-generated predictions are redundant to any previously-generated prediction with respect to multi-level cache 920, then those redundant predictions are voided so as to minimize the number of predictions. Further, inventory filter 912 is configured to compare the newly-generated predictions against those already generated and stored in prediction inventory 916. Thus, if one or more of the newly-generated predictions are redundant to those previously stored in prediction inventory 916, then any redundant prediction can be voided so as to minimize the number of predictions, thereby freeing up prefetcher resources.

Exemplary Embodiments for a Nonsequential Predictor

[0079] FIG. 10 is a block diagram illustrating an exemplary nonsequential ("NONSEQ") predictor 1010, according to a specific embodiment of the present invention. In this example, nonsequential predictor 1010 is shown to reside within a speculator 1008, which also includes a sequential predictor 1012 for generating sequential predictions. Prefetcher 1006, which includes speculator 1008, operates to "fetch" both program instructions and program data from a memory before being required (not shown), and then provide the fetched program instructions and program data to a processor upon request by that processor (not shown). By fetching them prior to use (i.e., "prefetching"), processor idle time (e.g., the time during which the processor is starved of data) is minimized. Nonsequential predictor 1010 includes a nonsequential prediction engine ("Prediction Engine") 1020 for generating predictions and a target cache 1030 for storing and prioritizing predictions.

[0080] Prefetcher 1006 also includes a filter 1014, an optional prediction inventory 1016, an optional arbiter 1018, and a multi-level cache 1040. Here, filter 1014 includes a cache filter (not shown) configured to compare newly-generated predictions to those previous predictions that caused program instructions and program data to be already prefetched into multi-level cache 1040. So if any of the newly-generated predictions is redundant to any previously-generated prediction that is stored in multi-level cache 1040, then that redundant prediction is voided so as to minimize the number of predictions, thereby freeing up prefetcher resources. Prediction inventory 1016 provides a temporary storage for storing generated predictions until selected by arbiter 1018 to access a memory. Arbiter 1018 is configured to determine which of the generated predictions are to be issued for accessing the memory to prefetch instructions and data.

[0081] Multi-level cache 1040 is composed of a first level data return cache ("DRC1") 1042 and a second level data return cache ("DRC2") 1044. First level data return cache 1042 can generally be described as a short-term data store and second level data return cache 1044 can generally be described as a long-term data store. According to an embodiment of the present invention, either first level data return cache 1042 or second level data return cache 1044, or both, can store prefetched program instructions and program data prefetched based on a predicted address (i.e., a target address). As shown, the prefetched predictive information stored in multi-level cache 1040 is represented as data(TRT1) and data(TRT2). This notation means that target addresses TRT1 and TRT2 have contributed to prefetching data representing predictive information. As shown, and as will be discussed

below, data(TRT1) and data(TRT2) are stored in multi-level cache 1040 with prediction identifiers ("PIDs") 1 and 2, respectively. When either data(TRT1) or data(TRT2) is requested by a processor, the corresponding target address (e.g., TRT1) and prediction identifier will be communicated to nonsequential predictor 1010.

[0082] In operation, speculator 1008 monitors a system bus as a processor requests access ("read requests") to a memory. As the processor executes program instructions, speculator 1008 detects read requests for addresses that contain program instructions and program data yet to be used by the processor. For purposes of discussion, an "address" is associated with a cache line or unit of memory that is generally transferred between a memory and a cache memory, such as multi-level cache 1040. Note that a cache memory is an example of a repository external to target cache 1030.

[0083] Based on detected read requests, nonsequential predictor 1010 can generate a configurable number of predicted addresses that might likely be requested next by the processor. In particular, nonsequential predictor 1010 is configured to generate one or more predictions (i.e., predicted addresses) subsequent to its detections of an address, even when that address is within a nonlinear stream of read requests. Typically, when there is no observable pattern of requested addresses upon which to predict a next address, predictions based on the preceding address alone is difficult. But in accordance with an embodiment of the present invention, nonsequential prediction engine 1020 generates nonsequential predictions, which include predicted address that are unpatternable from one or more preceding addresses. An "unpatternable" prediction is a prediction that cannot be patterned with or is irregular to a preceding address. One type of unpatternable prediction is the nonsequential prediction. A preceding address upon which a nonsequential prediction is based can be either an immediate address or any address configured as a trigger address. Notably, a lack of one or more patterns over two or more addresses in a stream of read requests is indicative of a processor executing program instructions in a somewhat scattershot fashion in terms of fetching instructions and data from various spatial locations of memory locations.

[0084] Nonsequential predictor 1010 includes a target cache 1030 as a repository for storing an association for a preceding address to one or more potential nonsequential addresses that can each qualify as a nonsequential prediction. Target cache 1030 is designed to compare its contents against incoming detected addresses for generating nonsequential predictions in an expeditious manner. Moreover, target cache 1030 is configured to prioritize

those nonsequential predictions in response to, for example, a hit in a cache memory. Or, nonsequential predictor 1010 can prioritize the first instance of establishing an association between a new nonsequential prediction and a particular trigger address. A "trigger" address is a detected address from which nonsequential predictor 1010 generates a nonsequential prediction with the resulting prediction referred to as a "target" of the unpatternable association between the two. Note that in accordance with at least one embodiment of the present invention, target cache 1030 can be a single-ported memory to conserve resources that otherwise would be used by multi-ported memories.

[0085] After prefetcher 1006 issues the predictions from nonsequential predictor 1010, the nonsequential predictions are used to access the memory. In response, the memory returns prefetched data with references relating to the predicted addresses, where the references can include a prediction identifier ("PID") and a corresponding target address. Thereafter, multi-level cache memory 1040 temporarily stores the returned data until such time that the processor requests it. As is described below, when the processor does request the prefetched data (i.e., predictive information) a reference is sent to nonsequential predictor 1010 for readjusting a priority of a nonsequential prediction, if necessary.

[0086] FIG. 11 illustrates an exemplary nonsequential predictor 1010, according to one embodiment of the present invention. Nonsequential predictor 1010 includes a nonsequential prediction engine ("NonSeq. Prediction Engine") 1120 operably coupled to a repository, as exemplified by target cache 1130. Further, nonsequential prediction engine 1120 includes a prediction generator 1122 and a priority adjuster 1124. Prediction generator 1122 generates predictions and manages trigger-target associations stored in target cache 1130. Priority adjuster 1324 operates to prioritize the trigger-target associations, for example, from the most recent, successful target addresses to the least recent or successful. Prediction generator 1122 and a priority adjuster 1124 are described more thoroughly in FIGs. 12 and 13, respectively.

[0087] Target cache 1130 maintains associations between each trigger address ("TGR") and one or more corresponding target addresses ("TRTs"). Note that FIG. 11 shows one of many ways with which to associate nonsequential addresses. Here, a tree structure relates a specific trigger address to its corresponding target addresses. In this example, target cache 1130 includes address "A" as a trigger address from which to form associations to addresses of possible nonsequential predictions, such as addresses "B," "X," and "L." These three target addresses are also trigger addresses for respective addresses "C" and "G," "Y,"

and "M." The formation and operation of target cache 1130 is discussed in more detail below, especially when prediction generator 1122 discovers a new trigger-target association and inserts that association in target cache 1130. Note that address "A" can also be a target address for a trigger address that is not shown in FIG. 11. Moreover, many other associations are also possible among addresses that are not shown.

[0088] As is shown, target cache can be constructed, for example, by nonsequential prediction engine 1120 in accordance with at least three variables: width ("w"), depth ("d"), and height ("h"), according to one embodiment of the present invention. Width, w, sets the number of possible targets from which a trigger address can predict, and depth, d, sets the number of levels associated with a trigger address. Height, h, sets the number of successive trigger addresses that are used to generate nonsequential predictions. As an example, consider that d indicates a depth of "four." This means that address A is at a first level, address B is at a second level, addresses C and G are at a third level, and address D is at fourth level. As another example, consider that w is set to "two." This means only two of the three addresses "B," "X," and "L" are used for nonsequential prediction as leg 0 and leg 1, all three addresses being of the second level. In a specific embodiment, variable h sets the number of levels beyond just the first level to effectuate multi-level prediction generation.

[0089] Consider that h is set to 2 as is shown in FIG. 11. This means that there are two levels of trigger addresses—a trigger address in the first level (e.g., address A) and a successive trigger address in the second level (e.g., address B). So with h set to 2, a first grouping of predictions is formed in response to trigger address A. That is, any of those target addresses of the second level can generate one or more groups of nonsequential addresses. For instance, any of addresses "B," "X," and "L" can be a basis for generating nonsequential predictions, where the number of these addresses are selected by the number of active legs (e.g., leg 0 through leg 2) defined by nonsequential prediction engine 1120. But in accordance with multi-level prediction generation (and with h set to 2), addresses "B," "X," and "L" could each be a successive trigger address for generating a second grouping of predictions based on target address of the next level down. Accordingly, target addresses C and G of the third level can be used to generate additional nonsequential predictions based on successive trigger address B. Similarly, target addresses Y and M can also be used to generate nonsequential predictions based on successive trigger addresses X and L, respectively. One having ordinary skill in the art should appreciate that there are many implementations possible by varying one or more of the three aforementioned variables.

[0090] Nonsequential prediction engine 1120 is configured to receive exemplary addresses 1101 of read requests. FIG. 11 conceptually depicts nonsequential address streams 1102, 1104, 1106, 1108 and 1110, each of which includes an address that is unpatternable to a previously detected address. For example, stream 1102 includes address "A" followed by address "B," which in turn is followed by address "C." As is the case with nonsequential addresses, detecting a pattern to predict "B" from "A," and to predict "C" from "B" is a difficult proposition without more than just monitoring read requests 1101. To this end, prediction generator 1122 establishes the contents of target cache 1130 to enable the prediction of unpatternable associations between a specific trigger address and its target addresses. For example, upon detecting address A (as well as subsequent addresses) of stream 1102, prediction generator 1122 populates target cache 1130 with associations, such as an association from A to B, an association from B to C, an association from C to D, etc. Nonsequential prediction engine 1120 does the same when it detects addresses of other streams 1104, 1106, etc.

[0091] In accordance with a specific embodiment, target cache 1130 stores these associations in tabular form, such as tables 1140, 1150 and 1160. These tables include a trigger column ("TGR") and a target column ("TGT") for respectively storing a trigger address and a target address. Next consider that addresses 1101 of all the streams are stored in tables 1140, 1150 and 1160. As shown in table 1140, trigger-target associations 1142, 1144, and 1146 describe associations from A to B, from B to C, and from G to Q, respectively. Other trigger-target associations 1148 include associations from C to D, and so on. Likewise, table 1150 includes trigger-target association 1152 to describe an association from A to X, and table 1160 includes trigger-target association 1162 to describe an association from A to L.

[0092] FIG. 11 shows that tables 1140, 1150 and 1160 are respectively identified as "way 0," "way 1," and "way 2," which describes the relative positions of multiple trigger-target associations in target cache 1130 for the same trigger address. Priority adjuster 1124 assigns priorities to trigger-target associations, and thus predictions, typically by associating memory locations with priority. In this case, way 0 is associated with the highest priority, way 1 with the second highest, and so on. In this example, trigger-target association 1142 of table 1140 indicates that the association from A to B is a higher priority than the association from A to X, which is trigger-target association 1152 of table 1150. So after target cache 1130 includes these associations, the next time nonsequential prediction engine 1120 detects

address A, then nonsequential prediction engine 1120 can provide one or more predictions. Typically, nonsequential prediction engine 1120 generates nonsequential prediction generated in order of priority. Specifically, nonsequential prediction engine 1120 generates predictions having the highest priority before generating predictions of lower priority. As such, nonsequential prediction engine 1120 can generate a configurable number of the predictions based on priority. For example, nonsequential prediction engine 1120 can limit the number of predictions to two: leg 0 and leg 1 (i.e., top two trigger-target associations). This means that in some cases, nonsequential prediction engine 1120 will be more inclined to provide address B rather than address X due to the relative priorities of the tables. Note that relative priorities among trigger-target associations are just that—relative. This means that target cache 1130 can position a highest priority association for a specific trigger address, for example, at way 4 and position the second highest priority association at way 9. But note that target cache 1130 can include any arbitrary quantity of "legs," beyond that just of leg 0 and leg 1, from one address.

[0093] FIG. 12 illustrates an exemplary prediction generator 1222, according to an embodiment of the present invention. In this example, prediction generator 1222 is coupled to a target cache 1230 to generate predictions as well as to manage trigger-target associations stored therein. Prediction generator 1222 includes an index generator 1204, a tag generator 1206, a target determinator 1208 and a combiner 1210. Also, prediction generator 1222 includes an inserter 1202 for inserting discovered trigger-target associations into target cache 1230.

[0094] When generating predictions, index generator 1204 and tag generator 1206 respectively operate to create an index and a tag for representing a first address "addr_1," which can be an address that precedes other addresses. Index generator 1204 forms an index, "index(addr_1)," from addr_1 to access a subset of memory locations in target cache 1230. Typically, the value of index(addr_1) selects each corresponding memory location of each selected way. Further, tag generator 1206 forms a tag "tag(addr_1)" so that prediction generator 1222 can access specific trigger-target associations in target cache 1230 that are associated with addr_1.

[0095] As an example, consider that addr_1 is "G." With this address, prediction generator 1222 generates index(G) to select memory locations associated with that index. In this instance, index(G) has a value "I," which is three (i.e., I=3). This means that index(G) can be used to select each memory location identified by I=3 for way ("way 0") 1240, way

("way 1") 1250, through to way ("way N") 1260, where N is a configurable number representing the number of ways available in target cache 1230. For the same address G, tag generator 1206 will create a tag of address G as tag(G) to identify specific memory locations associated with G. So, given an index of index(G) and a tag of tag(G), target addresses Q and P (or alternate representations thereof) can be retrieved from or stored at respective memory locations in way 1240 and way 1250, as is shown in FIG. 12. In a specific embodiment, each address consists of 36 bits. Bits 28:18 can represent a tag for an address and any group of bits 19:9, 18:8, 17:7 or bits 16:6 can represent a configurable index for that address. In one embodiment, a portion of an address alternately represents a target address. For example, bits 30:6 of a 36-bit target address are maintained in TRT columns of target cache 1230. With reduced representations of both target and trigger address, less hardware is required, thereby reducing costs in terms of material, resources, etc.

[0096] Target determinator 1208 determines whether a trigger-target association exists for a particular trigger, and if so, then it determines each target address for that trigger. Continuing with the previous example, target determinator 1208 retrieves target addresses Q and P in response to tag(G) being matched against the tags at index(G) that represent other trigger addresses. An ordinarily skilled artisan should appreciate that well known comparator circuits (not shown) are suitable for implementation in either prediction generator 1222 or target cache 1230 to identify matching tags. When one or more target addresses have been found, those addresses are passed to combiner 1210. Combiner 1210 associates each target address 1214 with a prediction identifier ("PID") 1212, which is composed of an index and tag of the trigger address. PID 1212 identifies the trigger address that caused target addresses Q and P to be predicted. So, if PID 1212 can be represented as [index(G),tag(G)], then a nonsequential prediction generated by prediction generator 1222 will have a form of [[index(G),tag(G)],Q] as a reference. Note that Q, as a prediction, is considered a "referenced prediction" when [index(G),tag(G)] is associated thereto. The predictive information prefetched into a cache memory, therefore, can be represented as data(Q)+[[index(G),tag(G)],Q].

[0097] Combiner 1210 can be configured to receive a "batch" signal 1226 for generating a number of additional predictions that are nonsequential to the trigger address. For example, consider that batch signal 1226 instructs combiner 1210 to generate "n" predictions as a group of predictions having a range that includes the matched target address. So if trigger address "G" generates a nonsequential prediction of address "Q" (i.e., Q0 as base

address), then the predicted addresses can include Q0, Q1, Q2, . . . Qb, where b is a number set by the batch signal. Note that in some cases where a back sector or a blind back sequential prediction is generated concurrently, then batch, b, can be set to b-1. As such, the group of predicted addresses would include Q(-1), Q0, Q1, Q2, . . . Q(b-1). Note that each in the group of predicted addresses can also be associated with PID 1212. In a specific embodiment, target address 1214 inherits attributes of the trigger address, where such attributes indicate whether the trigger address is associated with code or program data, and whether the trigger address is a processor demand address or not. In another specific embodiment, fewer than the number of predicted addresses in a group can also be associated with PID 1212. In one example, only target address Q0 is associated with PID 1212, while one or more of the others of the group (e.g., Q(-1), Q2, Q3, etc.) need not be associated with PID 1212. As such, when trigger address G is encountered, followed by target address Q0, then PID 1212 gets reported to the non-sequential predictor. Later, when Q2 or any others of the group are encountered, PID 1212 does not get reported. This reduces the number of redundant entries in the target cache. Accordingly, only the association "G -> Q0" is stored and re-prioritized as a result of a hit on that prediction. When address Q1 is detected in the address stream, the non-sequential predictor will need not have to insert association "G -> Q1."

[0098] Next, consider that target determinator 1208 does not detect a target address for addr_1. Target determinator 1208 then communicates to inserter 1202 that no trigger-target association exists for addr_1. In response, inserter 1202 forms a trigger-target association for addr_1 and inserts that association into target cache 1230. To do so, inserter 1202 first identifies a memory location using index(addr_1) with which to store tag(addr_1). Inserter 1202 is also configured to receive a subsequent address, "addr_2," to store as a target address to trigger address addr_1. As no trigger-target association existed prior to the newly-formed trigger-target association, inserter 1202 stores tag(addr_1) and addr_2 respectively in the TRG column and TGT column of way 1240, which is the highest priority way (i.e., way 0). For example, consider that for address stream 1104 of FIG. 11, this stream shows the first instance where "Z" follows "Y." After determining that no "tag(Y) to Z" trigger-target association exists, inserter 1202 of FIG. 12 then stores the new trigger-target association at index(Y). As such, "tag(Y) to Z" is stored as trigger-target association 1242 in way 1240. In a specific embodiment, inserter 1202 receives an insertion signal ("INS") 1224 from priority adjuster 1324, which is described next.

[0099] FIG. 13 illustrates an exemplary priority adjuster 1324, according to an embodiment of the present invention. Generally, priority adjuster 1324 operates to prioritize the trigger-target associations from the most recent, successful target addresses to the least recent or successful. For example, a trigger-target association will be assigned a highest priority (i.e., stored in way 0) when no previous target existed for a particular. Further, a trigger-target association can be assigned a highest priority when the predicted target address is proved successful (e.g., there has been a read of data by a processor, where the data was prefetched based on a nonsequential prediction). In this example, priority adjuster 1324 is coupled to target cache 1230 to, among other things, prioritize trigger-target associations stored therein. Priority adjuster 1324 includes a register 1302, an index decoder 1308, a tag decoder 1310, a target determinator 1318, a matcher 1314 and a reprioritizer 1316.

[0100] Generally, priority adjuster 1324 receives information external to nonsequential predictor 1010 indicating that a particular address was successful in providing data requested by a processor. Such information can be generated by a cache memory, such as multi-level cache 1040 described in FIG. 10. Priority adjuster 1324 receives this information into register 1302 as "Hit Info." Hit Info is a reference that includes at least the address 1304 of the data (e.g., program instructions and/or program data actually requested by a processor). Address 1304 is labeled as addr_2. The reference also includes PID 1306 associated with address 1304.

[0101] Index decoder 1308 and tag decoder 1310 respectively extract index(addr_1) and tag(addr_1) from PID 1306 to determine whether addr_2 has the appropriate level of priority. To do so, priority adjuster 1324 identifies whether addr_2 is a target address of an existing trigger-target association in target cache 1230. After priority adjuster 1324 applies tag(addr_1) and index(addr_1) to target cache 1230, any matching trigger addresses in TRG columns of target cache 1230 will be received by target determinator 1318. Upon detecting one or more target addresses associated to addr_1, target determinator 1318 provides those target addresses to matcher 1314.

[0102] But if target determinator 1318 determines that no target address exists in a trigger-target association (i.e., there is not any addr_2 associated with address addr_1), then it will communicate an insert signal ("INS") 1224 to inserter 1202 of FIG. 12 to insert a new trigger-target association. Insert signal 1224 typically includes address information, such as addr_1 and addr_2. Typically in situations where no matching target address exists for PID 1306 of the Hit Info means that a processor has hit upon a previously issued nonsequential

prediction. But target cache 1230 has since purged the trigger-target association that formed the basis for that previously issued nonsequential prediction. As such, nonsequential prediction engine 1010 will insert, or reinsert, a trigger-target association that can again be used to predict the nonsequential address that was successfully used by a processor.

[0103] When target determinator 1318 does detect one or more target addresses, it provides the detected target addresses to matcher 1314. Matcher 1314 compares each detected target address against addr_2 (i.e., address 1304) to determine how many associated target addresses exist for addr_1, and for each existing target address, the way in which a corresponding trigger-target association resides. Matcher 1314 provides the results of its comparisons to reprioritizer 1316 to modify priorities, if necessary.

[0104] First, consider an instance where one or more target addresses have been detected as being associated with PID 1306 (i.e., addr_1), which represents addr_1 as the trigger address, but there is no trigger-target association that includes addr_2. So, reprioritizer 1316 will insert a new trigger-target association into a position representing a highest priority (e.g., way 0) and will demote the priorities of existing trigger-target associations of the same trigger. For example, consider that as shown in FIG. 12 a "tag(A)-to-X" trigger-target association is at a memory location representing a highest priority, whereas a "tag(A)-to-L" association has a lower priority. Next, assume that PID 1306 represents address A as addr_1 and addr_2 is address B. Reprioritizer 1316 will operate to store, as shown in FIG. 13, a "tag(A)-to-B" association in way 0, with the other previous associations stored in other ways, which are of lower priority.

[0105] Second, consider an instance where two target addresses have been detected as being associated with PID 1306 (i.e., addr_1), but the two trigger-target associations have their priorities improperly swapped. In this case, reprioritizer 1316 will insert the highest priority trigger-target association into a position representing a highest priority (e.g., way 0) and will insert the previous highest priority trigger-target association into another position representing a second highest priority (e.g., way 1). For example, consider that as shown in FIG. 12 a "tag(B)-to-G" trigger-target association is at a memory location representing a highest priority whereas a "tag(B)-to-C" association has a lower priority. Next, assume that PID 1306 represents address B as addr_1 and address C is addr_2. Reprioritizer 1316 will operate to store, as shown in FIG. 13, a "tag(B)-to-C" association in way 0, with the other association in way 1, which is of lower priority. Note this technique of prioritization is useful

if at least the two top-most priorities are kept as "leg 0" and "leg 1," as the highest and second highest priories, respectively.

[0106] Next, consider an instance where two target addresses have been detected as being associated with PID 1306 (i.e., addr_1), and the two trigger-target associations have their priorities properly assigned. In this case, reprioritizer 1316 takes no action as the corresponding trigger-target associations are correct.

[0107] FIG. 14 depicts an exemplary pipeline 1400 for operating a predictor generator to form nonsequential predictions, according to a specific embodiment of the present invention. In FIG. 14, solid-lined boxes represent storage during or between stages and broken-lined boxes represent actions performed by a nonsequential predictor. During Stage 0, addr_1 of a read request is decoded by combined-tag-and-index generator 1402, which can be an amalgam of index decoder 1308 and tag decoder 1310 of FIG. 13. In one embodiment, combined-tag-and-index generator 1402 is a multiplexer configured to separate addr_1 into a first part of the address and second part of the address. The first part is held as tag(addr_1) at 1406 and the second part is held as index(addr_1) at 1408. Also during this stage, index(addr1) is applied to a target cache at 1410 to retrieve data describing trigger-target associations. Optionally, addr_1 of a read request can be temporarily stored in buffer 1404 while a target cache is being written.

[0108] During Stage 1, tag(addr_1) and index(addr_1) remain held respectively at 1412 and at 1414. At 1416, target addresses are read from the target cache. During Stage 2, a nonsequential prediction engine selects suitable nonsequential predictions by first matching tag(addr_1) against the tags associated with index(addr_1) at 1418. At 1420, a nonsequential prediction engine configures multiplexers, for example, to transfer the highest priority target address (i.e., from a way storing the highest priority trigger-target association) into a leg 0 prediction queue at 1422 and to transfer the second highest priority target address (i.e., from a way storing the second highest priority trigger-target association) into a leg 1 prediction queue at 1424. During Stage 3, these two nonsequential predictions are output at 1430 to a combiner, for example. Note that although FIG. 14 generates nonsequential predictions in four stages, other nonsequential prediction pipelines of other embodiments can have more or fewer stages.

[0109] FIG. 15 depicts an exemplary pipeline 1500 for operating a priority adjuster to prioritize nonsequential predictions, according to a specific embodiment of the present invention. Solid-lined boxes represent storage during or between stages and broken-lined

boxes represent actions that can be performed by a priority adjuster. Pipeline 1500 depicts an exemplary method of inserting trigger-target associations into a target cache and reprioritizing target cache associations. Stage –1 determines whether the priority adjuster will insert or prioritize. If the priority adjuster is going to perform an insertion, then address addr_1 of a read request at 1502 is stored at 1506 during this stage. This address has the potential to be a trigger address for a target address. If the priority adjuster is going to perform a prioritization, then at 1504 the priority adjuster receives a PID 1508 representing addr_1 address from an external source (e.g., a cache memory), and also receives address addr_2 at 1510 during this stage.

[0110] Note that FIGs. 14 and 15 exemplify nonsequential prediction using one level of prediction. To effectuate multi-level prediction generation, exemplary pipelines 1400 and 1500 can be modified to feed the generated predictions at the end of respective pipelines 1400 and 1500 back into pipelines 1400 and 1500 as input addresses. These predictions then are queued up for another level of prediction generation. For example, if A is detected, then target cache 1130 produces target addresses B and X (e.g., as two highest priority ways). Then, address B as a successive trigger address is input back into the top of the pipeline, whereby target cache 1130 produces address C and G. In short, a feedback loop can be added to exemplary pipelines 1400 and 1500 to implement more than one level of prediction.

[0111] First consider that during Stage 0, the priority adjuster is performing a trigger-target association insertion. In this instance, addr_1 is decoded by combined-tag-and-index generator 1514, and addr_2 is selected from 1512 through multiplexer 1516. Combined-tag-and-index generator 1514 performs a collective function of an index generator and a tag generator. In one embodiment, combined-tag-and-index generator 1514 is a multiplexer configured to select an address from either 1506 or 1508. In this case, combined-tag-and-index generator 1514 forms a first address part, which is held as tag(addr_1) at 1520, and forms a second part, which is held as index(addr_1) at 1522. Also during this stage, index(addr1) is applied via multiplexer 1518 to a target cache at 1524 to retrieve data describing trigger-target associations. Next, consider that during Stage 0, the priority adjuster is performing a prioritization of a target cache. In this instance, addr_1 (or its alternative representation) is received from 1508 and addr_2 is selected from 1510 through multiplexer 1516. Combined tag and index generator 1514 then forms a first and second parts from PID 1508. Index(addr_1) formed from PID 1508 is then applied via multiplexer 1518 to a target cache at 1524 to retrieve data describing trigger-target associations. From Stage 1 to Stage 3,

pipeline 1500 behaves similarly regardless of whether the priority adjuster is performing an insertion or a prioritization.

[0112] During Stage 1, tag(addr_1) and index(addr_1) remain held respectively at 1530 and at 1532. At 1534, target addresses are read from the target cache. During Stage 2, a priority adjuster first matches tag(addr_1) against the tags. If at 1540 no tags match, then multiplexers are configured at 1542 to prepare for inserting a trigger-target association. But if at least one tag from the ways of the target cache matches at 1544, and if the highest priority trigger-target association does not reside in a way corresponding to the highest priority, then trigger-target associations are reprioritized at 1554. To do this, multiplexers are selected at 1552 to reprioritize or insert a new trigger-target association. During Stage 3, fully-connected reprioritizing multiplexers are configured to store addr_2 from 1556. This address will be written as a target address at way 0 during stage 0, as determined by index(addr_1) held at 1550. As is shown, other trigger-target associations as determined by fully-connected reprioritizing multiplexers at 1560, are also written as cache write data into the target cache at 1524 using index(addr_1) held at 1550. After pipeline 1500 returns to Stage 0, the priority adjuster continues to operate accordingly.

Exemplary Embodiments for Issuing Predictions from an Inventory

[0113] FIG. 16 is a block diagram illustrating an exemplary prediction inventory 1620, according to a specific embodiment of the present invention. In this example, prediction inventory 1620 is shown to reside within a prefetcher 1606. Further, prefetcher 1606 is shown to operate within a memory processor 1604, which is designed to at least control memory accesses by one or more processors. Prefetcher 1606 operates to "fetch" both program instructions and program data from a memory 1612 before being required, and then provide the fetched program instructions and program data to a processor 1602 upon request by that processor. By fetching them prior to use (i.e., "prefetching"), processor idle time (e.g., the time during which processor 1602 is starved of data) is minimized. Prefetcher 1606 also includes a speculator 1608 for generating predictions and a filter 1622 for removing unnecessary predictions.

[0114] Filter 1622 is representative of either an inventory filter or a post-inventory filter, or both. By removing unnecessary predictions, prefetcher 1606 can preserve computational and memory resources that otherwise would be used to manage the duplicative

predictions needlessly. An inventory filter (as a pre-inventory filter) operates to remove unnecessary predictions prior to insertion to prediction inventory 1620, whereas a post-inventory filter removes unnecessary predictions prior to issuance to memory 1612. An example of a post-inventory filter is described in FIG. 20. Next, the operation of prefetcher 1606 and its components are discussed next.

[0115] In operation, speculator 1608 monitors system bus 1603 for requests ("read requests") by processor 1602 to access memory 1612. As processor 1602 executes program instructions, speculator 1608 detects read requests for addresses that contain program instructions and program data yet to be used by processor 1602. For purposes of discussion, an "address" is associated with a cache line or unit of memory that is generally transferred between memory 1612 and a cache memory (not shown). A cache memory is an example of a repository of predictions external to the prediction inventory. An "address" of a cache line can refer to a memory location, and the cache line can contain data from more than one address of memory 1612. The term "data" refers to a unit of information that can be prefetched, whereas the terms "program instructions" and "program data" respectively refer to instructions and data used by processor 1602 in its processing. So, data (e.g., any number of bits) can represent predictive information constituting program instructions and/or program data.

[0116] Based on detected read requests, speculator 1608 can generate numerous predictions to improve its chances of accurately predicting accesses to memory 1612 by processor 1602, those numerous predictions might include redundant predictions. Examples of such predictions include forward sequential predictions, reverse sequential predictions, back blind sequential predictions, back sector sequential predictions, nonsequential predictions, and the like. To remove such redundancies, inventory filter 1622 filters out duplicative predictions to generate surviving predictions, which are then stored in prediction inventory 1620. To remove redundancies, inventory filter 1622 compares generated predictions against the contents of a cache (not shown) prior to inserting those predictions into prediction inventory 1620. If a match is found between a prediction and one residing in prediction inventory 1620, then inventory filter 1622 voids that prediction. But if no match is found, then inventory filter 1622 inserts the surviving predictions into prediction inventory 1620. Note that it may be the case that some predictions within a new group of predictions (i.e., those generated by one event, or the same trigger address) match the contents of the cache, whereas other predictions do not. In this case, inventory filter 1622 voids the

individual predictions that match those in the cache and inserts those predictions that were not matched (e.g., not marked as "void") into prediction inventory 1620.

[0117] Once resident in prediction inventory 1620, predictions are maintained as "items" of inventory. The term "item" refers to either a "prediction" or a "triggering address" (which generates the prediction) as stored in prediction inventory 1620. These items can be compared against later-generated predictions for filtering purposes. Prefetcher 1606 manages these items in inventory while issuing them at varying rates to memory 1612. The rate of issuance depends on the type of predictions (e.g., forward sequential predictions, nonsequential predictions, etc.), the priority of each type of predictions, and other factors described below.

[0118] One way a prediction can become redundant is if the processor 1602 issues an actual read request for a particular address and a prediction for that address already exist in prediction inventory 1620. In this case, the prediction is filtered out (i.e., voided) and the actual read request of processor 1602 is maintained. This is particularly true for predictions such as sequential-type and back-type predictions. Also, some predictions become redundant between the time prediction inventory 1620 receives those predictions until prefetcher 1606 issues them to memory 1612, prefetcher 1606 can also filter out predictions prior to issuing an item. This again decreases the number of redundant predictions arising during the time a duplicate, but later-generated prediction is inserted in prediction inventory 1620. And as the number of redundant predictions decreases, the more resources are preserved.

[0119] After prefetcher 1606 issues the predictions from prediction inventory 1620, memory processor 1604 transports the remaining predictions (i.e., not filtered out by at least a post-inventory filter) via memory bus 1611 to memory 1612. In response, memory 1612 returns the prefetched data with references to the predicted addresses. A cache memory (not shown), which can reside within or without prefetcher 1606, temporarily stores the returned data until such time that memory processor 1604 sends that data to processor 1602. At an appropriate point in time, memory processor 1604 transports prefetched data via system bus 1603 to processor 1602 to ensure latency is minimized, among other things.

[0120] FIG. 17 depicts an exemplary prediction inventory 1620 in accordance with one embodiment of the present invention. Prediction inventory 1620 contains a number of queues 1710, 1712, 1714 and 1716 for storing predictions, where a queue can be a buffer or any like component for storing predictions until each is issued or filtered out. Prediction inventory 1620 also includes an inventory manager 1704 and one or more queue attributes

1706, whereby inventory manager 1704 configures the structure and/or operation of each of the queues in accordance with corresponding queue attributes 1706.

[0121] An individual queue maintains predictions as items, all of which are generally of the same particular type of prediction, such as a forward sequential prediction. As shown, prediction inventory 1620 includes four queues, a sequential queue ("S Queue") 1710, a back queue ("B Queue") 1712, a nonsequential zero-queue ("NS0 Queue") 1714, and a nonsequential one-queue ("NS1 Queue") 1716. Sequential queue 1710 can be configured to contain either forward sequential predictions or reverse sequential predictions, whereas back queue 1712 can contain either blind back sequential predictions or back sector sequential predictions. For purposes of discussion, note that forward sequential predictions, reverse sequential predictions, and the like can collectively be referred to as "series-type" predictions, whereas blind back sequential predictions, back sector sequential predictions, and the like can collectively be referred to as "back-type" predictions.

[0122] Prediction inventory 1620 includes a "0th" nonsequential queue and a "1st" nonsequential queue. Nonsequential ("zero-") queue 1714 and nonsequential ("one-") queue 1716 contain nonsequential predictions having the "highest" and the "second highest" priority, respectively. In particular, nonsequential zero-queue 1714 maintains nonsequential predictions, which includes the highest priority target addresses (of any number of target addresses) that can be generated by corresponding trigger addresses. A "trigger" address is a detected address from which speculator 1608 generates predictions. Such a prediction (i.e., predicted address) is a "target" address, which is unpatternable (e.g., nonsequential) with the trigger that generates the target. Similarly, nonsequential one-queue 1716 maintains nonsequential predictions, but instead includes the second highest priority target addresses that can be generated by corresponding trigger addresses.

[0123] Each queue can be composed of any number of groups 1720, such as Groups 0, 1, 2 and 3. Each group 1720 includes a configurable number of items, such as a triggering address and corresponding predictions that the triggering address generates. For example, groups 1720 of sequential queue 1710 each can include a triggering address and seven sequential predictions, whereas groups 1720 of back queue 1712 each can include a triggering address and one back-type prediction (or in some cases, these queues only contain predictions as items). Further, groups 1720 of either nonsequential zero-queue 1714 or nonsequential one-queue 1716, or both, can contain a trigger address and a group of four nonsequential predictions (or in some cases, they only contain predictions as items). In a

specific embodiment, speculator 1608 determines the number of items per group 1720 stored in prediction inventory 1620 by setting its "batch" number to generate a specific number of predictions. By storing predictions in prediction inventory 1620 as grouped items, groups 1720 reduce the amount of information that is typically used to manage each prediction individually, which in turn facilitates arbitration when issuing predictions.

[0124] Inventory manager 1704 is configured to manage the inventory of items in each queue, as well as control the structure and/or operation of the queues. To manage prediction inventory 1620, inventory manager 1704 does so, in whole or in part, using one or more queue attributes 1706. A first example of a queue attribute is a type of queue. For example, any of queues 1710 to 1716 can be configured to be a first-in first-out ("FIFO") buffer, a last-in first-out ("LIFO") buffer, or any other type of buffer. The types of queue, such as a FIFO or a LIFO, influence the manner of inserting and removing items in relation to a queue. In one embodiment, sequential queue 1710 is configured as a LIFO, and nonsequential zero-queue 1714 and nonsequential one-queue 1716 each are configured as FIFOs.

[0125] A second example of a queue attribute is an expiration time, or a lifetime, which is assignable to a queue, a group, or an item. This attribute controls the degree of staleness for predictions. As predictions in any group 1720 or queue ages, or becomes stale, then they will increasingly be less likely to reflect accurate predictions. So to minimize aged items, inventory manager 1704 enables a group to maintain its current inventory until a certain expiration time after which inventory manager 1704 purges either the entire aged group or any remaining items yet to be issued. In one embodiment of the present invention, a lifetime for a queue, a group, or an item can be configured so as to retain them indefinitely. That is, they can be set as "immortal," which means that immortal predictions will reside in a queue until issued or the immortality is withdrawn. In a specific embodiment, an expiration time is associated with a group when it is inserted into a queue. Thereafter, a timer counts down from the expiration time such that when it reaches zero, any remaining item of that group is invalidated. In yet another embodiment, an expiration time for groups 1720 of either nonsequential zero-queue 1714 or nonsequential one-queue 1716 is set longer than groups 1720 of sequential queue 1710 to increase the likelihood that a nonsequential prediction will be issued and consequently hit in the data cache.

[0126] A third example of a queue attribute is an insertion indicator associated with a queue to indicate how inventory manager 1704 is to insert predictions into a queue when that

queue is full. In one instance, the insertion indicator indicates whether inventory manager 1704 is to either drop a newly-generated prediction from being inserted, or overwrite an old item residing in the particular queue. If the insertion indicator is "drop," then inventory manager 1704 will discard any new prediction that otherwise would be inserted. But if the insertion indicator is "overwrite," then inventory manager 1704 takes one of two courses of action, depending on the type of queue to which the particular queue corresponds. If the queue is configured as a LIFO, then inventory manager 1704 will push the new prediction into the LIFO as a stack, which effectively pushes out the oldest item and/or group from bottom of the LIFO. But if the queue is configured as a FIFO, then the new prediction overwrites the oldest item in the FIFO.

[0127] A fourth example of a queue attribute is a priority associated with each of the queues to determine the particular queue from which the next item is to be issued. In one embodiment, a priority order is set relative to each of queues 1710, 1712, 1714 and 1716 for arbitrating among the queues to select the next prediction. In applications where series-type predictions are more abundantly generated, servicing sequential queue 1710 is important. So, this queue is typically associated with a relatively high priority. This means, for instance, that nonsequential zero-queue ("NS0 Queue") 1714 and a nonsequential one-queue ("NS1 Queue") 1716 are most likely set to a lower priority relative to sequential queue 1710. Another example of a queue attribute is a queue size associated with each of the queues to determine how many predictions can be store temporarily therein. For example, a sequential queue can have size or depth of two groups, a back queue can have a depth of one group, and the nonsequential queues can have a depth of four groups. Note that queue size can control the number of predictions that are issued by prefetcher 1606 by controlling how much inventory memory is assigned to the different types of predictions.

[0128] The priority of back queue 1712 can be dynamically promoted or modified to be higher than that of sequential queue 1710, according to one embodiment of the present invention. This feature is at in retrieving predictive information from memory 1612 after speculator 1608 detects an upper or "front" sector. This is because processor 1602 is likely to request a lower or "back" sector shortly after requesting the upper or front sector of the cache line. So by elevating the priority of back queue 1712, especially when it is maintaining back sector sequential predictions, there is an increased likelihood that prefetcher 1606 will issue the appropriate back sector sequential prediction to memory 1612. In a specific embodiment, a back queue counter (not shown) counts the number of items issued from queues other than

back queue 1712. When this counter reaches a threshold, back queue 1712 is promoted to a priority at least higher than sequential queue 1710. Then, an item (e.g., a back sector item) can be issued from back queue 1712. After it either issues at least one back-type item or back queue 1712 becomes empty (e.g., by aging or by issuing all items), the priority of back queue 1712 returns (or reverts back) to its initial priority and the back queue counter resets.

[0129] Generally for any group 1720 of nonsequential group of predictions, there can be a mix of series-type and back-type predictions as target addresses for nonsequential predictions. In particular, the group of nonsequential addresses can include just series-type (i.e., either forward or reverse) predictions. But those groups can also include a number of series-type predictions mixed with a back-type. As an example of the former, consider that speculator 1608 determines that a trigger address "A" is associated with a target address "B" and another target address "C." If target address B is of a higher priority than C, then B is maintained in nonsequential zero-queue 1714 along with a group of predictions nonsequential to trigger address A. Then, the group can include predictions B0 (i.e., address B), B1, B2 and B3, all of which are nonsequential to address A but are all forward series-type. As an example of the latter, group 1720 can include nonsequential predictions B(-1) (i.e., address B-1), B0, B1, and B2, where prediction B(-1) is a back-type prediction mixed with other series-type predictions. Or, group 1720 can include any other arrangement of predictions not specifically described herein. Since C has a second higher priority than B, C is maintained in nonsequential one-queue 1716 with a similar group of nonsequential predictions. Consequently, predictions B0, B1, B2 and B3 can be inserted as group 3 of nonsequential zero-queue 1714, and predictions C0, C1, C2 and C3 can be inserted as group 3 of nonsequential one-queue 1716.

[0130] FIG. 17 also shows that in one embodiment prediction inventory 1620 is configured to receive predictions 1701 via inventory filter 1702 through which surviving predictions pass. The surviving predictions are then inserted into the appropriate queue and managed by inventory manager 1704 as described above. An exemplary inventory filter 1702 is described next.

[0131] FIG. 18 illustrates an example of inventory filter 1702 in accordance with a specific embodiment of the present invention. Although this example applies to filtering forward sequential predictions against a sequential queue, such as sequential queue 1710 of FIG. 17, inventory filter 1702 can be used in cooperation with any queue to filter any type of prediction. Namely, inventory filter 1702 can be configured to compare any number of

predictions of any prediction type against at least one other queue that contains predictions of a different prediction type. For example, a number of forward sequential predictions can be filtered against a back queue, or the like. Inventory filter 1702 includes at least a matcher 1804 to match items in group 1806 and a number of predictions 1802. Group 1806 includes items A1 to A7, each of which are associated with item A0. A0 is the triggering address that previously generated the predictions identified as items A1 to A7. Also, group 1806 can reside as any group 1720 in sequential queue 1710. As for the number of predictions 1802, these include "TA" as the triggering address and predictions B1 to B7, all of which were generated by speculator 1608 upon detecting address TA. Note that although FIG. 18 shows only one group (i.e., group 1806), other groups 1720 of the same queue can be filtered in the same manner and at the same time.

[0132] In a specific embodiment, matcher 1804 is composed of a number of comparators identified as CMP0, CMP1, CMP2, . . . CMPM (not shown). Comparator CMP0 is configured to compare TA against N items in group 1806 and comparators CMP1, CMP2, . . . CMPM each compare a prediction from predictions 1802 against a number of N items in group 1806, where M is set to accommodate the largest number of predictions generated. As an example, consider that M is seven, thereby requiring seven comparators, and N is three so that each comparator compares one element in 1802 to three items in 1806. Further, consider that each element of predictions 1802 is matched to a corresponding item having the same position (e.g., first to first, second to second, etc.). As such, CMP0 will compare TA against A0, item A1 and item A2, CMP1 will compare prediction B1 against items A1, A2, and A3, and so forth. Number N can be set so as to minimize the amount of comparator hardware, but to sufficiently filter out consecutive streams and those predictions that might result from small jumps (i.e., no larger than N) in the streams of addresses detected on system bus 1603.

[0133] In one embodiment, a queue stores a page address to represent A0 and offsets each representing item A1, item A2, etc. To determine whether a match exists in this case, the page address of address TA and an offset of a specific prediction from predictions 1802 is respectively compared against the page address of A0 and a corresponding offset. In a specific embodiment of the present invention, inventory filter 1702 does not filter sequential predictions against nonsequential predictions and therefore does not cooperate with either nonsequential zero-queue 1714 or nonsequential one-queue 1716. This is because

nonsequential speculations may be less likely to have as many redundancies that exist with sequential predictions.

[0134] FIGs. 19A and 19B are diagrams illustrating exemplary techniques of filtering out redundancies, according to a specific embodiment of the present invention. Once matcher 1804 determines a match, then either the newly-generated prediction (i.e., new item K) or the previously generated item (i.e., old item K) is invalidated. FIG. 19A shows which of either new item K or old item K is filtered out or invalidated. In this case, queue 1902 is a FIFO. As such, new item K will be invalidated, thereby keeping old item K. By contrast, FIG. 19B shows that when queue 1904 is a LIFO, old item K will be invalidated, thereby keeping new item K. Generally, the earliest issuing of either new item K or old item K will be kept, whereas the one that issues the latest will be invalidated. An ordinarily skill artisan should appreciate that inventory filter 1702 can employ other techniques without deviating from the scope and the spirit of the present invention.

[0135] FIG. 20 shows another exemplary prediction inventory disposed within a prefetcher, according to one embodiment of the present invention. In this example, prefetcher 2000 includes a speculator 1608 and filter 2014. Prefetcher 2000 of FIG. 20 also includes a multi-level cache 2020 and a prediction inventory 1620. Here, multi-level cache 2020 is composed of a first level data return cache ("DRC1") 2022 and a second level data return cache ("DRC2") 2024. First level data return cache 2022 can generally be described as a short-term data store and second level data return cache 2024 can generally be described as a long-term data store. Multi-level cache 2020 stores prefetched program instructions and program data from memory 1612 until processor 1602 requires them. The caches of multi-level cache 2020 also store references to the predictions that generated the prefetched predictive information so that newly-generated predictions can be filtered against multi-level cache 2020. For example, DRC1 2022 and DRC2 2024 store two types of information as references in addition to the data for a cache line or unit of memory: (1) the address for a stored cache line that is used to filter against new predictions, and (2) the trigger address in case the cache line was brought into the cache as a result of a prediction. In particular, the trigger address is used to shuffle priorities of the nonsequential predictions in speculator 1608.

[0136] Prediction inventory 1620 provides temporary storage for generated predictions until selected by arbiter 2018. The stored predictions in prediction inventory 1620 are used to filter out redundancies that otherwise would be issued. Arbiter 2018 is

configured to determine, in accordance with arbitration rules, which of the generated predictions are to be issued to prefetch instructions and data. Generally, such arbitration rules provide a basis from which to select a particular queue for issuing a prediction. For example, arbiter 2018 selects and issues predictions based, in part or in whole, on relative priorities among queues and/or groups.

[0137] Filter 2014 includes at least two filters: cache filter 2010 and inventory filter 1702. Cache filter 2010 is configured to compare newly-generated predictions to those previous predictions that caused prefetched instructions and data to already become stored in multi-level cache 2020. So if one or more of the newly-generated predictions are redundant to any previously-generated prediction with respect to multi-level cache 2020, then a redundant prediction is voided so as to minimize the number of predictions requiring processing. Note that a redundant prediction (i.e., the extra, unnecessary prediction) can be the newly-generated prediction. Inventory filter 1702 is configured to compare the newly-generated predictions against those already generated and stored in prediction inventory 1620. In one embodiment, inventory filter 1702 is similar in structure and/or functionality of that shown in FIG. 18. Again, if one or more of the newly-generated predictions are redundant to those previously stored in prediction inventory 1620, then any redundant prediction can be voided so as to free up prefetcher resources.

[0138] To reduce further the number of redundant predictions, a post-inventory filter 2016 is included within prefetcher 2000. After or just prior to prefetcher 1606 issuing predictions from prediction inventory 1620, post-inventory filter 2016 filters out redundant predictions that arose between the time prediction inventory 1620 first receives those predictions until the time arbiter 2018 selects a prediction to issue. These redundancies typically arise because a prediction representing the same predicted address of an item in prediction inventory may have been issued from prediction inventory 1620 to a memory, but may not yet have returned any predictive information to cache 2020 (i.e., no reference is within cache 2020 with which to filter against). In one embodiment, post-inventory filter 2016 can be similar in structure and/or functionality of either inventory filter 1702 shown in FIG. 18 or cache filter 2002.

[0139] In one embodiment, post-inventory filter 2016 maintains issuance information for each item of each group 1720 in prediction inventory 1620. In particular, this issuance information indicates which item of a particular group has issued. But post-inventory filter 2016 does not remove issued items from prediction inventory 1620. Rather, they remain so

that they can be compared against when filtering out incoming redundant predictions. As each item in that particular group issues, the issuance information is updated to reflect this. Once all items have been issued, then the group is purged and the queue is freed up to take additional items.

[0140] In one embodiment, arbiter 2018 can control some aspects of prediction inventory 1620 that relate to issuing predictions. In particular, arbiter 2018 can modify relative priorities among the queues, groups or items so as to issue the most advantageous predictions. In a specific embodiment, arbiter 2018 is configured to effectively modify relative priorities to throttle back the generation of a large number of predictions that overly burdens a memory (i.e., memory over-utilization), such as memory 1612, cache memory 2020, or other components of the memory subsystem. For example, arbiter 2018 can assign a configurable load threshold to each queue. This threshold indicates a maximum rate at which a particular queue can issue predictions. This load threshold is compared against the contents of a workload accumulator (not shown), which maintains the accumulated units of work requested from memory 1612. A unit of work is any action requested of memory 1612, such as reading, writing, etc. As additional units of work are requested of memory 1612, the value in the workload accumulator increases. But as time goes by (e.g., for every certain number of clock cycles), that value decreases. In operation, arbiter 2018 compares the load threshold of each queue to the value of the workload accumulator. If the load threshold is surpassed by the workload value, then arbiter 2018 performs one of two exemplary actions. Arbiter 2018 can instruct prediction inventory 1620 to stop taking predictions for that specific queue so that items therein will either be issued or aged out. Or, arbiter 2018 can take items out of the queue by overwriting them. Once arbiter 2018 detects that the workload value falls below that of the load threshold, the queue will again be available to issue predictions.

Exemplary Embodiments for Performing Look-Ahead Lookup on Predictive Information in a Cache Memory

[0141] FIG. 21 is a block diagram illustrating a prefetcher 2100 including an exemplary multi-level cache 2120, according to a specific embodiment of the present invention. In this example, multi-level cache 2120 includes a cache filter 2110, a first level data return cache ("DRC1") 2122 and a second level data return cache ("DRC2") 2124. Cache filter 2110 is configured to expeditiously examine, or perform a "look-ahead lookup" on both first level DRC 2122 and second level DRC 2124 to detect either the presence or the

absence of an input address, such as a predicted address, in those caches. A look-ahead lookup is an examination of cache memory to determine, in parallel, whether a number of predictions already exist in, for example, multi-level cache 2120.

[0142] Depending on whether a prediction is present or absent, multi-level cache 2120 manages the contents of both first level DRC 2122 and second level DRC 2124 in accordance with caching policies, examples of which are described below. First level DRC 2122 can be generally described as a short-term data store and second level DRC 2124 can be generally described as a long-term data store, whereby predictions in first level DRC 2122 eventually migrate to second level DRC 2124 when a processor does not request those predictions. According to an embodiment of the present invention, either first level DRC 2122 or second level DRC 2124, or both, can store prefetched program instructions and program data based on a predicted address, as well as a processor-requested address. Also, cache filter 2110, first level DRC 2122 and second level DRC 2124 cooperate to reduce latency of providing prefetched program instructions and program data by reducing redundant predictions as well as by speeding up prefetching of predictive information (e.g., by anticipating page opening operations), for example. Note that although the following discussion relates to a multi-level cache memory (i.e., multiple caches), any of the following exemplary embodiments can include a single cache memory.

[0143] Cache filter 2110 is configured to compare a range of input addresses against each of a number of multiple caches in parallel, where the multiple caches are hierarchical in nature. For example, a first cache can be smaller in size and adapted to store predictions for a relatively short period of time, whereas a second cache can be larger in size and adapter to store predictions for durations longer than that of the first cache. Further the second cache receives its predicted address and corresponding predicted data only from the first cache, according to one embodiment of the present invention. To examine both caches in parallel, especially where the second cache is larger than the first, cache filter generates two representations of each address "looked up," or examined in the caches. With one representation used for the first cache and the second representation used for the second cache, both caches can be examined simultaneously. One reason for this can be that there are more addresses and entries requiring examination in a larger cache than in a smaller cache. So if both are to be examined at one time, a more efficient technique for examining addresses of the larger cache is needed than that of the smaller cache. The query interfaces discussed below perform these functions.

[0144] Prefetcher 2100 also includes a speculator 2108 for generating predictions. Specifically, speculator 2108 includes a sequential predictor ("SEQ. Predictor") 2102 to generate sequential predictions, such as forward sequential predictions, reverse sequential predictions, back blind sequential predictions, back sector sequential predictions, and the like. Also, speculator 2108 includes a nonsequential predictor ("NONSEQ. Predictor") 2104 for forming nonsequential predictions. Prefetcher 2100 uses these predictions to "fetch" both program instructions and program data from a memory (not shown), and then store the fetched program instructions and program data in multi-level cache 2120 before a processor (not shown) requires the instructions or data. By fetching them prior to use (i.e., "prefetching"), processor idle time (e.g., the time during which the processor is starved of data) is minimized.

[0145] Nonsequential predictor 2104 includes a target cache (not shown) as a repository for storing an association for a preceding address to one or more potential nonsequential addresses that can each qualify as a nonsequential prediction. The target cache is designed to compare its contents against incoming detected addresses for generating nonsequential predictions in an expeditious manner, whereby the target cache is configured to prioritize its stored nonsequential predictions in response to, for example, a hit in multi-level cache 2120. Specifically, when multi-level cache 2120 provides a predicted address to a processor upon its request, then the stored trigger-target association of which that address belongs is elevated in priority. A "trigger" address is a detected address from which nonsequential predictor 2104 generates a nonsequential prediction, with the resulting prediction referred to as a "target" of an unpatternable association formed between the two. Note that a trigger address can also refer to an address that gives rise to a sequential prediction, which also can be referred to as a target address.

[0146] Prefetcher 2100 also includes a filter 2114, an optional prediction inventory 2116, an optional post-inventory filter 2117, and an optional arbiter 2118. Here, filter 2114 can be configured to include an inventory filter (not shown) for comparing generated predictions to previously-generated predictions that reside in prediction inventory 2116. Prediction inventory 2116 provides a temporary storage for storing generated predictions until arbiter 2118 selects a prediction to access a memory. Arbiter 2118 is configured to determine which prediction of the generated predictions is to be issued for accessing the memory when prefetching instructions and data. In some embodiments, filter 2114 can include cache filter 2110, which can be configured to compare generated predictions to those

previously-generated predictions that have caused program instructions and program data to be already "prefetched" into multi-level cache 2120. So if any of the generated predictions is redundant to any previously-generated prediction stored in multi-level cache 2120, then that redundant prediction can be voided (or invalidated) so as to minimize the number of predictions requiring governance, thereby freeing up prefetcher resources.

[0147] In operation, speculator 2108 monitors a system bus as a processor requests access to a memory ("read requests"). As the processor executes program instructions, speculator 2108 detects read requests for addresses that contain program instructions and program data yet to be used by the processor. For purposes of discussion, an "address" is associated with a cache line or unit of memory that is generally transferred between a memory and a cache memory, such as multi-level cache 2120. An "address" of a cache line can refer to a memory location, and the cache line can contain data from more than one address of the memory. The term "data" refers to a unit of information that can be prefetched, and the terms "program instructions" and "program data" respectively refer to instructions and data used by the processor in its processing. So, data (e.g., any number of bits) can represent "predictive information," which refers to information that constitutes either the program instructions or program data, or both. Also, the term "prediction" can be used interchangeably with the term "predicted address." When a predicted address is used to access the memory, one or more cache lines containing that predicted address, as well as other addresses (predicted or otherwise), is typically fetched.

[0148] When prefetcher 2100 issues predictions, it can append or associate a reference to each prediction. In the case where a prediction is a nonsequential prediction, the reference associated therewith can include a prediction identifier ("PID") and a corresponding target address. A PID (not shown) identifies the trigger address (or a representation thereof) that caused the corresponding target address to be predicted. This reference is received by multi-level cache 2120 when the memory returns prefetched data. Thereafter, multi-level cache 2120 temporarily stores the returned data until such time that the processor requests it. During the time that multi-level cache 2120 stores the prefetched data, it manages that data for filtering against generated predictions, for ensuring coherency of the data stored therein, for classifying its data as either short term or longer term data, and the like. But when the processor does request the prefetched data (i.e., predictive information), that data is sent to the processor. If data being placed in multi-level cache 2120 is the result of a nonsequential prediction, then a reference can be sent to nonsequential

predictor 2104 for readjusting a priority of nonsequential prediction stored in the target cache, if necessary.

[0149] FIG. 22 illustrates an exemplary multi-level cache 2220, according to one embodiment of the present invention. Multi-level cache 2220 includes a cache filter 2210, a first level data return cache ("DRC1") 2222 and a second level data return cache ("DRC2") 2224. Cache filter 2210 includes a DRC1 query interface 2204 and a DRC2 query interface 2214 for respectively interfacing first level DRC 2222 and second level DRC 2124 with components of prefetcher 2100 as well as other components, such as those of a memory processor (not shown). One such memory processor component is a write-back cache 2290 of FIG. 21, which operates in accordance with a well known caching method whereby modifications to data in a cache are not copied to the cache source (e.g., system memory) until necessary. As write-back cache 2290 is similar in structure and functionality to those well known in the art, it need not be discussed in detail. Further, DRC1 query interface 2204 contains a DRC1 matcher 2206 and DRC1 handler 2208, and DRC2 query interface 2214 contains a DRC2 matcher 2216 and DRC2 handler 2218.

[0150] First level DRC 2222 includes a DRC1 address store 2230 for storing addresses (e.g., predicted addresses), where DRC1 address store 2230 is coupled to a DRC1 data store 2232, which stores data (i.e., predictive information) and PIDs. For example, prefetched data resulting from predicted address ("PA") can be stored as data(PA) 2232a in association with PID 2232b. This notation denotes a predicted address PA having contributed to prefetching data that represents predictive information. When data(PA) 2232a is requested by a processor, the corresponding predicted address, PA, and prediction identifier, PID 2232b, will be communicated to nonsequential predictor 2104 to modify the priority of that predicted address, if necessary. Prediction identifier, PID 2232b generally contains information indicating the trigger address giving rise to the PA. Note that a PA generated by nonsequential predictor 2104 can also be referred as a target address, as a processor-requested address (and related data), can also be stored in multi-level cache 2220. Further note that data(PA) 2232a need not necessarily be accompanied by a PID 2232b.

[0151] Further, both DRC1 address store 2230 and DRC1 data store 2232 are communicatively coupled to a DRC1 manager 2234, which manages the functionality and/or structure thereof. Second level DRC 2224 includes a DRC2 address store 2240 coupled to a DRC2 data store 2242, which stores data in similar form to that of data 2232a and PID

2232b. Both DRC2 address store 2240 and DRC2 data store 2242 are communicatively coupled to a DRC2 manager 2246, which manages the functionality and/or structure thereof.

[0152] In a specific embodiment of the present invention, second level DRC 2224 also includes a repository of "valid bits" 2244 for maintaining valid bits 2244 separate from DRC2 address store 2240, each valid bit indicating whether a stored prediction is either valid (and available for servicing a processor request for data) or invalid (and not available). An entry having an invalid prediction can be viewed as empty entry. By keeping bits of valid bits 2224 separate from addresses, resetting or setting one or more valid bits is less computationally burdensome and quicker than if DRC2 address store 2240 stores the valid bits with the corresponding addresses. Note that in most cases, valid bits for addresses of DRC1 are typically stored with or as part of those addresses.

[0153] In operation, DRC1 query interface 2204 and DRC2 query interface 2214 are configured to respectively examine the contents of first level DRC 2222 and second level DRC 2224 to determine whether they include any of one or more addresses applied as "input addresses." An input address can originate from speculator 2108 as a generated prediction, from a write-back cache as a write address, or from another element external to multi-level cache 2220. Generally an input address, as described herein, is a generated prediction that is compared against the contents of multi-level cache 2220 to filter out redundancies. But sometimes the input address is a write address identifying a location of a memory to which data is or will be written. In this case, multi-level cache 2220 is examined to determine whether an action is required to maintain coherency among a memory, DRC1 data store 2222, and DRC2 data store 2224.

[0154] DRC1 matcher 2206 and DRC2 matcher 2216 are configured to determine whether one or more input addresses on input/output port ("I/O") 2250 are resident in DRC1 address store 2230 and DRC2 address store 2240, respectively. When either DRC1 matcher 2206 or DRC2 matcher 2216 detect that an input address matches one in first level DRC 2222 and second level DRC 2224, then an associated handler, such as DRC1 handler 2208 or DRC2 handler 2218, operates to either filter out redundant predictions or ensure data in multi-level cache 2220 is coherent with a memory. Note that DRC1 matcher 2206 and DRC2 matcher 2216 can be configured to compare a range of input addresses against the contents of first level DRC 2222 and second level DRC 2224 in parallel (i.e., simultaneously or nearly simultaneously, such is in one or two cycles of operation (e.g., clock cycles), or other minimal number of cycles, depending on the structure of multi-level cache 2220). An

example of a range of input addresses that can be compared in parallel against the caches is address A0 (the trigger address) and predicted addresses A1, A2, A3, A4, A5, A6, and A7, the latter seven possibly being generated by sequential predictor 2102.

[0155] When examined simultaneously, matchers 2206, 2216 that perform such a comparison is said to be performing "a look-ahead lookup." In some embodiments, a look-ahead look up is performed when a processor is idle, or when not requesting data from prefetcher 2100. Also note that although similar in functionality, the respective structures of DRC1 matcher 2206 and DRC2 matcher 2216 are adapted to operate with DRC1 address store 2230 and DRC2 address store 2240, respectively, and therefore are not necessarily similarly structured. Examples of DRC1 matcher 2206 and DRC matcher 2216 are discussed below in connection with FIGs. 23A and FIG. 24, respectively, according to at least one specific embodiment of the present invention.

[0156] Next, consider situations when query interfaces 2204, 2214 are performing filtering operations. By comparing a number of input addresses against the contents of multi-level cache 2220, and by detecting input addresses that do not match, handlers 2208, 2218 can take appropriate action to filter out matched predictions (i.e., redundant predictions) while permitting unmatched input address to proceed as generated predictions to fetch predictive information sooner than if filtering was not performed. As such, multi-level cache 2220 and its cache filter 2210 decrease the latency by more quickly determining which cache line to start fetching. This in turn makes it more likely to decrease latency experienced by the processor because first level DRC 2222 and second level DRC 2224 caches are generally more likely to contain prefetched predictive information sooner than if predictions either were not compared in parallel or were not filtered out, or both.

[0157] DRC1 address store 2230 and DRC2 address store 2240 each store addresses associated with prefetched data stored in DRC1 data store 2232 and DRC2 data store 2242, respectively. Each of address stores 2230 and 2240 stores either the addresses, or an alternative representation of addresses. In accordance with one embodiment of the present invention, an exemplary DRC1 address store 2230 is fully associative and is configured to store a complete unique address. For example, bits 35:6 for each address are stored in DRC1 to uniquely identify those addresses. Note that the addresses stored in DRC1 address store 2230 can be viewed as including common portions (e.g., tags) and delta portions (e.g., indexes), both of which are used to represent addresses during look-ahead lookup of DRC1 in accordance with at least one embodiment. Further, DRC1 address store 2230 and DRC1 data

store 2232 are configured to store 32 entries of addresses and 64 byte cache lines per address entry of data, respectively. Although prefetched data generally originates from a memory, such as a dynamic random access memory ("DRAM"), it can originate from a write back cache if data in DRC1 data store 2232 requires updating.

[0158] By contrast, an exemplary DRC2 address store 2240 can be composed of four-way set associative entries and can be configured to store base portions (e.g., tags) to represent addresses. Further, DRC2 address store 2240 and DRC2 data store 2242 are configured to store 1024 entries of addresses and 64 byte cache lines per address entry of data, respectively. DRC2 data store 2242 stores prefetched data originating from DRC1 data store 2232, and in some implementations can be composed of any number of memory banks (e.g., four banks: 0, 1, 2, and 3).

[0159] Although the memory from which predictive information is prefetched is typically a DRAM memory (e.g., arranged in a Dual In-line Memory Module, or "DIMM"), the memory can be of any other known memory technology. Typically, the memory is subdivided into "pages," which are sections of memory available within a particular row address. When a particular page is accessed, or "opened," other pages are closed, with the process of opening and closing pages requiring time to complete. So, when a processor is executing program instructions in a somewhat scattershot fashion, in terms of fetching instructions and data from various memory locations of a DRAM memory, accesses to the memory are nonsequential. As such, a stream of read requests can extend over a page boundary. If the next address in the next page is not available, the processor normally must fetch program instructions and program data directly from the memory. This increases latency of retrieving such instructions and data. So by prefetching and storing predictive information that spans multiple pages in multi-level cache 2220, then latency related to opening pages is reduced in accordance with the present invention. And because data being prefetched comes from the cache, the latency seen by, or with respect to, the processor is reduced while an accessed page remains opened.

[0160] For example, consider that nonsequential predictor 2104 correctly predicts that address "00200" is to be accessed following a processor read of address "00100." Therefore, nonsequential predictor 2104 causes a range of lines (e.g., one target address and four predicted address, the number of predictions generate being configurable and defined by a batch, "b") starting at address "00200" (as well as addresses 00201, 00202, 00203 and 00204, if batch is four) to be fetched in advance of the processor actually accessing address "00200."

When the processor does actually perform a read for address "00200," look-ahead lookup of multi-level cache 2220 quickly determines which cache lines within a specified range following address "00200" have already been prefetched. Since nonsequential transitions in the read address stream can be accompanied by a DRAM page opening operation, the look-ahead lookup allows prefetcher 2100 to quickly look ahead in a stream of read requests and determine which address or cache line needs to be fetched. By beginning the fetch quickly, prefetcher 2100 can often hide the latency of the DRAM page opening operation, and thereafter provide a sequential stream of cache lines (albeit nonsequential with the trigger address forming the basis for the target address) without incurring a latency penalty on the processor.

[0161] FIG. 22 depicts DRC1 manager 2234 and DRC2 manager 2246 as separate entities, but they need not be. That is, DRC1 manager 2234 and DRC2 manager 2246 can be combined into a single management entity or can be disposed external to multi-level cache 2220, or both. As first level DRC 2222 and second level DRC 2224 are structurally and/or functionally unlike conventional L1 and L2 caches resident in a processor, unique policies of managing the predictive information stored within multi-level cache 2220 are employed. Examples of such policies include a policy for allocating memory in each data return cache, a policy for copying information from a short term to a long term data store, and a policy for maintaining coherency between multi-level cache 2220 and another entity, such as a write-back cache.

[0162] First, consider a copying policy used to manage the copying of predictive information from first level DRC 2222 to second level DRC 2224 as that information ages from short term to long term information. DRC1 manager 2234 cooperates with DRC2 manager 2246 to transfer data from DRC1 data store 2232 to DRC2 data store 2242 when that data has been in first level DRC 2222 up to a certain threshold of time. Note that the threshold can be constant or can otherwise vary during operation. Typically, aged data can be configured to be transferred whenever there are less than N invalid entries (i.e., available) in DRC1, where N is programmable. In operation, once the data has been copied from short term to long term storage, the entry in first level DRC 2222 is erased (i.e., invalidated).

[0163] Second, consider an allocation policy for inserting predictive information in first level DRC 2222 and second level DRC 2224. When inserting predictive information into first level DRC 2222, DRC1 manager 2234 selects any invalid entries in DRC1 data store 2232, excluding locked entries as candidates. If DRC1 manager 2234 does not detect

any invalid entries into which predictive information can be stored, then the oldest entry can by used to allocate space for an entry. As for allocating entries in DRC2 data store 2242, DRC2 manager 2246 can use any of a number of ways (e.g., one of four ways) for receiving data copied from first level DRC 2222 to second level DRC 2224. For example, an index of the predicted address can contain four entries in which to store data. Initially, DRC2 data store 2242 allocates any one of the number of ways that are not being used (i.e., invalidated). But if all ways are assigned, then the first one in is the first one out (i.e., the oldest is overwritten). But if the oldest entries have the same age and are valid, DRC2 manager 2246 allocates the unlocked entry. Lastly, if all of the entries in the set of ways are locked, then DRC2 manager 2246 suppresses writes from first level DRC 2222 to second level DRC 2224 while maintaining the entry in first level DRC 2222 as valid. Again, note that typically second level DRC 2224 receives data for storage from only first level DRC 2222.

[0164] Another policy to which DRC1 manager 2234 and DRC2 manager 2246 can adhere to relates to maintaining coherency. DRC1 manager 2234 maintains first level DRC 2222 coherency by updating the data of any entry that has an address that matches the write address to which data will be written. Typically, write-back cache 2290 (FIG. 21) transitorily stores a write address (and corresponding data) until it sends the write address to write to memory (e.g., DRAM). Note that in some cases where there is an address of a read request that matches a write address in write-back cache 2290, then multi-level cache 2220 merges data of the write address with that of the memory prior to forwarding the data to first level DRC 2222. DRC2 manager 2246 maintains second level DRC 2224 coherency by invalidating any entry whose address matches a write address when it is loaded into write back cache 2290. As second level DRC 2224 only receives data from DRC1, and since first level DRC 2222 maintains coherency with memory and write-back cache 2290, then second level DRC 2224 generally will not contain stale data. Further, any address that is to be copied from DRC1 to DRC2 can be first checked against the write back cache ("WBC") 2290. If a match is found in WBC 2290, then the copy operation is aborted. Otherwise, the copying of that address from DRC1 to DRC2 takes place. This additional check further helps maintain coherency.

[0165] FIG. 23A illustrates an exemplary DRC1 query interface 2323 for a first address store 2305 in accordance with a specific embodiment. In this example, a trigger address ("A0") 2300 (e.g., a processor-requested address) is composed, as an input address, of a common address portion 2302a and a delta address portion 2302b. Note that address

2300 can also be either a predicted address in some cases, or a write address in other cases (when maintaining coherency). When address 2300 is a trigger address that generates a group of predicted addresses, then such a group 2307 can include addresses such as those identified from address ("A1") 2301 through to address ("Am") 2303, where "m" represents any number of predictions that can be used to perform "look-ahead lookup" in accordance with at least one embodiment of the present invention. In some cases, "m" is set equivalent to batch size, "b."

[0166] Entries 2306 of DRC1 address store 2305 each include a first entry portion 2306a (e.g., a tag) and a second entry portion 2306b (e.g., an index). In a specific embodiment, first entry portion 2306a and second entry portion 2306b are respectively analogous to common address portion 2302a and delta address portion 2302b. Second entry portions 2306b indicate the displacement in terms of address from trigger address ("A0") 2300 to that particular entry 2306. So, when DRC1 matcher 2312 compares an input address, such as trigger address ("A0") 2300, to entries 2306, common portion 2302a can be used to represent the common portions of the addresses of group 2307. Further, as common portion 2302a of address 2300 is generally similar to the common portions for addresses ("A1") 2301 through to ("Am") 2303, then only common portion 2302a need be used to compare against one or more first entry portions 2306a of entries 2306. Also, delta portions 2302b for addresses ("A1") 2301 through to ("Am") 2303 can be matched against multiple second entry portions 2306b of entries 2306.

[0167] In one embodiment, DRC1 matcher 2312 includes common comparators 2308 to match common address portions against first entry portions, and delta comparators 2310 to match delta address portions against second entry portions. Specifically, common portion 2302a is simultaneously compared against first portions 2306a for Entry 0 through to the nth Entry, and delta portions 2302b are simultaneously compared against second portions 2306b for the same entries. In some embodiments, common comparator 2308 is a "wide" comparator for comparing high-order bits (e.g., bits 35:12 of a 36-bit address) and delta comparator 2310 is a "narrow" comparator for comparing low-order bits (e.g., bits 11:6 of a 36-bit address). Note that although FIG. 23A depicts one delta comparator per delta portion 2302b, in some cases, the number of delta comparators 2310 is equal to m * n (not shown), where each delta comparator would receive one delta portion 2302b and one second entry portion 2306b as inputs. The comparator sizes limit the amount of physical resources required to perform these comparisons, and as such, addresses that are looked up in parallel

are configured to lie within the same memory page (e.g., a memory page size is typically 4K bytes). Though this reduces the addresses of look-ahead lookups from crossing page boundaries, these configurations decrease the cost for performing look-ahead lookups in terms of physical resources. Again note that common portion 2302a and delta portions 2302b are each compared simultaneously, or nearly so, with entries 2306.

[0168] The output of common comparator 2308 and delta comparators 2310 are Hbase(0), Hbase(1), . . . Hbase(m) and H0, H1, H2, . . . HN, respectively, where each are either zero (e.g., indicating no match) or one (e.g., indicating match). The results form a hit vector of zeros and ones that are sent to DRC1 hander 2314 to take action, depending on whether it is filtering or maintaining coherency. Hit list generator 2313 generates a list of hits ("hit list") indicating which addresses in range "r" (i.e., group 2307) reside in DRC1 address store 2305. If an address is matched (i.e., a prediction is stored therein), then that address is included in the hit list, whereas an unmatched address (i.e., a prediction is not stored) is excluded from the hit list. This hit list is used to generate predictions or to manage coherency within DRC1 address store 2305.

[0169] FIG. 23B depicts a number of exemplary input addresses 2352 that can be examined in parallel using DRC1 query interface 2323 of FIG. 23A in accordance with a specific embodiment. Here, DRC1 query interface 2350 can accept any range of addresses 2352 to match against DRC1 address store 2305. Matcher 2312 of FIG. 23A is replicated as many times as is necessary to perform a parallel look-ahead lookup over a number of input addresses. As an example, for forward sequential predictions, with batch size "b" set to 27, DRC1 query interface 2350 would require matchers to match A0, as a base (or trigger) address in parallel with predicted addresses A1 to A7 as group 2307. For blind back predictions, only A(-1) requires matching besides base address A0 as group 2307, but for reverse sequential predictions, addresses A(-1) to A(-7) require matching. Note that range of addresses 2352 can be applied simultaneously, in parallel, to both the DRC1 and DRC2 query interface as well.

[0170] FIG. 24 illustrates an exemplary DRC2 query interface 2403 for DRC2 address store 2404 in accordance with a specific embodiment. DRC2 query interface 2403 is configured to receive an input address 2402 for comparing that address against the contents of DRC2 address store 2404. In this example, input address 2402 is a base portion (e.g., a tag) of an address, such as a tag(A0). Further to this example, DRC2 address store 2404 is composed of four banks 2406 of memory, banks 0, 1, 2, and 3, each bank including entries

2410. Note that in this case, an entry 2410 can be placed into any one of four ways (W0, W1, W2, and W3).

[0171] DRC2 matcher 2430 includes a number of comparators to compare tag(A0) against entries 2410. Generally, any matching address in DRC2 address store 2404 shares the same tag(A0), but can differ in relation to another group of bits (e.g., by an index). In a specific embodiment of the present invention, the determination of whether a tag matches any entry within DRC2 address store 2404 is generally performed as follows. First, for each bank 2406, one of the indexes in that bank is selected to be searched for potential matching addresses. This can vary per bank, as shown in FIG. 25A, because the selected bank to be searched depends on which one of the banks a specific address (e.g., A0 of FIG. 25) resides, as banks can be identified by certain index bits of the specific address (e.g., A0). Second, all four ways of the index chosen for each bank 2406 are accessed. Then, the tags stored in relation to the four ways (e.g., W0 to W3) are compared against tag(A0), which in this example is base address 2402. Generally, it is sufficient to compare tag(A0) without comparing other tags, such as tag(A1). This is because these tags are generally assumed to be identical (e.g., tag(A0) = tag(A1) = tag(A2)). Note that a simultaneous search for predictions is typically limited to those that lie in the same page, such as a 4 kbyte page, which causes the tags to be the same. Third, once an address match is made by DRC2 matcher 2430, results in the form of a hit vector and the valid bits are used to get a final hit vector similar to as described in connection with FIGs. 27 and 28.

[0172] Hit generator 2442 of DRC2 query interface 2403 receives the tag comparison results ("TCR") 2422 from DRC2 matcher 2430, and further compares those result against corresponding valid bits 2450 to generate an ordered set of predictions ("ordered predictions"). Here, tag comparison results from banks 1, 2, 3 and 4 are respectively labeled TCR(a), TCR(b), TCR(c), and TCR(d), each including one or more bits representing whether a tags matches one or more entries 2410. Ordered predictions can be an ordered set of predictions that match (or do not match) input address 2402. Or, ordered predictions can be a vector of bits each indicating whether an input address has an address that is present in DRC2 address store 2404. Note that any number of input addresses 2402 can be similarly matched against DRC2 query interface 2403 if additional DRC2 matchers 2430 are included. FIGs. 25A to 28 describe exemplary hit generators in accordance with some embodiments of the present invention.

[0173] FIG. 25A depicts possible arrangements of addresses (or representations thereof) as stored in DRC2 address store 2404, according to one embodiment of the present invention. Note that ways W0, W1, W2 and W3 are not shown so as to simplify the following discussion. Input addresses A0, A1, A2, and A3 are stored in DRC2 address store 2404. As an example, sequential predictor 2102 (not shown) can generate sequential predictions A1, A2, and A3 based on trigger address A0 (e.g., in any of four ways). A first arrangement 2502 results from A0 being stored in bank 0. Likewise, second arrangement 2504, third arrangement 2506 and fourth arrangement 2508 each respectively resulting from storing address A0 in banks 1, 2, and 3, with subsequent addresses stored in series following the trigger address. As such, these addresses (or portions thereof, such as in the form of tags) generally are output from DRC2 address store 2404 in no particular order.

[0174] FIG. 25B depicts an exemplary hit generator 2430 that generates results based on unordered addresses and corresponding valid bits, according to an embodiment of the present invention. In this example, sequential predictor 2102 generates sequential predictions A1, A2, A3, A4, A5, A6 and A7 based on trigger address A0, all of which are stored in the particular arrangement shown (i.e., trigger address A0 is stored in bank 1 with the others following). Hit generator 2430 receives unordered addresses A2, A6, A1, A5, A0, A4, A3, A7 and ordered valid bits VB0 to VB7, orders them, compares them and then generates results R0 to R7, which can be a bit vector or a list of addresses (either those that match or those that do not). Note that a valid bit indicating that a prediction is invalid prevents a stored, invalid prediction from being matched. This is one reason to match valid bits against contents of an address store. In accordance with a specific embodiment of the present invention, four addresses rather than eight are simultaneously considered, such as either addresses A2, A1, A0 and A3 or addresses A6, A5, A4 and A7. As such, there is no need to represent addresses A0 to A7 as being contiguously accessible in a "wrapped" manner, as is shown in FIG. 25B. To simultaneously consider addresses A0 to A7 of FIG. 25B, however, DRC2 can be configured as a double-ported random access memory ("RAM") to perform two independent and simultaneous accesses to the same RAM (or same DRC2).

[0175] FIG. 26 is a schematic representation of a hit generator 2600 for hit generator 2442 of FIG. 24. Hit generator 2600 generates one or more of results R0 to R7 by multiplexing addresses from ways 0 to 3 and/or valid bits for each input address, where the result, R, is determined by comparing the multiplexed bits of addresses or valid bits. If a valid bit indicates that the tag indicated by the corresponding tag comparison result ("TCR")

is valid, then that tag is output as result R. Note that a TCR can be a tag of an address or it can be a bit having a value of either a "1" (i.e., hit in DRC2) or "0" (i.e., no hit in DRC2). As is described below in connection with FIGs. 27 and 28, a tag for an address (e.g., tag(A1)) generally represents a single TCR bit for that tag.

[0176] FIG. 27 depicts one example of hit generator 2442, according to one embodiment of the present invention. Hit generator 2442 includes an orderer 2702 configured to order the unordered tags for address A3, A0, A1 and A2 from the ways of banks 0, 1, 2, and 3, respectively. But note that tags for address A3, A0, A1 and A2 each represent single bits representing TCRs for each tag. Next the ordered TCRs (shown as ordered tags for addresses A0, A1, A2, and A3) are tested against valid bits VB0-VB3 from valid bits 2244. AND operator ("AND") 2706 performs the test as a logical AND function. So, if a valid bit is true and a single-bit TCR is true, then there is a hit and the results, R, reflect this. That is, the results R0, R1, R2, and R3 form the ordered prediction results, which again can be bits representing match/no match, or can be matched tags for addresses or those that do not. Note that in cases where the tag itself is used as a TCR (e.g., Tag(A3) as TCR(a)), then AND operator 2706 operates to mask those bits if the corresponding valid bit is zero (e.g., a result, R, will contain all zeros if its corresponding valid bit is zero).

[0177] FIG. 28 depicts another example of hit generator 2442, according to another embodiment of the present invention. Hit generator 2442 includes a valid bit ("VB") orderer 2802 configured to disorder the ordered valid bits VB0-VB3 from valid bits 2224. That is, valid bit orderer 2802 reorders valid bits from having a order VB0, VB1, VB2, and VB3 to an order of VB3, VB0, VB1 and VB2, which matches the order of the TCRs, which are represented by tags for addresses A3, A0, A1 and A2. Next the unordered tags for the addresses (i.e., unordered TCRs for those tags) are tested against the similarly ordered valid bits by AND operators ("AND") 2806. The unordered results R3, R0, R1 and R2 pass through result orderer 2810 to obtain R0, R1, R2, and R3 as ordered prediction results, which is the form useable by prefetcher 2100 and its elements that performing filtering, coherency, etc. By reordering valid bits and results (which can be just result bits), less hardware is necessary than reordering addresses each composed of a number of bits. Note that the orderings of orderer 2702 and result orderer 2810 are exemplary and other mappings to order and reorder bit are within the scope of the present invention.

[0178] In a specific embodiment of the present invention, prefetcher 2100 of FIG. 21, which includes nonsequential predictor 2104 and multi-level cache 2120, is disposed within a

Northbridge-Southbridge chipset architecture, such as within a memory processor having at least some of the same functionalities of a Northbridge chip. A memory processor is designed to at least control memory accesses by one or more processors, such as CPUs, graphics processor units ("GPUs"), etc. In a Northbridge implementation, prefetcher 2100 can also be coupled via an AGP/PCI Express interface to a GPU. Moreover, a front side bus ("FSB") can be used as a system bus between a processor and a memory. Also, a memory can be a system memory. Alternatively, multi-level cache 2120 can be employed in any other structure, circuit, device, etc. serving to control accesses to memory, as does the memory processor. Further, multi-level cache 2120 and its elements, as well as other components of prefetcher 2100, can be composed of either hardware or software modules, or both, and further can be distributed or combined in any manner.

[0179] The foregoing description, for purposes of explanation, used specific nomenclature to provide a thorough understanding of the invention. However, it will be apparent to one skilled in the art that specific details are not required in order to practice the invention. Thus, the foregoing descriptions of specific embodiments of the invention are presented for purposes of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed; obviously, many modifications and variations are possible in view of the above teachings. In fact, this description should not be read to limit any feature or aspect of the present invention to any embodiment; rather features and aspects of one embodiment may readily be interchanged with other embodiments. The embodiments were chosen and described in order to best explain the principles of the invention and its practical applications, they thereby enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to the particular use contemplated. It is intended that the following claims and their equivalents define the scope of the invention.

In the claims:

1. A prefetcher for predicting accesses to a memory comprising:

a first address predictor configured to:

associate a subset of addresses to an address and to

predict a group of addresses based on at least one address of said subset,

wherein said at least one address of said subset is unpatternable to said address.

2. The prefetcher of claim 1 wherein said first address predictor further comprises a nonsequential predictor to generate said group of addresses as nonsequential predictions when said address is detected.

3. The prefetcher of claim 2 wherein said nonsequential predictor further comprises:

a repository for storing associations of said subset of addresses to said address in a manner that prioritizes each of said subset of addresses in relation to others, wherein said address is stored as a trigger address and said subset of addresses are stored as target addresses; and

a nonsequential prediction engine configured to detect said address in a stream of addresses and is configured further to select said at least one address as a nonsequential prediction based on its priority being a highest priority.

4. The prefetcher of claim 3 wherein said highest priority is at least indicative that a processor has requested said at least one address most recently relative to others of said subset of addresses.

5. The prefetcher of claim 3 wherein said first address predictor is configured to generate indexes and tags from addresses, and said repository includes a number of ways each having memory locations for storing trigger-target associations, wherein a trigger-target association stored in a first way is associated with a higher priority than another trigger-target association stored in a second way.

6. The prefetcher of claim 5 wherein each of said trigger-target associations includes a tag having a tag size and at least a portion of a target address having a portion size, wherein

said tag is representative of a trigger address and said tag size and said portions size are configured to minimize the size requirements for a memory location.

7. The prefetcher of claim 5 wherein said first address predictor is configured to compare a first address tag against each tag identified by one or more of said indexes to detect any trigger-target association that includes said first address tag, wherein said first address predictor either uses a target address from at least one trigger-target association to form a nonsequential prediction, or as a successive trigger address to form additional nonsequential predictions based on one or more other trigger-target associations, or both, wherein each of said one or more other trigger-target associations relate to lower levels in said target cache than either said trigger-target association or said another trigger-target association.

8. The prefetcher of claim 5 further comprising a priority adjuster configured to modify a priority for one of said trigger-target associations that includes a target address that matches a second address, said target address being identified by a trigger address composed of one or more of said indexes and a first address tag.

9. The prefetcher of claim 3 further comprising an expediter to designate a first address of a sequential stream of addresses as a new trigger address for said at least one address if said trigger address is in said sequential stream and if said nonsequential predictions are generated sooner in time with said new trigger address than with said trigger address.

10. The prefetcher of claim 3 further comprising a suppressor configured to suppress generating at least one predicted address.

11. The prefetcher of claim 10 wherein said suppressor is configured to decrease a batch quantity of addresses for said group if said address is related to either a request for data or a prefetch request, or both, thereby suppressing generation of said at least one predicted address.

12. The prefetcher of claim 10 wherein said suppressor is further configured to suppress generating said group of addresses as nonsequential predictions if an interval of time from

detection of said address as said trigger address to generation of said group of addresses as nonsequential predictions is less than a threshold, wherein said threshold is defined by at least an amount of time between a first processor request for said trigger address and a second processor request for said at least one address that is less than the time necessary to prefetch said at least one of said group of addresses from memory.

13. The prefetcher of claim 10 wherein said suppressor is further configured to track a base address and a last-detected address for each of a plurality of interleaved sequential streams and to determine whether another address is within an address stream from said base address to said last-detected address for any of said plurality of interleaved sequential streams, and if so, suppress generating said at least predicted address based on said another address.

14. The prefetcher of claim 13 wherein each of a plurality of interleaved sequential streams is a portion of one of a number of threads.

15. The prefetcher of claim 10 further comprising a second address predictor that includes a sequential predictor for generating a number of additional predicted addresses based on at least one other address.

16. The prefetcher of claim 15 wherein said number of additional predicted addresses includes either:

a first number of addresses sequenced in an ascending order from said at least one other address, or

a second number of addresses sequenced in a descending order from said at least one other address,

or both said first and said second number of addresses,

wherein said suppressor is further configured to

detect that said at least one other address is part of a first address stream ascending in order and to suppress those of said number of additional predicted addresses that are based on said second number of addresses sequenced in said descending order, and to

detect that said at least one other address is part of a second address stream descending in order and to suppress those of said number of additional predicted addresses that are based on said first number of addresses sequenced in said ascending order.

17. The prefetcher of claim 15 wherein said number of additional predicted addresses include either:

a back address sequenced by one in a descending order from said at least one other address, or

a back sector address of said at least one other address, or both,

wherein said suppressor is further configured to decrease said batch quantity by one if said number of additional predicted addresses includes either said back address or said back sector address.

18. The prefetcher of claim 15 further comprising:

a prediction inventory comprising a number of queues each configured to maintain predictions of the same type of prediction until either issued or filtered out; and

an inventory filter to generate a subset of filtered addresses, said inventory filter configured to filter a redundant address in either

said prediction inventory or

said group of addresses and said number of additional predicted addresses,

wherein said prefetcher is configured to provide at least one of said subset of filtered addresses.

19. The prefetcher of claim 18 said number of queues further comprises at least one queue that maintains a different type of prediction than maintained in other queues of said number of queues.

20. The prefetcher of claim 18 further comprising an inventory manager configured to control each of said number of queues in accordance with one or more queue attributes of:

a type of queue,

an expiration time,

a queue size,

an insertion indicator indicating a manner to insert an incoming prediction into a full queue, and

a priority with which to select a next prediction.

21. The prefetcher of claim 20 wherein said number of queues further comprises:

a sequential queue, said sequential queue having a sequential queue priority;

a back queue, said back queue having a back queue priority configurable to indicate a precedence over said sequential queue priority; and

one or more nonsequential queues each having a unique priority relative to the priorities of the other queues.

22. The prefetcher of claim 20 wherein said inventory manager manages predictions by groups of items, each group of items including a triggering address and at least one item, wherein each of said number of queues is configured to be searched for matching said predictions against other predictions external to said number of queues upon issuance.

23. The prefetcher of claim 22 further comprising an arbiter configured to issue said at least one item as an issued item for accessing a memory, wherein said issued item is selected based on the priority for an issuing queue, said priority being modifiable by said arbiter upon detecting that said issuing queue is contributing to a memory over-utilization.

24. The prefetcher of claim 23 further comprising:

a cache memory containing predictive information and references; and

a post-inventory filter configured to compare said issued item against said references for filtering out said issued item as a redundant prediction.

25. The prefetcher of claim 15 further comprising a cache memory for managing predictive accesses to a memory comprising:

a short term cache memory configured to store predictions having an age less than a threshold;

a long term cache memory configured to store predictions having an age greater than or equal to said threshold, said long term cache memory having more memory capacity than said short term cache; and

an interface configured to detect, in parallel, whether multiple predictions are stored in either said short term cache memory or said long term cache memory, or both,

wherein said interface uses at least two representations of each of said multiple predictions when examining said short term cache and said long term cache memories.

26. The prefetcher of claim 25 further comprising a data return cache manager configured to copy one stored prediction as a copied prediction from said short term cache memory into said long term cache memory when said one stored prediction ages past said threshold, said short term cache being the only source of data for said long term cache.

27. The prefetcher of claim 26 wherein said data return cache manager is further configured to either:

store said copied prediction in an entry of said long term cache memory, said entry being in one of a number of ways that are available, or

store said copied prediction in said entry of said long term cache memory if none of said number of ways are available, said entry containing an oldest stored prediction.

28. The prefetcher of claim 26 wherein said data return cache manager is further configured to either:

store a prediction in an entry of said short term cache memory, said entry containing an invalid prediction, or

store said prediction in another entry of said short term cache memory containing an oldest prediction.

29. The prefetcher of claim 26 wherein said data return cache manager is further configured to

match a write address against a next stored prediction to form a matched prediction,

merge at least a portion of data of said write address with a portion of predictive information of said matched prediction, if said next stored prediction is stored in said short term cache memory, and

invalidate said next prediction, if said next stored prediction is stored in said long term cache memory.

30. The prefetcher of claim 25 wherein said short and said long term caches are configured to reduce latency with respect to a processor due to opening pages of a memory by storing nonsequential predictions as a subset of a range of predictions, said subset including predictions in two or more pages of memory, wherein said first address predictor generates said range of predictions in response to a trigger address, said trigger address being in a different page of memory than any page containing said range of predictions, wherein said short and said long term caches are configured to store a prediction identifier associated with each entry of said stored predictions and transmit said prediction identifier to said first address predictor, said long term cache being configured to store valid bits separately from each entry configured to store predictions.

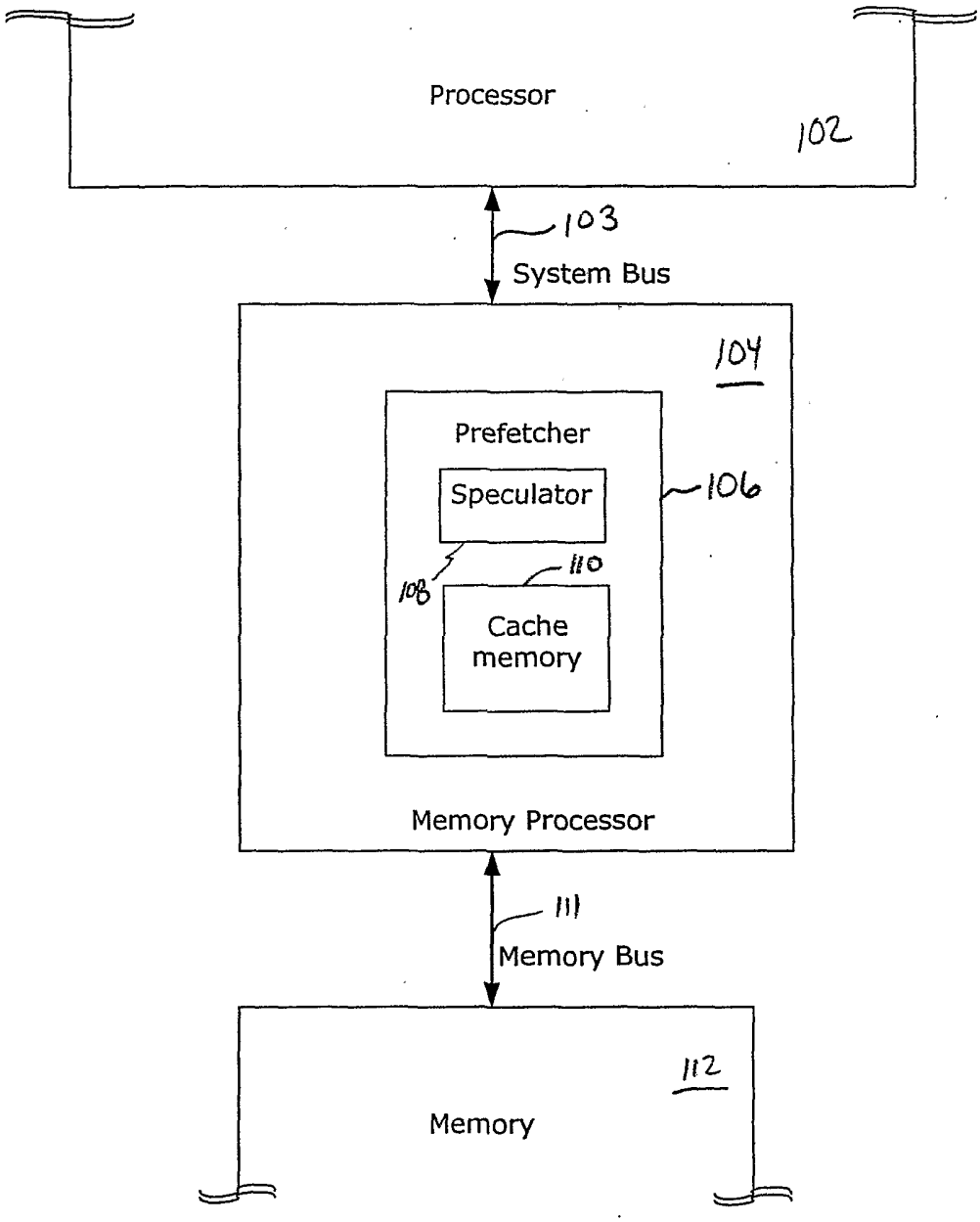

Processor
102
103
System Bus
104
Prefetcher
Speculator
106
108
110
Cache memory
Memory Processor
111
Memory Bus
112
Memory

FIG. 1

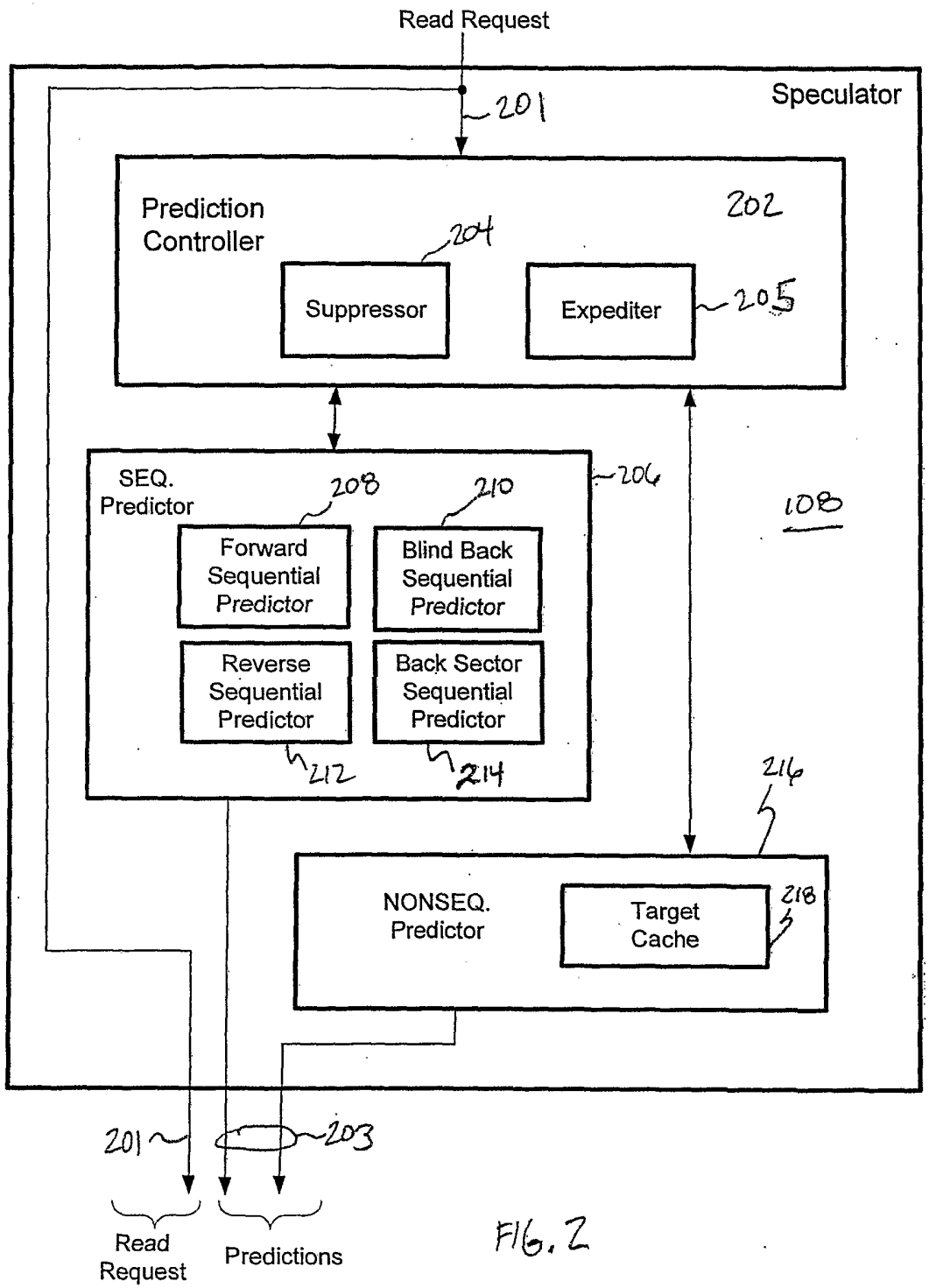

Read Request
201
Speculator
Prediction Controller
202
204
Suppressor
Expediter
205
SEQ. Predictor
206
208
210
Forward Sequential Predictor
Blind Back Sequential Predictor
Reverse Sequential Predictor
Back Sector Sequential Predictor
212
214
108
216
NONSEQ. Predictor
Target Cache
218
201
203
Read Request
Predictions

FIG. 2

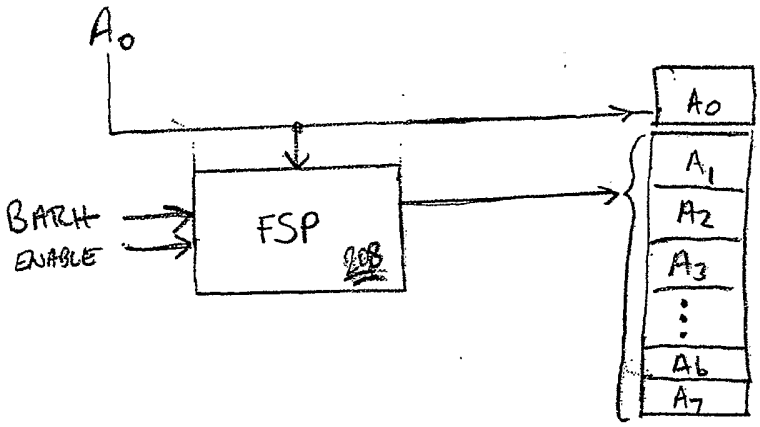


FIG. 3A

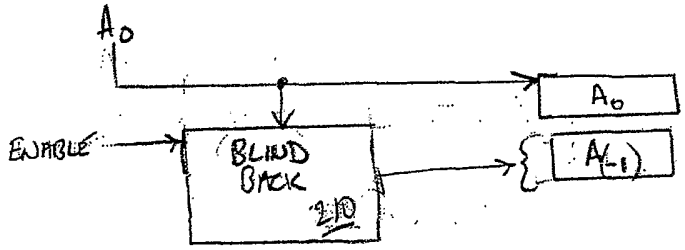


FIG. 3B

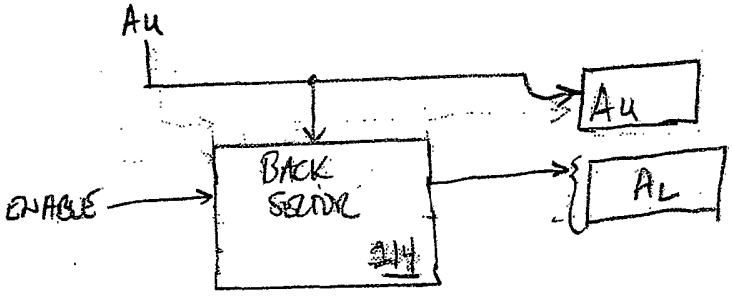


FIG. 3C

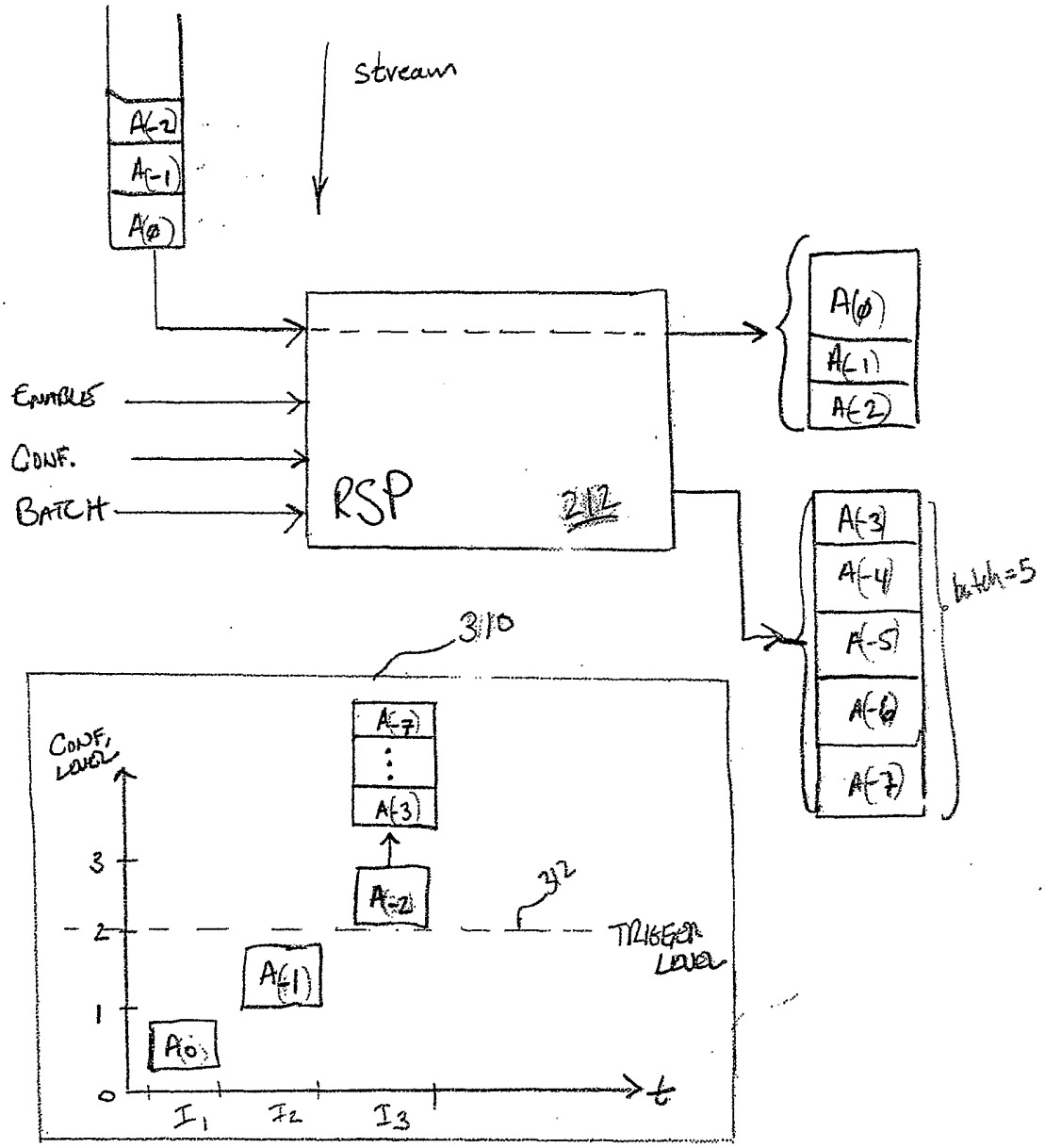

Stream
A(-2)
A(-1)
A(0)
ENABLES
CONF.
BATCH
RSP
212
A(0)
A(-1)
A(-2)
A(-3)
A(-4)
A(-5)
A(-6)
A(-7)
batch=5
310
CONF. LEVEL
3
2
1
0
A(-7)
A(-3)
A(-2)
A(-1)
A(0)
312
TRIGGER LEVEL
t
I1
I2
I3

FIG. 3D

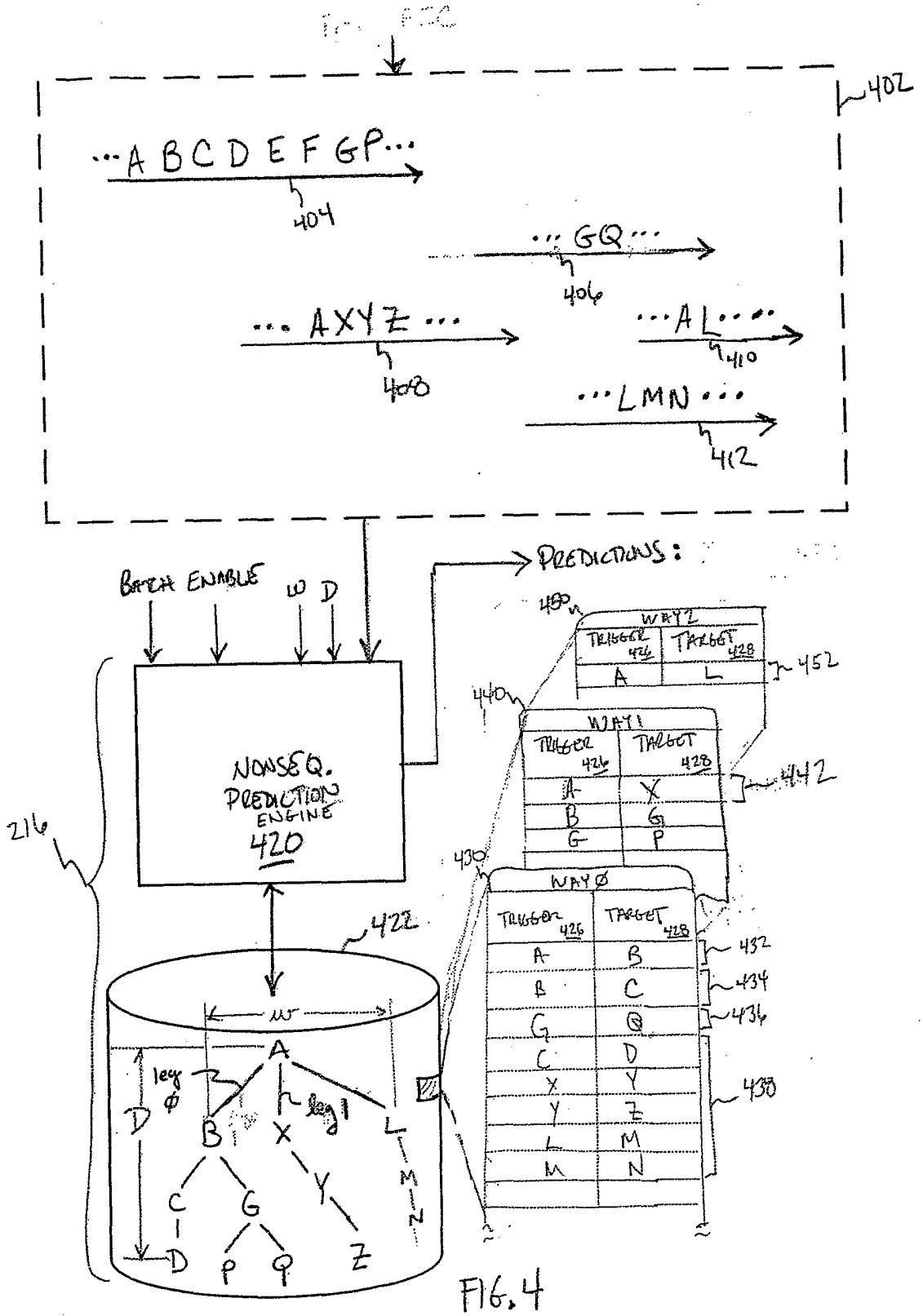

402
...ABCDEFGP...
404
...GQ...
406
...AXYZ...
408
...AL...
410
...LMN...
412
PREDICTIONS:
BATCH ENABLE
W
D
NONSEQ. PREDICTION ENGINE 420
216
422
450
WAY2
TRIGGER 426
TARGET 428
A
L
452
440
WAY1
A
X
B
G
G
P
442
430
WAY0
A
B
B
C
G
Q
C
D
X
Y
Y
Z
L
M
M
N
432
434
436
438
leg 0
leg 1
leg 2
leg 3

FIG. 4

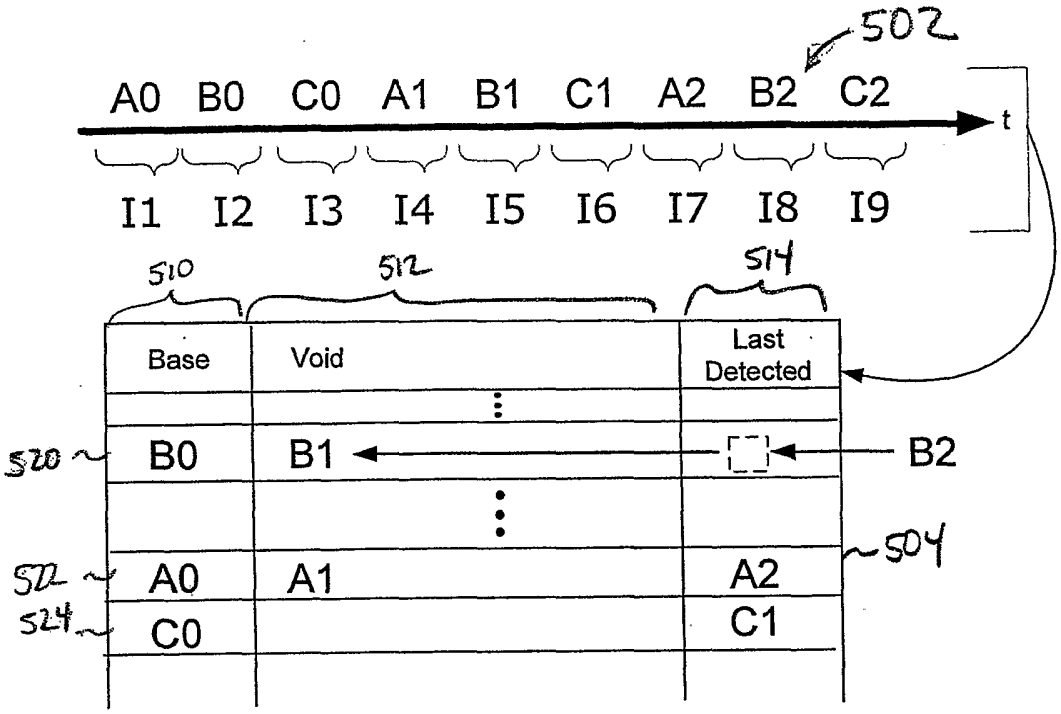

502
A0 B0 C0 A1 B1 C1 A2 B2 C2
t
I1 I2 I3 I4 I5 I6 I7 I8 I9
510
512
514
Base
Void
Last Detected
520
B0
B1
B2
504
522
A0
A1
A2
524
C0
C1

FIG. 5

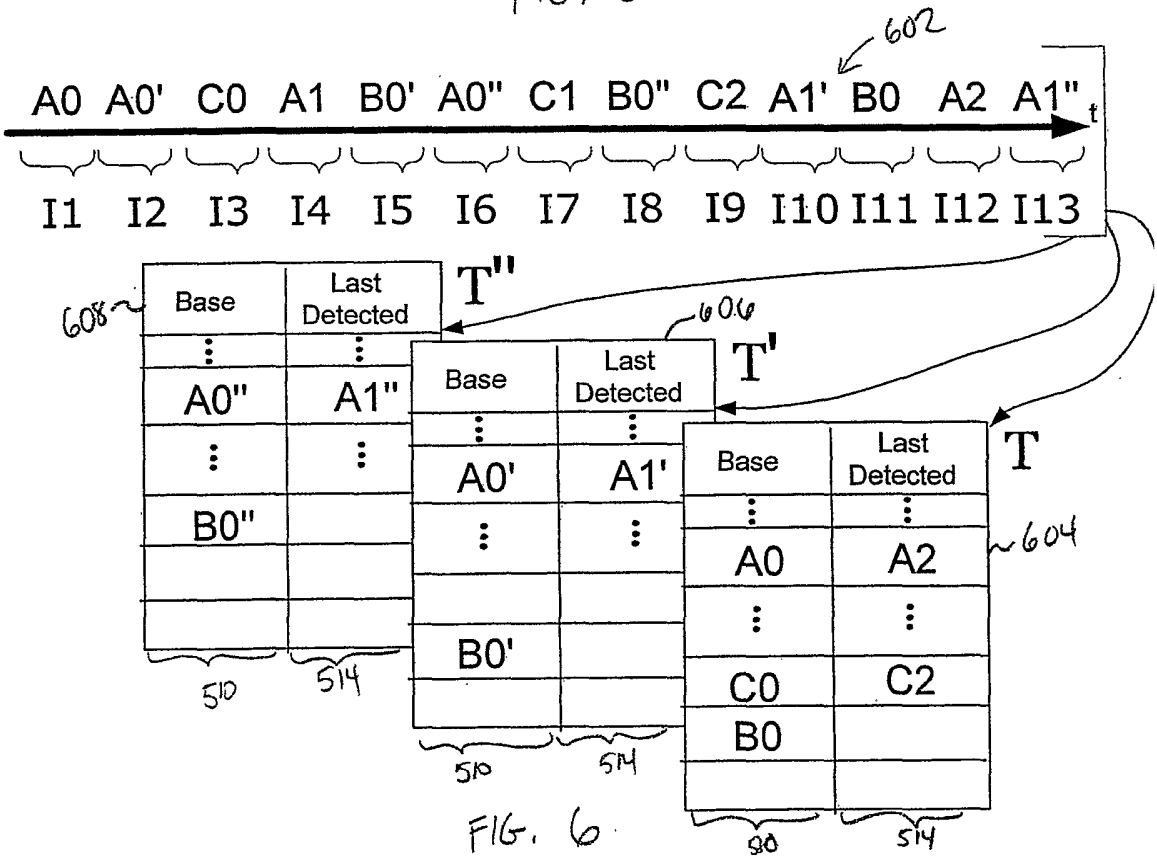

602
A0 A0' C0 A1 B0' A0" C1 B0" C2 A1' B0 A2 A1"
t
I1 I2 I3 I4 I5 I6 I7 I8 I9 I10 I11 I12 I13
T"
608
Base
Last Detected
A0"
A1"
B0"
510
514
606
T'
Base
Last Detected
A0'
A1'
B0'
510
514
T
Base
Last Detected
604
A0
A2
C0
C2
B0
90
514

FIG. 6

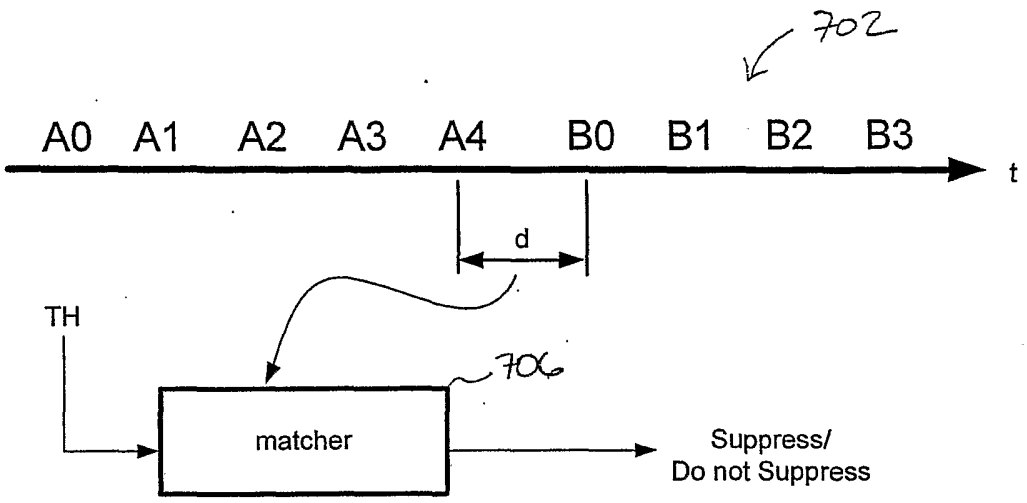

702
A0 A1 A2 A3 A4 B0 B1 B2 B3
t
d
TH
706
matcher
Suppress/
Do not Suppress

FIG. 7

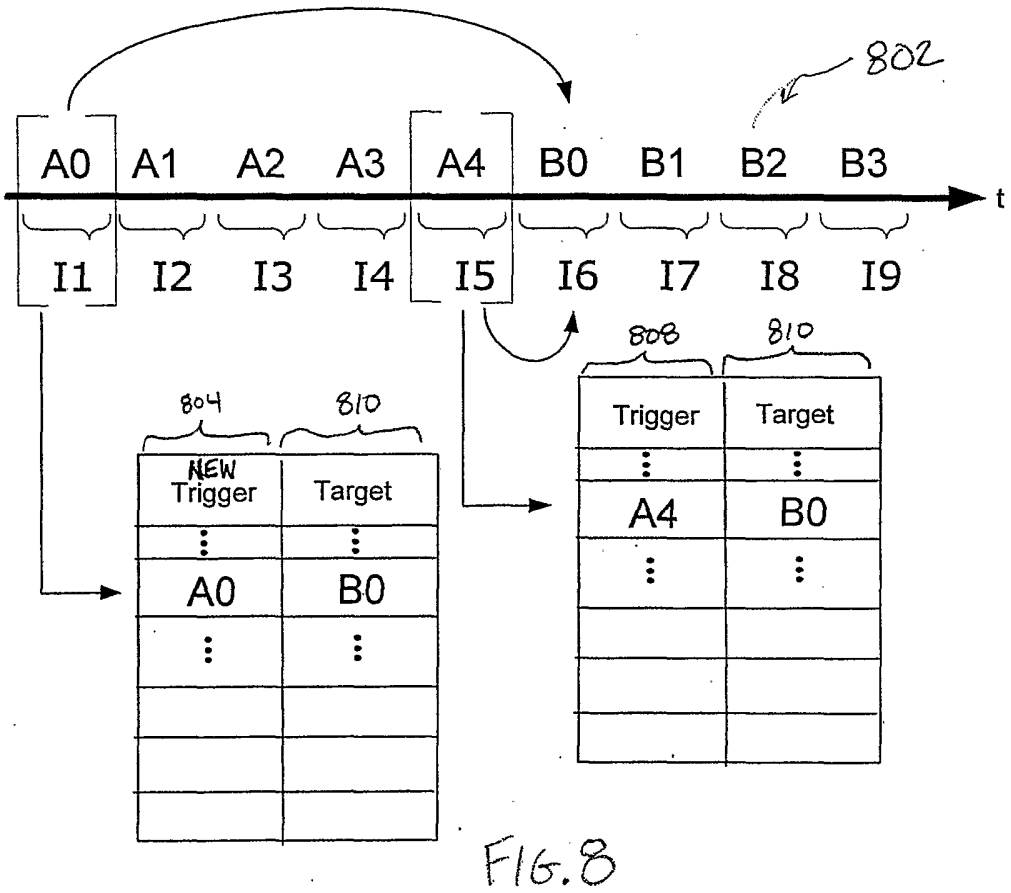

802
A0 A1 A2 A3 A4 B0 B1 B2 B3
t
I1 I2 I3 I4 I5 I6 I7 I8 I9
804
810
NEW Trigger
Target
A0
B0
808
810
Trigger
Target
A4
B0

FIG. 8

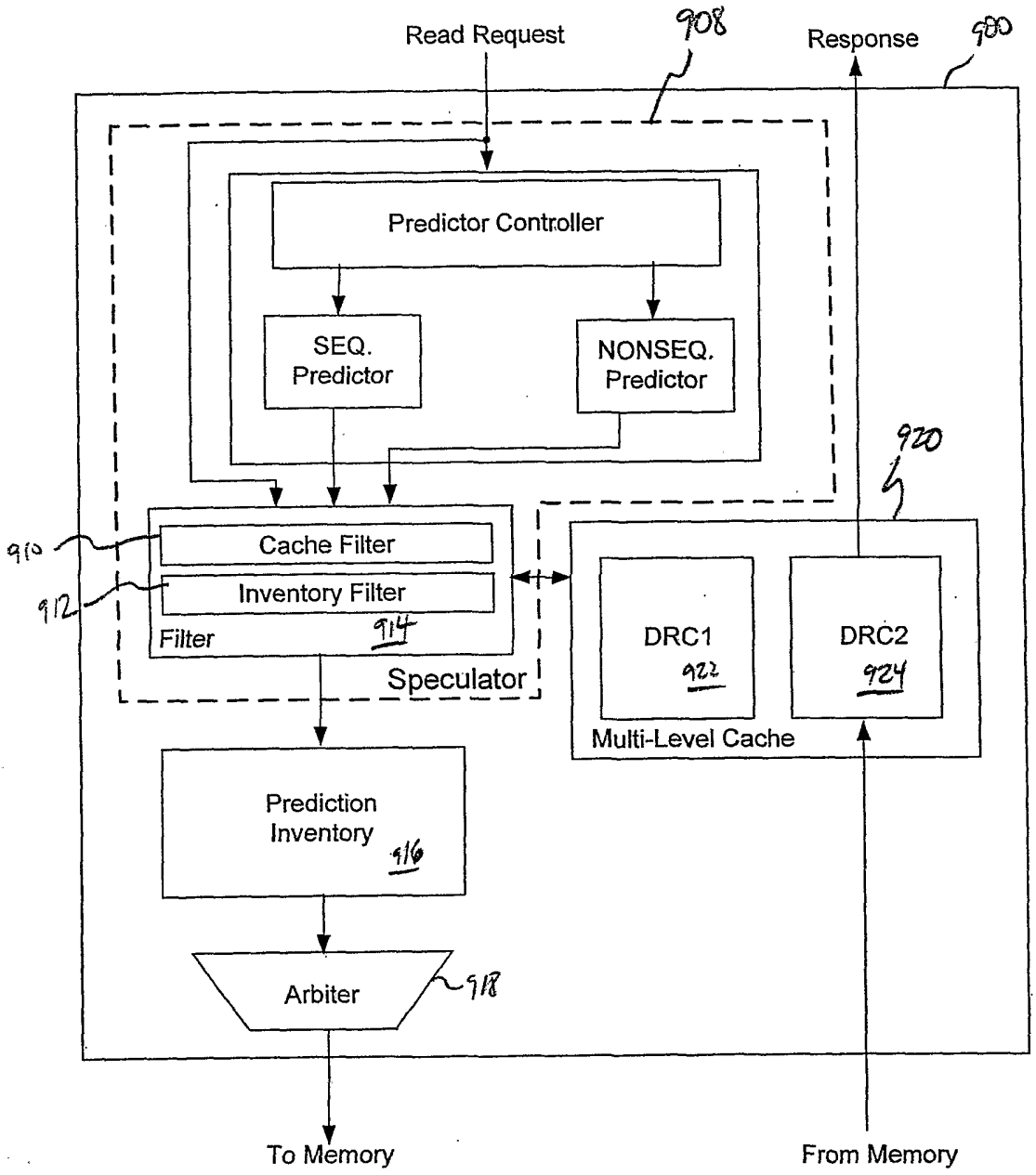

Read Request
908
Response
900
Predictor Controller
SEQ. Predictor
NONSEQ. Predictor
920
910
Cache Filter
912
Inventory Filter
Filter
914
Speculator
DRC1
922
DRC2
924
Multi-Level Cache
Prediction Inventory
916
Arbiter
918
To Memory
From Memory

FIG. 9

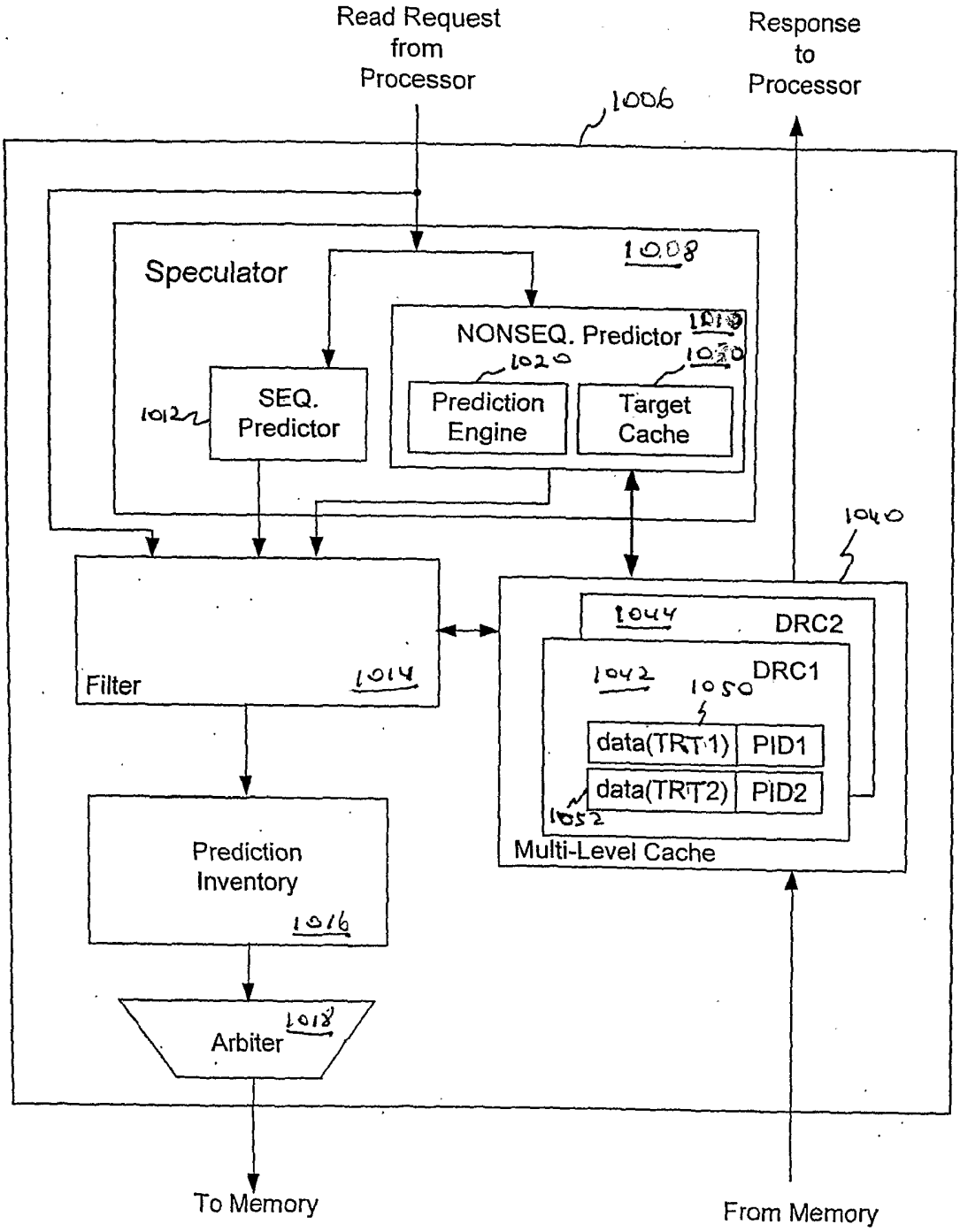

Read Request from Processor
Response to Processor
1006
Speculator
1008
NONSEQ. Predictor
1010
1020
1030
SEQ. Predictor
1012
Prediction Engine
Target Cache
1040
1044
DRC2
1042
DRC1
1050
data(TRT1)
PID1
data(TRT2)
PID2
1052
Multi-Level Cache
Filter
1014
Prediction Inventory
1016
Arbiter
1018
To Memory
From Memory

FIG. 10

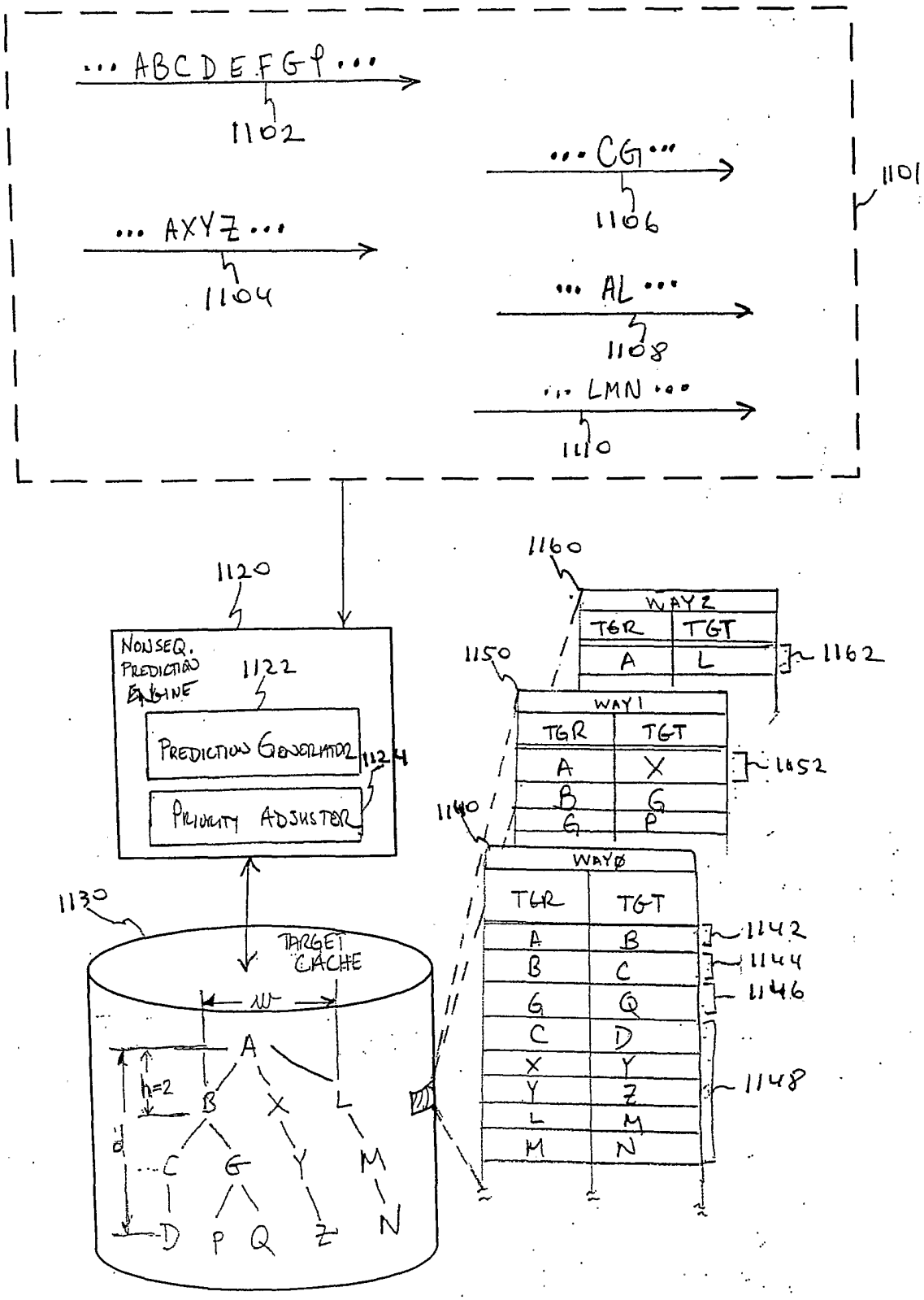

... ABCDEFGP ...
1102
... CG ...
1106
1101
... AXYZ ...
1104
... AL ...
1108
... LMN ...
1110
1120
NONSEQ. PREDICTION ENGINE
1122
PREDICTION GENERATOR
1124
PRIORITY ADJUSTER
1130
TARGET CACHE
W
h=2
d
A
B
X
L
C
G
Y
M
D
P
Q
Z
N
1160
WAY2
TGR
TGT
A
L
1162
1150
WAY1
TGR
TGT
A
X
B
G
G
P
1152
1140
WAY0
TGR
TGT
A
B
1142
B
C
1144
G
Q
1146
C
D
X
Y
Y
Z
L
M
M
N
1148

FIG. 11

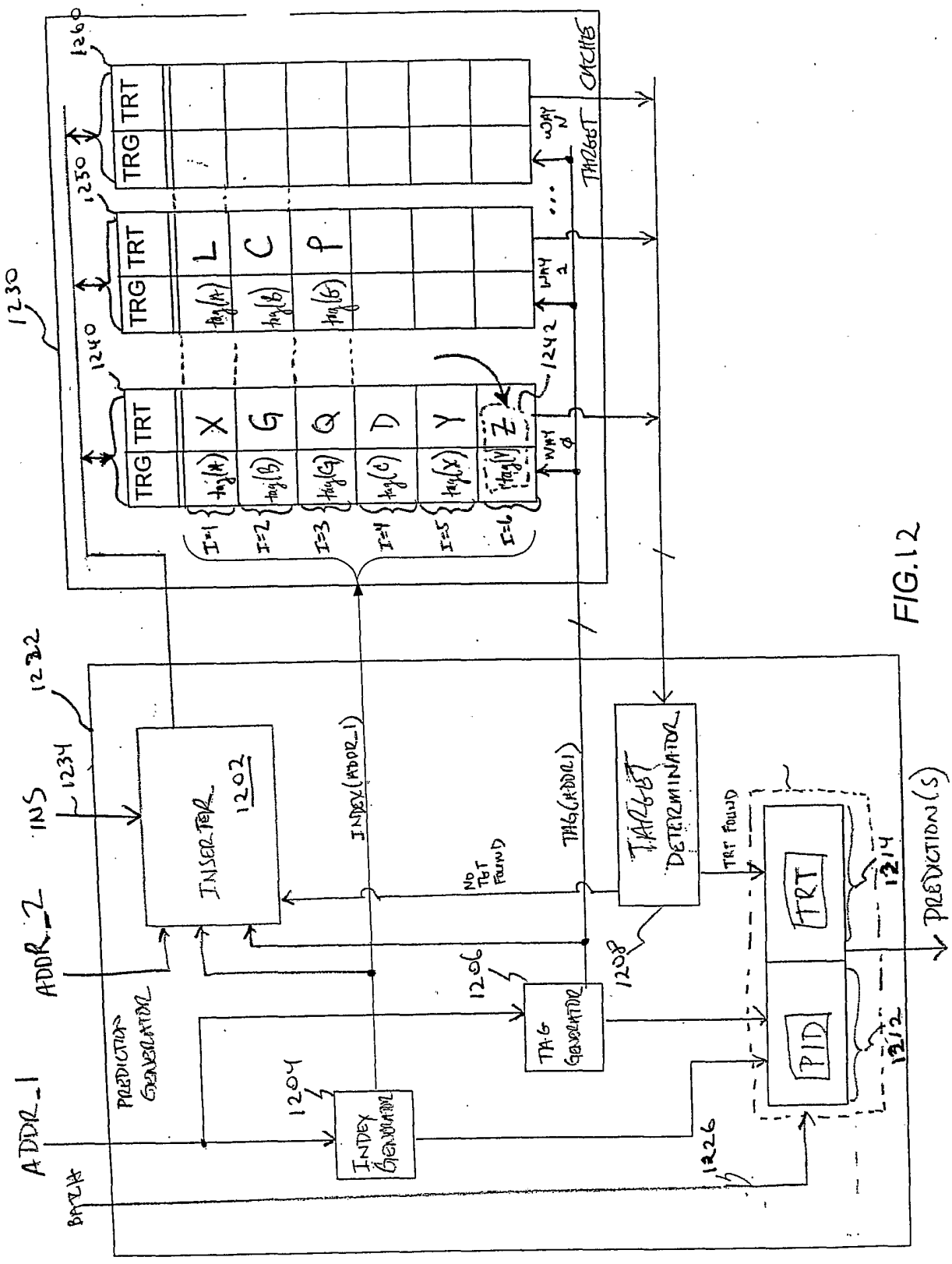

1230
1240
1250
1260
TRG
TRT
X
G
Q
D
Y
Z
L
C
P
WAY 0
WAY 1
WAY N
I=1
I=2
I=3
I=4
I=5
I=6
TAG BEST CACHES
1232
INS
1234
ADDR_2
ADDR_1
INSERTER
1202
INDEX(ADDR_1)
NO TRT FOUND
TAG(ADDR_1)
TAG BEST DETERMINATOR
TRT FOUND
1208
1206
TAG GENERATOR
1204
INDEX GENERATOR
PREDICTION GENERATOR
1226
PID
TRT
1212
1214
PREDICTION(S)

FIG. 12

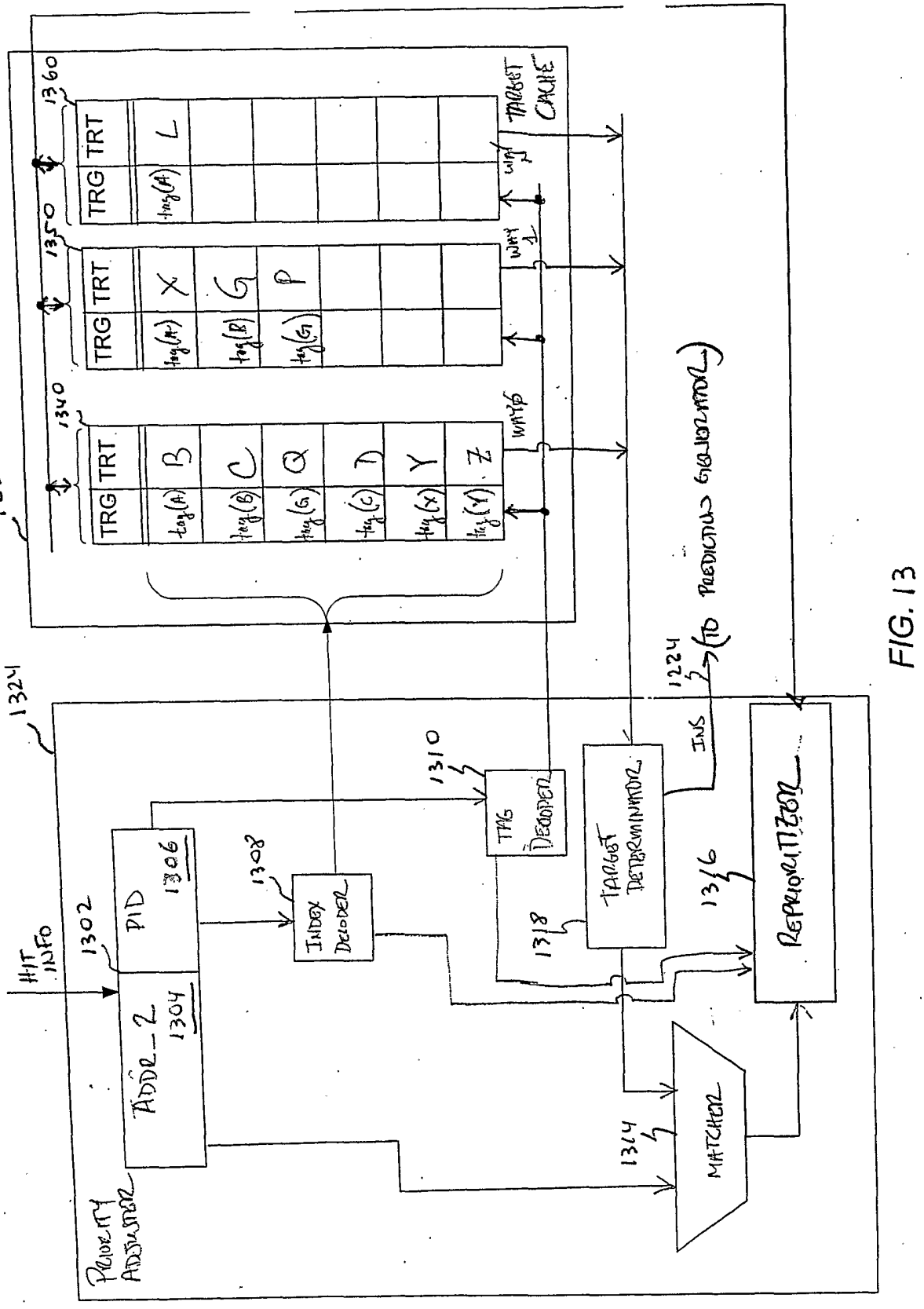

1230
1340
1350
1360
TRG
TRT
TARGET CACHE
1324
PRIORITY ADJUSTER
ADDR_2
1304
PID
1306
1302
HIT INFO
INDEX DECODER
1308
TAG DECODER
1310
TARGET DETERMINATOR
1318
INS
1224
(TO PREDICTING CIRCUITRY)
REPRIORITIZER
1316
MATCHER
1314

FIG. 13

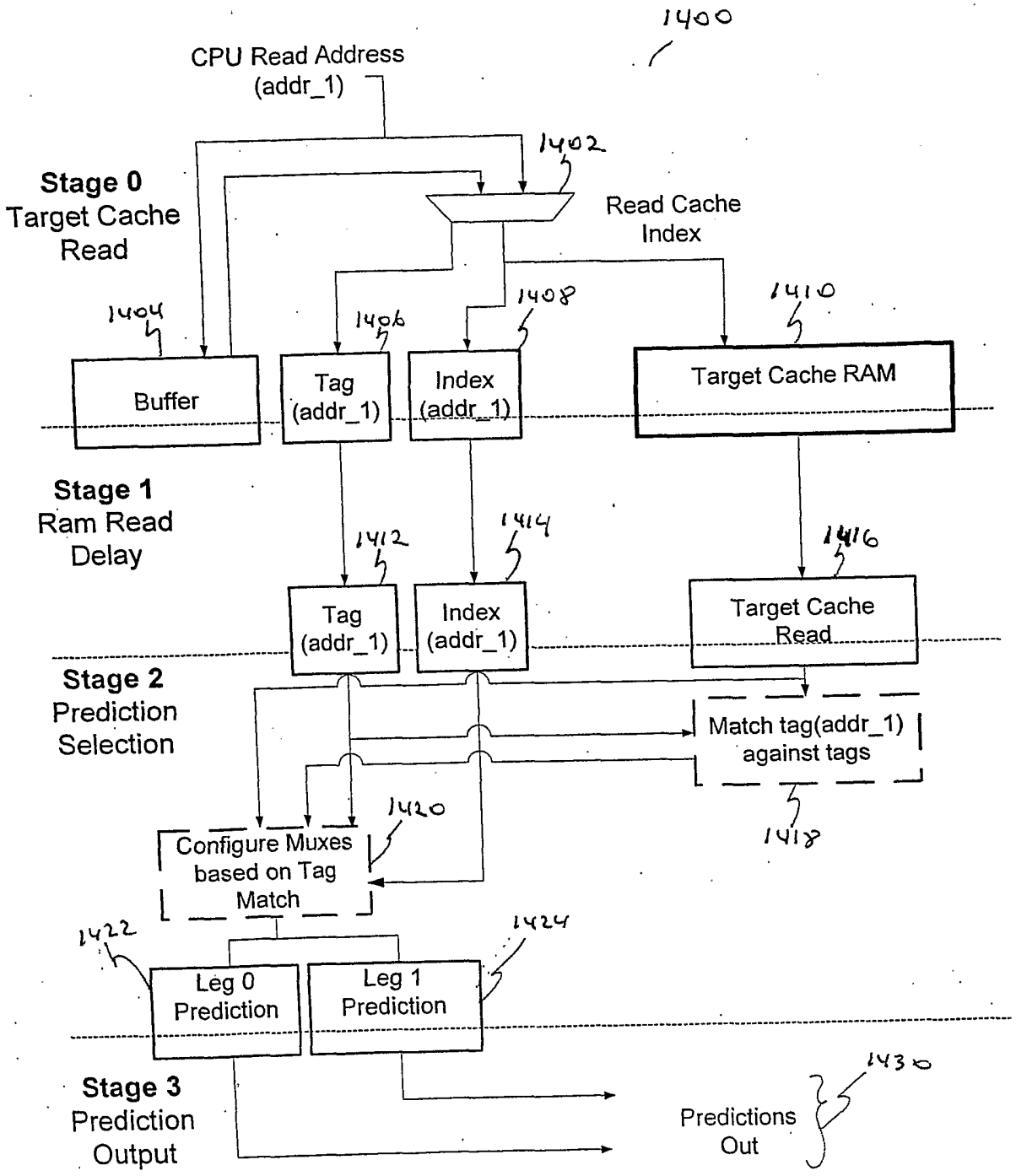

1400
CPU Read Address
(addr_1)
1402
Stage 0
Target Cache
Read
Read Cache
Index
1404
1406
1408
1410
Buffer
Tag
(addr_1)
Index
(addr_1)
Target Cache RAM
Stage 1
Ram Read
Delay
1412
1414
1416
Tag
(addr_1)
Index
(addr_1)
Target Cache
Read
Stage 2
Prediction
Selection
Match tag(addr_1)
against tags
1418
1420
Configure Muxes
based on Tag
Match
1422
1424
Leg 0
Prediction
Leg 1
Prediction
Stage 3
Prediction
Output
1430
Predictions
Out

FIG. 14

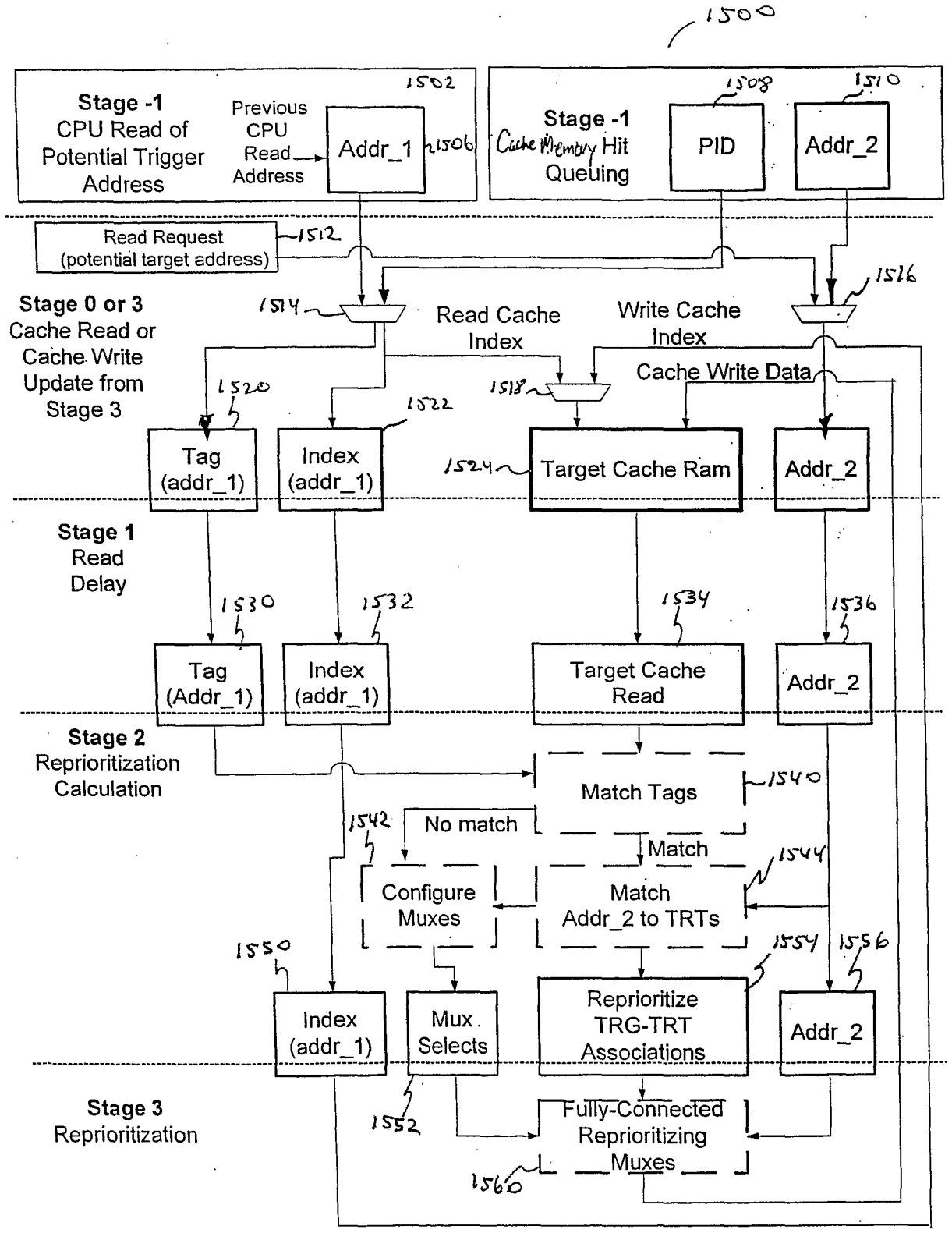

1500
Stage -1
CPU Read of
Potential Trigger
Address
Previous
CPU
Read
Address
1502
Addr_1
1506
Stage -1
Cache Memory Hit
Queuing
1508
PID
1510
Addr_2
Read Request
(potential target address)
1512
1516
Stage 0 or 3
Cache Read or
Cache Write
Update from
Stage 3
1514
Read Cache
Index
Write Cache
Index
Cache Write Data
1518
1520
1522
Tag
(addr_1)
Index
(addr_1)
1524
Target Cache Ram
Addr_2
Stage 1
Read
Delay
1530
1532
1534
1536
Tag
(Addr_1)
Index
(addr_1)
Target Cache
Read
Addr_2
Stage 2
Reprioritization
Calculation
Match Tags
1540
1542
No match
Match
1544
Configure
Muxes
Match
Addr_2 to TRTs
1550
1554
1556
Index
(addr_1)
Mux
Selects
Reprioritize
TRG-TRT
Associations
Addr_2
Stage 3
Reprioritization
1552
Fully-Connected
Reprioritizing
Muxes
1560

FIG. 15

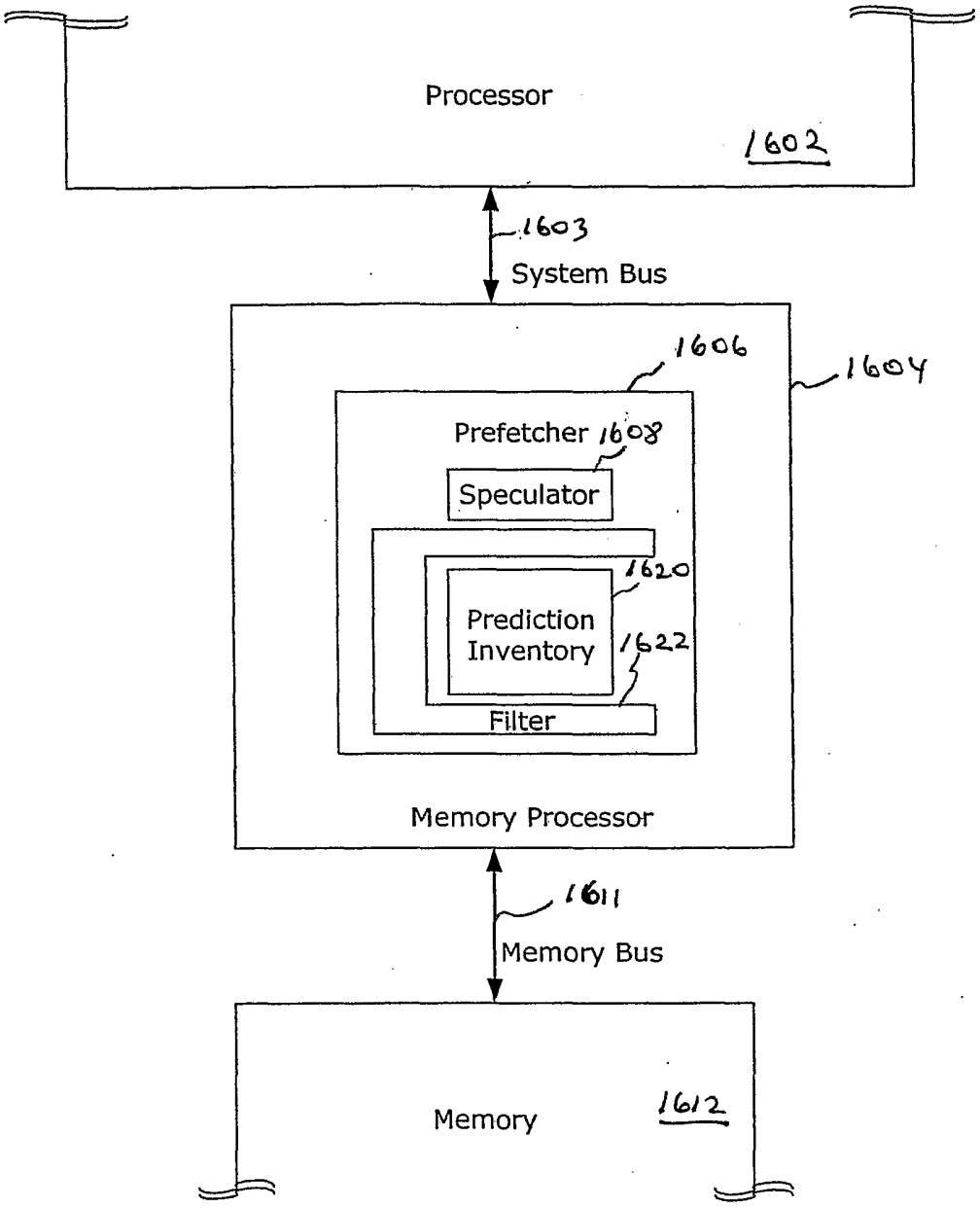

Processor
1602
1603
System Bus
1606
1604
Prefetcher 1608
Speculator
1620
Prediction Inventory
1622
Filter
Memory Processor
1611
Memory Bus
Memory
1612

FIG. 16

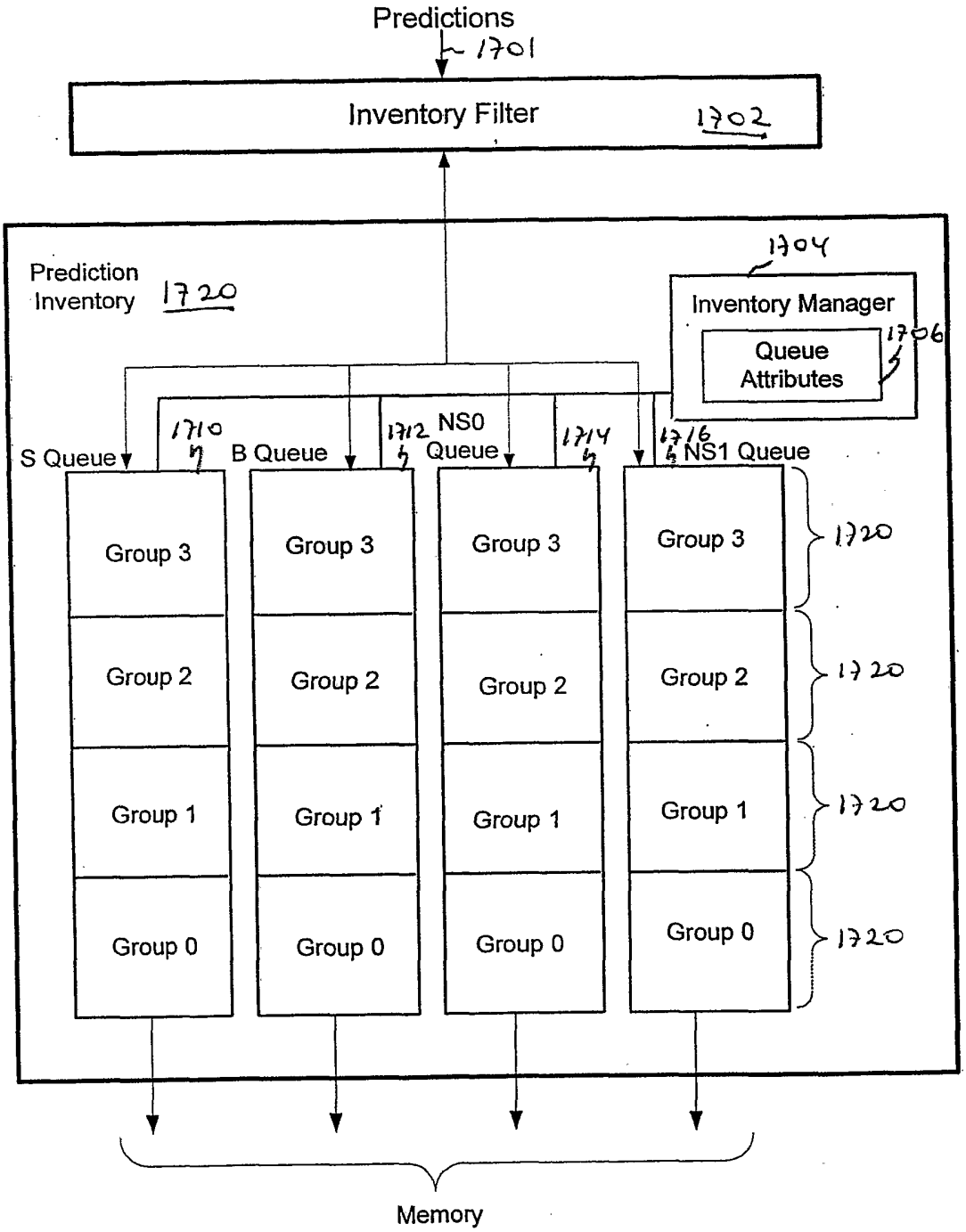

Predictions
1701
Inventory Filter
1702
Prediction Inventory
1720
1704
Inventory Manager
Queue Attributes
1706
S Queue
1710
B Queue
1712
NS0 Queue
1714
NS1 Queue
1716
Group 3
Group 2
Group 1
Group 0
1720
Memory

FIG. 17

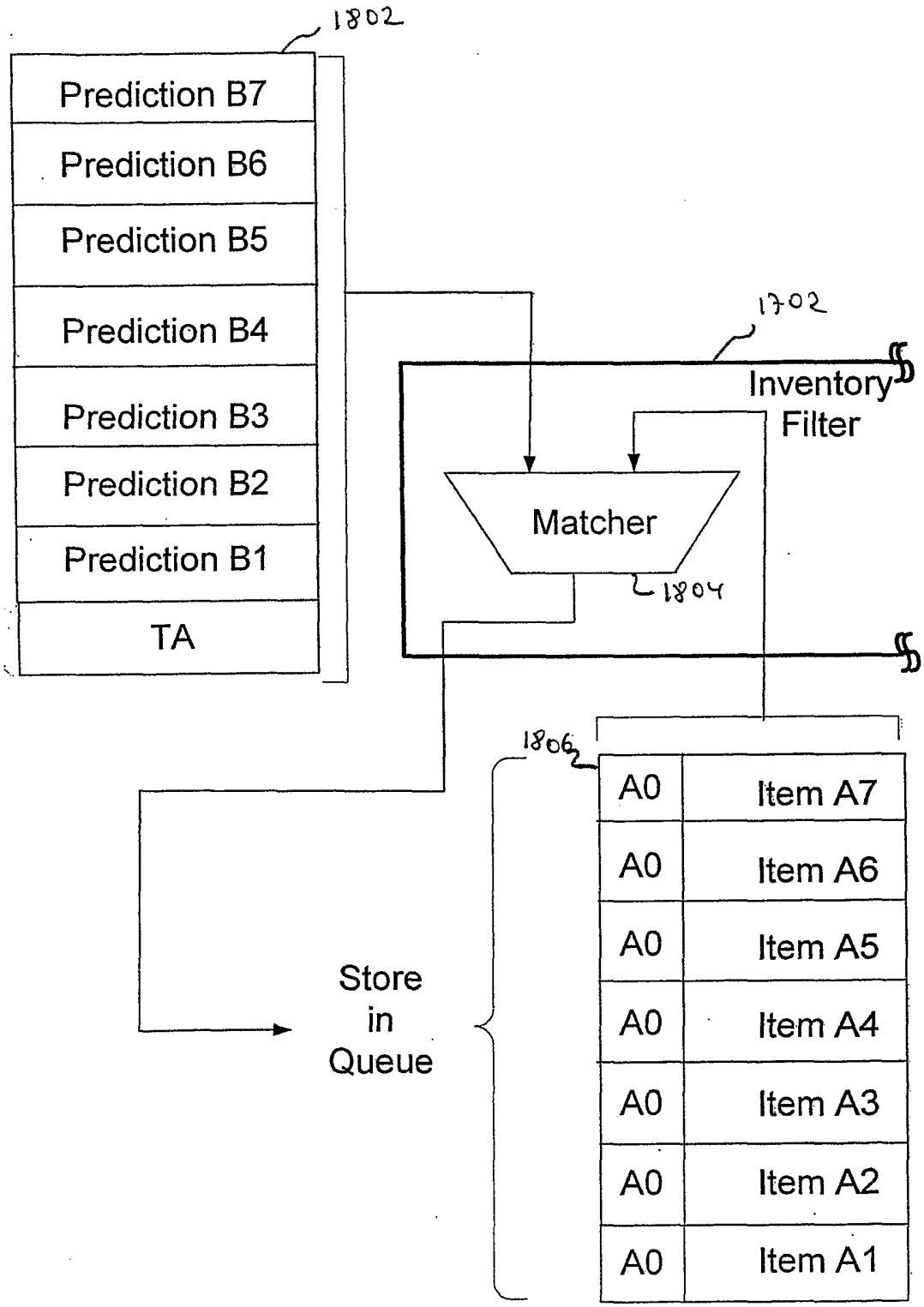

1802
Prediction B7
Prediction B6
Prediction B5
Prediction B4
Prediction B3
Prediction B2
Prediction B1
TA
1702
Inventory
Filter
Matcher
1804
1806
A0
Item A7
A0
Item A6
A0
Item A5
A0
Item A4
A0
Item A3
A0
Item A2
A0
Item A1
Store
in
Queue

FIG. 18

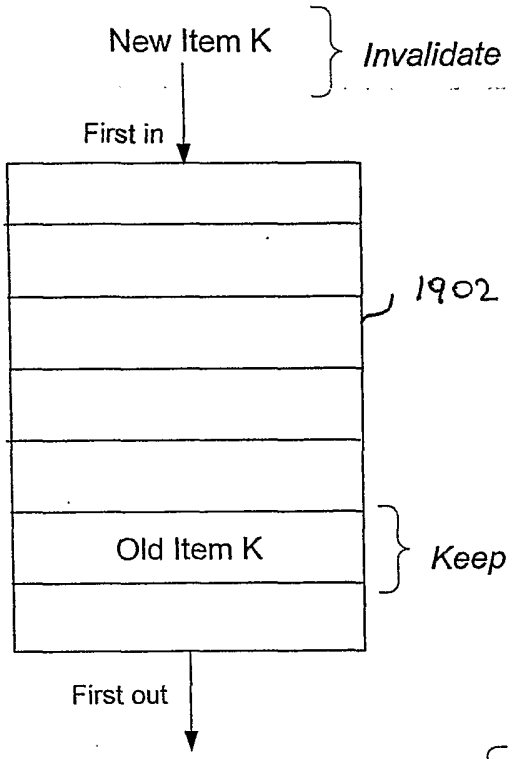


FIG. 19A

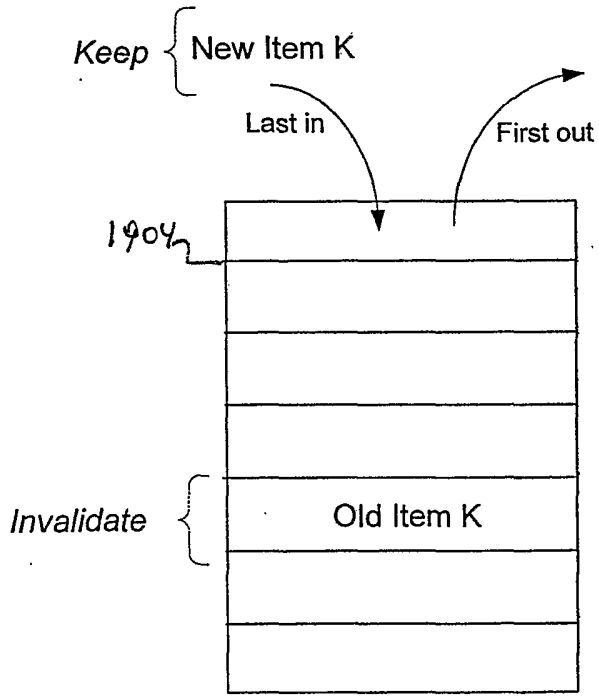


FIG. 19B

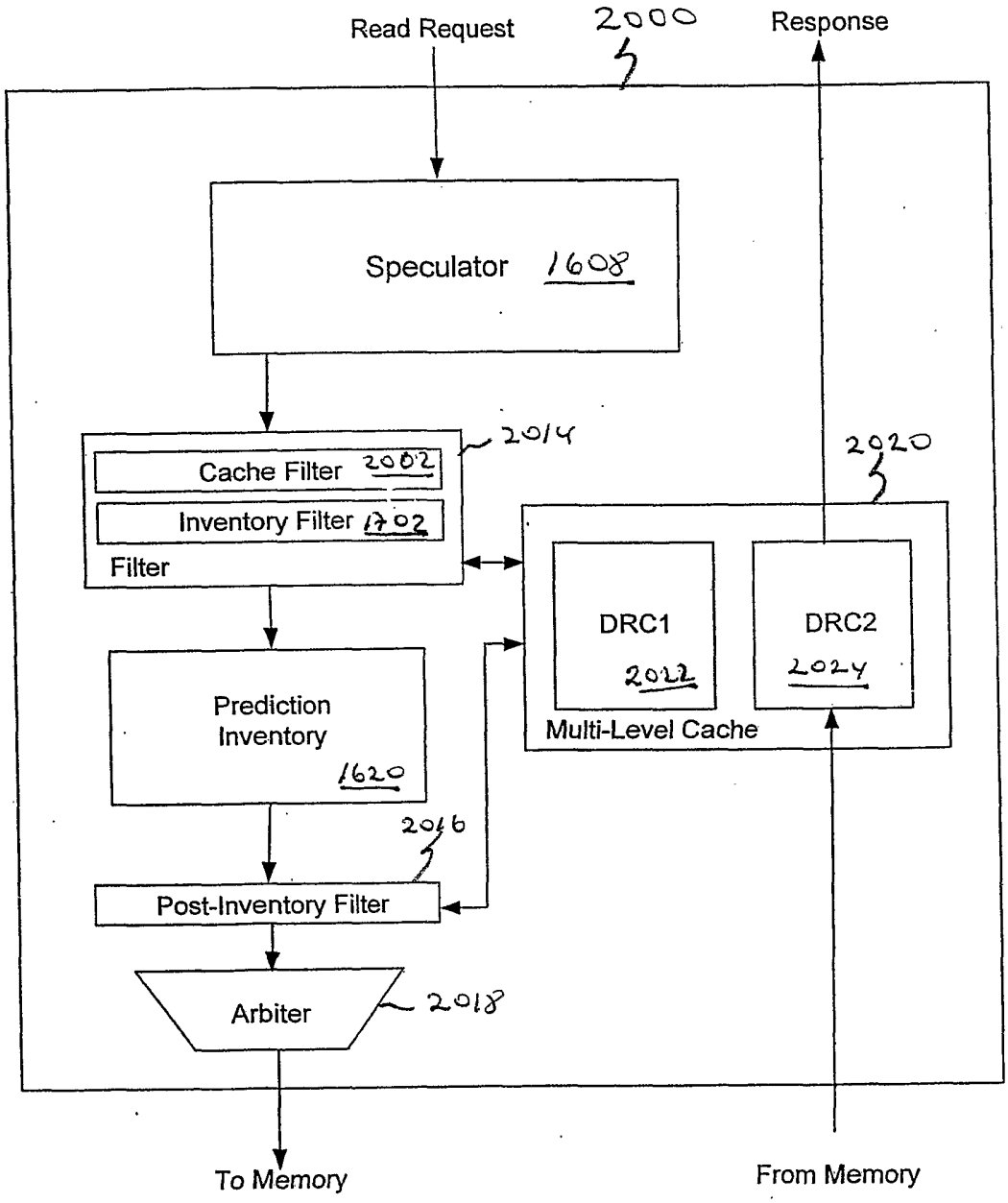

Read Request
2000
Response
Speculator 1608
2014
Cache Filter 2002
Inventory Filter 1702
Filter
2020
DRC1
2022
DRC2
2024
Multi-Level Cache
Prediction Inventory
1620
2016
Post-Inventory Filter
Arbiter
2018
To Memory
From Memory

FIG. 20

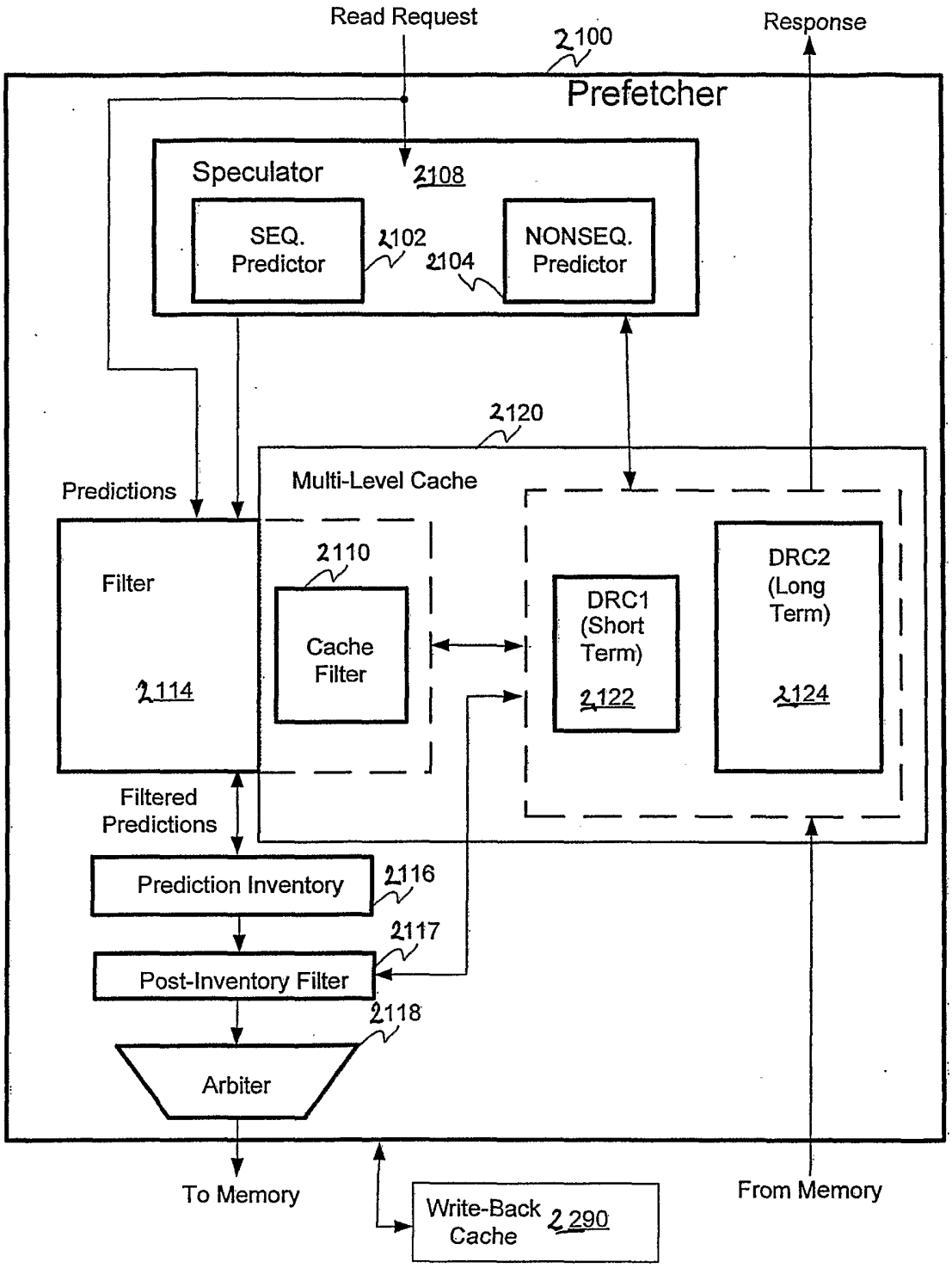

Read Request
2100
Response
Prefetcher
Speculator
2108
SEQ. Predictor
2102
2104
NONSEQ. Predictor
2120
Predictions
Multi-Level Cache
Filter
2114
2110
Cache Filter
DRC1 (Short Term)
2122
DRC2 (Long Term)
2124
Filtered Predictions
Prediction Inventory
2116
2117
Post-Inventory Filter
2118
Arbiter
To Memory
Write-Back Cache
2290
From Memory

FIG. 21

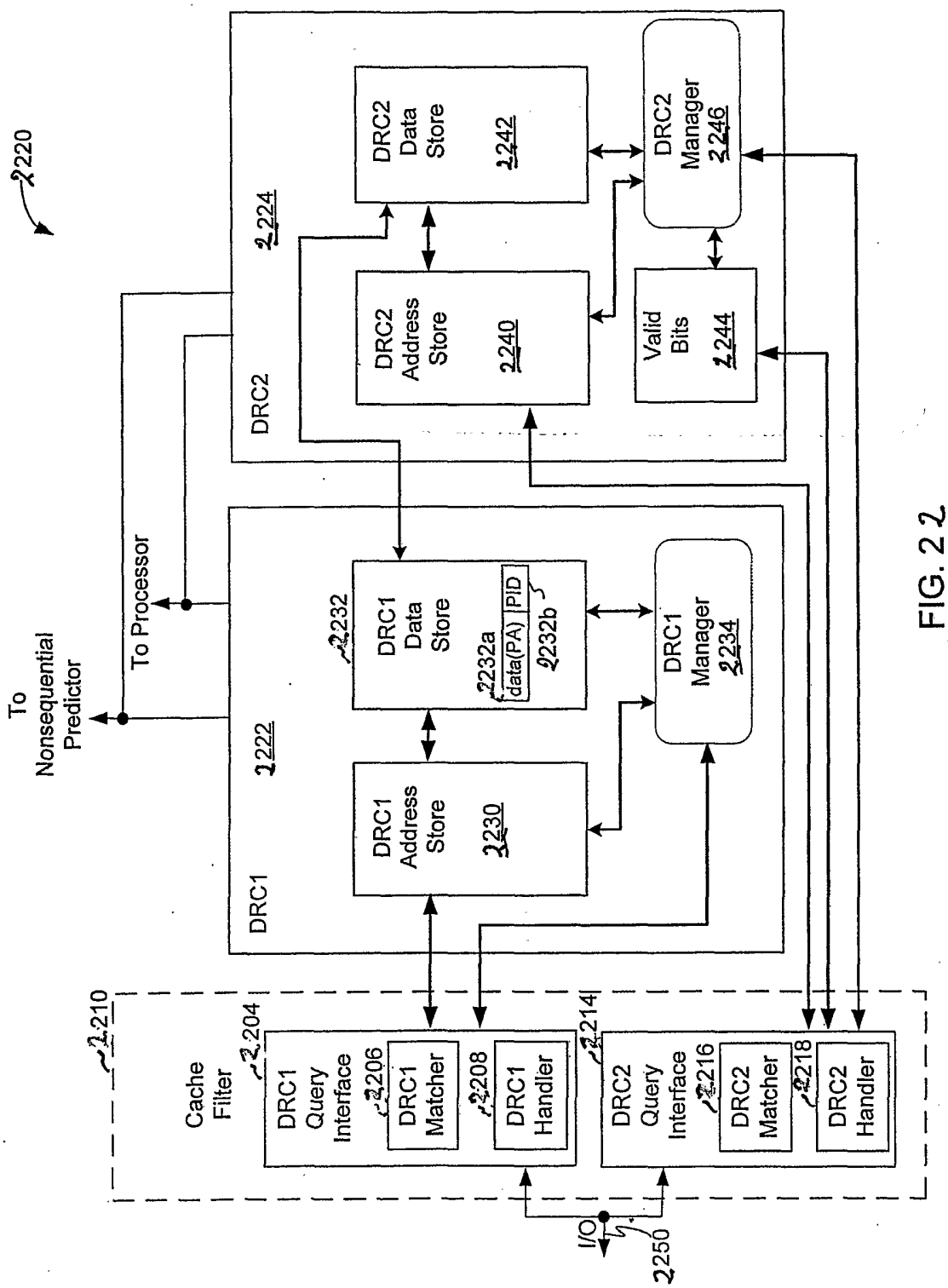

2220
2224
DRC2
DRC2 Data Store 2242
DRC2 Manager 2246
DRC2 Address Store 2240
Valid Bits 2244
To Processor
To Nonsequential Predictor
2222
DRC1
DRC1 Data Store 2232
2232a data(PA) PID 2232b
DRC1 Manager 2234
DRC1 Address Store 2230
2210
Cache Filter
2204
DRC1 Query Interface
2206
DRC1 Matcher
2208
DRC1 Handler
2214
DRC2 Query Interface
2216
DRC2 Matcher
2218
DRC2 Handler
I/O
2250

FIG. 22

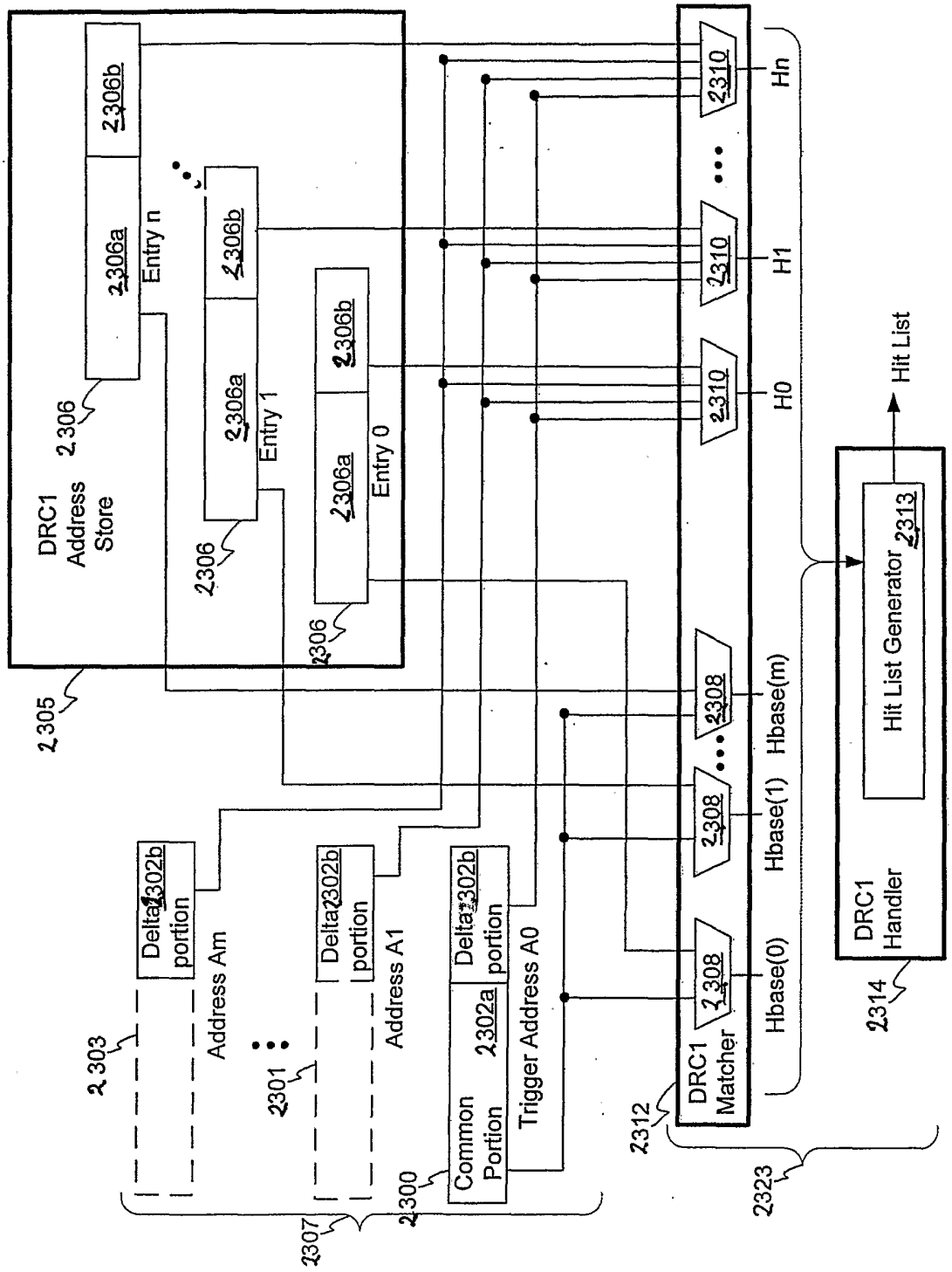

DRC1 Address Store
2306
2306a
2306b
Entry n
Entry 1
Entry 0
2305
2303
2301
2307
2300
Delta 2302b portion
Address Am
Address A1
Common Portion 2302a
Trigger Address A0
DRC1 Matcher
2312
2308
Hbase(0)
Hbase(1)
Hbase(m)
2310
H0
H1
Hn
2323
DRC1 Handler
2314
Hit List Generator
2313
Hit List

FIG.23A

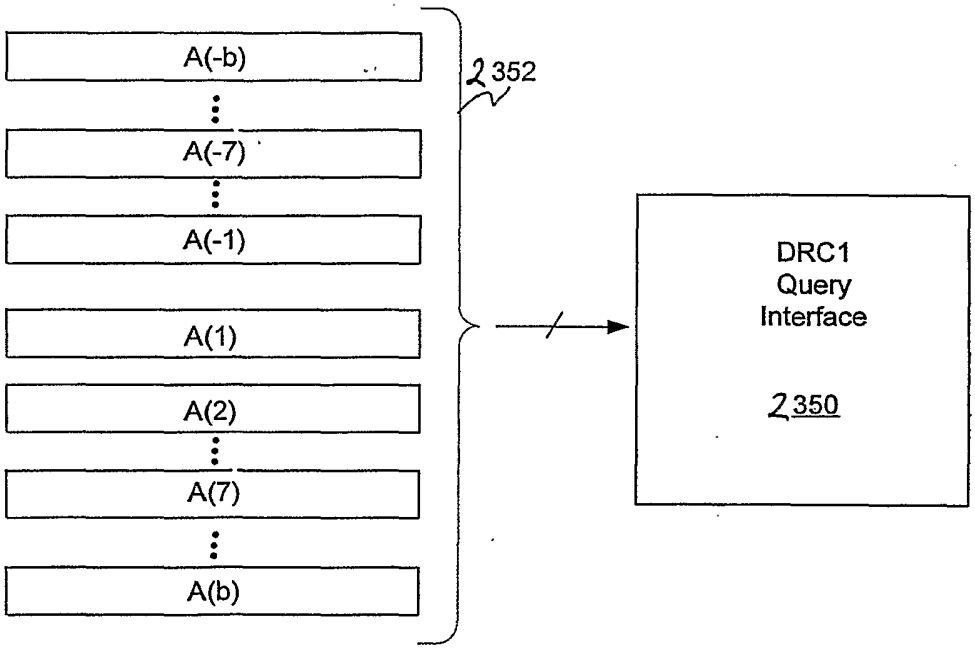


FIG.23B

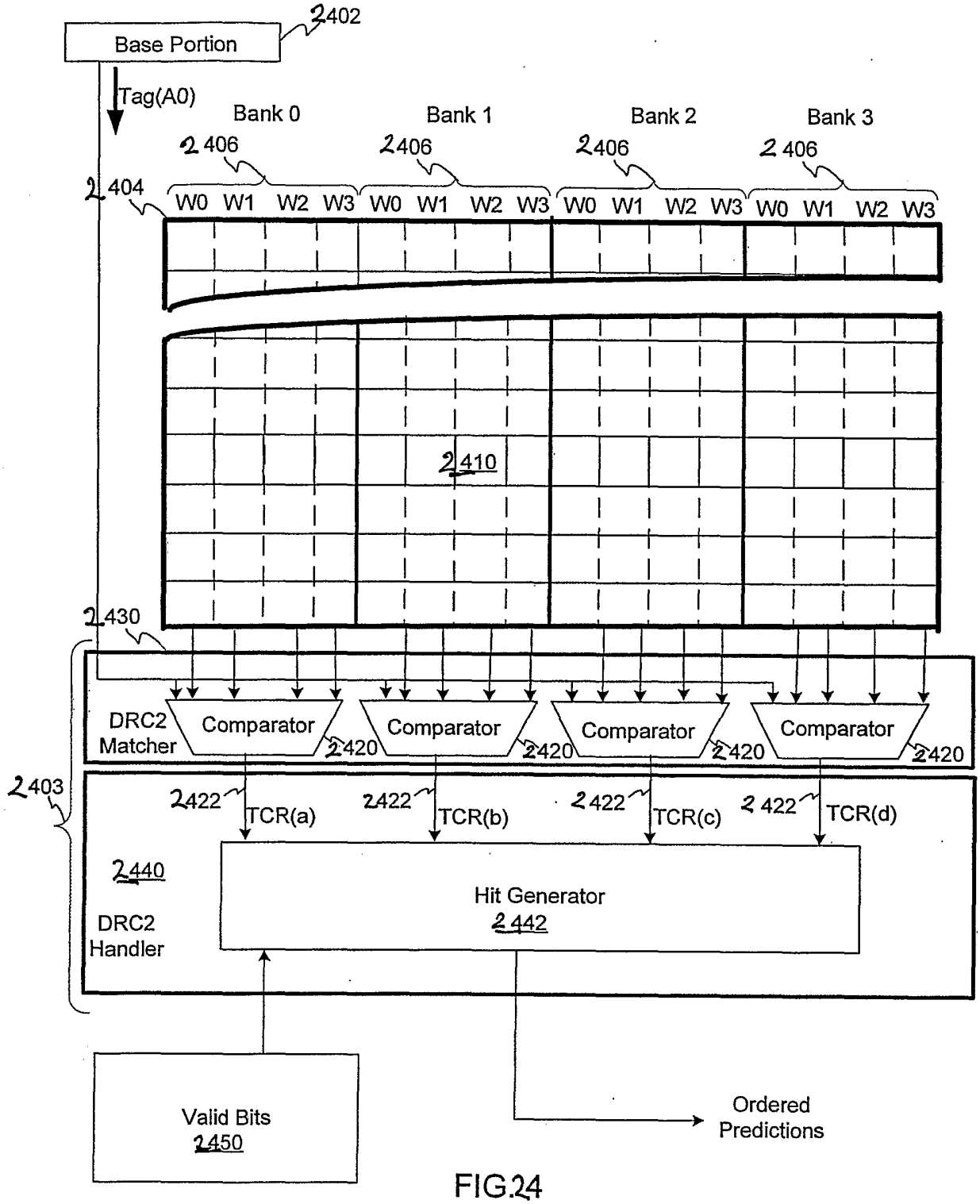

Base Portion
2402
Tag(A0)
Bank 0
Bank 1
Bank 2
Bank 3
2406
2404
W0 W1 W2 W3
2410
2430
DRC2 Matcher
Comparator
2420
2403
2422
TCR(a)
TCR(b)
TCR(c)
TCR(d)
2440
DRC2 Handler
Hit Generator
2442
Valid Bits
2450
Ordered Predictions

FIG.24

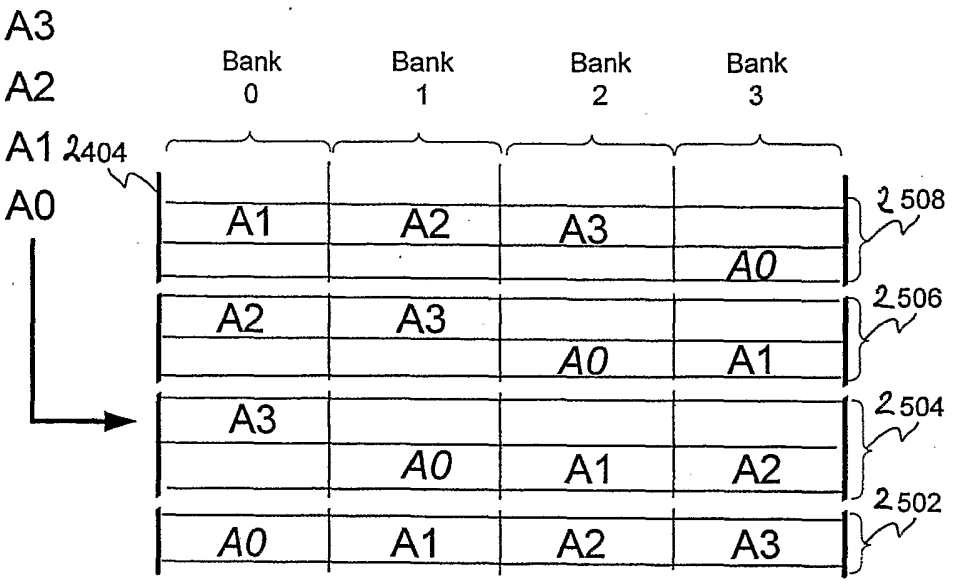


FIG.25A

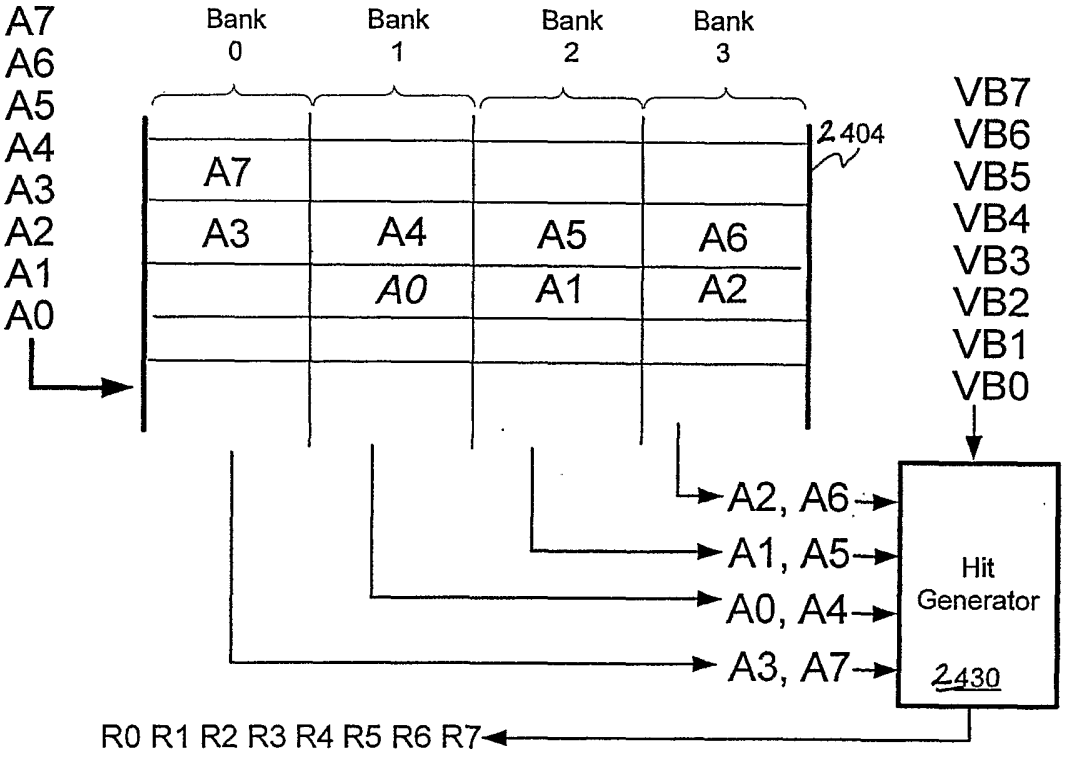


FIG.25B

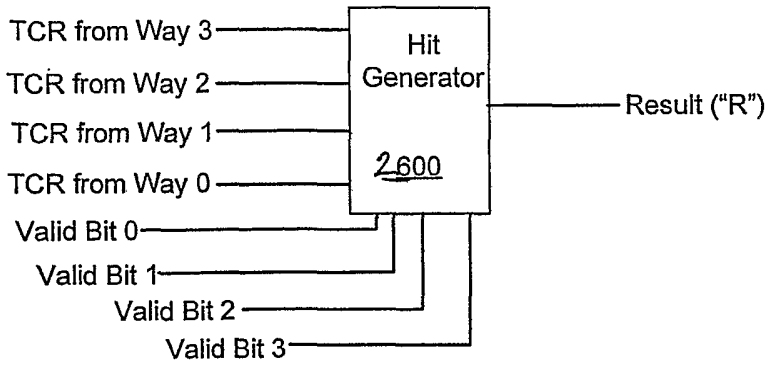

TCR from Way 3
TCR from Way 2
TCR from Way 1
TCR from Way 0
Valid Bit 0
Valid Bit 1
Valid Bit 2
Valid Bit 3
Hit Generator
2600
Result ("R")

FIG.26

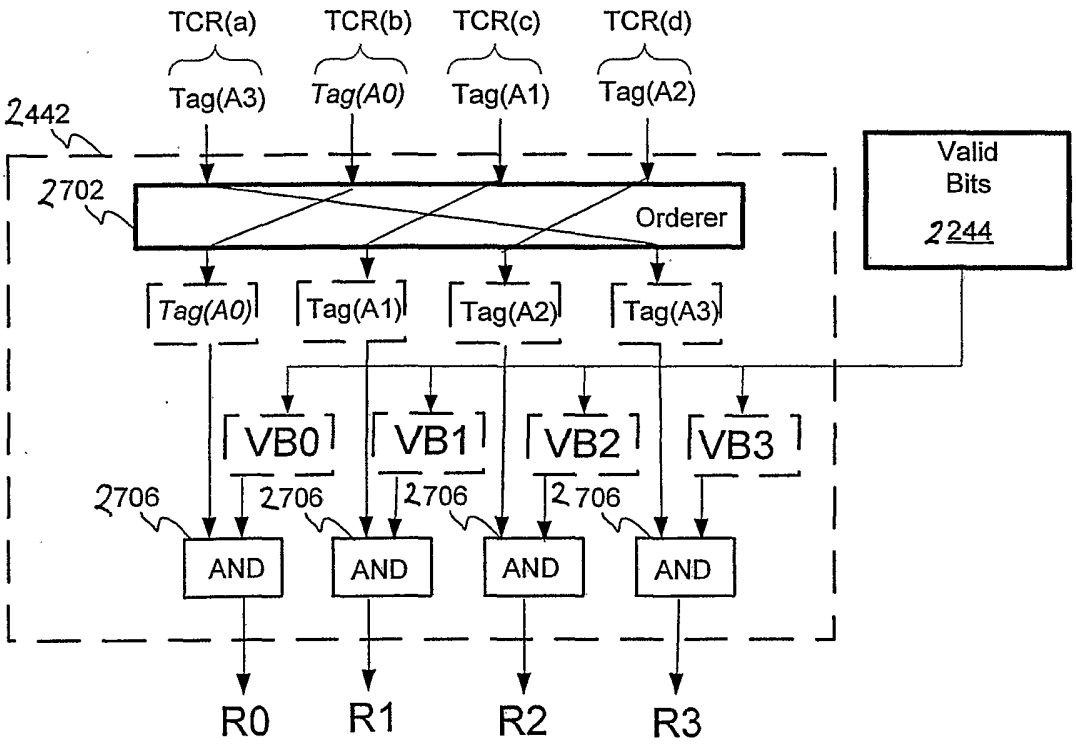

TCR(a)
TCR(b)
TCR(c)
TCR(d)
Tag(A3)
Tag(A0)
Tag(A1)
Tag(A2)
2442
2702
Orderer
Valid Bits
2244
Tag(A0)
Tag(A1)
Tag(A2)
Tag(A3)
VB0
VB1
VB2
VB3
2706
AND
R0
R1
R2
R3

FIG.27

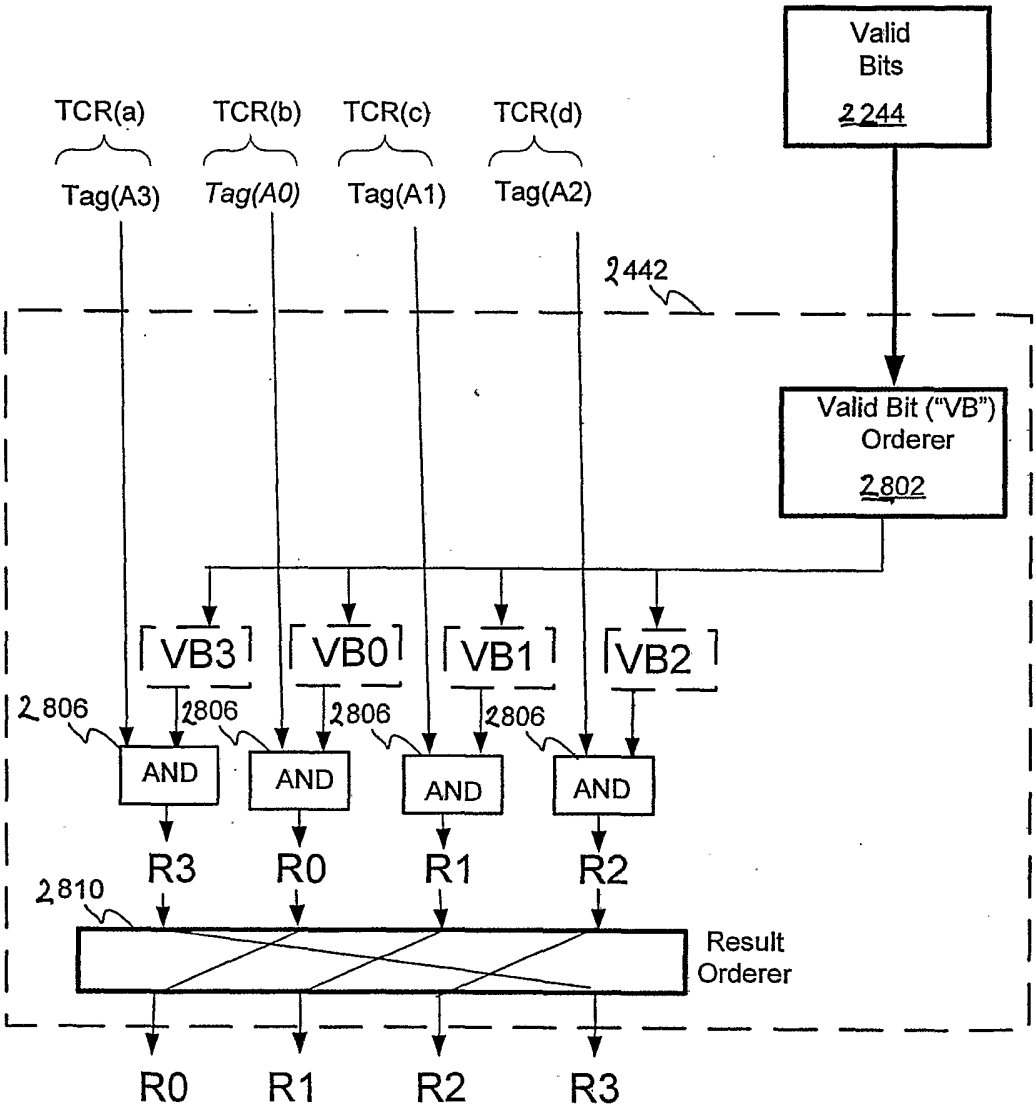

Valid Bits
2244
TCR(a)
TCR(b)
TCR(c)
TCR(d)
Tag(A3)
Tag(A0)
Tag(A1)
Tag(A2)
2442
Valid Bit ("VB") Orderer
2802
VB3
VB0
VB1
VB2
2806
2806
2806
2806
AND
AND
AND
AND
R3
R0
R1
R2
2810
Result Orderer
R0
R1
R2
R3

FIG.28