US 20040054867A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2004/0054867 A1**

Stravers et al. (43) **Pub. Date: Mar. 18, 2004**

(54) **TRANSLATION LOOKASIDE BUFFER**

(76) Inventors: **Paulus Stravers**, Sunnyvale, CA (US); **Jan-Willem van de Waerdt**, Sunnyvale, CA (US)

Correspondence Address:
Philips Electronic North America Corporation
Intellectual Property and Standards
1000 West Maude Avenue
Sunnyvale, CA 94085 (US)

(21) Appl. No.: **10/242,785**

(22) Filed: **Sep. 13, 2002**

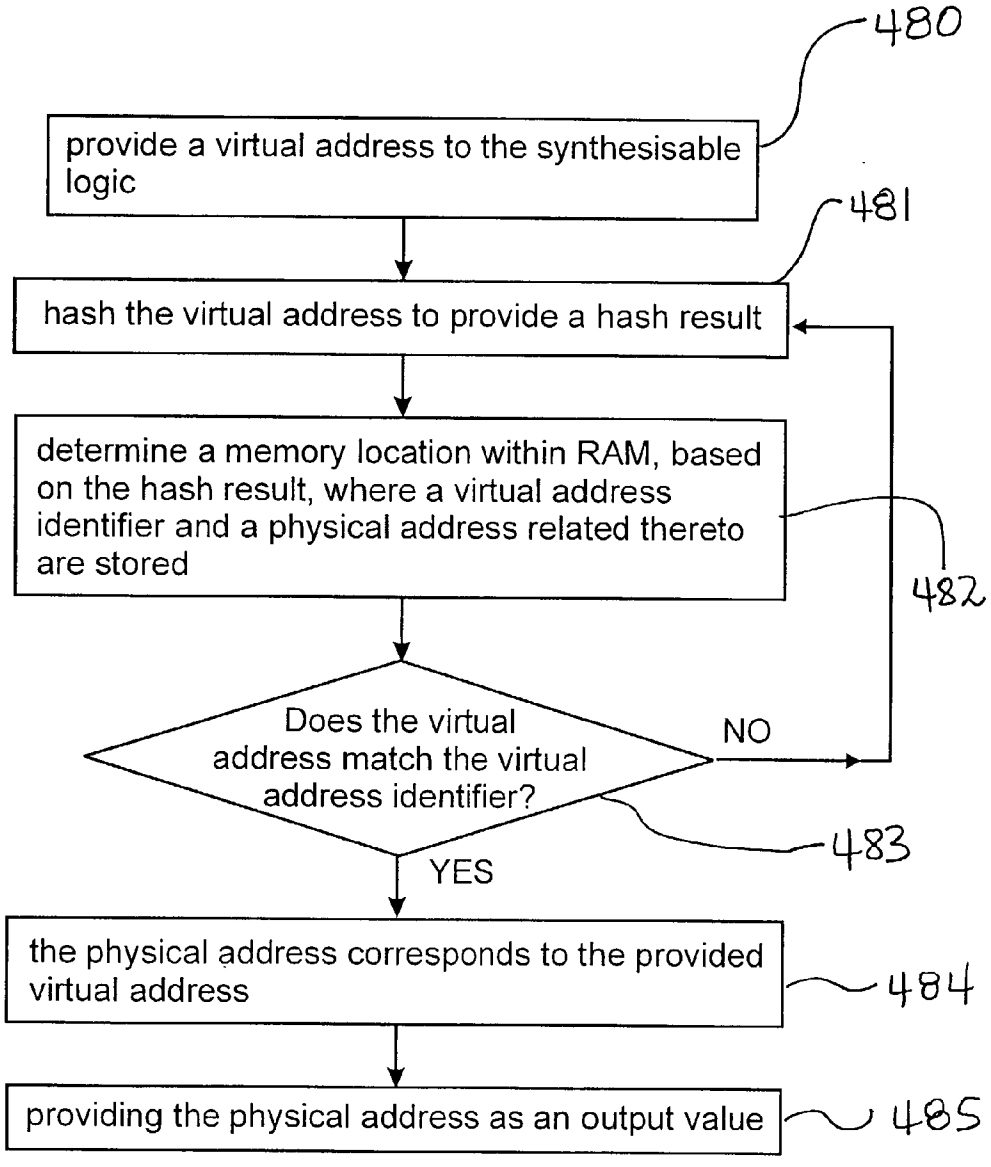
Publication Classification

(51) **Int. Cl.**[7] **G06F 12/10**; G06F 12/00

(52) **U.S. Cl.** **711/207**; 711/216

(57) **ABSTRACT**

A translation lookaside buffer (TLB) is disclosed formed using RAM and synthesisable logic circuits. The TLB provides logic within the synthesisable logic for pairing down a number of memory locations that must be searched to find a translation to a physical address from a received virtual address. The logic provides a hashing circuit for hashing the received virtual address and uses the hashed virtual address to index the RAM to locate a line within the RAM that provides the translation.



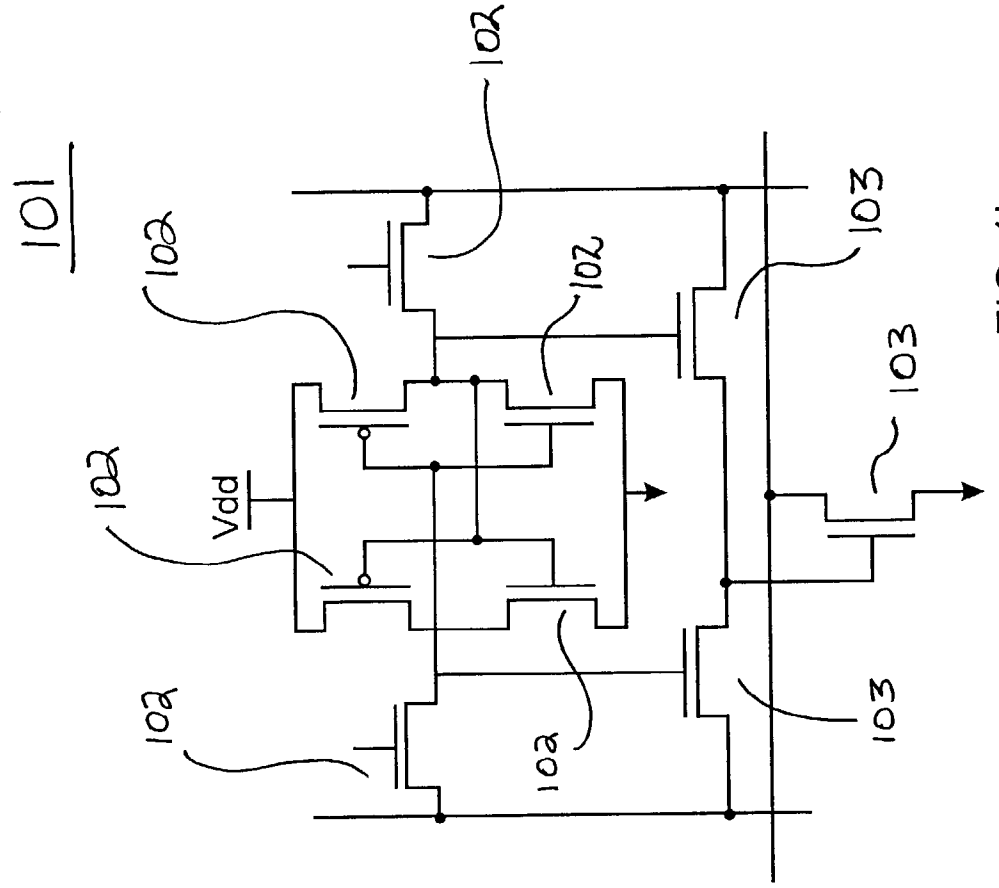


FIG. 1b
(PRIOR ART)

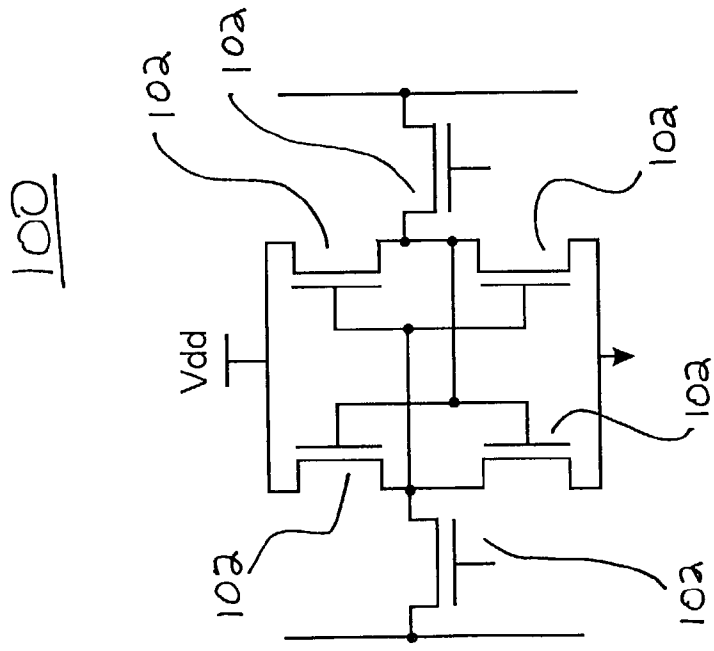


FIG. 1a
(PRIOR ART)

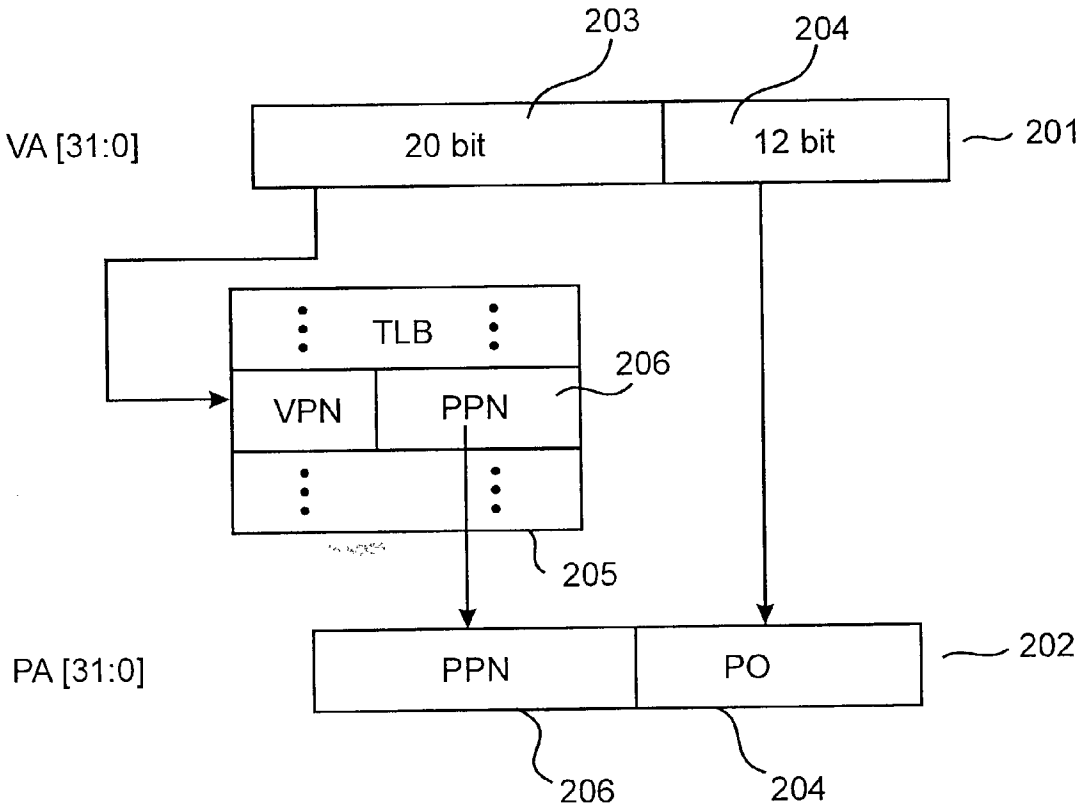


FIG. 2
(PRIOR ART)

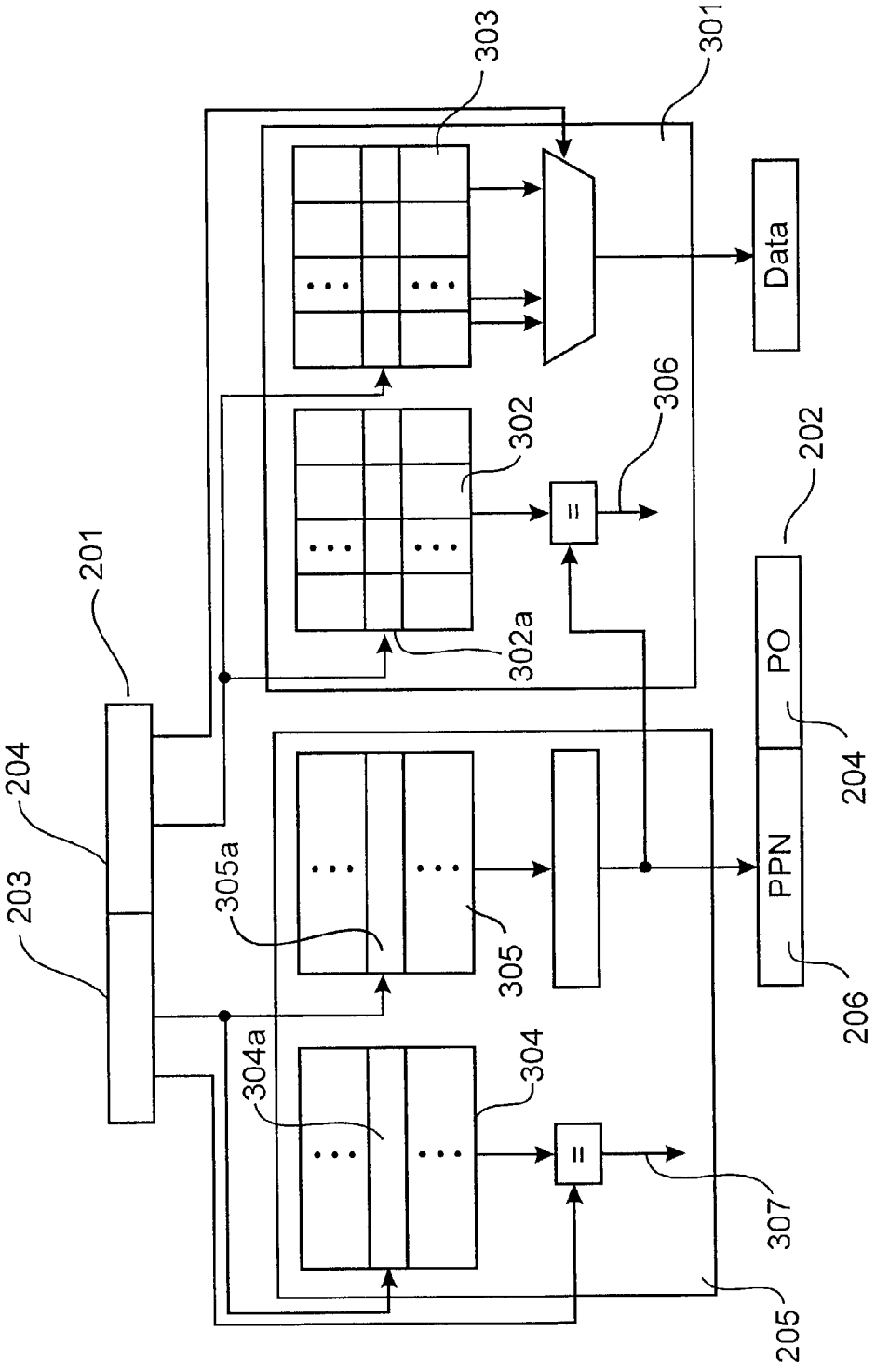


Fig. 3
(PRIOR ART)

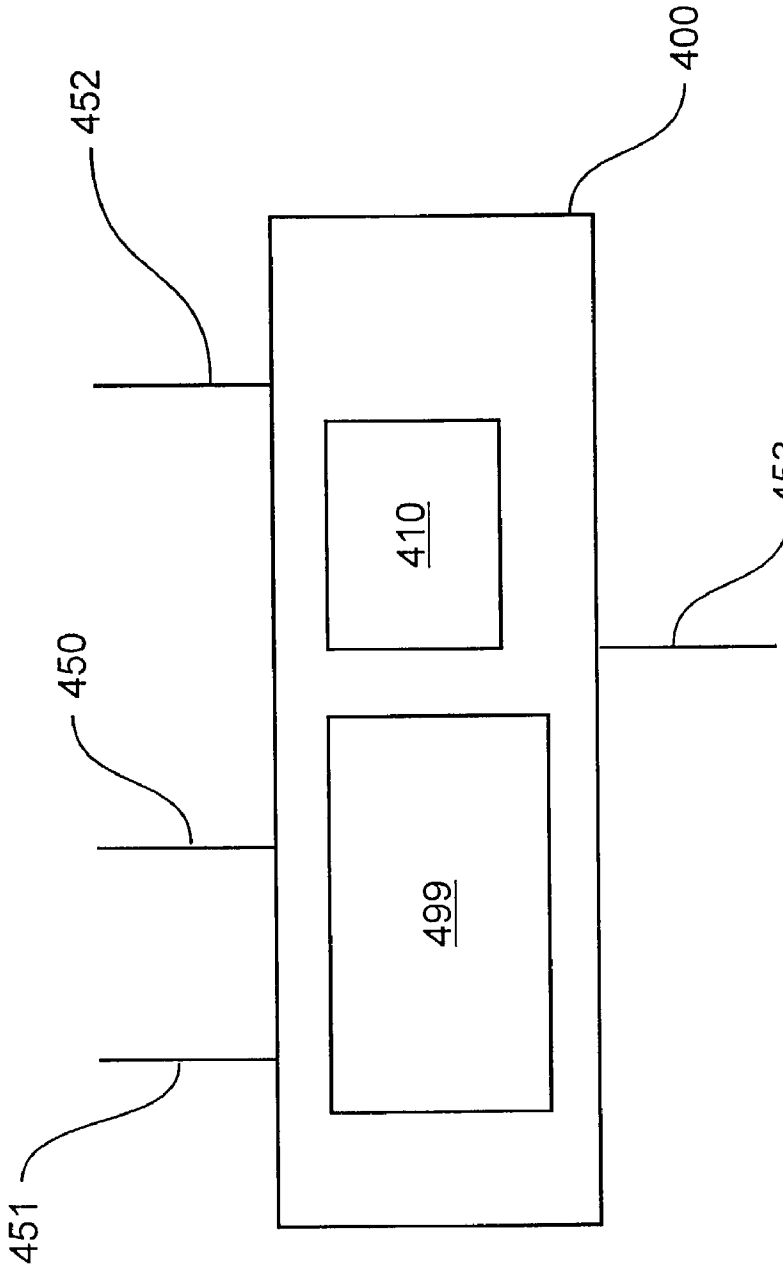


Fig. 4a

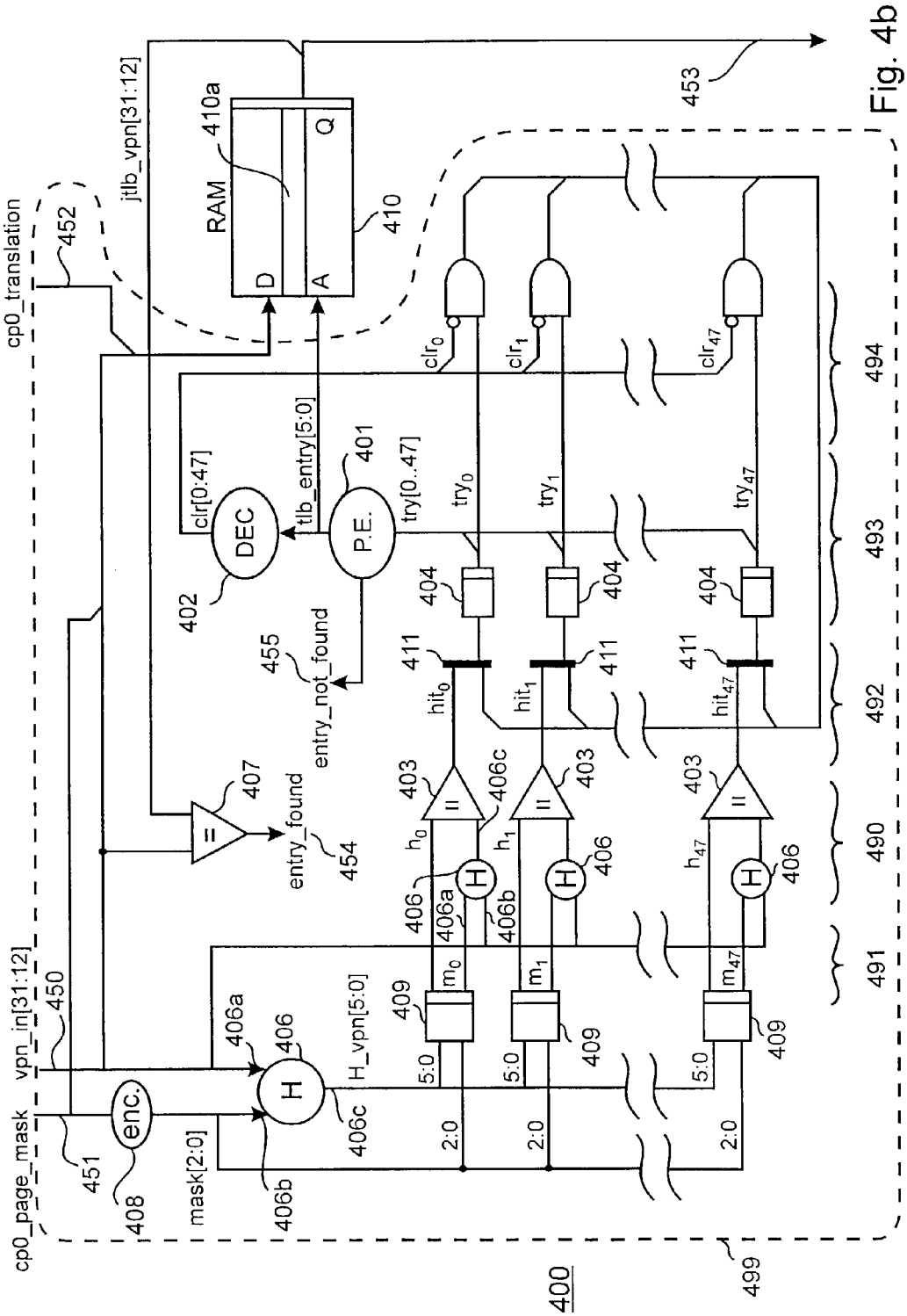

cp0_translation
452
jtlb_vpn[31:12]
RAM
410a
D
A
Q
410
453
clr0
clr1
clr47
494
clr[0:47]
DEC
402
tlb_entry[5:0]
P.E.
401
try[0..47]
try0
try1
try47
493
404
455
entry_not_found
411
hit0
hit1
hit47
492
407
=
entry_found
454
403
406c
h0
h1
h47
406
406a
406b
491
490
vpn_in[31:12]
450
406a
406
409
m0
m1
m47
H_vpn[5:0]
5:0
2:0
cp0_page_mask
451
enc.
408
mask[2:0]
406b
406c
400
499

Fig. 4b

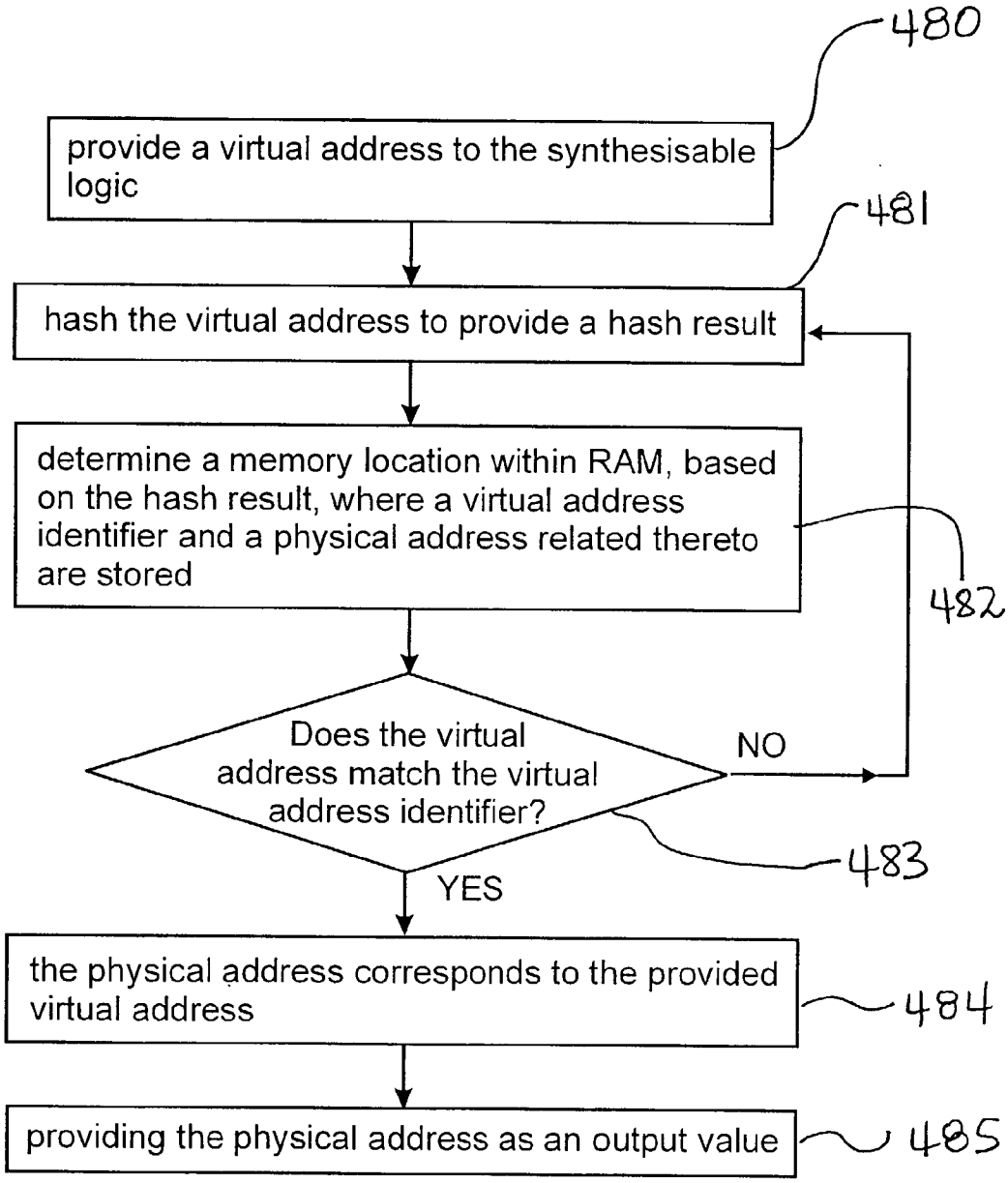

480
provide a virtual address to the synthesisable logic
481
hash the virtual address to provide a hash result
determine a memory location within RAM, based on the hash result, where a virtual address identifier and a physical address related thereto are stored
482
Does the virtual address match the virtual address identifier?
NO
483
YES
the physical address corresponds to the provided virtual address
484
providing the physical address as an output value
485

Fig. 4c

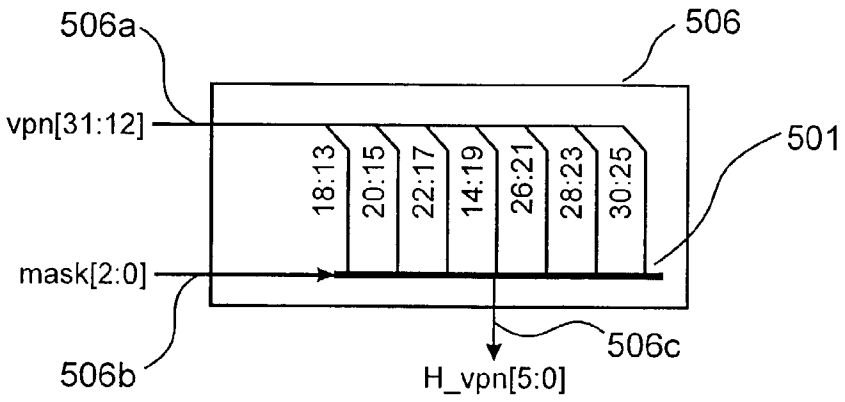

506a
506
vpn[31:12]
501
18:13
20:15
22:17
14:19
26:21
28:23
30:25
mask[2:0]
506c
506b
H_vpn[5:0]

Fig. 5

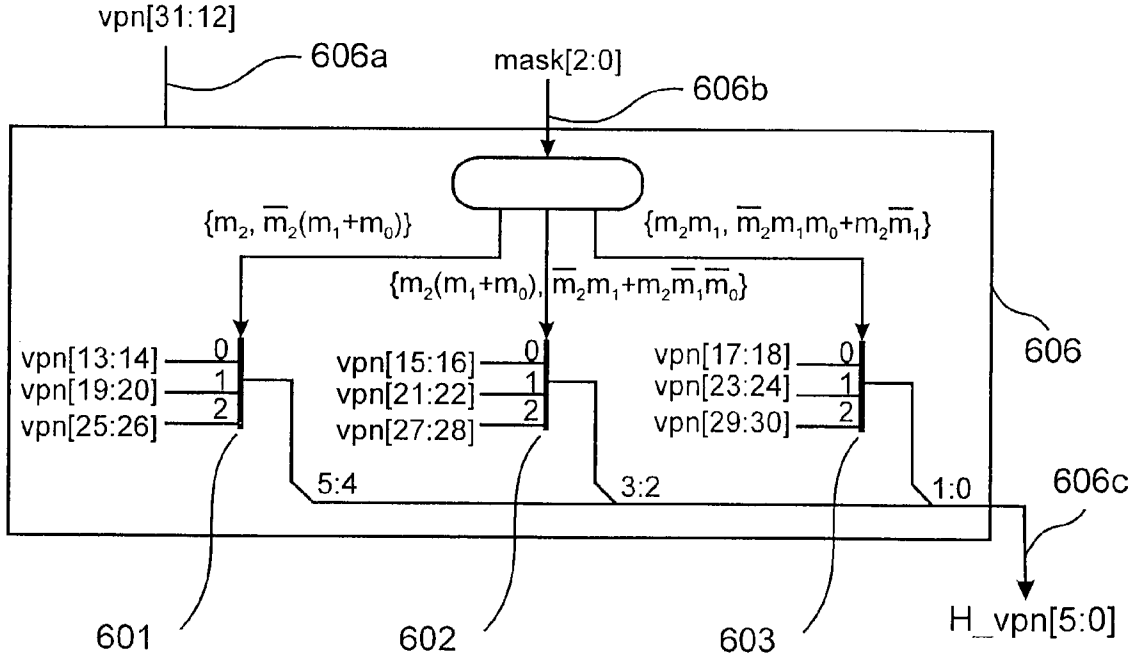

vpn[31:12]
606a
mask[2:0]
606b
{m2, m̄2(m1+m0)}
{m2m1, m̄2m1m0+m2m̄1}
{m2(m1+m0), m̄2m1+m2m̄1m̄0}
606
vpn[13:14]
vpn[19:20]
vpn[25:26]
vpn[15:16]
vpn[21:22]
vpn[27:28]
vpn[17:18]
vpn[23:24]
vpn[29:30]
0
1
2
5:4
3:2
1:0
606c
601
602
603
H_vpn[5:0]

Fig. 6

TRANSLATION LOOKASIDE BUFFER

FIELD OF THE INVENTION

[0001] This invention relates to the area of translation lookaside buffers and more specifically to translation lookaside buffer architectures for rapid design cycles.

BACKGROUND OF THE INVENTION

[0002] Modern microprocessor systems typically utilize virtual addressing. Virtual addressing enables the system to effectively create a virtual memory space larger than an actual physical memory space. The process of breaking up the actual physical memory space into the virtual memory space is termed paging. Paging breaks up a linear address space of the physical memory space into fixed blocks called pages. Pages allow a large linear address space to be implemented with a smaller physical main memory plus cheap background memory. This configuration is referred to as "virtual memory." Paging allows virtual memory to be implemented by managing memory in pages that are swapped to and from the background memory. Paging offers additional advantages, including reduced main memory fragmentation, selective memory write policies for different pages, and varying memory protection schemes for different pages. The presence of a paging mechanism is typically transparent to the application program.

[0003] The size of a page is a tradeoff between flexibility and performance. A small page size allows finer control over the virtual memory system but it increases the overhead from paging activity. Therefore many CPUs support a mix of page sizes, e.g. a particular MIPS implementation supports any mix of 4 kB, 16 kB, 64 kB, 256 kB, 1 MB, 4 MB and 16 MB pages.

[0004] A processor is then able to advantageously operate in the virtual address space using virtual addresses. Frequently, however, these virtual addresses must be translated into physical addresses—actual memory locations. One way of accomplishing this translation of virtual addresses into physical addresses is a use of translation tables that are regularly accessed and stored in main memory. Translation tables are stored in main memory because they are typically large in size. Unfortunately, regularly accessing of translation tables stored in main memory tends to slow overall system performance.

[0005] Modern microprocessor systems often use a translation lookaside buffer (TLB) to store or cache recently generated virtual to physical address translations in order to avoid the need to regularly access translation tables in main memory to accomplish address translation. A TLB is a special type of cache memory. As with other types of cache memories, a TLB is typically comprised of a relatively small amount of memory storage specially designed to be quickly accessible. A TLB typically incorporates both a tag array and a data array, as are provided in cache memories. Within the tag array, each tag line stores a virtual address. This tag line is then associated with a corresponding data line in the data array in which is stored a physical address translation for the virtual address. Thus, prior to seeking a translation of a virtual address from translation tables in main memory, a processor first refers to the TLB to determine whether the physical address translation of the virtual address is presently stored in the TLB. In the event that the virtual address and corresponding physical address are stored in the TLB, the TLB provides the corresponding physical address at an output port thereof, and a time and resource-consuming access of main memory is avoided. To facilitate operation of the TLB and to reduce indexing requirements therefore, a content addressable memory (CAM) is typically provided within the TLB. CAMs are parallel pattern matching circuits. In a matching mode of operation the CAM permits searching of all of its data in parallel to find a match.

[0006] Unfortunately, traditional TLBs require custom circuit design techniques to implement a CAM. Using custom circuit designs is not advantageous since each TLB and associated CAM requires a significant design effort in order to implement same in a processor system design. Of course, when a processor is absent CAM circuitry, signals from the processor propagate off chip to the CAM, thereby incurring delays.

[0007] It is therefore an object of this invention to provide a CAM architecture formed of traditional synthesisable circuit blocks.

SUMMARY OF THE INVENTION

[0008] In accordance with the invention there is provided a translation lookaside buffer (TLB) comprising: at least an input port for receiving a portion of a virtual address;

[0009] a random access memory; a set of registers; and, synthesisable logic for determining a hash value from the received portion of the virtual address and for comparing the hash value to a stored hash value within the set of registers to determine a potential that a physical address associated with the virtual address is stored within a line within the random access memory and associated with a register, from the set of registers, within which the hash value is stored.

[0010] In accordance with an aspect of the invention there is provided a translation lookaside buffer comprising: a random access memory; a first register associated with a line in the memory; and, a hashing circuit for receiving a virtual address other than a virtual address for which a translation is presently stored in the memory, for determining a hash value and for storing the hash value in the first register; and the hashing circuit for storing the virtual address and a translation therefor in the line in memory.

[0011] In accordance with yet another aspect of the invention there is provided a translation lookaside buffer comprising: RAM; and, synthesisable logic for determining from a virtual address at least one potential address within the RAM in fixed relation to which to search for a physical address associated with the virtual address, the at least one potential address being other than the one and only known address within the RAM in fixed relation to which the physical address associated with the virtual address is stored.

[0012] In accordance with yet another aspect of the invention there is provided a method of performing a virtual address lookup function for a translation lookaside buffer including RAM and synthesisable logic including the steps of: providing a virtual address to the synthesisable logic; hashing the provided virtual address to provide a hash result;

[0013] based on the hash result determining a memory location within the RAM relative to which is stored a virtual address identifier and a physical address related thereto;

[0014] comparing the virtual address to the virtual address identifier to determine if the physical address corresponds to the provided virtual address; and, when the physical address corresponds to the provided virtual address, providing the physical address as an output value.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] The invention will now be described with reference to the drawings in which:

[0016] **FIG. 1***a* illustrates a prior art transistor implementation of a SRAM circuit;

[0017] **FIG. 1***b* illustrates a prior art transistor implementation of a CAM circuit;

[0018] **FIG. 2** illustrates a prior art translation process from a virtual address (VA) to a physical address (PA);

[0019] **FIG. 3** illustrates a prior art translation from a VA to a PA when performed in conjunction with a direct mapped cache memory;

[0020] **FIG. 4***a* generally illustrates a translation lookaside buffer formed using synthesisable logic components and a random access memory;

[0021] **FIG. 4***b* illustrates a translation lookaside buffer in more detail formed from synthesisable logic components;

[0022] **FIG. 4***c* outlines the steps taken for operation of the TLB;

[0023] **FIG. 5** illustrates a hashing circuit in more detail; and,

[0024] **FIG. 6** illustrates a variation of the hashing circuit shown in **FIG. 5**.

DETAILED DESCRIPTION OF THE INVENTION

[0025] CAM circuits include storage circuits similar in structure to SRAM circuits. However, CAM circuits also include search circuitry offering an added benefit of a parallel search mode of operation, thus enabling searching of the contents of the CAM in parallel using hardware. When searching the CAM for a particular data value, the CAM provides a match signal upon finding a match for that data value within the CAM. A main difference between CAM and SRAM is that in a CAM, data is presented to the CAM representative of a virtual address and an address relating to the data is returned, whereas in a SRAM, an address is provided to the SRAM and data stored at that address is returned.

[0026] The cells of the CAM are arranged so that each row of cells holds a memory address and that row of cells is connected by a match line to a corresponding word line of the data array to enable access of the data array in that word line when a match occurs on that match line. In a fully associative cache each row of the CAM holds the full address of a corresponding main memory location and the inputs to the CAM require the full address to be input.

[0027] A prior art publication, entitled "A Reconfigurable Content Addressable Memory," by Steven A Guccione et al., discusses the implementation of a CAM within an FPGA. As is seen in Prior Art **FIG. 1**, at a transistor level, the implementation of a CAM circuit **101** is very similar to a standard SRAM **100**. Both CAM and SRAM circuits are almost identical, each having 6 transistors **102** except for the addition of three match transistors **103** that provide for the parallel search capability of the CAM **101**. Unfortunately, using standard programmable logic devices does not facilitate implementing such transistor level circuits.

[0028] In the prior art publication the implementation of the CAM in an FPGA is discussed. Using gate level logic to implement a CAM often results in an undesirable size of the CAM. Flip-flops are used as the data storage elements within the CAM and as a result the size of the CAM circuit attainable using an FPGA is dependent upon the number of flip-flops available within the FPGA. Implementing the CAM in an FPGA quickly depletes many of the FPGA resources and as a result is not a viable solution. Unfortunately this has lead prior designers to conclude that the CAM is only efficiently implemented at a transistor level.

[0029] The prior art publication also addresses implementing of a CAM using look up tables (LUTs) in an FPGA. Rather than using flip-flops within the FPGA to store the data to be matched, this implementation addresses the use of LUTs for storing of the data to be matched. By using LUTs rather than flip-flops a smaller CAM architecture is possible.

[0030] Unfortunately, forming CAMs from synthesisable elements is not easily done so prior art processors that offer CAM are provided with a CAM core within the processor. Providing a CAM core within the processor unfortunately makes the resulting circuit expensive because of the added design complexity. Such additional design complexity is ill-suited for small batch custom design processors.

[0031] **FIG. 2** illustrates the translation process from a virtual address (VA) **201** to a physical address (PA) **202**. The VA **201** is a 32-bit address, VA[31:0], and the PA **202** is also a 32-bit address PA[31:0]. The VA has two portions, a virtual page number (VPN) **203** and a page offset (PO) **204**. The VPN **203** is typically located in the upper portion of the VA **201** and the PA **202** is typically located in the lower portion, though this need not be so. Typically for a 32-bit addressing scheme, the VPN is 20 bits and the PA is 12 bits. The PA, or lower 12 bits translate directly into the PA. The VPN **203** is used for indexing the TLB **205** to retrieve a physical page number (PPN) **206** therefrom. In other words, the VPN **203** undergoes translation to the PPN **206**. Combining the PPN **206** in the upper portion of the PA **202** and the PO into the lower portion of the PA provides a translation from the VA to the PA.

[0032] **FIG. 3** illustrates the translation from a VA **201** to a PA **202** when performed in conjunction with a direct mapped cache memory **301**. At the beginning of a translation cycle, the VA is used to access both the cache memory **301** and the TLB **205**. The page-offset portion of the VA is used to access the cache memory **301**—the page offset being the portion of the address that remains unmodified by the translation process. The page offset is used to index a tag array **302** and a data array **303** found in cache memory **301** where the page offset is used to index a cache line **302** a within the cache memory **301**. Access to the TLB **205** is performed using the VPN **203** portion of the VA **201**. The TLB **205** typically comprises a TLB tag array **304** and a TLB

data array **305**. Both the TLB tag array **304** and the TLB data array **305** contain bits from the VPN **203** such that when a VPN is provided to both of these arrays, the bits making up the VPN are compared to those stored within the arrays **304**, **305** to locate an entry within the TLB **205**.

[**0033**] Once the TLB data array **305** is accessed and a match is found between the VPN and an entry within the TLB data array **305***a*, the PPN **206** is retrieved and is provided to the cache memory **301** and used for comparison to the tag retrieved **302***a* from the tag array **302**. A match being indicative of a cache "hit"**306**. If a match is found between the VPN **203** and an entry within the TLB tag array **304***a* then a TLB hit signal **307** is generated. In this manner, the cache is only accessed using bits of the PPN **206**. The above example illustrates the use of a direct mapped cache memory; however, the same translation of a VA to a PA is applicable to set-associative caches as well. When set-associative caches are used, those of skill in the art appreciate that the size of a cache way is less than or equal to the size of a virtual page.

[**0034**] Unfortunately, when a TLB is implemented in SRAM, an exhaustive search of the memory is required to support CAM functionality. Thus, when a TLB has storage for 1024 virtual addresses and their corresponding Physical Address, each address translation requires up to 1024 memory access and comparison operations. Such a CAM implementation is unworkable as the performance drops linearly with CAM size.

[**0035**] **FIG. 4***a* generally illustrates a TLB **400** formed using synthesisable logic components **499** and a random access memory (RAM) **410**. A VPN for translation is provided via a VPN_IN input port **450**, where bits VPN_IN [31:12] are provided from the VA[31:0] to this input port **450**. A page mask signal is provided via a CPO_PAGE_MASK input port **451**. A CP0_TRANSLATION input signal is provided via a CP0_TRANSLATION input port **452**. A TLB_TRANSLATION output signal is provided via TLB_TRANSLATION output port **453**, in dependence upon a translation from a VA to a PA using the TLB **400**.

[**0036**] **FIG. 4***b* illustrates a TLB **400** in more detail formed from synthesizeable logic components, and in **FIG. 4***c* the steps of operation for the TLB **400** are shown in summary. In more detailed description of the TLB operation, a VPN for translation is provided **480** via a VPN_IN input port **450**, where bits VPN_IN[31:12] are provided from the VA[31:0] to this input port **450** as the VPN. A page mask is provided via a CP0_PAGE_MASK input port **451**. This page mask is provided to a page mask encoder **408**, for encoding the page mask according to Table 1.

TABLE 1

Page Mask Encoding

page size	mask[2:0]
4 kB	000
16 kB	001
64 kB	010
256 kB	011
1 M	100
4 M	101
16 M	110

[**0037**] The page mask encoder **408** is used for accepting the CPO_PAGE_MASK input signal on an input port thereof and for correlating this input signal to a 3-bit vector, MASK[2:0]. The 3-bit vector MASK[2:0] is further provided to a hashing circuit **406**. The hashing circuit **406** receives VPN_IN[31:12] via a first input port **406***a* and MASK[2:0] via a second input port **406***b*. A hashed vector H_VPN[5:0] is provided from an output port **406***c* thereof via a hashing operation **481** of the hashing circuit **406**. The hashed vector H_VPN[5:0] and the MASK[2:0] are further provided to each one of 48 registers **409**, where each register consists of multiple flip-flops collectively referred to as **491**. Each of the registers **409** has two output ports. A first output signal from a first output port thereof is provided to a comparator circuit **403**. A second output signal from a second output port is provided to the second input port **406**b on one of 48 hashing circuits **406**. The first input port on this hashing circuit receives VPN_IN[31:12]. The hashing circuit **406** output port is coupled to one of 48 comparator circuits **403** for performing a comparison between the register output and the hashing circuit output signal. Each of the comparators, in dependence upon a comparison of two input signals, provides a '1' if the signals are the same and a '0' if they are different. Output signals hit, from each of the 48 comparators is provided to one of 48 single bit 2-input multiplexers **411**. Outputs ports from each of the multiplexers are coupled to a flip-flop **404**. Each of the flip-flop **404** generates an output signal provided at the output ports labeled try_1, where collectively these output signals try [0 . . . 47], for $0 \leqq i \leqq 47$ are provided to a priority encoder circuit **401**. The priority encoder circuit is further coupled to a binary decoder circuit **402**, where the priority encoder circuit asserts a TLB_ENTRY[5:0] signal to the binary decoder circuit **402** and to the RAM **410**. Three output ports are provided within the TLB **400**, an ENTRY_FOUND output port **454**, an ENTRY_NOT_FOUND output port **455** and a TLB_TRANSLATION output port **453**, for providing ENTRY_FOUND, ENTRY_NOT_FOUND, and TLB_TRANSLATION output signal, respectively.

[**0038**] An address for translation from a VA to a PA is stored in a random access memory (RAM) **410**, with the RAM **410** preferably having 48-entries, in the form of lines. In use, whenever a new translation is to be performed, input signals VPN_IN, CPO_PAGE_MASK, and CP0_TRANSLATION are provided to the TLB circuit **400** via input ports **450**, **451**, and **452**, respectively. Translations performed by the TLB are stored in RAM **410** for a given index, i. The given index, indexes one of the lines **410***a* within the RAM that holds the translation to the PPN. The hashing circuit **406** computes the hash function H (VPN_IN, MASK) and stores the result in a corresponding 6-bit register h_1 **490**. The page mask is stored in the 3-bit register m_i **491**.

[**0039**] When a translation is requested using the TLB, a VPN is provided via the input port **450** and the hash functions H (VPN_IN, m,) is computed for all i and compared to h_1. This yields a 48 bit vector **492** $hit_0 . . . hit_{47}$ which is subsequently loaded into a 48 bit register **493** $try_0 . . . try_{47}$. In order to determine whether the requested VPN_IN is present in the translation table stored in RAM **482**, only those entries, or lines, in RAM are checked for which try_i is asserted. An entry in the 48-bit try, vector is

asserted if it yields a '1'**483**. Of course, there may be more than one bit asserted in the try, vector, but the priority encoder **401** selects the entry with the lowest index to address entries within the RAM. The decoder **402** converts this index to a 48-bit one-hot vector **494** $clr_0 \ldots clr_{47}$. When the clock pulse arrives from a clock circuit (not shown), the try, vector is reloaded, except for a bit corresponding to an index just used to address the RAM, which is cleared. This process is repeated, one entry at a time **483**. The process stops as soon as the requested entry is found **484**, as indicated by the ENTRY_FOUND signal on the ENTRY_FOUND output port **454**, or when all bits in try, are 0. When all bits in try_i are '0' then the ENTRY_NOT_FOUND signal is provided via the ENTRY_NOT_FOUND output port **455**. In the first case the translation is successful and information for the translation is provided **485** from the RAM **410** using a TLB_TRANSLATION signal on the TLB_TRANSLATION output port **453**. In the second case the translation is not successful and the TLB reports a TLB refill exception.

[**0040**] **FIG. 5** illustrates a hashing circuit **506** in more detail. Using the MASK[2:0] and VPN[31:12] input signals to the hashing circuit **506**, a 7 to 1 multiplexer **501** provides the H_VPN[5:0] output signal from the hashing circuit **506** in dependence upon the MASK[2:0] signal provided to the second input port **506***b*. This hashing circuit selects the 6 least significant bits from the VPN. The selection is controlled by the page mask because the definition of "least significant" changes with the page size. For example, with a 4 kB page size, the 6 least significant bits (LSB)s of the VPN are bits 22:17, but with a 16 kB page size the 6 LSBs are bits 19:14. Since the TLB **400** stores two adjacent virtual pages per TLB entry, called an odd/even pair, the 6 LSBs for a 4 kB page odd/even pair are bits 18:13. Thus bit **12** decides whether to return the even (0) or odd (1) translation, and for a 16 kB odd/even pair the bits are 20:15. This hash function, however, is redundant, since the ordering of bits H_VPN [5:0] is irrelevant. **FIG. 6** exploits the fact that ordering of bits is irrelevant.

[**0041**] **FIG. 6** illustrates a variation of the hashing circuit shown in **FIG. 5**. A VPN[31:12] signal is provided to the first input port **606***a*, and a MASK[2:0] signal is provided to the second input port **606***b*. The mask signal MASK[2:0] is comprised of bits m0, m1, and m2. Within this hashing **606** circuit there are 3, 3 to 1 multiplexers **601** through **603**. The first multiplexer **601** receives the following bits, $\{m_2, \bar{m}_2 (m_1+m_0)\}$ on its selection input ports, where bits from the VPN, VPN[13:14], VPN[19:20], VPN[25:26] are provided to multiplexer data input ports, 0 through 2, respectively. Multiplexer **601** thus provides bits **5** and **4** to the H_VPN [5:0] output signal. The second multiplexer **602** receives the following bits $\{m_2(m_1+m_0), \bar{m}_2 m_1+m_2\bar{m}_1\bar{m}_0)\}$ on its selection input ports, where bits from the VPN, VPN[15:16], VPN[21:22], VPN[27:28] are provided to multiplexer data input ports, labeled **0** through **2**, respectively. Multiplexer **602** thus provides bits **3** and **2** to the H_VPN[5:0] output signal. The third multiplexer **603** receives the following bits $\{m_2m_1, \bar{m}_2m_1m_0+m_2\bar{m}_1)\}$ on its selection input ports, where bits from the VPN, VPN[17:18], VPN[23:24], VPN[29:30] are provided to multiplexer data input ports, labeled **0** through **2**, respectively. Multiplexer **603** thus provides bits **1** and **0** to the H_VPN[5:0] output signal.

[**0042**] Preferably, the hash function H_VPN[5:0] is uniformly distributed for MASK[2:0] and for VPN_IN[31:12] input signals. In the case of a TLB miss, all entries within the RAM are looked up for which try, is initially asserted. The number of cycles N_{miss} is given by the following equation:

$$N_{miss} = \sum_{j=0}^{48} \binom{48}{j} p^j (1-p)^{48-j} (1+j)$$

[**0043**] where p is the probability that a comparator output signal hit_1 is asserted. The term:

$$\binom{48}{j} p^j (1-p)^{48-j}$$

[**0044**] gives the probability that exactly j bits in the try vector try_i are initially asserted. Having a uniform hashing function H with n bits at the output signal thereof, $p=2^{-n}$, wherein the case of **FIG. 4***b*, n=6.

[**0045**] In the case of a TLB hit, at least one access to the RAM **410** us required, as opposed to a TLB miss condition which is detected without accessing the RAM, since in a TLB miss condition the try vector try, contains all zeros.

[**0046**] The average number of cycles to perform a translation that hits in the TLB is given by the following formula:

$$N_{hit} = \sum_{k=0}^{47} \binom{47}{k} p^k (1-p)^{47-k} \left(1+\frac{k}{2}\right)$$

[**0047**] For a TLB hit, there must be at least one '1' in the try vector try_1. The only uncertainty is with the remaining elements within the vector. The variable k is used to represent the number of remaining entries that are set to '1' within the try vector try_1 for k in the range from 0 . . . 47. If k=0 then only one entry within the RAM is looked up. Therefore, since one clock cycle was used to find the translation in the first location for i=0, then a total of two clock cycles are utilized to perform the translation. On average, it takes 2+k/2 cycles to return the requested translation from RAM **410**.

[**0048**] In terms of performing the translation and interrupt latency, the number of clock cycles required is examined for long lookup sequences, for instance having a k as high as 25 or more. The following relation:

$$P\{N_{25}\} = \sum_{25}^{48} \binom{48}{j} p^j (1-p)^{48-j}$$

[**0049**] gives the probability that the TLB will use 25 or more cycles to complete a translation. Table 2 lists, for a range of hash function widths (n), the average number of cycles it takes to find a translation N_{hit}, to detect a miss N_{miss} and the probability that the TLB operation takes 25 cycles or more.

TABLE 2

TLB latency as a function of the number of hash bits 'n'

n	N_{hu}	N_{hitq}	N_{miss}	$P\{N_{25}\}$
3	4.94	3.94	7.00	$4.3\ 10^{-11}$
4	3.46	2.46	4.00	$5.9\ 10^{-18}$
5	2.73	1.73	2.50	$3.6\ 10^{-25}$
6	2.37	1.37	1.75	$1.5\ 10^{-32}$
7	2.18	1.18	1.38	$5.4\ 10^{-40}$

[0050] From Table 2 it is evident that P $\{N_{25}\}$ is so small that even with a 4 bit hash function it takes more than 6000 years of continuous operation to run into a case where the TLB translation requires between 25 and 48 clock cycles.

[0051] The column N_{hltq} ("hit quick") applies to the case where the VPN_IN is applied continuously to the TLB circuit **400**. From this table it is evident that having n=5 or n=6 is sufficient when focusing on the most important number, which is N_{hlt}. There is not much to be gained beyond 6 bits, since N_{hit} approaches 2.0 when n=>20. A value of n=6 is used in the TLB circuit **400** since the hash function may not be very uniform. Therefore, 6-bit hash function used within the TLB approximates the performance of a 5-bit truly uniform hash function.

[0052] Advantageously, when VA is provided to the TLB it is propagated to the synthesized logic for each line and a result is provided indicated by at least an asserted bit within the try_1 vector of bits. Only those lines for which a result indicative of a match occured are then physically accessed to provide the PPN As such only a small fraction of the TLB lines are accessed for the translation process, thus resulting in a substantial performance improvement.

[0053] Numerous other embodiments may be envisaged without departing from the spirit or scope of the invention.

What is claimed is:

1. A translation lookaside buffer (TLB) comprising:

at least an input port for receiving a portion of a virtual address;

a random access memory;

a set of registers; and,

synthesisable logic for determining a hash value from the received portion of the virtual address and for comparing the hash value to a stored hash value within the set of registers to determine a potential that a physical address associated with the virtual address is stored within a line within the random access memory and associated with a register, from the set of registers, within which the hash value is stored.

2. A translation lookaside buffer according to claim 1, wherein the synthesisable logic comprises an encoder circuit for receiving at least a page size signal indicative of a page size and for determining from the at least a page size signal at least a mask bit for use in determining the hash value.

3. A translation lookaside buffer according to claim 2, wherein the synthesisable logic comprises a first hashing circuit including an output port and for receiving the received portion of the virtual address and the at least a mask bit, the first hashing circuit for deterministically generating a first hash value from the received portion of the virtual address and the at least a mask bit and for providing the first hash value to the output port thereof.

4. A translation lookaside buffer according to claim 3, wherein the synthesisable logic comprises at least a register (**409**) for storing the at least a mask bit.

5. A translation lookaside buffer according to claim 4, wherein the synthesisable logic comprises a second hashing circuit for receiving the portion of the virtual address and at least a stored mask bit from the at least a register, the second hashing circuit for determining the hash value.

6. A translation lookaside buffer according to claim 5, comprising a comparator and wherein the random access memory comprises a translation table stored therein, wherein the line is stored within the translation table and wherein the hash value is compared within the comparator to determine whether the virtual address is present at an associated line within the translation table, a positive indication indicative of a potential match and a negative indication indicative of no potential match for the virtual address within the associated line.

7. A translation lookaside buffer according to claim 5, comprising a third hashing circuit, the third hashing circuit for receiving the portion of the virtual address and at least a stored mask bit from the at least a register, the third hashing circuit for in parallel with the second hashing circuit for determining a hash vector, the hash vector for having stored therein the deterministically generated hash value for at least two lines in the random access memory.

8. A translation lookaside buffer according to claim 7, wherein the first hashing circuit pre-hashes the virtual address and at least a retrieved mask bit to provide a pre-hash value for provision to the second and third hashing circuits.

9. A translation lookaside buffer according to claim 8, comprising at least a second register, the at least a second register for storing a comparison result between the hash value and an associated line within the memory, the comparison result providing a positive indication indicative of a potential match and a negative indication indicative of no potential match for the virtual address within the associated line.

10. A translation lookaside buffer according to claim 9, wherein a plurality of the at least a second register includes a number of registers equal or greater than an amount of lines in memory of the translation lookaside buffer.

11. A translation lookaside buffer according to claim 1, wherein the synthesisable logic comprises a plurality of comparators for comparing a plurality of hash values to a plurality of expected hash values each expected hash value associated with a line in memory to determine a potential that a translation for the virtual address is stored within each associated line in memory, the comparators for, in combination providing a vector including a value associated with each line in memory.

12. A translation lookaside buffer according to claim 11, wherein the synthesisable logic comprises a resolution circuit for resolving between the plurality of values indicative of a potential translation of a line in memory containing the translation.

13. A translation lookaside buffer according to claim 1, comprising a hashing circuit for hashing a virtual address other than a virtual address for which a translation is presently stored to determine a hash value and for storing the

hash value in a register in association with an available line in memory, the hashing circuit for storing the virtual address and a translation therefor in the available line in memory.

14. A translation lookaside buffer comprising:

a random access memory;

a first register associated with a line in the memory; and,

a hashing circuit for receiving a virtual address other than a virtual address for which a translation is presently stored in the random access memory, for determining a hash value and for storing the hash value in the first register, the hashing circuit for storing the virtual address and a translation therefor in the line in memory.

15. A translation lookaside buffer comprising:

a RAM; and,

synthesisable logic for determining from a virtual address at least one potential address within the RAM in fixed relation to which to search for a physical address associated with the virtual address, the at least one potential address being other than the one and only known address within the RAM in fixed relation to which the physical address associated with the virtual address is stored.

16. A translation lookaside buffer according to claim 15, wherein the synthesisable logic comprises a hashing circuit for hashing the virtual address to determine a value indicative of a line in the RAM at which to begin searching for a translation for the virtual address.

17. A translation lookaside buffer according to claim 16, wherein the synthesisable logic comprises a lookup resolving circuit for iteratively comparing virtual address data stored within a plurality of lines in the RAM with the virtual address to determine a presence of a translation for the virtual address, the lookup resolving circuit for providing the resolved translation at an output port thereof.

18. A translation lookaside buffer according to claim 17, wherein the lookup resolving circuit includes an incrementing circuit for incrementing an address of the line within the memory by a line between successive comparison operations.

19. A translation lookaside buffer according to claim 17, wherein the lookup resolving circuit includes a hashing circuit for varying an address of the line within the memory by a line between successive comparison operations.

20. A method of performing a virtual address lookup function for a translation lookaside buffer including RAM and synthesisable logic including the steps of:

providing a virtual address to the synthesisable logic;

hashing the provided virtual address to provide a hash result;

based on the hash result determining a memory location within the RAM relative to which is stored a virtual address identifier and a physical address related thereto;

comparing the virtual address to the virtual address identifier to determine if the physical address corresponds to the provided virtual address; and,

when the physical address corresponds to the provided virtual address, providing the physical address as an output value.

21. A method according to claim 20, wherein the step of determining a memory location includes the steps of:

comparing the hash result against a known hash result associated with a line in RAM to determine a comparison result;

forming a hit vector including a plurality of values, each value associated with the line in RAM associated with the known hash result and relating to a likelihood that the line in RAM with which the value is associated contains a translation for the virtual address.

22. A method according to claim 21, wherein the step of determining is performed for each line within the RAM.

23. A method according to claim 22, wherein the step of comparing is performed for sequentially for different lines in RAM associated with a value indicative of a potential translation for the virtual address until the line containing the translation is located.

24. A method according to claim 20, wherein the step of determining is performed for each line within the RAM.

25. A method according to claim 24, wherein a line in memory includes a single address translation from a single virtual address to a single corresponding physical address.

26. A method according to claim 20, wherein the step of hashing includes the steps of hashing the virtual address and a stored value associated with each line in memory to determine independently a hash value associated with each line in memory.

27. A method according to claim 26, wherein a line in memory includes a single address translation from a single virtual address to a single corresponding physical address.

28. A method according to claim 20, wherein the step of hashing includes the steps of:

pre-hashing the virtual address;

combining the pre-hashed virtual address independently with each of a plurality of stored values, each associated with a line in memory to determine independently a hash value associated with each line in RAM.

29. A method according to claim 28, wherein the step of comparing is performed for sequentially for different lines in RAM associated with a value indicative of a potential translation for the virtual address until the line containing the translation is located.

30. A method according to claim 28, wherein a line in memory includes a single address translation from a single virtual address to a single corresponding physical address.

31. A method according to claim 20, wherein the step of hashing includes the step of hashing the virtual address to determine a value indicative of a line in RAM where a translation for the virtual address is potentially stored.

32. A method according to claim 31, wherein the step of comparing includes the step of iteratively comparing in a known order starting from the line in RAM related to the determined value a plurality of lines in RAM until a translation for the virtual address is located.

33. A method according to claim 32, wherein the known order is incremented by one line between successive comparisons

34. A method according to claim 32, wherein the known order is determined by solving a function based on the determined value.

* * * * *