·体系结构与软件技术· 文章编号：1000—3428(2013)05—0073—05 文献标识码：A 中图分类号：TP311

基于基地址寄存器映射的数据缓存研究

沈秀红，赵朝君，孟建熠，项晓燕
(浙江大学超大规模集成电路设计研究所，杭州 310027)

摘 要：针对深流水线中加载指令的延时长和功耗高的问题，提出一种基于基地址寄存器映射的数据缓存访问方法。该方法在加载指令执行过程中，动态构建基地址寄存器与目标数据的局部性访问历史，并通过设计基地址寄存器跟踪缓存器，在指令译码后直接获得目标数据，从而加速加载指令的数据获取过程，减少地址计算和对高速缓存的访问。测试结果表明，该方法的处理器性能平均提高约 3.7%，数据高速缓存功耗平均降低约 18.7%。

关键词：映射关系；基地址寄存器映射；内存访问局部性；数据一致性；高速缓存

Data Buffer Research Based on Base Address Register Mapping

SHEN Xiu-hong, ZHAO Chao-jun, MENG Jian-yi, XIANG Xiao-yan
(Institute of VLSI Design, Zhejiang University, Hangzhou 310027, China)

【**Abstract**】Aiming at the problem of load latency and power consumption in deep pipeline, this paper proposes a method of getting data immediately which is based on base address register. Building locality accessing history between base address register and destination data dynamically during the execution of load instruction, and designing a base address register tracking buffer, which let load instruction get data immediately at decode stage. This method accelerates the speed of load instruction to get destination data, and avoids address calculating and cache accessing. Results of benchmark show that performance of processor with this method increases about 3.7% averagely, and data cache power reduces about 18.7% averagely.

【**Key words**】mapping relationship; base address register mapping; locality of memory access; data coherence; high rate cache
DOI: 10.3969/j.issn.1000-3428.2013.05.015

1 概述

随着高性能处理器流水线级的不断增加，内存访问延时成为制约处理器整体性能的重要因素。内存访问分为存储指令访问和加载指令访问，存储指令的长延时问题可通过写缓存器机制解决[1]，而加载指令由于需要从内存中获取数据延时很长，且由它产生的数据相关性会直接影响后续指令的执行[2]，因此减少加载指令的延时对提高处理器性能至关重要。加载指令延时为指令运行的惯性延时与缓存缺失延时之和。缓存缺失延时指加载指令由于 Cache 缺失而从片外获得数据的延时，降低该延时的方法是提高 Cache 命中率，文献[3-5]从数据预取的角度出发，根据数据局部性原理提出一些预取机制，从而减少缓存缺失延时。而指令的惯性延时是 Cache 命中情况下，加载指令需要运行的时钟周期数，因此，是加载指令固有的最短执行周期。

目前旨在减少惯性延时的方法分为提前执行加载指令、预测内存地址、预先缓存数据等方面考虑。文献[2]提出了一种基于带预译码功能的指令队列，将指令队列中的加载指令提前执行，并在指令执行阶段通过一种侦查方法检查操作的正确性。该方法对硬件要求较高，且控制逻辑复杂度高。文献[6]提出一种能够准确预测内存地址的方法，该方法在译码阶段准确获得堆栈操作的地址，发起内存访问请求，并且增加堆栈 Cache 从而加速堆栈访问。该方法提前了载入指令的执行过程，但它只适用于堆栈操作，对普通的内存访问没有加速作用。文献[7]提出一种提前计算内存地址的方法，在译码阶段访问寄存器堆并计算内存地址，可提前访问数据 Cache。这种方法的硬件开销大，需在译码阶段增加计算地址的逻辑，对时序要求高，需要降低处理器工作频率。文献[8]提出一种用加载/存储队列临时缓存数据，加载指令先访问队列，若匹配失败则再访问 Cache。该方法有效降低 Cache 的动态访问功耗，但对性能提升不大。文献[9]在文献[8]的基础上将加载/存储队列的数据位宽与 Cache 带宽设计相同，避免了对 Cache 的多次访问。文献[10]提出一种数据写入 Cache 的同时将数据缓存一个存储队列中，若后续的加载指令地址命中存储队列，从存储队列获得数据，从而避免 Cache 访问。该方法对加载和存储指令的相关性要求较高，性能提升有限。文献[8-10]都从缓存数据和地址的映射关系角度降低加载指令的访问延

作者简介：沈秀红(1987－)，女，硕士研究生，主研方向：计算机体系架构；赵朝君，博士研究生；孟建熠，讲师、博士后；项晓燕，博士研究生

收稿日期：2012-05-28 **修回日期**：2012-07-24 **E-mail**：shenxiuhong@zju.edu.cn

时，但即使数据命中在缓存中，加载指令仍需执行到加载/存储单元才能获得数据，延时降低有限。

为降低加载指令的数据相关延时，本文提出一种适用于嵌入式处理器的基于基地址寄存器映射的数据缓存访问方法。本文主要内容包括：分析内存访问的局部性和加载指令编译特征，提出利用基地址寄存器与目标数据映射关系的数据缓存方法；在译码阶段增加基地址寄存器跟踪缓存器，用来存储加载指令执行过程中动态构建的基地址寄存器与目标数据映射关系，缓存目标数据，后续加载指令在译码阶段直接访问该缓存器，获取加载数据，避免地址计算和数据 Cache 访问；在加载/存储单元增加地址检查表，维护基地址寄存器跟踪缓存器与 Cache 的数据一致性。

2 内存访问局部性研究

嵌入式处理器一般采用精简指令集，加载指令需在执行阶段完成地址的计算，在内存访问阶段完成对 Cache 访问，流水线越长，所需周期越多。以经典 5 级流水线为例进行研究，加载指令至少需执行 5 个周期才能回写寄存器。对与加载指令有数据相关性的后续指令，须先停顿流水线等待数据从数据 Cache 中取回后才能执行，如以下指令流：

```
ldb    r3, (r2, 0x0)
add    r6, r3
```

加法指令的源寄存器是加载指令的目的寄存器，加法指令必须停顿在译码级等待数据取回。程序流水线序列如表 1 所示，流水线需停顿 2 个周期。若将加载指令所需数据预先缓存在译码级，提前获得数据，避免地址计算与 Cache 访问，则流水线无需停顿。

表 1 程序流水线序列

指令	周期数							
	1	2	3	4	5	6	7	8
ldb r3,(r2,0x0)	IF	ID	EX	MEM	WB			
addu r6,r3		IF	ID	停顿	停顿	EX	MEM	WB
ldb r3,(r2,0x0)	IF	ID	WB					
addu r6,r3		IF	ID	EX	MEM	WB		

内存拷贝、堆栈操作一般表现为连续的内存访问，因此程序在编译过程中设置相同的基地址寄存器，然后通过偏移量的递增实现内存的连续访问。加载指令根据译码信息可获得基地址寄存器号和偏移量，将该信息与目标数据建立映射关系，则后续加载指令通过跟踪基地址寄存器，能够准确获得目标数据。为了研究基地址寄存器命中率，本文对 Powerstone 测试基准进行了统计，统计结果如表 2 所示。

表 2 基准测试程序统计结果

测试基准	总指令数	加载指令	加载指令占总指令数的比例/(%)	跟踪命中加载指令	跟踪命中加载指令占总加载指令数的比例/(%)
adpcm	58 140	9 456	16.26	3 751	39.67
bcnt	5 795	388	6.70	113	29.12
blit	32 314	2 020	6.25	1 754	86.83
compress	200 296	29 601	14.78	5 924	20.01
engine	447 028	94 990	21.25	20 163	21.23
g3fax	1 243 440	107 316	8.63	56 916	53.04
pocsag	53 758	6 787	12.63	2 504	36.89

统计结果表明，加载指令平均占总指令数的 12.36%，命中其中基地址寄存器可映射的占 40.97%。程序体现出良好的内存访问局部性，为本文研究提供了重要的理论依据。

3 基地址寄存器跟踪技术

基于上述实验结果，本文提出在译码级建立基地址寄存器跟踪缓存器，用来缓存已完成的加载指令的基地址寄存器号、偏移量和目标数据以及之间的映射关系。当新的加载指令在译码级译码后，通过基地址寄存器号检索基地址寄存器跟踪缓存器，并根据偏移量判断指令是否命中。若命中，取出跟踪缓存器对应的目标数据，后续指令无需停顿流水线就能获得操作数，完全消除数据相关延时；若发生缺失，则加载指令按传统方式执行，也不会导致额外的性能损失。

流水线架构如图 1 所示。

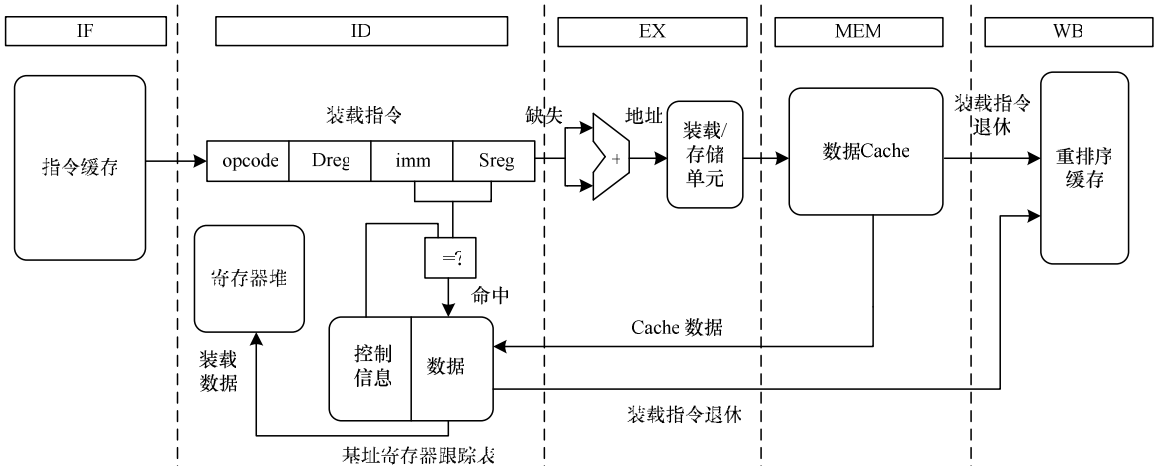

图 1 流水线架构

本文方法不仅加速后续与加载指令有数据相关性的指令，而且也加速加载指令本身的执行。当加载指令从基地址寄存器跟踪缓存器中获得数据后，可回写通用寄存器堆，避免地址计算和数据 Cache 访问，降低了加载指令的执行周期。

3.1　基地址寄存器跟踪缓存器

基地址寄存器跟踪缓存器由两部分组成：加载指令信息和目标数据，具体架构如图 2 所示。

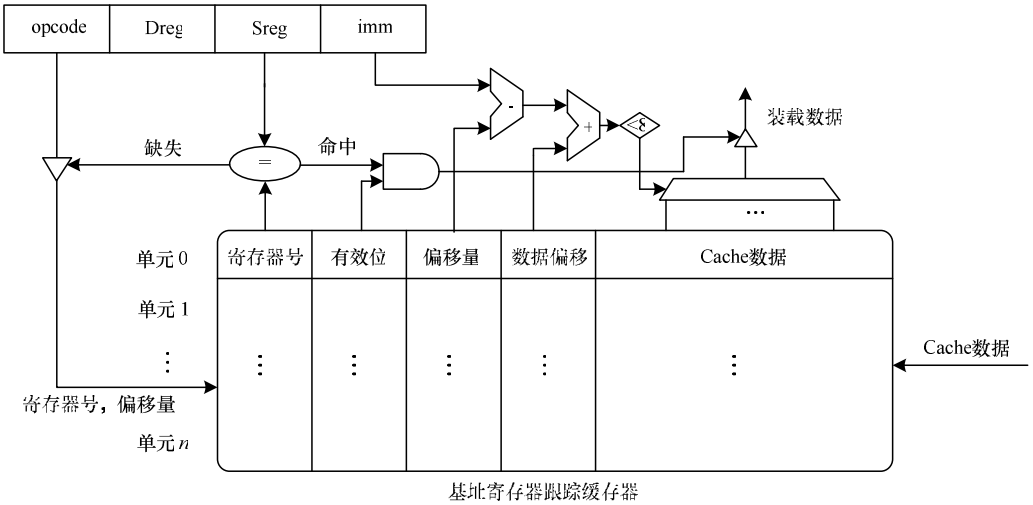

图 2　基地址寄存器跟踪缓存器

加载指令信息包括寄存器号、单元有效位、偏移量和数据偏移，寄存器号表示所缓存加载指令的基地址寄存器号，单元有效位表示所缓存目标数据是否有效，偏移量表示所缓存加载指令的偏移量，数据偏移表示所需加载数据在 Cache 数据中的位置。目标数据包括加载数据所在的 Cache 行。

当加载指令到达译码级并译码后，根据基地址寄存器号索引基地址寄存器跟踪缓存器，共有 4 种索引结果：

(1)跟踪缓存器中存在基地址寄存器号相同的有效单元，根据译码得到的偏移量与单元中的偏移量、数据偏移进行计算，如果偏移在一个缓存数据范围内，直接取出所需的数据回写寄存器，避免了地址计算和 Cache 访问。

(2)跟踪缓存器中不存在基地址寄存器号相同的单元，根据该加载指令的寄存器号创建新的单元，并设置偏移量信息，同时发射加载指令到执行单元进行地址计算和 Cache 访问，将加载数据及相邻数据回写到新创建的单元，并设置数据偏移信息和有效位。

(3)跟踪缓存器中存在基地址寄存器号相同的单元，但有效位为无效，这种情况只需将指令偏移量替换当前单元的偏移量，不必创建新的单元，其余操作与第(2)种情况相同。

(4)跟踪缓存器中存在基地址寄存器号相同的有效单元，但偏移量的计算结果超出缓存数据范围，则需要重新访问 Cache。此时将加载指令的偏移量写入该单元，并将有效位置无效，其余操作与第(2)种情况相同。

基地址寄存器跟踪缓存器采用最近最少使用替换策略，当没有空余的单元时，替换最近最少使用的单元。有效位置无效的情况有 2 种：(1)寄存器号相同，但偏移量超出缓存数据范围，需将有效位置无效，写入新的数据时，才置有效；(2)其他指令修改了寄存器的值，可在译码阶段判断指令的目的寄存器号是否与跟踪缓存器中缓存的寄存器号相同，若相同，则将相应单元的有效位置无效。

在数组操作中，经常出现将加载指令的基地址寄存器做加减操作的情况，对于立即数较小的情况，基地址寄存器中的值还在缓存数据范围内，此时，无需将该单元的有效位置无效。该特殊处理硬件实现简单，只需在译码阶段判断指令是否为加减法指令，以及立即数与缓存的偏移量、数据偏移的计算结果是否偏移出缓存数据。如果偏出，将有效位置无效，否则，将单元中的数据偏移修订为该计算结果。

3.2　数据一致性

地址检查表的具体结构如图 3 所示。

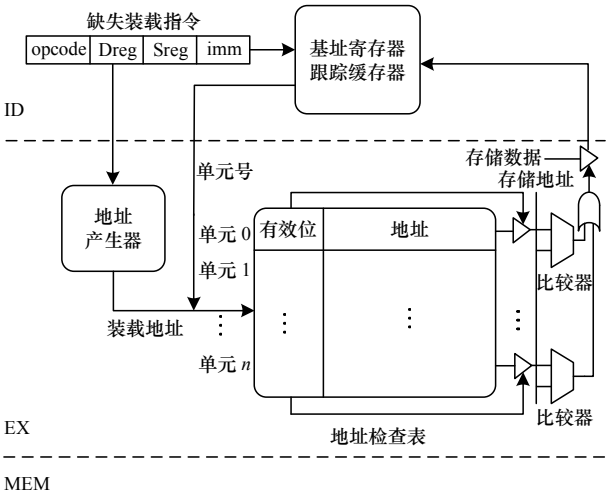

图 3　地址检查表

基地址寄存器跟踪缓存器中缓存的数据与 Cache 中的数据存在一致性问题。若存储指令将 Cache 中的数据改写，而跟踪缓存器的目标数据没有得到更新，则后续加载指令可能从单元中获取到错误的数据。为了防止错误的发生，需在加载/存储单元中加入地址检查表，用于检测存储指令的地址是否命中在已缓存的数据上。

地址检查表中的单元与基地址寄存器跟踪缓存器一一对应，当有加载指令发射到执行单元时，说明该指令在跟踪缓存器中创建了新的单元，将单元号信息传递到执行单元，根据单元号在检查表中创建相应的单元，写入加载指令的地址，并置有效。当跟踪缓存器中有单元置无效时，检查表中相应的单元也要置无效。若存储指令的地址命中在检查表，写 Cache 的同时，将跟踪缓存器中相应的数据更新。

当发生 Cache 缺失时，Cache 行有可能被替换。若在基地址寄存器跟踪缓存器中缓存的数据在 Cache 中被替换，该数据仍有效，虽然与 Cache 的数据不一致，但数据在内存中没有被修改。

4 实验与分析

本文实验采用 powerstone 基准测试程序，基于国产 32 位高性能嵌入式处理器 CK510[11]的时钟精确模型。模型具有以下特点：

(1)单发射乱序执行。

(2)七级流水线。

(3)执行单元有：1 个加载/存储单元，1 个算数运算单元，1 个逻辑运算单元，1 个跳转单元，1 个乘法单元。

(4)8 KB L1 Cache，2 路组相连，Cache 行大小 32 Byte、16 Byte 可配，64 bit 访问宽度。

(5)Cache 访问需 3 个周期。

Cache 行配置位 32 Byte 和 16 Byte，基地址寄存器跟踪缓存器中目标数据大小为一个 Cache 行。加载指令基地址寄存器跟踪命中率如图 4 所示。

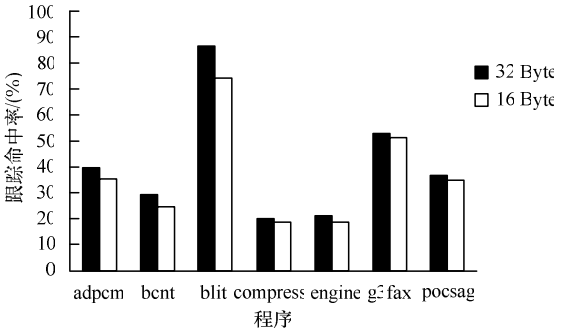

图 4 基地址寄存器跟踪命中率

实验结果表明，对于内存访问局部性较好的程序，基地址寄存器跟踪命中率很高，如 blit 程序，跟踪命中率高达 87%；同时 32 Byte 的 Cache 行命中率大于 16 Byte，其原因在于缓存的数据越多，加载指令在译码级获得数据的概率越大。

采用 16 个单元的基地址寄存器跟踪缓存器，在不同 Cache 行大小配置下，统计了采用本文方法的处理器的性能提升，实验结果如图 5 所示。

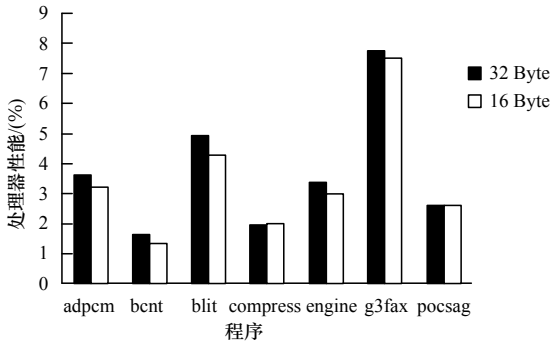

图 5 处理器性能

在 32 Byte 大小 Cache 行配置下，处理器性能平均提升 3.7%；在 16 Byte 大小 Cache 行配置下，处理器性能平均提升 3.4%。Cache 行越大，数据命中率越高，性能提升越多，但对 Cache 的数据带宽要求变高，需权衡考虑。

基地址寄存器跟踪缓存器采用不同的单元数，实验结果如图 6 所示。实验结果显示，对于一些测试程序来说，跟踪缓存器的单元个数对性能提升的影响不大，如 g3fax、blit；而对其他测试程序来说，单元个数越多，提升性能效果越好，这种特性由程序内部使用基地址寄存器个数决定，g3fax 和 blit 只使用 1 个~2 个基地址寄存器，所需单元数少。同时可以看出，8 个单元和 16 个单元对性能的提升差别不大。

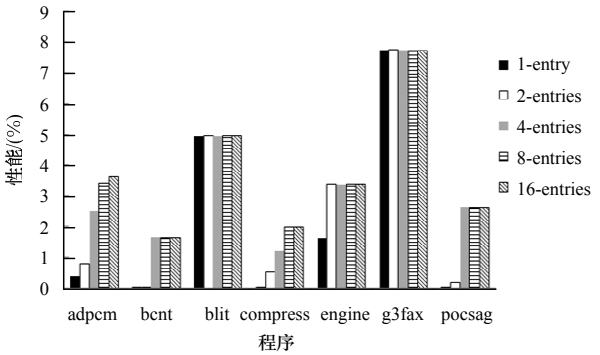

图 6 不同单元数配置下的性能提升

由于基地址寄存器跟踪缓存器需读取 Cache 数据，对 Cache 功耗有一定影响。当跟踪缓存器缺失率较高时，缓存数据使用率不高，可能增加 Cache 的功耗。为了降低 Cache 功耗，对 Cache 的访问策略进行优化。每次只缓存加载指令所需数据以及它之后的数据，且当有指令修改基地址寄存器的值时，马上中断映射关系的建立。同时，当相同的基地址寄存器被访问 2 次时才创建该寄存器与目标数据的映射关系，该优化能提高缓存数据的使用率。

Cache 功耗主要由标识阵列和数据阵列的访问功耗组成，计算公式为 $P=P_{tag}\times N_{tag}+P_{data}\times N_{data}$，$P_{tag}$ 和 P_{data} 分别表

示每次访问标识阵列和数据阵列的功耗，N_{tag} 和 N_{data} 分别表示程序执行过程中访问标识阵列和数据阵列的次数。本文对 Cache 功耗的分析不考虑基地址寄存器跟踪缓存器的功耗，图 7 给出了各个测试程序下 Cache 功耗下降的百分比。

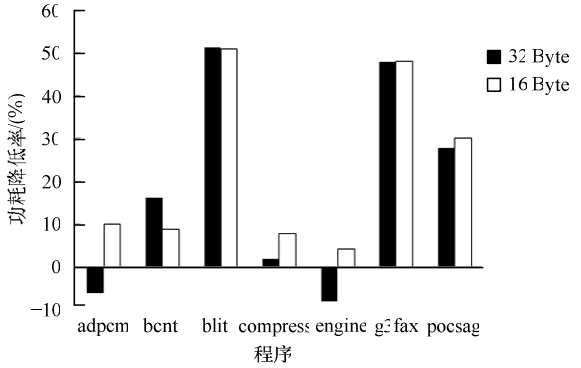

图 7　Cache 功耗降低率

实验结果显示，在 32 Byte 和 16 Byte 的配置下，功耗平均减少 18.7%和 22.9%。对于有些数据局部性较差的程序，Cache 功耗有可能增加，如在 32 Byte Cache 行配置下，对于 adpcm 和 engine 测试程序，Cache 功耗分别增加了 6.2%和 8.0%，这是因为 Cache 功耗降低率与基地址寄存器跟踪命中率呈正比，缓存数据使用率越高，Cache 功耗越小。

5　结束语

本文提出在加载指令执行过程中动态构建基地址寄存器与目标数据的局部性访问历史，并通过设计基地址访问缓存器在指令译码后直接获得目标数据的方法，能有效加速加载指令的数据获取过程，同时减少地址计算和对高速缓存的访问，有效地减少加载指令的惯性延时。该方法适用于对内存操作频繁的程序，能明显提高性能，并降低 Cache 功耗。对于内存操作空间局部性差的程序，该方法效果不明显，且有可能增加 Cache 功耗。如何在本文方法基础上通过编译器优化，采用软硬件结合的方法降低内存访问延时是后续的主要研究方向。

参考文献

[1] Skadron K, Clark D W. Design Issues and Tradeoffs for Write Buffers[C]//Proceedings of the 3rd International Symposium on High Performance Computer Architecture, in San Antonio. [S. 1.]: Spring, 1997.

[2] Chang S C, Li W Y H, Kuo Y J, et al. Early Load: Hiding Load Latency in Deep Pipeline Processor[C]//Proceedings of the 13th Asia-Pacific Computer Systems Architecture Conference. [S. 1.]: Spring, 2008.

[3] Joseph D, Grunwald D. Prefetching Using Markov Predictors[C]//Proceedings of the 24th Annual International Symposium on Computer Architecture. Denver, USA: [s. n.], 1997.

[4] Vanderwiel S P, Lilja D J. Data Prefetch Mechanisms[J]. ACM Computing Surveys, 2000, 32(2): 174-199.

[5] Nesbit K J, Smith J E. Data Cache Prefetching Using a Global History Buffer[J]. IEEE Micro, 2005, 25(1): 90-97.

[6] Bekerman M, Yoaz A, Gabbay F, et al. Early Load Address Resolution via Register Tracking[C]//Proceedings of the 27th Annual International Symposium on Computer Architecture. [S. 1.]: IEEE Press, 2000.

[7] Hwang Yuan-Shin, Li Jia-Jhe. On Reducing Load Store Latencies of Cache Accesses[J]. Journal of Systems Architecture, 2010, 56(1): 1-15.

[8] Nicolaescu D, Veidenbaum A. Caching Values in the Load Store Queue[C]//Proceedings of IEEE Computer Society's 12th Annual International Symposium. [S. 1.]: IEEE Press, 2004.

[9] Jin Lei, Cho Sangyeun. Macro Data Load An Efficient Mechanism for Enhancing Loaded Data Reuse[J]. IEEE Transactions on Computers, 2011, 60(4): 526-537.

[10] Roth A. A High Bandwidth Load/Store Unit for Single and Multi Threaded Processors[EB/OL]. (2012-04-15). http://repository.upenn.edu/cis reports/1.

[11] C-SKY Microsystems. 32 Bit High Performance and Low Power Embedded Processor[EB/OL]. (2012-04-15). http://www.c-sky.com.

编辑　索书志

(上接第 72 页)

[7] Dezert J. Foundations for a New Theory of Plausible and Paradoxical Reasoning[J]. Information and Security, 2002, 9(1): 13-57.

[8] Smarandache F, Dezert J. A Simple Proportional Conflict Redistribution Rule[J]. International Journal of Applied Mathematics and Statistics, 2005, 16(3): 1-36.

[9] Smarandache F, Dezert J. Advances and Applications of DSmT for Information Fusion: Vol 1[M]. Rehoboth, USA: American Research Press, 2004.

[10] Smarandache F, Dezert J. Advances and Applications of DSmT for Information Fusion: Vol 2[M]. Rehoboth, USA: American Research Press, 2006.

[11] 胡丽芳, 关　欣, 何　友. 基于 DSmT 的多传感器目标识别[J]. 弹箭与制导学报, 2008, 28(2): 186-188.

编辑　索书志