SIMD指令集设计空间的形式化描述

李春江 徐 颖 黄娟娟 杨灿群
(国防科学技术大学计算机学院计算机研究所 长沙 410073)

摘 要 SIMD(Single-Instruction-Multiple-Data)并行体系结构在现代处理器体系结构中扮演非常重要的角色。SIMD指令集已经成为处理器指令集中重要的子集。SIMD结构和指令集实现了短向量并行处理能力,SIMD指令集实现了对多种数据类型、多种操作模式的支持。采用形式化的方法,描述SIMD指令集的设计空间,从多个正交的维度刻画SIMD指令集的设计,基于此详细讨论了SIMD指令集的设计问题。该形式化方法有益于对SIMD指令集体系结构的分析和设计。

关键词 SIMD,指令集,设计空间,形式化描述
中图法分类号 TP314 **文献标识码** A

Formal Description of Design Space of SIMD Instruction Sets

LI Chun-jiang XU Ying HUANG Juan-juan YANG Can-qun
(Institute of Computer, School of Computer Science, National University of Defense Technology, Changsha 410073, China)

Abstract SIMD (Single-Instruction-Multiple-Data) parallel architecture plays an important role in the architecture of modern processors. The instruction set of SIMD becomes the most important subset of the whole instruction set of a processor. SIMD architecture and instruction set provides the parallel processing ability for short vectors. The SIMD instruction set supports multiple data types and multiple operations. This paper described the design space of SIMD instruction set using formal method, which depicts SIMD instruction set from multiple orthogonal dimensions. Then the discussion about the design of SIMD instruction set was presented. The formal method will be beneficial to analyze and design the SIMD instruction set architecture.

Keywords SIMD, Instruction set, Design space, Formal description

1 引言

SIMD(单指令多数据)体系结构是非常普遍而高效的并行体系结构,在现代微处理器体系结构中扮演非常重要的角色。它可以在不提升取指令、指令解码带宽的基础上提高执行吞吐率[1]。目前几乎所有的通用微处理器都实现了SIMD扩展,众多国产CPU也支持SIMD扩展。目前一个重要的发展趋势是处理器实现SIMD操作的向量长度更长、数据元素精度更高、SIMD指令集的指令数更多、功能更强大。如Intel在最新的基于32nm工艺实现的X86_64处理器中支持AVX(Advanced Vector eXtension,高级向量扩展)[2],最大支持4路双精度SIMD计算;在兼容以前的SIMD指令集的情况下,当前X86_64处理器的SIMD指令集中SIMD指令数已达到400条左右。

在通用微处理器中实现SIMD体系结构并设计相应的指令集,已经经历了将近20年的发展,未来高性能处理器中SIMD指令集必将更加复杂。

指令集作为处理器硬件和软件间的界面,其设计非常复杂。当前任何一个使用广泛的指令集系统都经历了多年的发展,指令集的设计中凝结了非常多的计算机科学家、系统工程师的智慧。

针对一个已有的通用处理器,在分析其历经多年发展而来的SIMD指令集的时候,常常受复杂性、兼容性、体系结构演进历程等多方面因素的困扰,不知从何入手条分缕析地进行;基于一个高性能处理器的指令集体系结构(ISA),设计SIMD指令集的时候也常常不知如何入手才能设计出针对体系结构特点的适用且尽可能完备的指令集。

为了深入探讨SIMD指令集的分析和设计,本文采用形式化的方法描述了SIMD指令集的设计空间。从公开的文献来看,国内外还没有这方面的工作,本文试图在这个方面进行一些探索。采用形式化方法进行描述,对深入分析现有SIMD指令集以及设计新的SIMD指令集都有非常重要的指导意义。

2 背景

2.1 Intel X86系列处理器SIMD体系结构和指令集的演进

当前,并行体系结构已经成为几乎所有高性能处理器的

到稿日期:2012-08-21 返修日期:2012-11-22 本文受国家自然科学基金项目(61170046,61170045)资助。
李春江(1974—),男,博士,副研究员,硕士生导师,主要研究方向为计算机体系结构、编译及优化技术,E-mail:chunjiangli@gmail.com;徐 颖(1992—),女,硕士生,主要研究方向为编译及优化技术;黄娟娟(1988—),女,硕士生,主要研究方向为编译及优化技术;杨灿群(1968—),男,博士,研究员,硕士生导师,主要研究方向为系统软件。

普遍结构，在高性能通用处理器芯片内的并行有两方面最重要的发展趋势：1）多核甚至众核的并行；2）功能更强大的SIMD并行。

Intel公司的X86系列处理器的SIMD体系结构和指令集[3]经过将近20年的发展，目前已经相当丰富和复杂，在非常多的领域得到了广泛应用。这里首先回顾一下X86系列处理器SIMD指令集的演进过程。

MMX是Intel公司1997年在其X86系列处理中最先支持的SIMD实现，MMX指令集是Intel公司X86指令集中最早支持的SIMD指令集。MMX指令可以将一个有效数据位为64位的寄存器作为一个64位量或2个32位量或4个16位量或8个8位量来操作。MMX指令集使用了FPU(处理器中的浮点单元)中的寄存器，因此同一时刻不能同时执行浮点指令和MMX指令。MMX指令集中提供了8个64位寄存器(mm0—mm7)。除了emms指令、movd指令和movq指令外，MMX指令的助记符都以字符"p"开头。MMX指令集仅支持整型数的SIMD操作，推出该类指令集主要是用于对当时刚刚开始流行的多媒体应用进行加速，当时个人计算机上多媒体应用刚刚兴起。

SSE于1999年推出，它增加了8个新的128位寄存器，实现四路单精度浮点支持。这些寄存器为xmm0—xmm7，另外增加了一个控制寄存器MXCSR(用来设置和检查SSE指令的状态)。SSE指令集中包含了70条指令，用来操作128位寄存器，有时也操作32位通用寄存器。

SSE2最早在Intel Pentium 4处理器中引入，SSE2指令在结构上和SSE指令非常相近，但在SSE的基础上提供了更大的处理短向量数据的灵活性。SSE和SSE2最大的区别是，SSE2能处理双精度(64位)、单精度(32位)数据，也能处理xmm寄存器中128位整数。在原有SIMD指令集的基础上，SSE2共新增了144条指令。SSE2指令集直到2000年2月前后才完全公开。而AMD直到2003年才在其Opteron和Athlon64处理器中支持SSE2指令集。

SSE3在2004年初发布，仅在SSE2的基础上增加了13条指令，提供了新的功能，如水平操作(对一个短向量寄存器的数据元素进行操作)功能，并增加了一些允许存储器非对齐访问的指令。随着Intel酷睿2(Core 2)结构的推出，SSE3指令集中又加入了16条新的指令，新的指令集称为SSSE3。

SSE4在2006年9月27日正式发布，在2007年初发布的Intel和AMD处理器中实现了对SSE4的支持。SSE4在向量寄存器配置上和SSE2、SSE3相同，只是增加了更多的计算和访存功能。SSE4又分为3个子类：SSE4.1、SSE4.2和SSE4a；总计增加了54条指令，其中47条属于SSE4.1，其他7条指令属于SSE4.2。SSE4a是AMD实现的支持，它没有支持SSE4的所有指令，但增加了6条用于位操作的指令。

Intel的AVX是在其最新一代X86_64体系结构中支持的处理器体系结构增强和ISA扩展，首先在其Sandy Bridge(研发代号)处理器中实现，2011年正式在产品处理器中发布。

AVX有如下关键特性：

·向量宽度由128位增加到256位，由两个128位的load端口为256位的向量寄存器供数，可以提供更高峰值的双精度浮点计算性能和更高的能效。

·数据组织能力增强，使用新的256位向量来进行广播、带掩码的load及数据重排(又称为数据置换)，可以更快更有效率地组织、存取和提取必要的数据。

·支持3个甚至4个操作数的指令，源操作数所在向量寄存器不破坏，可以实现更少的寄存器拷贝，对向量和标量代码可以更好地使用向量寄存器。

·支持灵活的非对齐内存存取，可以有更多的机会使用向量存取和计算操作。

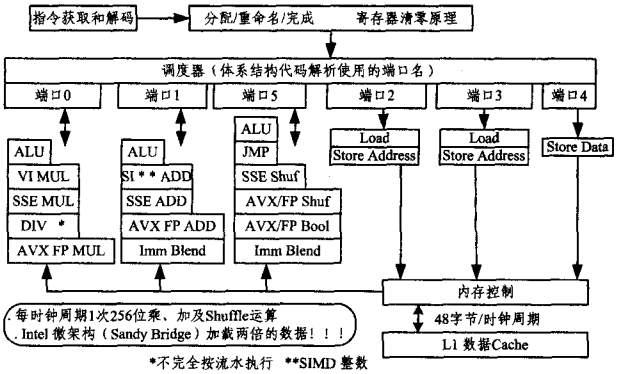

图1 Intel AVX微架构(引自IDF2011)

图1取自Intel开发者论坛中介绍AVX架构的资料，通过对图1的研读，可以得到如下结论：

为了实现256位向量的操作，处理器内核新增了如下关键模块：AVX浮点乘、AVX浮点加、AVX/浮点数据的混洗、AVX/浮点布尔操作、两个Imm Blend(立即数混洗)、一个取(Load)单元和一个存(Store)单元。

如果将对AVX的支持和原有对SSE的支持都算作短向量支持，那么在新的Intel的处理器架构中，其内核计算单元中大部分计算和操作资源都用于实现对SIMD的支持。在Intel的这款处理器内核中，传统意义上的处理标量数据的ALU(算术逻辑单元)仅仅相当于当前整个ALU单元的很小一部分，并且向量寄存器和标量寄存器之间可以直接传递数据。

在支持AVX的架构中，向量的存取没有旁路一级Cache，其一级单向存取链路的数据宽度为16个字节，两个取数单元同时工作刚好可以填满一个长256位的向量寄存器；这一访存能力对充分发挥向量处理能力至关重要。

新增加了取、存单元，可以提升数据、指令的存取能力，也对带掩码的取、存提供支持，这对发挥向量重组(数据重排)指令的作用也很关键。

在AVX架构下，将原来的16个128位的xmm0—xmm15寄存器增大为256位，命名为ymm0—ymm15，y寄存器的低128位仍命名为xmm0—xmm15。这样，AVX架构下的向量寄存器采用双路(double lane)模式，这种寄存器结构扩展也使得继续支持SSE成为可能，但是在操作xmm寄存器的时候，ymm寄存器的高128位被清0。但是SSE和AVX指令混合使用将会导致性能损失，因为AVX和SSE指令混合会导致硬件频繁保存和恢复高128位，所以最好在局部代码中全部使用SSE或AVX指令。如果非要混合AVX和SSE指令，那么使用vzeroupper指令清零高128位就不会出现性能损失。

AVX采用了全新的指令格式和编码方式，它可以实现一些更加复杂的指令，从而提升了X86 CPU的性能。其SIMD

指令采用指令前缀的形式，增强的寄存器配置可以在指令头部不断增加前缀；另外，体系结构中又增加了 8 个寄存器，实现了对于 REX 前缀的支持。基于这些新的寄存器配置，未来 Intel 可以在很多体系结构和指令集设计方面进行增强，以便提供更加流畅的架构，使大幅度扩充指令集成为可能，为未来的 512 位和 1024 位的 SIMD 扩展打下基础。

AVX 架构可以灵活支持 256 位的 AVX 代码、128 位的 AVX 代码、遗留的 128 位的 SSE 代码以及标量代码，有非常好的兼容性，也为后续进一步拓展打下了坚实的基础。

2.2 SIMD 体系结构和指令集的发展趋势

SIMD 体系结构已经成为高性能通用处理器中非常重要的细粒度并行体系结构，紧耦合的 SIMD 结构是非常值得重视的处理器内核体系结构。回顾 SIMD 体系结构及指令集的发展历史，对于 SIMD 体系结构和指令集发展趋势可以得到以下结论：

- SIMD 结构采用独立的向量寄存器文件已经成为必然趋势。
- SIMD 指令集越来越丰富，功能越来越强大。
- SIMD 结构中向量操作与标量操作间的协作关系越来越复杂。
- SIMD 结构对数据存取单元的能力提出了更高的要求。

因此，从一个典型的 SIMD 体系结构模型出发，采用形式化的方法描述 SIMD 指令集的设计空间，对 SIMD 指令集的设计和 SIMD 体系结构的设计都非常有意义。

3 SIMD 指令集设计空间的形式化描述

3.1 SIMD 体系结构抽象模型

为了形式化地描述 SIMD 指令集的设计空间，基于目前主流的 64 位通用处理器的一般结构特点，构建如下 SIMD 体系结构抽象模型：

- 寄存器文件：64 位整数寄存器 RI(支持所有的整型，包括字节、短整数、32 位整数及 64 位长整数)、单精度浮点寄存器 RF、双精度浮点寄存器 RD、向量寄存器 RV，各组寄存器文件相互独立，没有重叠。
- 存储器体系结构：处理器采用主流的多级高速缓存(Cache)存储器体系结构，所有寄存器的数据存取都通过一级 Cache，一级 Cache 的 Cache 行大小是向量寄存器 RV 长度的整数倍。
- 向量寄存器 RV 和标量寄存器 RI、RF、RD 之间可以直接传递数据，用 RS 表示所有的标量寄存器。
- SIMD 操作后的状态保存到状态寄存器 VSTAT 中。

这种 SIMD 体系结构模型，可以涵盖当前及未来众多的 RISC(精简指令集)和 CISC(复杂指令集)[4] 处理器中 SIMD 体系结构的主要特点。

3.2 SIMD 指令模型

为了完备地描述 SIMD 指令的模式，本文采用如下基本思路(或者称为方法学)：

1)对 SIMD 指令进行分类，并为每类指令分别定义指令模型；

2)指令模型采用非破坏性语法，即结果操作数不污染源操作数；

3)严格定义每类指令的功能和行为，保证每类指令间无功能重叠；

4)只有访存指令能访问存储器，其他指令的操作均在寄存器间进行，这一点和 RISC 指令集的设计原则相同。

这样的分类和抽象对于分析 SIMD 指令集、设计 SIMD 指令有非常大的帮助。在分类并设计指令模型之前，对指令模型中采用的公共表述作如下说明：

- “type”表示向量元素的数据类型。
- “bincal”表示二元运算操作。
- “cal”表示一元、二元、甚至多元操作。

根据以上的分类原则，将 SIMD 指令分为如下 8 类。

1)向量垂直计算指令(V_VCAL)

所谓的垂直计算，指的是进行某种操作的两个向量元素来自不同的向量，并且此类计算的源和结果操作数都保存在向量寄存器中。

此类指令采用 3 操作数模式，用“v[bincal][type]RV_{src1}，RV_{src2}，RV_{des}”来描述，v[bincal][type]代表指令助记符，RV_{src1} 和 RV_{src2} 是两个源操作数向量寄存器，RV_{des} 是结果操作数向量寄存器。两个源向量寄存器中的对应元素进行由 bincal 指定的二元运算，结果保存到结果向量寄存器中的对应位置。

2)向量水平计算指令(V_HCAL)

向量水平计算是指进行某种计算的源操作数来自一个向量中的不同元素，得到的结果可以是一个标量或一个向量；为统一指令描述，将结果的标量作为一个向量元素存储到结果向量的某一位置。

该类指令用“v[cal][type][pos] RV_{src1}，RV_{des}”来描述，特征码 pos 表示作为结果的一个标量或几个标量在结果向量寄存器中存放的位置。

在已有的 SIMD 指令集实现中，有些用一个源向量寄存器和一个结果向量寄存器的指令，本文把这类指令归为水平操作指令，如“sqrt v1，v2”指令将 v1 向量中的元素开方，将结果存入向量寄存器 v2 中，也作为水平操作指令。

3)向量连续访存指令(V_MAC)

向量连续访存是指向量元素从存储器中连续地读取到向量寄存器或从向量寄存器连续地写入存储器。连续的向量访存是最常见的向量访存方式，在此类向量访存中，并不限定存储器首地址是否按向量长度对齐。虽然面向基于高速缓存结构的通用处理器，通常要求向量访存时存储器对齐[5]，但是也有直接支持非对齐存储访问的 SIMD 指令实现，当然非对齐的存储器访问的开销要大很多。

此类指令采用 4 操作数模式，用指令模型“vst[type]VR，imm0，imm1，%mem”和“vld[type]%mem，imm0，imm1，VR”来描述。前者表示向量寄存器 VR 中的由 imm0 指定的位置开始的 imm1 个向量元素写入%mem 开始的地址处；后者表示将从%mem 开始的地址处的数据读入向量寄存器 VR 中，由 imm0 指定起始位置的 imm1 个向量位置。type 和 imm1 决定了连续访存的长度，而 type 和 imm0 决定了访存操作所影响到的向量寄存器的位置。

4)向量非连续访存指令(V_MADSC)

向量的非连续访存指的是将向量中的全部或部分数据元素，存储到存储器中非连续的位置中。

此类指令采用3操作数模式，用指令模型“vst[type]VR，imm_{VR}，%mem”和“vld[type]%mem，imm_{VM}，VR”来描述。imm_{VR}是特征立即数，用该立即数描述将源向量寄存器VR中的不连续的两个或多个向量元素，存储到存储器开始地址为%mem的存储器位置中，被存入的向量元素之间的相互关系（距离）维持不变，其他未被写入的存储器位置维持原值不变。imm_{VM}也是特征立即数，它描述从存储器中以%mem开始的不连续的若干元素取到向量寄存器VR中，被读取的向量元素间的相互关系（距离）不变，VR中未填入新读取向量元素的位置，保持原值。

5）不带向量元素复制的向量重组指令（V_SNDUP）

向量重组指令将来自两个源向量寄存器中的向量重组成一个新的向量保存到结果向量寄存器中，不带向量元素复制指的是在重组过程中不对来自源向量寄存器的向量元素进行复制。

此类指令采用3操作数模式，用“v[shuf][type]RV_{src1}，RV_{src2}，RV_{des}”来描述。v[shuf][type]代表向量操作，RV_{src1}和RV_{src2}是两个源操作数向量寄存器，RV_{des}是结果操作数向量寄存器。特征码shuf表示重组的模式。

6）不带向量元素复制的数据移动类指令（V-MVNDUP）

这里的数据移动是指在向量寄存器之间、向量寄存器和标量寄存器之间移动数据，不带向量元素复制是指被移动的向量元素在移动过程中没有被复制。

该类指令用“vmov[type]RV_{src}，RV_{dst}”和“vmov[type][pos]RS_{src}，RV_{dst}”来描述，前者表示向量寄存器RV_{src}中的向量完全移动到RV_{dst}向量寄存器中，后者表示将标量寄存器RS_{src}中的标量移动到向量寄存器RV_{dst}中，移动到的位置由特征代码pos决定。

7）带向量元素复制的数据移动类指令（V-MVDUP）

这里的数据移动是指在向量寄存器之间、向量寄存器和标量寄存器之间移动数据，带向量元素复制是指被移动的数据元素在移动的同时进行复制。

该类指令用“vmov[type][dupos][pos]RV_{src}，RV_{dst}”和“vmov[type][dupos][pos]RS_{src}，RV_{dst}”描述。前者表示源向量寄存器RV_{src}中的元素移动到目的向量寄存器RV_{dst}中，并且在此过程中RV_{src}中的一个向量元素被复制移动到RV_{dst}中，究竟复制的哪个向量元素、复制后移动到RV_{dst}中的哪些位置，由特征代码dupos和pos决定，其填充了复制数据的结果向量RV_{dst}中的位置，不再受源向量RV_{src}中对应位置向量元素的影响。后者表示标量寄存器RS_{src}中的标量复制并移动到向量寄存器RV_{dst}中的多个位置，移动到哪些位置由特征码dupos和pos共同决定。

8）辅助指令（V-AU）

这类指令的作用是对SIMD操作的状态进行判读和设置。通常SIMD操作完成之前需要设置处理器的状态，SIMD操作完成之后处理器进入新的状态。通常，在处理器面向SIMD的操作中可以设计一些辅助指令，如读取状态、设置状态或根据状态进行一些特定操作。在现有的处理器的SIMD指令集体系结构设计中，这类指令通常不多，但是也很重要。

该类指令用“v[stat][op][VSTAT]”来描述，其中stat指状态，op指进行的操作，通过定义stat和op来确定指令的功能。

如上的分类描述方法比较完备地刻画了SIMD指令的所有可能形式，各类指令的功能单一，彼此无重叠，对于更复杂的指令（比如在访存的同时进行了复制，向量重组中进行了向量元素的复制等），可以采用以上指令的组合完成，可将其看作复合指令。

3.3 SIMD指令集设计空间的形式化描述

根据以上建立的SIMD体系结构模型和对指令集的分类与指令模型的定义，本文用如下形式化的方法描述整个SIMD指令集的设计空间，如图2所示。

SIMD_INSN：V_VCAL
| V_HCAL
| V_MAC
| V_MADSC
| V_SNDUP
| V_MVNDUP
| V_MVDUP
| V_AU

V_VCAL：v[bincal][type] RV_{src1}，RV_{src2}，RV_{des}

V_HCAL：v[cal][type][pos] RV_{src1}，RV_{des}

V_MAC：vld[type] %mem，imm_0，imm_1，VR
| vst[type] VR，imm_0，imm_1，%mem

V_MADSC：vld[type] %mem，imm_{VM}，VR
| vst[type] VR，imm_{VR}，%mem

V_SNDUP：v[shuf][type] RV_{src1}，RV_{src2}，RV_{des}

V_MVNDUP：vmov[type] RV_{src}，RV_{dst}
| vmov[type][pos] RS_{src}，RV_{dst}

V_MVDUP：vmov[type][dupos][pos] RV_{src}，RV_{dst}
| vmov[type][dupos][pos] RS_{src}，RV_{dst}

V_AU：v[stat][op] [VSTAT]

图2 SIMD指令集的形式化描述

4 SIMD指令集设计

4.1 设计空间的遍历

上一节将SIMD指令集分为8个正交的子类，并为每类指令建立了指令模型。根据指令分类和指令模型，进一步讨论每类指令可以设计出多少种不同功能的指令，来实现对设计空间的遍历。

1）向量垂直计算指令（V_VCAL）

该类指令的模型为“v[bincal][type]RV_{src1}，RV_{src2}，RV_{des}”。其中，type表示向量元素的数据类型，当前及今后相当长一段时间内，通用高性能处理器的数据类型不外乎如下：字节（8位，有符号或无符号）、短整型（16位，有符号或无符号）、整型（32位，有符号或无符号）、长整型（64位，有符号或无符号）、单精度浮点（32位）、双精度浮点（64位）。bincal表示所有同类型数据元素的二元操作。那么可以根据功能的需要和体系结构实现的约束来定义该类SIMD指令设计。

2）向量水平计算指令（V_HCAL）

该类指令的模型为“v[cal][type][pos]RV_{src1}，RV_{des}”。其中，cal可以是一元操作、二元操作或归约操作。cal如果是一元运算，那么含义是对源向量寄存器中的每个向量元素执行cal所代表的一元运算，结果写入结果向量的对应位置，此时pos域可以忽略。如果是归约操作，则是对源向量RV_{src1}中的所有元素进行归约，得到一个归约结果，归约结果存入到

由特征量 pos 所指定的结果向量寄存器中的位置。如果是二元操作，情况就复杂得多，可以设计很多复杂的操作模式。例如，源向量寄存器 RV_{src1} 中有 8 个向量元素，每相邻两个做二元的 cal 运算，结果就有 4 个向量元素，可以设计足够的特征码 pos 指示将这 4 个结果向量元素存储到结果向量 RV_{des} 中的某些位置。

3)向量连续访存指令(V_MAC)

该类指令用指令模型“vst[type]VR，imm_0，imm_1，%mem”和“vld[type]%mem，imm_0，imm_1，VR”描述。根据不同的向量元素类型，在固定向量长度的情况下，可以根据 imm_0，imm_1 的组合形式设计覆盖全部模式的此类指令。例如，如果向量长度为 256 位，type 为双精度浮点，那么 imm_0 的可能取值为 1～4(我们定义向量元素序号从 1 开始)，而 imm_1 的可能取值为 1～4，可以设计出多种双精度向量元素连续存取的访存指令。注意我们将仅存取一个向量元素也定义为一种连续存取。

4)向量非连续访存指令(V_MADSC)

向量的非连续存取用指令模型“vst[type]VR，imm_{VR}，%mem”和“vld[type]%mem，imm_{VM}，VR”来描述。基于该指令模型，通过操作的个数来与向量的连续存取相区分；另外，这里的非连续存取的向量元素个数至少为 2，这也是与向量的连续存取不同之处。根据向量元素的类型，在固定向量位长的情况下可以知道向量的元素个数，那么可以遍历其不连续的两个或两个以上向量元素的组合形式，据此设计特征 imm_{VR} 就可以实现指令集设计空间的遍历。

5)不带向量元素复制的向量重组指令(V_SNDUP)

此类指令用“v[shuf][type]RV_{src1}，RV_{src2}，RV_{des}”来描述。在向量元素类型固定且没有向量元素复制的情况下，可以设计出遍历整个重组空间的特征码 shuf，实现向量无复制重组指令的遍历。

6)不带向量元素复制的数据移动类指令(V-MVNDUP)

该类指令用“vmov[type]RV_{src}，RV_{dst}”和“vmov[type][pos]RS_{src}，RV_{dst}”来描述。前者模式简单，只用不同向量元素类型下对应的不同标量寄存器类型就可实现该类指令设计空间的遍历。而后者通过设计不同的向量元素类型以及对应的标量寄存器类型、移动到结果向量寄存器 RV_{dst} 中的位置 pos，就可以实现该类指令设计空间的遍历。

7)带向量元素复制的数据移动类指令(V-MVDUP)

该类指令用“vmov[type][dupos][pos]RV_{src}，RV_{dst}”和“vmov[type][dupos][pos]RS_{src}，RV_{dst}”描述。对于前者，通过支持不同的向量类型，指定复制的源向量中的元素位置 dup，并指定复制后的两份向量元素移动到结果向量 RV_{dst} 中的位置 dupos，就可以实现该类指令设计空间的遍历。对于后者，dupos 特征码仅起到区分指令类型的作用(被复制的元素由于是来自一个标量寄存器中的一个量，不存在指定被复制元素位置的问题)，复制后移动到结果向量 RV_{dst} 中的哪两个位置由特征码 pos 决定，可以根据穷举出 pos 的所有可能来实现该类指令的设计空间遍历。

8)辅助指令(V-AU)

辅助指令用指令模型“v[stat][op][VSTAT]”来描述。这类指令通常根据 SIMD 部件的控制方式和状态来设计，理论上设计空间无穷大；但是，根据实际控制需要，通常设计实现的指令非常少。

4.2 SIMD 指令集设计的其他考量

根据前面对 SIMD 指令的分类，以及对每类指令的指令功能和指令模型的定义，并结合对每类指令设计空间的遍历说明，可以清晰地理清本文对整个 SIMD 指令集设计空间的描述思想和实现。本文定义的指令类和指令模型可以作为 SIMD 指令集的原子模式，更复杂的指令可以通过结合上述指令中的多条指令实现。

在指令集设计实现中，可以参考本文的分类方法和模型描述，结合指令集体系结构的特点以及 SIMD 计算单元的功能需求，设计实现实用的、可成长的指令集合。另外在指令集设计中，存储器对向量访存地址是否对齐有很强的约束[5]，但是这属于实现约束问题，与指令集本身的抽象描述无关。因此，可以在本文定义的指令集模型的基础上，结合物理实现约束定义相应的特定 SIMD 指令。如在 Intel 的 SIMD 指令集中，对于向量访存指令就定义了对齐访问和非对齐访问两种。另外，具体设计 SIMD 指令集时，还必须考虑指令编码长度、编码空间等物理实现约束。

总之，应该按照先抽象再具体的步骤设计实现 SIMD 指令集，本文的工作对在抽象层面开展 SIMD 指令集的分析和设计工作提供了有益的指导。

结束语 处理器指令集作为处理器软件与硬件之间的界面，在处理器体系结构设计中处于特别重要的地位。处理器的指令集设计是非常复杂的工作，涉及到应用目标、向下兼容需求、体系结构实现复杂度等多方面的考量。迄今为止，广泛使用的高性能通用处理器的指令集都经过了十余年甚至数十年的发展。

在通用处理器中扩展 SIMD 体系结构，已经成为处理器芯片内并行体系结构发展的一条重要的技术路线。面向 SIMD 体系结构，设计 SIMD 指令集也同样非常复杂。本文面向通用高性能处理器扩展 SIMD 体系结构的需求，基于普遍的 SIMD 体系结构模型，采用指令分类并为每类指令建立指令模型的方法，描述了 SIMD 指令集的设计空间，并讨论了 SIMD 指令集设计空间遍历和指令集设计问题。

本文的工作为 SIMD 指令集的分析和设计提供了比较好的思路，对 SIMD 指令集的设计甚至 SIMD 体系结构的设计都有指导意义。

参 考 文 献

[1] Maruyama T, Motokurumada T, Morita K, et al. Past, Present and Futures of SPARC64 Processors[J]. FUJITSU Sci. Tech. J., 2011, 47(2): 130-135

[2] Firasta N, Buxton M, Jinbo P, et al. Intel AVX: New Frontiers in Performance Improvements and Energy Efficiency[M]. Intel white paper, 2008

[3] Intel 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes: 1, 2A, 2B, 3A and 3B[M]. Intel Corporation, May 2011

[4] Hennessy J L, Patterson D A. Computer Architecture: A Quantitative Aproach(3rd ed)[M]. San Francisco: Morgan Kaufmann Publish, 2002

[5] Nuzman D, Henderson R. Multi-platform Auto-vectorization[C]// Proc. of the 4th Annual International Symposium on Code Generation and Optimization (CGO). March 2006