2024 3rd International Conference on Cloud Computing, Big Data Application and Software Engineering (CBASE) | 979-8-3315-0658-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/CBASE64041.2024.10824427

Research on Vector Extension of Instruction Set Architecture

Jiangdong Wang*
Beijing Microelectronic Technology Institute
BMTI
Beijing, China
*e-mail: 1820293109@qq.com

Lixin Yu
Beijing Microelectronic Technology Institute,
BMTI
Beijing, China
e-mail: yulx2024@163.com

Wei Zhuang
Beijing Microelectronic Technology Institute,
BMTI
Beijing, China
e-mail: zhwei09@126.com

Xue Yang
Beijing Microelectronic Technology Institute,
BMTI
Beijing, China
e-mail: 15911106595@163.com

Shiyuan Zhang
Beijing Microelectronic Technology Institute
BMTI
Beijing, China
e-mail: zhangshiyuan@buaa.edu.cn

Zhiyong Qin
Beijing Microelectronic Technology Institute,
BMTI
Beijing, China
e-mail: qinzy2010@163.com

Abstract—**With the development of instruction set architecture, vector extension has become an important way to improve processor performance. The vector instruction set adds support for single instruction multiple data (SIMD) operations by extending the base instruction set, so that the processor can use one instruction to perform the same operation on multiple data elements. The extension of this instruction set greatly improves the parallelism of data processing. It has brought significant system performance improvement for scientific computing, multimedia processors and other fields. With the wide application of vector instructions in various fields, different instruction set architectures have developed their own unique vector instruction sets. This paper surveys the vector instruction set of three architectures: RISC-V, ARM, x86. Analyze the characteristics of three vector instruction sets, and they provide developers with rich parallel processing capabilities to meet the needs of different application scenarios. The purpose of this paper is to study and compare the vector instruction sets of RISC-V, ARM and x86, to provide reference for the selection and optimization of processor architecture.**

Keyword-vector instruction set; RISC-V; ARM; x86

I. INTRODUCTION

With the rapid development of information technology, the demand for high-performance computing is increasing. As the basis of processor design and implementation, instruction set architecture (ISA) is one of the main factors affecting computing performance. High-performance instruction set design has become a hot research topic in the field of computer science. RISC-V, ARM, and x86 occupy a pivotal position in the industry with their unique advantages in many instruction set architectures. In order to meet the needs of high-performance computing, the three architectures have introduced vector instruction sets to improve the parallel ability of the processor.

As a new instruction set architecture, RISC-V architecture has attracted wide attention due to its open, concise and scalable characteristics[1]. RISC-V vector instruction set (RVV) is designed to provide efficient parallel processing capabilities while maintaining the simplicity of the architecture. Through modular design, RVV enables developers to select the appropriate vector length according to their needs, so as to achieve flexible expansion.

In order to meet the needs of high-performance multimedia processing and machine learning applications, ARM introduced the Neon vector instruction set. The Neon instruction set has rich data types and operators, which can effectively improve the data processing speed. The Neon instruction set continuously optimizes performance while maintaining compatibility, so that the ARM processor has broad application prospects in mobile devices and embedded systems.

As the mainstream architecture of the desktop and server market, the advanced vector extension (AVX) instruction set of x86 has significant advantages in scientific computing, big data processing and other fields. The AVX instruction set greatly improves the parallel processing ability of the processor by expanding the width of the register and increasing the vector operation ability.

II. RISC-V VECTOR INSTRUCTION

RISC-V instruction set includes basic instruction set and extended instruction set[2]. The extended instruction set of RISC-V is used to provide specific functional operation instructions for ISA. An ISA can choose to add multiple extended instruction sets. In order to enable multiple instruction sets to coexist, the coding space of each instruction set is divided according to the coding requirements of RISC-V

International Foundation (RVI) to avoid conflicts. There are mainly 33 kinds of RISC-V extended instruction sets.

RISC-V enriches the functions of the instruction set by extending the instruction categories of specific functions, where the vector operation extension (' V ' extension) is used to support data parallel execution in the 32-bit instruction coding space. The extension enhances the SIMD[3] operation of the processor, introduces a set of vector registers and vector operation instructions, and allows the same operation to be applied to quantities of data elements, so that the data processing capability of the RISC-V system is qualitatively changed, while maintaining the simplicity of the instruction code. The vector extension to the basic scalar RISC-V adds 32 vector registers and 7 unprivileged Control and Status Registers (CSRs), as shown in Table 1.

TABLE 1 New vector CSR

Name	Address	Description
vstart	0x008	Vector start position
vxsat	0x009	Fixed-Point Saturate Flag
vxrm	0x00A	Fixed-Point Rounding Mode
vcsr	0x00F	Vector control and status register
vl	0xC20	Vector length
vtype	0xC21	Vector data type register
vlenb	0xC22	VLEN/8 (vector register length in bytes)

A. Vector instruction format

The vector extension instruction is applicable to two existing major opcodes (LOAD-FP and STORE-FP) and a new major opcode (OP-V). Vector loads and stores are encoded in the scalar floating-point loads and stores main opcode LOAD-FP / STORE-FP. The vector arithmetic instruction uses a new main opcode OP-V. The vector instruction set format provided by RISC-V is shown in Fig. 1, Fig. 2 and Fig. 3.

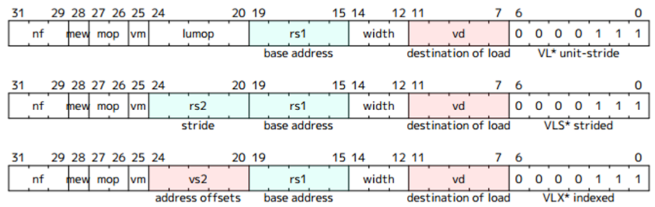


Figure 1 Format for Vector Load Instructions under LOAD-FP major opcode

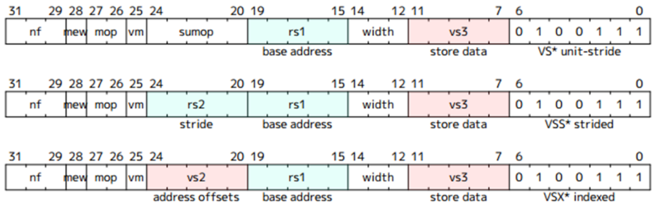


Figure 2 Format for Vector Store Instructions under STORE-FP major opcode

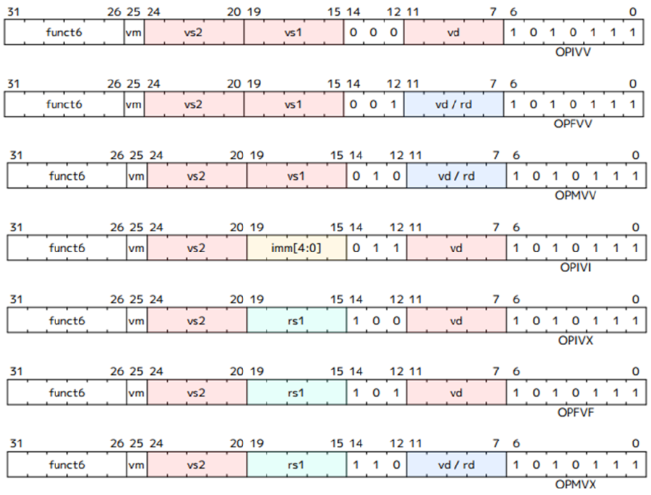


Figure 3 Format for Vector Arithmetic Instructions under OP-V major opcode

Vector instructions can use scalar or vector source operands and produce scalar or vector results. Most vector instructions can be executed unconditionally under masks. Vector loading and storage move between vector register elements and memory. The vector arithmetic instruction manipulates the values stored in the vector register element.

B. Introduction of vector instructions

Vector instructions can be divided into the following categories according to their specific purposes and operating characteristics when performing data processing tasks: Configuration-Setting Instructions, Vector Loads and Stores Instructions, Vector Integer Arithmetic Instructions, Vector Fixed-Point Arithmetic Instruction, Vector Floating-Point Instructions, Vector Reduction Instructions, Vector Mask Instruction and Vector Permutation instruction.

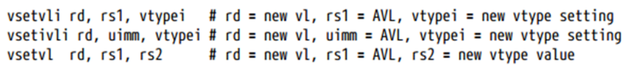
In addition, the encryption instruction is also a vector instruction. However, in the RISC-V instruction set, it is used as a separate extension category and is not in the ' V ' extension.

1) Configuration-Setting Instructions

Configuration-Setting Instructions is used to initialize and configure the environment for vector processing, ensuring that vector operations can be performed as expected. Programmers can flexibly configure the parameters of vector processing through these instructions to meet different computing needs.

One of the common methods for processing a large number of elements is ' stripmining ', that is, processing a certain number of elements at a time, iteratively, until all elements are processed. RISC-V vector specification (vector element size, vector instruction format, etc.) provides direct and portable support for this method. The application specifies the total number of elements to be processed (AVL) as the candidate value of vl. The hardware implements and vtype settings according to the micro-architecture, and responds to the hardware through a common register. The number of elements processed (stored in vl) in each iteration.

Through the vset(i)vl(i) instruction, the fast configuration of vl and vtype is realized to match the needs of the application, as shown in Fig. 4.



```
vsetvli rd, rs1, vtypei   # rd = new vl, rs1 = AVL, vtypei = new vtype setting
vsetivli rd, uimm, vtypei # rd = new vl, uimm = AVL, vtypei = new vtype setting
vsetvl  rd, rs1, rs2      # rd = new vl, rs1 = AVL, rs2 = new vtype value
```

Figure 4 vset (i) vl (i) instruction configuration vl and vtype

2) Vector Loads and Stores Instructions

Vector loads and stores move values between vector registers and memory. Vector loads and stores are encoded in the scalar floating-point loading and storage main opcode (LOAD-FP / STORE-FP). The vector instruction format is shown in Fig. 1 and Fig. 2. Vector loads and stores are encoded using standard scalar floating-point loading and storage of undeclared width values, as shown in Table 2.

Mem bits is the size of each element accessed in memory. Data reg bits is the size of each data element accessed in the register. The mew is used when the extended memory size of 128 bits and above is set, and these codes are currently retained.

TABLE 2Width encoding for vector loads and stores

Name	mew	Mem bits	Data reg bits
Standard scalar FP	x	16	FLEN
Standard scalar FP	x	32	FLEN
Standard scalar FP	x	64	FLEN
Standard scalar FP	x	128	FLEN
Vector 8b element	0	8	8
Vector 16b element	0	16	16
Vector 32b element	0	32	32
Vector 64b element	0	64	64
Vector 8b element	0	SEW	SEW
Vector 16b element	0	SEW	SEW
Vector 32b element	0	SEW	SEW
Vector 64b element	0	SEW	SEW
Reserved	1	-	P

3) Vector Integer, Fixed-Point, Floating-Point Instructions

Vector integer arithmetic instructions allow the processor to perform the same operation on multiple data elements at the same time, make full use of data-level parallelism, improve data processing speed, and make the processor more suitable for processing data-intensive tasks. Vector integer arithmetic instructions include the following instructions: vector single-width integer add/subtract instructions, vector widening integer add/subtract instructions, vector integer add-with-carry /subtract instructions, vector bit logic instructions, vector integer minimum/maximum instructions, vector single-width integer multiply/divide instructions, etc.

Vector integer arithmetic instructions are divided into three types: vector-vector, vector-scalar and vector-immediate number. In the above instructions, the vector subtract instructions, the vector widening integer add and subtract instructions, the vector integer divide instructions and the vector widening integer multiply instructions have no vector-immediate number.

The vector fixed-point arithmetic instructions extend the above integer instruction set to support fixed-point operations. Fixed-point instructions maintain the accuracy of narrow operands by supporting scaling and rounding, and can handle overflows by saturating the results to the target format range. Vector fixed-point arithmetic instructions include: vector single-width saturating add/subtract, vector single-width averaging add/subtract, vector single-width fractional multipy with rounding and saturation, vector single-width scaling shift instructions, etc.

The current vector floating-point instruction set includes support for 32-bit and 64-bit floating-point values. When 16-bit and 128-bit element widths are added, they will also be treated as IEEE-754 / 2008 compatible values. Vector floating-point instructions include: vector single-width floating-point add/ subtract instructions, vector single-width floating-point multiply/divide instructions, vector floating-point square root instructions, vector floating-point reciprocal square root estimate instructions, vector floating-point min/max instructions, vector floating-point comparison instructions, single-width floating-point/integer type-convert instructions, etc.

4) Other Vector Instructions

In addition to the above several vector instructions, there are Vector Reduction Instructions, Vector Mask Instruction and Vector Permutation instruction.

Vector reduction operations take a vector register group of elements and a scalar held in element 0 ofa vector register, and perform a reduction using some binary operator, to produce a scalar result in element 0 of a vector register.The scalar input and output operands are stored in the element 0 of a single vector register, not in a vector register group. Vector reduction instructions include: vector single-width integer reduction instructions, vector widening integer reduction instructions, vector single-width floating-point reduction instructions and vector widening floating-point reduction instruction, etc.

Vector mask instructions are provided to help operate on mask values held in a vector register,including: vector mask register logic instructions, vector element index instruction, etc.

Vector permutation instructions are a series of permutation instructions provided to move elements in vector registers, including: integer scalar move instructions, floating-point scalar move instructions, vector register gather instructions, vector compression instruction and whole vector register move, etc.

C. Summary

RISC-V vector instruction set has the characteristics of modularization, configurability and high compatibility. The design idea is as follows: First, define a set of basic vector instructions, including vector loading, storage, arithmetic, logic operations, etc. These instructions can cover the basic needs of most vector processing. Secondly, the vector instruction set can dynamically set the parameters of vector processing (vector length, scalar element width, etc.) by configuring instructions to adapt to different application requirements. In addition, the vector extension is compatible with the existing RISC-V instruction set, ensuring that the new vector instruction does not destroy the existing software ecosystem. RISC-V encourages third parties to expand and customize vector instruction set according to specific application requirements, so as to promote the development and innovation of RISC-V instruction set architecture.

III. ARM VECTOR INSTRUCTION

By utilizing data parallelism, ARM vector instructions significantly improve the performance of multimedia processing, signal processing and other fields. With the update of the ARM architecture version, the vector instructions are also increasing. Table 3 shows the main vector extensions of the ARM v8-ARM v9 version. In the ARM architecture, the vector instructions of ARM v8 are mainly Neon instructions. The continuous iteration of the v8 version increases the expansion of the encryption algorithm, the scalable vector instructions, etc., and supplements the basic vector instructions. In ARM v9, the scalable vector extension version is updated and more encryption algorithms are supported. In addition, the scalable matrix extension is added and the version is updated[4]. The continuous evolution of ARM vector instructions has greatly improved performance and flexibility when dealing with vector-intensive tasks.

TABLE 3ARM v8-ARM v9 version vector extension

Version	Description
ARM v8	encryption extension
	scalable vector extension
	Floating point half-precision multiplication instruction
	Single precision matrix multiplication（FEAT_F32MM）
	Single precision matrix multiplication（FEAT_F64MM）
	Float point complex instruction
ARM v9	Scalable vector extension version 2(SVE2)
	Scalable vector AES instruction
	Scalable vector bit arrangement instruction
	Scalable vector PMULL instruction
	Scalable vector SHA3 instruction
	Scalable vector SM4 instruction
	Scalable Matrix Extension (AME)
	Scalable Matrix Extension 2 (SME2)
	Scalable matrix extension version 2.1 (SME2.1)
	Scalable vector extension version 2.1 (SVE2.1)

The development of vector instructions has gone through many stages, the most representative of which is the Neon instruction. Neon instruction set brings high-performance SIMD operations to ARM processors. On this basis, ARM further introduced Scalable Vector Extension (SVE) and Scalable Matrix Extension (SME) instruction sets, which not only enhance vector processing capabilities, but also provide more flexible and efficient solutions for various application scenarios.

A. Neon

Neon is the implementation of advanced SIMD extension of ARM[5]. All processors that conform to the ARMv8-A or ARMv9-A architecture include Neon. Neon provides 32 128-bit vector registers, each register has multiple data channels, and SIMD instructions operate simultaneously on these multi-channel data to accelerate the operation. Neon instruction set plays an important role in improving the performance of multimedia and signal processing, 3D graphics, image processing and other applications.

In the AArch64 state, ARM v8 is a 64-bit architecture and uses 64-bit registers. However, in order to improve parallelism and obtain better performance, the Neon unit uses 128-bit registers for SIMD processing. The Neon unit is fully integrated into the processor and shares processor resources for integer operations, loop control, and caching. Compared with the hardware accelerator, this greatly reduces the area and power cost. As shown in Fig. 5, the 128-bit Neon vector can contain 16 8-bit elements, 8 16-bit elements, 4 32-bit elements or 2 64-bit elements. The 64-bit Neon vector can contain 8 8-bit elements, 4 16-bit elements or 2 32-bit elements.

There are about 400 Neon vector instructions in ARM v8. In order to improve the speed of data operation, the basic vector operation instructions and data type conversion instructions are extended. In addition, a series of encryption instructions are extended to improve the speed of encryption operations.

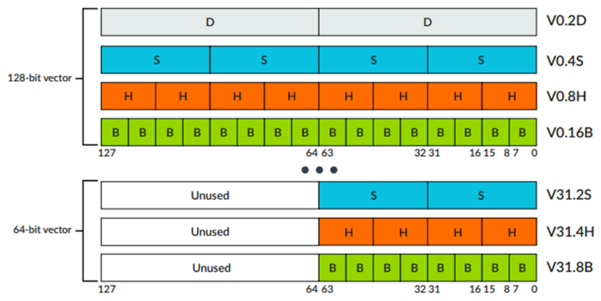


Figure 5 64-bit and 128-bit Neon vector data

B. SVE

SVE is the next generation SIMD extension of ARMv8-A instruction set[6]. It is not an extension of Neon, but a new set of vector instructions developed for high performance computing (HPC) workloads. Unlike other SIMD architectures, SVE can be vector length unknown (VLA). VLA means that the size of the vector register is not fixed, and the hardware implementer can freely choose the vector size that is most suitable for the expected workload. SVE contains instructions that allow vector code to automatically adapt at run time to implement vector length. There are about 900 ARM SVE vector instructions. It is mainly divided into register and state instructions, data loading and storage instructions, mathematical operation instructions, logical operation instructions, conversion and type instructions, etc. It introduces many new architectural features, such as variable vector length, per-channel prediction, aggregated loading and decentralized storage, and horizontal operations. These features enable SVE to handle more complex data structures and algorithms, and support longer vector operations up to 2048 bits.

Neon instructions always operate on 64-bit or 128-bit vectors. In SVE, the instruction set operates on a new set of vector and predicate registers: 32 Z registers, 16 P registers and a first fault register (FFR). P registers and FFR registers are unique to SVE.

SVE2 is a superset of SVE and Neon. SVE2 allows more function domains in data-level parallelism. SVE2 inherits the concept, vector register and operation principle of SVE. In addition to extensible vectors, SVE and SVE2 include: per-channel prediction; collect loading and distributed storage; speculated vectorization. These features help vectorize and optimize loops when dealing with large data sets.

The main difference between SVE2 and SVE is the functional coverage of the instruction set. SVE is designed for HPC and ML applications. SVE2 extends the SVE instruction set, making the field of data processing beyond HPC and ML. Both SVE and SVE2 can collect and process large amounts of data. SVE and SVE2 are not extensions of Neon instruction set. On the contrary, SVE and SVE2 are redesigned to achieve better data parallelism than Neon. However, the hardware logic of SVE and SVE2 covers the Neon hardware implementation. When microarchitecture supports SVE or SVE2, it also supports Neon. Each scalable vector register, Z0-Z31, can be 128-2048 bits, 128-bit increments. The bottom 128 bits are shared with Neon 's fixed 128-bit long V0-V31 vector, as shown in Fig. 6.

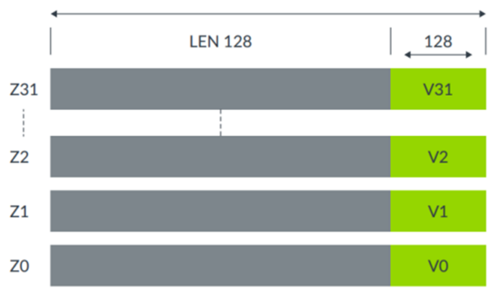


Figure 6 Scalable vector registers

In SVE2, many instructions are added to copy the existing instructions in Neon, including converted Neon integer operation, converted Neon widening, narrowing and pairing operation. SVE2 can optimize emerging applications outside the HPC market, such as machine learning, computer vision (TBL and TBX instructions), baseband networks (CADD and CMLA instructions), genomics (BDEP and BEXT instructions) and servers (MATCH and NMATCH instructions). SVE2 enhances the overall performance of the general-purpose processor 's large-capacity data operations without the need for other off-chip accelerators.

Compared with Neon, the advantages of SVE2 include: SVE2 programs can be built at one time and run on hardware with various vector lengths; SVE2 has more vectorization flexibility; SVE2 extends the instruction set so that it can be applied to more fields.

C. SME

Scalable Matrix Extension (SME)[7] is introduced into ARMv9-A to improve the efficiency and performance of matrix operations. It adds a new architecture state: ZA storage; it also introduces a new execution mode: streaming SVE mode, which can execute new SME instructions and SVE2 instruction subsets. SME instructions operated on ZA include: external product calculation in ZA tiles; inserting/extracting ZA tiles in the Z vector register; load and store ZA tiles.

SME instructions support matrix operations, such as multiplication, inversion and dynamic transpose. These instructions play an important role in digital filtering, linear equation solving and convolution.

Scalable Matrix Extension 2 (SME2) extends the SME architecture. SME2 expands SME by introducing multi-vector data processing instructions, multi-vector loading and multi-vector storage, and multi-vector prediction mechanisms. There are about 300 ARM SME vector instructions. SME is based on SVE2 and adds the ability to process matrices efficiently. Its key functions include: calculating the outer product of two SVE vectors; storage of matrix tiles; access the vectors in the matrix block, insert the vectors into the matrix block and extract the vectors in the matrix block, including the on-the-fly matrix transpose; streaming SVE mode.

The characteristics of SME2 include:

- Multi-vector multiplication accumulation instruction, with Z vector as multiplier and multiplier input, and the results are accumulated into the ZA array vector.
- Multi-vector loading, storing, moving, arranging and converting instructions, using multiple SVE Z vectors as source and target registers to preprocess the input and post-processing output ZA target SME2 instructions.
- 'Predicate as counter', an alternative prediction mechanism is added to the original SVE prediction mechanism to control the operation performed on multiple vector registers.
- Compressed neural network function using special look-up table instructions and external product instructions that support binary neural networks.

D. Summary

The vector instruction extension of ARM architecture is mainly reflected in Neon, SVE and SME. The parallelism of ARM vector instructions is similar to RISC-V, which aims to improve performance through parallel processing, especially when dealing with multimedia and signal processing tasks. The subsequent introduction of SVE provides a scalable vector length that can be dynamically adjusted according to the implementation of the processor to optimize performance and power consumption. In order to improve the efficiency and performance of matrix operations, a scalable matrix extension (SME) is introduced into ARMv9-A.

IV. X86 VECTOR INSTRUCTION

The x86 vector instruction also performs parallel processing of multiple data through a single instruction[8], which greatly improves the computational performance. x86 uses a complex instruction set, which is the biggest difference from the other two architectures, and its vector instruction set was first developed. Since the introduction of the first SIMD instruction set MMX in 1997, Intel has continuously promoted the development of vector instruction set. After SSE, AVX and other stages, it has brought revolutionary improvements to scientific computing, multimedia processing and other fields. Today, AVX-512, as the latest member of the Intel vector instruction set, further expands the 512-bit vector processing capability, which greatly improves the performance of the processor when dealing with high-density data computing tasks.

A. Vector instruction development

The development of x86 vector instructions mainly includes the introduction and extension of instruction sets such as MMX, SSE, SSE2, SSE3 and AVX[9].

Multimedia extensions (MMX): In 1996, Intel introduced the first generation of SIMD instruction set MMX, which is mainly used on Pentium processors, and then AMD added MMX instruction support on K6 processors. MMX provides three types of integer values, including 64-bit packaged byte integers, 64-bit packaged integers, and 64-bit packaged two-character integers. These data types need to use FPU registers to save and perform mathematical operations. The MMX instruction improves the ability to process multimedia and data parallelism. The instruction usually starts with ' P ', and the use of MMX gradually decreases with the emergence of SSE and AVX.

Streaming SIMD Extensions (SSE): In 1999, Intel introduced the SSE instruction set, which was first applied to the Pentium III processor. SSE mainly performs SIMD operations on floating-point data, and introduces new data types, such as 128-bit packaged single-precision floating-point data types. The use of 128-bit XMM registers has important implications in the fields of scientific computing, multimedia processing, and game development, laying the foundation for AVX and AVX512 instruction sets.

Streaming SIMD Extensions Second Implementation (SSE2): In 2001, Intel launched the SSE2 instruction set. SSE2 extends the core architecture of SSE and introduces more data types, including 128-bit packaged double-precision floating-point value, 128-bit packaged byte integer value, 128-bit packaged word integer value, 128-bit packaged double-word integer value and 128-bit packaged four-word integer value.

SSE3: SSE3 adds data advanced processing instructions on the basis of SSE2, further enhancing SIMD processing capabilities.

Advanced Vector Extensions (AVX)[10]: In 2011, Intel launched the AVX instruction set, providing more advanced vector processing capabilities, supporting a wider range of vector operations and higher performance. Through the AVX2, AVX-512 version, the register is extended to 256 bits and 512 bits, and new instructions such as conditional execution, vector loading storage and operation are added.

B. AVX-512

AVX-512 is Intel 's latest SIMD instruction set. It is a richer and more flexible instruction set. It introduces many new concepts, such as mask operation, broadcast, instruction set extension, register mode and bit ternary instruction. AVX-512 extends the width of the existing AVX2 SIMD register from 256 bits to 512 bits, thus doubling the amount of work that can be completed in a single operation.

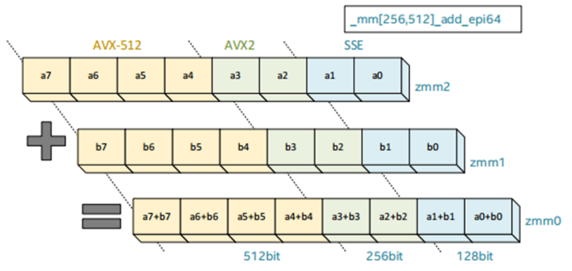
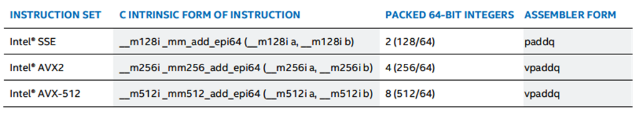


Figure 7 SSE, AVX2, AVX-512 addition operation



INSTRUCTION SET	C INTRINSIC FORM OF INSTRUCTION	PACKED 64-BIT INTEGERS	ASSEMBLER FORM
Intel® SSE	__m128i _mm_add_epi64 (__m128i a, __m128i b)	2 (128/64)	paddq
Intel® AVX2	__m256i _mm256_add_epi64 (__m256i a, __m256i b)	4 (256/64)	vpaddq
Intel® AVX-512	__m512i _mm512_add_epi64 (__m512i a, __m512i b)	8 (512/64)	vpaddq

Figure 8 Comparison of 64-bit SSE, VX2, AVX-512 instruction sets

Taking vector addition as an example, each time the instruction set is generated, the register size and the workload of execution are doubled, which is represented by the packaged integers a0 to a7. As shown in Fig. 7 and Fig. 8, a SSE, AVX2, AVX-512 vector addition instruction is used to complete the addition operation. Taking vector addition as an example, each time the instruction set is generated, the register size and the workload of execution are doubled, which is represented by the packaged integers a0 to a7. As shown in Fig. 6 and Fig. 7, a SSE, AVX2, AVX-512 vector addition instruction is used to complete the addition operation.

SSE uses 128-bit registers, each of which contains two packaged 64-bit integers, thus completing two different 64-bit additions in one instruction.

AVX2 extends integer vector addition to 256-bit registers. Four different 64-bit additions are completed in a single instruction.

AVX-512 extends the packaged integer addition to 512-bit registers, completing eight different 64-bit additions in a single instruction.

C. The development of AVX-512

AVX-512 is not a single instruction set, but an improvement and extension set of each generation of Intel scalable processors.

The first-generation Intel Scalable Processor introduces the following extensions:

- AVX-512 Basic (AVX512F): AVX-512 's basic instruction set, which extends most AVX functionality to support 512-bit registers.
- Conflict detection (AVX512CD): It is mainly used to detect data conflicts and optimize memory access.
- Vector length (AVX512VL): Allows the use of shorter vector lengths (128-bit and 256-bit) to improve code flexibility and efficiency.
- Double-character four-character (VX512DQ): Extend double-precision and four-character operations to improve the ability to process double-precision floating-point numbers and integers.
- Byte and word (AVX512BW): For byte and word operations, improve the efficiency of processing 8-bit and 16-bit integers.

The second-generation Intel Scalable Processor introduces the following extensions:

- Vector neural network (AVX-512_VNNI): designed to accelerate the design based on convolutional neural network (CNN). By introducing new instructions, the internal loop of the convolutional neural network is accelerated.

The third-generation Intel Scalable Processor introduces the following extensions:

- Vector byte operations (AVX512_VBMI and AVX512_ VBMI2): For bit operations, allowing efficient bit operations on 512-bit vector registers, suitable for encryption, data compression and other bit operation-intensive applications.
- Vector two-word and four-word bit counting (VPOPCNTDQ): It is used to calculate the number of bits in integer values and is suitable for encryption applications.
- Vector AES (VAES) and Galois Field (GFNI): VAES provides hardware acceleration for Advanced Encryption Standard (AES) algorithms and is suitable for encryption and decryption operations. GFIN is used to perform Galois field operations and is suitable for encryption algorithms and coding theory applications.

D. Summary

As x86 vector instructions continue to evolve, the register width has increased from the initial 128 bits (SSE) to 256 bits (AVX) and 512 bits (AVX-512), and the number of registers has also increased to handle more data at the same time. x86 designs special instructions for specific types of applications (such as image processing, encryption, deep learning, etc.). However, these extensions also bring some compatibility problems. Most extensions maintain compatibility, but some advanced features may require newer operating system and compiler support. The extension of vector instructions in x86 opens up a new path for the improvement of processor performance. Especially when dealing with complex data-intensive tasks, the extension of vector instructions has achieved remarkable results in neural networks, image processing, encryption and decryption algorithms.

V. Vector instruction comparison

Taking the multiplication of two fourth-order matrices with single-precision floating-point data as an example, if ordinary instructions are used, 64 multiplication operations, 48 addition operations, and 48 data access operations are required, which requires a lot of time. The comparison of the three architectures using vector instructions is shown in table 4.

TABLE 4 Comparison of three architecture

	RISC-V	ARM	x86
Vector register bit width	256	128	512
Number of data processed	8	4	16
Number of vector load and store	12	12	3
Number of vector arithmetic instruction	28	16	4

RISC-V vector instruction, a 256-bit vector register can process 8 single-precision floating-point numbers at one time, and three matrix data can be accessed by sharing 12 accesses. There is no vector multiply-add instruction in RISC-V, and 16 vector multiplication instructions and 12 vector addition instructions are required to complete the multiplication of two matrices.

ARM vector instruction, a 128-bit vector register can process four single-precision floating-point numbers at one time, and three matrix data can be accessed by sharing 12 accesses. With 16 vector multiply-add instructions, the multiplication of two fourth-order matrices can be completed.

The 512-bit vector register of the AVX-512 instruction vector instruction can handle up to 16 single-precision floating-point numbers at one time. In theory, three access instructions and four vector multiply-add instructions can be used to complete the fourth-order matrix multiplication operation. However, the specific number of instructions will be affected by many factors, such as loop unrolling, data prefetching, register usage and so on.

VI. Conclusions

The vector instruction sets of RISC-V, ARM and x86 all adopt the expansion idea of improving execution efficiency through parallel execution, but they are different in specific instruction set implementation, functions and application scenarios.

The vector instruction set of x86, such as SSE, AVX, etc., is widely used in the fields of PC and server, and provides strong software ecological support.

The vector instruction set of RISC-V is open source, flexible and scalable, and can be customized according to different application requirements.

The vector instruction set of ARM, such as SVE and NEON, play an important role in mobile and embedded devices due to their high efficiency and energy efficiency.

The differences of the three architecture vector instructions are compared as shown in table 4 below. In addition to the basic SIMD instructions, ARM and x86 both extend the matrix operation to provide special matrix operation support for high performance computing (HPC) and machine learning. However, RISC-V does not have a special matrix extension, and the matrix operation is performed through the ' V ' extension instruction. ARM and x86 vector instruction users cannot extend the hardware level by themselves, but the RISC-V architecture allows users to extend vector instructions according to their needs. Both RISC-V and ARM (ARM v8.2 and later) support variable vector length vector operations, but the x86 vector instruction length is fixed. The comparison of three architecture vector instructions is shown in table 5.

TABLE 5 Comparison of three architecture vector instructions

Name	RISC-V	ARM	x86
Instruction set	RISC	RISC	CISC
User self-expansion	Support	Non-Support	Non-Support
vector length	Variable	Variable	immutable
Specialize in matrix operation	No	SME	AMX
Ecologically	Imperfect	Perfection	Perfection
power loss	Low	Low	High
Application	Intelligent wearable devices, etc. (low energy consumption)	Mobile communication (good scalability)	PC-side high-performance computing (strong processing power)

In summary, improving the parallelism of instruction execution is an extension of the general high-performance instruction set. The demand for computing in the fields of artificial intelligence and image processing is increasing, and the requirements for large-scale data processing speed will also continue to increase. The vector instruction expansion that improves the speed of matrix operations also needs to be continuously improved. In addition, with the wider application requirements, flexible and configurable vector instructions have always been one of the key directions of instruction set expansion.

References

[1] Li, F., Guo, S. (2024) Implementation of basic mathematics library for RISC-V. Acta Electronica Sinica, 52(05): 1633 - 1647.

[2] Liu, C., Wu, Y. (2021) A survey of RISC-V instruction set architecture. Journal of Software, 32 (12): 3992−4024.

[3] Xie, H., Xiao, Q. (2024) Vector instruction set and communication extension based on RISC-V architecture. Informatization-Research, No.1,2024: 89-90.

[4] ARM, 2024. ARM A-profile A64 Instruction Set Architecture. https://developer.ARM.com/documentation/ddi0602/2024-06/?lang=en.

[5] Qi, J., Cheng, Y. (2024) Matrix multiplication acceleration technology based on NEON parallel computing architecture. Aeronautical Computing Technique, 54 (03): 48-52.

[6] ARM, 2021. Learn the architecture-Migrate Neon to SVE. https://developer.ARM.com/Architectures/Scalable%20Vector%20Extensions.

[7] ARM, 2022.SME Programmer's Guide. https://developer.ARM.com/documentation/109246/0100/?lang=en.

[8] Li, D., Zhu, Z. (2024) Security vulnerability analysis of x86 processor vector conditional access instructions. Chinese Journal of Computers, 47 (03): 525-543.

[9] Intel, 2018. Intel® AVX-512 - Instruction Set for Packet Processing. https://www.intel.cn/content/www/cn/zh/developer/articles/technical/the-intel-advanced-vector-extensions-512-feature-on-intel-xeon-scalable.html?wapkw=avx512

[10] Ahmed, S. (2014) Enhancing the Matrix Transpose Operation Using Intel AVX Instruction Set Extension. AIRCC's International Journal of Computer Science and Information Technology. 6: 67-78.