



AltiVec™ Technology

- On May 7, 1998, Motorola disclosed a new technology which integrates into the existing PowerPC Architecture a new high bandwidth, parallel operation vector execution unit
- Motorola's new AltiVec™ technology expands the capabilities of PowerPC Microprocessors by providing leading-edge, general-purpose processing performance while concurrently addressing high-bandwidth data processing and algorithmic-intensive computations in a single-chip solution
- Branding of the new AltiVec technology defines and expresses Motorola's leadership and long-term commitment to the PowerPC Architecture and its customers



AltiVec is a trademark of Motorola, Inc.



AltiVec™ Technology in Publications

- “The ability to access a high-performance vector unit in addition to the integer and floating point units in the RISC core provides a leap in performance beyond the traditional advances in CPU clock frequency. . . In the end, AltiVec should have a profound impact on digital signal processing applications that require both flexibility and performance. The fact that it will be available in a high-performance general-purpose RISC processor is icing on the cake.” Richard Jaenicke, SKY Computers, Real Time Computing 5/98.
- Jim Turley, a senior editor at Microprocessor Report, said AltiVec is similar to MMX, but it doubles the amount of data that can be processed in the MMX technology. Turley said that as the PowerPC Architecture moves further away from the desktop, its network capabilities will become more important. “AltiVec makes PowerPC a much more interesting Alternative than it was just last week,” Turley said. “It really does bring quite a lot to the party.” Austin American Statesman 5/8/98.
- “Think of the chip as having a doorway in and out of the chip that’s 32 bits wide, but the hallways are 128 bits wide,” Turley offers. “Once data is moved out of (main memory) and into the chip, they can really swing lots of data around and do special number crunching, more so than other chips.” CNET 5/6/98.



MOTOROLA

Semiconductor Products Sector

 **DigitalDNA**
from Motorola



Technology Introduction

- Development of AltiVec technology was driven by customer demands for next-generation performance for high bandwidth applications
- Motorola's premier installed base and long term customer relationships in the networking and telecommunications markets provide early and widespread design win opportunities
- Combination of the powerful superscalar PowerPC microprocessor core with the new functionality provided by AltiVec technology on a single chip creates a unique new product class
- This new class of product will enable new microprocessor applications and performance levels through ultra-high bandwidth and parallel data manipulation
 - Provides leading edge general purpose processing performance while concurrently addressing high-bandwidth data handling processing and algorithmic intensive computations in a single chip solution
 - Targets networking, computing and other performance-driven applications
- The existing comprehensive tool vendor support for PowerPC Architecture will be leveraged to rapidly deliver full infrastructure support for the AltiVec technology
 - Motorola is working with leading tools providers to develop simulators, assemblers, linkers and compilers to assure full support for the AltiVec technology



MOTOROLA

Semiconductor Products Sector

 **DigitalDNA**
from Motorola



Goals of AltiVec

- Meet computational demands of networking infrastructure
- Enable faster, more secure encryption methods optimized for SIMD processing model
- Provide compelling performance on multimedia-oriented desktop computers, desktop publishing, and digital video processing
- Enable real-time processing of the most demanding data streams (MPEG-2 decode, continuous speech recognition, real-time, high-resolution 3D graphics . . .)



What is a Vector Architecture?

- A **vector architecture** allows the simultaneous processing of many data items in parallel
- Vector architecture has roots in supercomputing, which attempted to extract large amounts of parallelism from software
 - Massive parallel capabilities but limited data types
 - Great for computation intensive applications but not for systems requiring more diverse processing and real-time constraints
- Operations are performed on multiple data elements by a single instruction – referred to as Single Instruction Multiple Data (SIMD) parallel processing
- AltiVec technology is a short vector architecture
 - Uses 128-bit wide registers to provide 4-, 8-, or 16-way parallelism
 - Supports a wide variety of data types
- SIMD extension to PowerPC ISA
 - Processes multiple data streams/blocks in a single cycle
 - Common approach to accelerate processing of next-gen data types (audio, video, packet data)



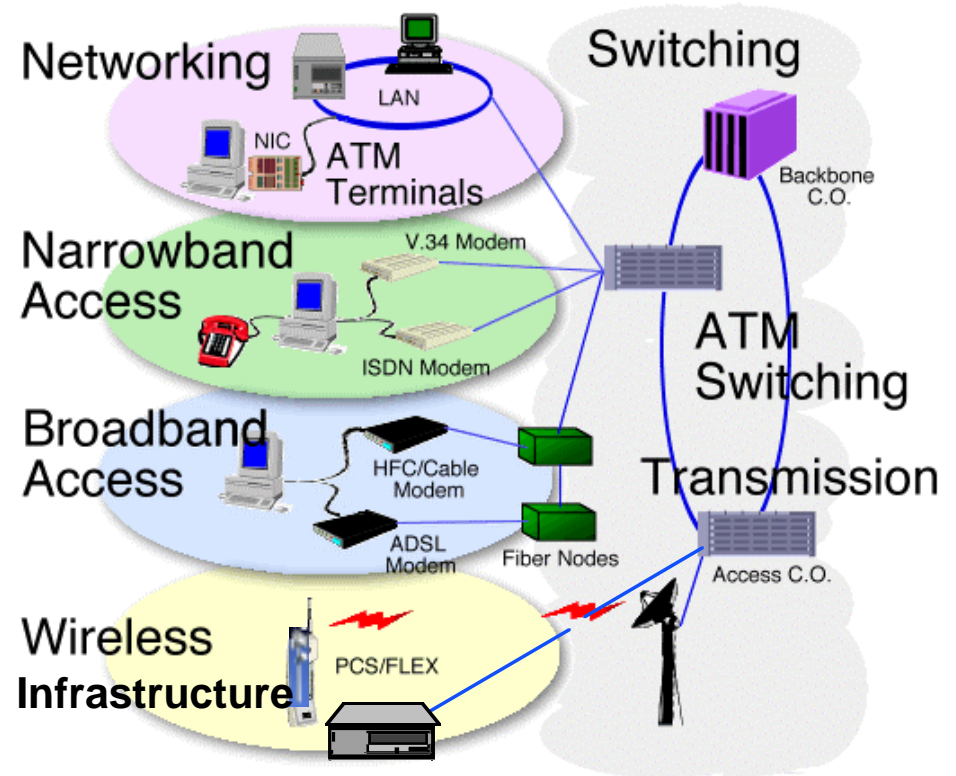
Benefits of AltiVec

- Provides a single high-performance RISC microprocessor with DSP-like compute power for controller and signal processing functions
 - Supplements the performance-leading PowerPC architecture with a new class of execution unit
 - New vector processing engine provides for highly parallel operations, allowing for the simultaneous execution of up to 16 operations in a single clock cycle
 - Can accelerate many traditional computing and embedded processing operations with its wide datapaths and wide field operations
- Provides product designers and customers with a new “one part – one code base” approach to product design while also providing a tremendous jump in performance
- Offers a programmable solution that can easily migrate via software upgrades to follow changing standards and customer requirements
 - Because this integrated solution is still 100% compatible with the industry standard PowerPC architecture, design and support are simplified
 - Leverage PowerPC compatibility and legacy code, add AltiVec performance as you need it



AltiVec – A Solution to Many Embedded Computing Problems

- Access Concentrators/DSLAMs
 - ADSL and Digital Data Concentrators
- Speech Recognition
- Voice/Sound Processing
- Image and Video Processing
- Array Numeric Processing
- Basestation Processing





AltiVec – The Best Solution to Computing Problems

- High bandwidth data communications
- Realtime Continuous Speech I/O
 - HMM, Viterbi Acceleration, Neural Algorithms
- Soft-Modem
 - V.34, 56K
- 3D Graphics
 - Games, Entertainment
 - High precision CAD
- Virtual Reality
- Motion Video
 - MPEG2, MPEG4
 - H.234
- High Fidelity Audio
 - 3D Audio, AC-3
- Machine Intelligence



MOTOROLA
Semiconductor Products Sector

 **DigitalDNA™**
from Motorola



AltiVec Features

- Provides SIMD (Single Instruction, Multiple Data) functionality for embedded applications with massive data processing needs
- Key elements of Motorola's AltiVec technology include:
 - 128-bit vector execution unit with 32-entry 128-bit register file
 - Parallel processing with Vector permute unit and Vector ALU
 - 162 new instructions
 - New data types:
 - » Packed byte, halfword, and word integers
 - » Packed IEEE single-precision floats
 - Saturation arithmetic
- Simplified architecture
 - No interrupts other than data storage interrupt on loads and stores
 - No hardware unaligned access support
 - No penalty for running AltiVec and standard PowerPC instructions simultaneously
 - Streamlined architecture to facilitate efficient implementation
- Maintains PowerPC architecture's RISC register-to-register programming model

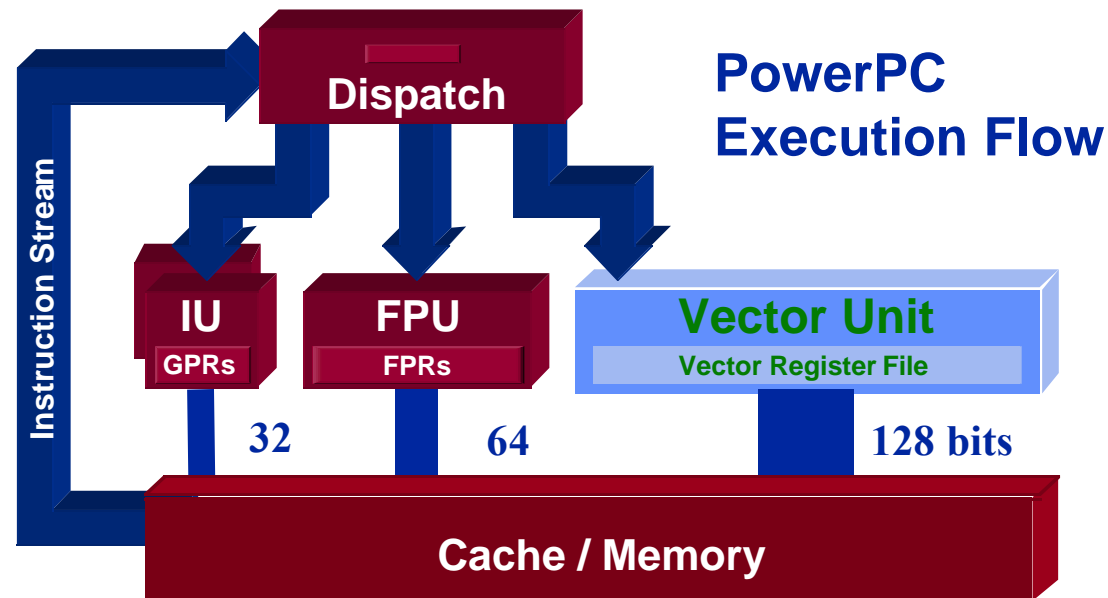


AltiVec Features

- Supports parallel operation on byte, halfword, word and 128-bit operands
 - Intra and inter-element arithmetic instructions
 - Intra and inter-element conditional instructions
 - Powerful Permute, Shift and Rotate instructions
- Vector integer and floating-point arithmetic
 - Data types
 - » 8-, 16- and 32-bit signed- and unsigned-integer data types
 - » 32-bit IEEE single-precision floating-point data type
 - » 8-, 16-, and 32-bit Boolean data types (e.g. 0xFFFF % 16-bit TRUE)
 - Modulo & saturation integer arithmetic
 - 32-bit “IEEE-default” single-precision floating-point arithmetic
 - » IEEE-default exception handling
 - » IEEE-default “round-to-nearest”
 - » fast non-IEEE mode (e.g. denorms flushed to zero)
- Control flow with highly flexible bit manipulation engine
 - Compare creates field mask used by select function
 - Compare Rc bit enables setting Condition Register
 - » Trivial accept/reject in 3D graphics
 - » Exception detection via software polling



AltiVec's Vector Execution Unit



- Concurrency with integer and floating-point units
- Separate, dedicated 32 128-bit vector registers
 - **Larger namespace = reduced register pressure/spillage**
 - **Longer vector length = more data-level parallelism**
 - **Separate files can all be accessed by execution units in parallel**
 - **Deeper register files allow more sophisticated software optimizations**
- No penalty for mingling integer, floating point and AltiVec technology operations



128-bit Vector Architecture

- 128-bit wide data paths between L1 cache, L2 cache, Load/Store Units and Registers
 - Wider data paths speed save and restore operations
- Offers SIMD processing support for
 - 16-way parallelism for 8-bit signed and unsigned integers and characters
 - 8-way parallelism for 16-bit signed and unsigned integers
 - 4-way parallelism for 32-bit signed and unsigned integers and IEEE floating point numbers
- Two fully pipelined independent execution units
 - Vector Permute Unit is a highly flexible byte manipulation engine
 - Vector ALU (Arithmetic Logical Unit) performs up to 16 operations in a single clock cycle
 - » Contains Vector Simple Fixed-Point, Vector Complex Fixed-Point, and Vector Floating-Point execution engines
 - Dual AltiVec instruction issue: One arithmetic, one “permute”



Sample Based Processing

SISD

AC3 - Audio Decode

```
do {
```

```
  decode ( channel 1 )
```

```
  decode ( channel 2 )
```

```
  decode ( channel 3 )
```

```
  decode ( channel 4 )
```

```
  decode ( channel 5 )
```

```
  decode ( channel 6 )
```

```
} while (Amplifier is on; step time)
```

SIMD

AC3 - Audio Decode

```
do {
```

```
  decode (ch 1, ch2, c3, c4, c5, c6)
```

```
} while (Amplifier is on; step time)
```

Approx 6x performance improvement.



MOTOROLA

Semiconductor Products Sector

 **DigitalDNA**
from Motorola



Simplified Systems Design

- Existing DSP based systems model is Single MPU with Multiple DSPs
 - 2 different architectures, software code bases, and hardware types
 - DSPs limit frequency and algorithm support
 - Upgrades in performance require new hardware, and at least 1 major software change
 - **Result: High-cost upgrades, slow time-to-market, in-field hardware swap required**
- New Model is Single or Multiple MPUs
 - One architecture, one code base, and one hardware type
 - Very high frequency with total algorithm support
 - Performance upgrades available with only a software change
 - **Result: Low cost upgrades, fast time-to-market, no in-field hardware swap**



AltiVec Instruction Set Features

- 162 new instructions added to the PowerPC ISA
- 4-operand, non-destructive instructions
 - Up to three source operands and a single destination operand
 - Supports advanced “multiply-add/sum” and permute primitives
- All instructions fully pipelined with single-cycle throughput
 - Simple ops: 1 cycle latency
 - Compound ops: 3-4 cycle latency
 - No restriction on issue with scalar instructions
- Enhanced cache/memory interface
 - Software hints for data re-use probability
 - Prefetch support (stride-N access)
- Simplified load/store architecture
 - Simple byte, halfword, word and quadword loads & stores
 - No unaligned accesses – software-managed via permute instruction



In summary . . .

- **AltiVec™ Technology provides a robust instruction set**
 - In terms of operations
 - In terms of data types and sizes
 - In terms of other options (such as saturation, data movement)
 - Allows a streamlined hardware implementation (cycle time, latency, throughput)
- **AltiVec enables a broad range of embedded and computing applications**



AltiVec Instruction Set

➤ Vector Intra-element Instructions

- Integer Arithmetic Instructions
- Floating-Point Arithmetic Instructions
- Conditional Control Flow Instructions
- Rotate, Shift and Logical Instructions
- Memory Access Instructions

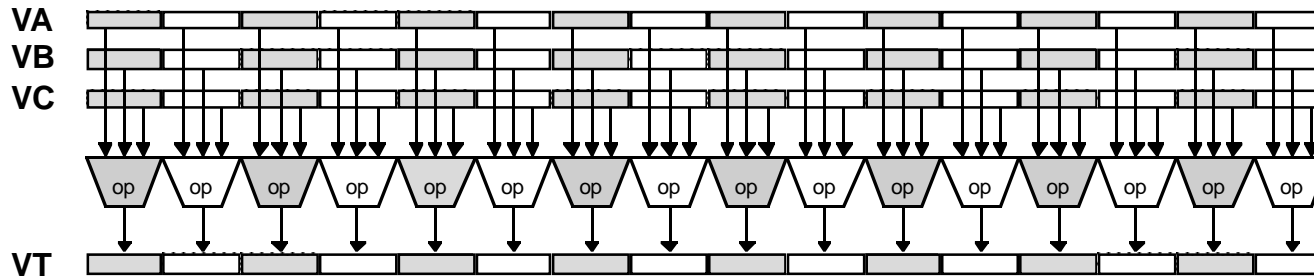
➤ Vector Inter-element Instructions

- Permute Instruction
- Data Movement/Manipulation Instructions
- Integer Multiply Odd/Even Instructions
- Integer Multiply-Sum Instructions
- Integer Sum Across Instructions

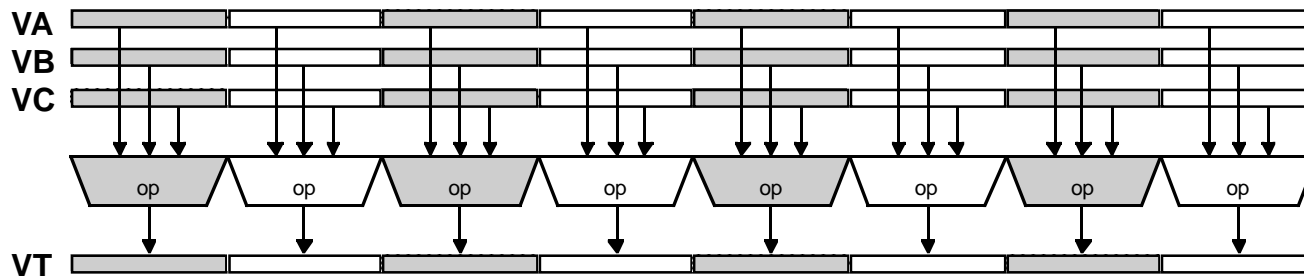


SIMD Intra-element Instructions

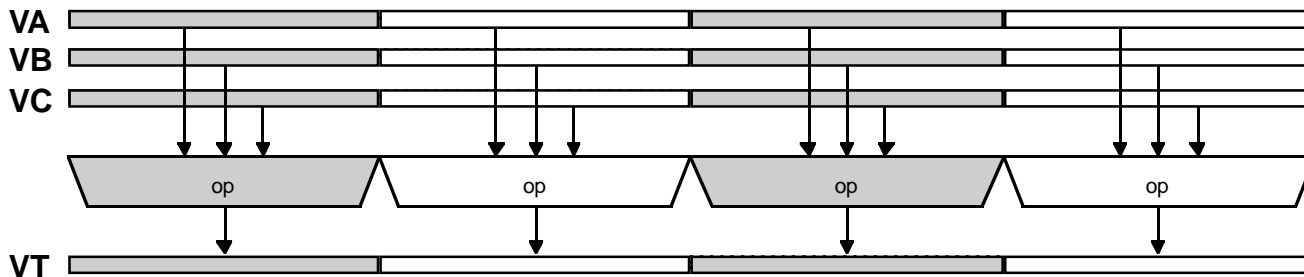
➤ 16 x 8-bit elements



➤ 8 x 16-bit elements

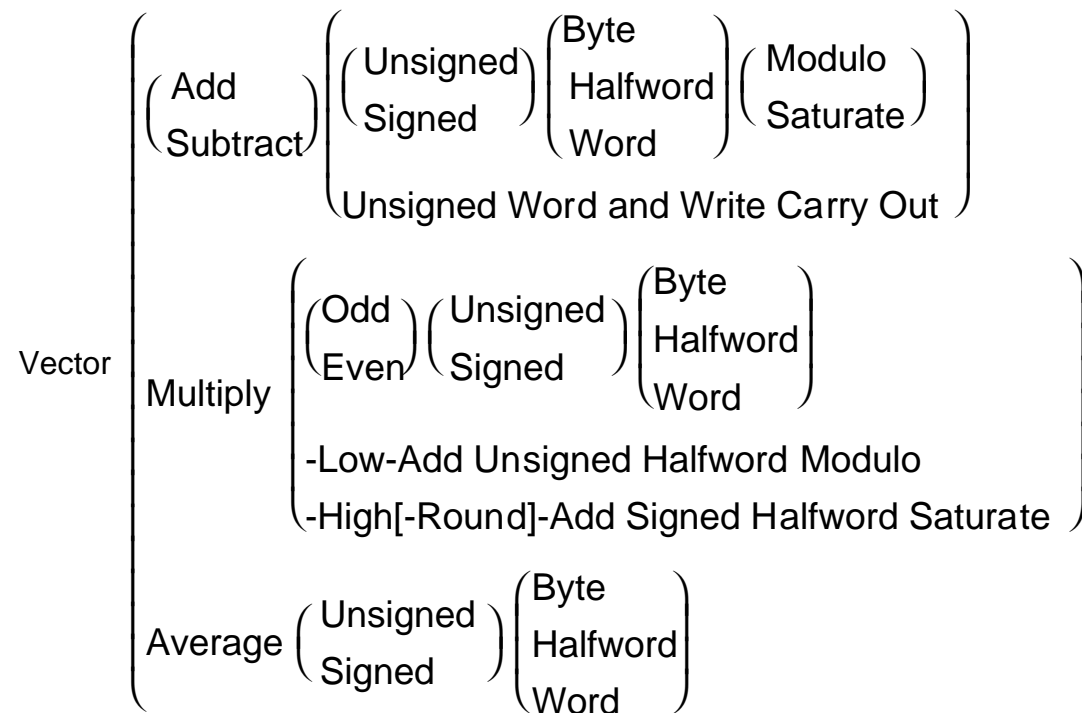
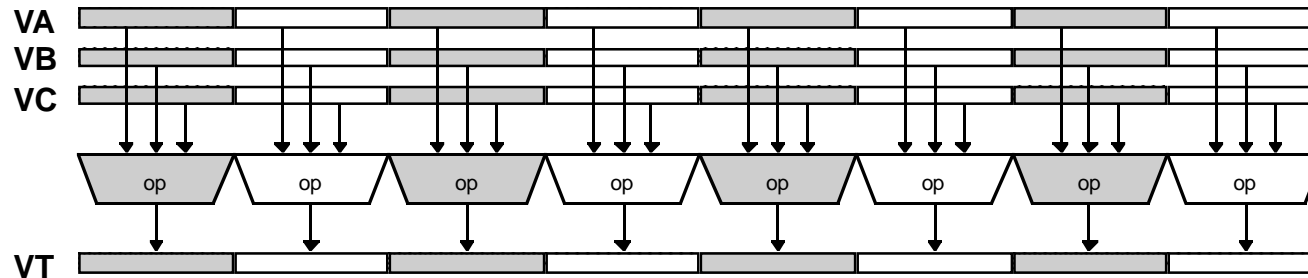


➤ 4 x 32-bit elements





Integer Arithmetic Instructions

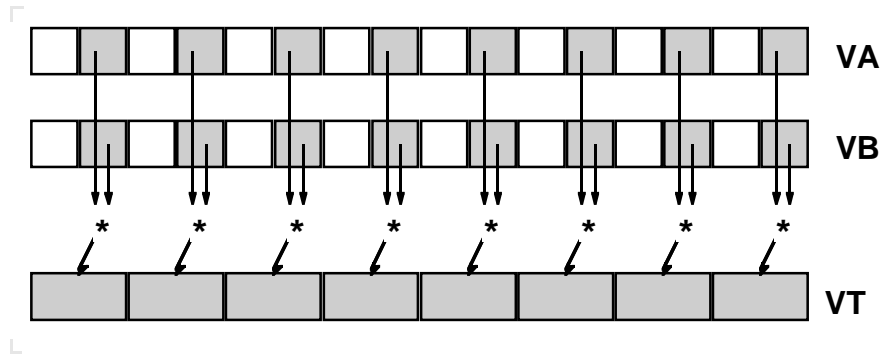




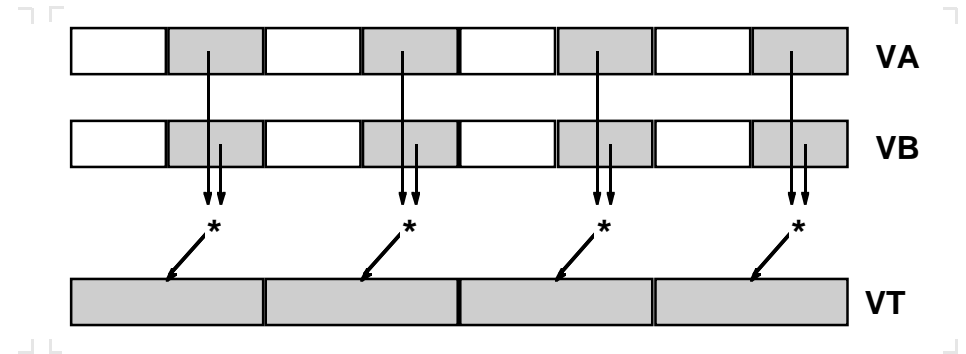
Multiply Odd and Even

Multiply Odd

byte

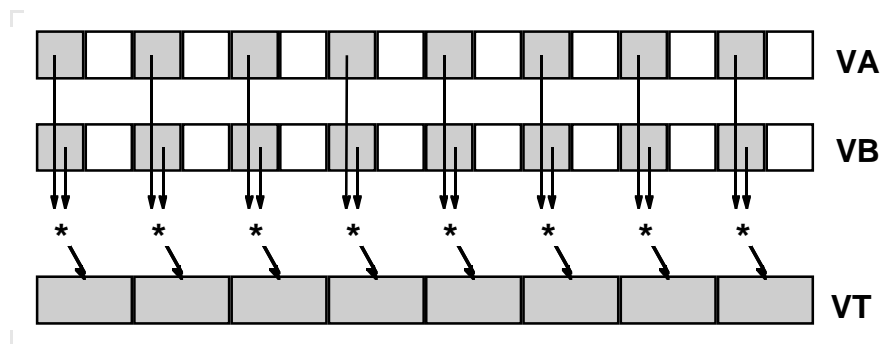


half-word

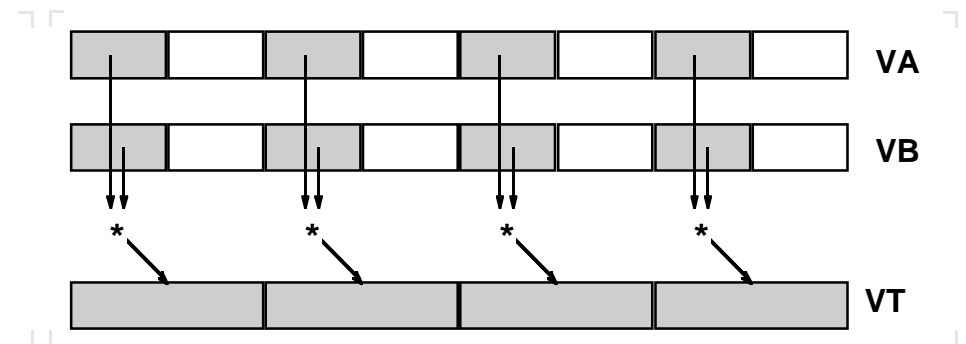


Multiply Even

byte

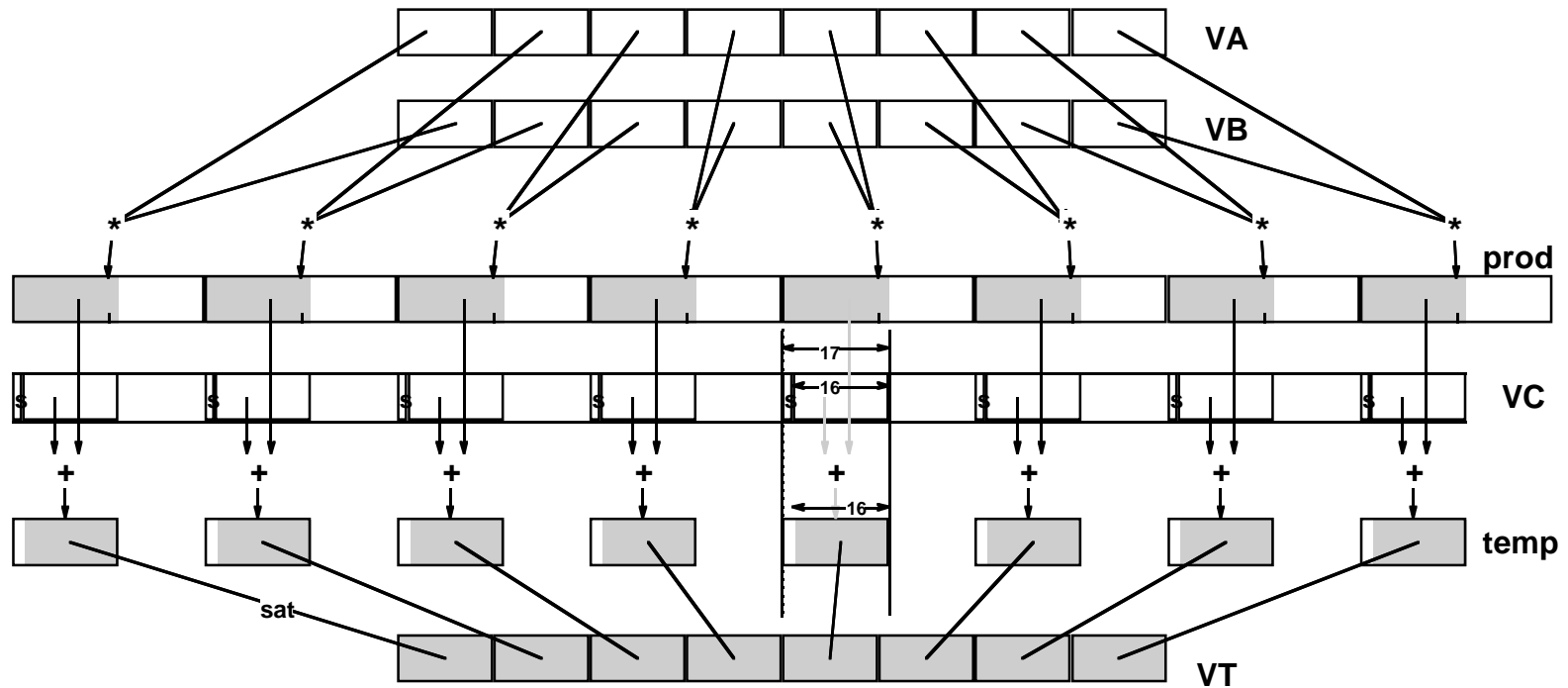


half-word





Multiply-High and Add

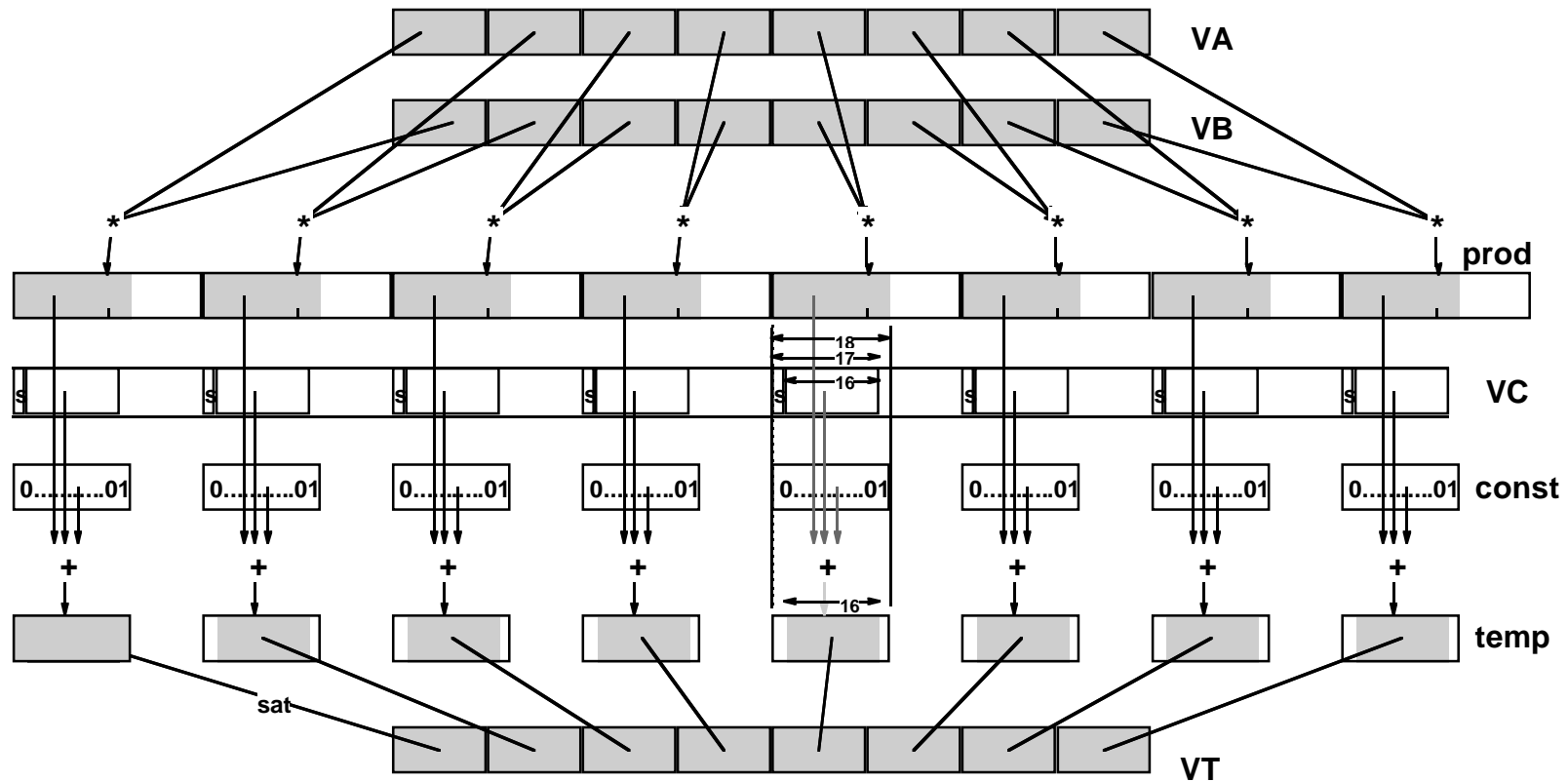


Vector Multiply High-Add Signed Halfword Saturate

`vmhaddshs VT, VA, VB, VC`



Multiply-High Round and Add

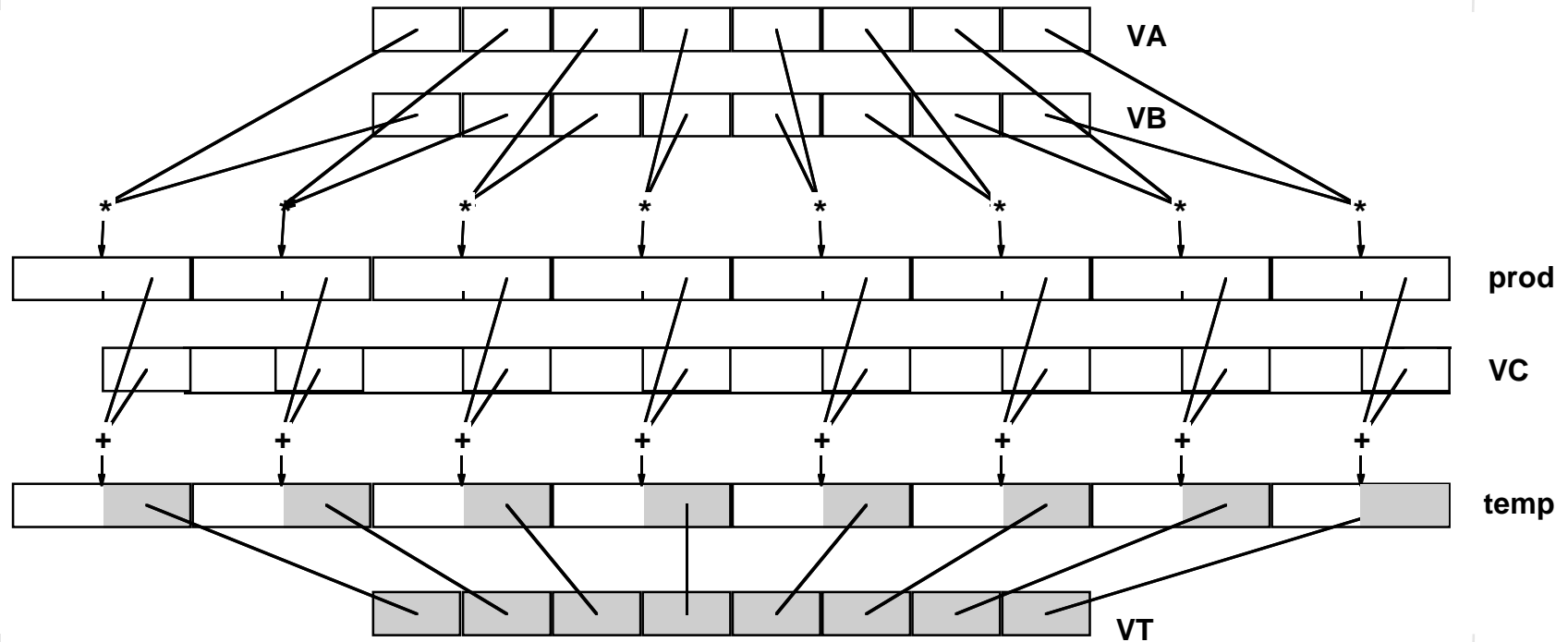


Vector Multiply High-Round and Add Signed Halfword Saturate

vmhraddshs VT, VA, VB, VC



Multiply Low and Add

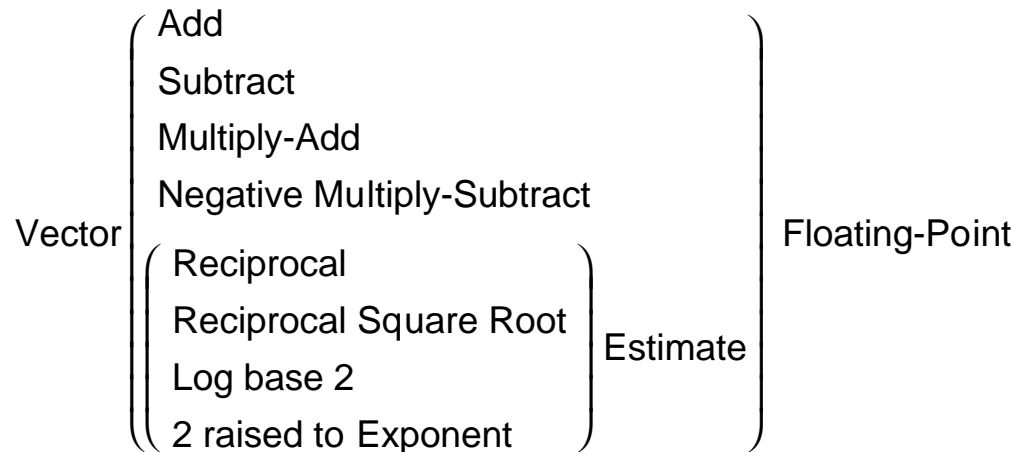
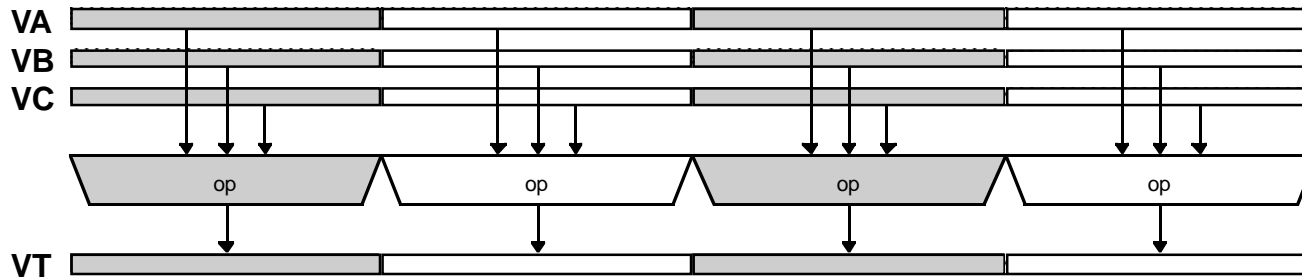


Vector Multiply-Low and Add Unsigned Halfword Modulo

`vmladduhm VT, VA, VB, VC`



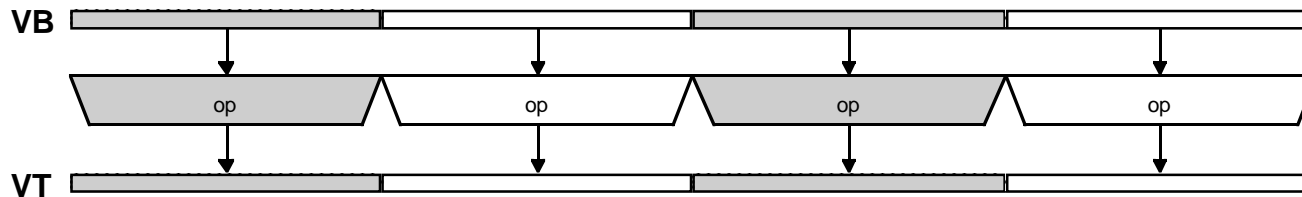
Floating-Point Arithmetic Instructions



- Performs four 32-bit “IEEE-default” single-precision floating point arithmetic operations in parallel
- Pipelined 4-cycle latency for Add, Subtract, and Fused Multiply-Add



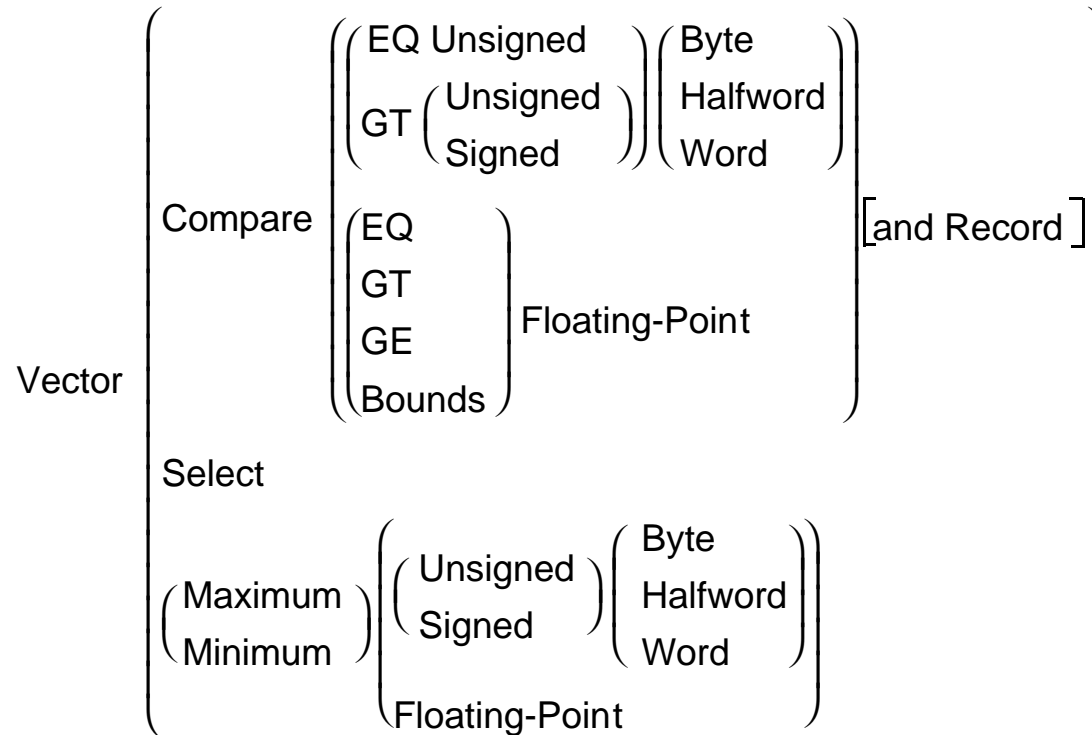
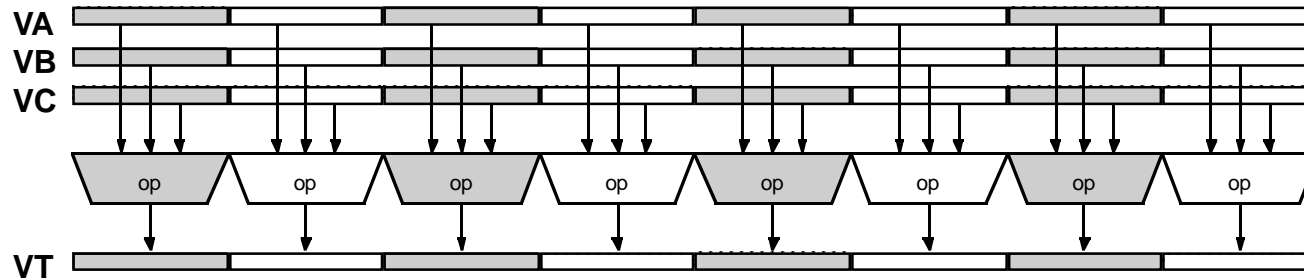
Floating-Point Conversion Instructions



Vector $\left(\begin{array}{l} \text{Round to Floating-Point Integer} \left(\begin{array}{l} \text{to Nearest} \\ \text{toward Zero} \\ \text{toward +infinity} \\ \text{toward -infinity} \end{array} \right) \\ \text{Convert} \left(\begin{array}{l} \text{To} \\ \text{From} \end{array} \right) \left(\begin{array}{l} \text{Unsigned} \\ \text{Signed} \end{array} \right) \text{ Fixed-Point Word} \end{array} \right)$



Conditional Instructions



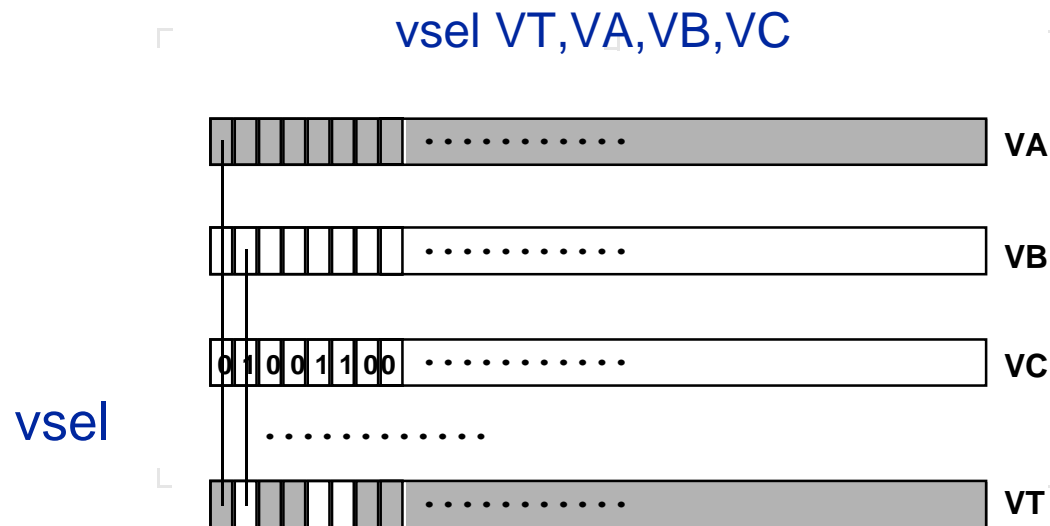
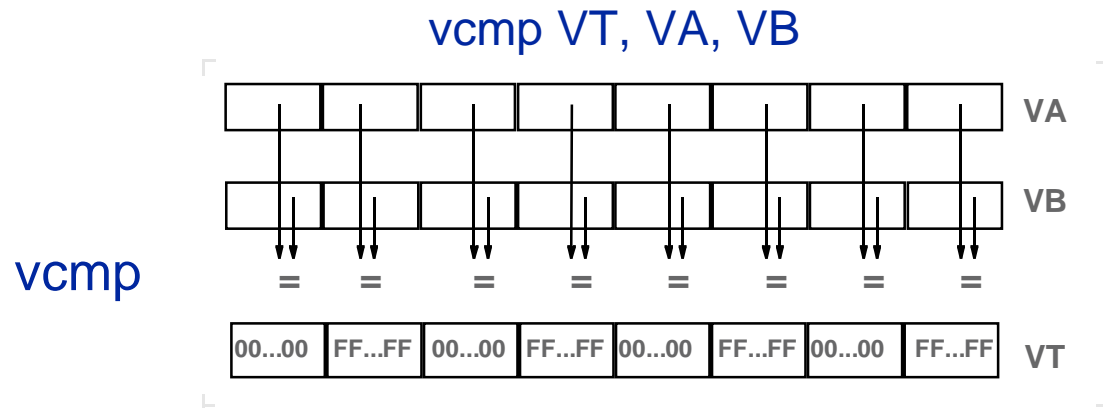


Compare-Select

- Supports IF-THEN-ELSE by computing results for both branch paths and then selecting correct path's results
- Reduces inefficient control flow feedback to branch unit
- **Compare** creates field masks
 - target element set to ones where comparison is true
 - target element set to zeroes where comparison false
- **Select** performs a bit-wise selection based on generated mask



Compare-Select





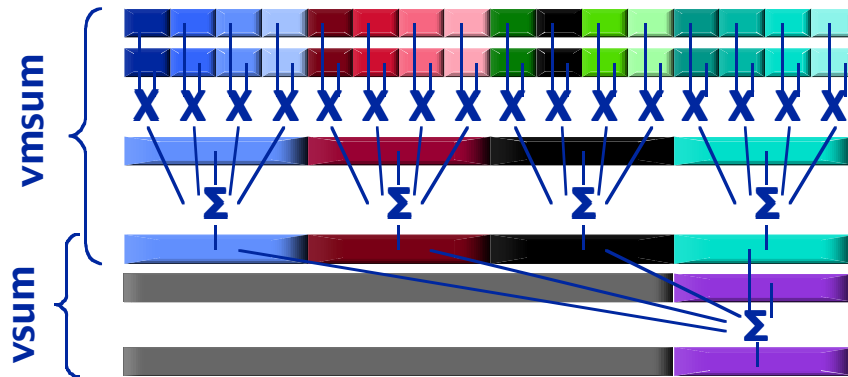
Rc Bit

- Rc bit enables setting CR field 6 with summary comparison status
 - CR bit 24 set if all compare result elements are equal to all ones
 - CR bit 26 set if all compare result elements are equal to all zeroes
- Provides limited but critical control flow support
 - Trivial accept/reject on 3D clipping
 - 3D lighting
 - Special handling of exceptions
 - » Software polls for occurrence of saturation
 - » Image processing application identified that needs to reset thresholds of other pixel components when one component saturates



Algorithmic Features

Vector Dot Product (FIR)



Application:

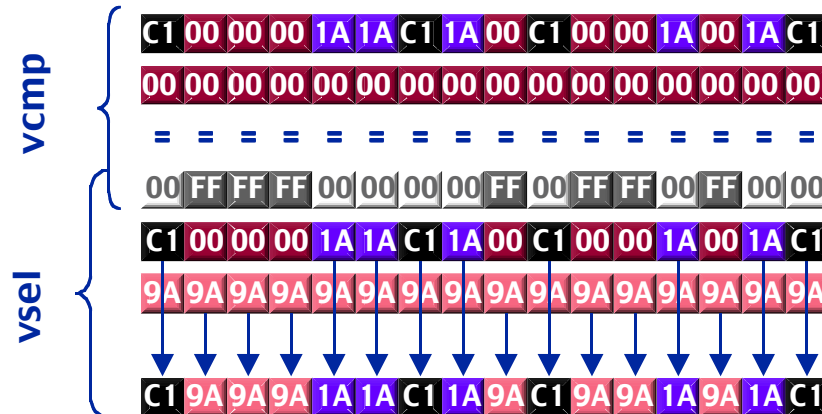
Used heavily in DSP code

Performance:

PowerPC: **36 instructions**
(18 cycles throughput)

PowerPC + **Altivec**: **2 instructions**
(**2 cycles** throughput)

Vector Compare and Select



Application:

Permits removal of control flow

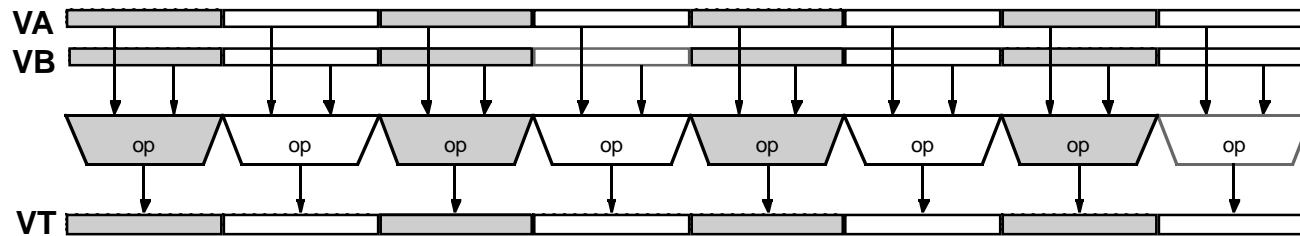
Performance:

PowerPC: **48 instructions**
(32 cycles throughput)

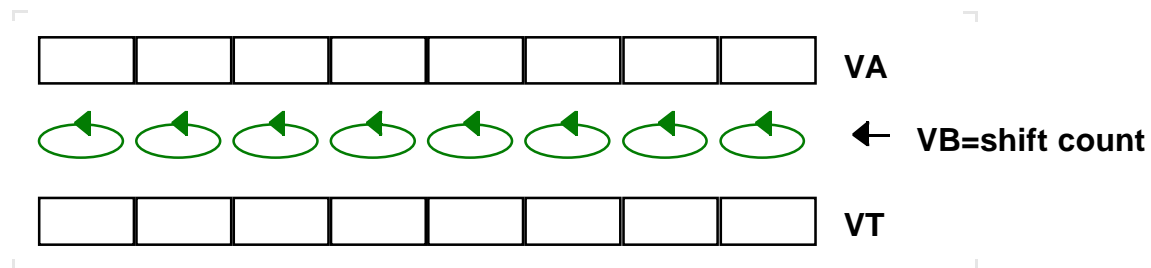
PowerPC + **Altivec**: **2 instructions**
(**3 cycles** throughput)



Rotate, Shift and Logical Instructions



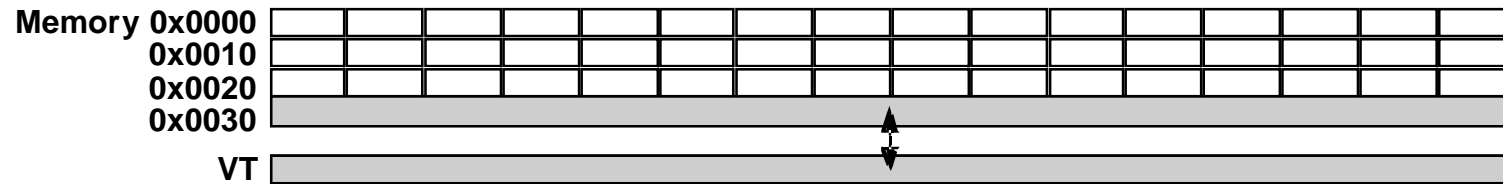
Vector $\left(\begin{array}{l} \text{Rotate Left Unsigned} \\ \text{Shift} \left(\begin{array}{l} \text{Left Unsigned} \\ \text{Right} \left(\begin{array}{l} \text{Unsigned} \\ \text{Signed} \end{array} \right) \end{array} \right) \end{array} \right) \left(\begin{array}{l} \text{Byte} \\ \text{Halfword} \\ \text{Word} \end{array} \right)$



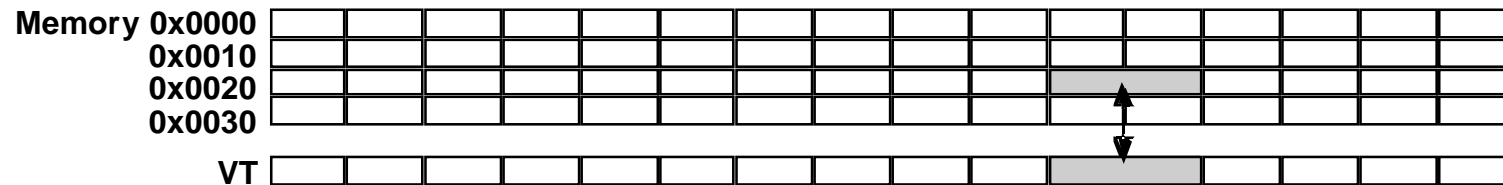


Load/Store Vector Instructions

(Load)
(Store) Vector (MRU
LRU)



(Load)
(Store) Vector Element (Byte
Halfword
Word)



Load Vector for Shift (Left
Right)





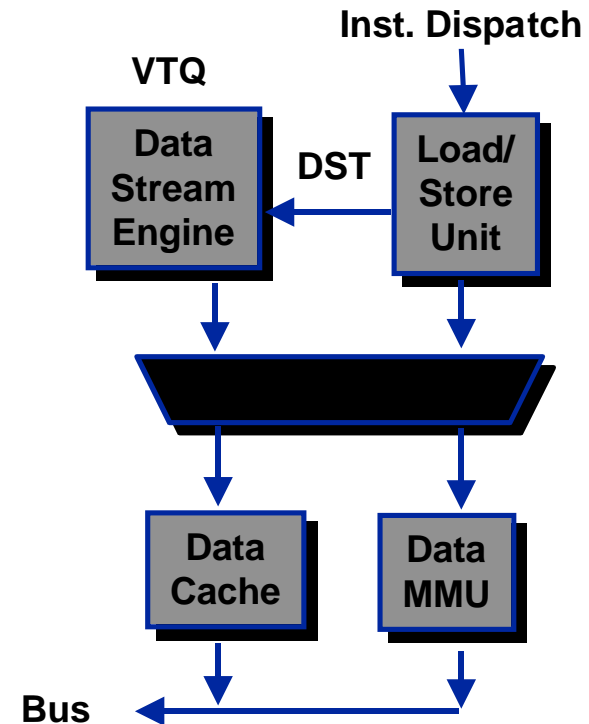
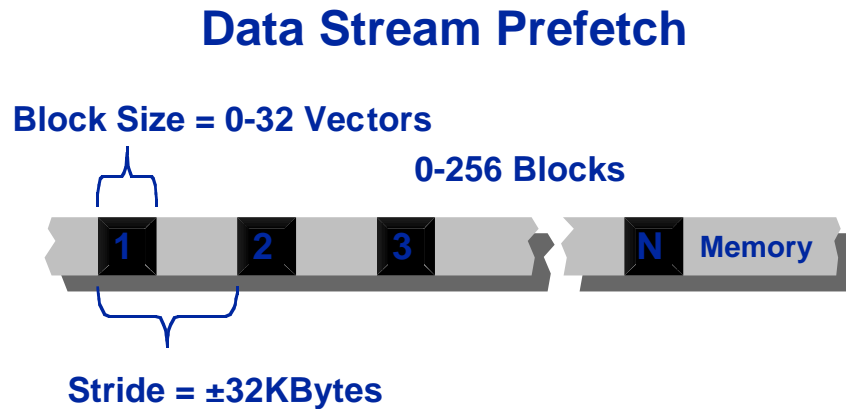
Enhanced Cache/Memory Interface

- Load & store with LRU instructions
 - Performs memory access and marks cache block “next to be replaced”
 - Avoids flushing cache with multimedia data exhibiting limited data reuse
 - “Software-managed” memory buffer in cache
- Data stream touch and stop instructions
 - Enables reuse of data cache as a memory access buffer
 - Alleviates memory access latency by enabling early data prefetch
 - **Data stream touch** offers software directed cache prefetch
 - » Specifies N blocks of data to be prefetched from memory into cache
 - » Each block to be prefetched is K bytes in length
 - » The first block to be prefetched is located at address X, the second at address $X+N$, the third at address $X+2N$, . . .
 - **Data stream stop**
 - » Allows stopping of a data stream operation when speculative-incorrectly started
 - » Selectively stop any stream via ID tag
- “Transient/static” hint for cache placement strategy



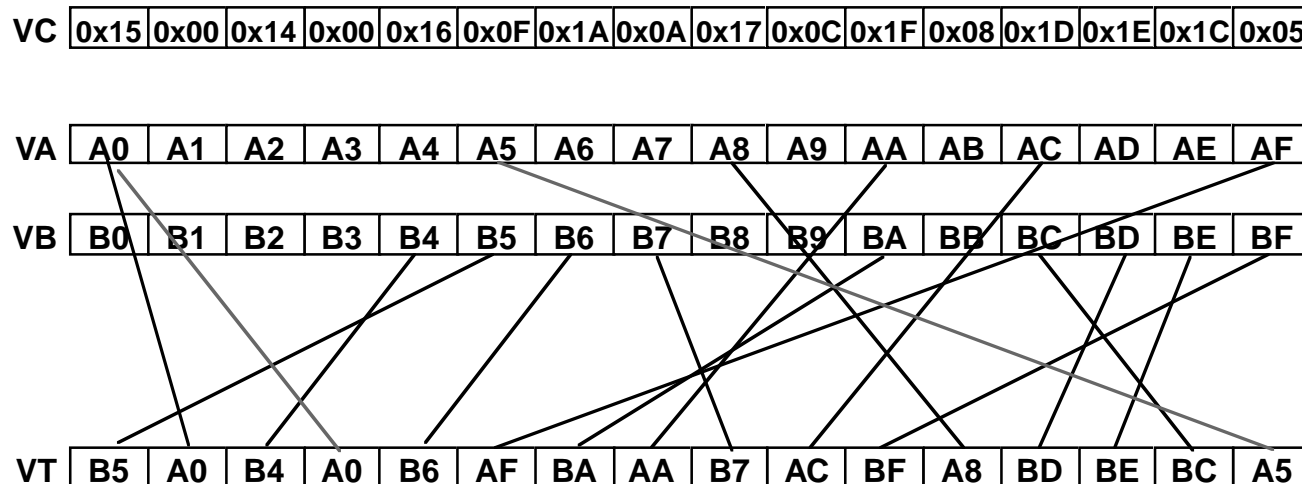
AltiVec Data Stream Prefetch

- Provides a hardware engine for Data Stream Prefetching
- Four simultaneous streams are supported, independent and asynchronous, addressable via 2-bit ID tag





Permute Instruction



- Provides full byte-wide data crossbar for 128-bit registers
- Selects any 16 8-bit elements from 2x16 8-bit elements
- Other AltiVec instructions are special cases of permute
 - Pack, unpack, and merge
 - Splat (element or literal replication)
 - Shift left long immediate
 - Permute also supports other higher-level functions
 - Software-managed unaligned access
 - Table look-up

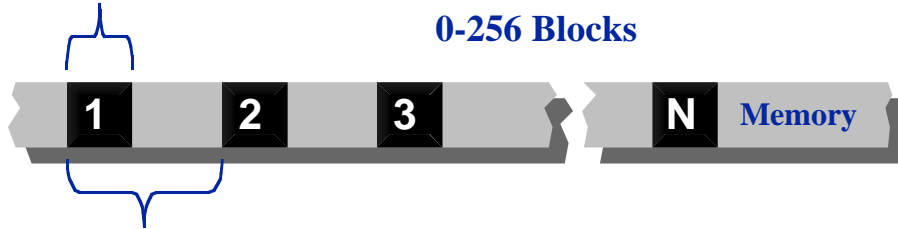


Data Management Features

Data Stream Prefetch

Block Size = 0-32 Vectors

0-256 Blocks



Stride = $\pm 32\text{KBytes}$

Vector Permute

Control Vector

1718 D E F 1E 1 0 1211 10A 14141414 VR_C

VR_A

Input Vectors

VR_B

0 1 2 3 4 5 6 7 8 9 A B C D E F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F

Output Vector VR_T

Output Vector

Applications:

- Software-directed prefetch into cache
- Up to 4 simultaneous streams
- Hides memory latency for stream processing applications (MPEG, FIR, etc)

Performance:

PowerPC: **Does not exist**

PowerPC + **AltiVec**: **1 instruction**
(**Can save hundreds of clocks**)

Applications:

- Byte interleaving (conv. encoding)
- Dynamic memory address alignment
- Fast 32-element table lookup

Performance:

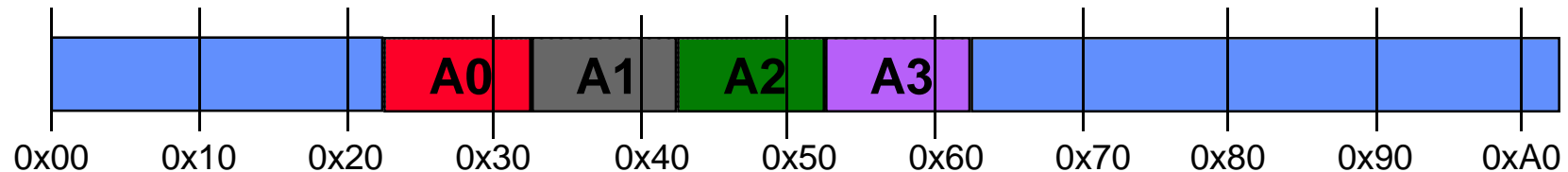
PowerPC: **5 to 50 instructions**,
depending on application

PowerPC + **AltiVec**: **1 instruction**
(**1 cycle throughput**)



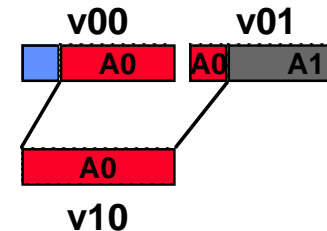
Unaligned Accesses

- For a single vector, A0: $\text{mod}16(\text{addr}(\text{A0}))$



- For a series of array elements: A0, A1, A2, A3

```
lvx    v00, &A0
lvx    v01, &A1
lvsl   v02, &A0
vperm  v10, v00, v01, v02
```



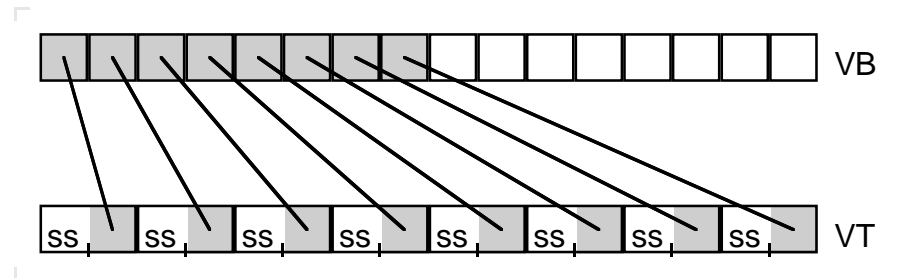
- N+1 memory accesses for software, 2N accesses for hardware

	v00	v01	v02	vT (v10-v13)
lvx v00, &A0	[A0]	[A0, A1]	⇒	[A0]
lvx v01, &A1				
lvsl v02, &A0				
vperm v10, v00, v01, v02				
lvx v00, &A2		[A0, A1]	[A1, A2]	⇒ [A1]
vperm v11, v01, v00, v02				
lvx v01, &A3	[A1, A2]	[A2, A3]		⇒ [A2]
vperm v12, v00, v01, v02				
lvx v00, &(A3+16)		[A2, A3]	[A3]	⇒ [A3]
vperm v13, v01, v00, v02				

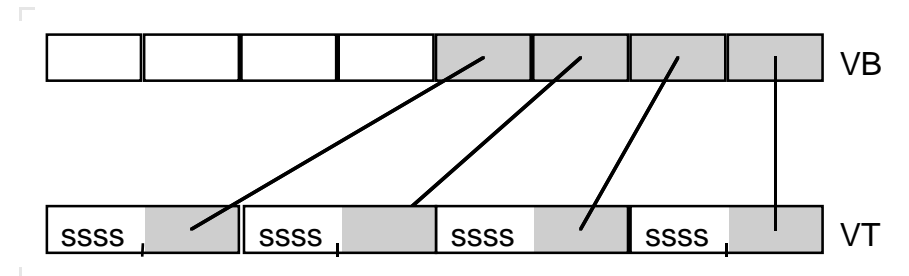


Unpack Instructions

Vector Unpack High $\left(\begin{array}{c} \text{Signed (Byte Halfword)} \\ \text{Pixel*} \end{array} \right)$



Vector Unpack Low $\left(\begin{array}{c} \text{Signed (Byte Halfword)} \\ \text{Pixel*} \end{array} \right)$

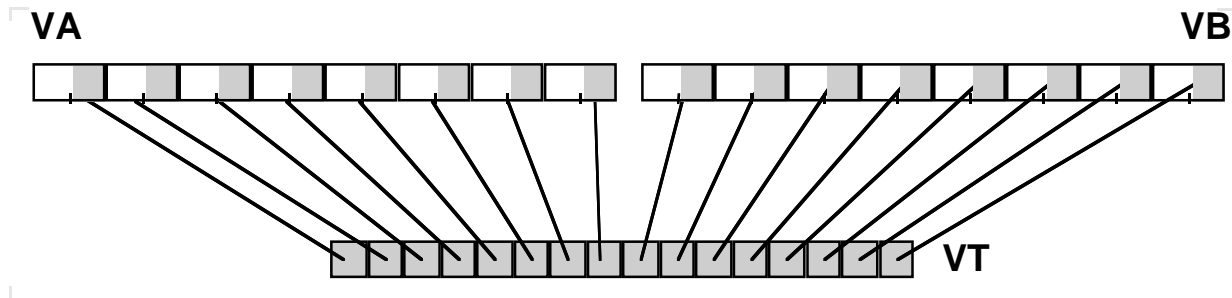


* unpacks 1-5-5-5 pixels into 4 8-bit components



Pack Instructions

Vector Pack $\left(\begin{array}{l} \text{Unsigned} \left(\begin{array}{l} \text{Halfword} \\ \text{Word} \end{array} \right) \left(\begin{array}{l} \text{Modulo} \\ \text{Saturate} \end{array} \right) \\ \text{Signed} \left(\begin{array}{l} \text{Halfword} \\ \text{Word} \end{array} \right) \text{Saturate} \\ \text{Pixel}^* \end{array} \right)$

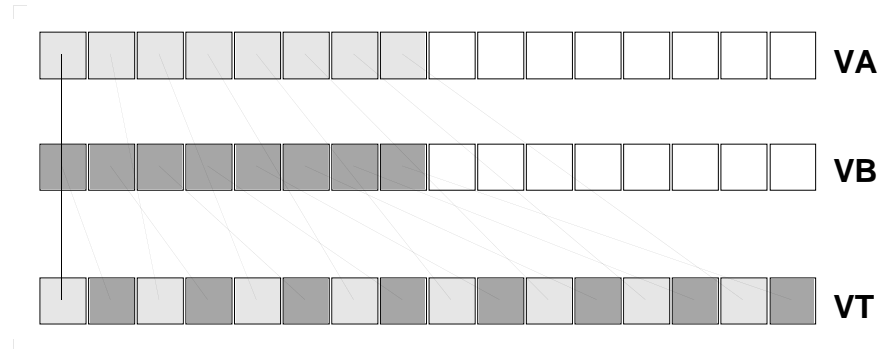


* packs 4 8-bit components into 1-5-5-5 pixels

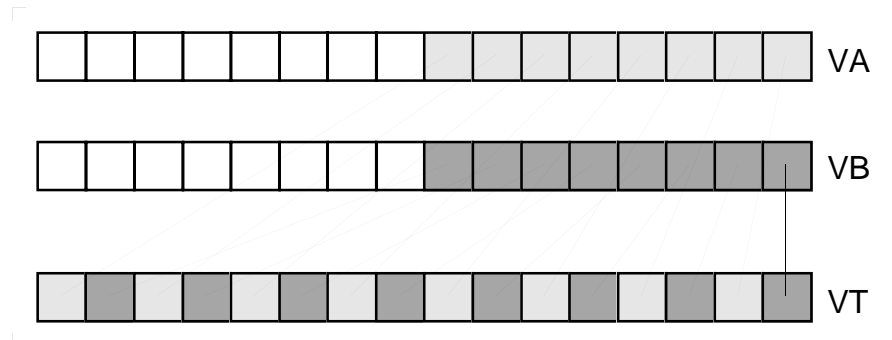


Merge Instructions

Vector Merge High $\left(\begin{array}{c} \text{Byte} \\ \text{Halfword} \\ \text{Word} \end{array} \right)$

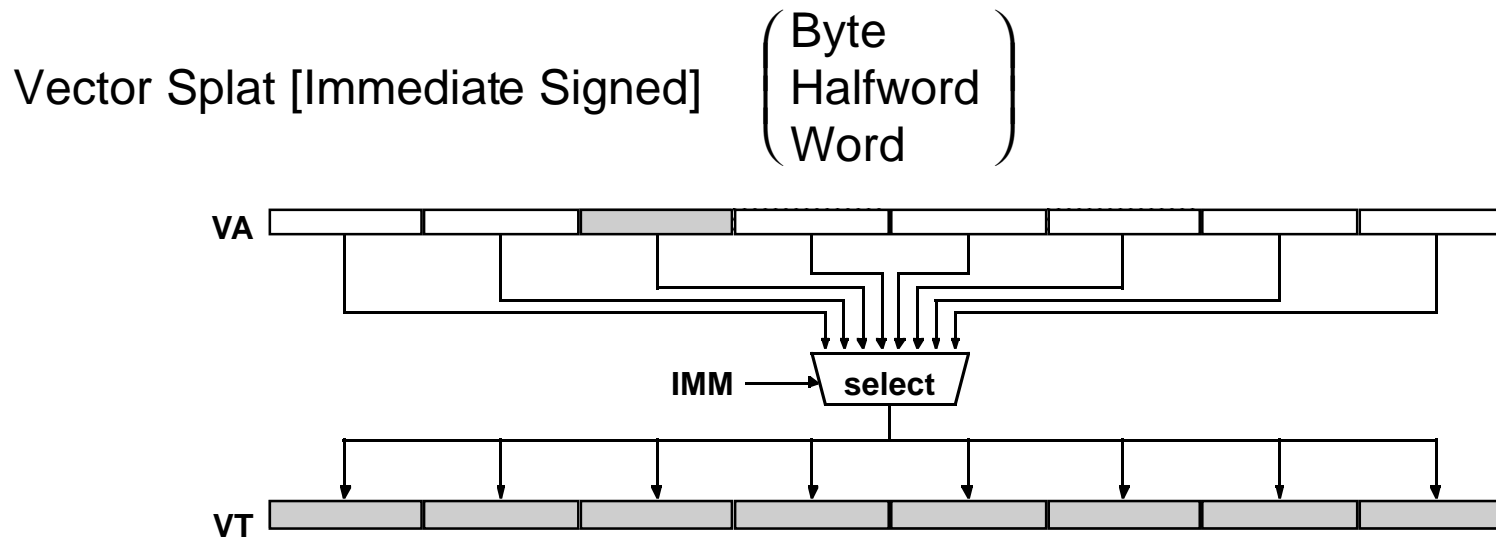


Vector Merge Low $\left(\begin{array}{c} \text{Byte} \\ \text{Halfword} \\ \text{Word} \end{array} \right)$





Splat Instructions



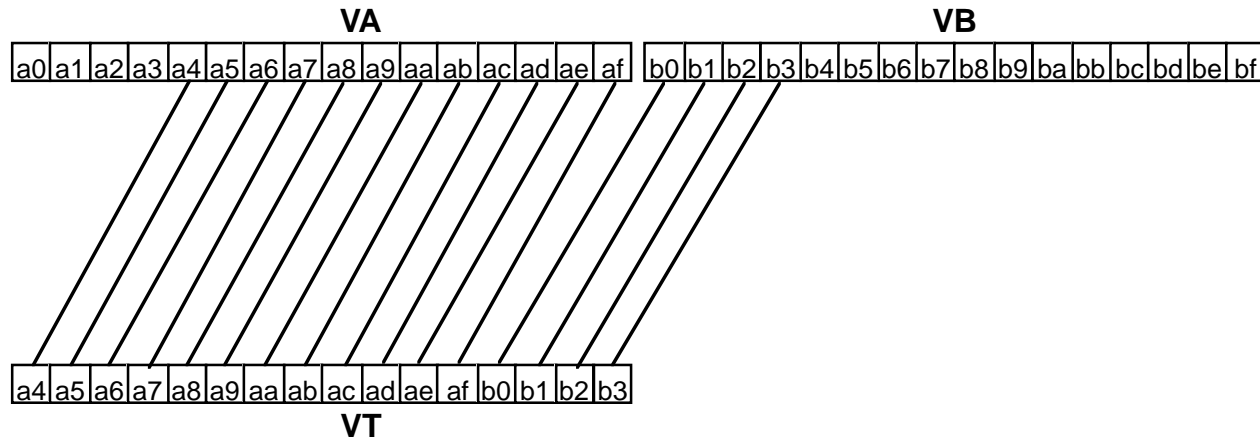
- Useful for scalar operands



Shift Left Double Immediate

➤ **Example: vsldi VT,VA,VB,#0004**

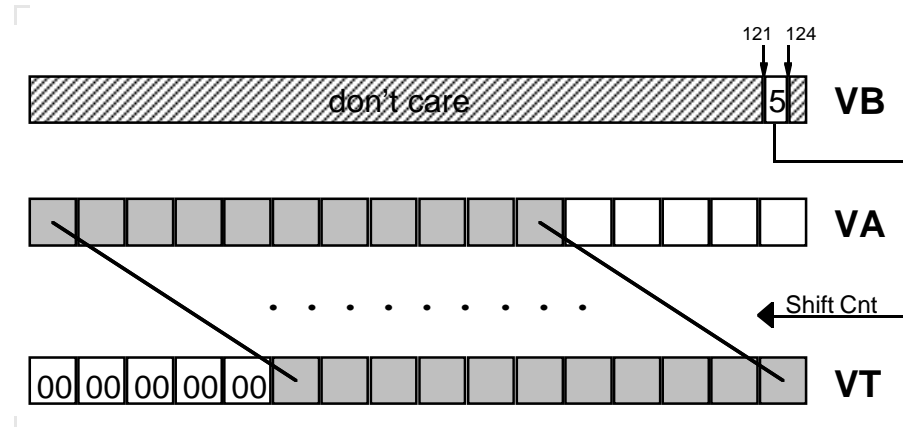
IMM = 4



- Immediate form of permute
- Supports 128-bit rotate/shift left/right
 - Rotate via specifying VA=VB
 - Shift left via specifying (VB)=0
 - Shift right via specifying (VA)=0 & shift left count = 16–shift right count



Shift Quadword Instructions

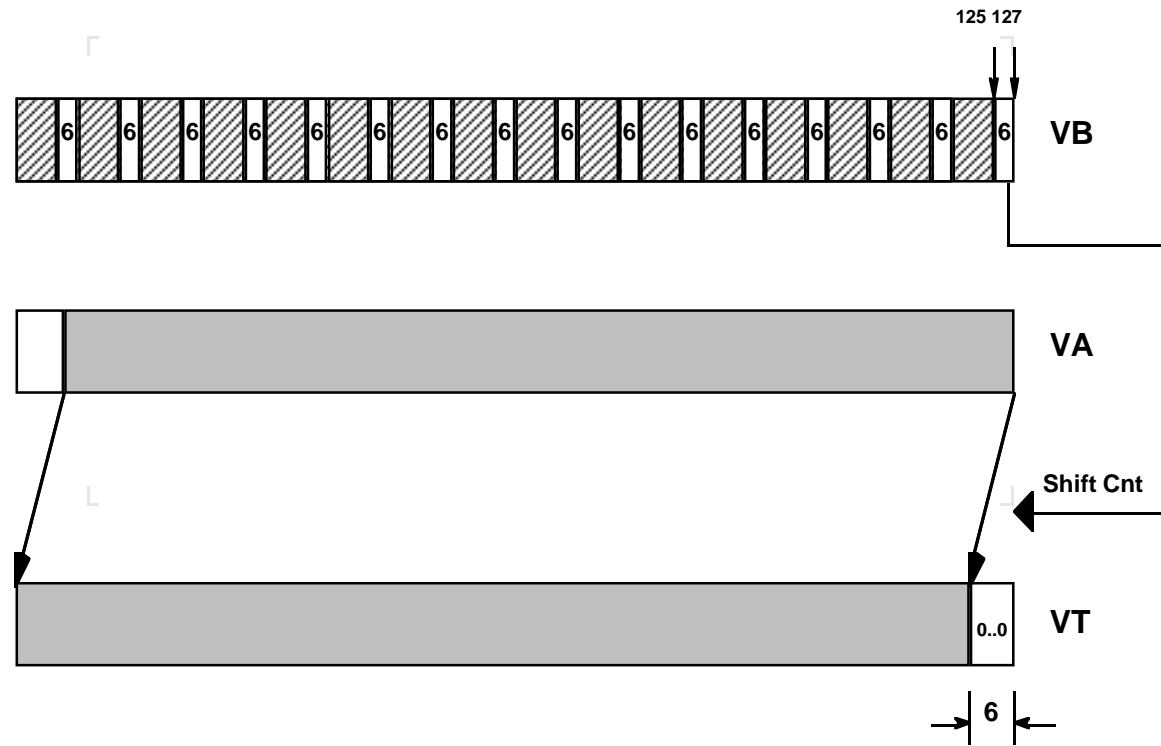


Vector Shift $\left(\begin{matrix} \text{Left} \\ \text{Right} \end{matrix} \right) \left(\begin{matrix} \text{by Octet} \\ \text{by bit} \end{matrix} \right)$

Shift vector left or right by octets, zero fill.



Vector Shift



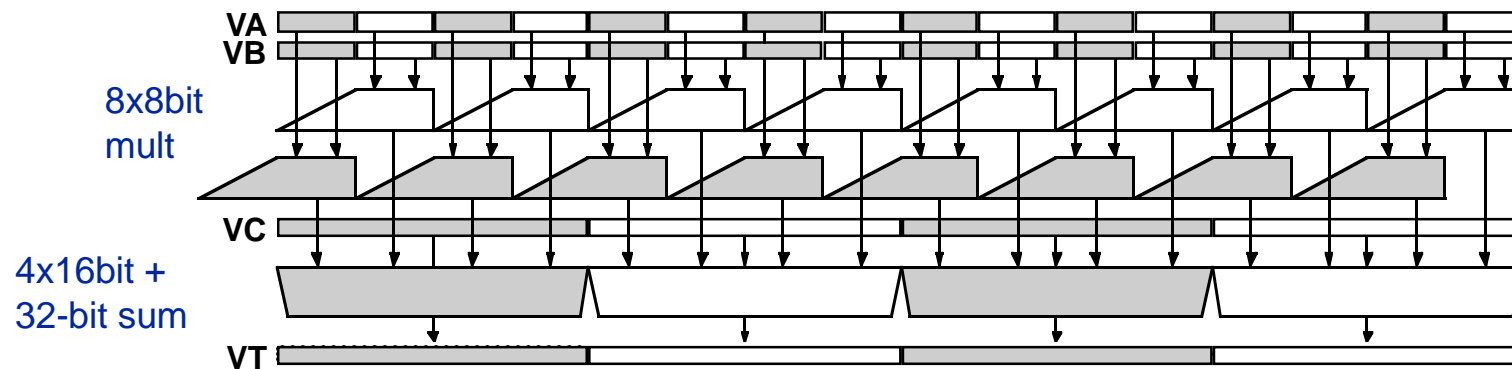
Vector Shift (Left
Right)

Shift entire vector left or right by bit up to 8 bits, zero fill.

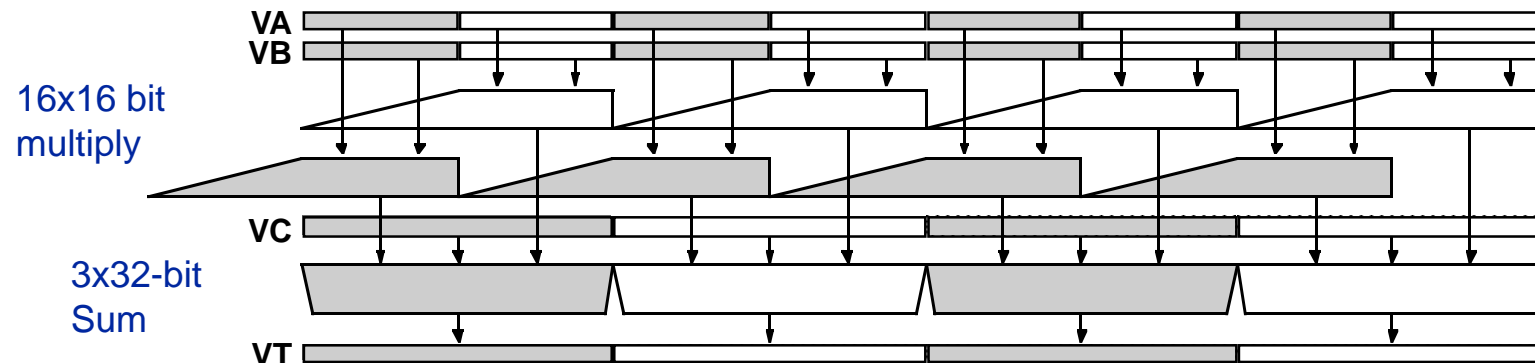


Multiply-Sum Integer Instructions

Vector Multiply-Sum (Unsigned
Mixed-signed) Byte Modulo



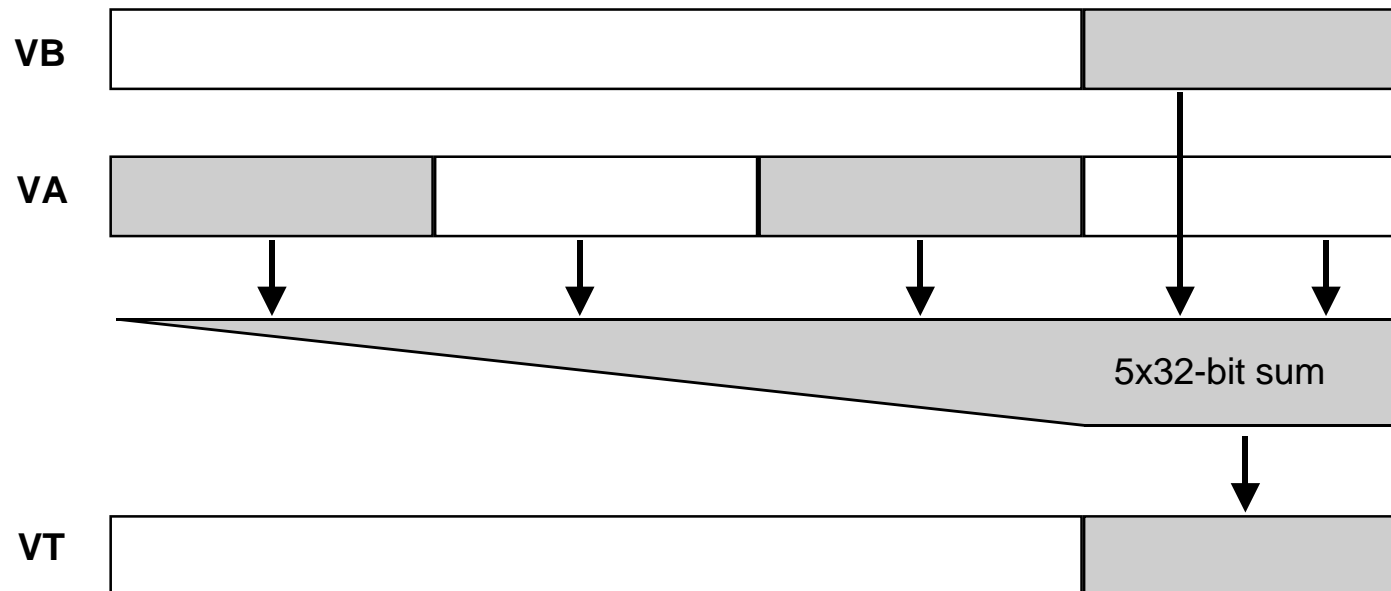
Vector Multiply-Sum (Unsigned
Signed) Halfword (Modulo
Saturate)





Sum Across Integer Instructions

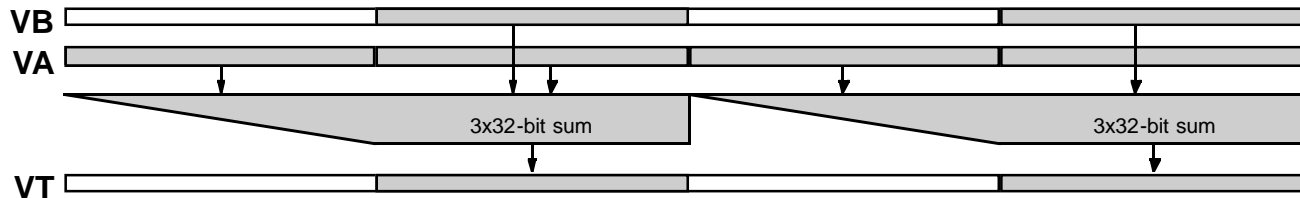
Vector Sum Across Signed Integer32 Signed Saturate



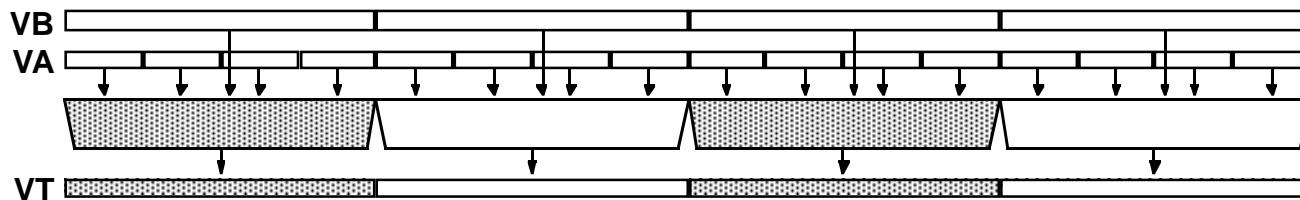


Sum Across Partial Integer Instructions

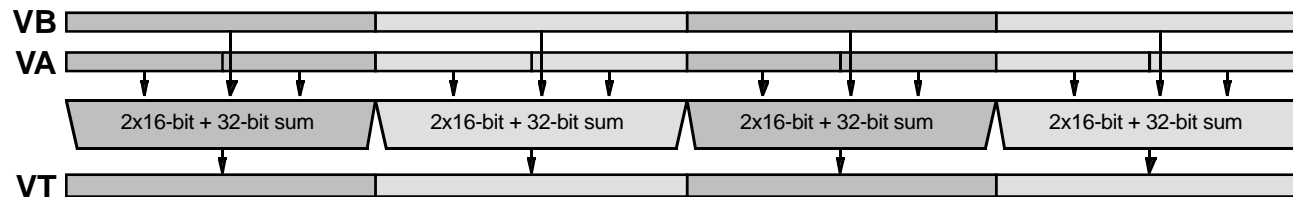
Vector Sum Across Partial (1/2) Signed Integer32 Saturate



Vector Sum Across Partial (1/4) Signed Integer8 Saturate

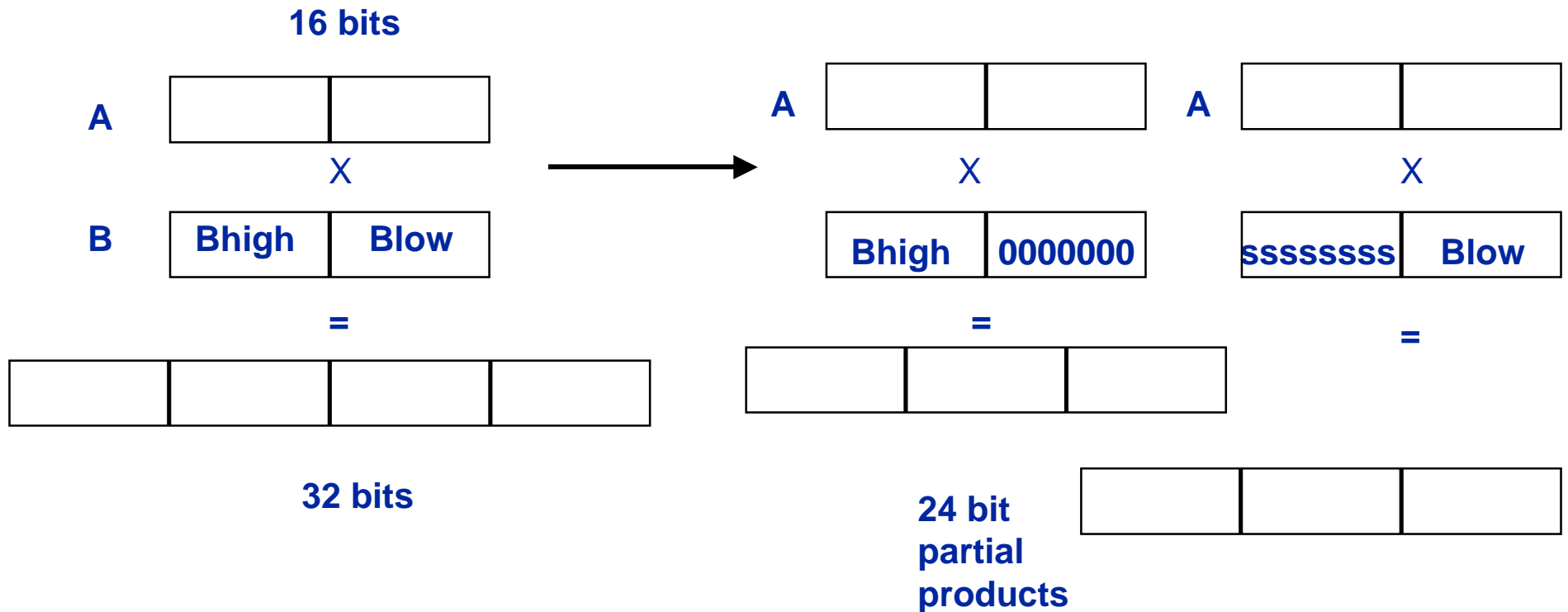


Vector Sum Across Partial (1/4) Signed Integer16 Saturate





Partial Product

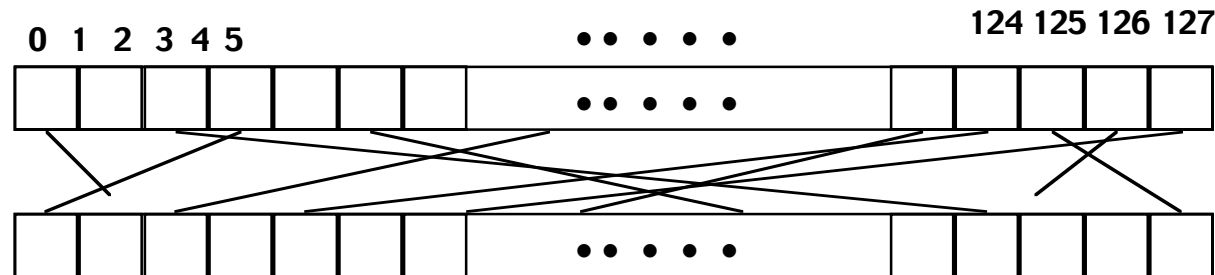


- Use permute to duplicate the A operands and spread the B operands
- Can then use mulsum to produce up to 40 bits of accumulation
- Need to combine partial sums after the main loop



128 Bit Permute

Problem: arbitrary permute of each bit in a vector register, assume static permute

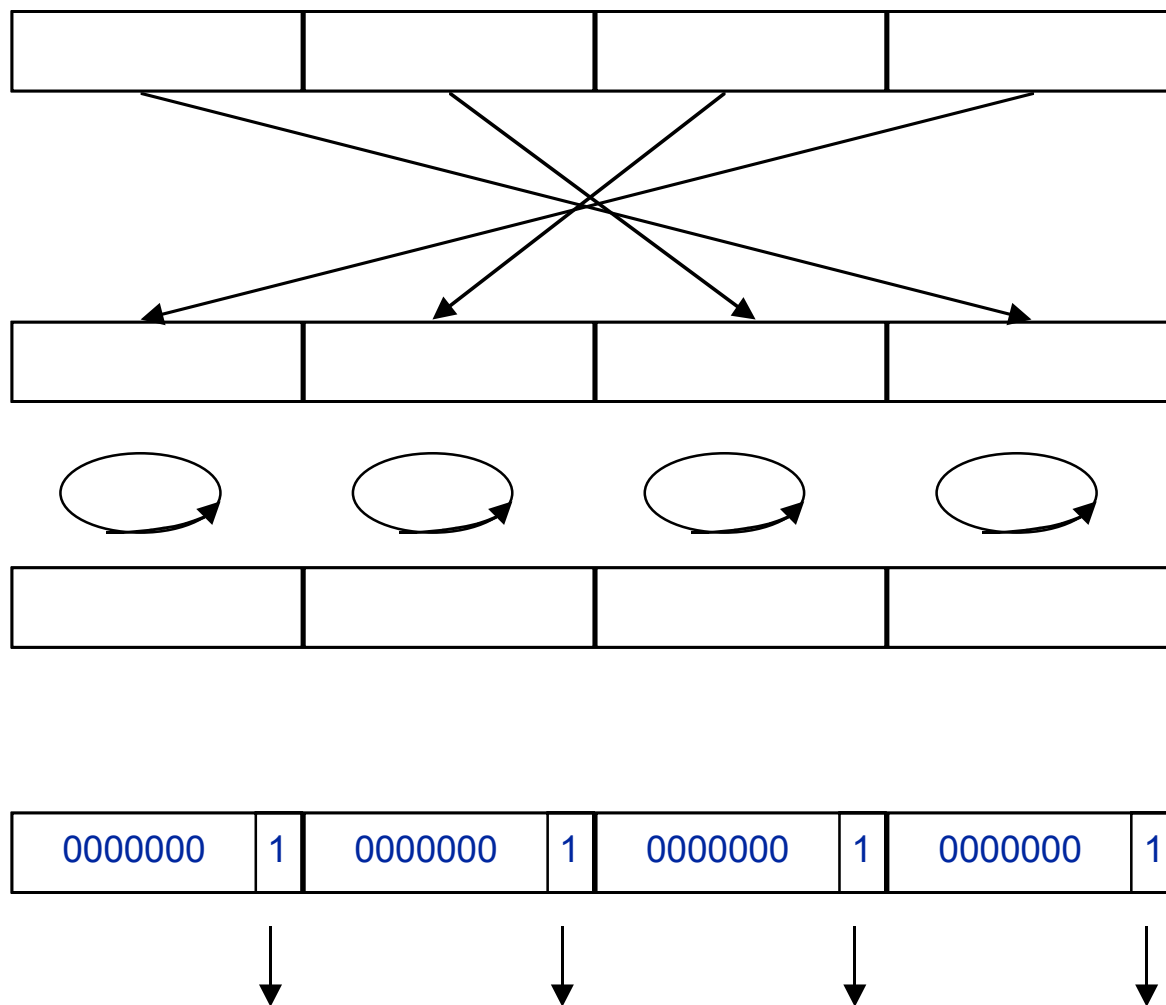


Strategy:

- 1) break 128 bit vector into 16 bytes
- 2) pre-calculate permute, rotate and select control vectors
- 3) use permute to get byte to correct location
- 4) use rotate to get bit to correct location with each byte
- 5) use select to merge bit into result byte
- 6) repeat for all eight bits in each byte



128 Bit Permute



permute

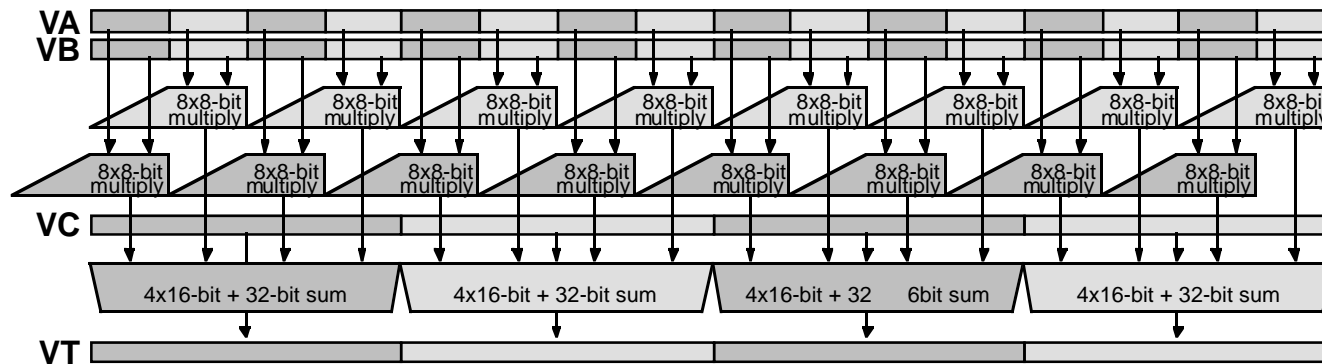
rotate

select

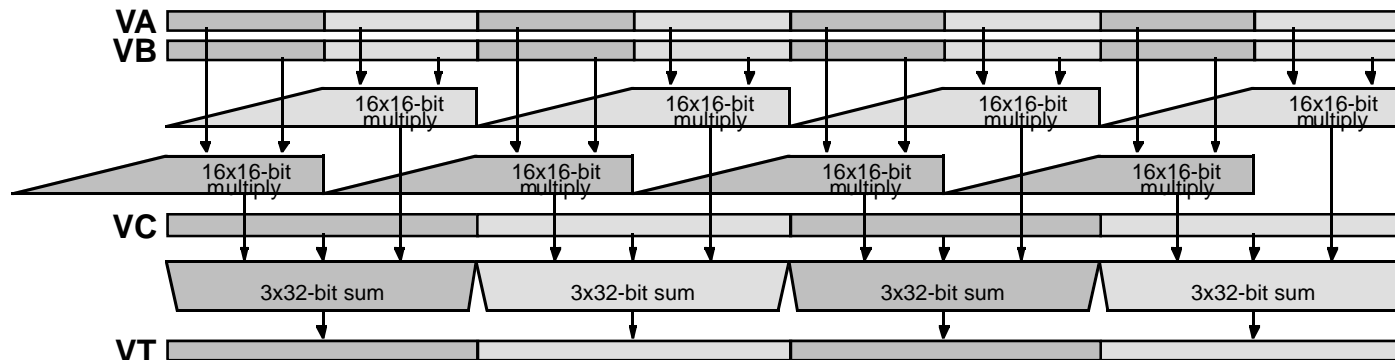


Multiply-Sum Integer Instructions

Vector Multiply-Sum $\begin{pmatrix} \text{Unsigned} \\ \text{Mixed-signed} \end{pmatrix}$ Byte Modulo



Vector Multiply-Sum $\begin{pmatrix} \text{Unsigned} \\ \text{Signed} \end{pmatrix}$ Halfword (Modulo Saturate)





Parallel Table Lookup

➤ Example: 64 entry table, 8-Bit entries

Strategy

1. Divide table into two 32-Entry tables
2. Load table into vector registers
3. Load values to be looked up (indices) into VC
4. Do two permutes, one using the upper half of the table as inputs, one using the lower half
5. Do a vector compare greater than each index with constant 0001 1111. This results in 0X00 when the index is less than 31 (entry is in first half of table) and 0XFF when the index is greater than 31 (entry is in second half of table)
6. Use vector select with the result of the compare to select (MUX) the correct entry. This is equivalent to a MUX using sixth bit of the index as the select input.



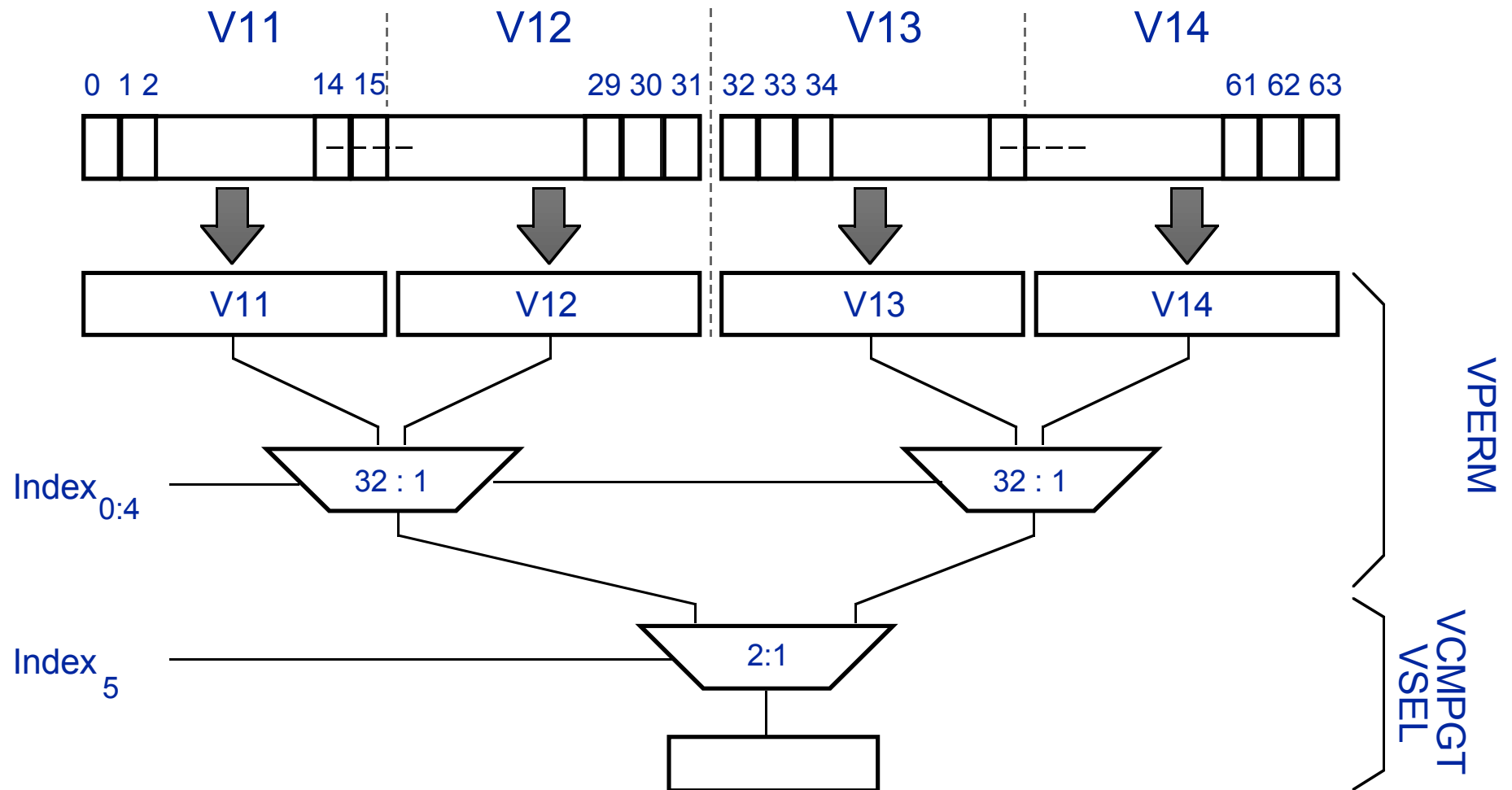


Parallel Table Lookup

- Parallel lookup is scalable:
 - table can be up to size of vector register file (minus a few)
 - 32 entry table = 16 lookup/inst
 - 64 entry table = 4 lookup/inst
 - 256 entry table = 4/5 lookup/inst
 - non-SIMD code ~ 1/2 lookup/inst but table size unlimited
- Can be further improved
 - piecewise linear approx.
- Applications
 - Galois Field multiplication (used in Reed-Solomon ECC)
 - image/video processing (color correction)



Parallel Table Lookup





Parallel Table Lookup

Altivec code for 64-entry parallel table lookup
16-element parallel lookup into a table of 64 byte elements

; Assume V31 holds the 16 valid 6-bit index values
; which are to be looked up from a 64-element table,
; contained in V11 through V14.
; Assume V02 holds the looked up values.
; Assume following replicated constant: 0b00011111, in V27.

vperm V00,V11,V12,V31 ; lookup within first 32 elems
vperm V01,V13,V14,V31 ; lookup within second 32 elems

vcmpgtub V08,V31,V27 ; this comparison with 0b00011111
 ; will splat/replicate the 6th bit

vsel V02,V00,V01,V08 ; use 6th bit for choosing between
 ; the two lookups



256-Byte Table

