VAX Vector Architecture

Dileep Bhandarkar

History

- VAX architecture announced in 1977
- Vectors considered for VAX 8600 in 1980
  - increased memory bandwidth requirement would add ~10% to cost of all systems
  - Vector processors on only ~5% of systems
  - no business case for vectors
- Vector architecture defined for PRISM in 1985
- Vector architecture ported to VAX in 1987-88

VAX Vector Architecture Goals

- Consistent with existing VAX architecture
  - Opcode space was not a problem
  - Memory Management
  - Exception handling
  - Vector/Scalar synchronization
- Implementation constraints of ongoing CPU
- Higher performance designs in the future
- Minimize OS impact

Major Design Decisions

- Vector register based, no mem-mem ops
- Allow maximum concurrency
  - multiple loads, store, operates concurrently
- Allow overlapped operation between scalar and vector
  - explicit synchronization by software
- Allow imprecise vector exceptions
- Support virtual memory

Basic Architecture

- Heavily influenced by CRAY-2
- 16 Vector Registers
  - 64 elements per register
  - 64 bits per element
  - each vector register occupies 1 VAX page!
- 7-bit Vector Length Register
- 64-bit Vector Mask Register
- Vector Count Register

Vector Instructions

- 2-byte extended VAX opcodes
- Use standard VAX operand specifiers
- 7 instruction classes
  - all instructions in a class has same number and types of operands
  - appear as 7 new instructions to scalar processor
  - allows block decoding by scalar processor
- Vector control word contains operand info

Vector control word

- 16-bit control word
- 4-bit Va, Vb, Vc fields select source and destination registers
- Masked Operation Enable (MOE) bit
- Match True/False (MTF) bit
- Exception enable (EXC) integer overflow and FP underflow, and Modify Intent (MI)

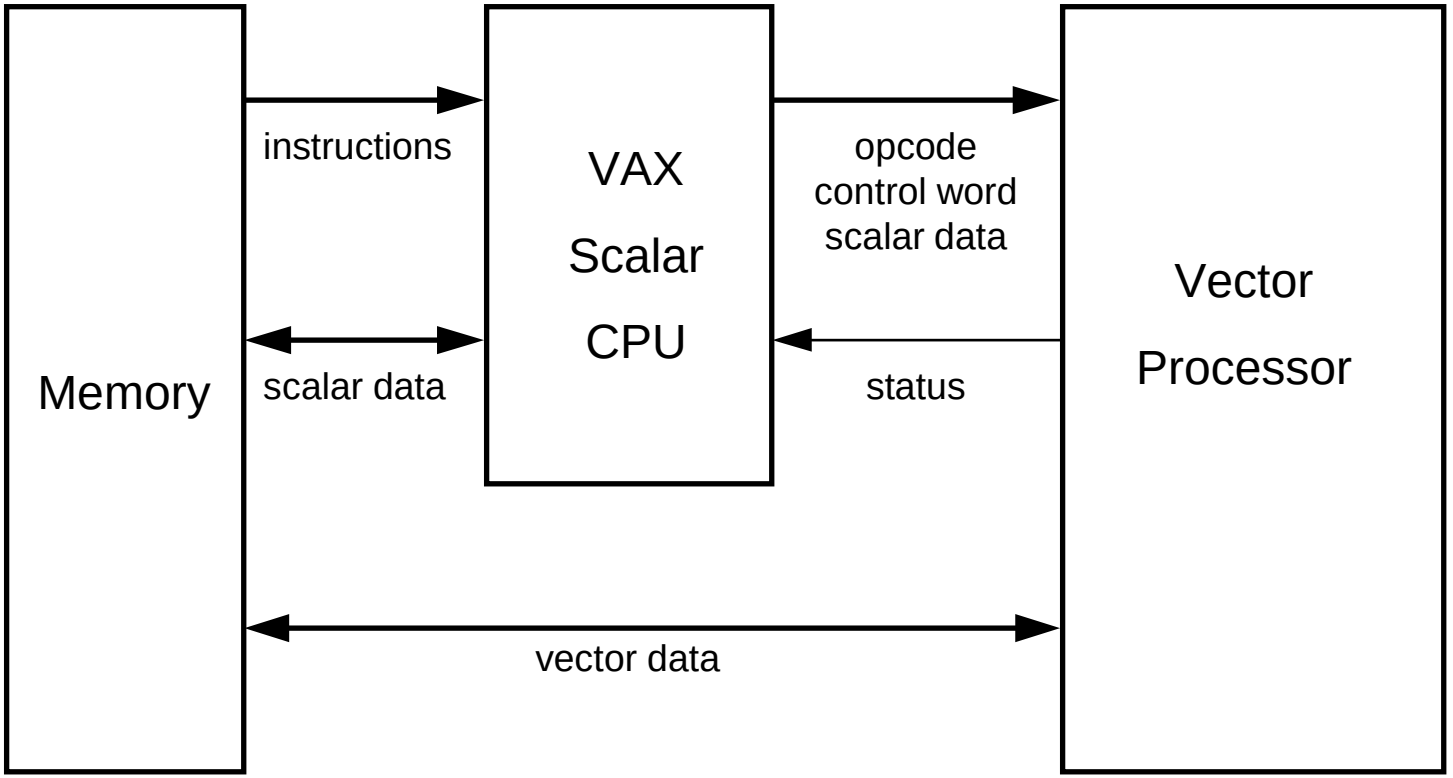
Instruction Classes

- Vector Memory with constant stride
  - opcode cntrl.rw, base.ab, stride.rl
- Vector Memory with random stride
  - opcode cntrl.rw, base.ab
- Vector-Scalar 32-bit arithmetic
  - opcode cntrl.rw, scalar.rl
- Vector-Scalar 64-bit arithmetic
  - opcode cntrl.rw, scalar.rq

Instruction Classes

- Vector-Vector arithmetic
  - opcode cntrl.rw
- Vector Control Register Read
  - opcode regnum.rw, dst.wl
- Vector Control Register Write
  - opcode regnum.rw, src.rl
- Assembler notation uses VADD V0,V1,V2

Execution Model



Execution Model

- Scalar processor decodes instruction and passes all operands and opcode to VP queue
- Scalar processor continues to execute other instructions
- Vector processor execution is decoupled
- Scalar can take interrupts and exceptions
- Vector exceptions do not disturb scalar CPU

Execution Model

- If VP is disabled, scalar CPU takes a VP disabled fault on an attempt to dispatch vector instruction
- Vector processor disables itself
  - on vector exception
  - input queue full
  - VP status register shows exceptions

Execution Model

- VP can execute any number of instructions concurrently subject to data dependencies
- Arithmetic instructions propagate bad data
- Exception synchronization through special instructions
- VP may have its own TB, cache etc
- Software manages scalar/vector data consistency

Memory Management Exceptions

- Synchronous or Asynchronous mode allowed
- Synchronous mode
  - scalar CPU waits for VP to signal no MME
  - On MME, scalar processor faults and dispatches through scalar exception vector
- Asynchronous mode
  - Scalar CPU does not wait for VP
  - VP disables itself on MME, sets VPSR, holds state

Context Switching

- VP state is distinct from scalar CPU
- VP operates asynchronously
- Scalar CPU need not wait for VP to complete
- Scalar disables VP, but VP continues to execute instructions in its queue
- Scalar CPU must ensure that VP memory activity is completed

Vector Processor Special Registers

- Vector Processor Status Register (VPSR)
- Vector Arithmetic Exception Register (VAER)
- Vector Memory Activity Check (VMAC)
- Vector TB Invalidate All (VTBIA)
- Vector State Address register (VSAR)

Synchronization

- Special instructions implemented as MFVP
- SYNC waits for VP to signal no exceptions
- MSYNC ensures that all writes are visible to other scalar/vector pairs
- MFPR VMAC ensures that vector writes are visible to scalar CPU
- VSYNC synchronizes memory access within a vector processor

Implementations

- VAXvector 6000 Model 400
  - implements synchronous MME
  - implements deferred instruction queue for one arithmetic instruction
  - implements chain into store
  - uses 1 MB cache, 136 entry TB
  - uses 4 non-pipelined FP chips to generate 2 double precision results every 44.4 ns cycle
  - Vector speed up of 25 for Linpack 1000

Implementations

- VAX 9000
  - implements synchronous MME
  - shares scalar CPU 128 KB cache and TLB
  - 500 MB/sec memory bandwidth
  - chains mul-add-store
  - register file has 5 read and 3 write ports
  - one ADD and one MUL result every cycle
  - achieves 80 Mflops on Linpack 1000 at 16 ns

Conclusions

- Vector extensions can be added to existing architectures
- Good performance requires overlapped execution
- Challenges in memory management and exception architectures can be overcome
- Caches do work
- Memory bandwidth is critical