# 15-740/18-740
# Computer Architecture
# Lecture 1: Processing Paradigms and Intro

Prof. Onur Mutlu
Carnegie Mellon University
Fall 2011, 9/12/2011

# Agenda

- Announcements
- Homework and reading for next time
- Projects
- Some fundamental concepts
  - Computer architecture
  - Levels of transformation
  - ISA vs. microarchitecture
  - Design point
  - Tradeoffs
    - ISA, microarchitecture, system/task
  - Von Neumann model
  - Performance equation and Amdahl's Law

# Last Time …

- Course logistics, info, requirements
  - See slides for Lecture 0 and syllabus online
- Homeworks and Review Sets for this week
- Readings for first week
  - Ronen et al., "Coming Challenges in Microarchitecture and Architecture," Proceedings of the IEEE, vol. 89, no. 11, 2001.
  - Y. N. Patt, "Requirements, bottlenecks, and good fortune: agents for microprocessor evolution," Proceedings of the IEEE, vol. 89, no. 11, 2001.
  - G. M. Amdahl "Validity of the single processor approach to achieving large scale computing capabilities," AFIPS Conference, April 1967.
  - G. E. Moore, "Cramming more components onto integrated circuits," Electronics, April 1965.

# Summary of Assignments This Week

- Homework 0
  - Due September 14 (Next Wed)
- Homework 1
  - Due September 16 (Next Fri)
- Review Set 0 (two readings)
  - Due September 14 (Next Wed)
- Review Set 1 (two seminal readings)
  - Due September 16 (Next Fri)
- Project ideas and groups
  - Proposal due September 26: Read, think, and brainstorm

# Reminder: How to Do the Paper Reviews

- Brief summary
  - What is the problem the paper is trying to solve?
  - What are the key ideas of the paper? Key insights?
  - What is the key contribution to literature at the time it was written?
  - What are the most important things you take out from it?
- Strengths (most important ones)
  - Does the paper solve the problem well?
- Weaknesses (most important ones)
  - This is where you should think critically. Every paper/idea has a weakness. This does not mean the paper is necessarily bad. It means there is room for improvement and future research can accomplish this.
- Can you do (much) better? Present your thoughts/ideas.
- What have you learned/enjoyed most in the paper? Why?

- Review should be short and concise (~half a page or shorter)

# Research Project (I)

- Your chance to explore in depth a computer architecture topic that interests you
- Your chance to publish your innovation in a top computer architecture/systems conference.

- Start thinking about your project topic from now!
- Interact with me and Yoongu, Justin

- Groups of 3
- Proposal due: Sep 26

- See website for project handout and topics

# Research Project (II)

- Goal:
  - Develop new insight
  - Approach 1:
    - Develop novel ideas to solve an important problem
    - Rigorously evaluate the benefits and limitations of the ideas
  - Approach 2:
    - Derive insight from rigorous analysis and understanding of previously proposed ideas
    - Propose potential new solutions based on the new insight
- The problem and ideas need to be concrete
- You should be doing problem-oriented research

# Research Proposal Outline

- The Problem: What is the problem you are trying to solve
  - Define clearly.
- Novelty: Why has previous research not solved this problem? What are its shortcomings?
  - Describe/cite all relevant works you know of and describe why these works are inadequate to solve the problem.
- Idea: What is your initial idea/insight? What new solution are you proposing to the problem? Why does it make sense? How does/could it solve the problem better?
- Hypothesis: What is the main hypothesis you will test?
- Methodology: How will you test the hypothesis/ideas? Describe what simulator or model you will use and what initial experiments you will do.
- Plan: Describe the steps you will take. What will you accomplish by Milestone 1, 2, 3, and Final Report? Give 75%, 100%, 125% and moonshot goals.

*All research projects can be and should be described in this fashion.*

# Heilmeier's Catechism (version 1)

- What are you trying to do? Articulate your objectives using absolutely no jargon.
- How is it done today, and what are the limits of current practice?
- What's new in your approach and why do you think it will be successful?
- Who cares?
- If you're successful, what difference will it make?
- What are the risks and the payoffs?
- How much will it cost?
- How long will it take?
- What are the midterm and final "exams" to check for success?

# Heilmeier's Catechism (version 2)

- What is the problem?
- Why is it hard?
- How is it solved today?
- What is the new technical idea?
- Why can we succeed now?
- What is the impact if successful?

# Readings Referenced Today

- ISA and Compilers (Review Set 2)
  - Colwell et al., "Instruction Sets and Beyond: Computers, Complexity, and Controversy," IEEE Computer 1985.
  - Wulf, "Compilers and Computer Architecture," IEEE Computer 1981.

- Alternative ISA and execution models
  - Fisher, "Very Long Instruction Word Architectures and the ELI-512," ISCA 1983.
  - Russell, "The CRAY-1 computer system," CACM 1978.
  - Lindholm et al., "NVIDIA Tesla: A Unified Graphics and Computing Architecture," IEEE Micro 2008.

# Readings Referenced Today (Perhaps)

- On-chip networks
  - Dally and Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," DAC 2001.
  - Wentzlaff et al., "On-Chip Interconnection Architecture of the Tile Processor," IEEE Micro 2007.

- Main memory controllers
  - Moscibroda and Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," USENIX Security 2007.
  - Rixner et al., "Memory Access Scheduling," ISCA 2000.

- Architecture reference manuals
  - Digital Equipment Corp., "VAX11 780 Architecture Handbook," 1977-78.
  - Intel Corp. "Intel 64 and IA-32 Architectures Software Developer's Manual"

# Review Set 2

- Colwell et al., "Instruction Sets and Beyond: Computers, Complexity, and Controversy," IEEE Computer 1985.
- Wulf, "Compilers and Computer Architecture," IEEE Computer 1981.

- Due next Monday

# Comp Arch @ Carnegie Mellon

- Computer Architecture Lab at Carnegie Mellon (CALCM) @ www.ece.cmu.edu/CALCM
- Send mail to calcm-list-request@ece
  - body: subscribe calcm-list

- Seminars
  - CALCM weekly seminar
  - SDI weekly seminar

# CALCM Seminar Wednesday

- "Efficient, Heterogeneous, Parallel Processing: The Design of a Micropolygon Rendering Pipeline"
- Kayvon Fatahalian, CMU
- 4-5 pm, September 14, Wednesday
- Hamerschlag Hall D210
- http://www.ece.cmu.edu/~calcm/doku.php?id=seminars:seminar_11_09_14

- Attend and provide a review online
  - Review Set 3

# Data Parallelism

- Concurrency arises from performing the same operations on different pieces of data
  - Single instruction multiple data (SIMD)
  - E.g., dot product of two vectors
- Contrast with thread ("control") parallelism
  - Concurrency arises from executing different threads of control in parallel
- Contrast with data flow
  - Concurrency arises from executing different operations in parallel (in a data driven manner)
- SIMD exploits instruction-level parallelism
  - Multiple instructions concurrent: instructions happen to be the same

# SIMD Processing

- Single instruction operates on multiple data elements
  - In time or in space
- Multiple processing elements

- Time-space duality
  - Array processor: Instruction operates on multiple data elements at the same time
  - Vector processor: Instruction operates on multiple data elements in consecutive time steps

# SIMD Processing

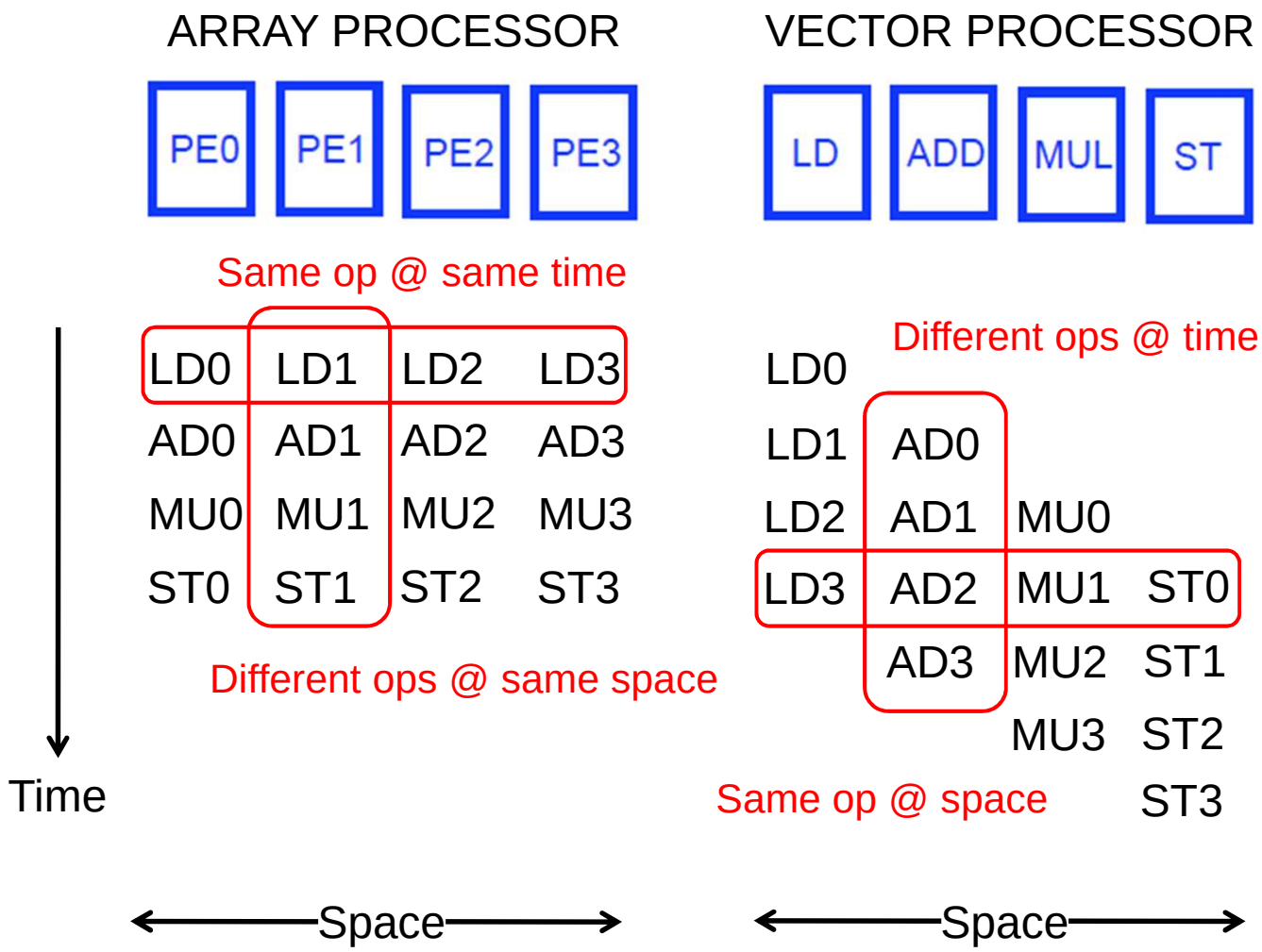
- Single instruction operates on multiple data elements
  - In time or in space
- Multiple processing elements

- Time-space duality
  - Array processor: Instruction operates on multiple data elements at the same time
  - Vector processor: Instruction operates on multiple data elements in consecutive time steps

# Array vs. Vector Processors

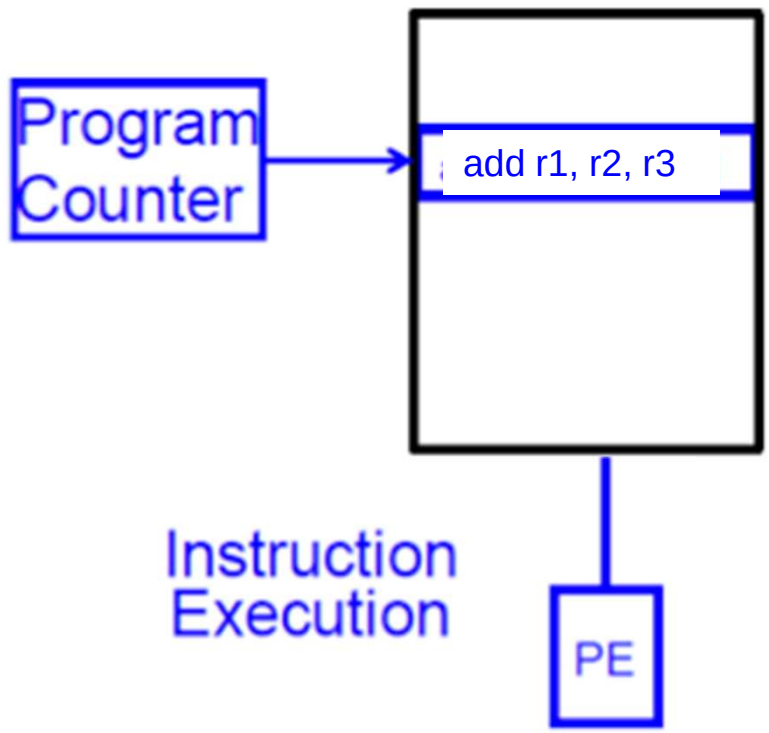
Instruction Stream

LD VR ← A[3:0]
ADD VR ← VR, 1
MUL VR ← VR, 2
ST A[3:0] ← VR



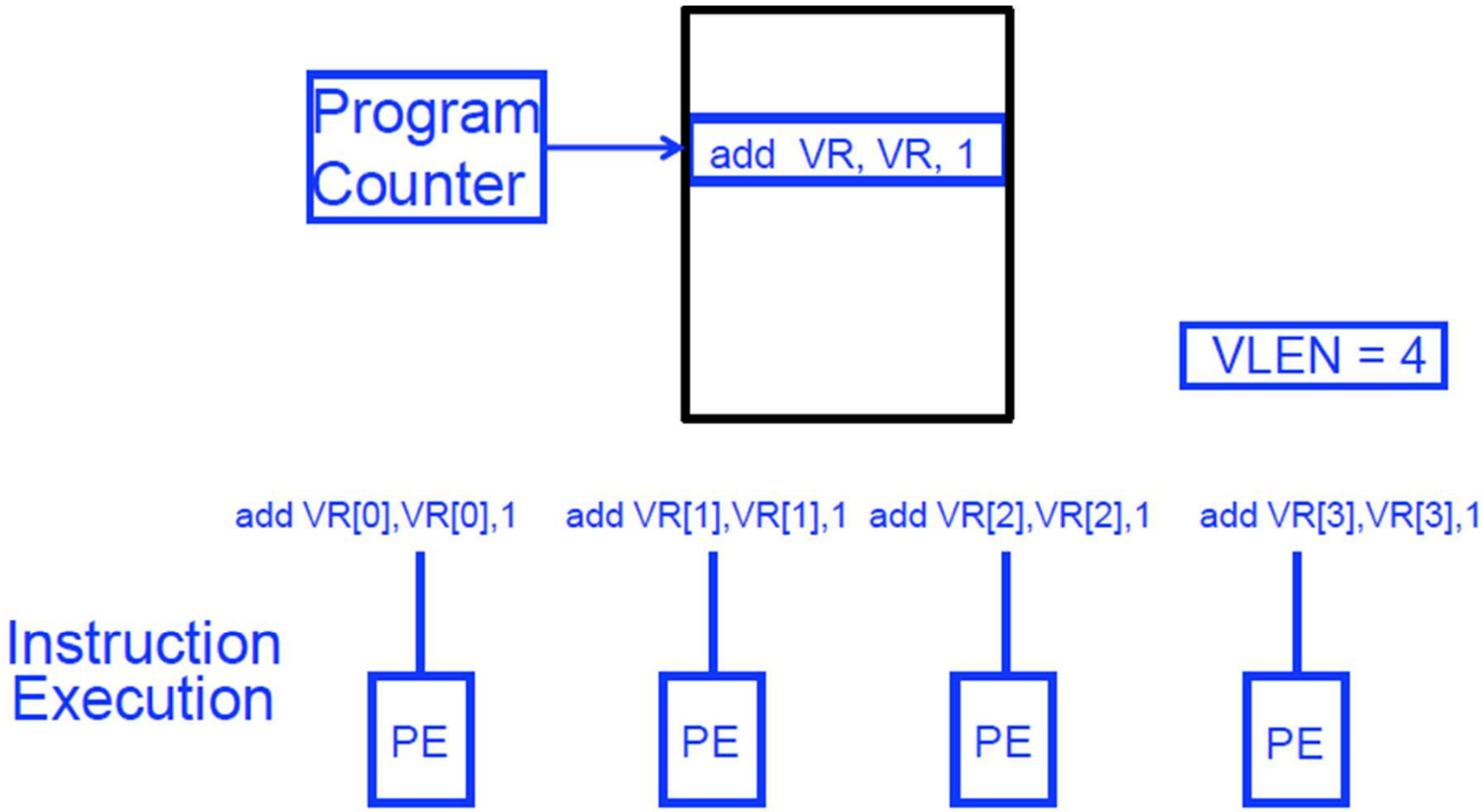
# Scalar Processing

- Conventional form of processing (von Neumann model)
  - We will get back to this later today and this week



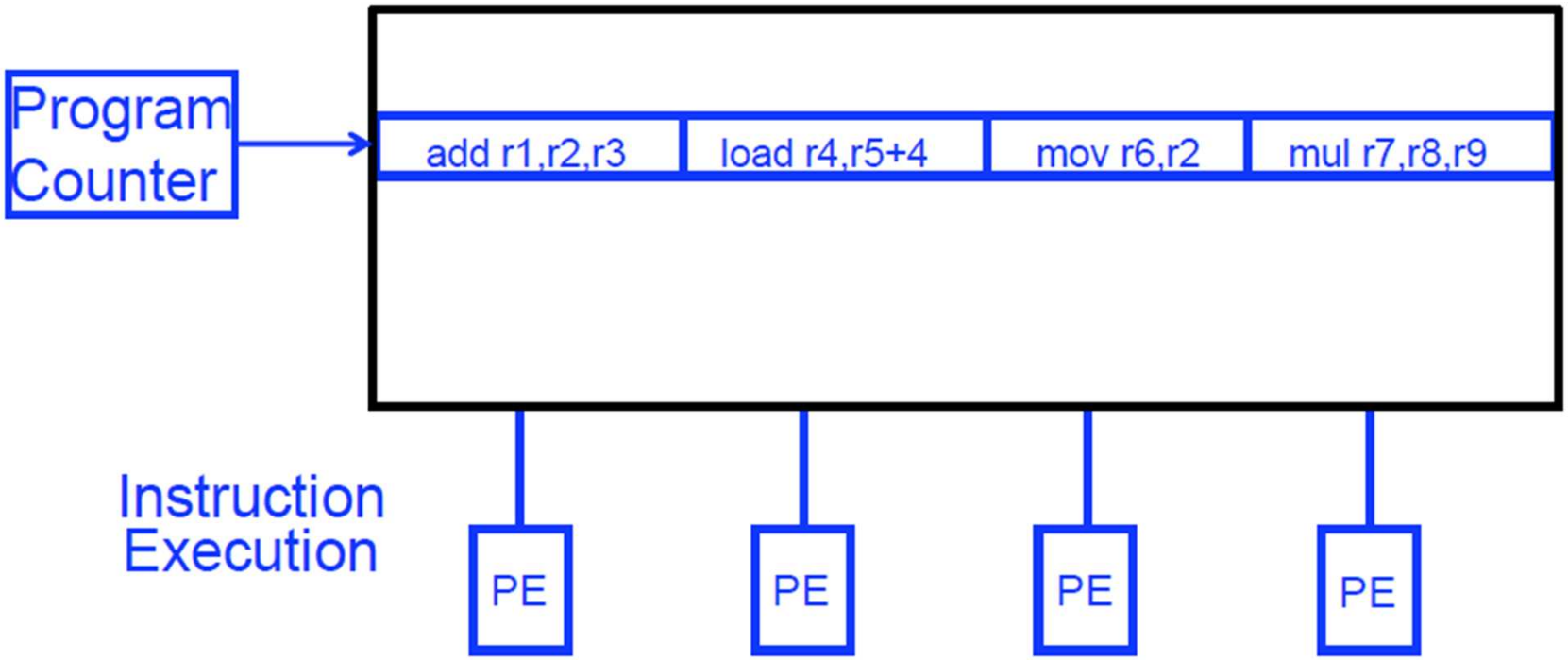
# SIMD Array Processing

- Array processor



# VLIW Processing

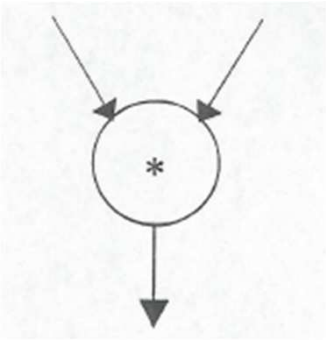
- Very Long Instruction Word

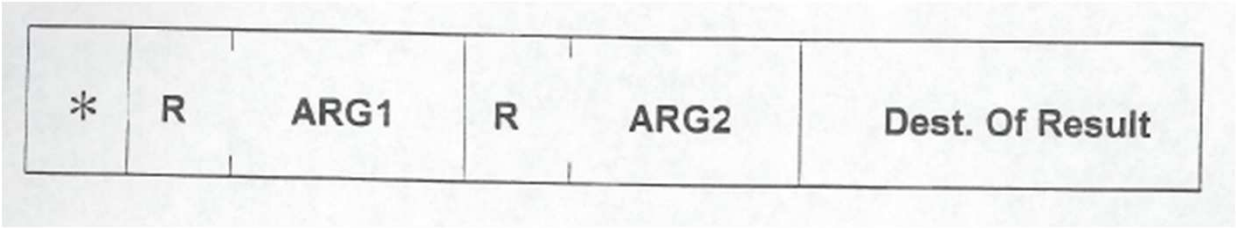


# Data Flow Processing

- No program counter
- In a data flow machine, a program consists of data flow nodes
- A data flow node fires (fetched and executed) when all its inputs are ready
  - i.e. when all inputs have tokens

- Data flow node and its ISA representation



# Vector Processors

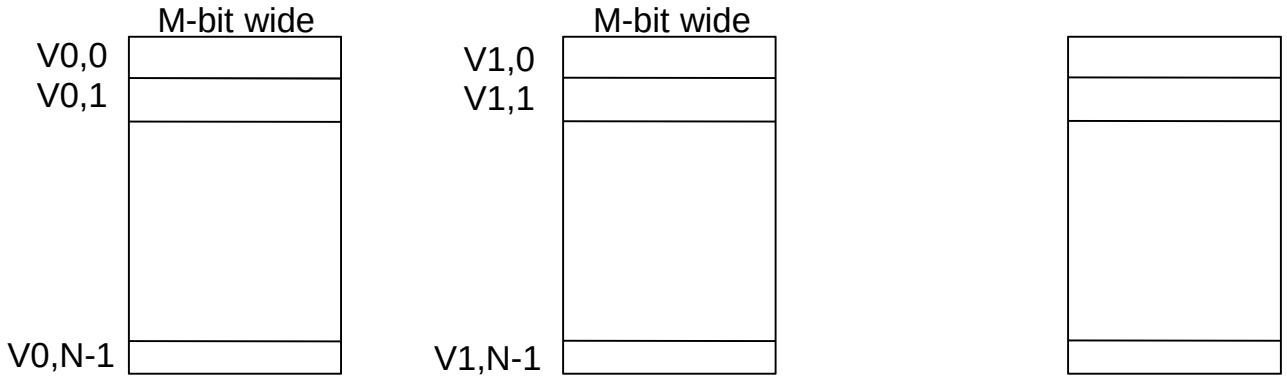
- A vector is a one-dimensional array of numbers
- Many scientific/commercial programs use vectors

```
for (i = 0; i<=49; i++)
    C[i] = (A[i] + B[i]) / 2
```

- A vector processor is one whose instructions operate on vectors rather than scalar (single data) values
- Basic requirements
  - Need to load/store vectors → vector registers (contain vectors)
  - Need to operate on vectors of different lengths → vector length register (VLEN)
  - Elements of a vector might be stored apart from each other in memory → vector stride register (VSTR)
    - Stride: distance between two elements of a vector

# Vector Registers

- Each vector data register holds N M-bit values
- Vector control registers: VLEN, VSTR, VMASK
- Vector Mask Register (VMASK)
  - Indicates which elements of vector to operate on
  - Set by vector test instructions
    - e.g., VMASK[i] = ($V_k$[i] == 0)
- Maximum VLEN can be N
  - Maximum number of elements stored in a vector register



# Vector Processors (II)

- A vector instruction performs an operation on each element in consecutive cycles
  - Vector functional units are pipelined
  - Each pipeline stage operates on a different data element

- Vector instructions allow deeper pipelines
  - No intra-vector dependencies → no hardware interlocking within a vector
  - No control flow within a vector
  - Known stride allows prefetching of vectors into memory

# Vector Processor Advantages

+ No dependencies within a vector
  - Pipelining, parallelization work well
  - Can have very deep pipelines, no dependencies!

+ Each instruction generates a lot of work
  - Reduces instruction fetch bandwidth

+ Highly regular memory access pattern
  - Interleaving multiple banks for higher memory bandwidth
  - Prefetching

+ No need to explicitly code loops
  - Fewer branches in the instruction sequence

# Vector ISA Advantages

- Compact encoding
  - one short instruction encodes N operations
- Expressive, tells hardware that these N operations:
  - are independent
  - use the same functional unit
  - access disjoint registers
  - access registers in same pattern as previous instructions
  - access a contiguous block of memory (unit-stride load/store)
  - access memory in a known pattern (strided load/store)
- Scalable
  - can run the same code in parallel pipelines (*lanes)*

# Scalar Code Example

- For I = 1 to 50
  - C[i] = (A[i] + B[i]) / 2

- Scalar code

```
    MOVI R0 = 50                1
    MOVA R1 = A                 1
    MOVA R2 = B                 1
    MOVA R3 = C                 1
X:  LD R4 = MEM[R1++]           11  ;autoincrement addressing
    LD R5 = MEM[R2++]           11
    ADD R6 = R4 + R5            4
    SHFR R7 = R6 >> 1           1
    ST MEM[R3++] = R7           11
    DECBNZ --R0, X              2   ;decrement and branch if NZ
```

304 dynamic instructions

# Vector Code Example

- A loop is vectorizable if each iteration is independent of any other
- For I = 0 to 49
  - C[i] = (A[i] + B[i]) / 2

7 dynamic instructions

- Vectorized loop:

| | |
|---|---|
| MOVI VLEN = 50 | 1 |
| MOVI VSTR = 1 | 1 |
| VLD V0 = A | 11 + VLN - 1 |
| VLD V1 = B | 11 + VLN – 1 |
| VADD V2 = V0 + V1 | 4 + VLN - 1 |
| VSHFR V3 = V2 >> 1 | 1 + VLN - 1 |
| VST C = V3 | 11 + VLN – 1 |

We did not cover the following slides in lecture.
These are for your preparation for the next lecture.

# Scalar Code Execution Time

- Scalar execution time on an in-order processor with 1 bank
  - First two loads in the loop cannot be pipelined 2*11 cycles
  - 4 + 50*40 = 2004 cycles

- Scalar execution time on an in-order processor with 16 banks (word-interleaved)
  - First two loads in the loop can be pipelined
  - 4 + 50*30 = 1504 cycles

- Why 16 banks?
  - 11 cycle memory access latency
  - Having 16 (>11) banks ensures there are enough banks to overlap enough memory operations to cover memory latency

# Vector Code Execution Time

- No chaining
  - i.e., output of a vector functional unit cannot be used as the input of another (i.e., no vector data forwarding)
- 16 memory banks (word-interleaved)


- 285 cycles

# Vector Processor Disadvantages

-- Works (only) if parallelism is regular (data/SIMD parallelism)

++ Vector operations

-- Very inefficient if parallelism is irregular

-- How about searching for a key in a linked list?

To program a vector machine, the compiler or hand coder must make the data structures in the code fit nearly exactly the regular structure built into the hardware. That's hard to do in first place, and just as hard to change. One tweak, and the low-level code has to be rewritten by a very smart and dedicated programmer who knows the hardware and often the subtleties of the application area. Often the rewriting is

Fisher, "Very Long Instruction Word Architectures and the ELI-512," ISCA 1983.

# Vector Processor Limitations

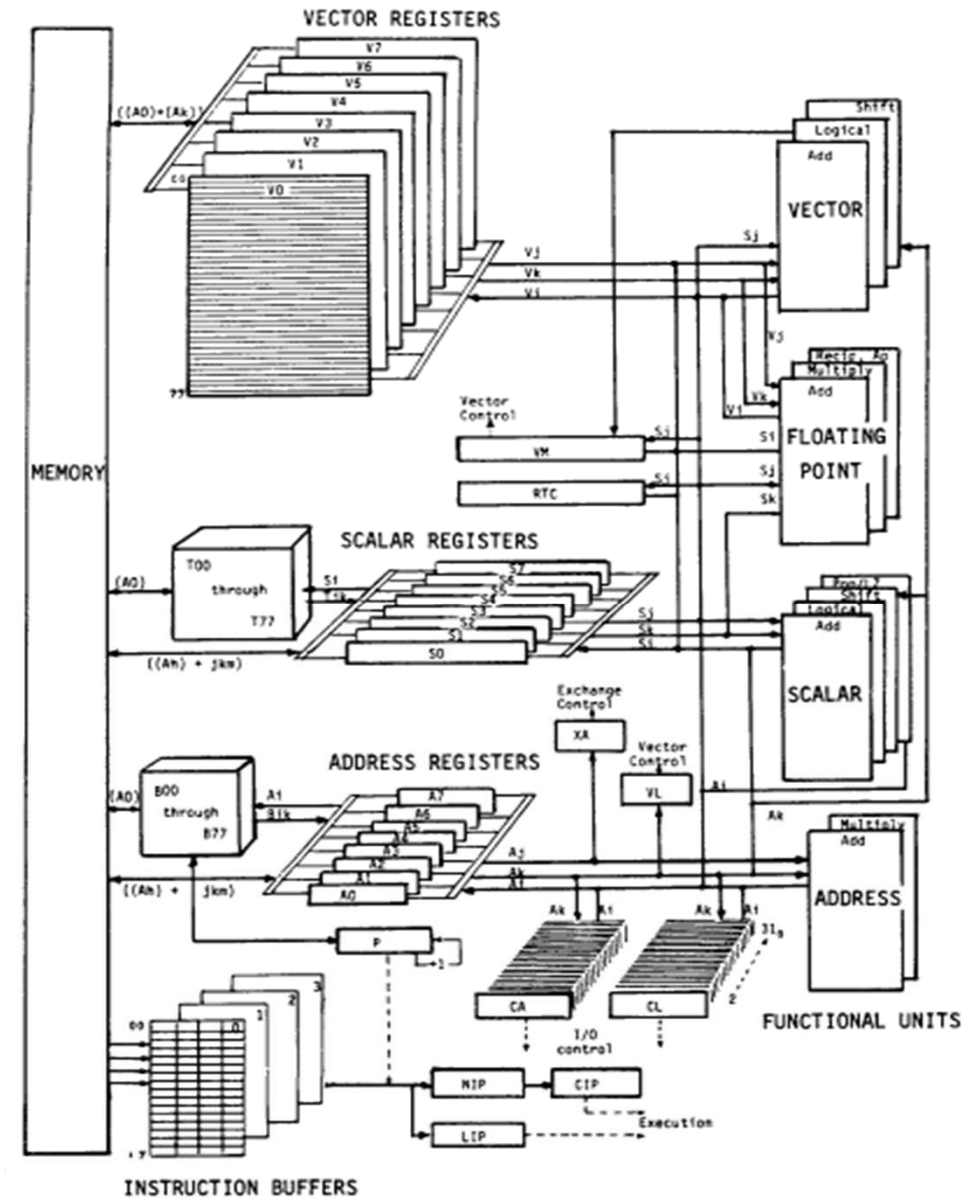
-- Memory (bandwidth) can easily become a bottleneck, especially if

1. compute/memory operation balance is not maintained
2. data is not mapped appropriately to memory banks

# Further Reading: SIMD and GPUs

- Recommended
  - H&P, Appendix on Vector Processors
  - Russell, "The CRAY-1 computer system," CACM 1978.
  - Lindholm et al., "NVIDIA Tesla: A Unified Graphics and Computing Architecture," IEEE Micro 2008.

# Vector Machine Example: CRAY-1




- CRAY-1
- Russell, "The CRAY-1 computer system," CACM 1978.

- Scalar and vector modes
- 8 64-element vector registers
- 64 bits per element
- 16 memory banks
- 8 64-bit scalar registers
- 8 24-bit address registers