

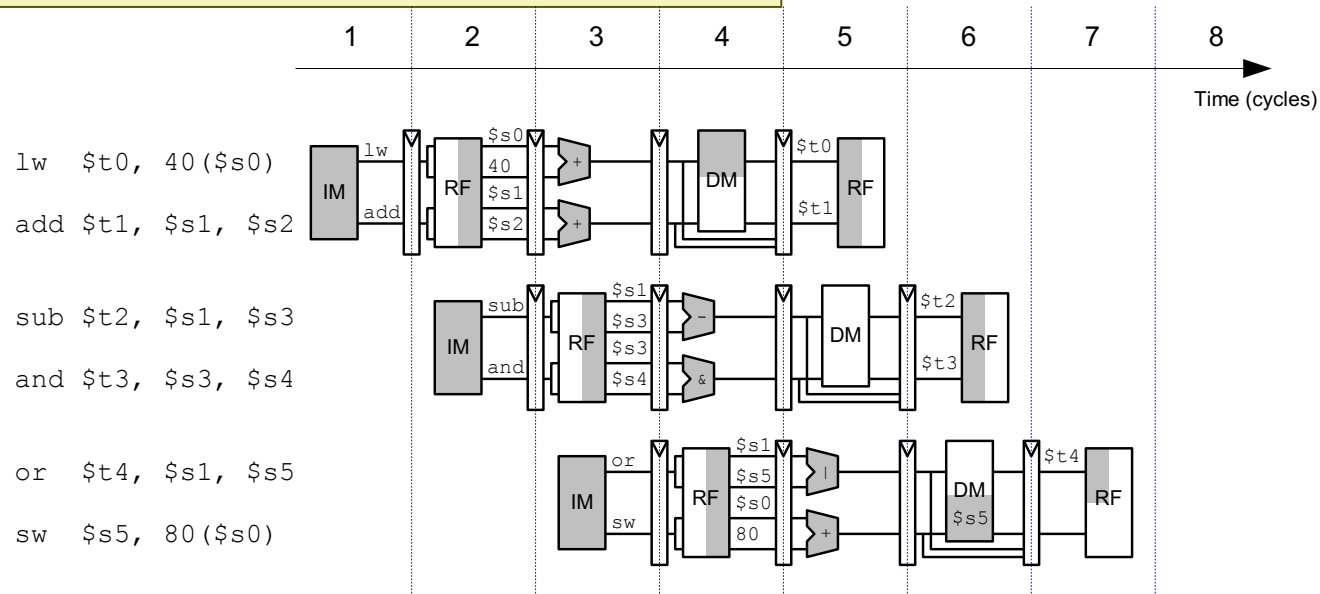
Superscalar Execution

- Idea: Fetch, decode, execute, retire **multiple instructions per cycle**
 - N-wide superscalar → N instructions per cycle
- Need to add the hardware resources for doing so
- **Hardware** performs the **dependence checking between concurrently-fetched instructions**
- **Superscalar** execution and **out-of-order** execution are **orthogonal concepts**
 - Can have all four combinations of processors:
[in-order, out-of-order] x [scalar, superscalar]

In-Order Superscalar Performance Example

```
lw $t0, 40($s0)
add $t1, $s1, $s2
sub $t2, $s1, $s3
and $t3, $s3, $s4
or $t4, $s1, $s5
sw $s5, 80($s0)
```

Ideal IPC = 2



Actual IPC = 2 (6 instructions issued in 3 cycles)

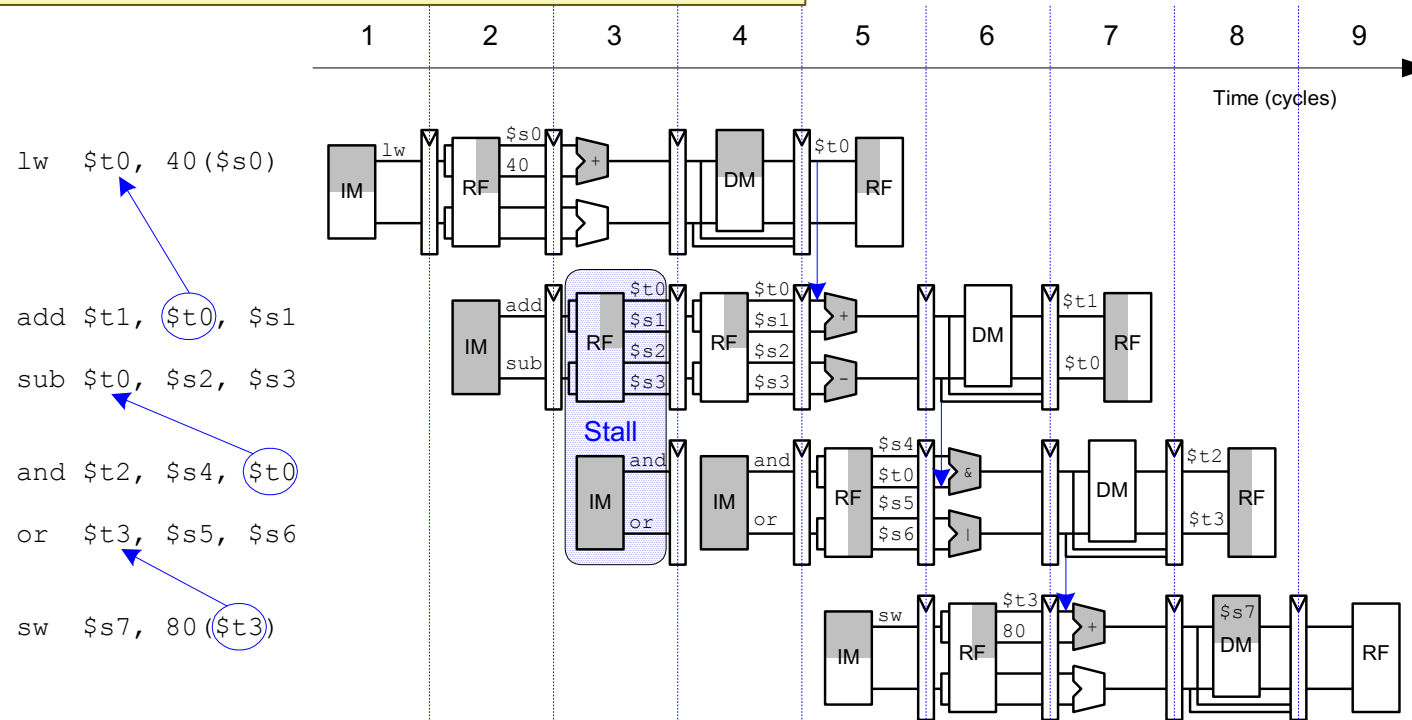
Superscalar Performance with Dependences

```

lw  $t0, 40($s0)
add $t1, $t0, $s1
sub $t0, $s2, $s3
and $t2, $s4, $t0
or  $t3, $s5, $s6
sw  $s7, 80($t3)
    
```

Ideal IPC = 2

Can you reorder the instructions to get IPC = 2?

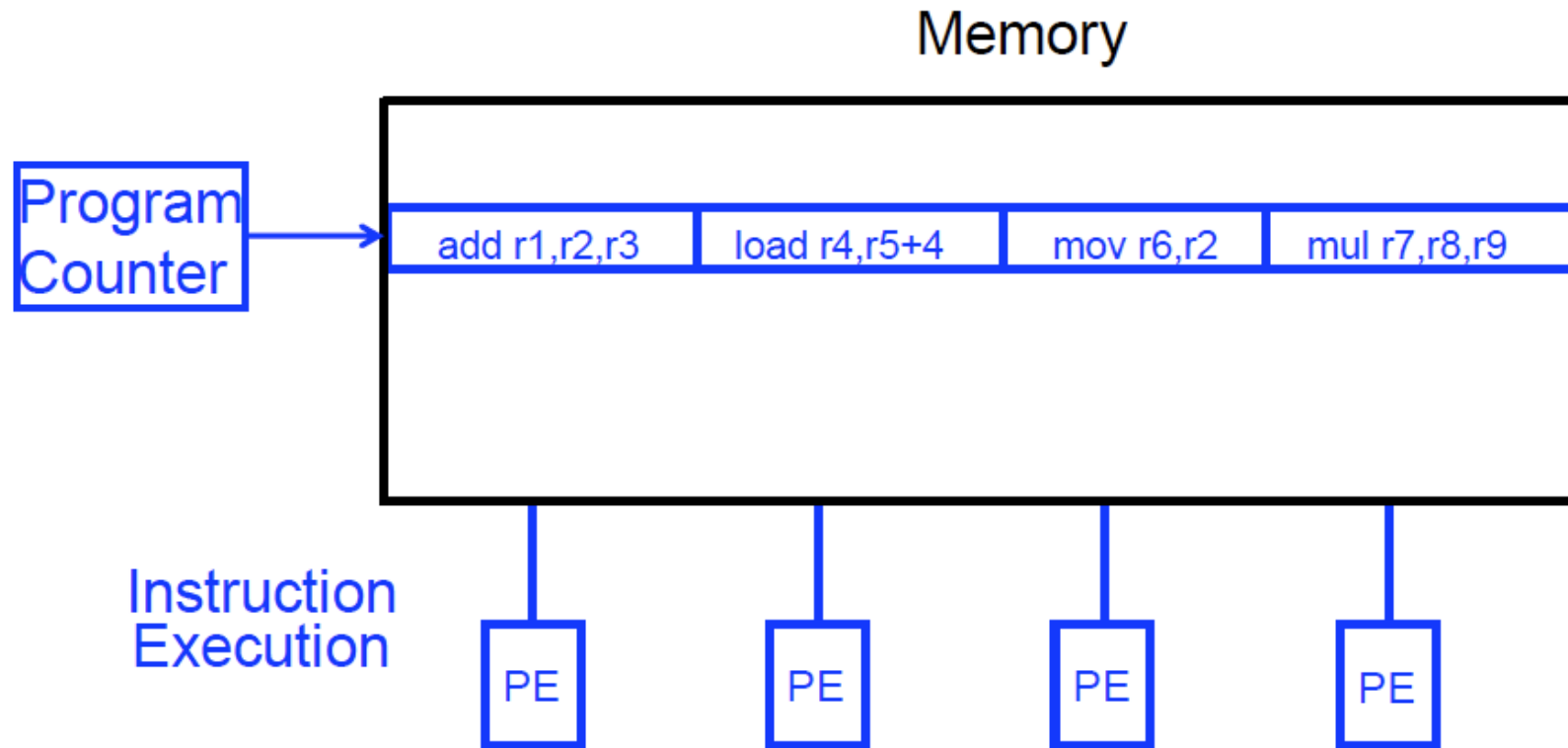


Actual IPC = 1.2 (6 instructions issued in 5 cycles)

VLIW Concept

- Superscalar
 - **Hardware** fetches multiple instructions and checks dependencies between them
- VLIW (Very Long Instruction Word)
 - **Software (compiler) packs independent instructions** in a larger “instruction bundle” to be fetched and executed concurrently
 - Hardware fetches and executes the instructions in the bundle concurrently
- **No need for hardware dependency checking** between concurrently-fetched instructions **in the VLIW model**
 - **Simple hardware, complex compiler**

VLIW Concept



- Fisher, “**Very Long Instruction Word architectures and the ELI-512,**” ISCA 1983.
 - ELI: Enormously longword instructions (512 bits)

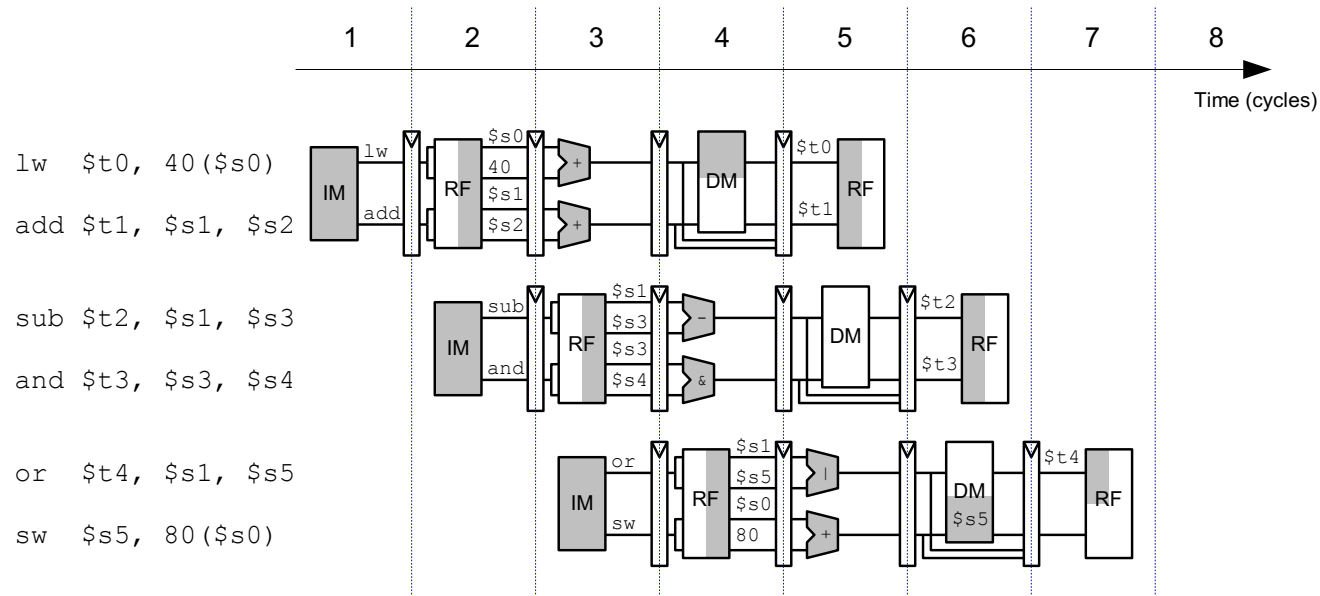
VLIW (Very Long Instruction Word)

- A very long instruction word consists of **multiple independent instructions packed together by the compiler**
 - Packed instructions can be logically unrelated (contrast with SIMD/vector processors, which we will see soon)
- Idea: Compiler finds independent instructions and statically schedules (i.e. packs/bundles) them into a single VLIW instruction
- Traditional VLIW Characteristics
 1. Multiple instruction fetch/execute, multiple functional units
 2. All instructions in a bundle are executed in lock step
 3. Instructions in a bundle statically aligned to be directly supplied into the functional units

VLIW Performance Example (2-wide bundles)

<code>lw \$t0, 40(\$s0)</code>	<code>add \$t1, \$s1, \$s2</code>	Bundle 1
<code>sub \$t2, \$s1, \$s3</code>	<code>and \$t3, \$s3, \$s4</code>	Bundle 2
<code>or \$t4, \$s1, \$s5</code>	<code>sw \$s5, 80(\$s0)</code>	Bundle 3

Ideal IPC = 2



Actual IPC = 2 (6 instructions issued in 3 cycles)

Flynn's Taxonomy of Computers

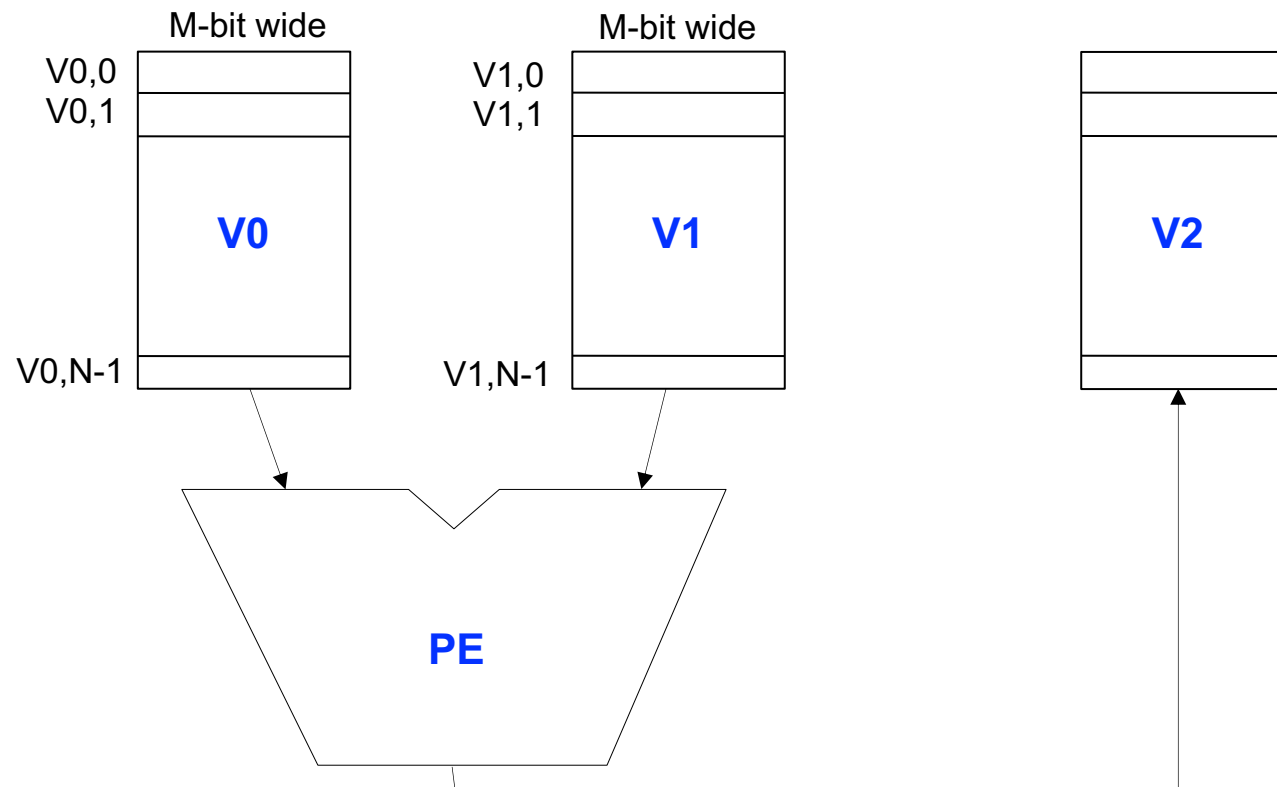
- Mike Flynn, “**Very High-Speed Computing Systems**,” Proc. of IEEE, 1966
- **SISD**: Single instruction operates on single data element
- **SIMD**: Single instruction operates on multiple data elements
 - Array processor
 - Vector processor
- **MISD**: Multiple instructions operate on single data element
 - Closest form: systolic array processor, streaming processor
- **MIMD**: Multiple instructions operate on multiple data elements (multiple instruction streams)
 - Multiprocessor
 - Multithreaded processor

SIMD Processing Paradigm

- Single instruction operates on multiple data elements
 - In time or in space
- Multiple processing elements (PEs), i.e., execution units
- Time-space duality
 - **Array processor**: Instruction operates on multiple data elements at the **same time** using **different spaces (PEs)**
 - **Vector processor**: Instruction operates on multiple data elements in **consecutive time steps** using the **same space (PE)**

Storing Multiple Data Elements: Vector Registers

- Each **vector data register** holds N M-bit values
 - Each register stores a vector
 - Not a (single) scalar value as we saw before



Array vs. Vector Processors

ARRAY PROCESSOR

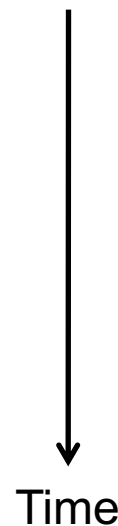


VECTOR PROCESSOR

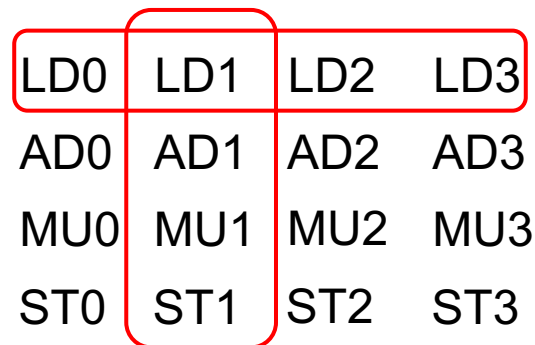


Instruction Stream

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



Same op @ same time



Different ops @ same space



Different ops @ time



Same op @ space



Vector Processors (I)

- 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 in memory between two elements of a vector

Vector Stride Example: Matrix Multiply

- A and B matrices, both stored in memory in **row-major order**

→

A ₀	0	1	2	3	4	5
	6	7	8	9	10	11

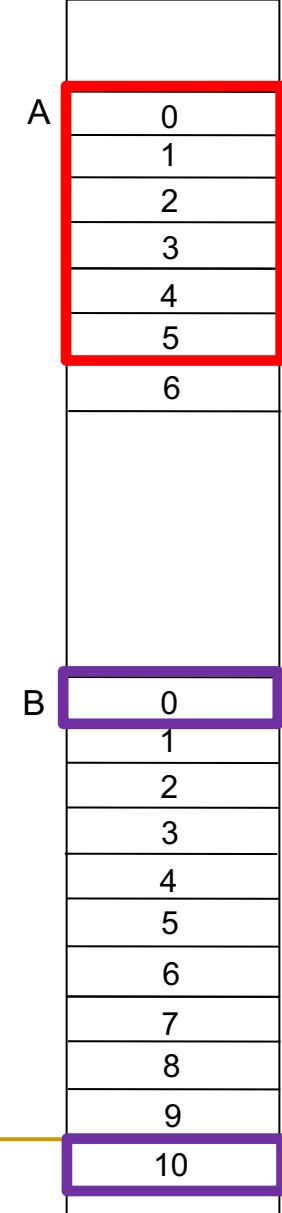
$$A_{4 \times 6} B_{6 \times 10} \rightarrow C_{4 \times 10}$$

↓

B ₀	0	1	2	3	4	5	6	7	8	9
	10	11	12	13	14	15	16	17	18	19
	20									
	30									
	40									
	50									

Dot product of each row vector of A with each column vector of B

Linear Memory



- Load A's row 0 (A0 through A5) into vector register V_1
 - Each time, increment address by **1** to access the next column
 - Accesses have a **stride of 1**
- Load B's column 0 (B0 through B50) into vector register V_2
 - Each time, increment address by **10** to access the next row
 - Accesses have a **stride of 10**