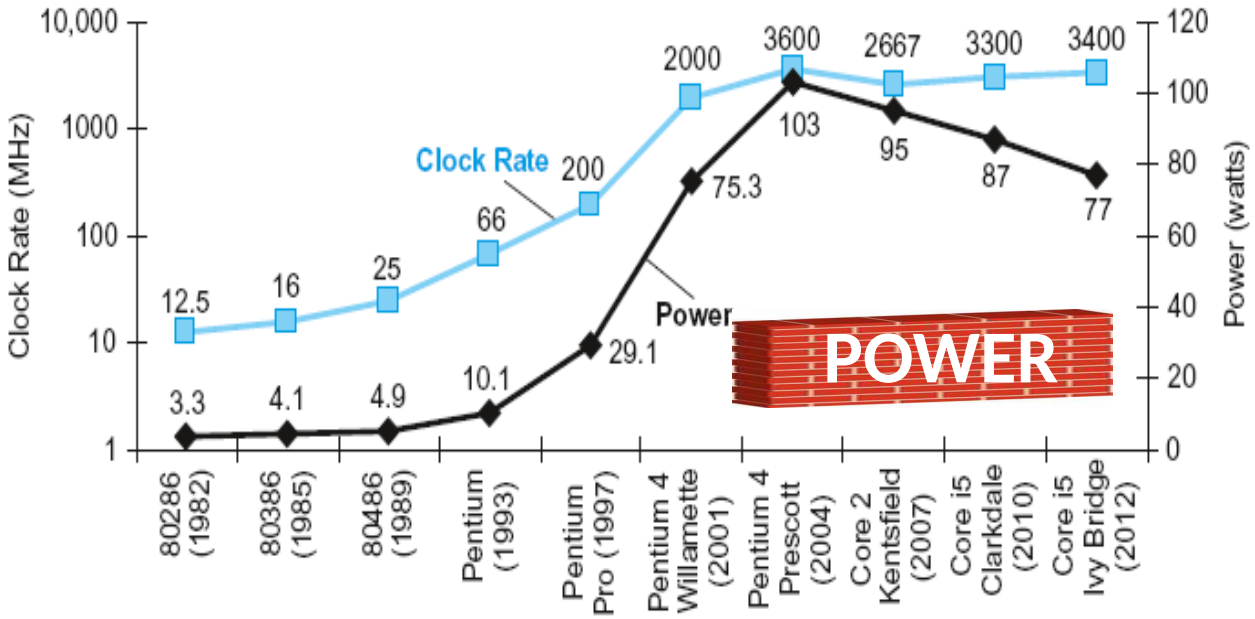
Parallel Processing from Client to Cloud

Dr. Ahmed Zahran

Power Wall



In Complementary Metal Oxide Semiconductor (CMOS) IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30

5V → 1V

×1000

Power wall refers to the inability to improve the performance and maintain low power consumption due to leak current in CMOS.
speeding the clock → expensive cooling requirements.

Performance is improved by increasing the processor throughput (parallel processors) rather than response time (speeding the clock).

Hardware and Software Parallelism

		Software	
		Sequential	Concurrent
Hardware	Serial	Matrix Multiply written in MatLab running on an Intel Pentium 4	Windows Vista Operating System running on an Intel Pentium 4
	Parallel	Matrix Multiply written in MATLAB running on an Intel Core i7	Windows Vista Operating System running on an Intel Core i7

parallel processing program

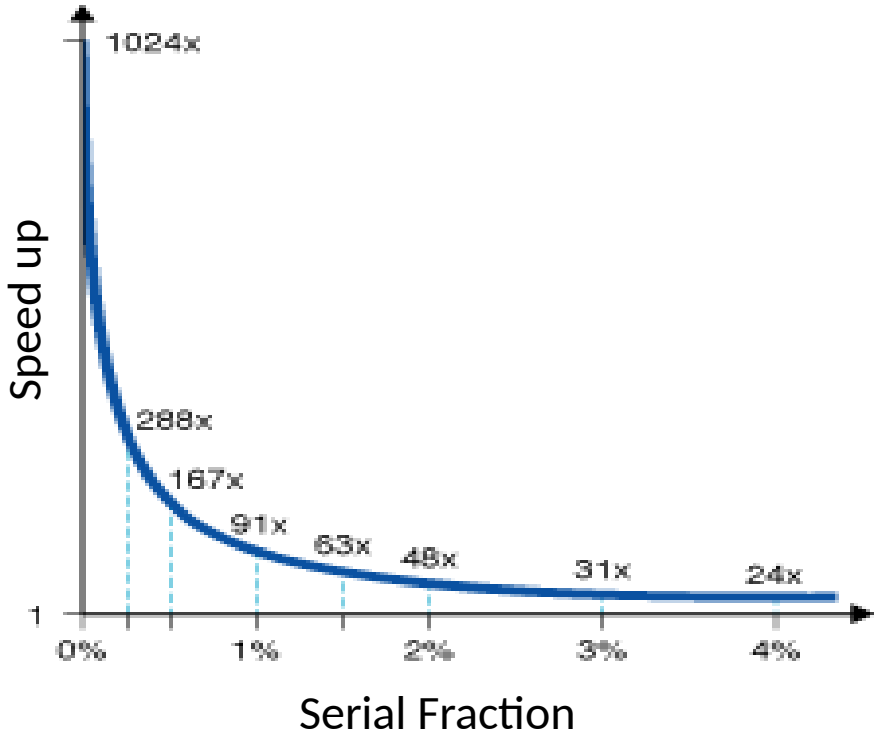
task-level parallelism

Sequential/concurrent software can run on serial/parallel hardware

- Challenge: making effective use of parallel hardware

Why is it difficult to write parallel processing programs?

- Involve tasks
  - Partitioning the work into parallel pieces
  - Scheduling (Balancing the load evenly between the workers)
  - Synchronization
  - Minimize overhead communications
- Amdahl's Law
  - Serial tasks



$$\text{Speed-up} = \frac{1}{(1 - \text{Fraction time affected}) + \frac{\text{Fraction time affected}}{\text{Amount of improvement}}}$$

% of parallelizable tasks

Parallel Processing Program

- Consider the following program with 100 tasks can be parallelized and 10 tasks are serial
  - With 10 processors
    - execution time = 10 + 100/10 = 20 time units
    - Speed up = 110t / 20t = 5.5
  - With 40 processors
    - Execution time = 10 + 100/40 = 12.5 time units
    - Speed up = 110 /12.5 = 8.8
- Increasing the size of the parallelizable jobs increases the speed up
  - Check 100 tasks → 400 tasks
- Unbalanced processor load reduces the speed-up factor.
  - One processor gets 20% parallelizable jobs → exec. time = 10 + max(0.2*100 /1, 0.8*100/9)= 30 time units → speedup = 110 / 30 = 3.67

Forms of Parallelism [Flynn's taxonomy]

Single instruction stream and single data stream (Conventional processors)

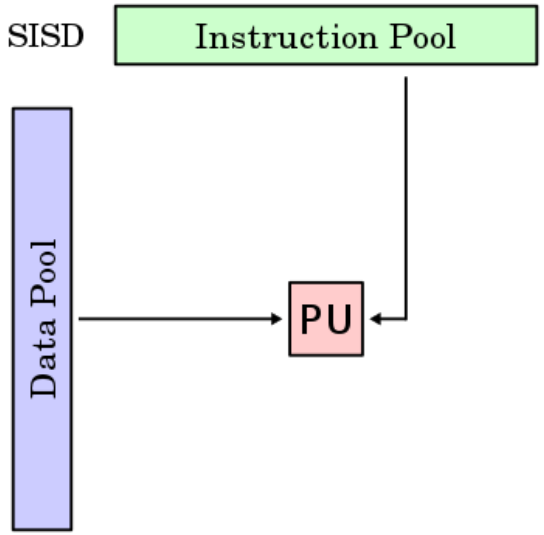
Single instruction stream with multiple data streams

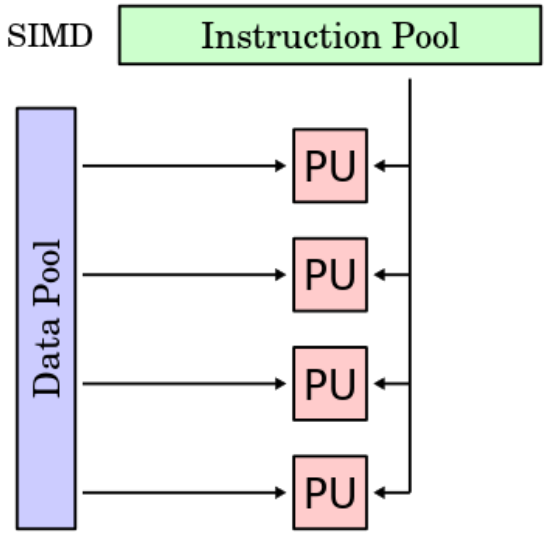
		Data Streams	
		Single	**Multiple**
Instruction Streams	Single	SISD: Intel Pentium 4	SIMD: SSE instructions of x86
	Multiple	MISD: No examples today	MIMD: Intel Core i7

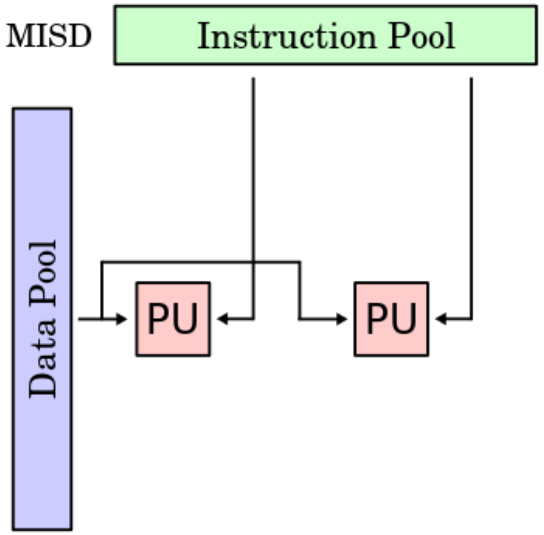
Uncommon – Fault tolerance

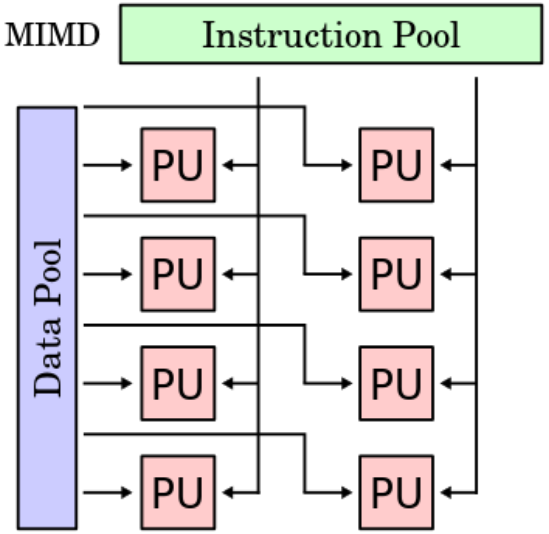
Multiple instruction stream with multiple data streams (e.g., multiprocessors, multiple computers)

Flynn's taxonomy









Source: https://en.wikipedia.org/wiki/Flynn%27s_taxonomy

Instruction-Level Parallelism (ILP)

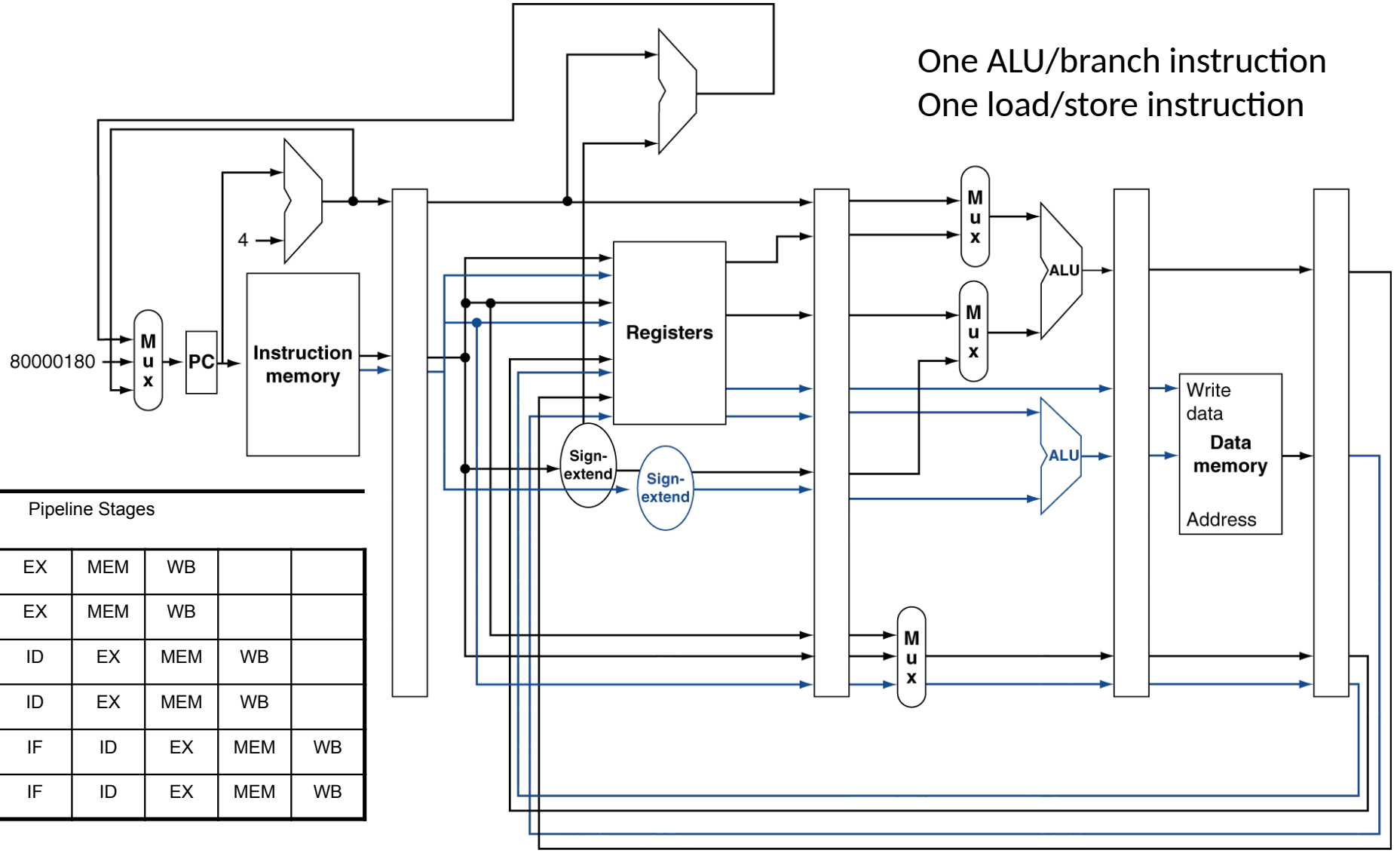
- Pipelining exploits potential parallelism among instructions
  - executing multiple instructions in parallel is known as Instruction-Level Parallelism
- Multiple Issue
  - Replicate pipeline stages ⇒ multiple pipelines
  - Start multiple instructions per clock cycle
  - CPI < 1, so use Instructions Per Cycle (IPC)
  - E.g., 4GHz 4-way multiple-issue
    - 16 BIPS, peak CPI = 0.25, peak IPC = 4
    - But dependencies reduce this in practice

Multiple Issue Approaches

- Static multiple issue [majorly Compiler-based]
  - Compiler groups instructions to be issued together
  - Packages them into “issue slots”
  - Compiler detects and avoids hazards
- Dynamic multiple issue [majorly hardware-based]
  - CPU examines instruction stream and chooses instructions to issue each cycle (one or more)
  - Compiler can help by reordering instructions
  - CPU resolves hazards using advanced techniques at runtime

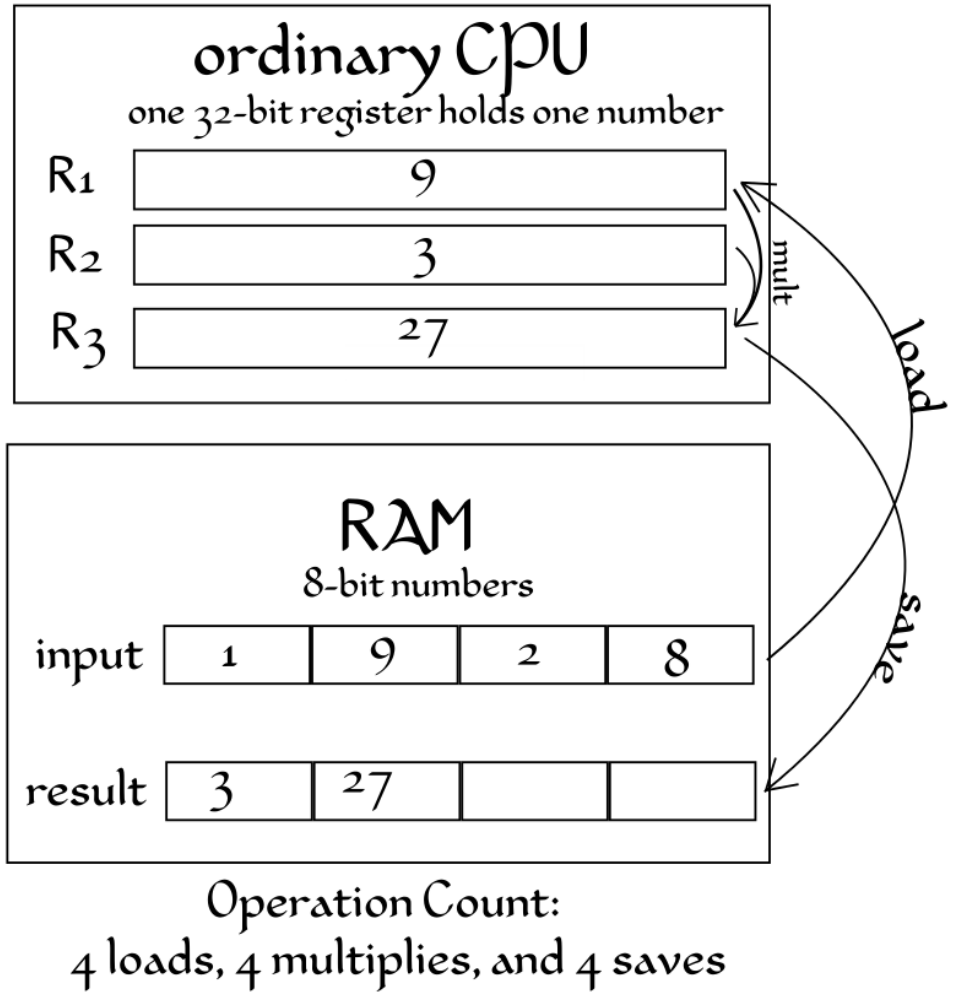
MIPS with Static Dual Issue

One ALU/branch instruction
One load/store instruction



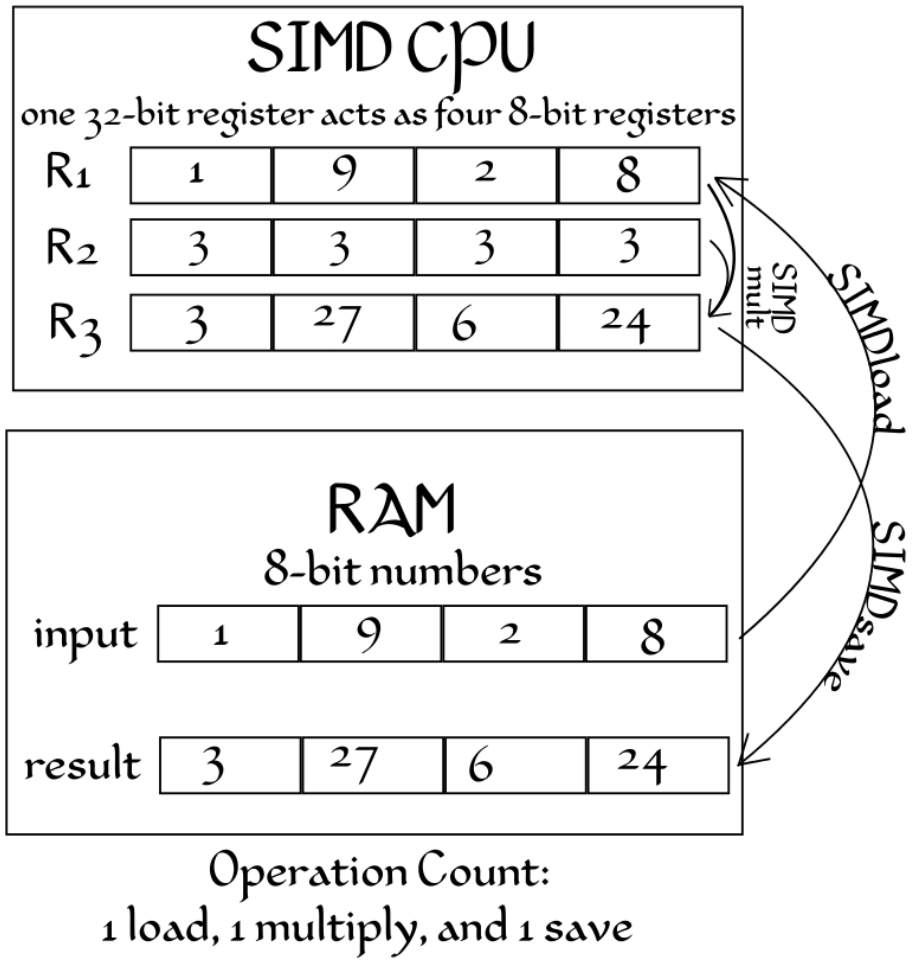
Address	Instruction type	Pipeline Stages						
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ALU/branch		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ALU/branch			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB

SIMD



Operation Count:
4 loads, 4 multiplies, and 4 saves

Traditional Processor

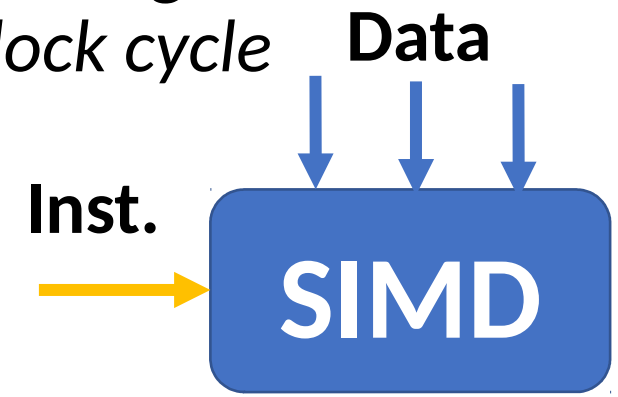


Operation Count:
1 load, 1 multiply, and 1 save

SIMD Processor

SIMD

- In a single program counter, **one instruction** is executed **on multiple data units**
  - *a single SIMD instruction might add 64 numbers by sending 64 data streams to 64 ALUs to form 64 sums within a single clock cycle*
  - Each execution unit has its own address registers



- Advantages
  - One control unit for all execution units
  - Reduced instruction bandwidth (one copy [memory] of code is executed vs multiple for multicore processors)
  - Very efficient with identically structured data [loops][fewer control hazard]
  - **Data-level parallelism:** performing the same operation on independent data
- Inefficient with control logic (if, case,..)

Reading

- Sections 6.1 – 6.3