Parallel Programming

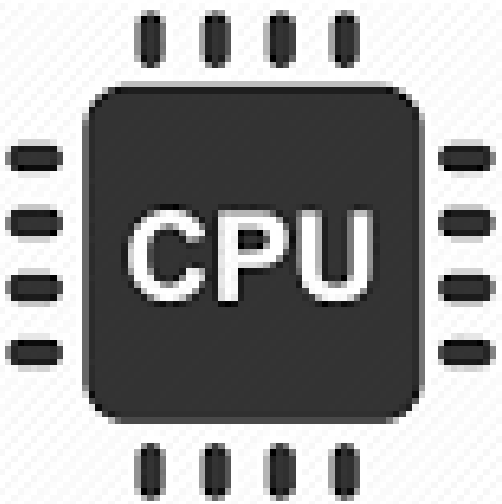
Architectures – Pt.1

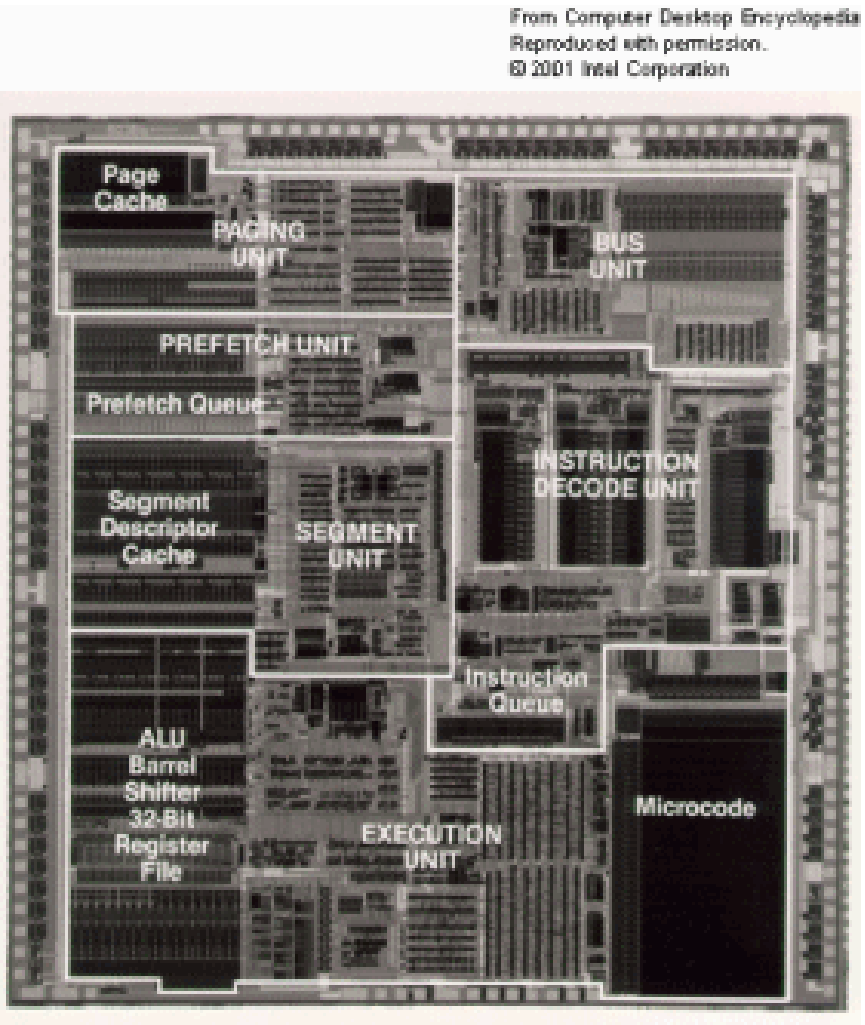
Prof. Paolo Bientinesi

HPAC, RWTH Aachen
pauldj@aices.rwth-aachen.de

WS17/18

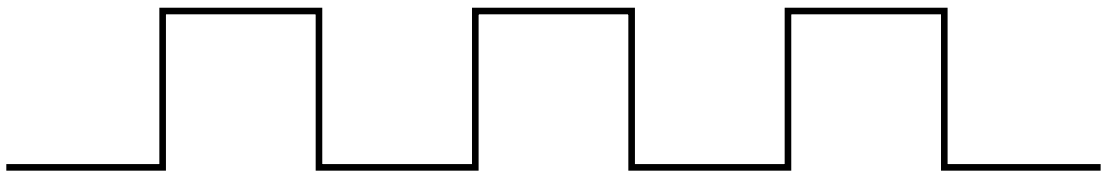
Quick architecture review





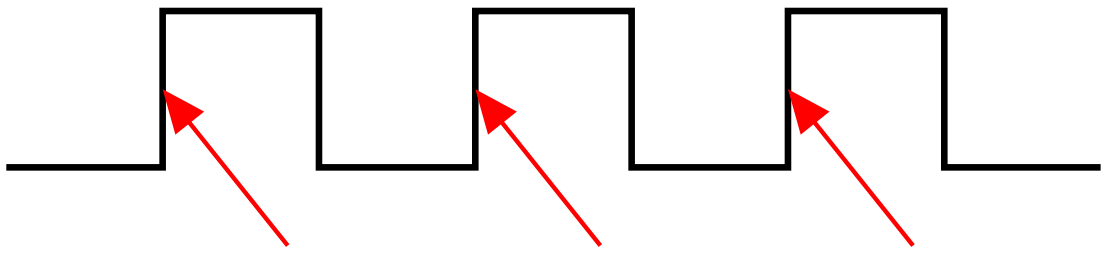
Clock, cycle, frequency

- Clock determines when events take place in the hardware



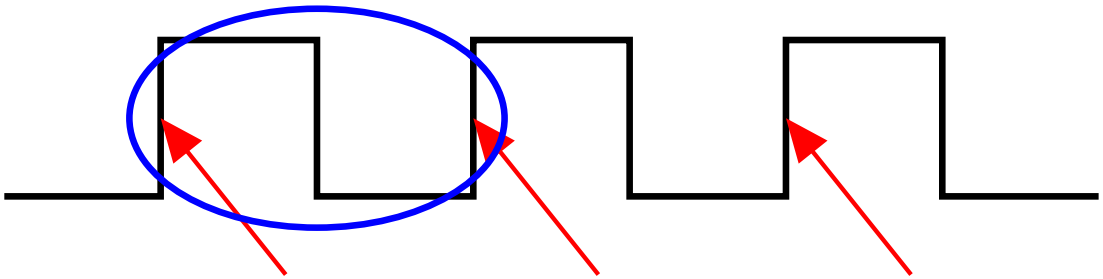
Clock, cycle, frequency

- Clock determines when events take place in the hardware



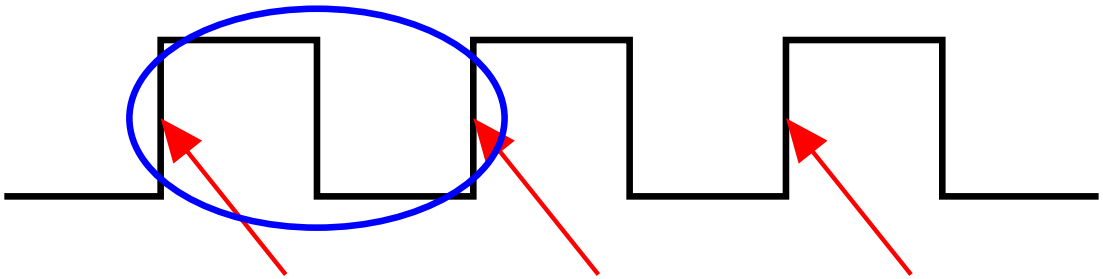
Clock, cycle, frequency

- Clock determines when events take place in the hardware



Clock, cycle, frequency

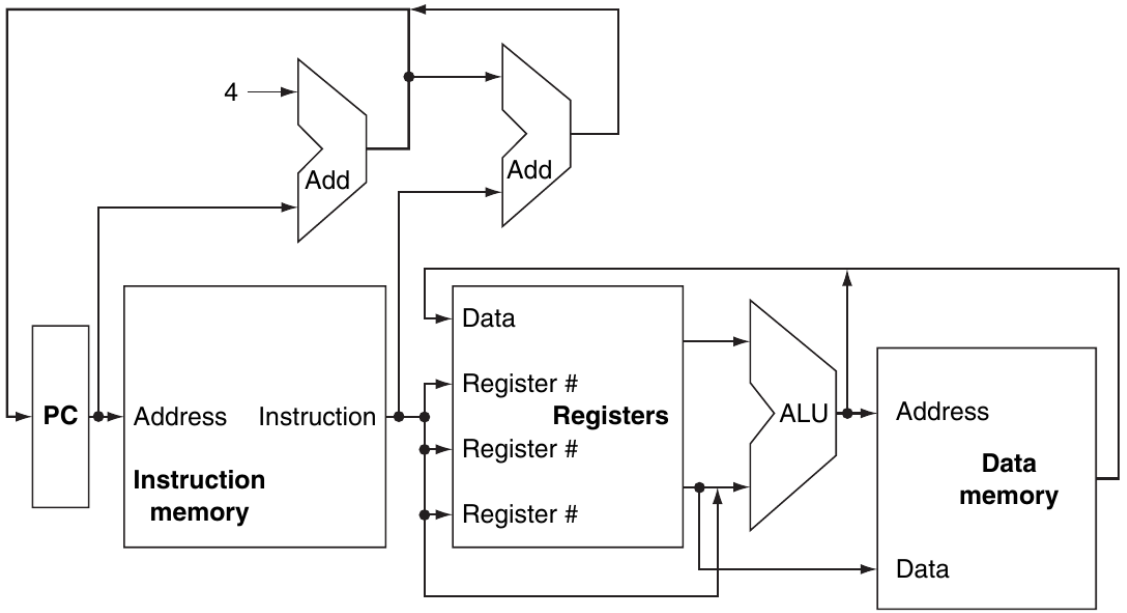
- Clock determines when events take place in the hardware



- Frequency (or clock rate): # of cycles per second.

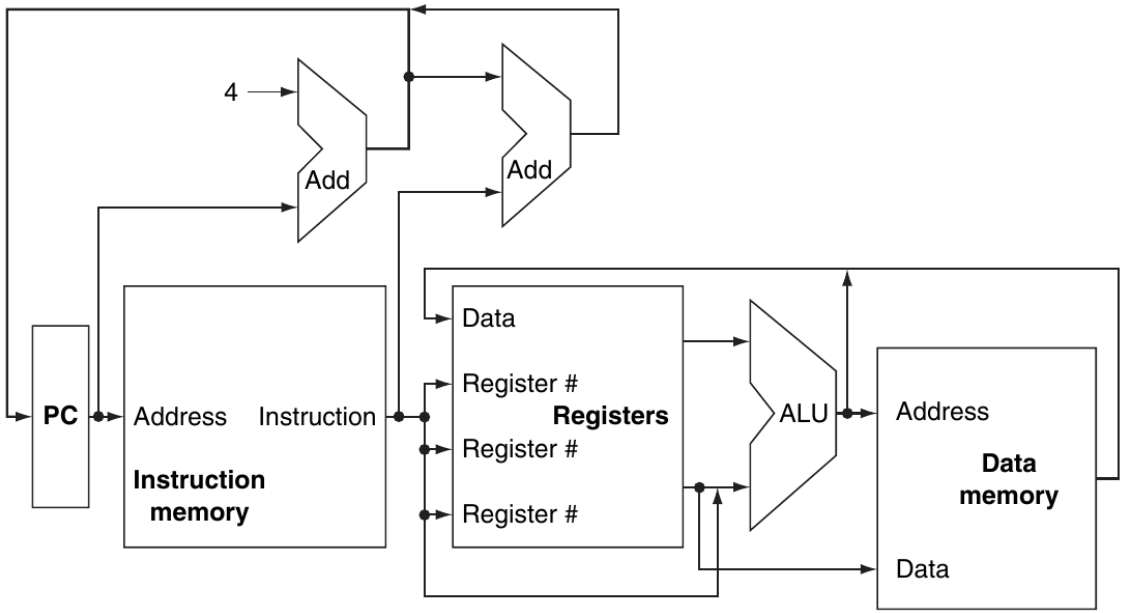
  For instance: 2GHz $\rightarrow 2 \times 10^9$ cycles per second

Basic processor architecture



Source: *Computer organization and design.* Patterson, Hennessy.

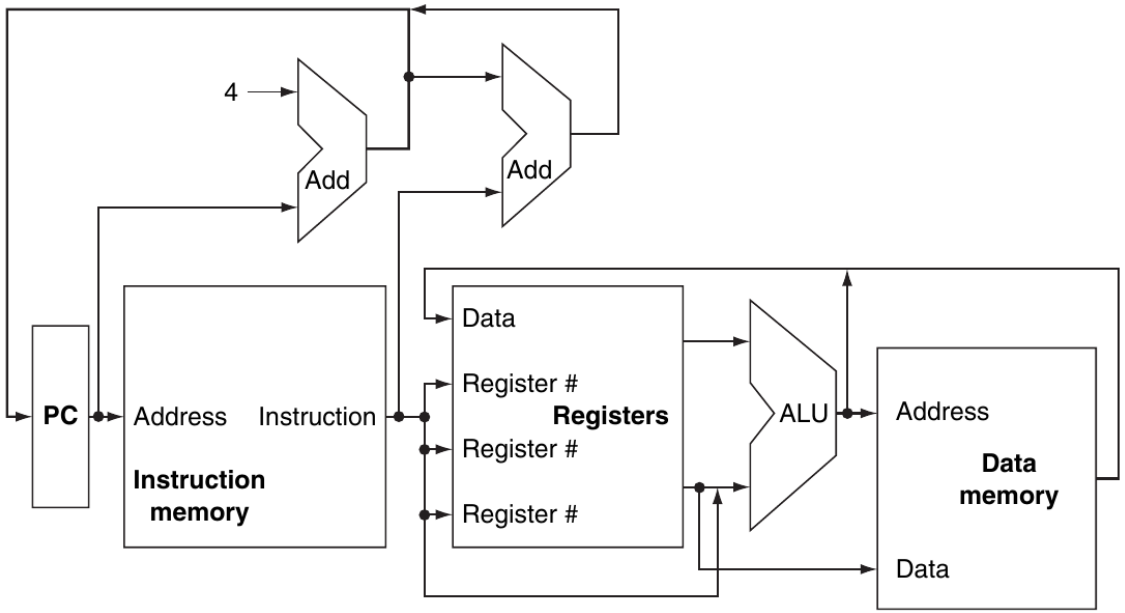
Basic processor architecture



Source: *Computer organization and design.* Patterson, Hennessy.

- Instruction Fetch (IF): read instruction from cache

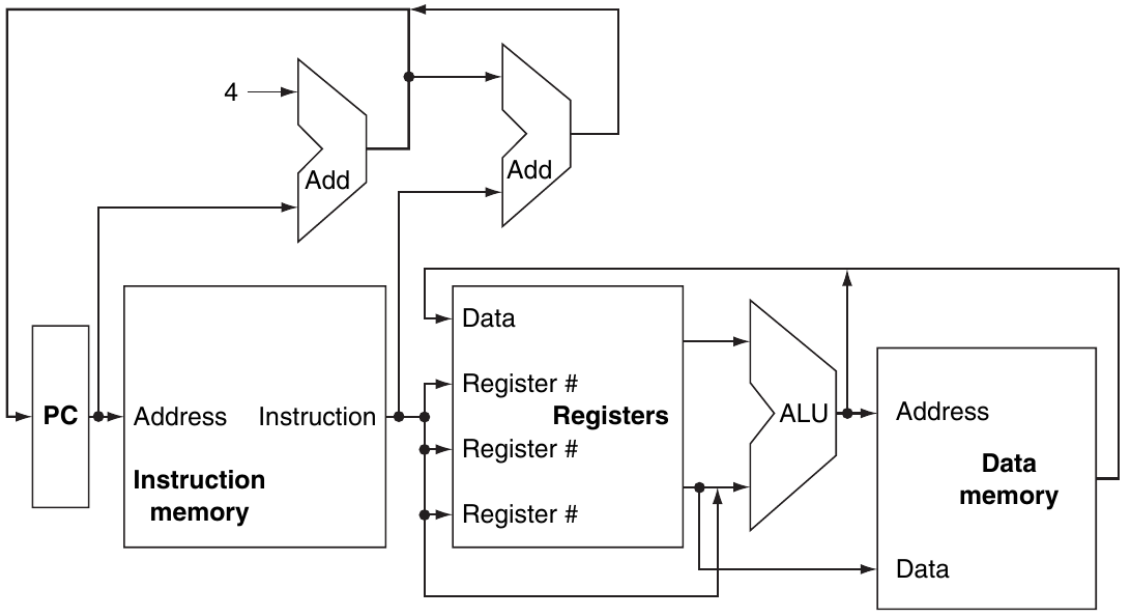
Basic processor architecture



Source: *Computer organization and design.* Patterson, Hennessy.

- Instruction Fetch (IF): read instruction from cache
- Instruction Decode (ID): read register data

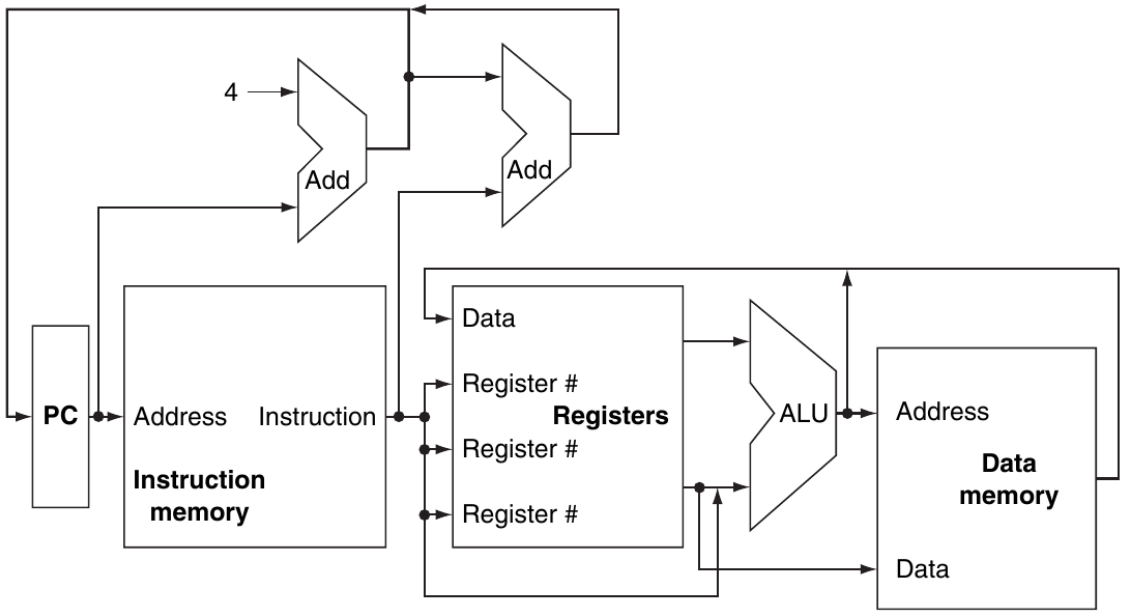
Basic processor architecture



Source: *Computer organization and design.* Patterson, Hennessy.

- Instruction Fetch (IF): read instruction from cache
- Instruction Decode (ID): read register data
- Execute (EX): execute arithmetic/logic operation

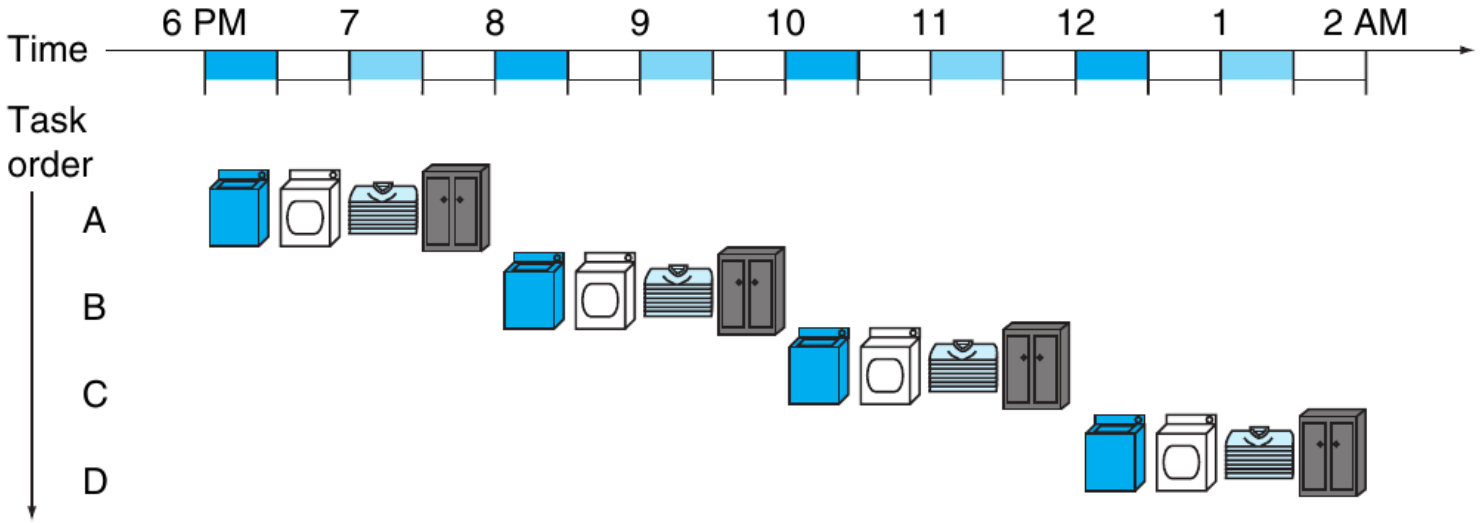
Basic processor architecture



Source: *Computer organization and design.* Patterson, Hennessy.

- Instruction Fetch (IF): read instruction from cache
- Instruction Decode (ID): read register data
- Execute (EX): execute arithmetic/logic operation
- Store (ST): store the result

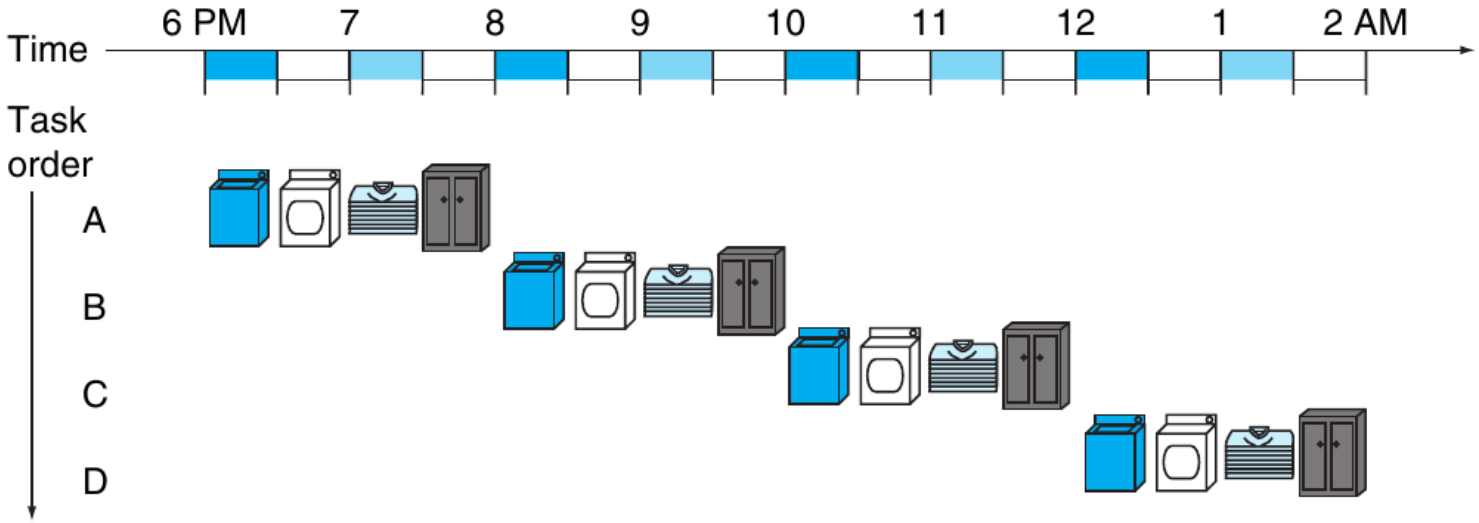
The laundry analogy



Source: *Computer organization and design.* Patterson, Hennessy.

- Latency: 1 load takes 2 hours

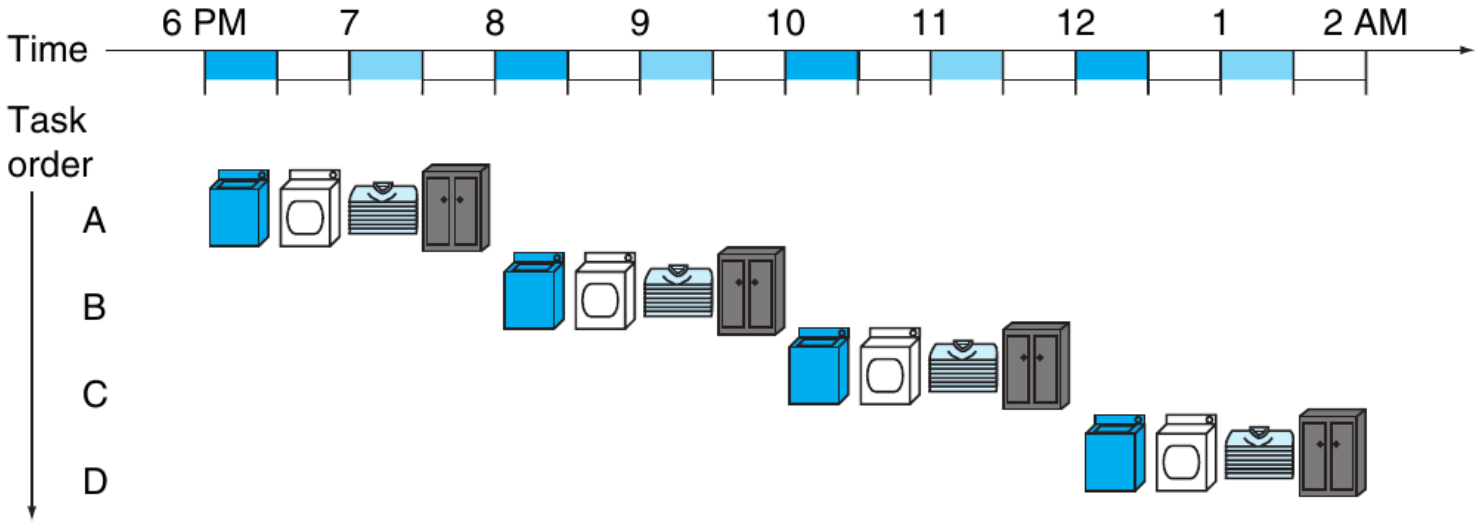
The laundry analogy



Source: *Computer organization and design.* Patterson, Hennessy.

- Latency: 1 load takes 2 hours
- Throughput: n loads take $2 * n$ hours $\Rightarrow \frac{1}{2}$ load per hour

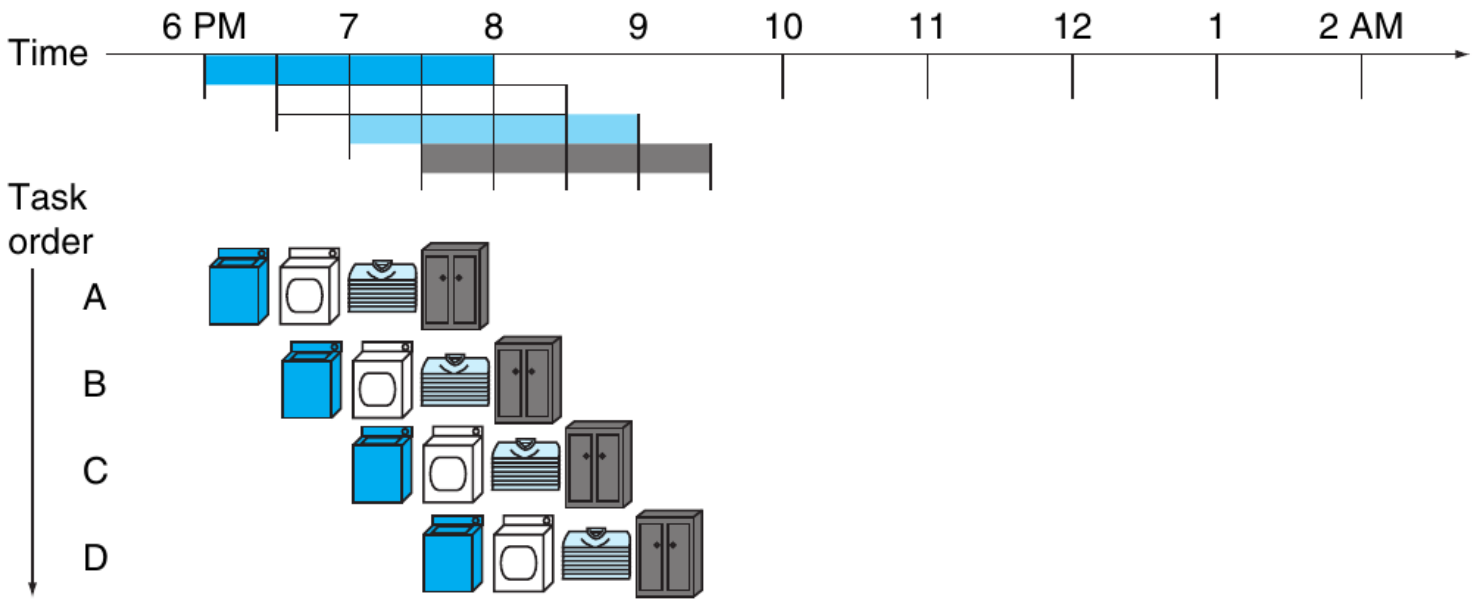
The laundry analogy



Source: *Computer organization and design.* Patterson, Hennessy.

- Latency: 1 load takes 2 hours
- Throughput: n loads take $2 * n$ hours $\Rightarrow \frac{1}{2}$ load per hour
- How can we improve the throughput? **Pipelining**

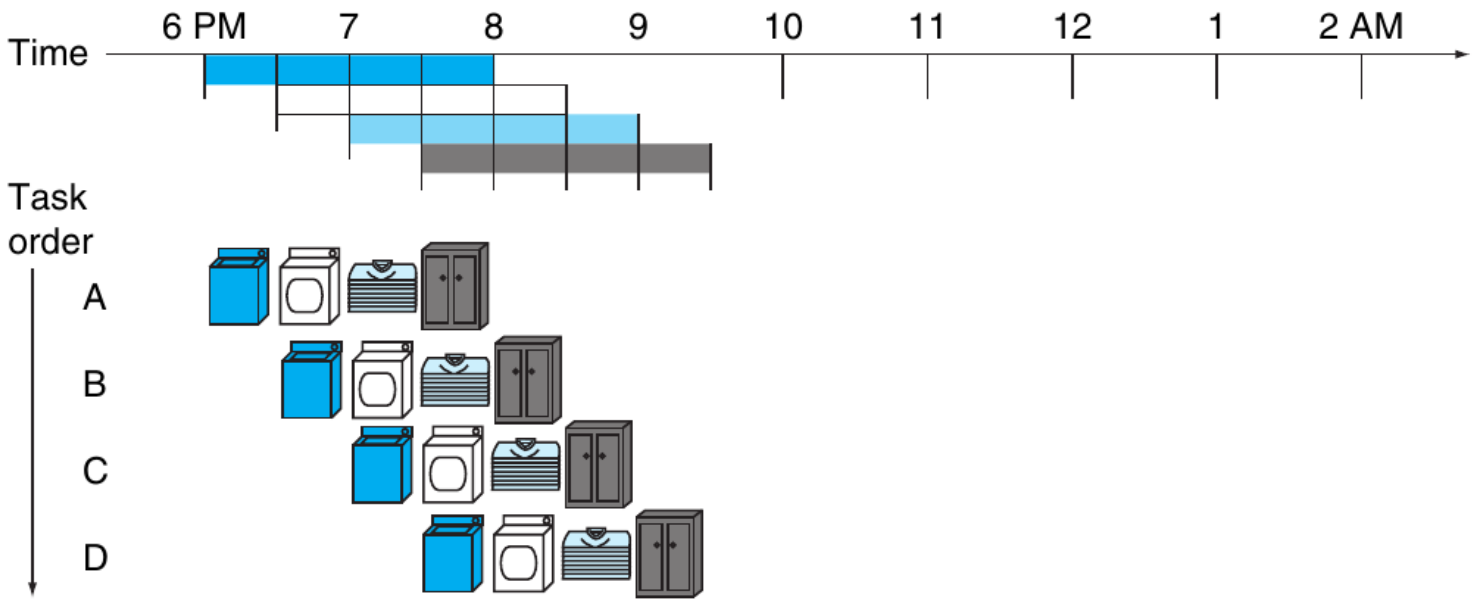
Pipelining



Source: *Computer organization and design.* Patterson, Hennessy.

- Latency: still 2 hours

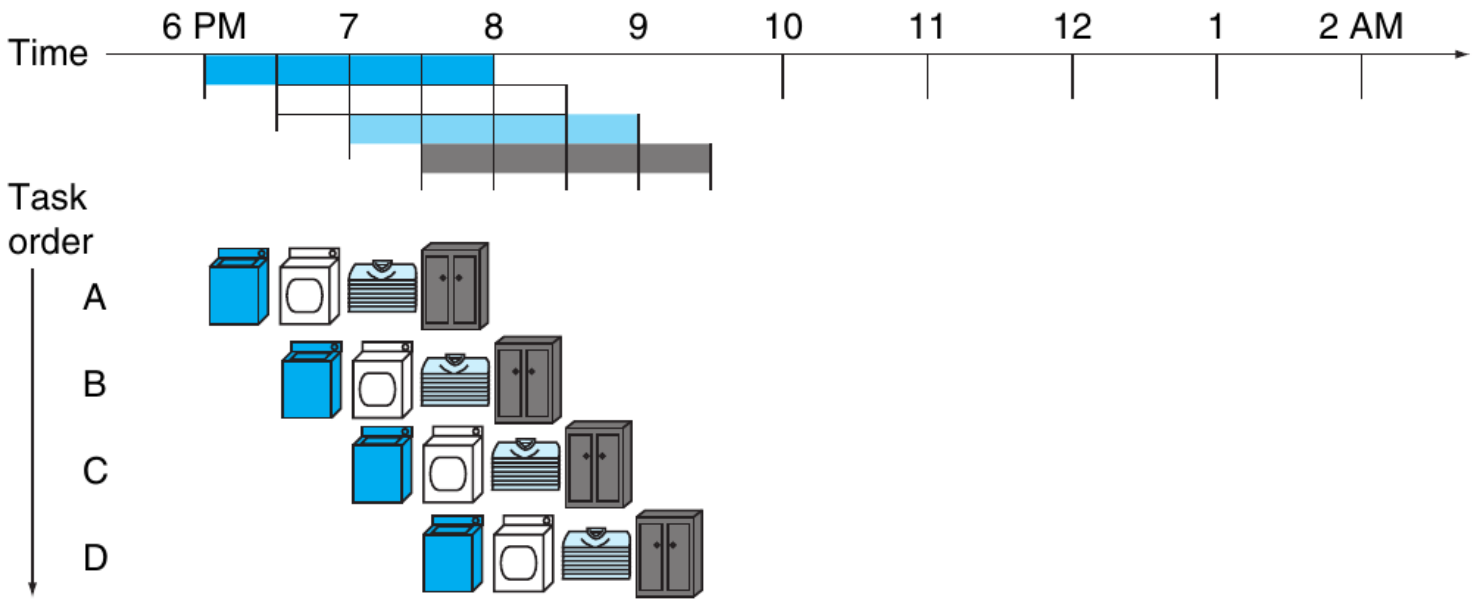
Pipelining



Source: *Computer organization and design.* Patterson, Hennessy.

- Latency: still 2 hours
- Throughput: $(\frac{n}{2+(n-1)*0.5})$ loads per hour ($\frac{4}{3.5} \approx 1.14$ loads/hour)

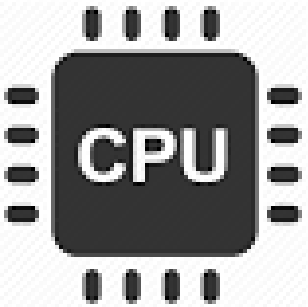
Pipelining

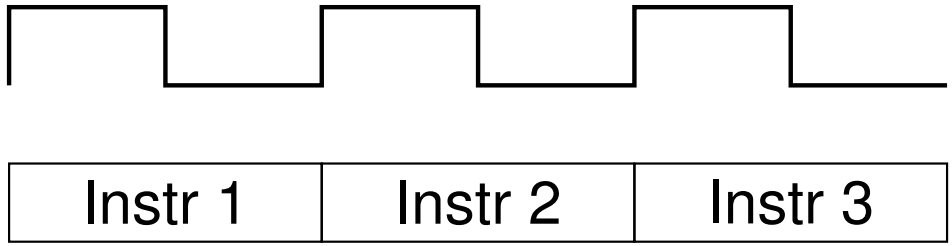


Source: *Computer organization and design.* Patterson, Hennessy.

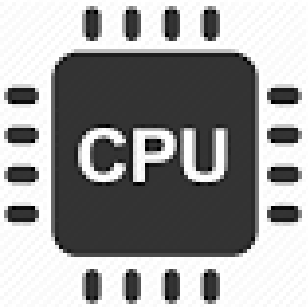
- Latency: still 2 hours
- Throughput: $(\frac{n}{2+(n-1)*0.5})$ loads per hour $(\frac{4}{3.5} \approx 1.14$ loads/hour)
- $\lim_{n\to\infty} \mathit{Throughput} = 2$ (vs original $\frac{1}{2}$)

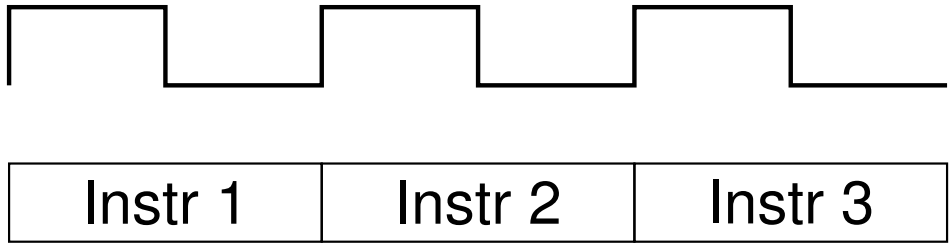
Pipelined processors

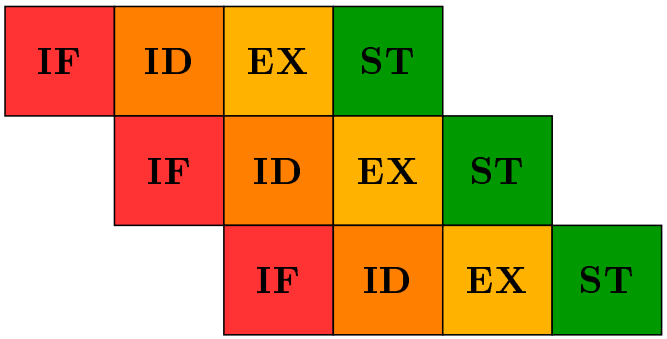




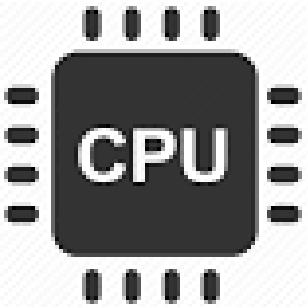
Pipelined processors

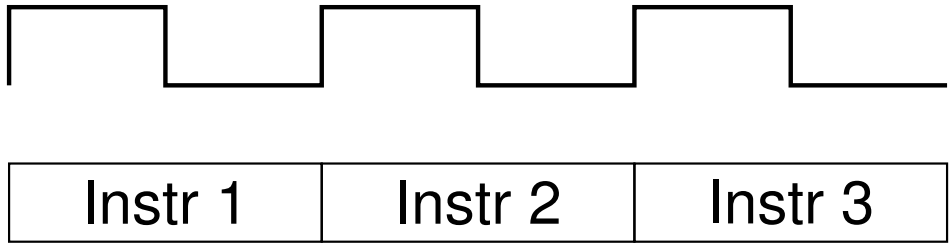


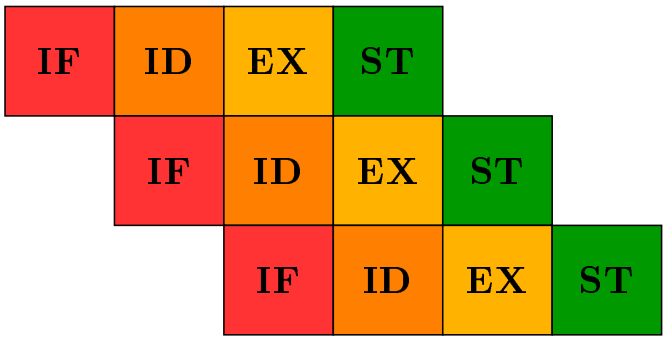




Pipelined processors







- Each step is known as *stage*
- 4-stage pipeline

Throughput

- Program $\mathcal{P}$: n instructions
- s-stage pipeline
- latency(stage) = k secs

Single resource
$\Rightarrow$ **serial execution**

Latency($\mathcal{P}$) = nsk secs

Throughput:
$\frac{n}{nsk} = \frac{1}{sk}$ instr/sec

Throughput

- Program $\mathcal{P}$: n instructions
- s-stage pipeline
- latency(stage) = k secs

Single resource

$\Rightarrow$ **serial execution**

Latency($\mathcal{P}$) = nsk secs

Throughput:

$\frac{n}{nsk} = \frac{1}{sk}$ instr/sec

Multiple resources

$\Rightarrow$ **pipelined execution**

Latency($\mathcal{P}$) = $sk + (n-1)k$ secs

Throughput:

$\lim_{n\to\infty} \frac{n}{sk+(n-1)k} = \frac{1}{k}$ instr/sec

Throughput

- Program $\mathcal{P}$: n instructions
- s-stage pipeline
- latency(stage) = k secs

Single resource
$\Rightarrow$ **serial execution**

Latency($\mathcal{P}$) = nsk secs

Throughput:
$\frac{n}{nsk} = \frac{1}{sk}$ instr/sec

Multiple resources
$\Rightarrow$ **pipelined execution**

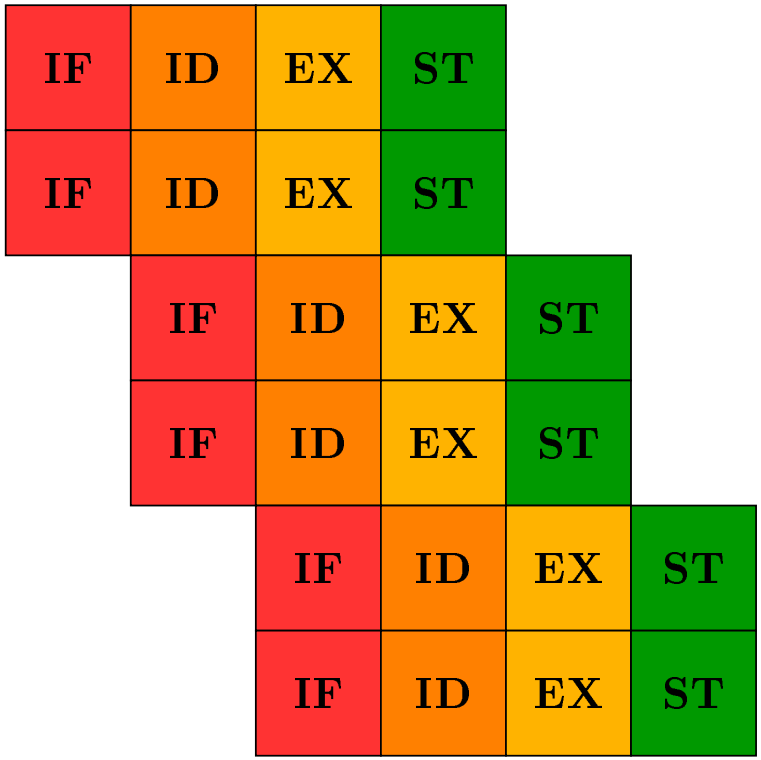
Latency($\mathcal{P}$) = $sk + (n-1)k$ secs

Throughput:
$\lim_{n\to\infty} \frac{n}{sk+(n-1)k} = \frac{1}{k}$ instr/sec

Morale: The throughput for the pipelined execution is s time as large as the one for the serial execution. Also, the more stages (the larger s), the smaller k, and the higher the throughput.

Multiple-issue processors

- Replicate internal components to launch multiple instructions per cycle
- Allows instruction execution rate $>$ clock rate
- That is, allows to complete the execution of more than one IPC



Towards the multi-core era

Limitations in ILP

Trends in multiple-issue processors.

	486	Pentium	Pentium II	Pentium 4	Itanium	Itanium 2	Core2
Year	1989	1993	1998	2001	2002	2004	2006
Width	1	2	3	3	3	6	4

Towards the multi-core era

Limitations in ILP

Trends in multiple-issue processors.

	486	Pentium	Pentium II	Pentium 4	Itanium	Itanium 2	Core2
Year	1989	1993	1998	2001	2002	2004	2006
Width	1	2	3	3	3	6	4

- High-performance processors:
  - Issue width has stabilized at 4-6
  - Alpha 21464 (8-way) was canceled (2001).
  - Need hardware/compiler scheduling to exploit the width
- Embedded/Low-power processors:
  - Typical width of 2
  - Simpler architectures, no advanced scheduling

Towards the multi-core era

Limitations in ILP

Microarchitecture	Pipeline stages
i486	3
P5 (Pentium)	5
P6 (Pentium Pro/II)	14
P6 (Pentium 3)	8
P6 (Pentium M)	10
NetBurst (Northwood)	20
NetBurst (Prescott)	31
Core	12
Nehalem	20
Sandy Bridge	14
Haswell	14

Table: **Evolution of the pipeline depth for a sample of Intel microarchitectures.**
Source: wikipedia.org