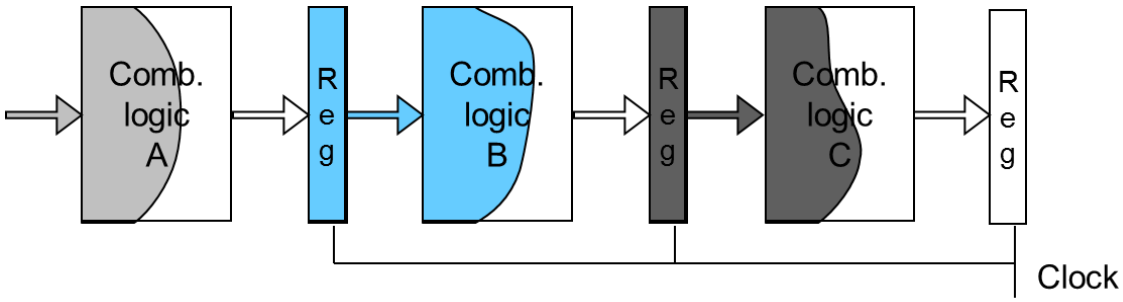
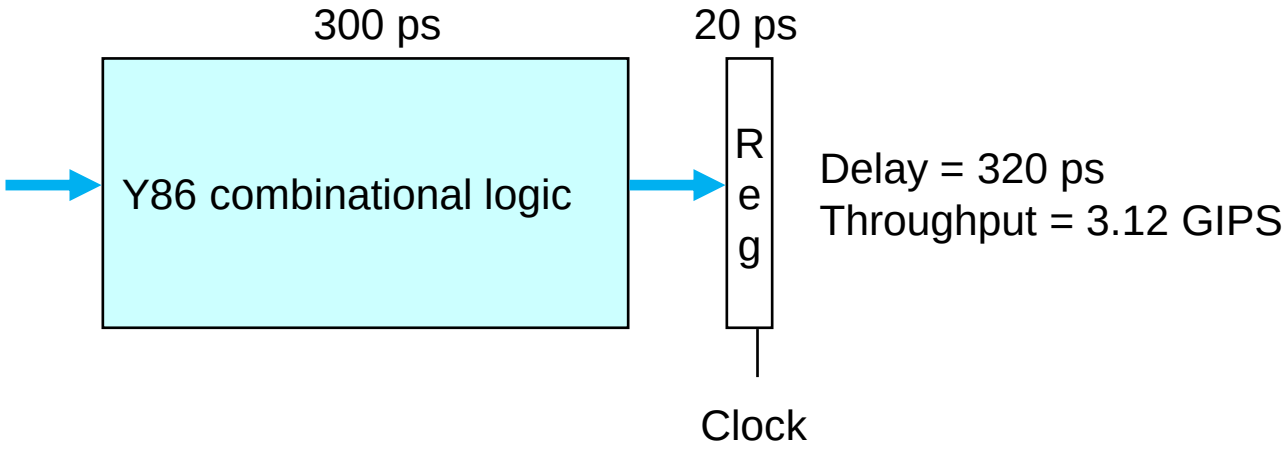
Processor Architecture

Principles of Pipelining



Limitations of Sequential Circuits



- Sequential Implementation
  - Computation requires total of 300 picoseconds
  - Additional 20 picoseconds to save result in register
  - Must have clock cycle of at least 320 ps

Sequential vs. Parallel vs. Pipelining

- Car Wash – Sequential

Sequential vs. Parallel vs. Pipelining

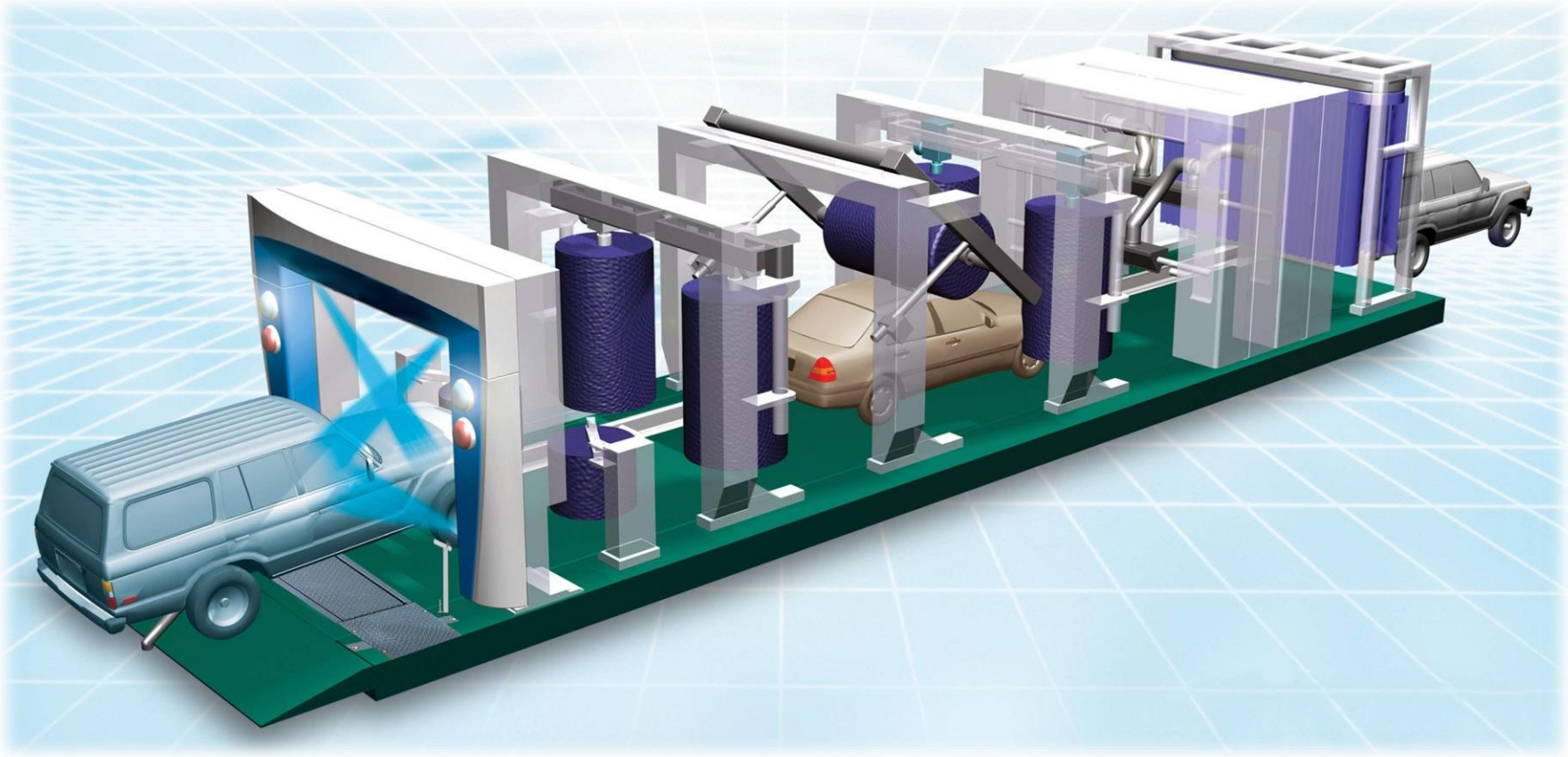
- Car Wash – Parallel (Coarse-grained)

Sequential vs. Parallel vs. Pipelining

- Car Wash – Parallel (Fine-grained)

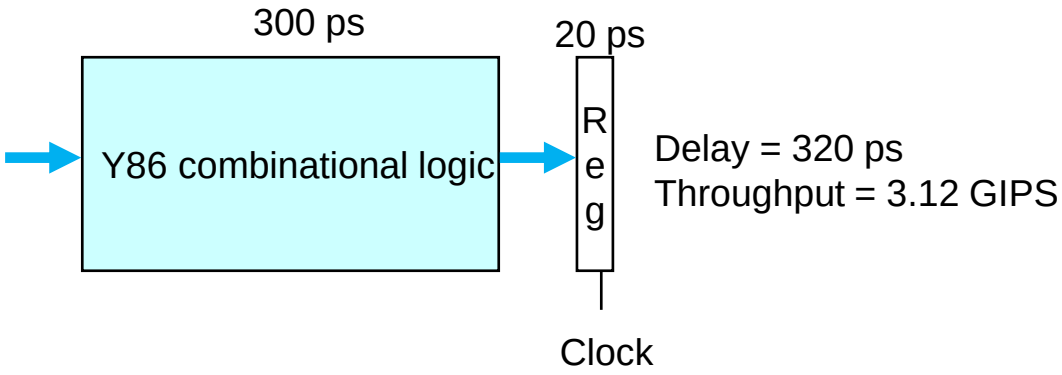
Sequential vs. Parallel vs. Pipelining

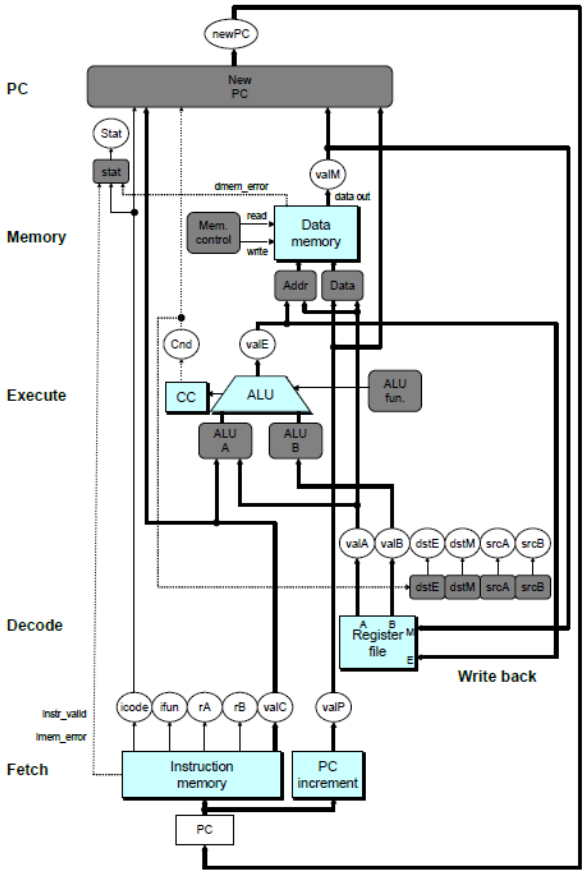
- Car Wash – Pipelined

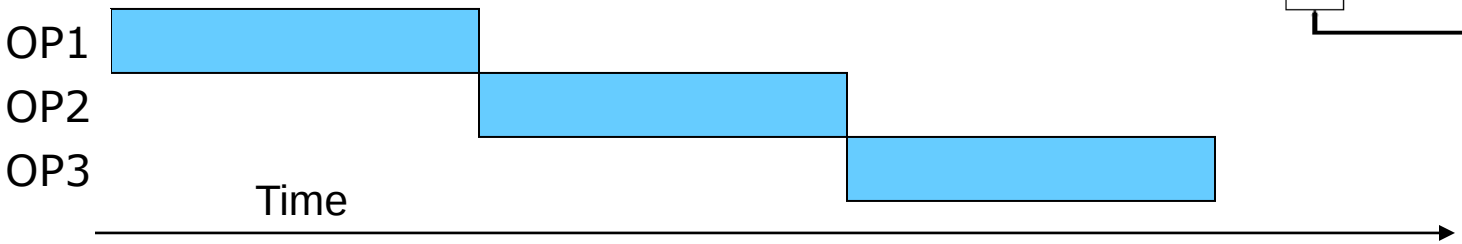


Sequential Architectures

- Sequential Logic in Parallel

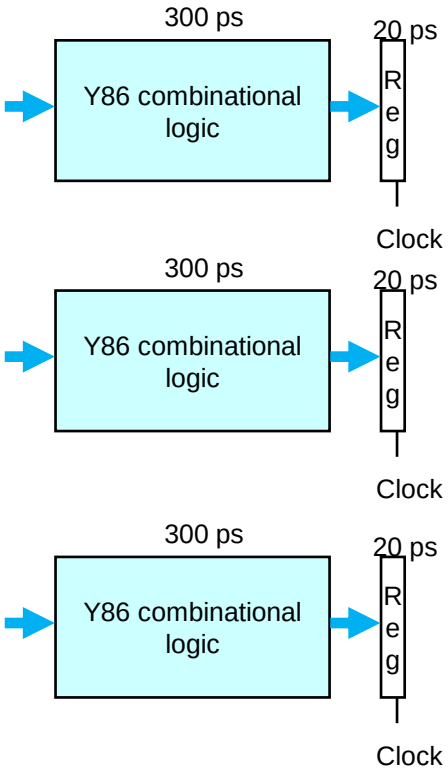




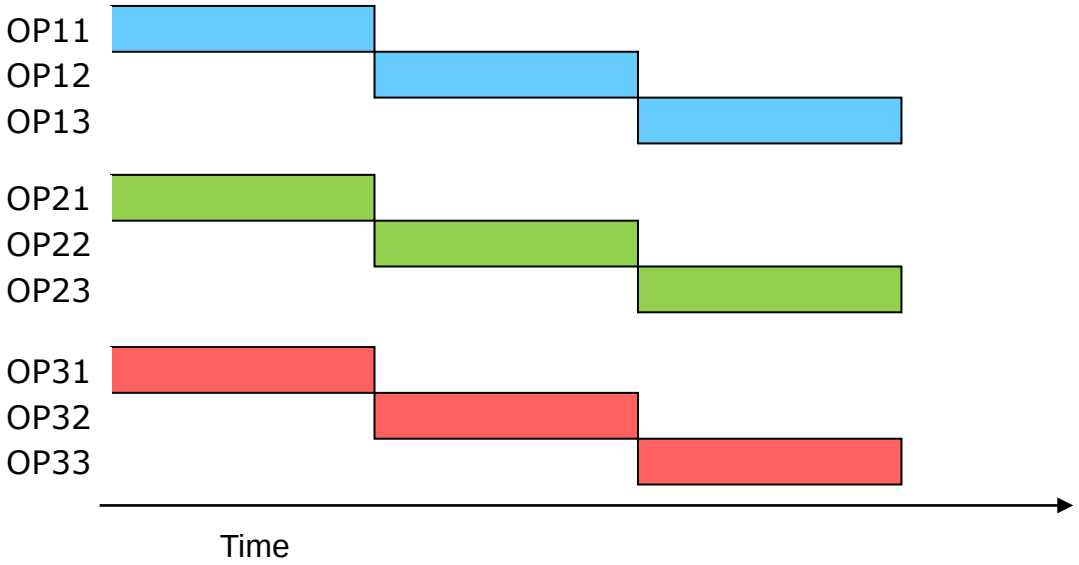


Parallel Architectures

- Coarse-grained Parallelism – Multiprocessor, Multicore

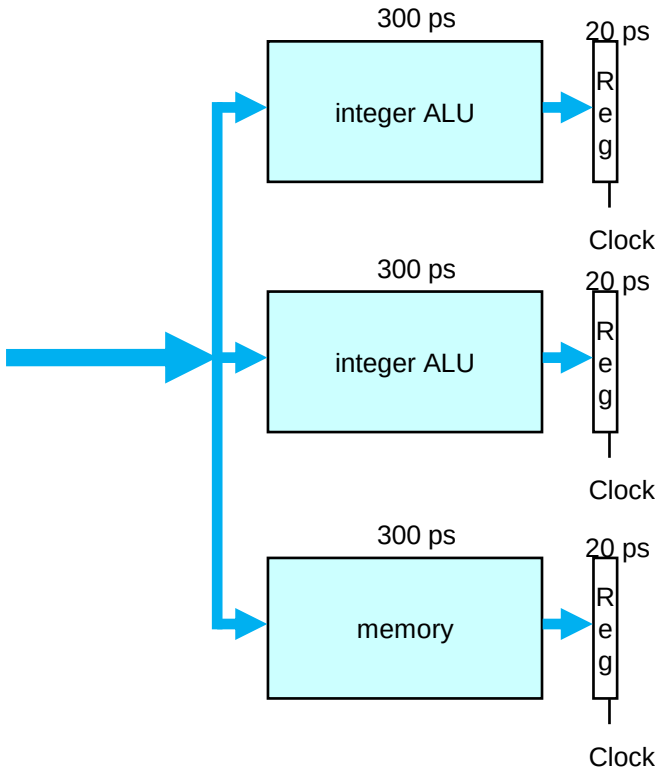


Delay = 320 ps
Throughput = 3 * 3.12 GIPS
= 9.36 GIPS

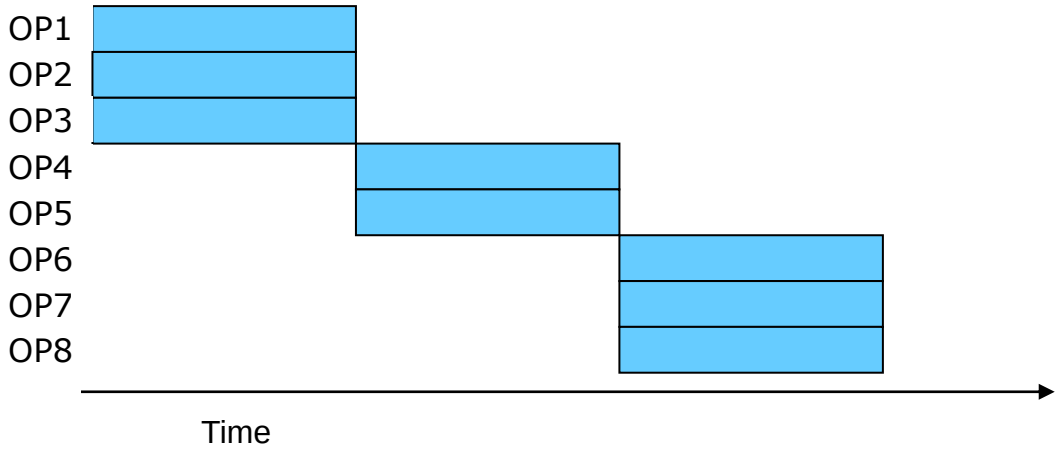


Parallel Architectures

- Fine-grained Parallelism – Superscalar, VLIW

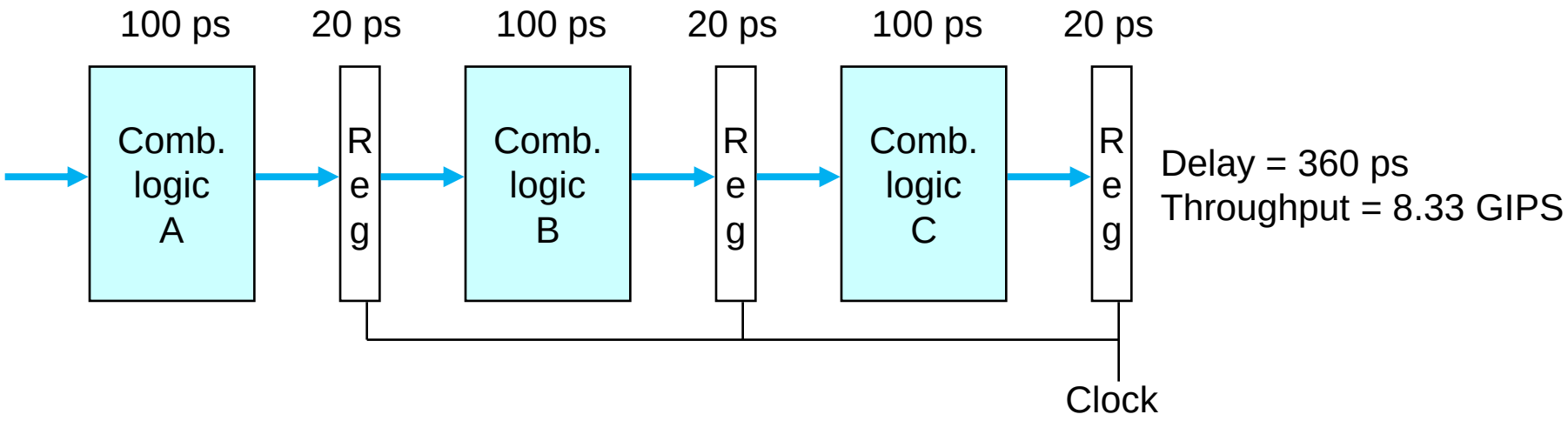


Delay = 320 ps
Throughput <= 3 * 3.12 GIPS
<= 9.36 GIPS



Pipelined Architectures

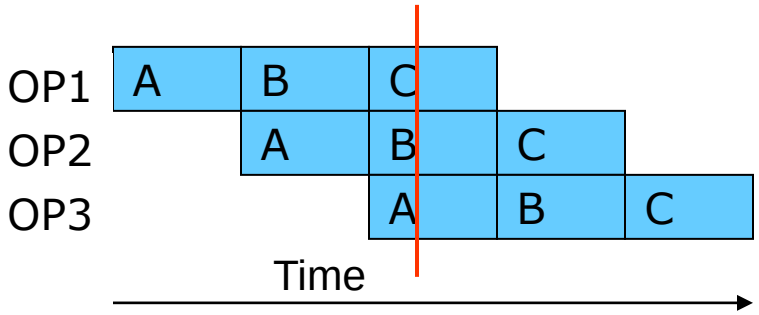
- Pipelining – (almost) all modern processors
  - idea:
    - split work up into several independent phases
    - run independent phases in parallel



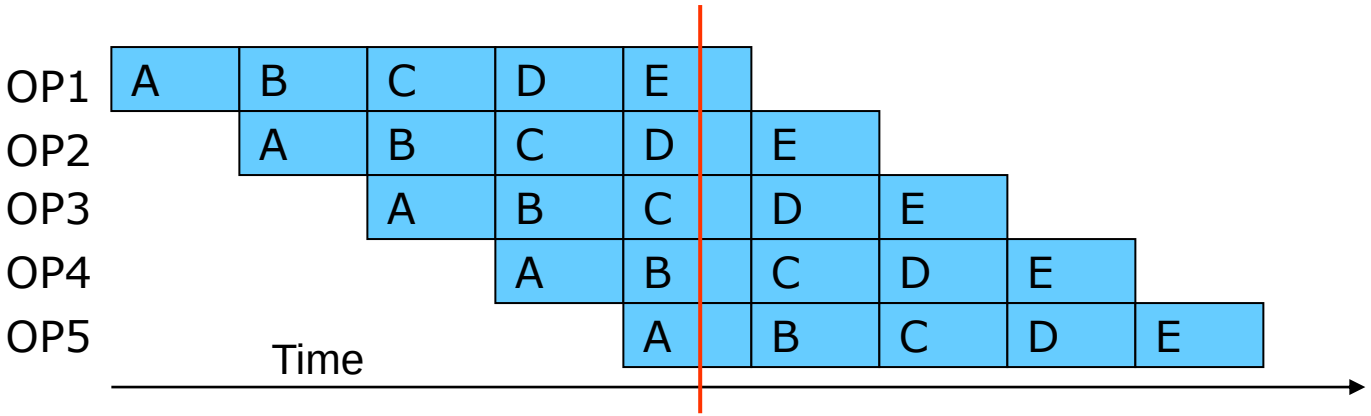
  - can begin new operation as soon as previous has passed through stage A
  - overall latency (for one operation) *increases*

Pipeline Stages

- 3-way pipelined

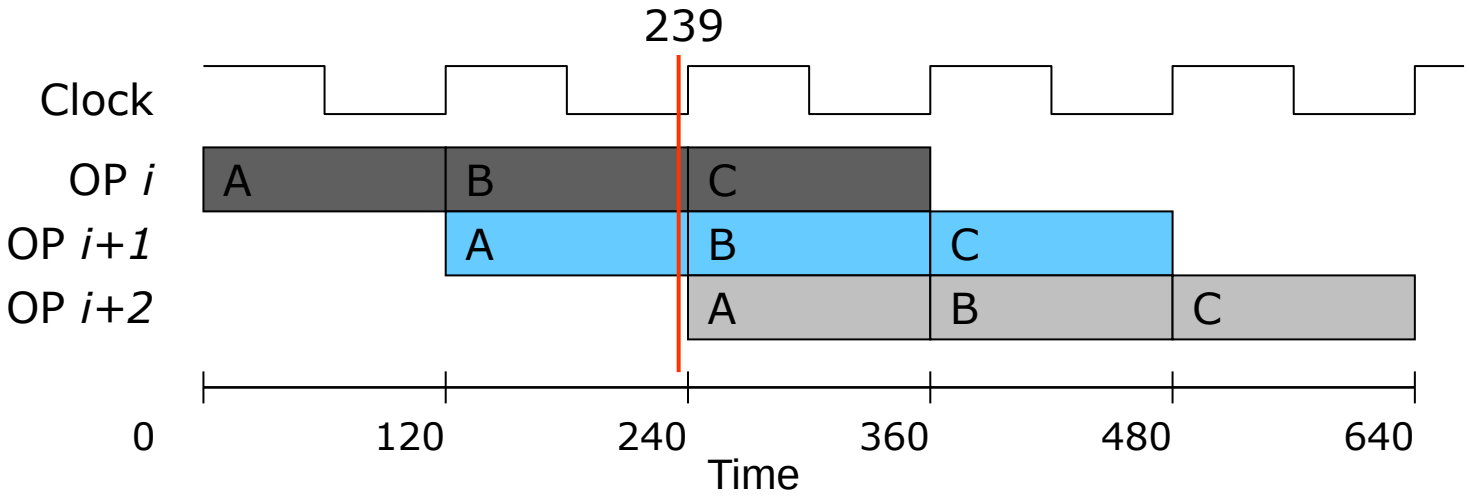


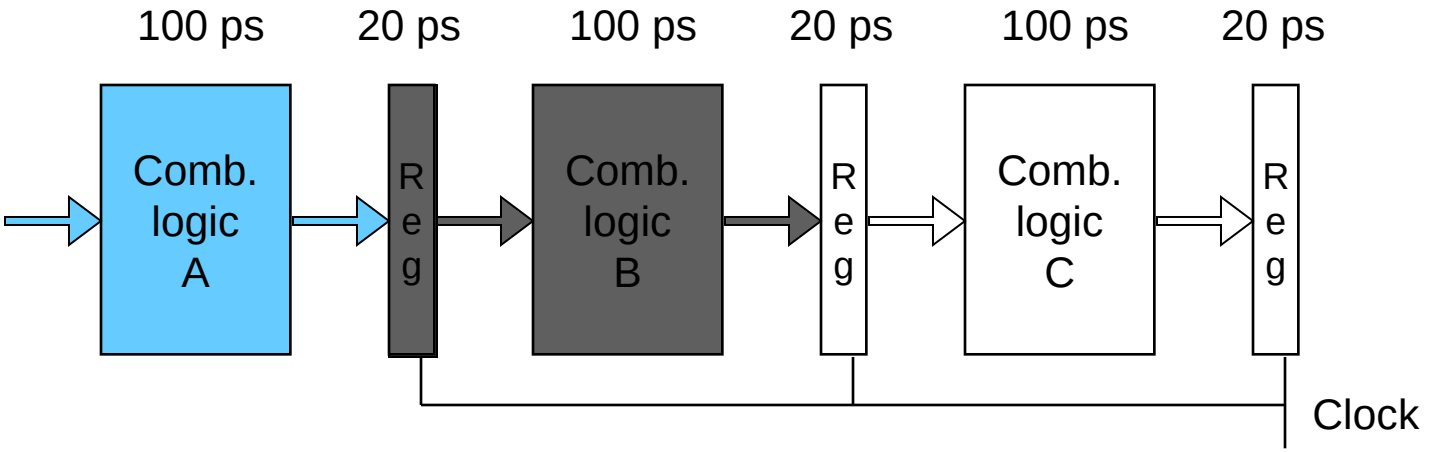
- 5-way pipelined



Pipeline Operation

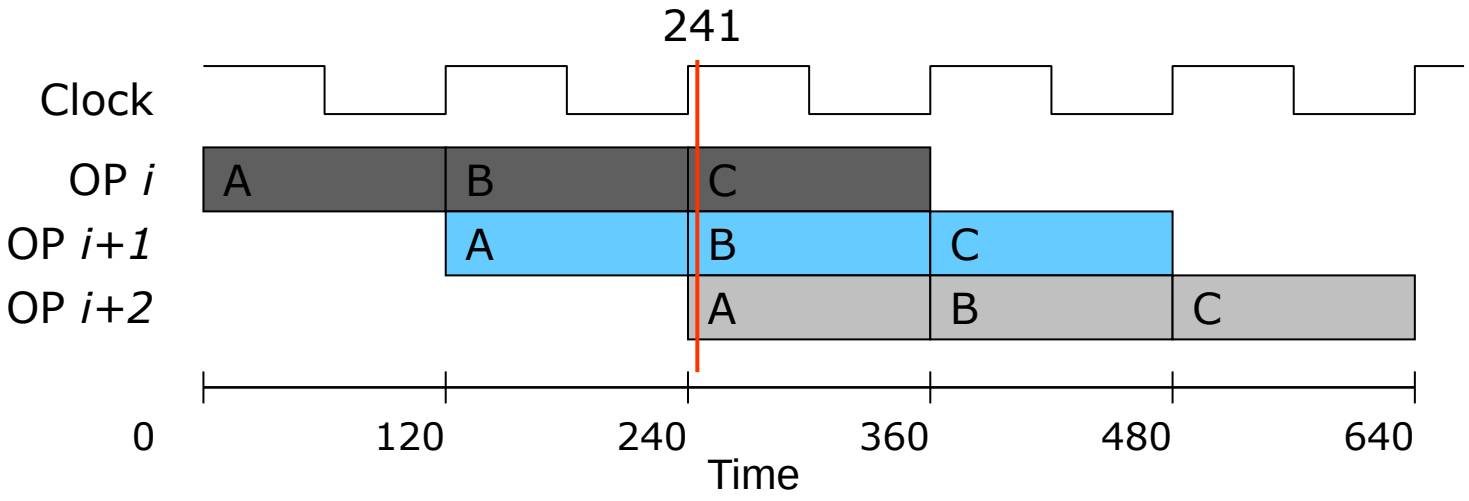
- Immediately before a new clock cycle starts

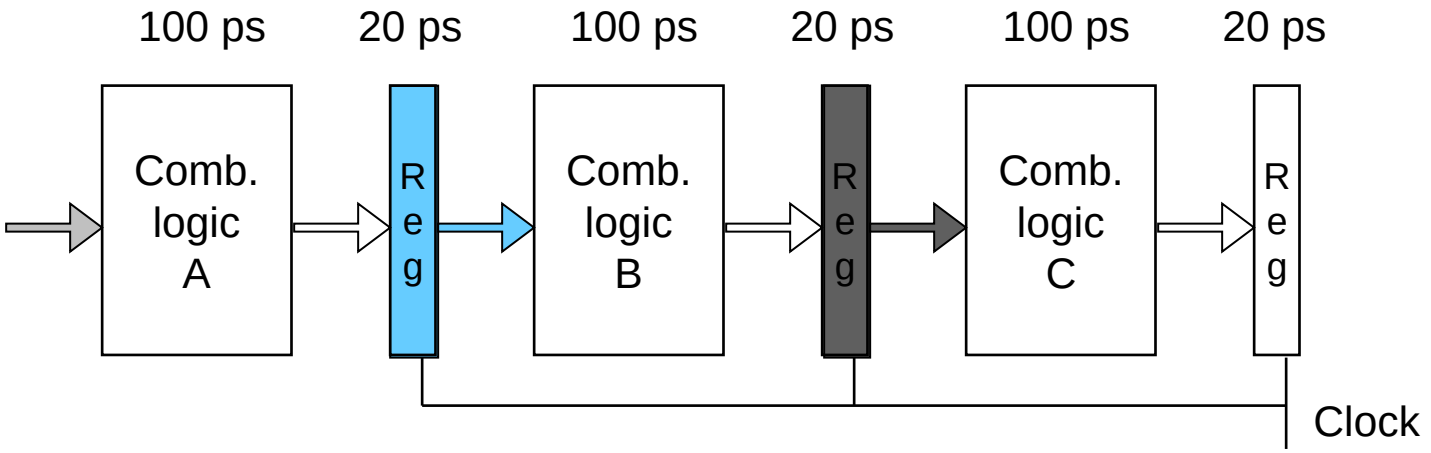




Pipeline Operation

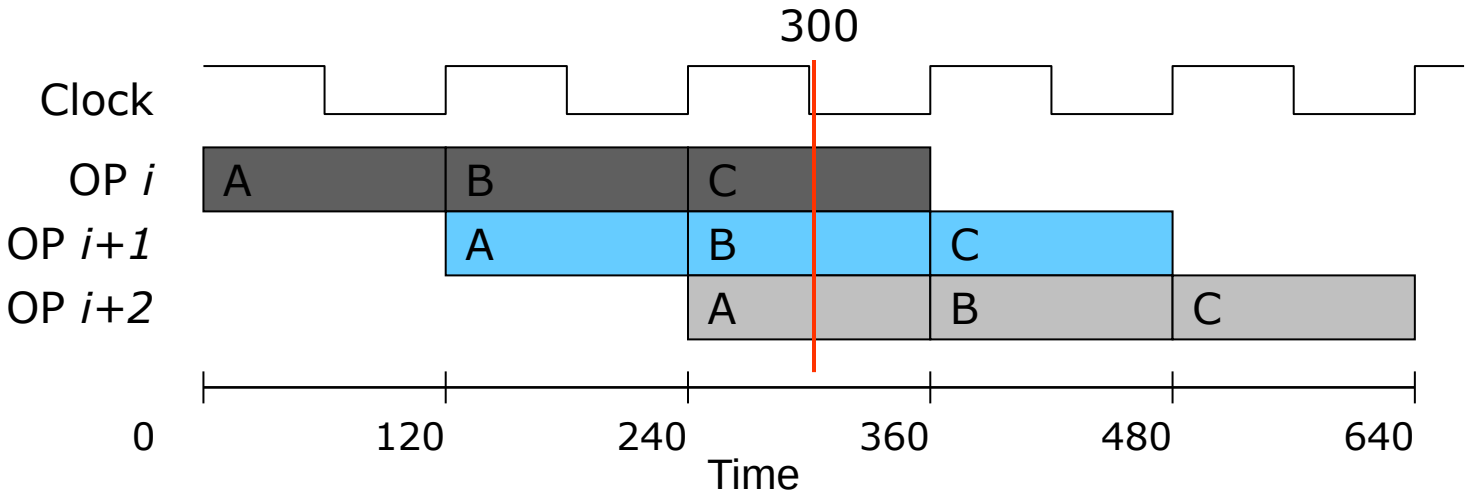
- Immediately after a new clock cycle has started

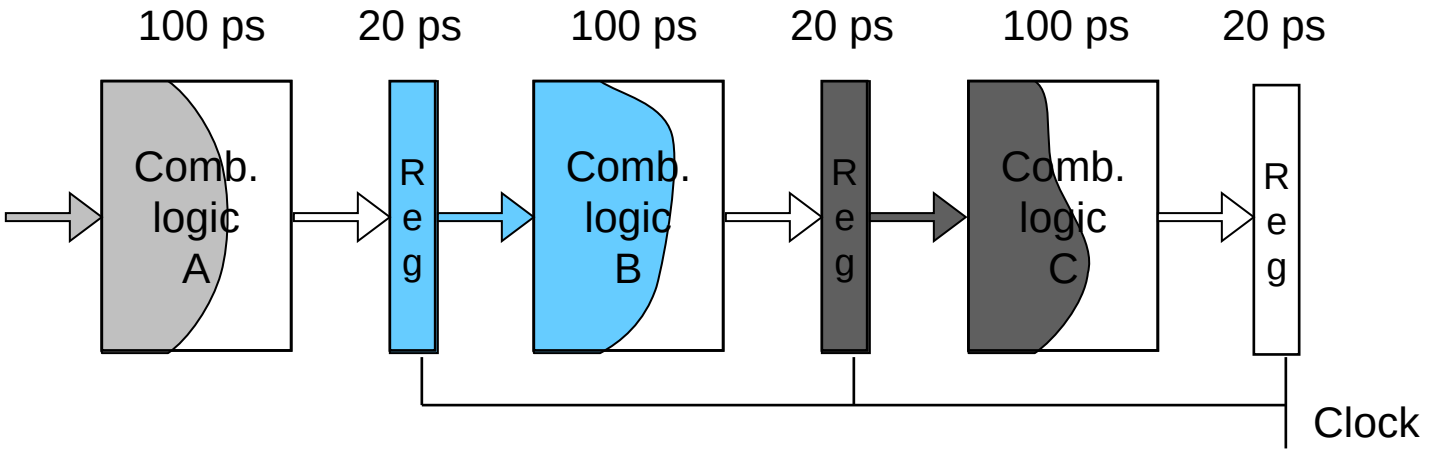




Pipeline Operation

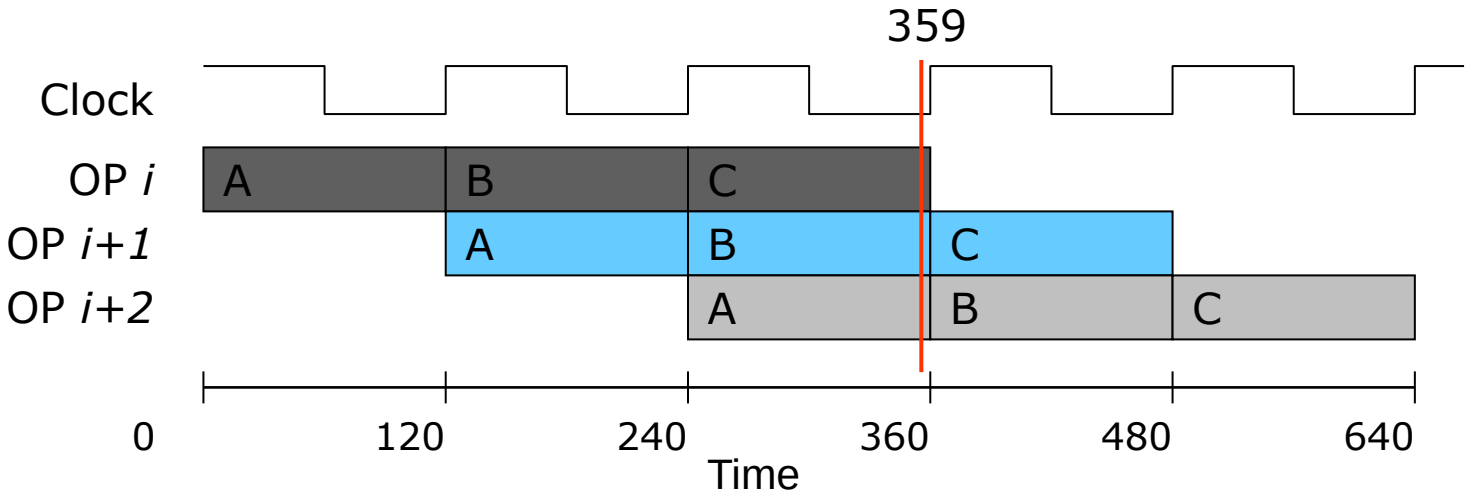
- In the middle of a clock cycle

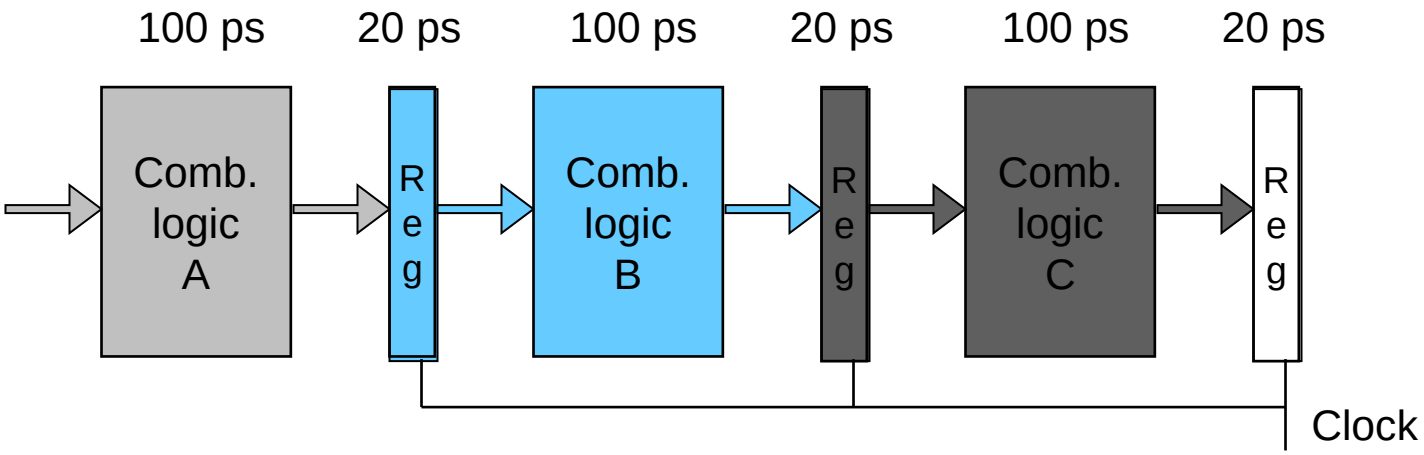




Pipeline Operation

- Immediately before a clock cycle ends





Basic Pipeline

- Pipeline Stages
  1. Instruction fetch step **(F)**
  2. Instruction decode/register fetch step **(D)**
  3. Execution/effective address step **(E)**
  4. Memory access **(M)**
  5. Register write-back step **(W)**

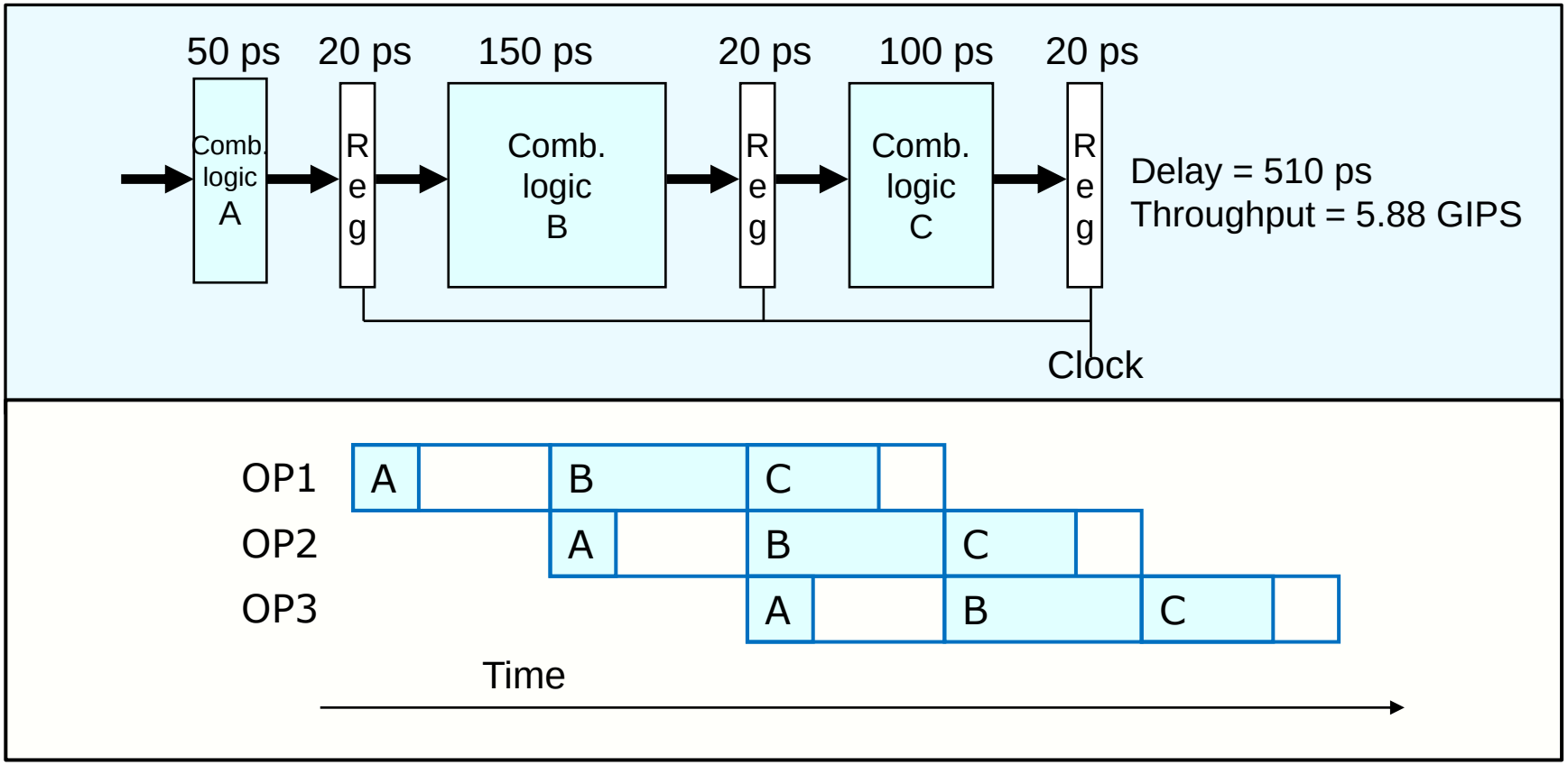
- Pipeline Operation

	Clock number								
Instruction number	1	2	3	4	5	6	7	8	9
Instruction *i*	F	D	E	M	W				
Instruction *i* + 1		F	D	E	M	W			
Instruction *i* + 2			F	D	E	M	W		
Instruction *i* + 3				F	D	E	M	W	
Instruction *i* + 4					F	D	E	M	W

Limitations and Major Hurdles of Pipelining

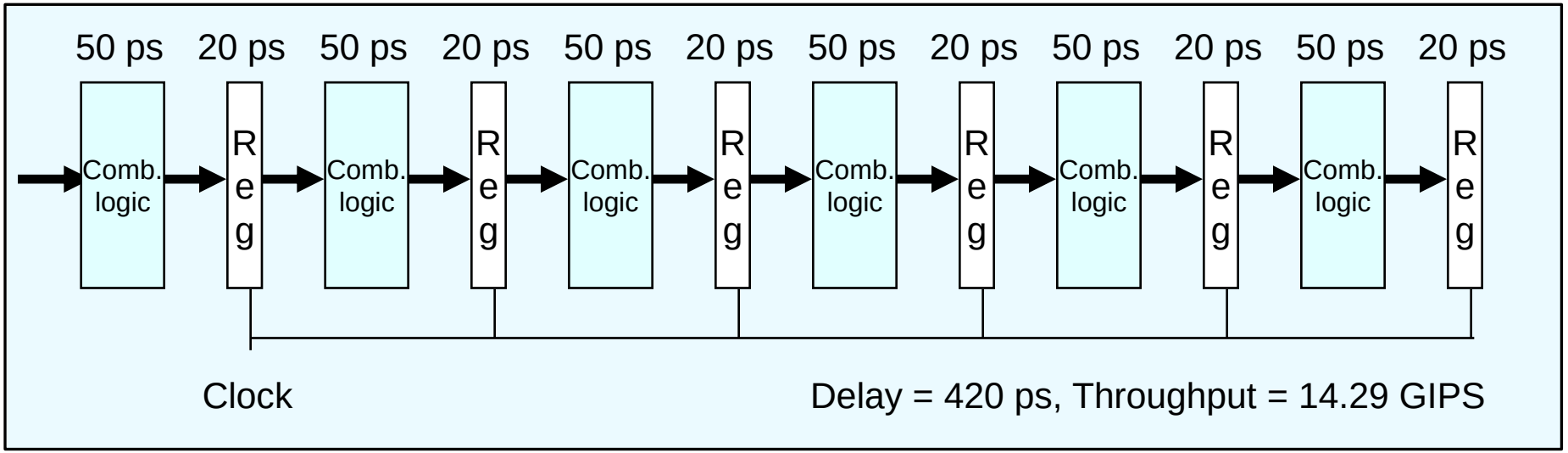
- Non-uniform delays
- Overhead
- Pipeline hazards

Limitations: Nonuniform Delays



- Throughput limited by slowest stage
- Other stages sit idle for much of the time
- Challenging to partition system into balanced stages

Limitations: Register Overhead

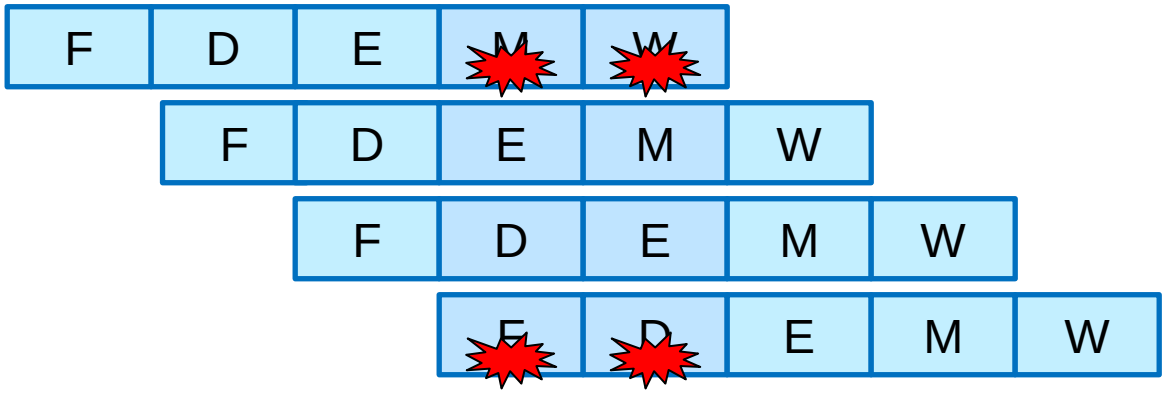


- As we try to deepen the pipeline, the overhead of loading registers becomes more significant
- Percentage of clock cycle spent loading register:
  - 1-stage pipeline: 6.25%
  - 3-stage pipeline: 16.67%
  - 6-stage pipeline: 28.57%
- High speeds of modern processor designs obtained through very deep pipelining

Pipeline Hazards

- Three major hurdles of pipelining
  - Structural Hazard
  - Data Hazard
  - Control Hazard

Structural Hazard



Instruction number	Clock number								
	1	2	3	4	5	6	7	8	9
Load Instruction	F	D	E	M	W				
Instruction $i+1$		F	D	E	M	W			
Instruction $i+2$			F	D	E	M	W		
Instruction $i+3$				F	D	E	M	W	
Instruction $i+4$					F	D	E	M	W

- and any other form of hardware resource contention

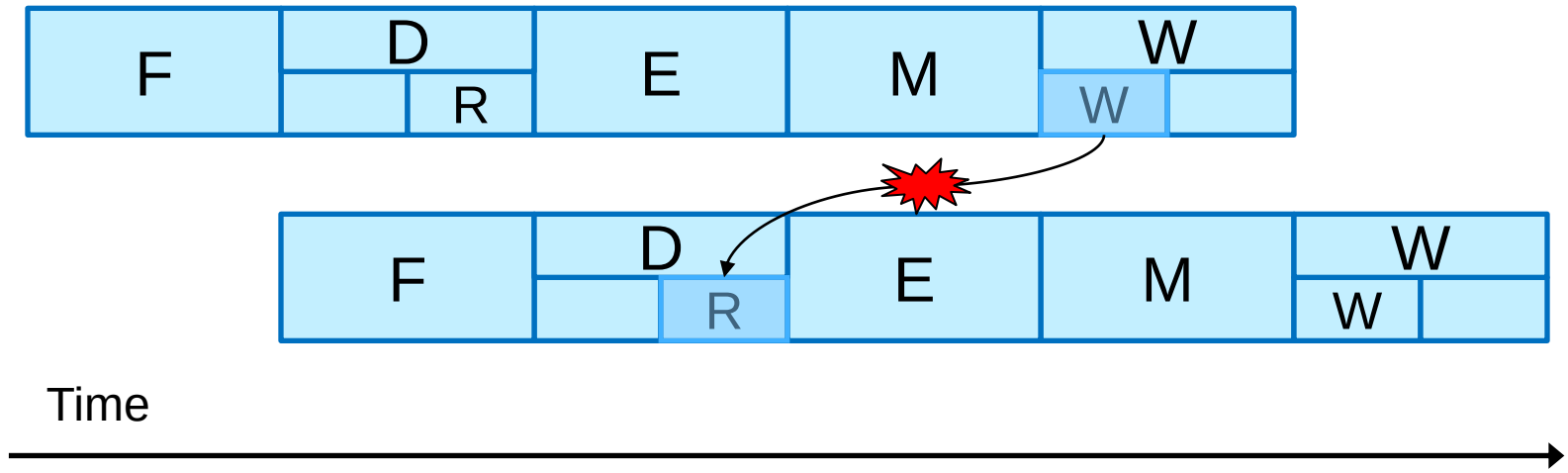
Solutions to Structural Hazards

- Resource Duplication
  - example
    - Separate I and D caches for memory access conflict
    - Time-multiplexed or multi-port register file for register file access conflict

Data Hazard (RAW hazard)

OP *i* addl %edx, %eax

OP *i+1* subl %eax, %ecx

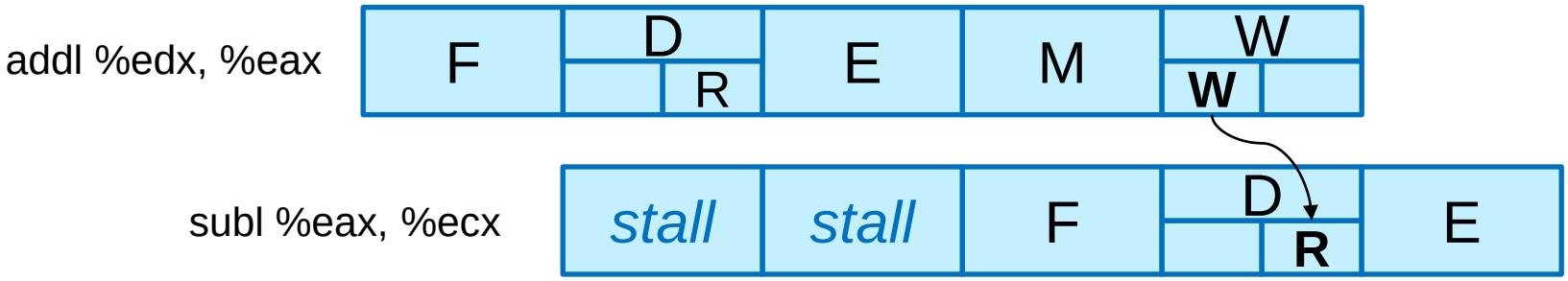


Solutions to Data Hazards

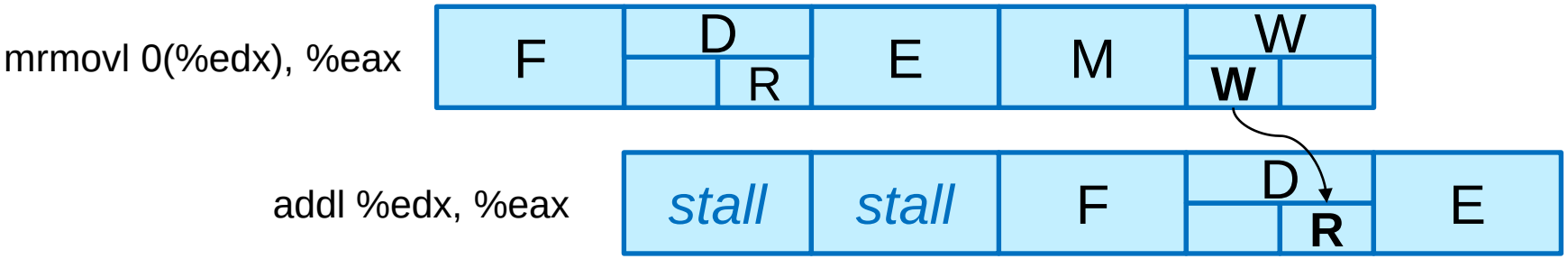
- Freezing the pipeline
- (Internal) Forwarding
- Compiler scheduling

Freezing The Pipeline

- ALU result to next instruction

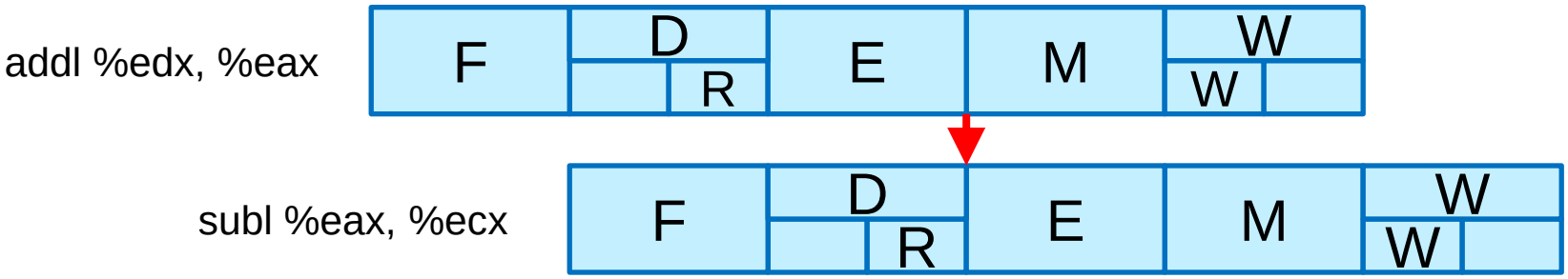


- Load result to next instruction

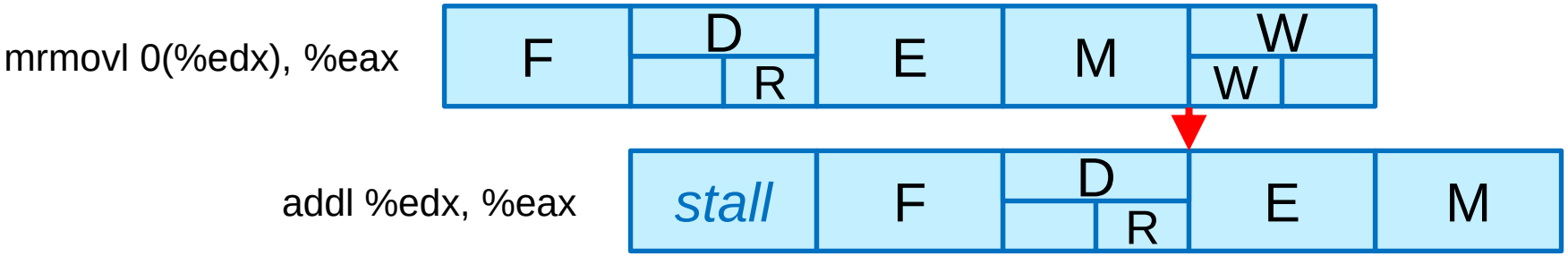


(Internal) Forwarding

- ALU result to next instruction (Stall X)
  - requires forwarding path



- Load result to next instruction (Stall 1)
  - requires forwarding path

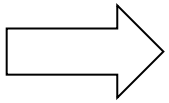


Load interlock

Compiler Scheduling

- Rearranging instructions in order to avoid data hazards

```
...
addl %eax, %ebx
rmmovl 0(%edx), %ebx
mrmovl 4(%edx), %eax
addl %eax, %ebx
irmovl $4, %ecx
...
```



```
...
addl %eax, %ebx
mrmovl 4(%edx), %eax
rmmovl 0(%edx), %ebx
irmovl $4, %ecx
addl %eax, %ebx
...
```

- requires a careful data flow analysis and knowledge about the inner workings of the pipeline

Control Hazard

- Caused by PC-changing instructions
  - (conditional/unconditional) jumps, function call/return

Branch Instruction	F	D	E	M	W				
Branch successor		F	*stall*	*stall*	F	D	E	M	W
Branch successor + 1						F	D	E	M	W
Branch successor + 2							F	D	E	M
Branch successor + 3								F	D	E
Branch successor + 4									F	D
Branch successor + 5										F

For 5-stage pipeline, 3 cycle penalty
15% branch frequency. CPI = 1.45

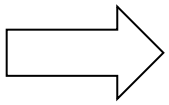
Solutions to Control Hazards

- Branch delay slots
- Branch prediction

Branch Delay Slots

- Execute the next *n* instructions following a branch no matter what the result of the branch
  - up to the compiler to make good use of the branch delay slots
  - exposes the pipeline structure to the user

```
  addl %eax, %ebx
  mrmovl 0(%edx), %eax
  cmpl %ecx, %esi
  jge .false
.true
  ...
```



actual jump

```
  cmpl %ecx, %esi
  jge .false
  addl %eax, %ebx
  mrmovl 0(%edx), %eax
.true
  ...
```

branch taken: stall 2 cycles
not taken: no stall

addl, mrmovl executed no matter whether the jump is taken or not (no stall)

Branch Prediction

- Predict-taken

Taken branch instruction	F	D	E	M	W					
Branch target		F	D	E	M	W				
Branch target + 1			F	D	E	M	W			
Branch target + 2				F	D	E	M	W		
Branch target + 3					F	D	E	M	W	

Untaken branch instruction	F	D	E	M	W					
Branch target		F	*idle*	*idle*	*idle*	*idle*				
Untaken branch instruction + 1			F	D	E	M	W			
Untaken branch instruction + 2				F	D	E	M	W		
Untaken branch instruction + 3					F	D	E	M	W	

~60% success rate

- other techniques:
  - predict backwards-taken, forward-untaken
  - hardware branch prediction