Lecture 16: Pipelining

CMPE 120: Computer Organization & Architecture

Olivia Weng

Logistics

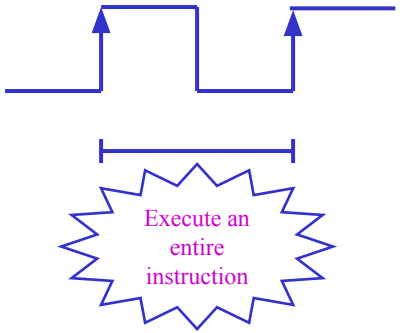
- Reminder: check-in with professor is 1% of your final grade
  - Sign up link

- HW3 is due today at 11:59pm

- HW4 out today - start early!!
  - 3 weeks to do for a reason

Review: single cycle processor

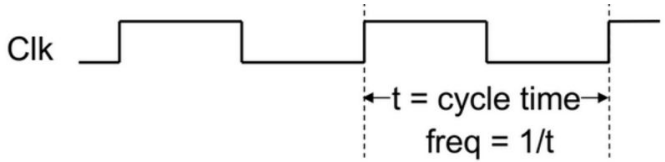
- Processor design (datapath and control) will determine:
  - Cycle time (CT)
  - Cycles per instruction (CPI)
- Single cycle processor:
  - Advantage: One clock cycle per instruction
  - Disadvantage: long cycle time



- ET = Instr Count * CPI * CT

Review: Logic design basics

- From basics we can understand how to implement an ISA as a processor



- Clock: measures speed of processor
- Logic elements
  - Combinational logic
  - State (sequential) elements
- Memory elements
  - Large storage separate from logic elements

Review: Programmer visible (architectural) state

- Memory
- Registers
- Program counter (PC)
  - register storing memory address of current instruction

Instructions (and programs) specify how to transform values of programmer visible state

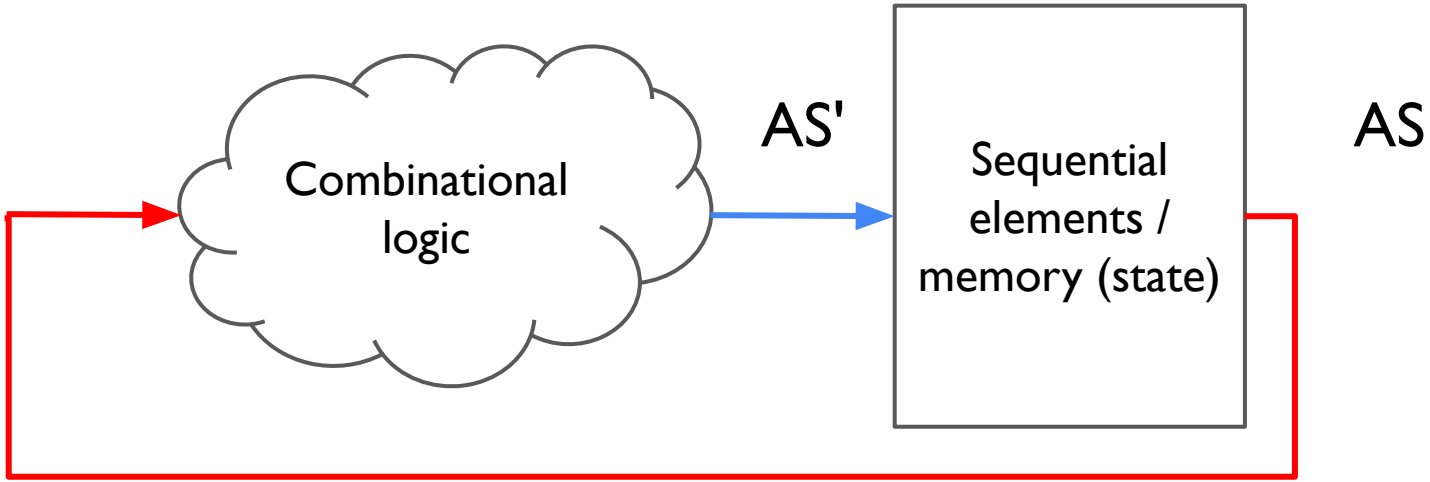
Review: Instruction processing

- ISA specifies what AS' should be given instruction and AS

- Microarchitecture implements how AS transforms to AS'
  - How instruction is processed by machine (organization)
  - Choice 1: AS → AS' (transform in 1 cycle)

Review: A single cycle processor

- Each instruction takes single cycle to execute



Single cycle processor summary

- Inefficient: all instructions run as slow as slowest instruction

Single cycle processor summary

- Inefficient: all instructions run as slow as slowest instruction

- Not easy to optimize/improve performance
  - Optimizing the common case does not work!
  - Need to optimize the worst case

Single cycle processor summary

- Inefficient: all instructions run as slow as slowest instruction

- Not easy to optimize/improve performance
  - Optimizing the common case does not work!
  - Need to optimize the worst case

- All resources are not fully utilized
  - Some resources sit idle!

Single cycle processor summary

- Inefficient: all instructions run as slow as slowest instruction

- Not easy to optimize/improve performance
  - Optimizing the common case does not work!
  - Need to optimize the worst case

- All resources are not fully utilized
  - Some resources sit idle!

How to do better?

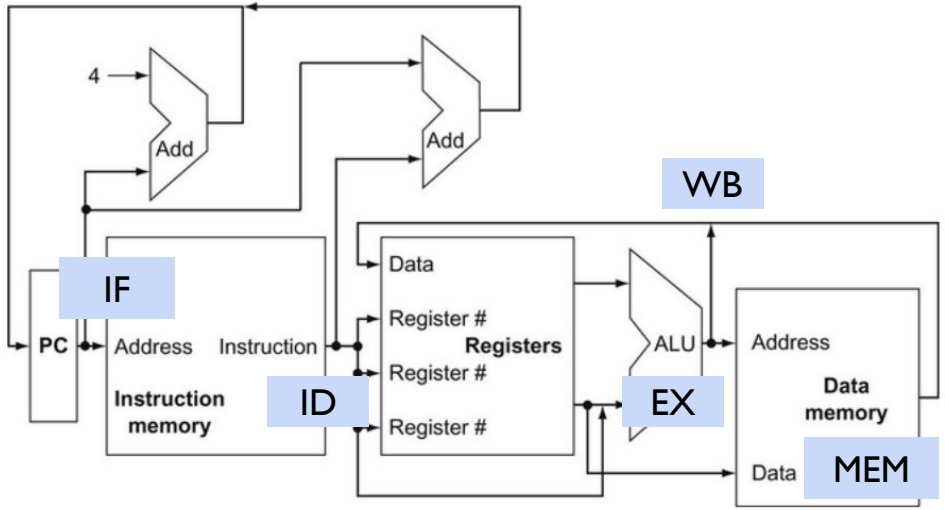
Review: Instruction processing

- ISA specifies what AS' should be given instruction and AS

- Microarchitecture implements how AS transforms to AS'
  - How instruction is processed by machine (organization)
  - Choice 1: AS → AS' (transform in 1 cycle)
  - Choice 2: AS → AS+MS1 → AS+MS2 → AS+MS3 → AS'
    - MS = microarchitectural state
    - Transform AS → AS' in multiple cycles

Instruction processing steps

- Example: 5 steps to implement RISC-V
  - Instruction fetch (IF)
  - Instruction decode and register operand fetch (ID)
  - Execute/evaluate memory address (EX)
  - Memory operand fetch (MEM)
  - Store/writeback result (WB)

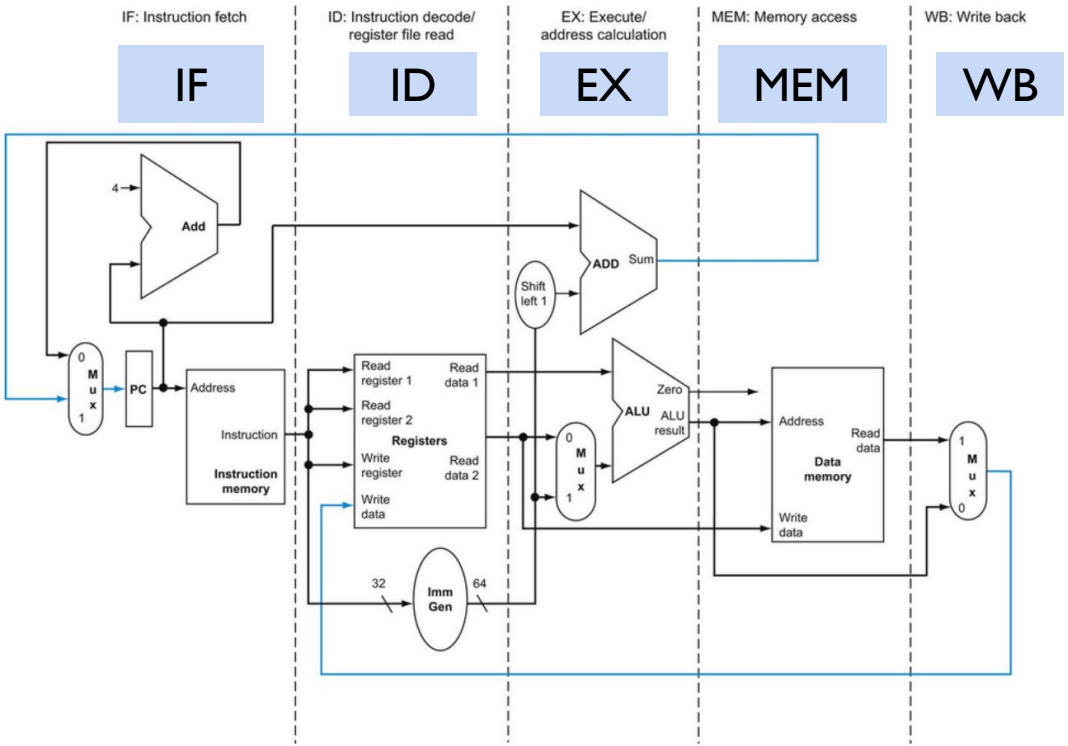


The multi-cycle approach

- Process each step in 1 cycle → take 5 cycles to process the 5 steps
  - Instead of 1 cycle to process all 5 steps

Multi-cycle processor in action

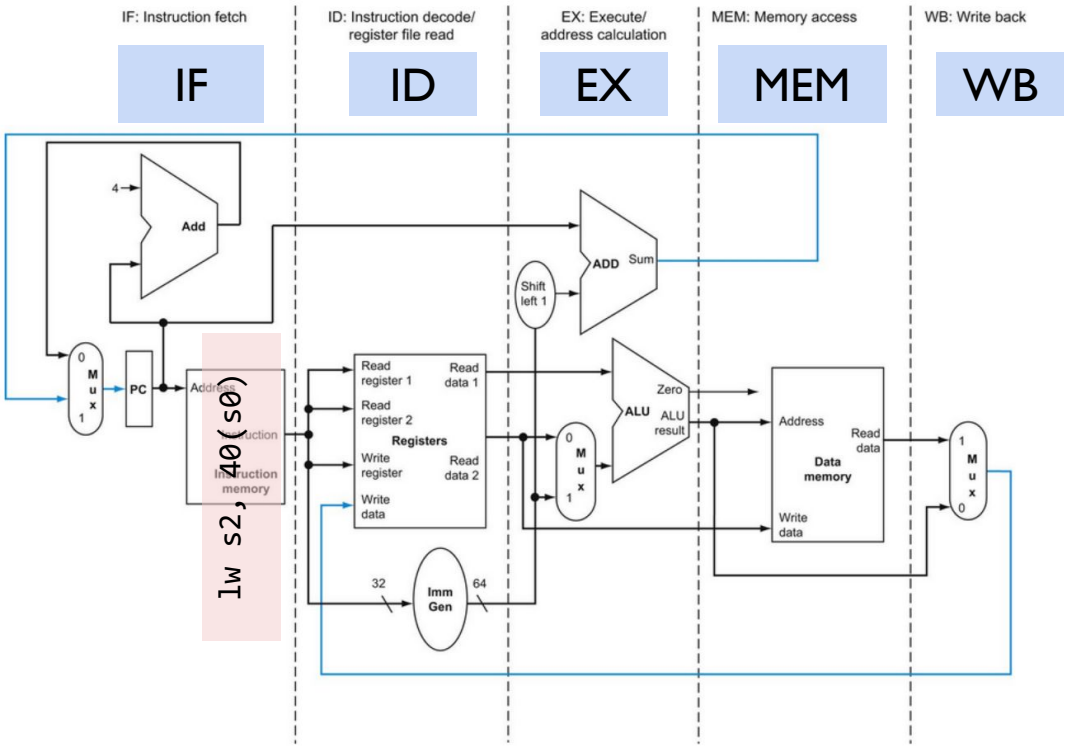
```
lw s2, 40(s0)
```



Multi-cycle processor in action

```
lw s2, 40(s0)
```

- Fetch

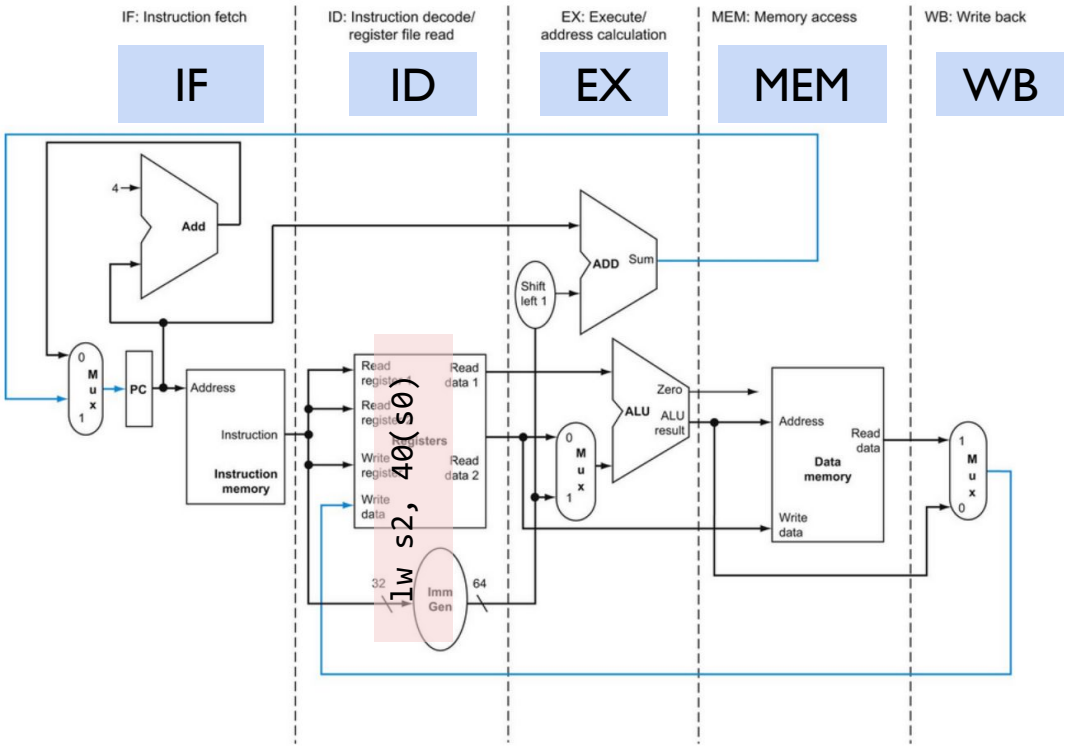


cycle 1

Multi-cycle processor in action

```
lw s2, 40(s0)
```

- Decode

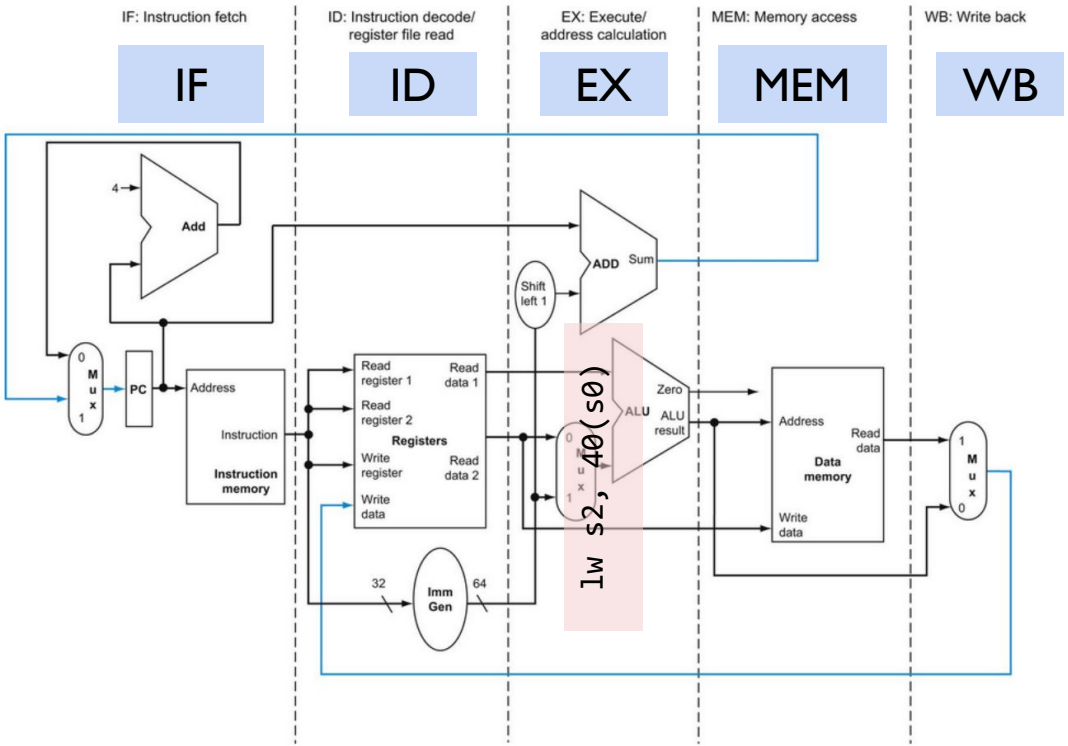


cycle 2

Multi-cycle processor in action

```
lw s2, 40(s0)
```

- Execute

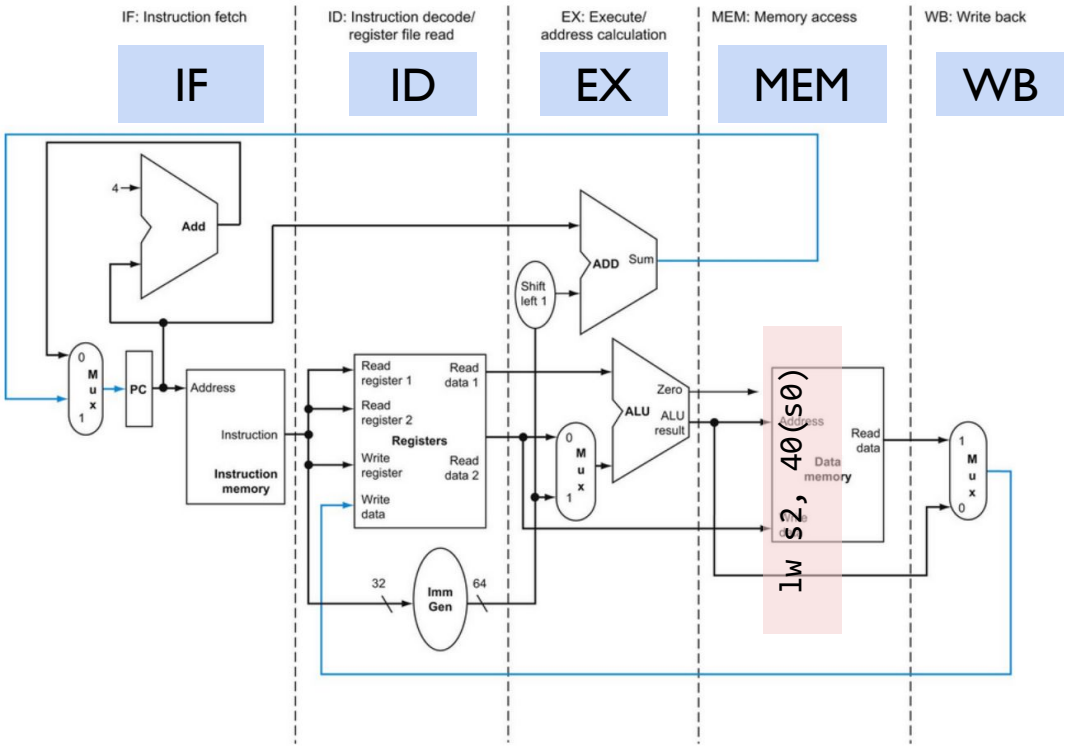


cycle 3

Multi-cycle processor in action

```
lw s2, 40(s0)
```

- Memory

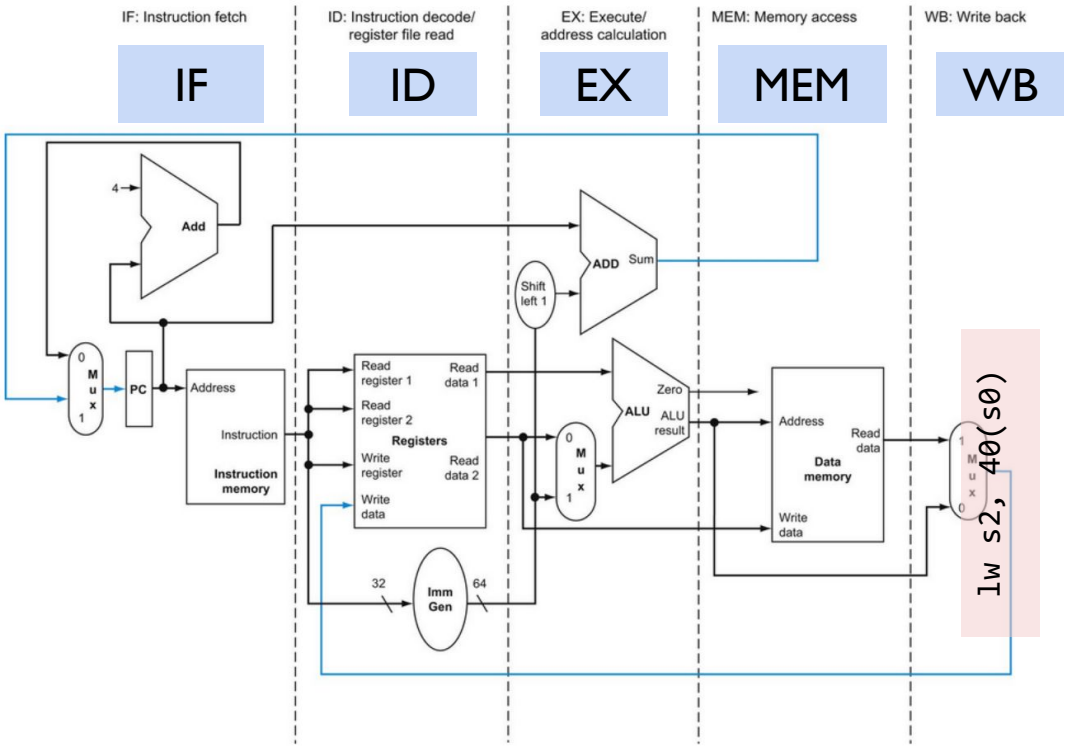


cycle 4

Multi-cycle processor in action

```
lw s2, 40(s0)
```

- Writeback



cycle 5

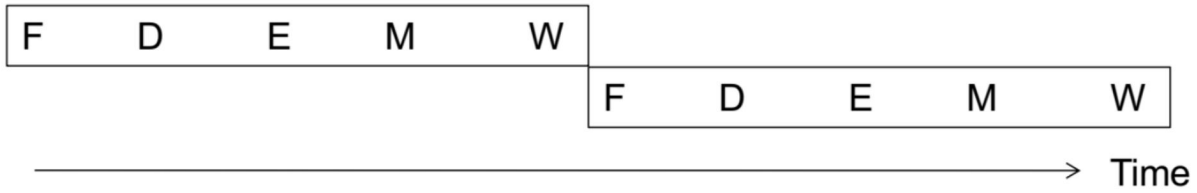
The multi-cycle approach

- Process each step in 1 cycle → take 5 cycles to process 5 steps
  - Instead of 1 cycle to process all 5 steps

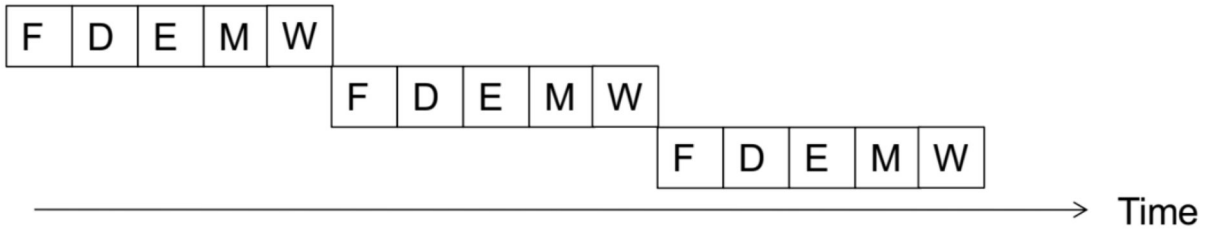
  - Can also use multiple cycles for 1 processing step
    - Ex: MEM can take 100 cycles
    - Different instructions will take different number of cycles to finish

Single-cycle vs multi-cycle

- Single-cycle: 1 cycle per instruction
  - Cycle time is long



- Multi-cycle: 5 cycles per instruction
  - Cycle time is short



Multi-cycle processors

- Multi-cycle processors were dominant until the 90s
- CPI = average number of steps required per instruction
- What determines cycle time?

Multi-cycle processors

- Multi-cycle processors were dominant until the 90s
- CPI = average number of steps required per instruction
- What determines cycle time?
  - The slowest step

Multi-cycle performance

- What are the main concerns with multi-cycle design?

A. Too long of a cycle time
B. Too high of a CPI
C. Too many idle resources
D. Cannot optimize the common case

Multi-cycle considerations

- Does this design make good use of hardware?

- How could it make better use of hardware?

Pipelining: an assembly line for instructions

- Idea: Divide instruction processing into distinct stages
  - Process a different instruction in each stage
  - Use all stages all the time to leave no resource idle!
  - → Instruction-level parallelism

Pipelining: an assembly line for instructions

- Idea: Divide instruction processing into distinct stages
  - Process a different instruction in each stage
  - Use all stages all the time to leave no resource idle!
  - → Instruction-level parallelism

- For every cycle
  - start an instruction
  - finish an instruction

- Does this help improve instruction processing latency or throughput?

Pipelining: an assembly line for instructions

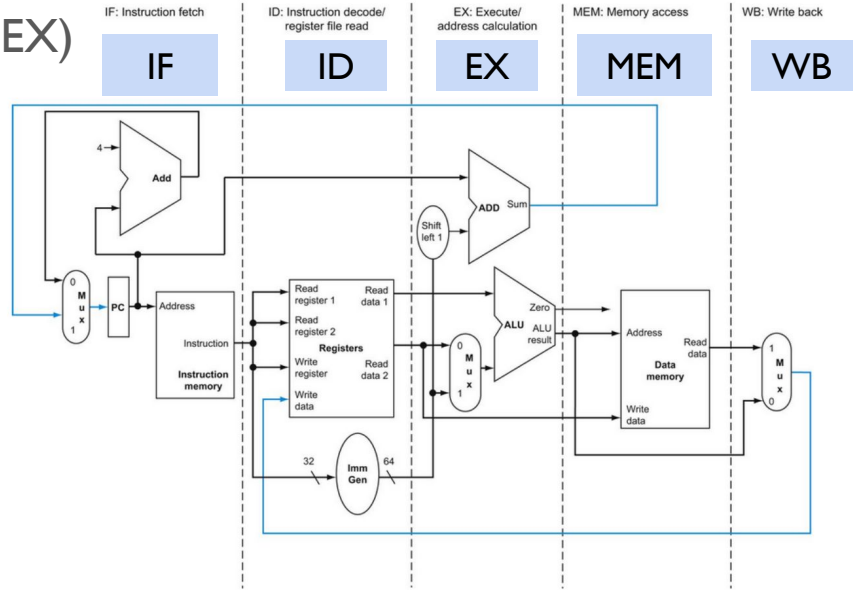
- Idea: Divide instruction processing into distinct stages
  - Process a different instruction in each stage
  - Use all stages all the time to leave no resource idle!
  - → Instruction-level parallelism

- For every cycle
  - start an instruction
  - finish an instruction

- Does this help improve instruction processing latency or throughput?
  - Increase instruction processing throughput

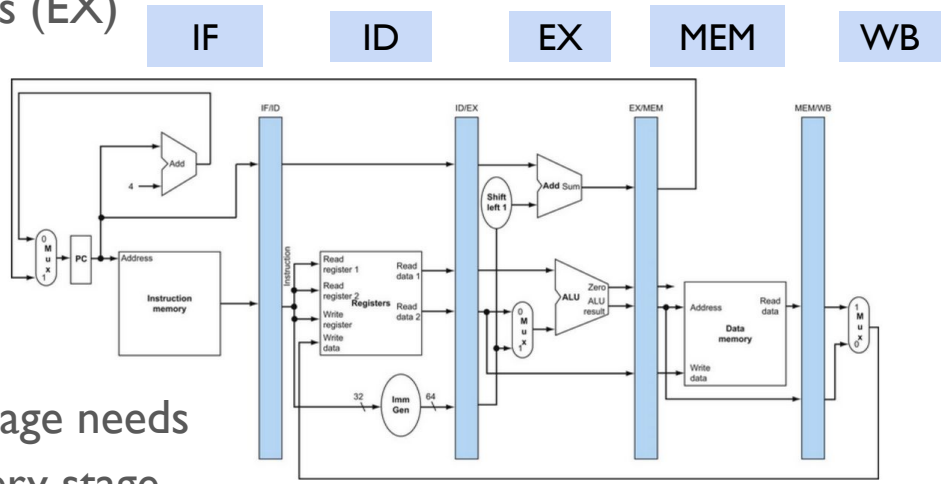
Pipeline stages

- 5 stages to process RISC-V instructions
  - Instruction fetch (IF)
  - Instruction decode and register operand fetch (ID)
  - Execute/evaluate memory address (EX)
  - Memory access (MEM)
  - Store/writeback result (WB)



Pipeline stages

- 5 stages to process RISC-V instructions
  - Instruction fetch (IF)
  - Instruction decode and register operand fetch (ID)
  - Execute/evaluate memory address (EX)
  - Memory access (MEM)
  - Store/writeback result (WB)



- Pipeline registers between stages
  - Store intermediate values each stage needs
  - Ex: instruction data needed in every stage

Pipelining in action

```
lw s2, 40(s0)

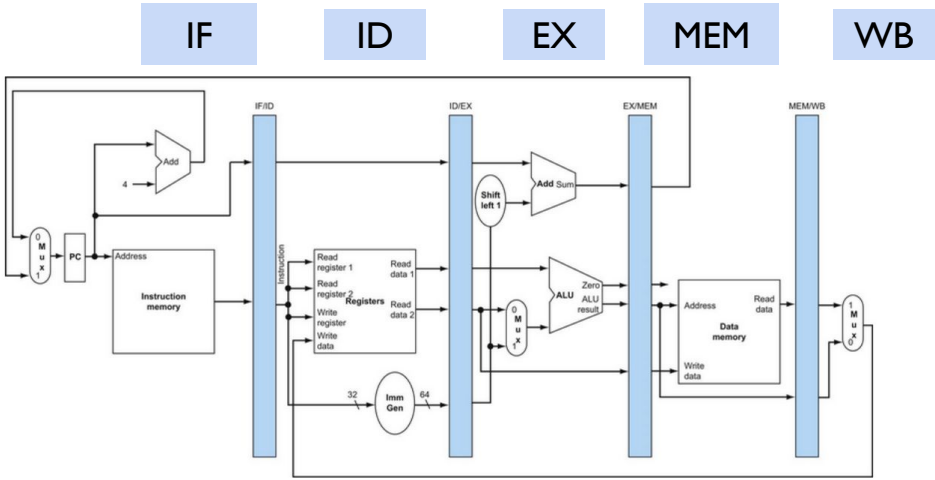
add s3, s9, s10

sub s4, t1, s8

and s5, s11, t0

sw s6, 20(t4)

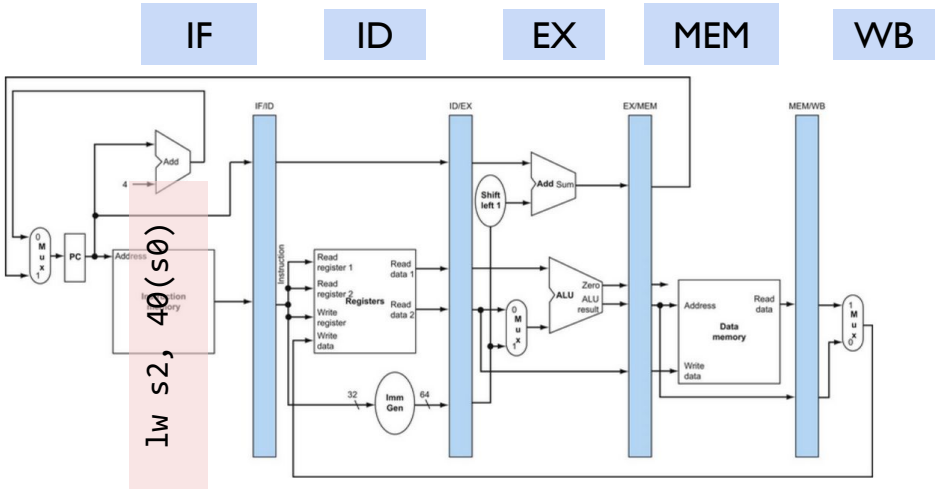
or s7, t2, t3
```



Pipelining in action

```
lw s2, 40(s0)
add s3, s9, s10
sub s4, t1, s8
and s5, s11, t0
sw s6, 20(t4)
or s7, t2, t3
```

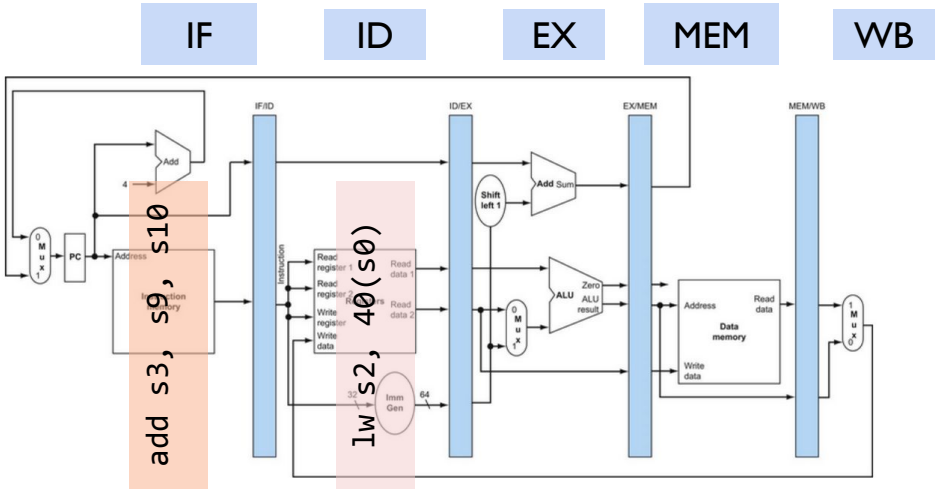
cycle 1



Pipelining in action

```
lw s2, 40(s0)
add s3, s9, s10
sub s4, t1, s8
and s5, s11, t0
sw s6, 20(t4)
or s7, t2, t3
```

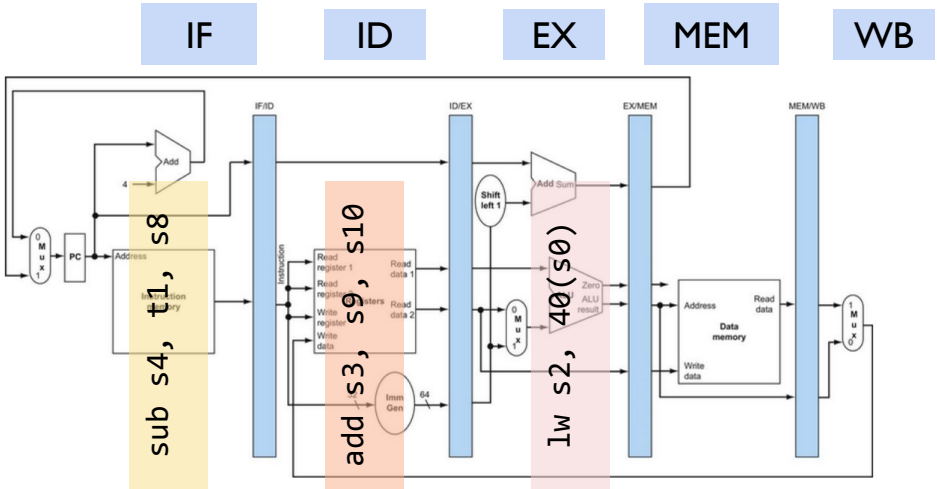
cycle 2



Pipelining in action

```
lw s2, 40(s0)
add s3, s9, s10
sub s4, t1, s8
and s5, s11, t0
sw s6, 20(t4)
or s7, t2, t3
```

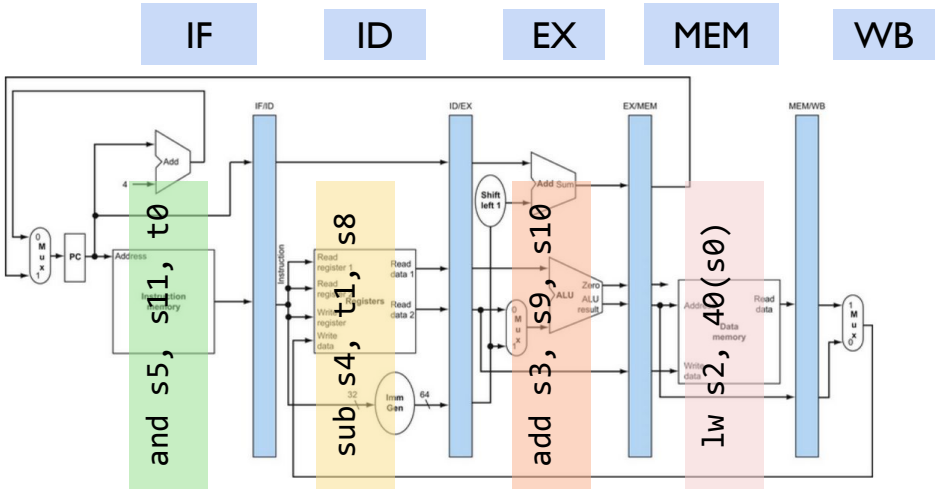
cycle 3



Pipelining in action

```
lw s2, 40(s0)
add s3, s9, s10
sub s4, t1, s8
and s5, s11, t0
sw s6, 20(t4)
or s7, t2, t3
```

cycle 4



Pipelining in action

```
lw s2, 40(s0)

add s3, s9, s10

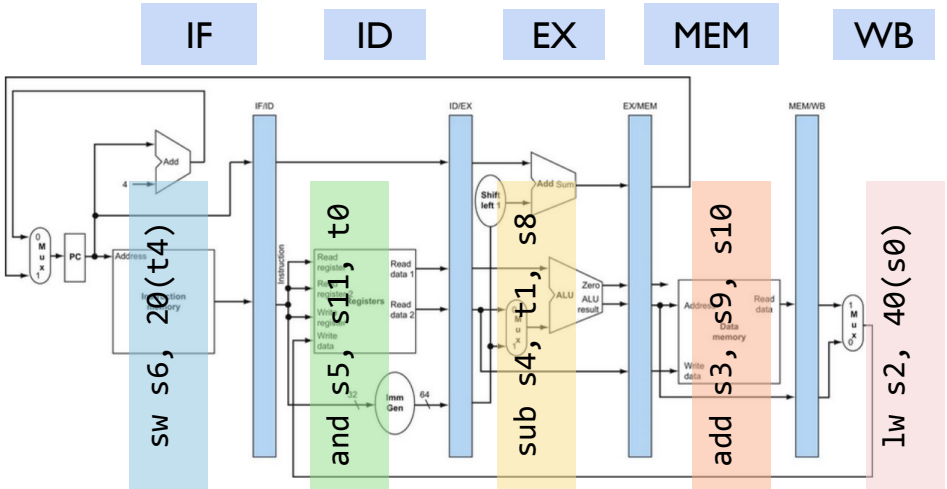
sub s4, t1, s8

and s5, s11, t0

sw s6, 20(t4)

or s7, t2, t3
```

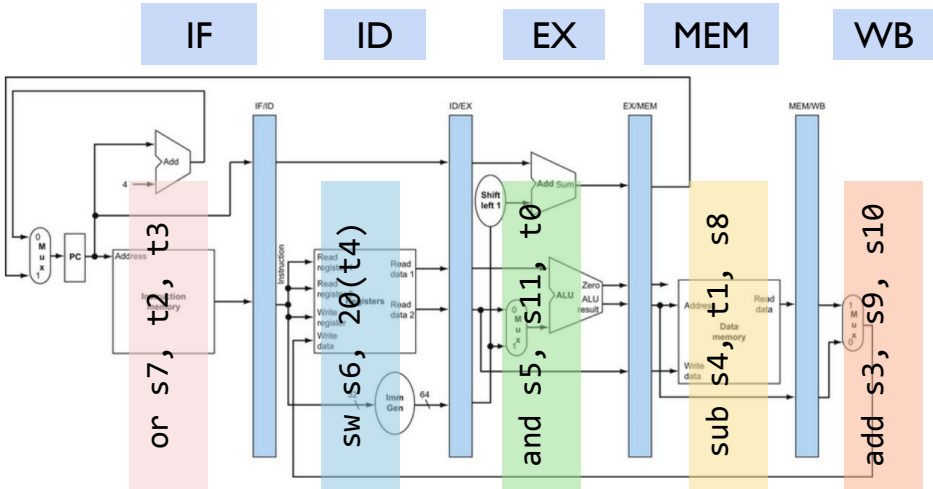
cycle 5



Pipelining in action

```
lw s2, 40(s0) - retired!
add s3, s9, s10
sub s4, t1, s8
and s5, s11, t0
sw s6, 20(t4)
or s7, t2, t3
```

cycle 6



Pipelining in action

```
lw s2, 40(s0) - retired!

add s3, s9, s10 - retired!

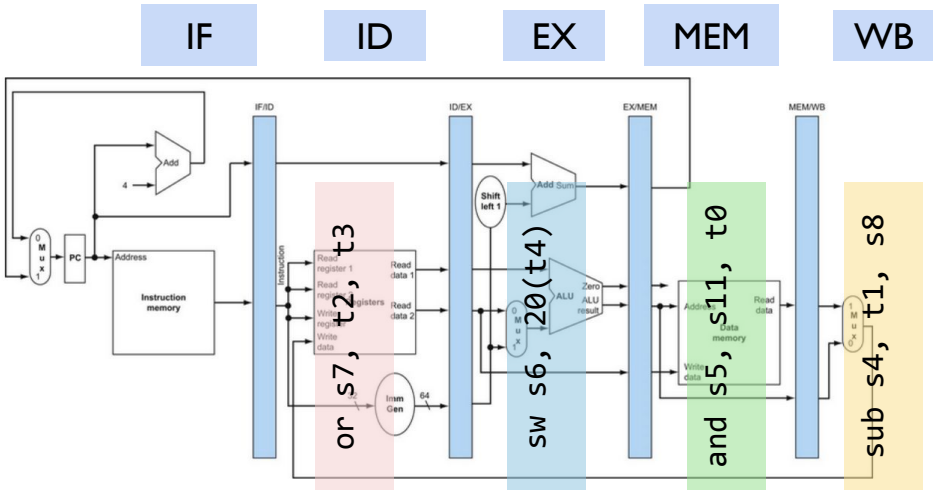
sub s4, t1, s8

and s5, s11, t0

sw s6, 20(t4)

or s7, t2, t3
```

cycle 7



Pipelining in action

```
lw s2, 40(s0) - retired!

add s3, s9, s10 - retired!

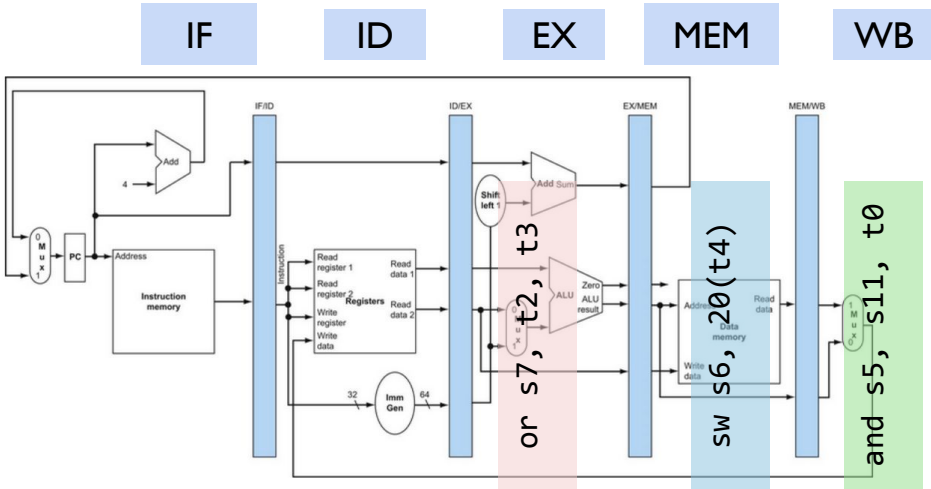
sub s4, t1, s8 - retired!

and s5, s11, t0

sw s6, 20(t4)

or s7, t2, t3
```

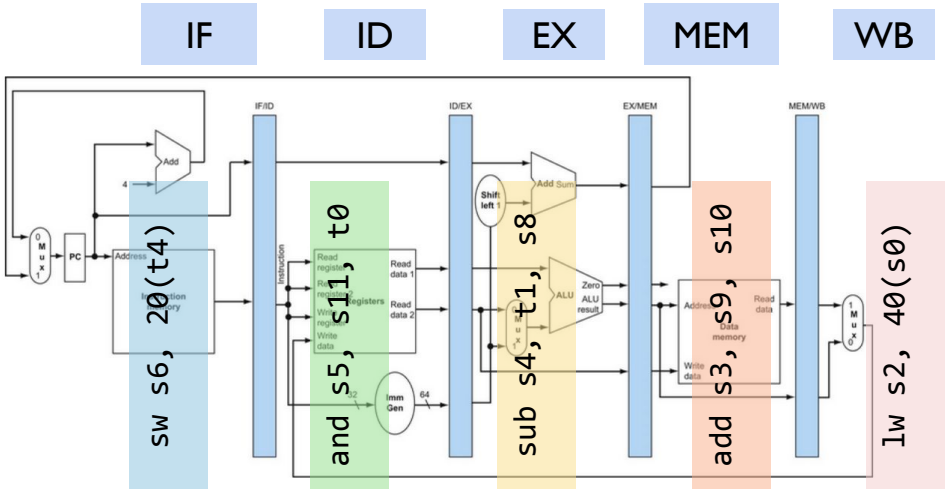
cycle 9



Pipelining in action

- In the steady state, every stage is active.
  - No idle hardware!

cycle 5

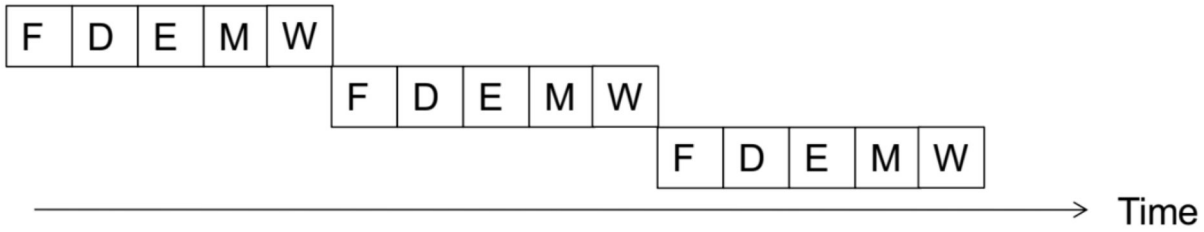


Pipelining improves instruction processing

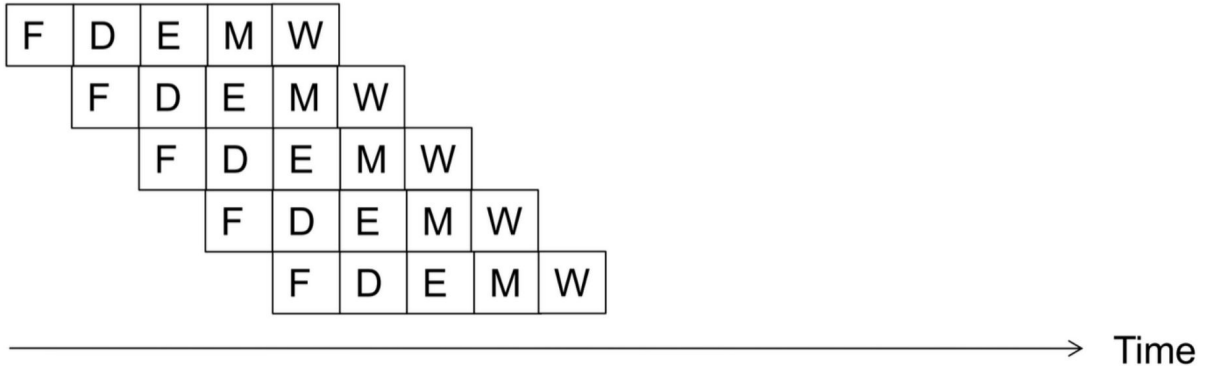
A. Throughput
B. Latency

Multi-cycle vs pipelined

- Multi-cycle: 5 cycles per instruction



- Pipelined: 5 cycles per 5 instructions (in steady state)



How does pipelining impact performance?

	Pipelined CPI	# cycles to execute 1 instruction
A	1	Goes down by 4
B	5	Goes down by 1
C	varies	Depends on the instructions
D	1	Remains roughly constant
E	6	Goes up by 1

Pipelining and the performance equation

- Instruction count
  - Determined by ISA and compiler

- CPI
  - Without pipelining, CPI = __5__
  - With pipelining, CPI = _1___, in the best case _1__ instruction starts and finishes execution each cycle

- CT
  - Set by critical path in the slowest stage
  - Pipelining does not change much here.

Key idea of pipelining

- Throughput >>> latency

- Computers are useful because they do a lot of things well
  - Much less important how well any one thing is done

- Exploits instruction-level parallelism