CMPT 295

Unit – Microprocessor Design & Instruction Execution

Lecture 30 – Staged and Pipelined Execution

Last Lecture

- We put combinational logic circuits and sequential logic circuits together -> datapath of a microprocessor
- Various models of ***Microprocessor machine instruction execution***:
  - **Model 1: Sequential execution of machine instructions**
    - The microprocessor we have just constructed is a **sequential execution of machine instructions** type of microprocessor since it executes one machine instruction **at a time** and **per clock cycle**
    - ***Single-cycle*** microprocessor (**CPI = 1**)
- In general, how to analyze various models of ***microprocessor instruction execution***:
  1. Latency (a.k.a. propagation delay): Time required to execute a single instruction
  2. Throughput: Number of instructions executed per second
- Conclusion of analyzing **sequential execution of machine instructions**
  - Because this model requires a long clock cycle
  - → Creates slow microprocessors with small throughput

Today's Menu

- Instruction Set Architecture (ISA)
  - Definition of ISA
- Instruction Set design
  - Design principles
  - Look at an example of an instruction set: MIPS
  - Create our own
  - ISA evaluation
- Implementation of a microprocessor (CPU) based on an ISA
  - Execution of machine instructions (datapath)
  - Intro to logic design + Combinational logic + Sequential logic circuit
  - Sequential execution of machine instructions
  - Pipelined execution of machine instructions + Hazards

How to speed up our microprocessor, i.e., improve its throughput?

- Staged execution
  - Let's divide the execution of a machine instruction into stages
    - Example: fast food order counter versus cafeteria
  - Stages:

1 machine instruction now split into several micro-instructions (micro-operations)!

F
- **Fetch**: read instruction located at address contained in PC into instruction register, compute address of "next" instruction

D
- **Decode**: decode instruction and read content of register operands (or implicit stack pointer register) from register file

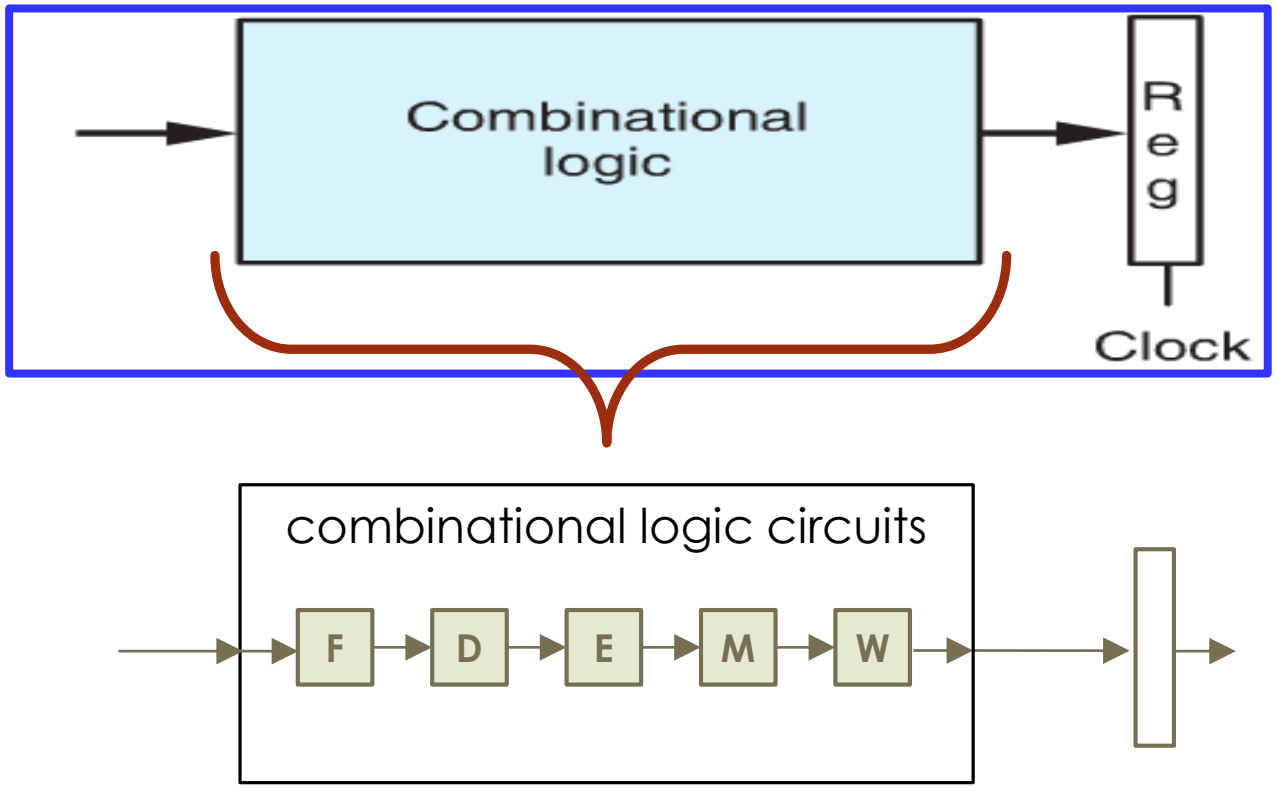
E
- **Execute**: perform operation using ALU unit according to opcode, increment/decrement stack pointer register, compute effective address, set condition codes

M
- **Memory**: read or write data values from/to memory (memory-access instruction)

W
- **Write back**: write values produced by ALU to file register, write values read from memory to file register, update stack pointer register

Model 2:
Staged execution of machine instructions

- So now, let's build a combinational logic circuit that performs each stage:



Example of Staged Datapath for MIPS

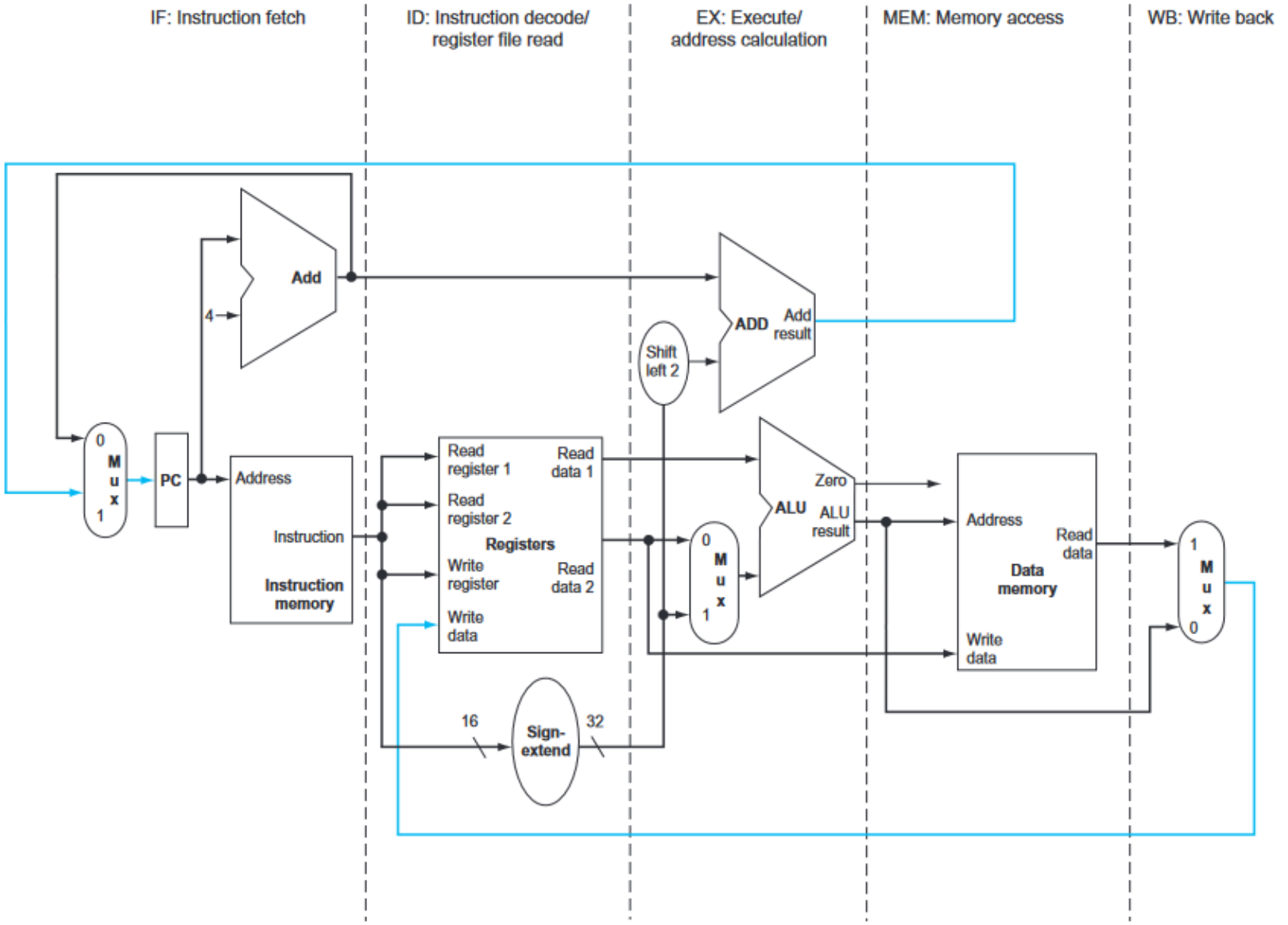


FIGURE 4.33 The single-cycle datapath from Section 4.4 (similar to Figure 4.17). Each step of the instruction can be mapped onto the datapath from left to right. The only exceptions are the update of the PC and the write-back step, shown in color, which sends either the ALU result or the data from memory to the left to be written into the register file. (Normally we use color lines for control, but these are data lines.)

Walking through the 5 stages with `lw`

instruction: `lw $t0, 4($t1)`

Meaning: $t0 = M[$t1+4]

instruction format:

Instruction composed of 1 word (32 bits)

opcode	rs	rt	Immediate or address

machine code:

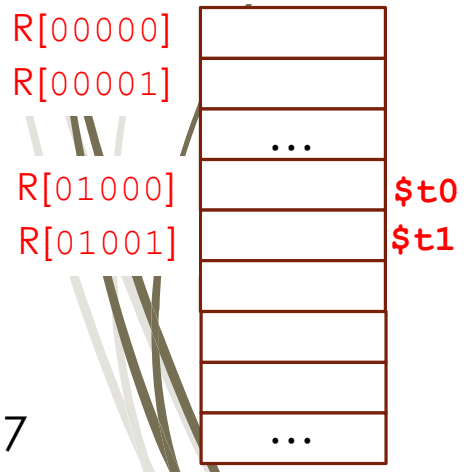
100011	01001	01000	0000 0000 0000 0100

Instruction register IR

1. Fetch (F)
   1. IR <- M[PC] -> fetching 1 word worth of instruction
   2. PC <- PC + 4 bytes -> next instruction is 1 word (4 bytes) further in memory
2. Decode (D)
   1. valA <- **R**[01001]
   2. val**I** <- 4 i.e., 0000 0000 0000 0100

   valA passed from D to E via the pipeline registers so that valA available in E stage
3. Execute (E) → valE <- valA + val**I**
4. Memory (M) → valM <- M[valE]
5. Write back (W) → **R**[01000] <- valM

R: Register File



Homework: go over this slide & make sure you understand what is happening!

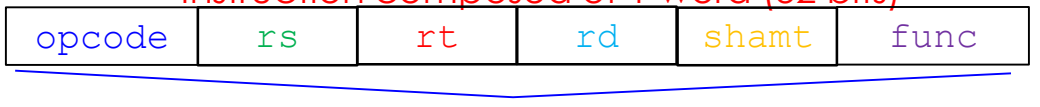
Walking through the 5 stages with `add`

instruction: `add $t0,$s1,$s2`

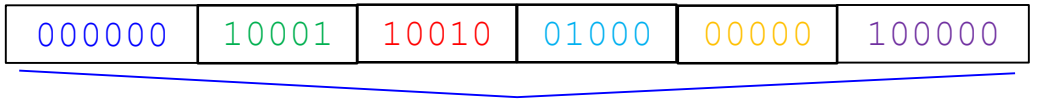
Meaning: $t0 = $s1 + $s2

instruction format:

Instruction composed of 1 word (32 bits)

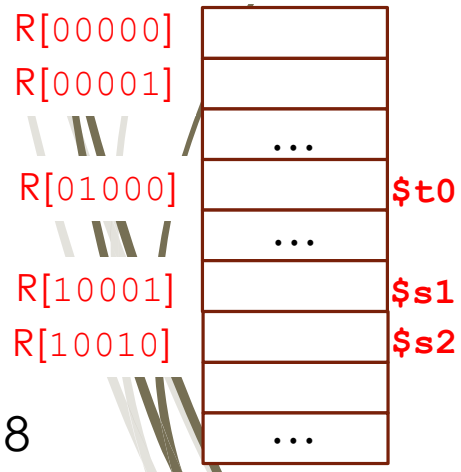


machine code:



Instruction register IR

R: Register File

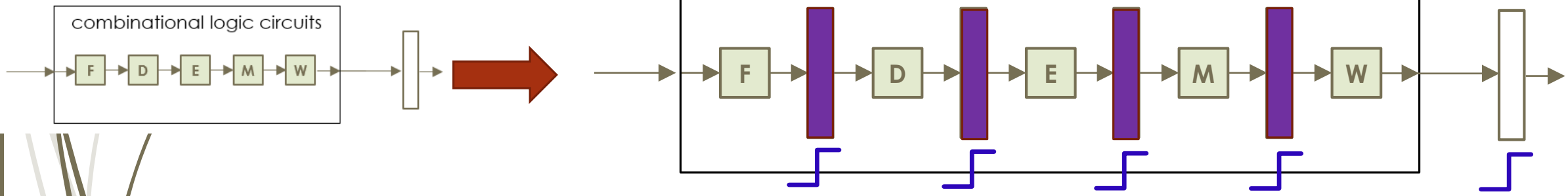


1. Fetch (F)
   1. IR <- M[PC] -> fetching 1 word worth of instruction
   2. PC <- PC + 4 bytes -> next instruction is 1 word (4 bytes) further in memory
2. Decode (D)
   1. valA <- **R**[10001]
   2. valB <- **R**[10010]
3. Execute (E) — valE <- valA + valB
4. Memory (M) – nothing happens in this stage
5. Write back (W) — **R**[01000] <- valE

Model 2:
Staged execution of machine instructions

- Since we need to save intermediate values after the execution of each stage (e.g., valA, valB, valE,…) , we place *clocked registers* after each stage
- These are called *pipeline registers*:

From Slide *5*:



- Add *pipeline registers* after each stage
  - These *pipeline registers* are invisible to us, s/w developers
  - Unlike the program registers and the PC which are visible to us

Example of MIPS Datapath with Pipeline Registers

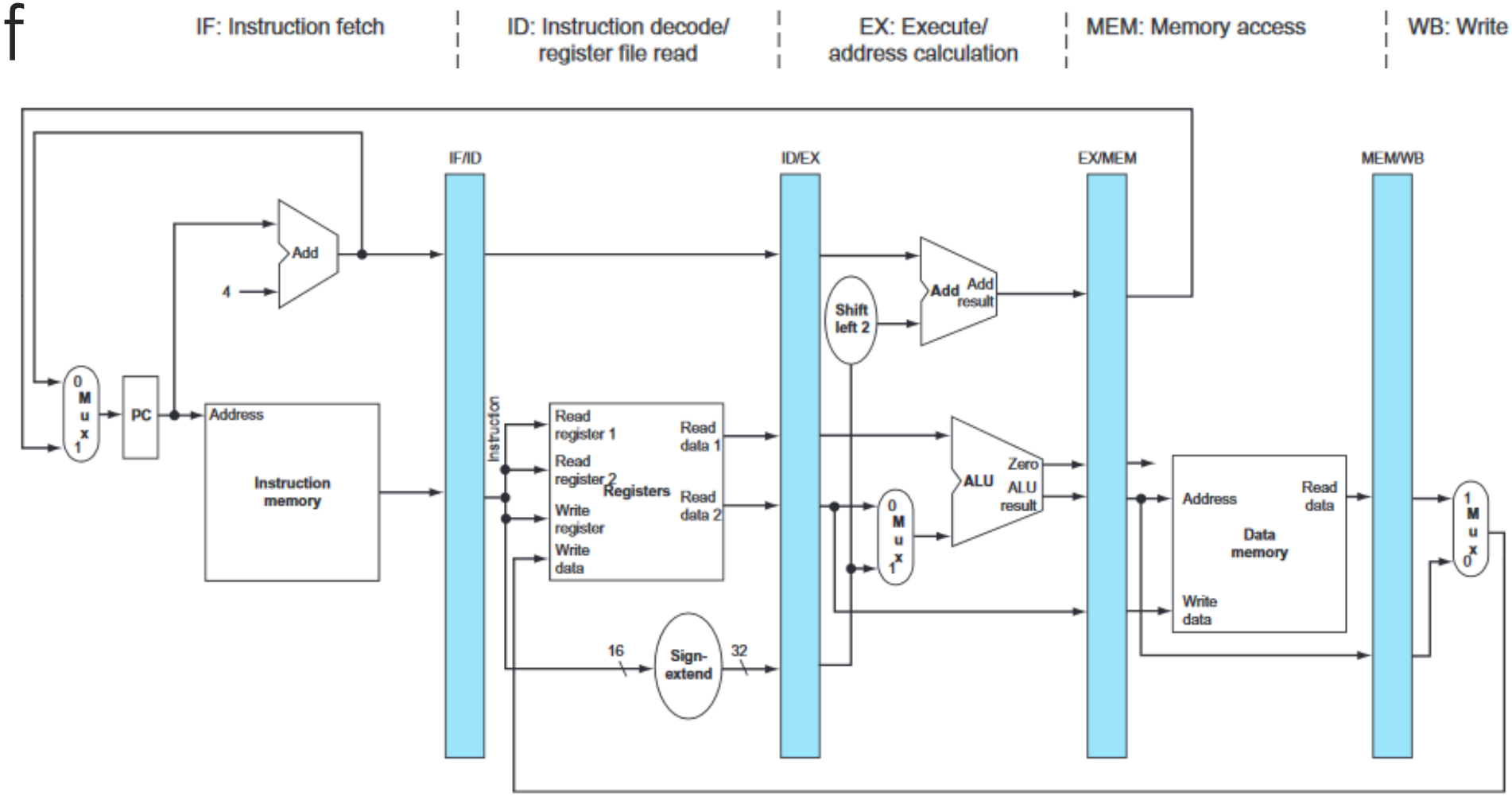
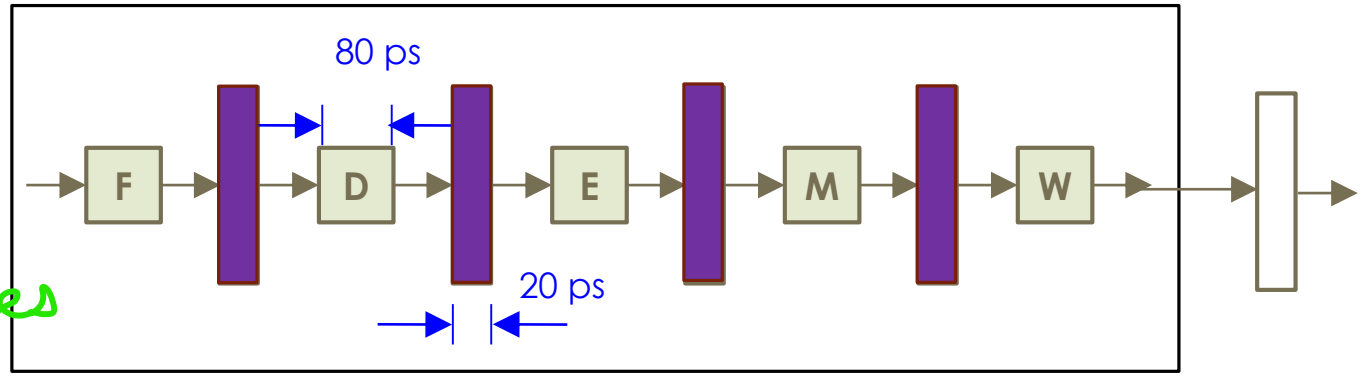


FIGURE 4.35 The pipelined version of the datapath in Figure 4.33. The pipeline registers, in color, separate each pipeline stage. They are labeled by the stages that they separate; for example, the first is labeled *IF/ID* because it separates the instruction fetch and instruction decode stages. The registers must be wide enough to store all the data corresponding to the lines that go through them. For example, the IF/ID register must be 64 bits wide, because it must hold both the 32-bit instruction fetched from memory and the incremented 32-bit PC address. We will expand these registers over the course of this chapter, but for now the other three pipeline registers contain 128, 97, and 64 bits, respectively.

Analysis: has the **throughput** improved?

of Staged execution (Model 2.)

⇒ microprocessor that executes 1 machine instruction @ a time and each instruction requiring 5 clock cycles in order to go through all 5 stages.



- Analysis: 1. Latency: $5 \cdot 80ps + 5 \cdot 20ps = 500ps$

Homework. 2. Throughput: $\frac{1}{\text{latency}} = \frac{1 \text{ instruction}}{500ps} = 2 \text{ GIPS}$

3. CPI = 5 → (1 instruction needs 5 ticks (clock cycles) in order to be executed)

- Issues:
  - Add 80 ps, but may not have to do all stages
  - Stages may not all have the same propagation delay

4. Minimum clock cycle = 80ps + 20ps = 100ps

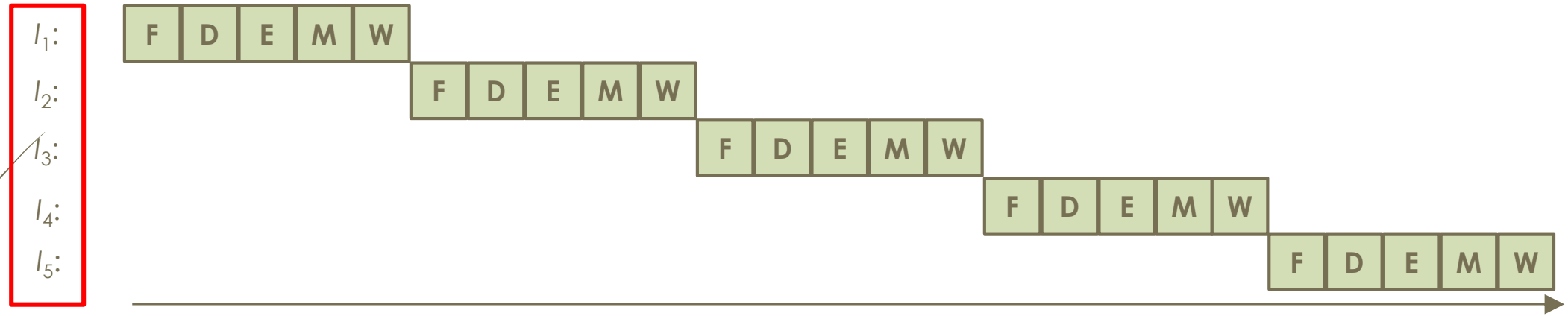
Back to the *cafeteria*

- The cafeteria is divided into sections
- It takes the same time for one customer to go through the sections of the cafeteria whether she is alone in the queue or not
  - Cafeteria has not decrease the time to service one customer
- The cafeteria is more efficient when there are more than one customer because these customers can be served at the same time: one customer per section
  - During the same amount of time, more customers are served: how many?
  - In "steady-state": when there is a customer in each section
  - So, the cafeteria improves the throughput
    - Throughput: # of customers served in a certain amount of time

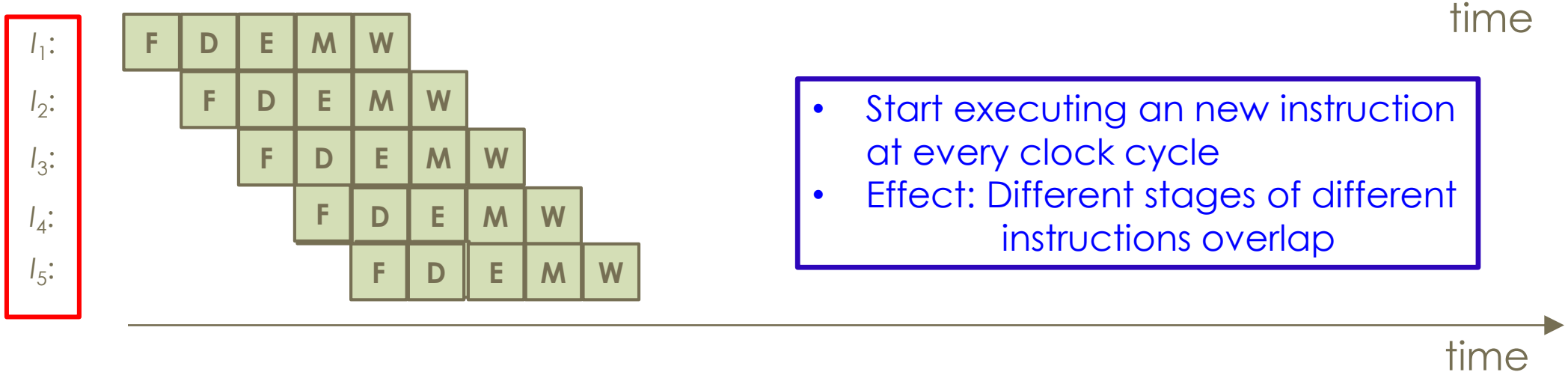
Pipelining

- Example: cafeteria with 1 customer in the queue versus cafeteria with many customers in the queue (1 customer in each section <= "steady-state")

Model 2. Staged Execution



Model 3. Pipelined Execution



- Start executing an new instruction at every clock cycle
- Effect: Different stages of different instructions overlap

Summary

Microprocessor machine instruction execution:

- How to analyze microprocessor instruction execution (performance):
  1. Latency (propagation delay): Time required to execute a single instruction
  2. Throughput: Number of instructions executed per second - GIPS

Model 1. Sequential execution of machine instructions

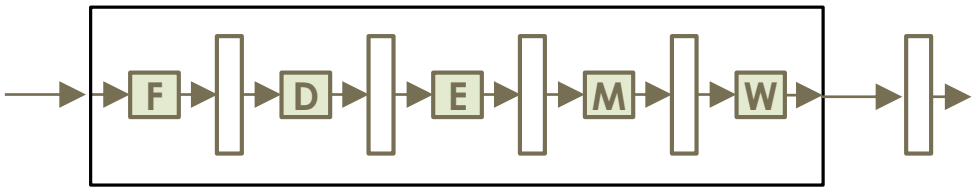
- Executing one machine instruction **at a time** and **per clock cycle** requires a long clock cycle
  - ***Single-cycle*** microprocessor (**CPI = 1**)
  - Result: computer with small throughput

How to improve throughput?

Model 2. Staged execution of machine instructions

- Divide the execution of instructions into stages: ***fetch***, ***decode***, ***execute***, ***memory***, ***write back***
- Introduce pipeline registers after each stage
- Result: Shorter (faster) clock cycle

  -> faster computer ☺
- Issues:
  - Adding pipeline registers increase latency
  - Stages may not have same propagation delay

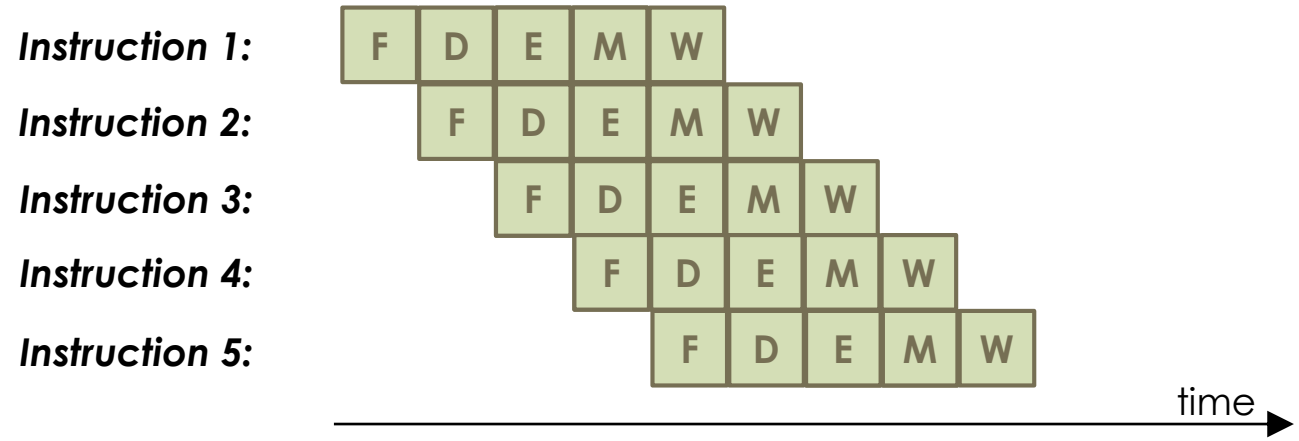


Summary

Microprocessor machine instruction execution:

Model 3. Pipelined execution of machine instructions

- Start executing 1 instruction at each clock cycle
- Effect: overlap different stages as different instructions are executing



With n stages, now executing n instructions in n clock cycles

- Analysis of pipelined execution using best case pipeline scenario (all stages have the same propagation delay) and "steady-state"
  - Increases CPU throughput

Next Lecture

- Instruction Set Architecture (ISA)
  - Definition of ISA
- Instruction Set design
  - Design principles
  - Look at an example of an instruction set: MIPS
  - Create our own
  - ISA evaluation
- Implementation of a microprocessor (CPU) based on an ISA
  - Execution of machine instructions (datapath)
  - Intro to logic design + Combinational logic + Sequential logic circuit
  - Sequential execution of machine instructions
  - Pipelined execution of machine instructions + Hazards