



# CMPT 295

Unit – Microprocessor Design & Instruction Execution

Lecture 29 – Sequential Logic Circuits and Sequential Execution

# Last Lecture

- ✓ Combinational logic circuits
  - Made of many logic gates
  - Multi-functional combinational logic circuits such as ALU have control input lines to indicate which function to perform
  - Combinational logic circuits do not store (remember) values
- ✓ Sequential logic circuits
  - Made of combinational logic circuits, memory elements (e.g., clocked registers) and clock
  - Circuit that “remembers” values (state) and perform computations on these values

# 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

# About the word *register*


- **Warning:** the word *register* can mean different things!
  1. **In hardware**, a *register* is a hardware component directly connected to the rest of the circuit by its input and output lines (wires)
    - Called **hardware register** or **clocked register**
  2. **In machine-level programming**, a *register* is one of the addressable words in the CPU, i.e., in the register file (e.g., `%rdi`, `%rax`, etc...)
    - Called **program register**
    - Implemented using **hardware/clocked registers**

# Memory elements and Clock signals

- **Memory element #1** -> clocked registers a.k.a. hardware registers

- On the microprocessor
- A clocked register stores 1 bit (state)
- Synchronized by system-wide clock

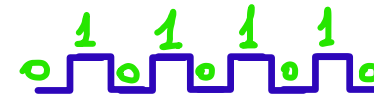
- System-wide clock

- A system-wide clock sends 0 1 0 1 0 1 0 1...
- Clock period: 1 full cycle duration: 
- Clock frequency: 1/period

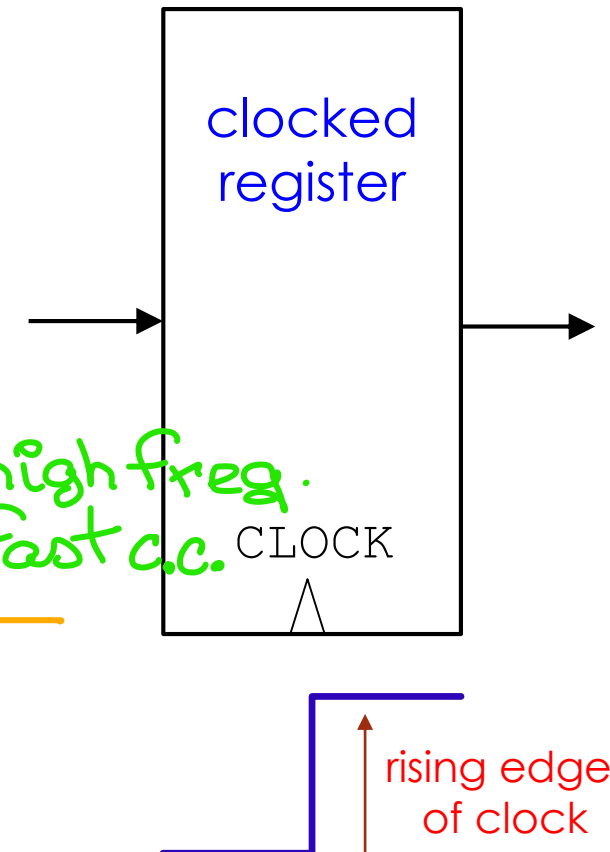
- How clocked registers work:

- Output current state
- Input next state
- Next state remembered only on rising edge of clock

not when clock signal is 1



→ high freq.  
small period → fast c.c.

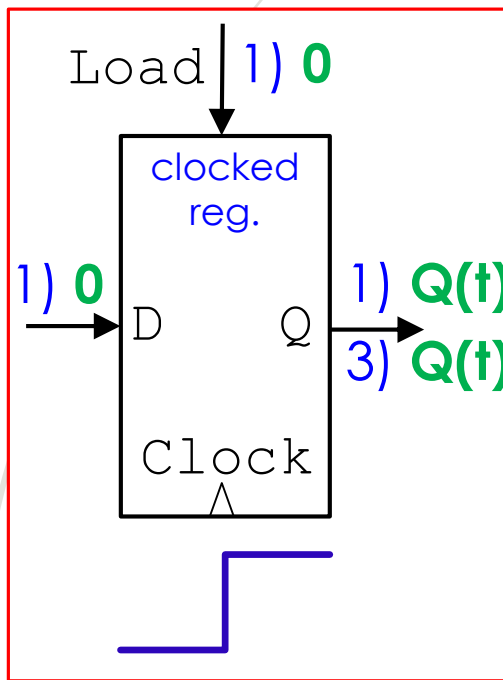


## Function Table

| Load | D | $Q(t + 1) \rightarrow$ next state |
|------|---|-----------------------------------|
| 0    | X | $Q(t) \rightarrow$ current state  |
| 1    | 0 | 0                                 |
| 1    | 1 | 1                                 |

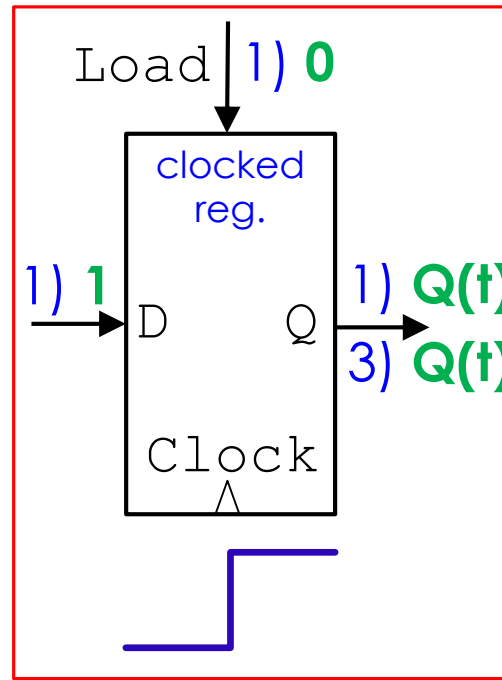
# How a *clocked register* function!

D flip flop



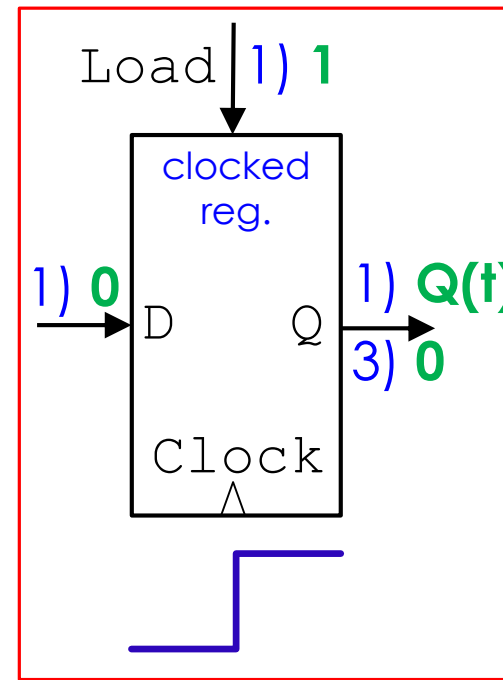
2) When the clock ticks, i.e., when edge of clock rises

Row 1 of Function Table



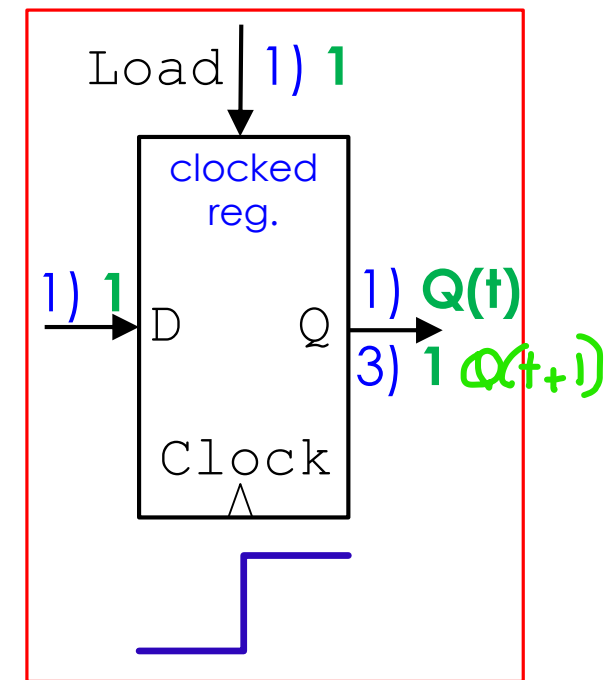
2) When the clock ticks, i.e., when edge of clock rises

Row 1 of Function Table



2) When the clock ticks, i.e., when edge of clock rises

Row 2 of Function Table



2) When the clock ticks, i.e., when edge of clock rises

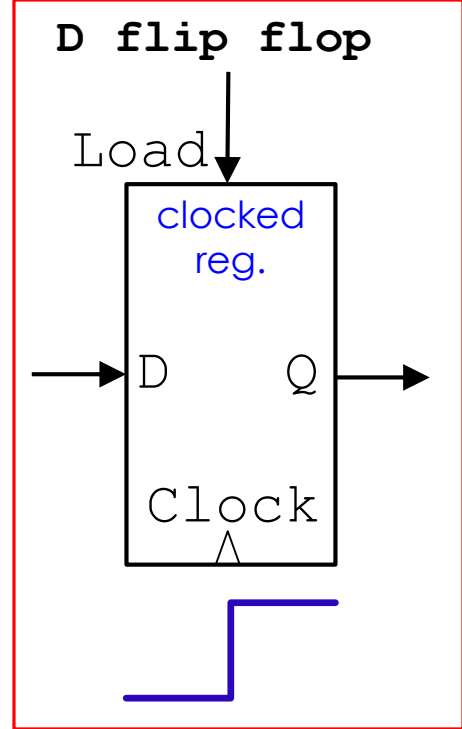
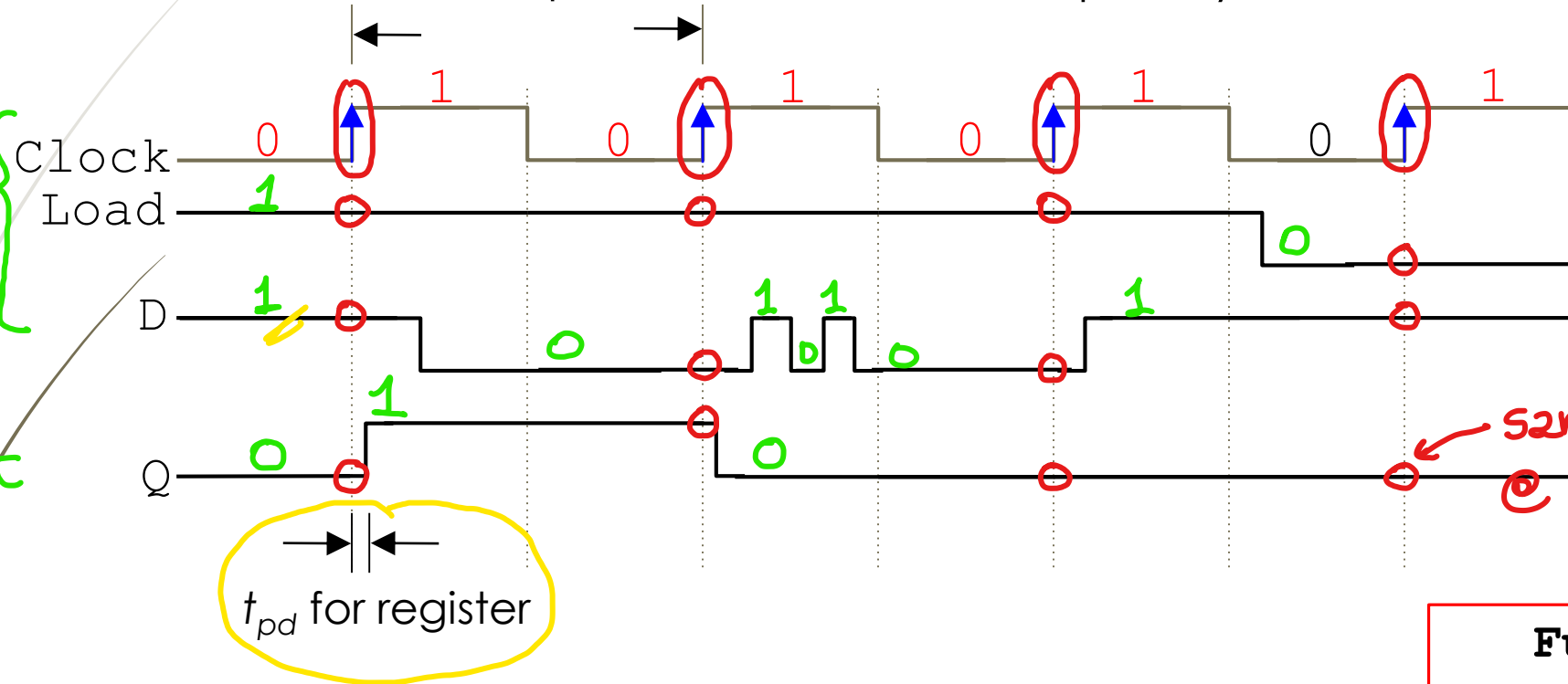
Row 3 of Function Table

# Timing Diagram

Edge-triggered  
clock

If clock period, then clock frequency is

input  
output



sampling happens  
@ these moments

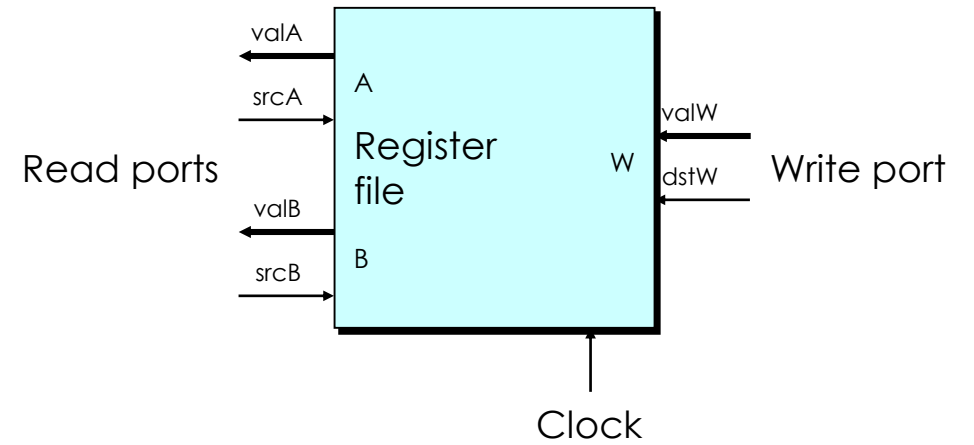
| Function Table |   |            |
|----------------|---|------------|
| Load           | D | $Q(t + 1)$ |
| 0              | X | $Q(t)$     |
| 1              | 0 | 0          |
| 1              | 1 | 1          |

## Memory elements #2 -> **Random access** memories

- Made of several clocked registers, i.e., store multiple **words** of data
- **Random access** (as opposed to sequential access)
  - Use an **address** to select which of the many memory words should be read or written
- Examples:
  1. RAM (see video)
  2. Register file -> register *id* used as address
    - Program registers
    - For example: x86-64 register file holds 16 registers

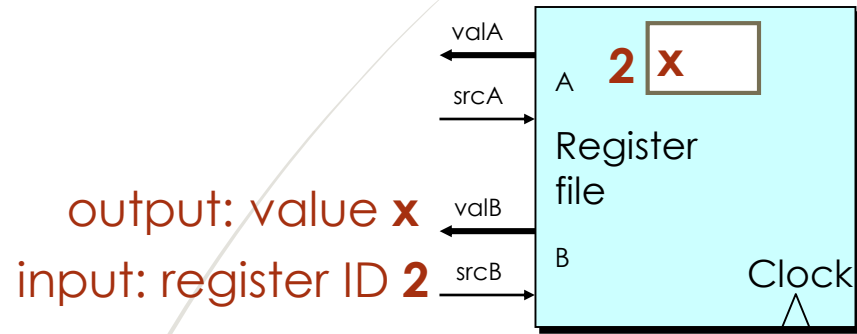


# Register File



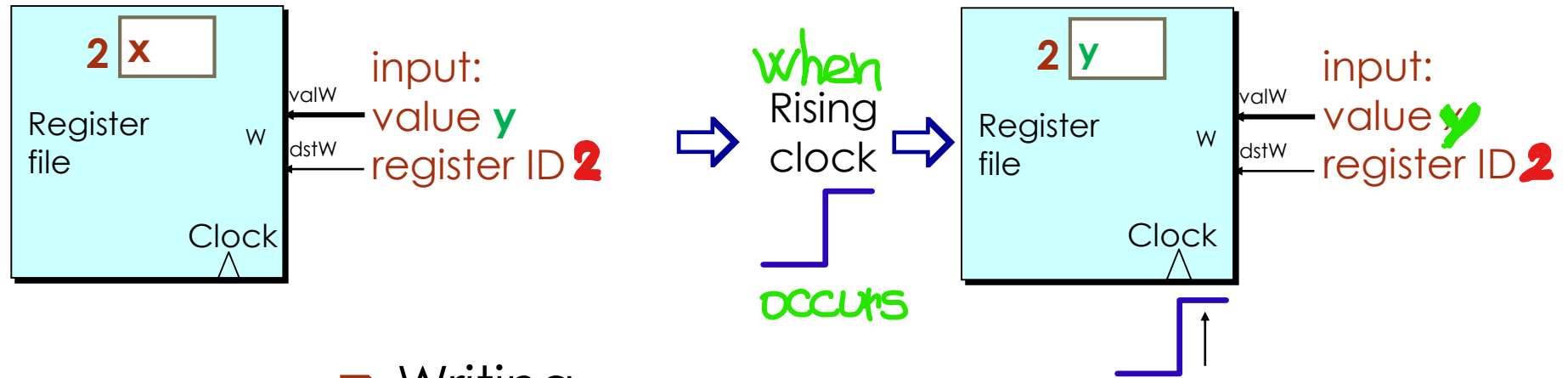
- Stores multiple words
- Each word is the value of a **program register**
  - `%rax`, `%rsp`, etc.
  - **Register ID** serves as address to specify which word to read or write
- Multiple ports
  - Can read and/or write multiple words in one clock cycle
    - Each port has separate address and data input/output

# Register file – how it works?



## ➤ Reading

- Combinational logic circuit – does not need clock
- Output data generated based on input address only, after some propagation delay

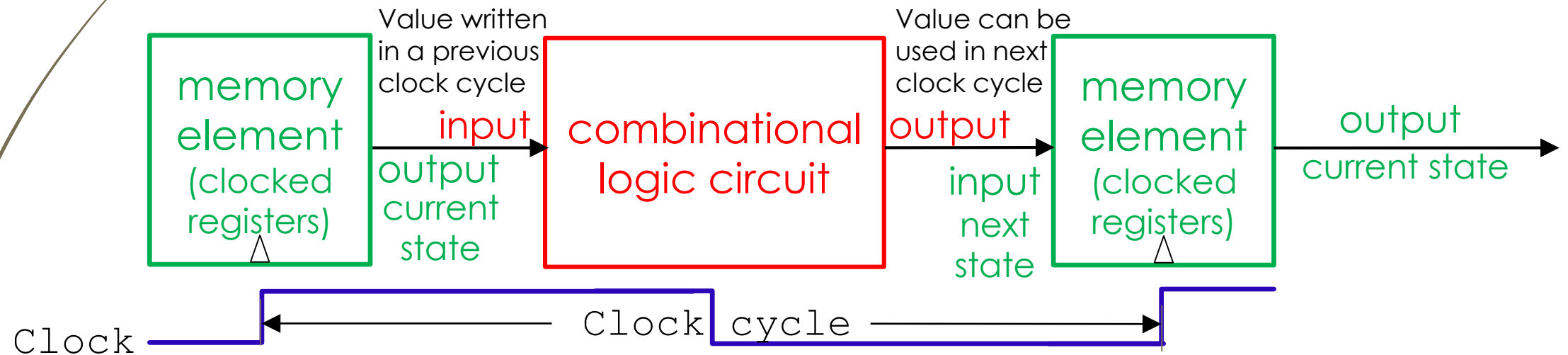


## ➤ Writing

- Sequential logic circuit -> clocked registers
- Remembers only when clock “ticks” (rising edge of clock)

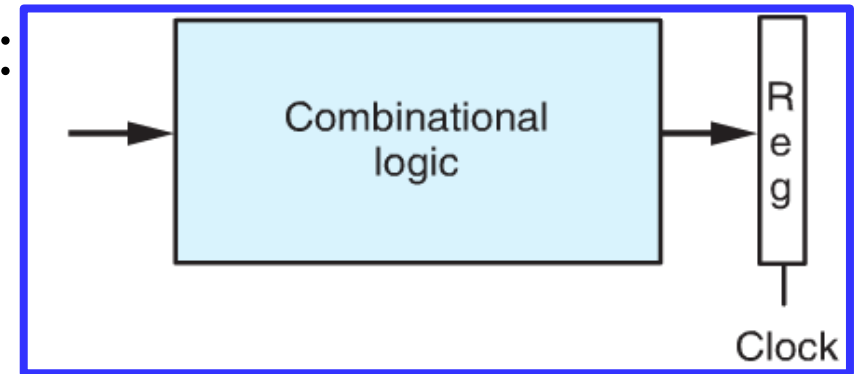
# Put together:

- Combinational logic circuits “deal” with an instruction
- Memory elements hold results (state), i.e., “remember”

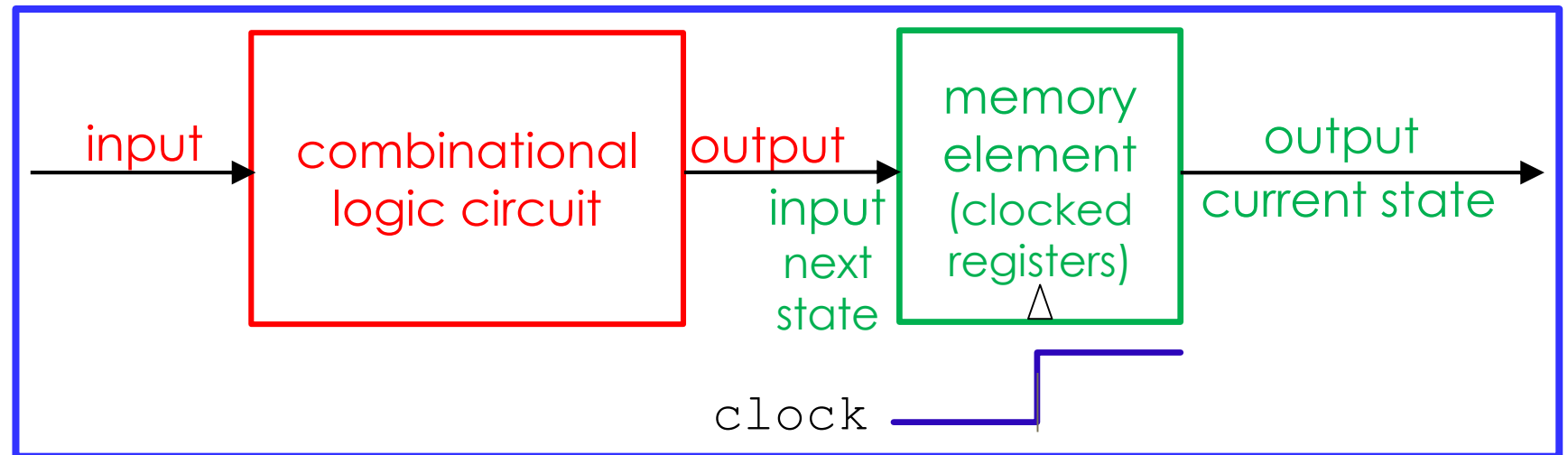


# Sequential execution

- Now, imagine we have designed and implemented a microprocessor (its datapath):



Called **sequential execution** because at every clock cycle, a new machine instruction is executed!



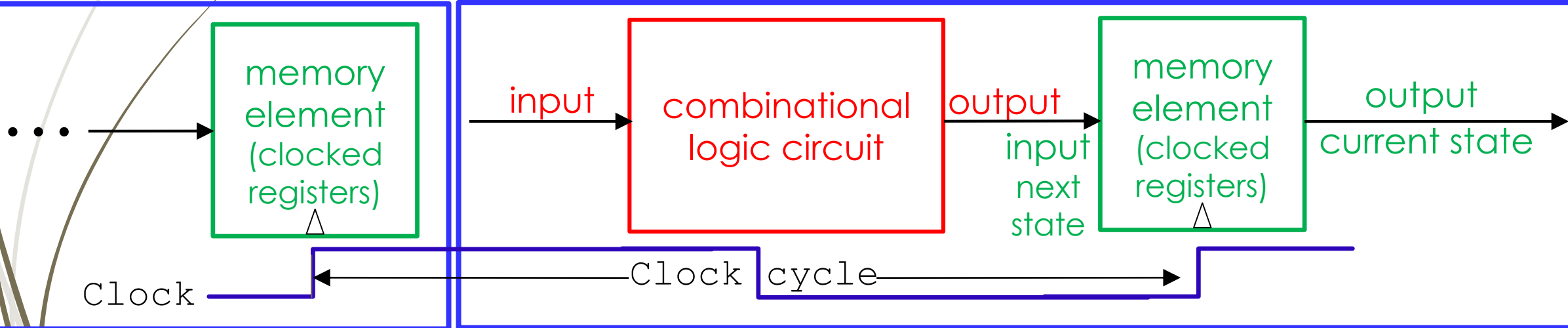
# Analysis of sequential execution

the time between two **ticks** of the clock

- **Clock cycle** required to execute 1 instruction must exceed  $t_{pd}$  of combinational logic circuit +  $t_{pd}$  of clocked register

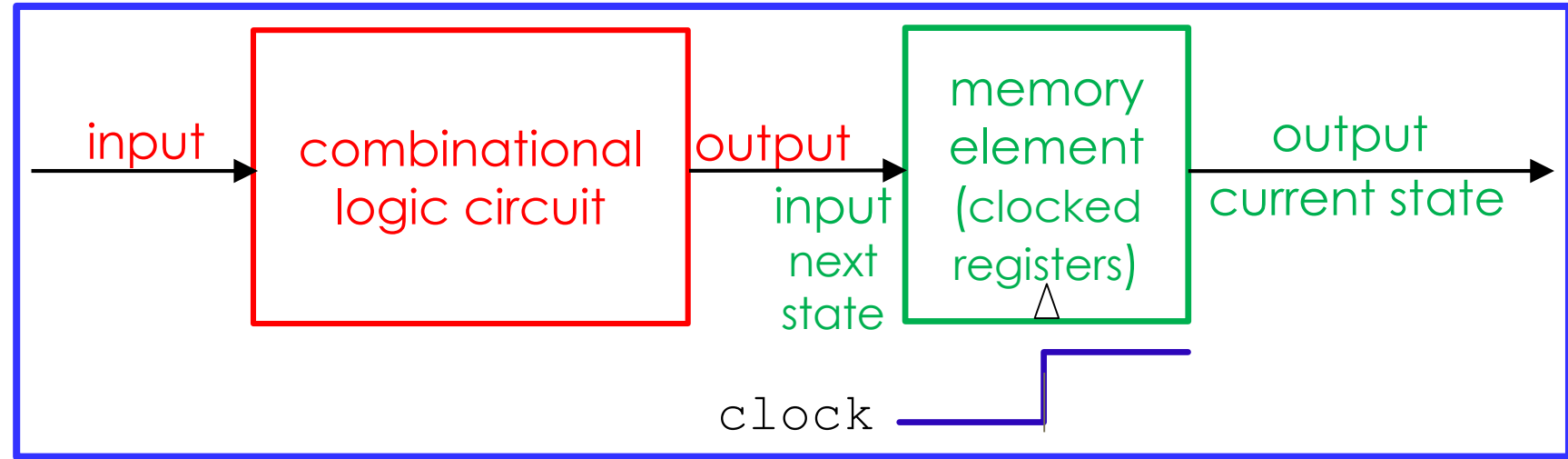
execute 1 instruction

execute 1 instruction



->  $t_{pd} + t_{pd}$  is long clock cycle!

# Analysis of sequential execution (cont'd)

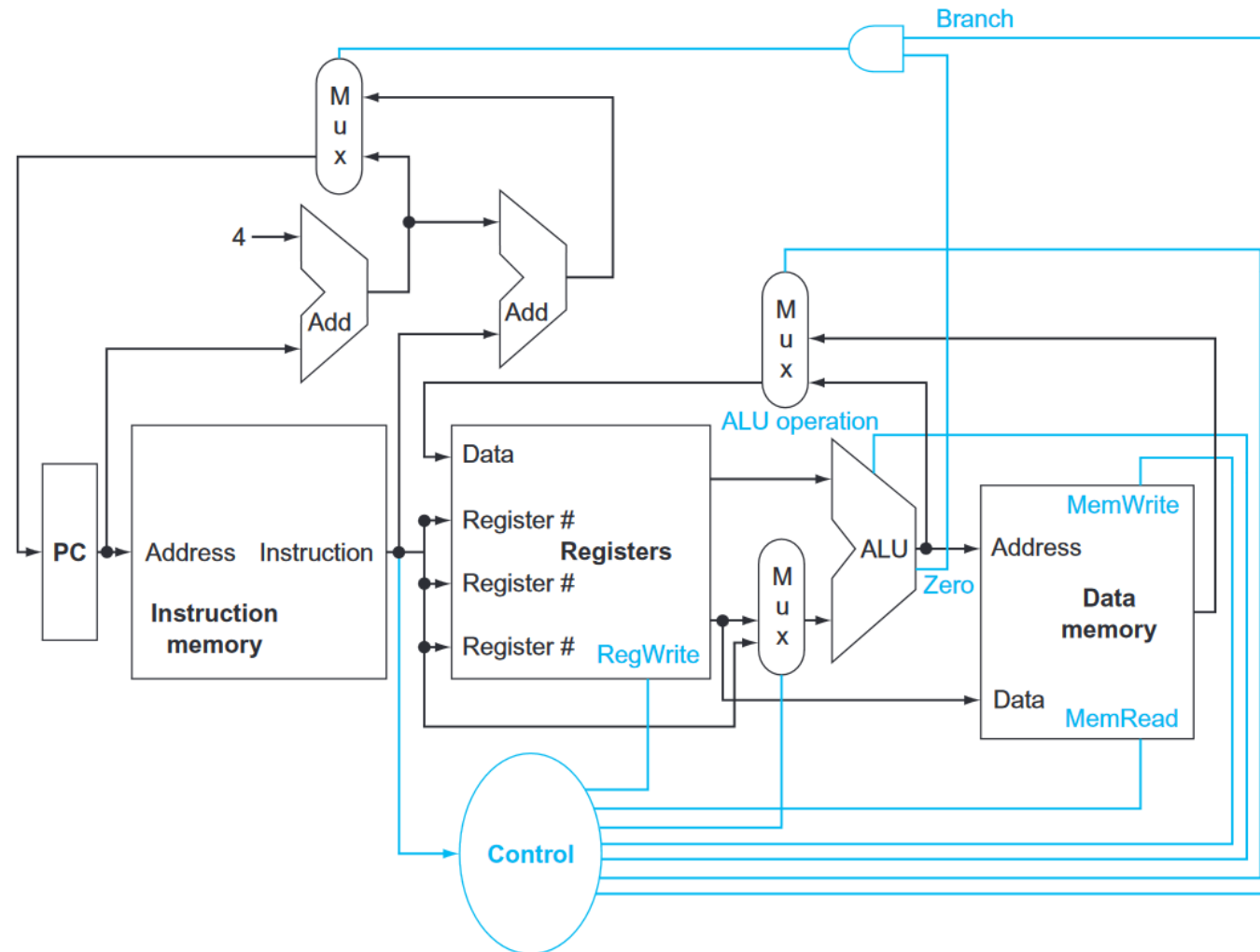


- Since the clock cycle is a factor in determining how fast the microprocessor executes machine instructions (i.e., execution speed)
- Then a microprocessor with a long clock cycle is a **slow microprocessor**
  - as it takes a long time to execute machine instructions, i.e., to execute a program

# Conclusion: sequential execution

- Such microprocessor called **single-cycle** microprocessor
- Execute each instruction in 1 clock cycle  $\rightarrow$   $CPI = 1$ 
  - **CPI  $\Rightarrow$  clock cycles per instruction**
    - Def: the average number of clock cycles per instruction for a program or program fragment
    - inverse of **instructions per cycle**
    - one aspect of a processor's performance
- CPI of 1 is inefficient:
  - Within the same clock cycle, *functional* units (of datapath like ALU) cannot be used more than once
    - So we would need to duplicate them if they are needed more than once during the execution of an instruction
  - Results in a long clock cycle and a low **throughput**
    - Def of throughput: Number of instructions executed per second

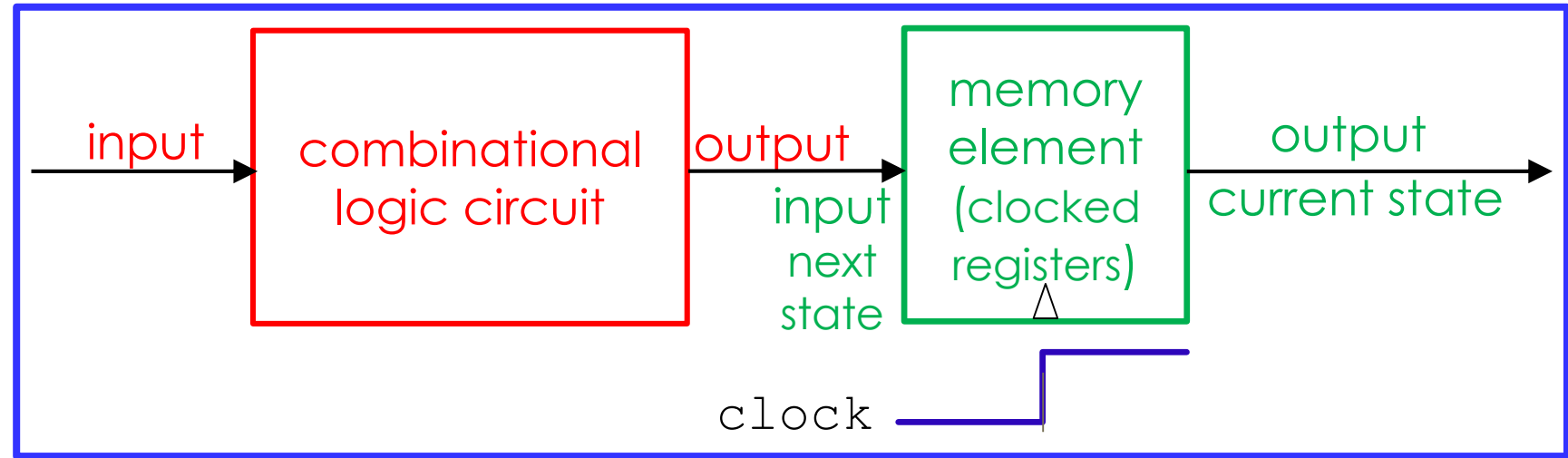
# Example of a single-cycle microprocessor (datapath)



**FIGURE 4.2** The basic implementation of the MIPS subset, including the necessary multiplexors and control lines.



# Analysis:



1. **Latency** (same as propagation delay):
  - Def: Time required to execute a single instruction
  - Example:
2. (instruction) **Throughput** -> speed of microprocessor:
  - Def: Number of instructions executed per second - GIPS
  - Example:

# Summary

- 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

# 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