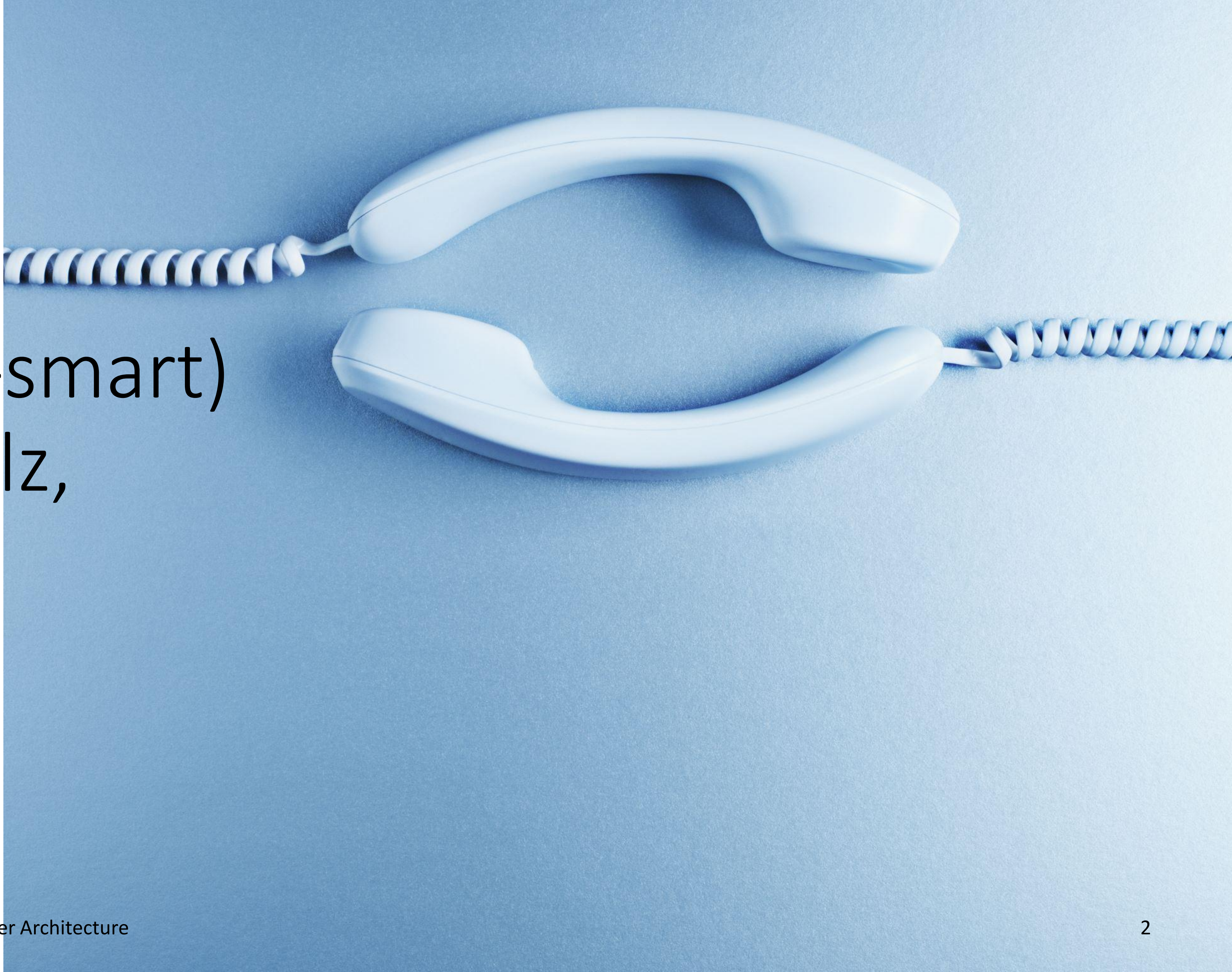
CS230: Digital Logic Design and Computer Architecture

Lecture 5: Intro to ISA and instructions

https://www.cse.iitb.ac.in/~biswa/courses/CS230/main.html

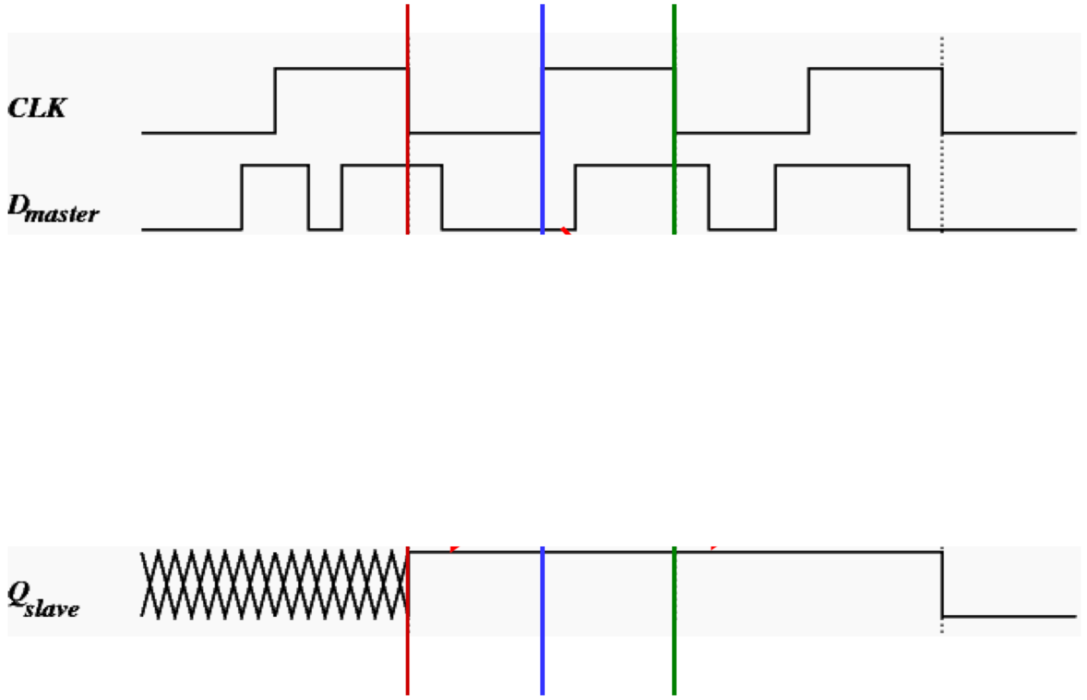
https://www.cse.iitb.ac.in/~biswa/

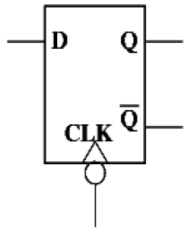


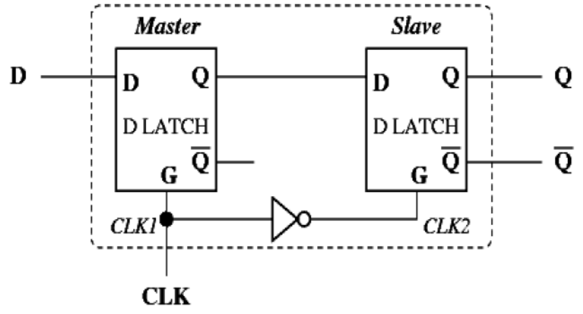
Phones (smart/non-smart) on silence plz, Thanks

Recap of D Flip-flop

- **Example:** When CLK is high, output of master is allowed to change with D; when CLK is low (falling edge), the output of the master is fixed and propagated through to the output of the slave ⇨ this flip-flop triggers on *falling* or *negative edge*.





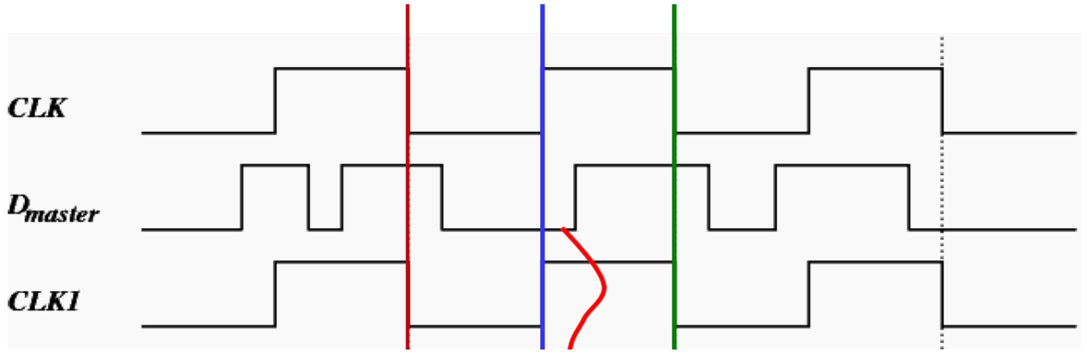


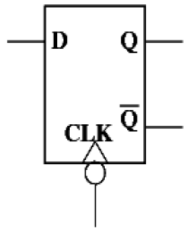
CLK	D	Q*	$\overline{Q*}$
↴	0	0	1
↴	1	1	0

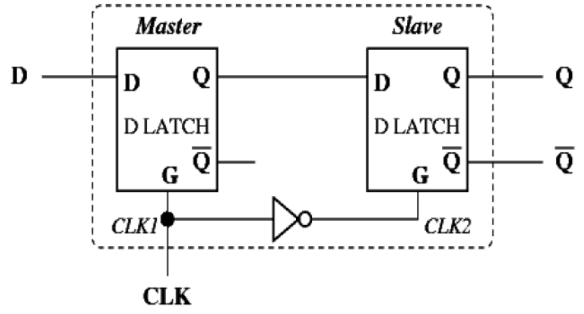
Characteristic Table

Recap of D Flip-flop

- **Example:** When CLK is high, output of master is allowed to change with D; when CLK is low (falling edge), the output of the master is fixed and propagated through to the output of the slave ⇨ this flip-flop triggers on *falling* or *negative edge*.





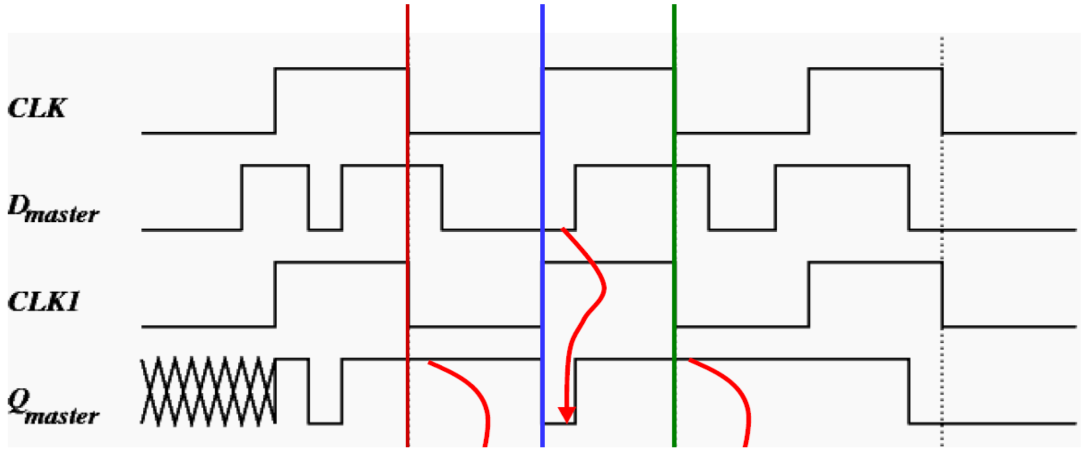


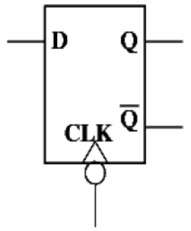
CLK	D	Q*	$\overline{Q}$*
↴	0	0	1
↴	1	1	0

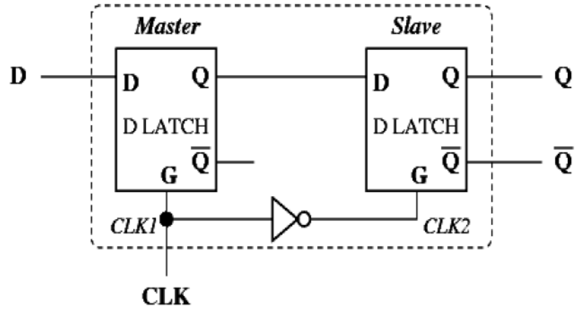
Characteristic Table

Recap of D Flip-flop

- **Example:** When CLK is high, output of master is allowed to change with D; when CLK is low (falling edge), the output of the master is fixed and propagated through to the output of the slave ⇨ this flip-flop triggers on *falling* or *negative edge*.





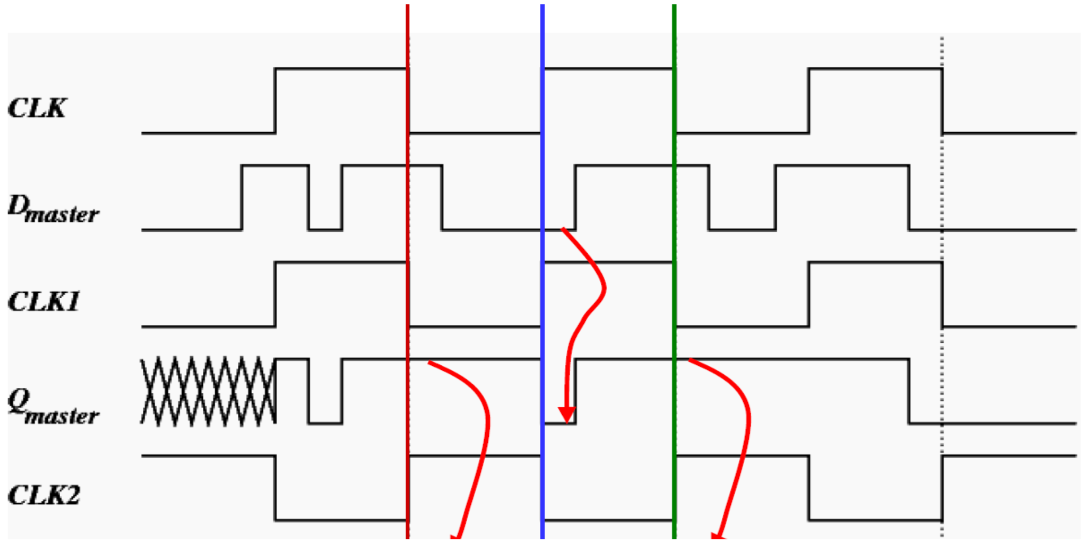


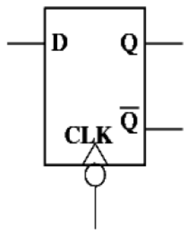
CLK	D	Q*	$\overline{Q}$*
↴	0	0	1
↴	1	1	0

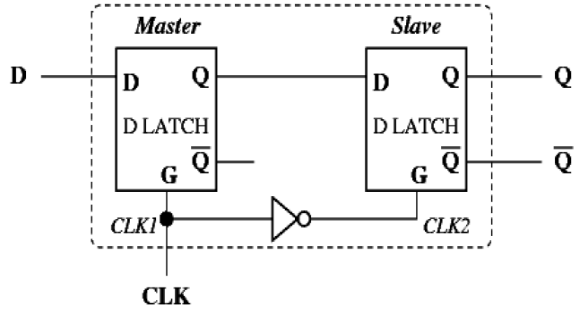
Characteristic Table

Recap of D Flip-flop

- **Example:** When CLK is high, output of master is allowed to change with D; when CLK is low (falling edge), the output of the master is fixed and propagated through to the output of the slave ⇨ this flip-flop triggers on *falling* or *negative edge*.





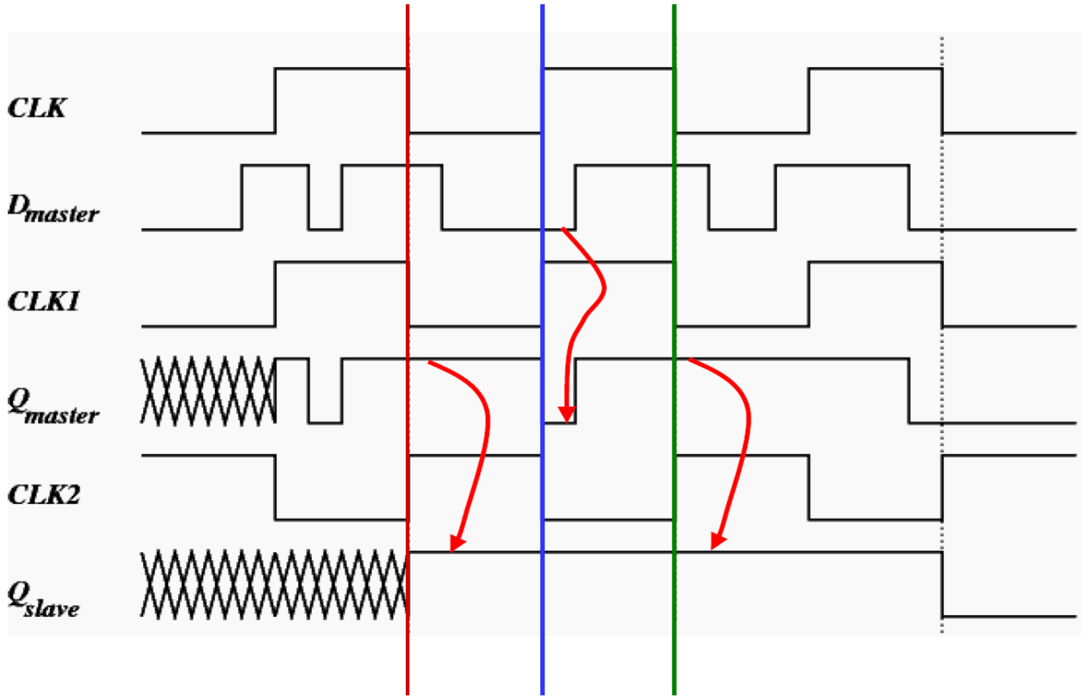


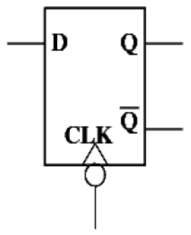
CLK	D	Q*	$\overline{Q}$*
↴	0	0	1
↴	1	1	0

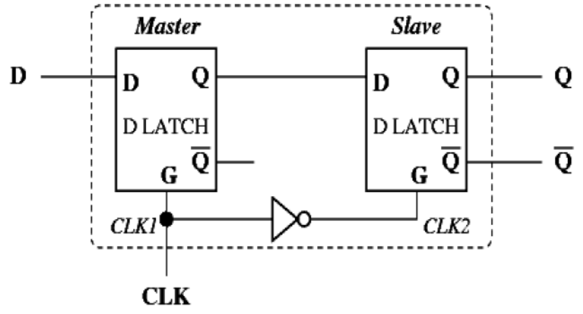
Characteristic Table

Recap of D Flip-flop

- **Example:** When CLK is high, output of master is allowed to change with D; when CLK is low (falling edge), the output of the master is fixed and propagated through to the output of the slave ⇨ this flip-flop triggers on *falling* or *negative edge*.







CLK	D	Q*	$\overline{Q}$*
↴	0	0	1
↴	1	1	0

Characteristic Table

Delay to make sure all is well

- **Setup time**, t_{su}, is the time period prior to the clock becoming active (edge or level) during which the flip-flop inputs must remain stable.

- **Hold time**, t_h, is the time after the clock becomes inactive during which the flip-flop inputs must remain stable.

- Setup time and hold time define a *window of time during which the flip-flop inputs cannot change* – quiescent interval.

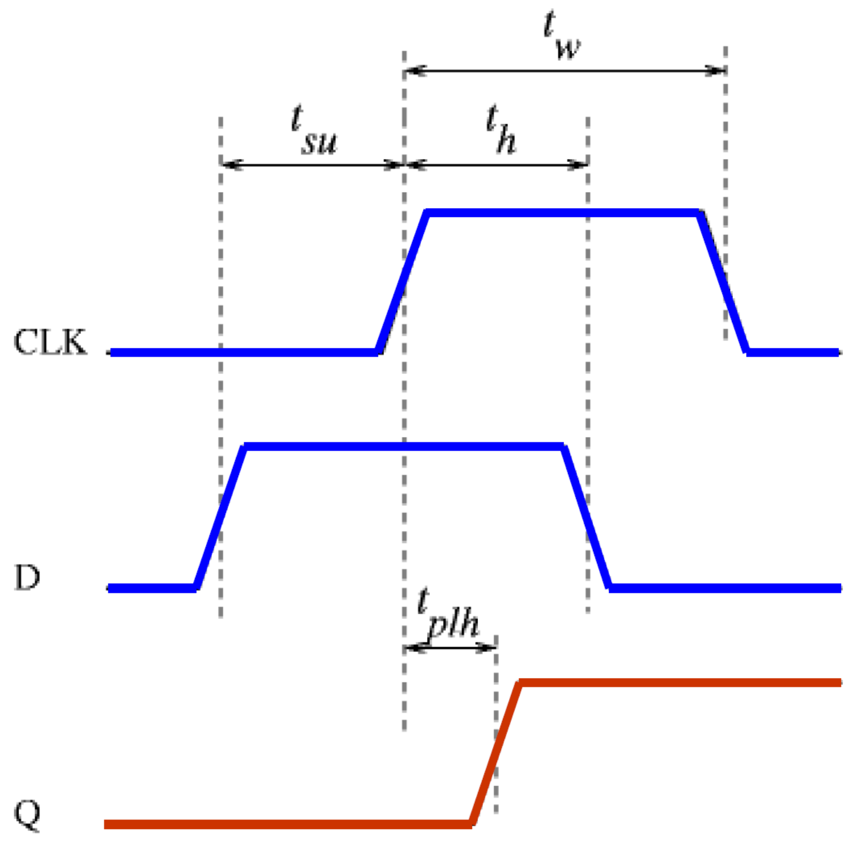
More Delay

- **Propagation delay,** t_{pHL} and t_{pLH} , has the same meaning as in combinational circuit – beware propagation delays usually will not be equal for all input to output pairs. There can be two propagation delays: $t_{C\text{-}Q}$ (*clock*→Q delay) and $t_{D\text{-}Q}$ (*data*→Q delay).

- For a level or pulse triggered latch:
  - Data input should remain stable till the clock becomes inactive.
  - Clock should remain active till the input change is propagated to Q output. That is, active period of the clock,

$$t_w > \max\{t_{pLH}, t_{pHL}\}$$

All in One



D Flip–flop (edge–triggered)
(positive edge triggering)

World of State machines (FSMs) Moore and Mealy Machines

Computer Architecture

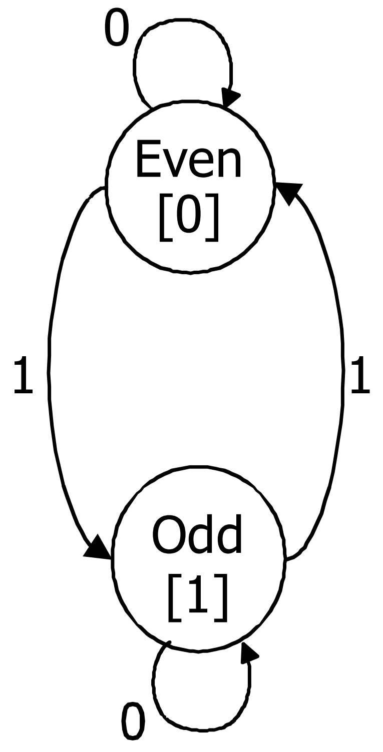
Moore machine: Output depends on the current state

Mealy machine: Output depends on the current state and inputs

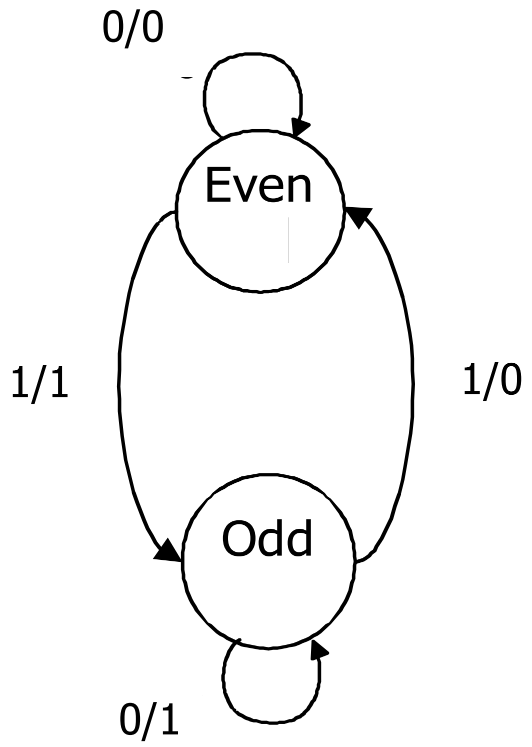
Odd Parity Checker

- Serial input string
    - OUT=1 if odd # of 1s in input
    - OUT=0 if even # of 1s in input
- Let's do this for Moore and Mealy

Moore



Mealy



State Transitions

Output changes only when the state changes

Appears after the state transition takes place

outputs change at clock edge

Even = 0

Odd = 1

Moore

Present State	Input	Next State	Present Output
Even	0	Even	0
Even	1	Odd	0
Odd	0	Odd	1
Odd	1	Even	1

Output changes when the state and input changes

Appears before the state transition is completed

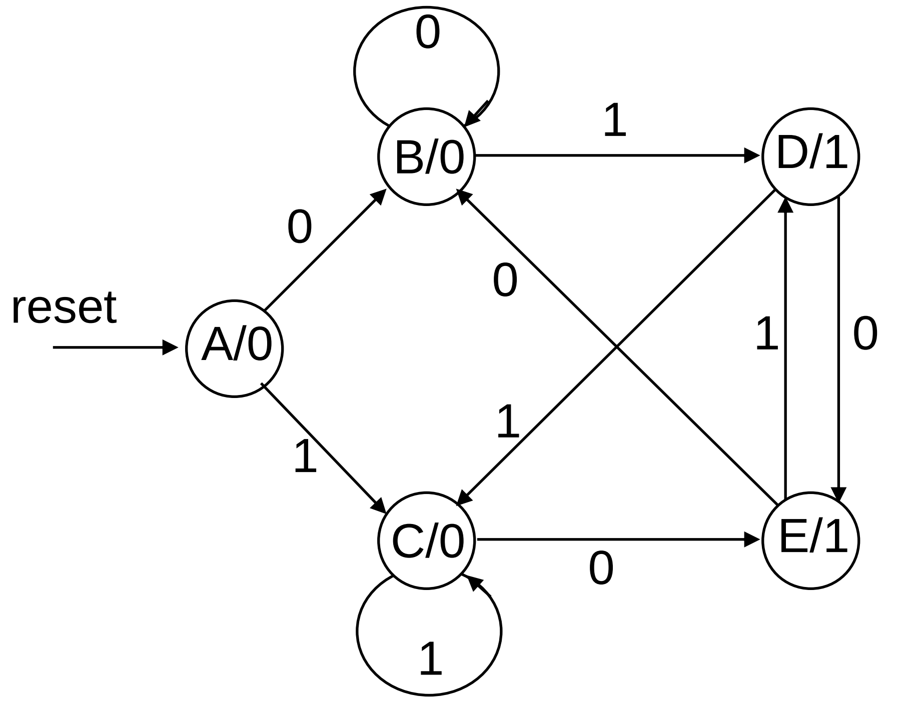
React faster to inputs — don't wait for clock

Mealy

Present State	Input	Next State	Present Output
Even	0	Even	0
Even	1	Odd	1
Odd	0	Odd	1
Odd	1	Even	0

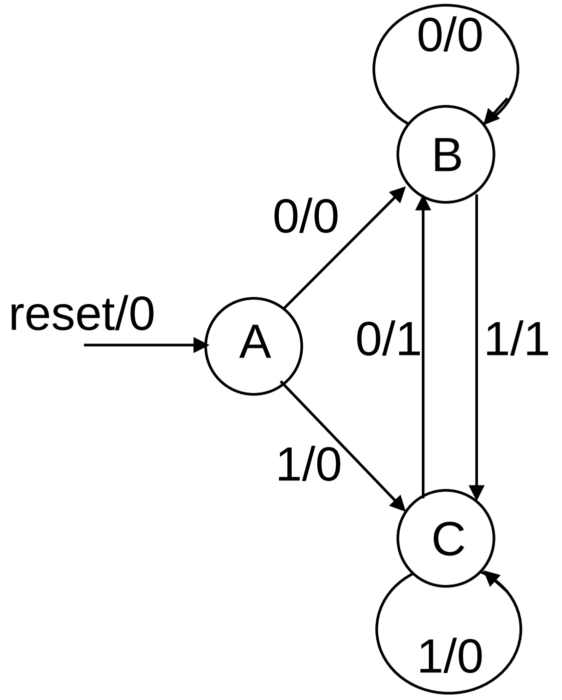
Try on your own

01/10 detector: Moore Machine



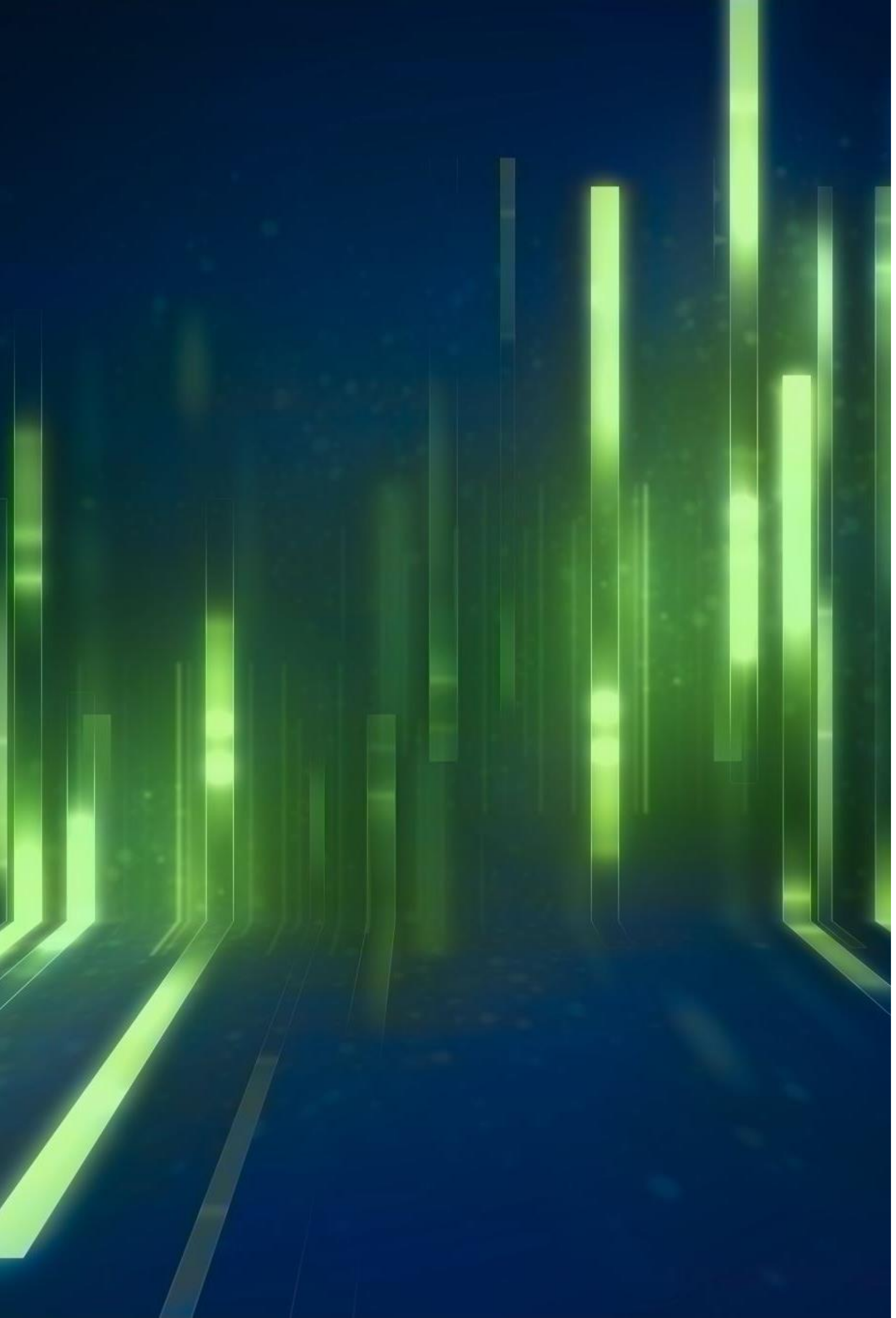
reset	input	current state	next state	current output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	D	0
0	0	C	E	0
0	1	C	C	0
0	0	D	E	1
0	1	D	C	1
0	0	E	B	1
0	1	E	D	1

01/10 detector: Mealy Machine



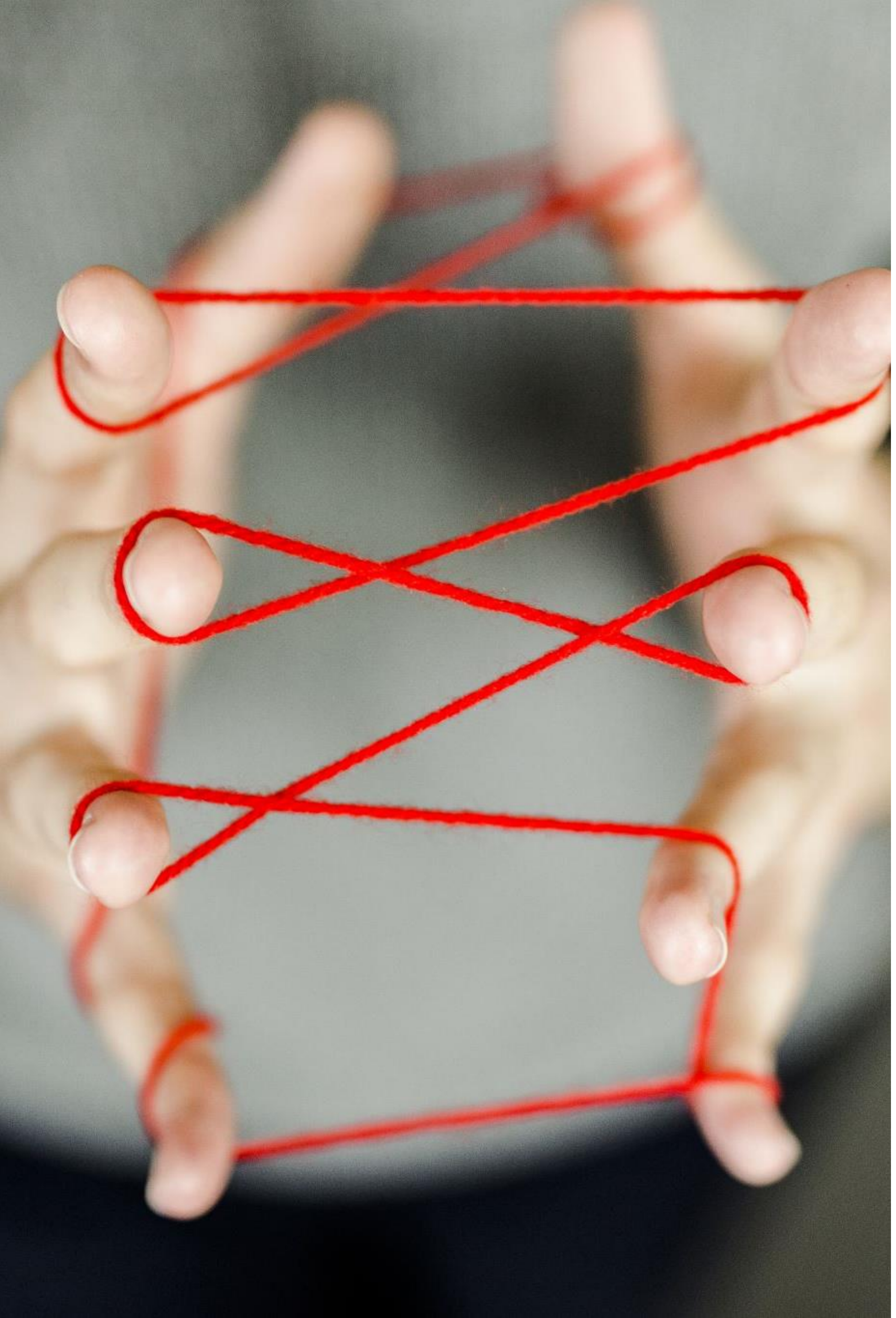
reset	input	current state	next state	current output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	C	1
0	0	C	B	1
0	1	C	C	0

Architecture-101



Next Few Lectures

- HOW CAN A PROGRAMMER INTERACT WITH THE PROCESSOR?
- THE LANGUAGE OF COMPUTER: INSTRUCTIONS
- INSTRUCTIONS HAVE A VOCABULARY CALLED INSTRUCTION SET
- DRIVEN BY INSTRUCTION SET ARCHITECTURE (ISA)
- ISA: X86, ARM, RISC-V, MIPS

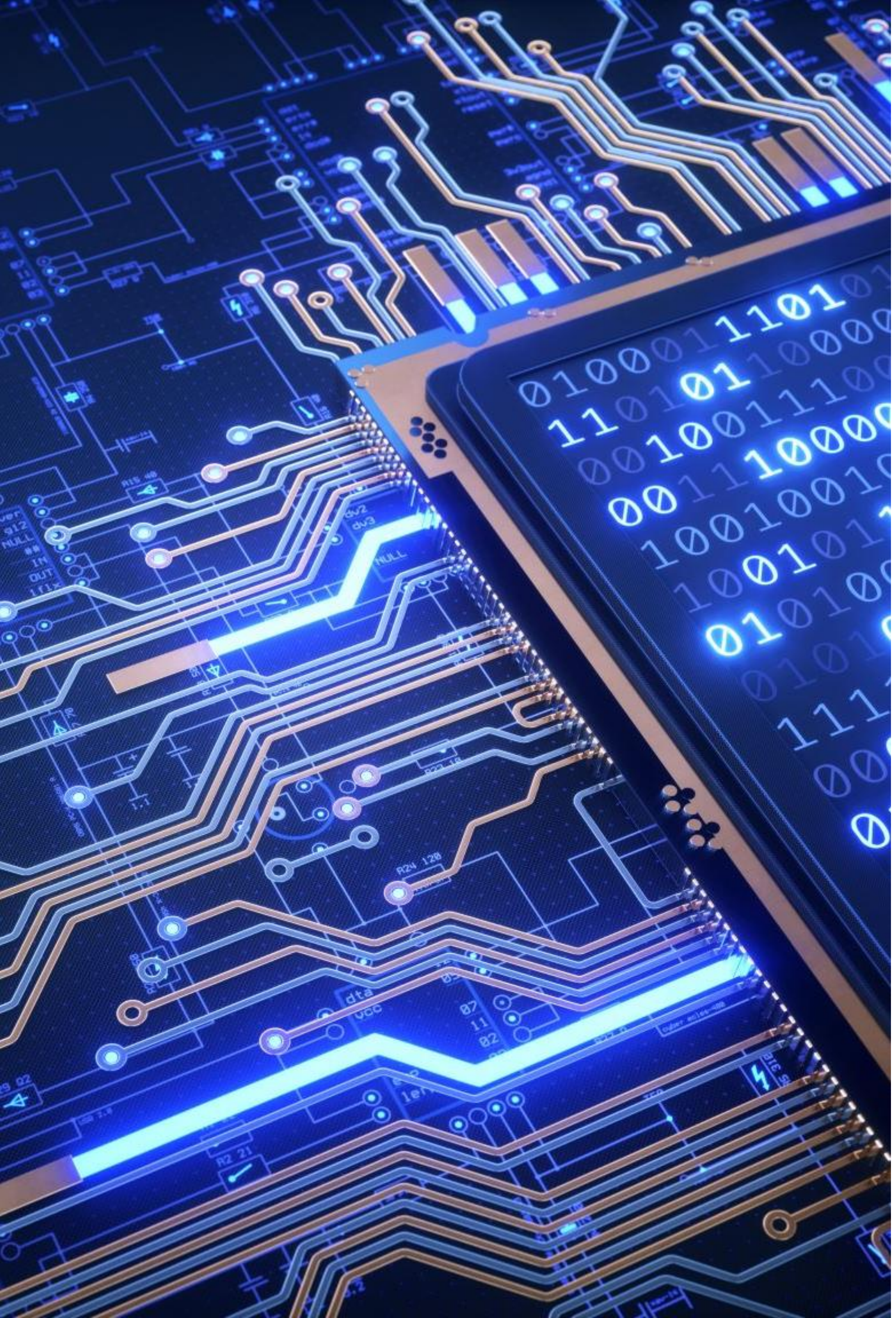


Why MIPS?

Simple yet expressive

Basic principles are similar if not the same. e.g., ARM ISA

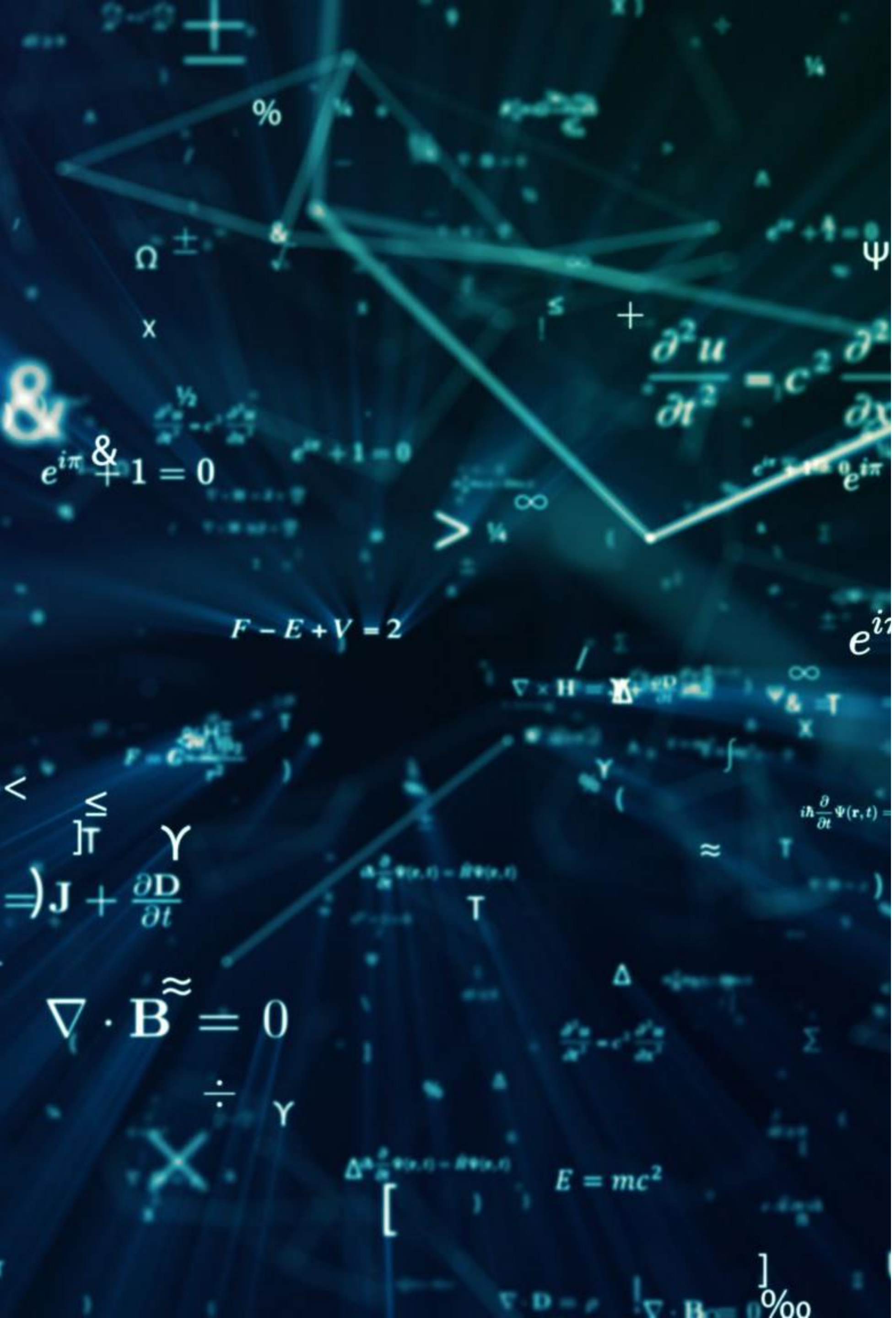
Still in use today: embedded devices, routers, modems etc.



ISA: Abstraction layer

Interface between hardware and software

hides complexity from the software through a set of simple instructions



Abstraction Example: 101

a = b + c ; // C code

compiler

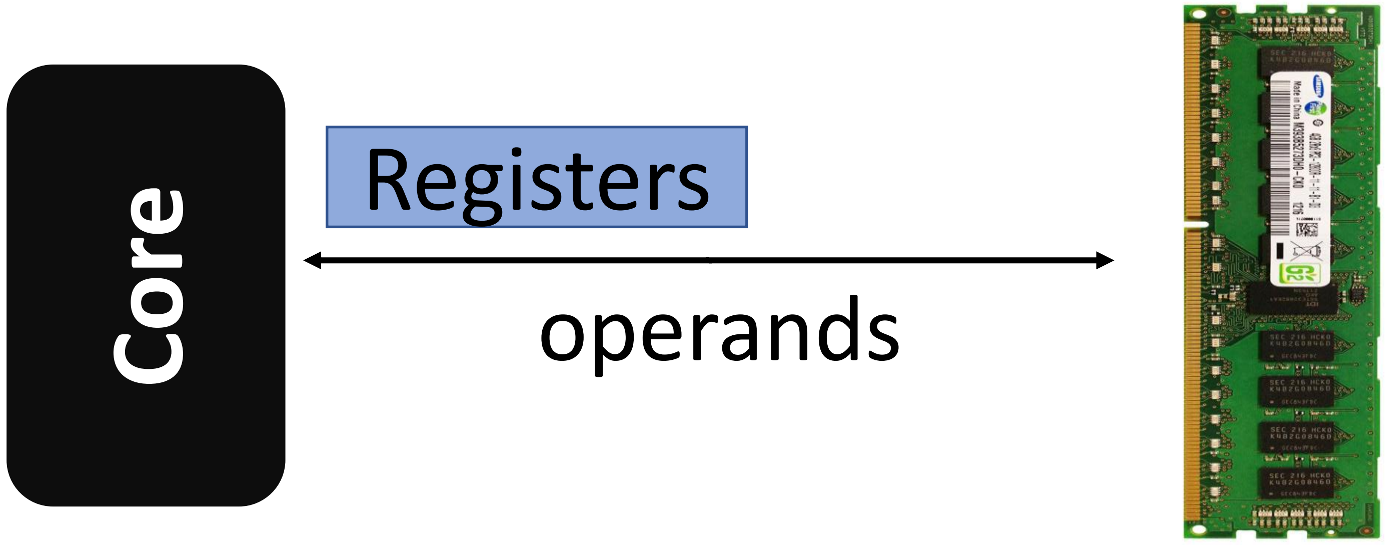
add \$1, \$2, \$3 // assembly language as per the ISA

assembler

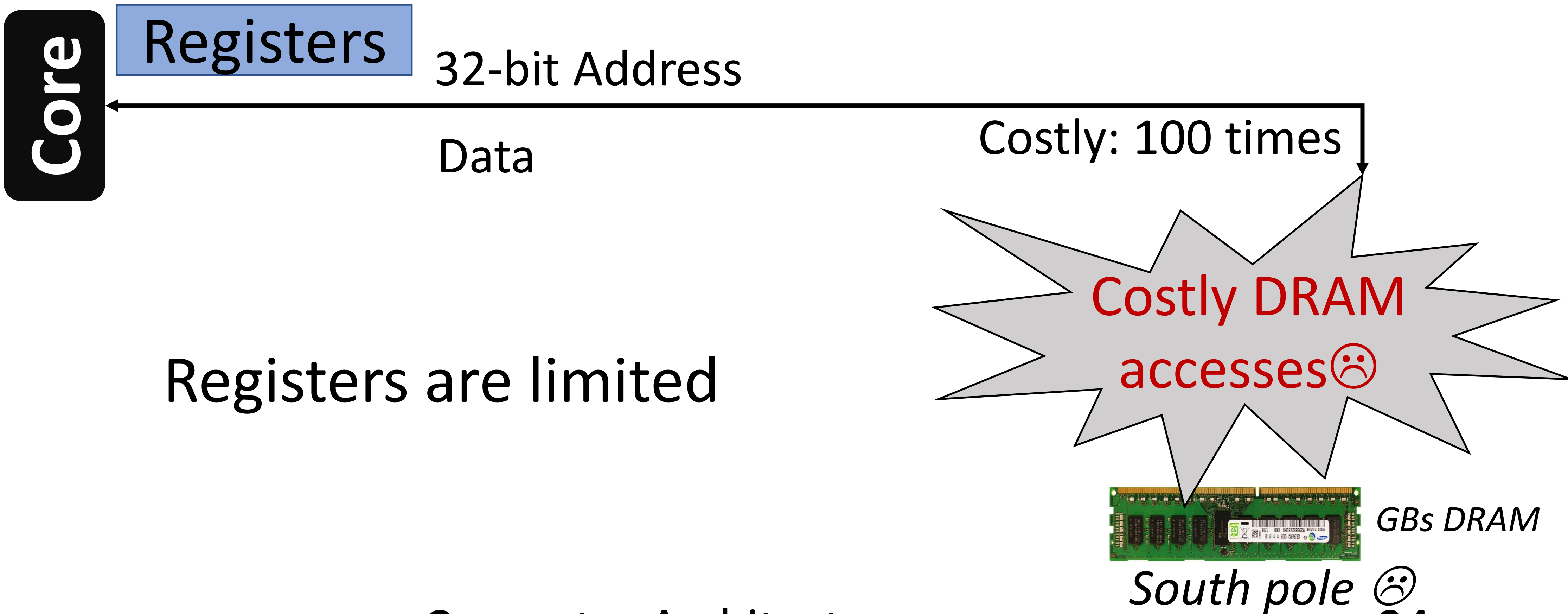
010101010101010 // machine language, 0s and 1s

Abstraction Example: 101

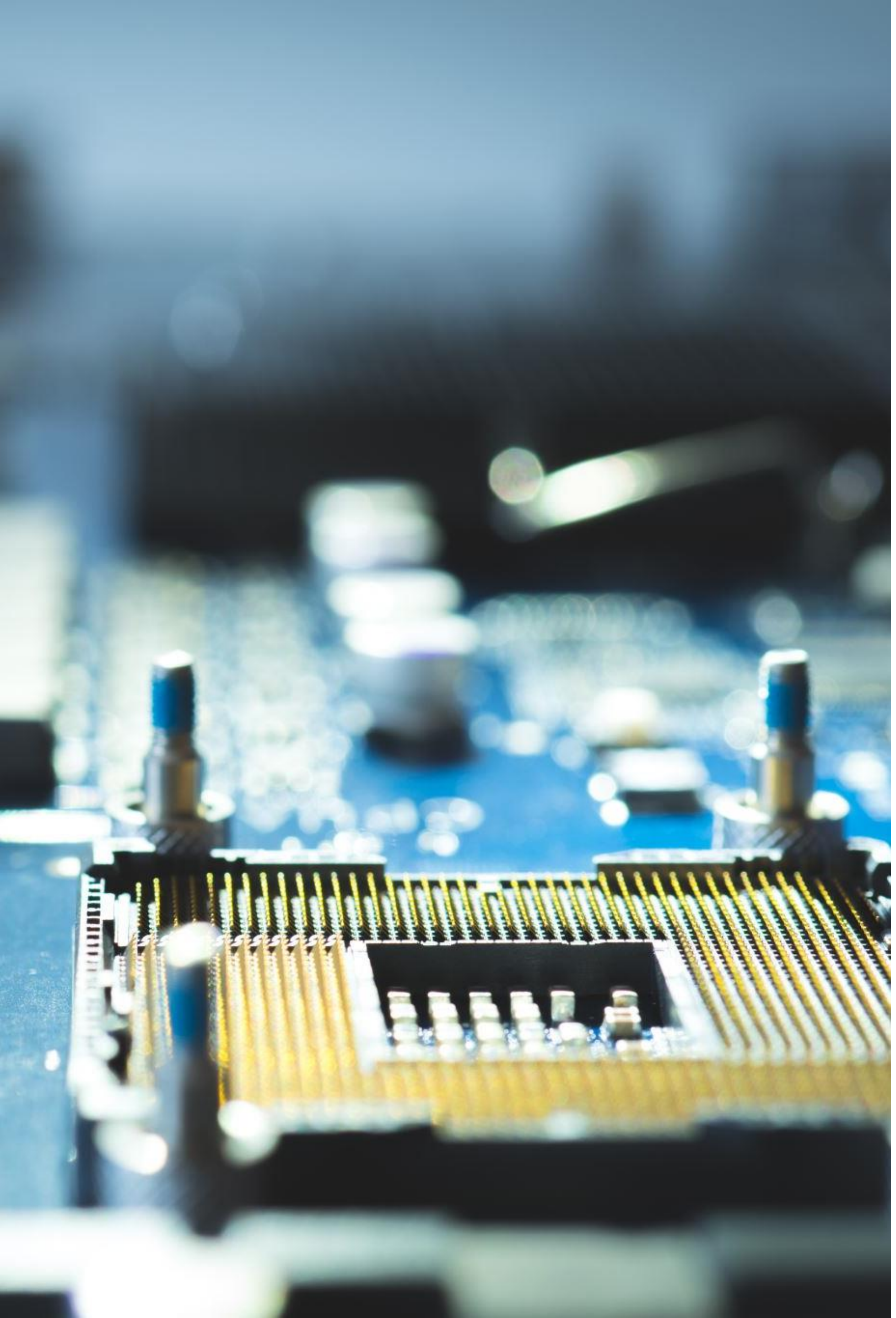
Operands can be in registers or in memory



A bit detailed



Registers are limited



Instructions

Programmers’ order/command to the processor

Why Instructions?

Programmer knows what it can/cannot
Processor knows what it should

Power of abstraction:
World with no instructions:
Programmers – communicate a sequence of 0s and 1s

World with no instructions

```
000000 00000 00000 00010 00000 100101
000000 00000 00101 01000 00000 101010
000100 01000 00000 00000 00000 000011
000000 00010 00100 00010 00000 100000
001000 00101 00101 11111 11111 111111
000010 00000 10000 00000 00000 000001
```

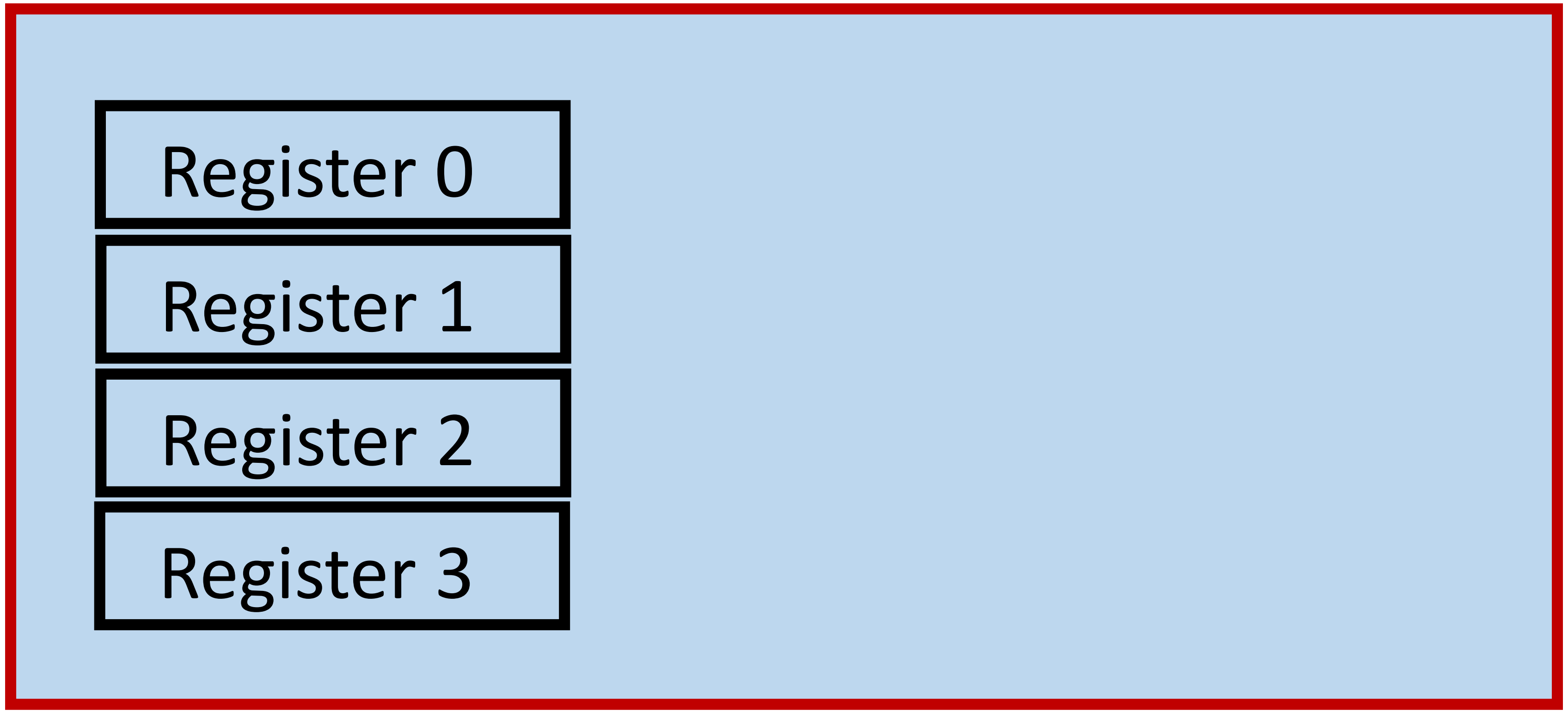
World of 18 instructions

A n Add the number in storage location n into the accumulator.
E n If the number in the accumulator is greater than or equal to zero execute next the order which stands in storage location n; otherwise proceed serially.
Z Stop the machine and ring the warning bell.

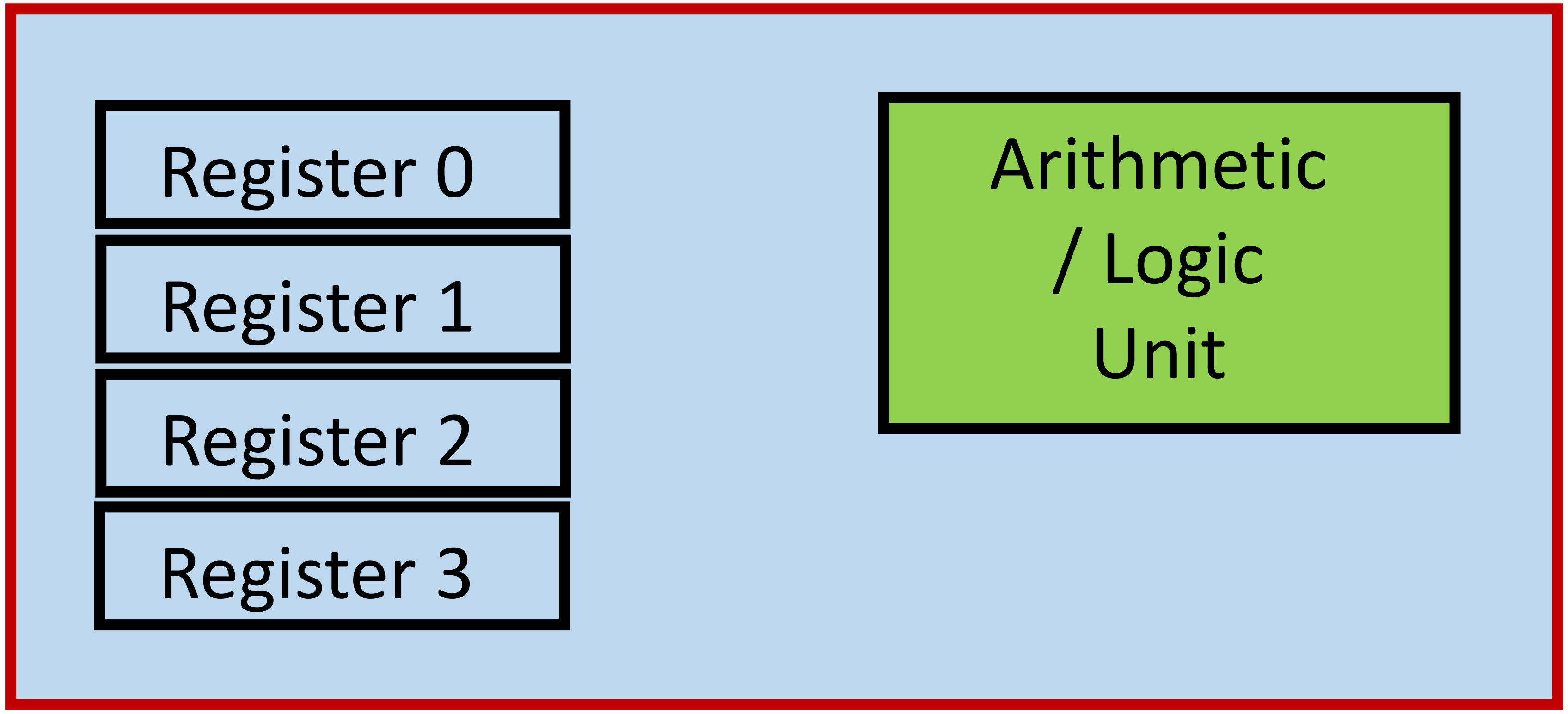
Wilkes and Renwick Selection from the List of 18 Machine Instructions for the EDSAC (1949)

2023: How many x86 instructions?

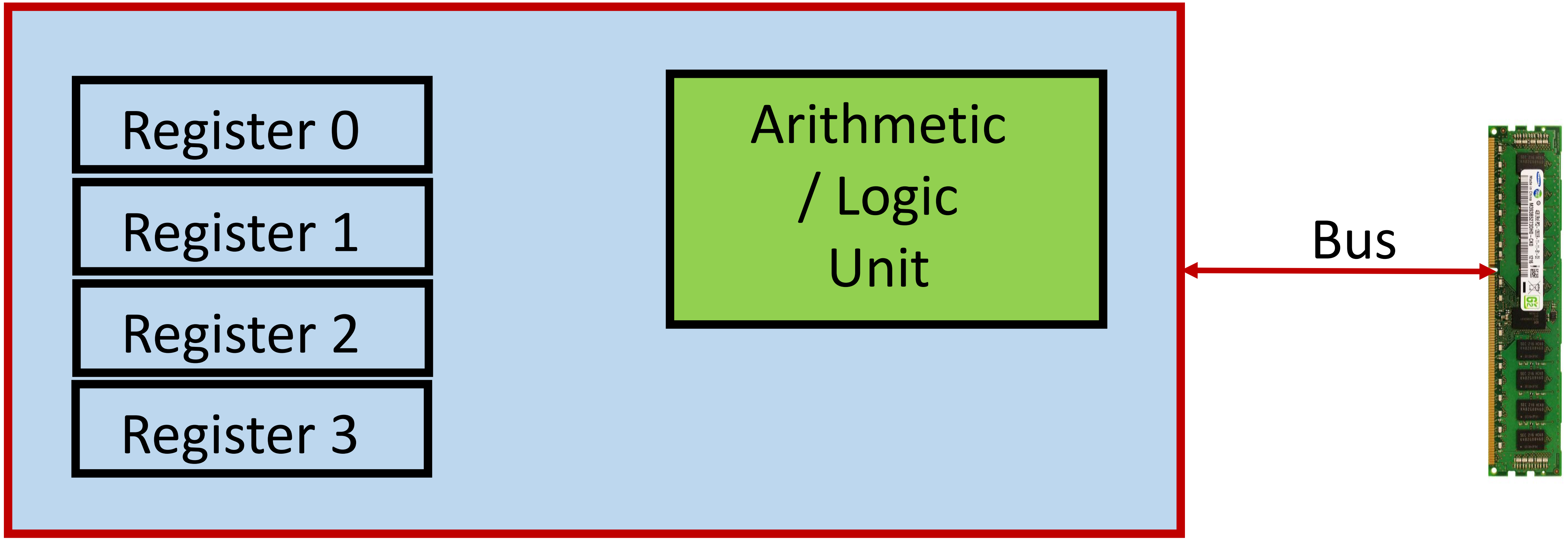
Let’s Open the Processor Core



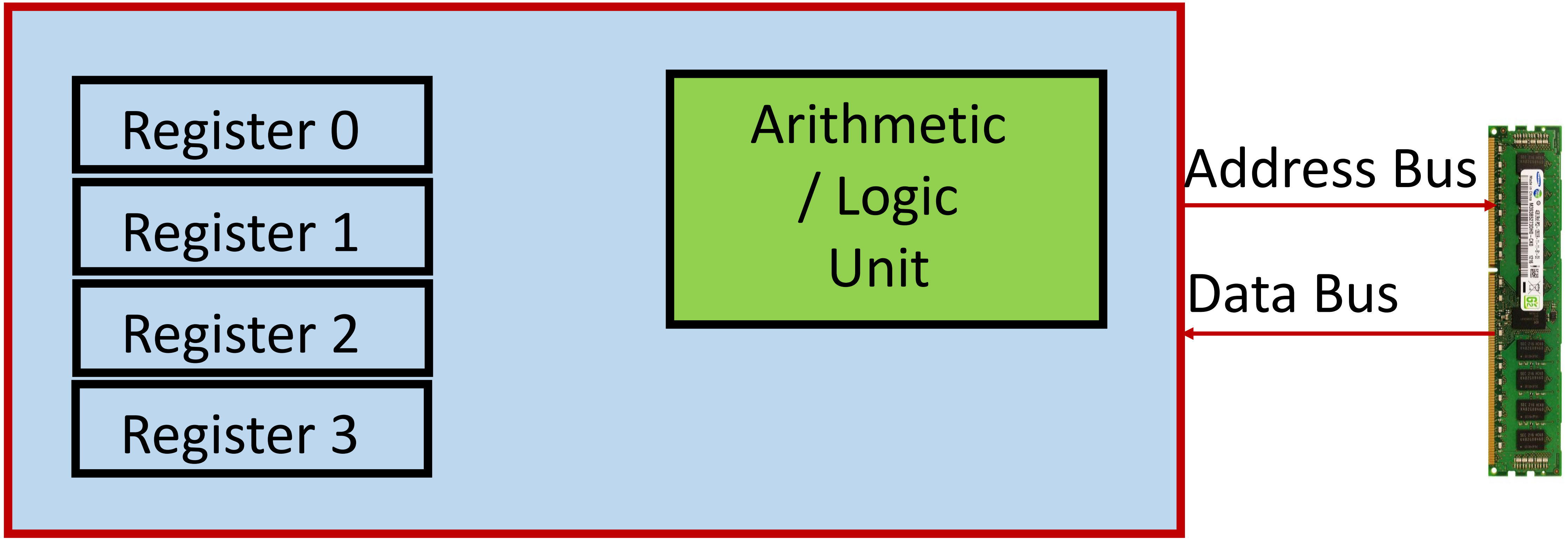
Let’s Open the Processor Core



Let’s put the Memory (not inside the core)



Let’s put the Memory (not inside the core)



MIPS Instructions: 101

add $0, $1, $2

add: operation, $0: Destination, $1 & $2: Source(s)

Most of the arithmetic/logical: two sources and one destination

What to do for “a=b+c-d”?

What to do for “a=b+c-d”?

```
add $t0, $s1, $s2      #$t = b+c
sub $s0, $t0, $s3       #$s = $t-d
```

Temporary register

Try out:

f=(g+h) – (i+j)

Constants and Immediate

x=x+10

$\searrow$

No need of a register

addi $s0, $s0, 10

i: immediate, for constants, constant: 2s complement

Constants and Immediate

x=x+10

No need of a register

addi $s0, $s0, 10

Do we need a subi ? ☺

i: immediate, for constants

constant: **16 bits**, 2s complement form

Constants and Immediate

x=x+10

No need of a register

addi $s0, $s0, 10

$\boxed{\text{Do we need a subi ? ☺}}$ $\boxed{\text{NO}}$

i: immediate, for constants, constant: 2s complement form

Special treatment for zero

$0 or $zero is a special register that contains ZERO

Why add if we can move?

a=b   becomes add $s1 $s2 $zero

Pseudo Instruction 101

a=b

```
move $S0, $s1
```

Not an actual instruction.
It is used for programming convenience

Logical Operations

Bitwise operations and shifts (Refer Section 2.6 P&H)

sll, srl, and, or, nor, andi, ori etc

No not instruction ☺, well not is nor with one operand=0

32 raw bits instead of a 32-bit number.

How to store a 32-bit constant into a 32-bit register? Remember 16-bit ☺

For example, 10101010 10101010 11110000 11110000

Trivia? How to store a 32-bit constant into a 32-bit register?

For example, 10101010 10101010 11110000 11110000

lui $t0, 0xAAAA #1010101010101010, lower bits all 0s.

ori $t0, $t0, 0xF0F0 #1111000011110000

Trivia? How to store a 32-bit constant into a 32-bit register?

For example, $\boxed{\text{10101010 10101010}}\boxed{\text{11110000 11110000}}$

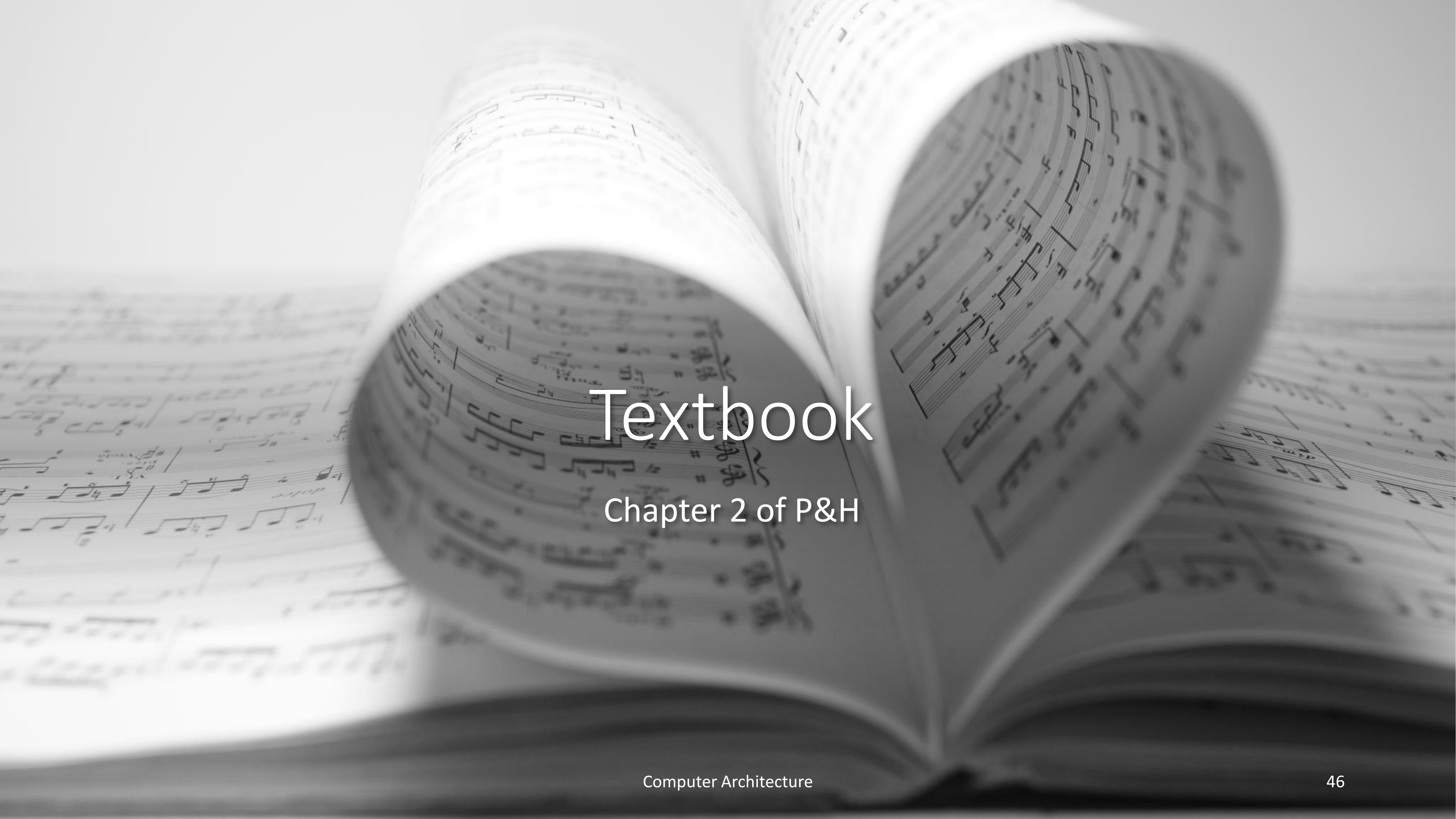
lui $t0, 0xAAAA $\boxed{\text{\#1010101010101010}}$, lower bits all 0s.

Basically it will be 0xAAAA0000 (in hexadecimal)

ori $t0, $t0, 0xF0F0 $\boxed{\text{\#1111000011110000}}$

it will be 0xAAAAF0F0

lui: upper bits, ori/addi: lower bits



Textbook

Chapter 2 of P&H

Coffee Points

Café closed ☺

ਤੁਹਾਡਾ ਦਿਨ ਚੰਗਾ ਬੀਤੇ