CASPER

CS230: Digital Logic Design and Computer Architecture

Lecture 6: Hola Computer Architecture

https://www.cse.iitb.ac.in/~biswa/courses/CS230/autumn23/main.html


Coffee Points
Café closed ☺

Architecture-101

Next Few Lectures

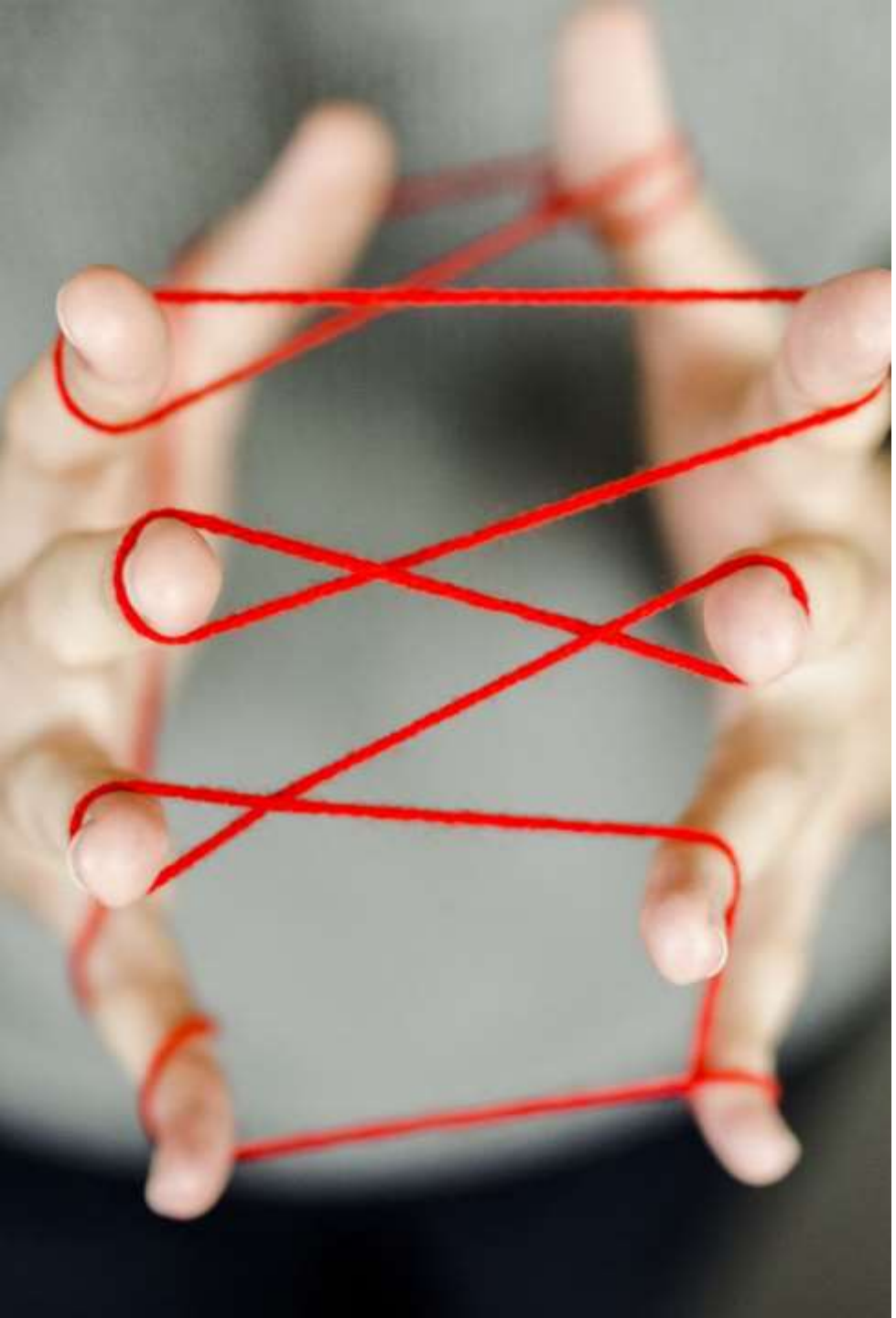
HOW CAN A PROGRAMMER INTERACT WITH THE PROCESSOR?

THE LANGUAGE OF COMPUTER: INSTRUCTIONS

INSTRUCTIONS HAVE A VOCABULARY CALLED INSTRUCTION SET

DRIVEN BY INSTRUCTION SET ARCHITECTURE (ISA)

ISA: X86, ARM, RISC-V, MIPS

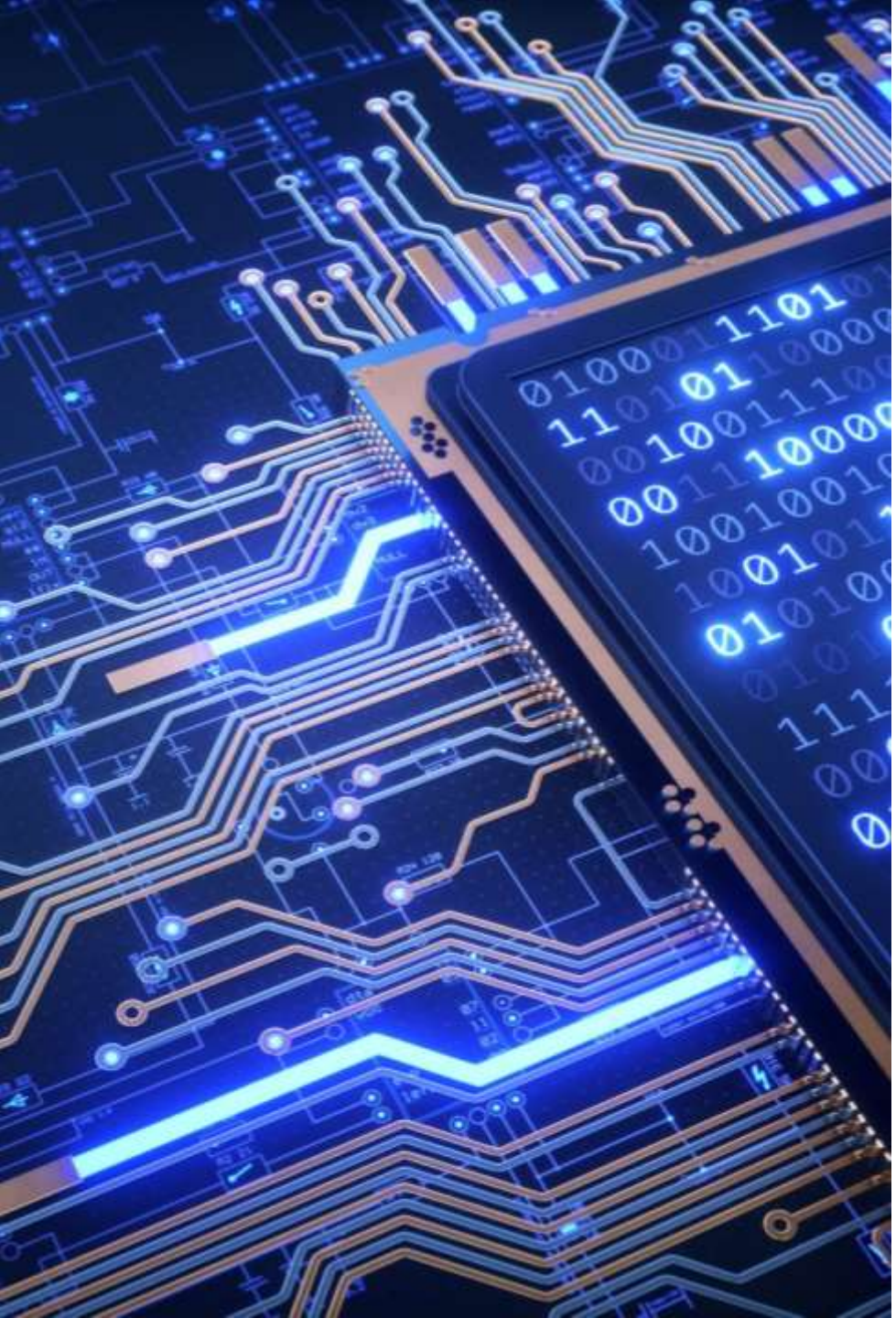


Why MIPS?

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

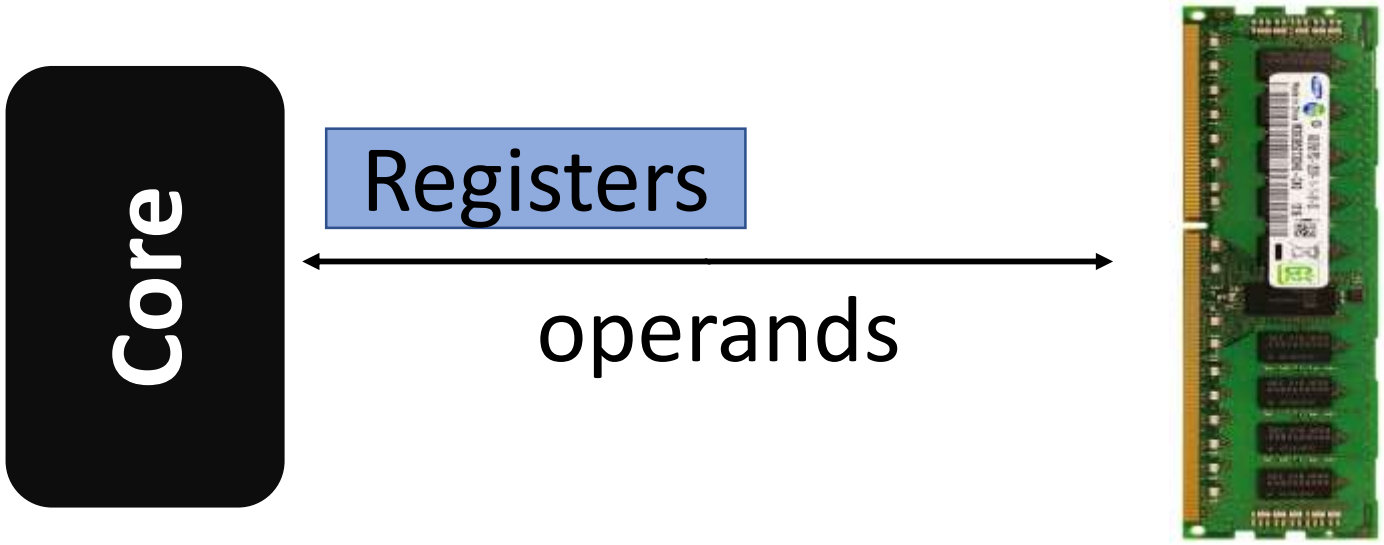
add $1, $2, $3 // assembly language as per the ISA

assembler

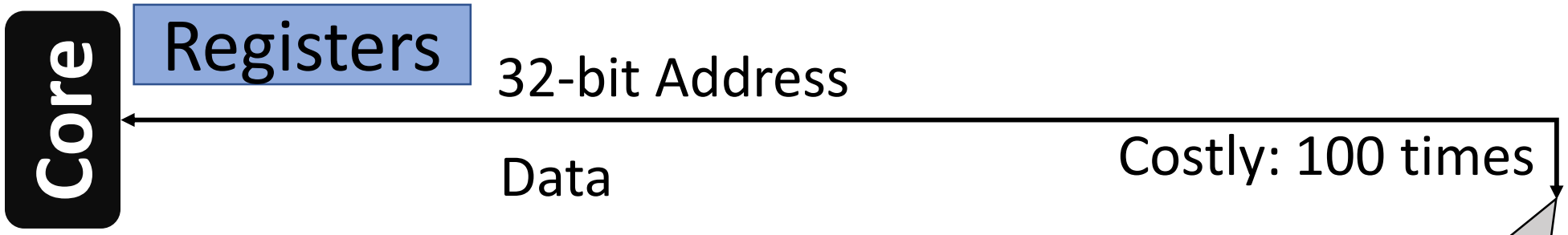
01010101010101010 // machine language, 0s and 1s

Abstraction Example: 101

Operands can be in registers or in memory



A bit detailed



Registers are limited

GBs DRAM

South pole ☹

Instructions

Programmers’ order/command to the processor

Why Instructions?

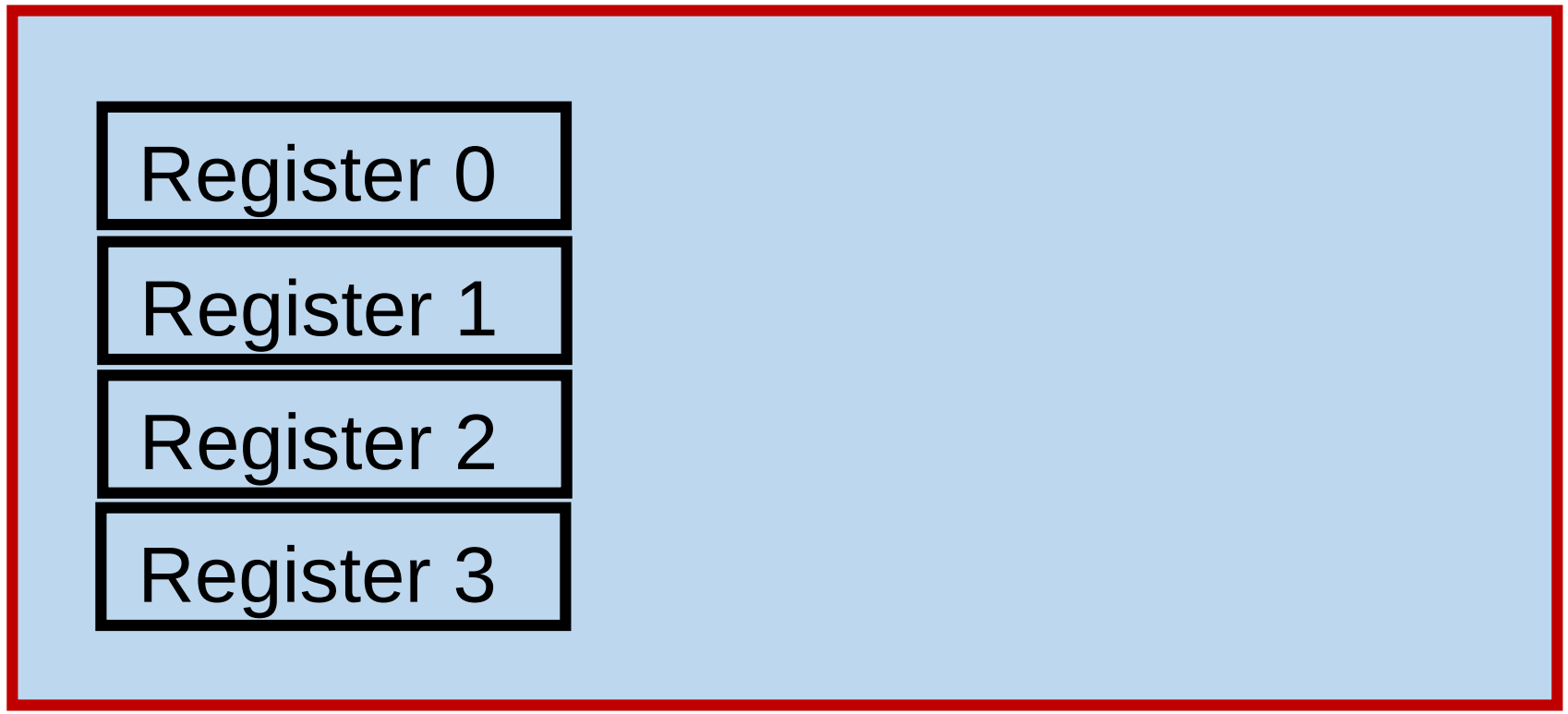
Programmer knows what it can/cannot
Processor knows what it should

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

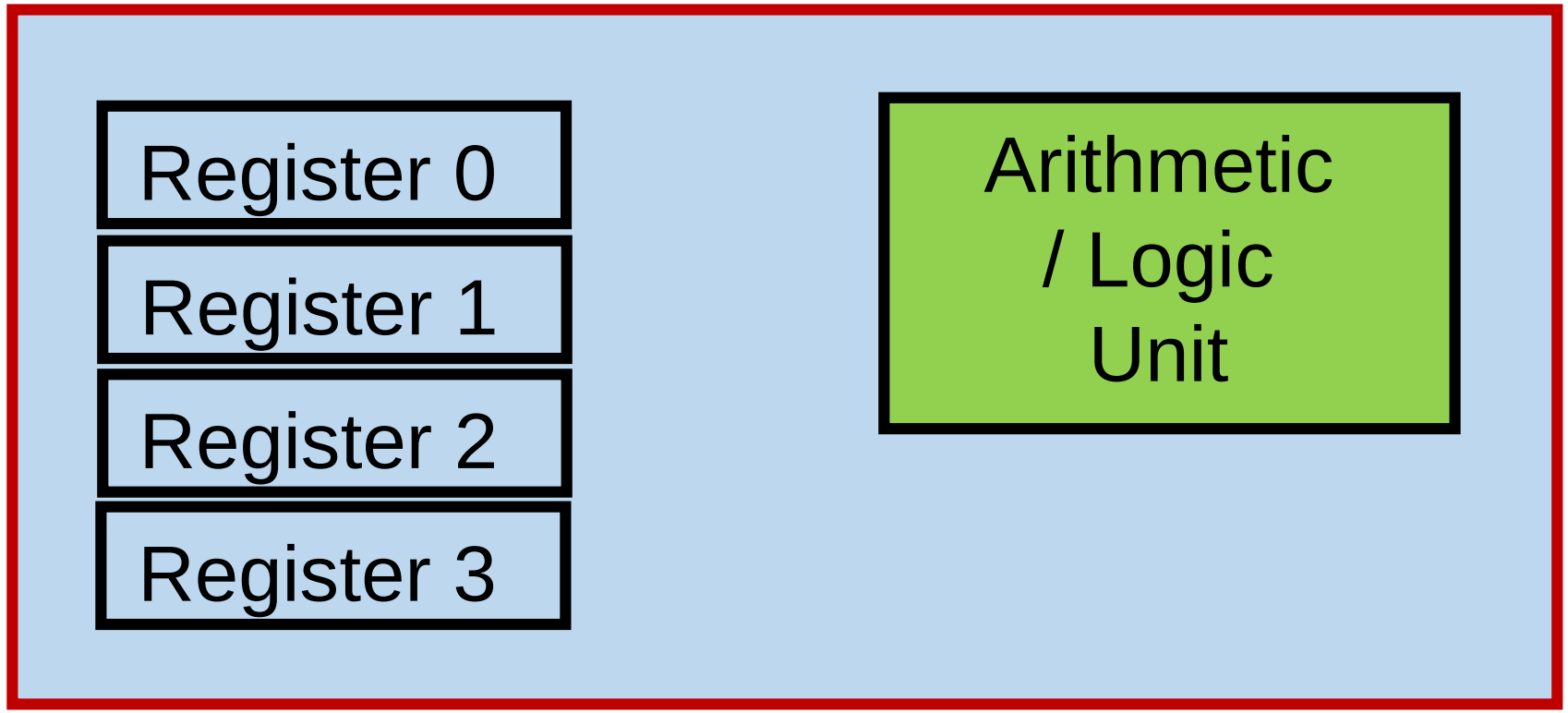
World with no instructions

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

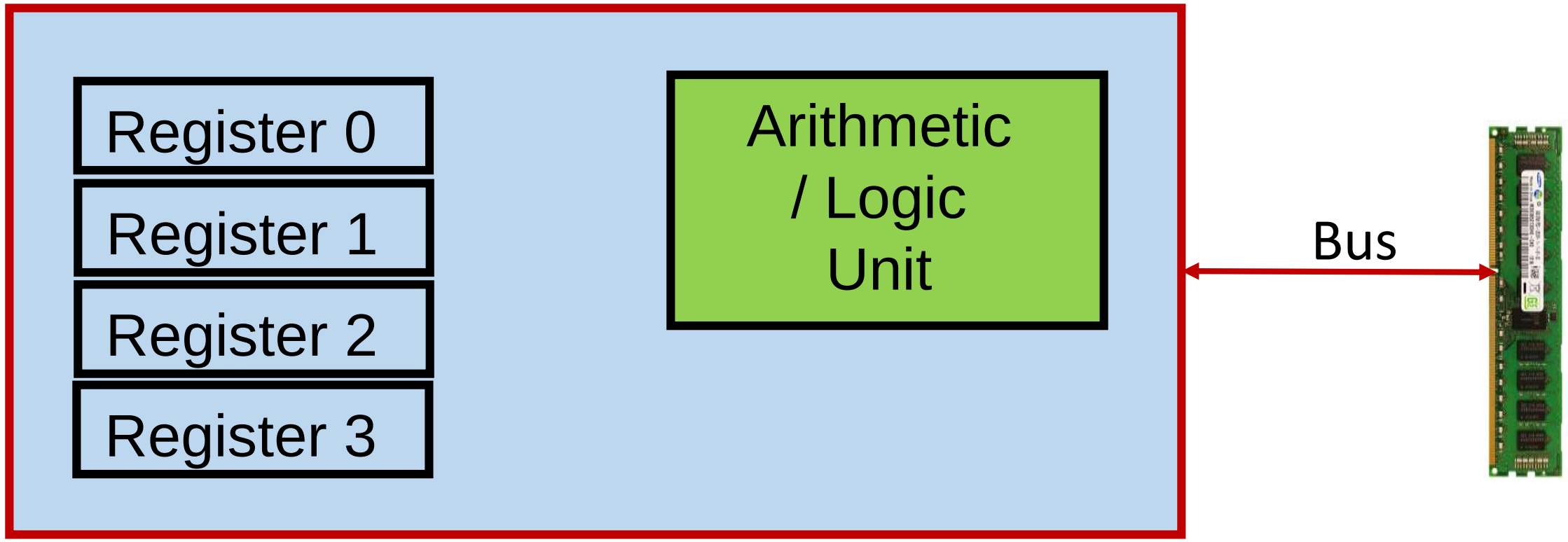
Let’s Open the Processor Core



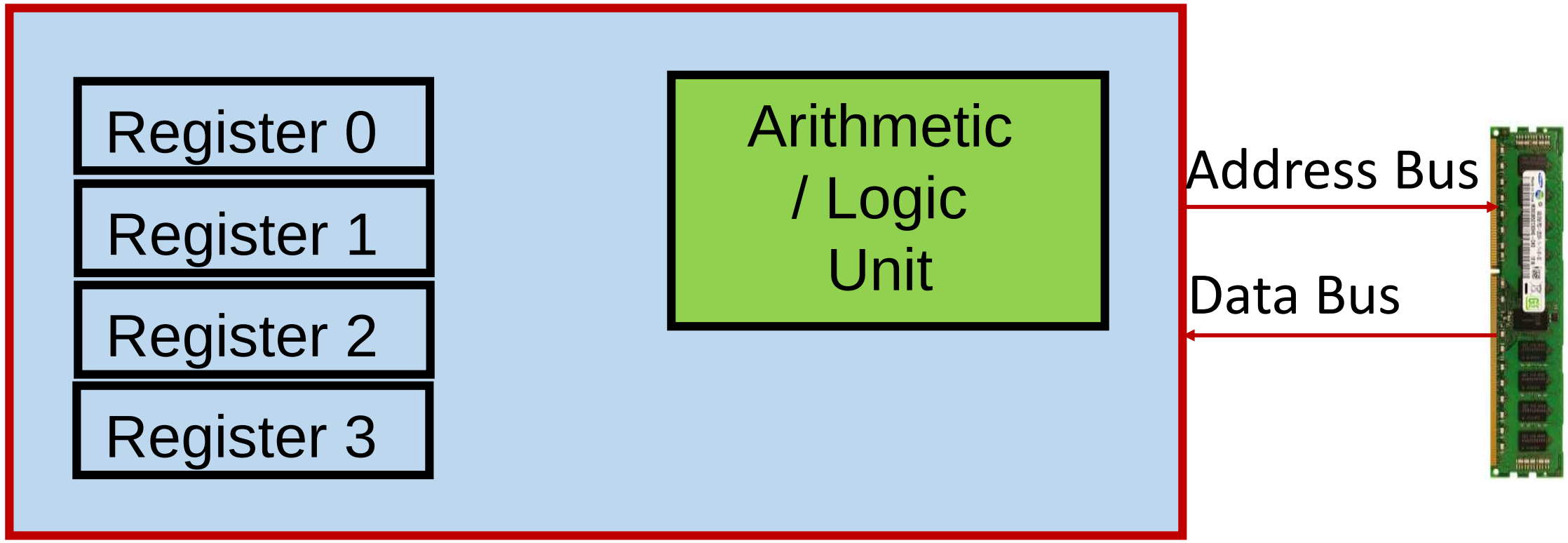
Let’s Open the Processor Core



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

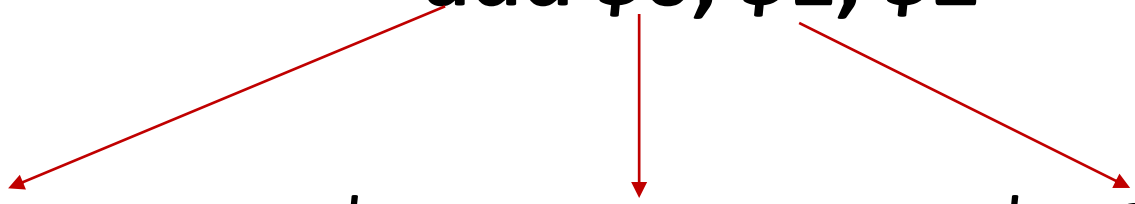


Let’s put the Memory (not inside the core)



MIPS Instructions: 101

add $0, $1, $2



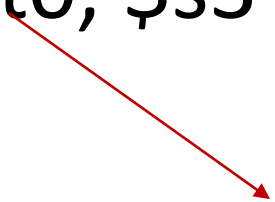
add: operation, $0: Destination, $1 & $2: Source(s)

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
```



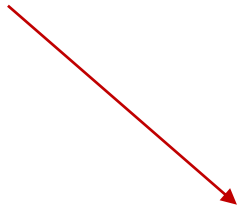
Temporary register

Try out:

f=(g+h) – (i+j)

Constants and Immediate

x=x+10



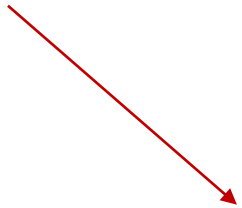
No need of a register

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

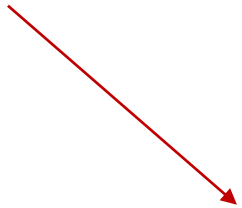
Do we need a subi ? ☺

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

Do we need a subi ? ☺	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, 10101010 10101010 11110000 11110000

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

Basically it will be 0xAAAA0000 (in hexadecimal)

ori $t0, $t0, 0xF0F0 #1111000011110000

it will be 0xAAAAF0F0

lui: upper bits, ori/addi: lower bits

Coffee Credits

- Navya +1
- Satyankar +1

Congrats ISRO, Well Done India

Processor used: Vikram (some avatars of Vikram), 32-bit, 150 instructions

Photo credit: To the owner!