Instruction Set-Part 1

Lecture Objectives:

1) Explain the difference between Harvard and Von Neumann architectures in a computer.
2) Define instruction set
3) Explain the concept of the source and destination registers for the MIPS instruction set.
4) Using the MIPS instruction set, explain how to add a set of variables.
5) Define the term computer register
6) Define the term data transfer instruction
7) Using the MIPS instruction set, initialize a register to a fixed value.

No class Friday Morning.

will be a short video on website to watch.

will be a short quiz on

Monday ..

⇒ Over chapter 1 material

⇒ Homework will be e-mailed

out.

- Calculate the geometric mean of the following numbers
  - 1, 2, 4
  - 2, 4, 8, 16, 32

Leadin

$\sqrt[n]{\text{Product of \#}}$

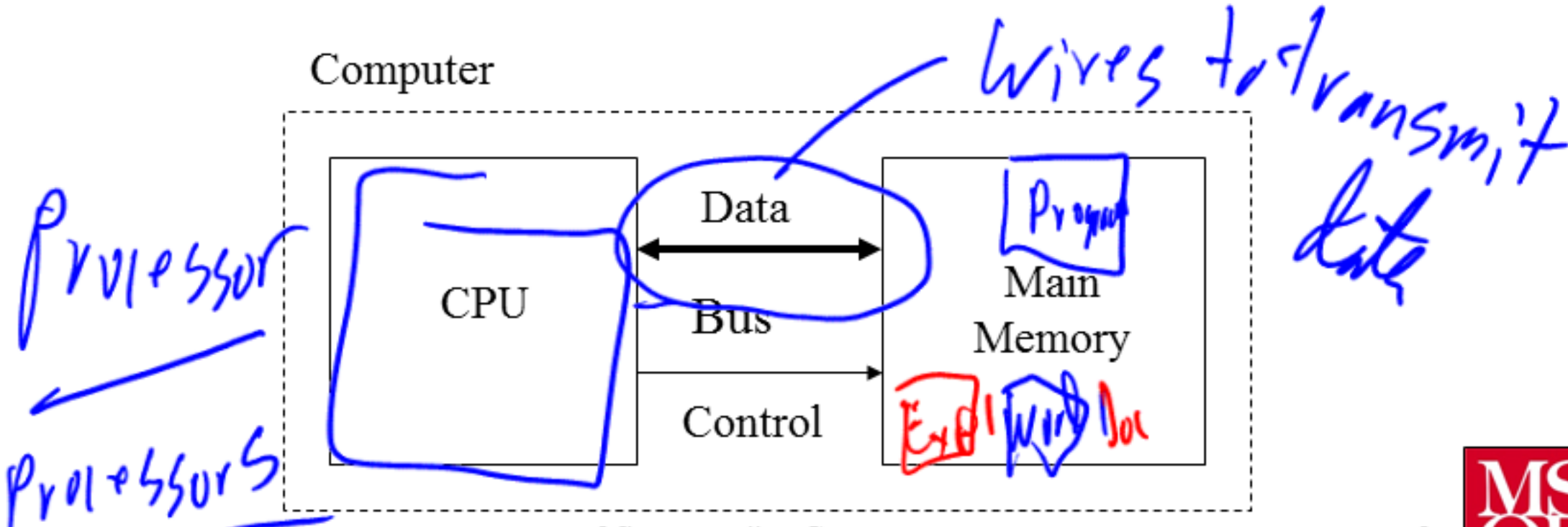
$\sqrt[3]{1*2*4} \Rightarrow \sqrt[3]{8} \Rightarrow 2$

$\sqrt[5]{2*4*8*16*32} \Rightarrow$ Big #

$\sqrt[5]{32768} \Rightarrow 8$

von Neumann Architecture

- Principles
  - Data and instructions are both stored in the main memory(stored program concept)
  - The content of the memory is addressable by location (without regard to what is stored in that location)
  - Instructions are executed sequentially unless the order is explicitly modified
  - The basic architecture of the computer consists of:

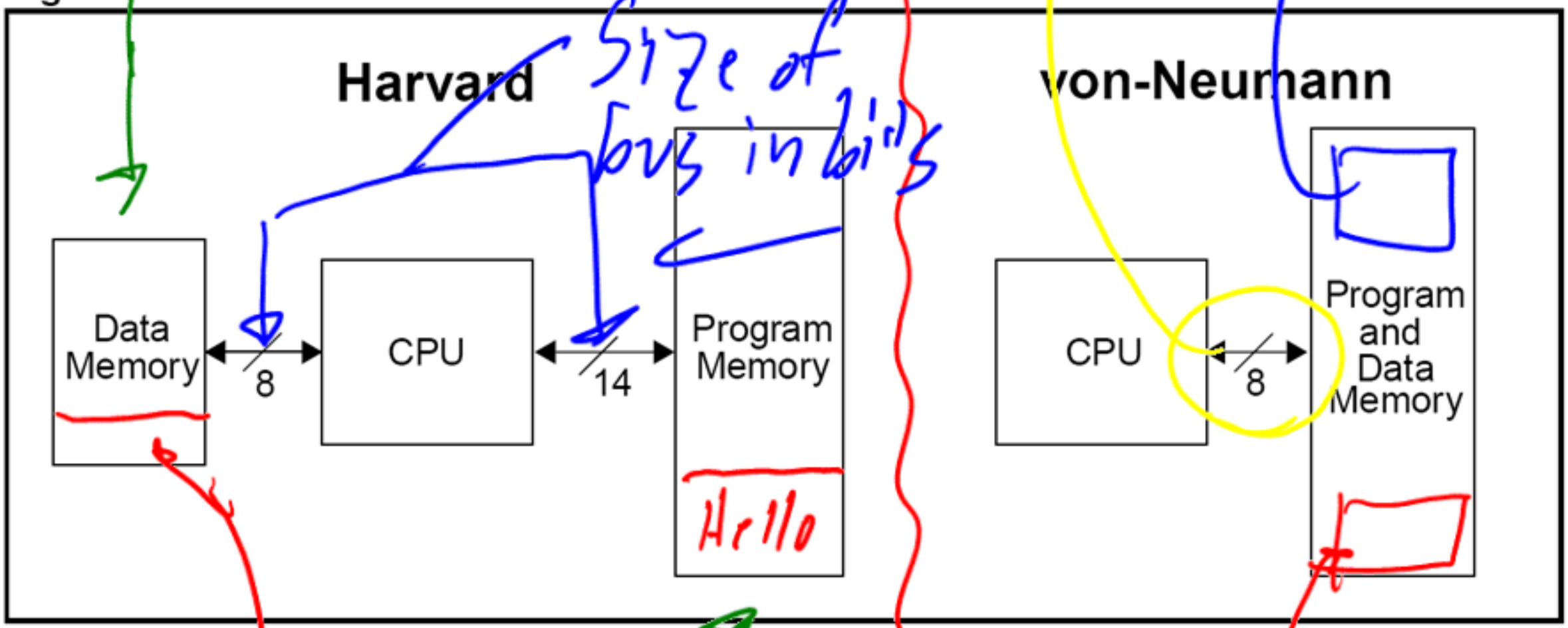


Harvard Architecture

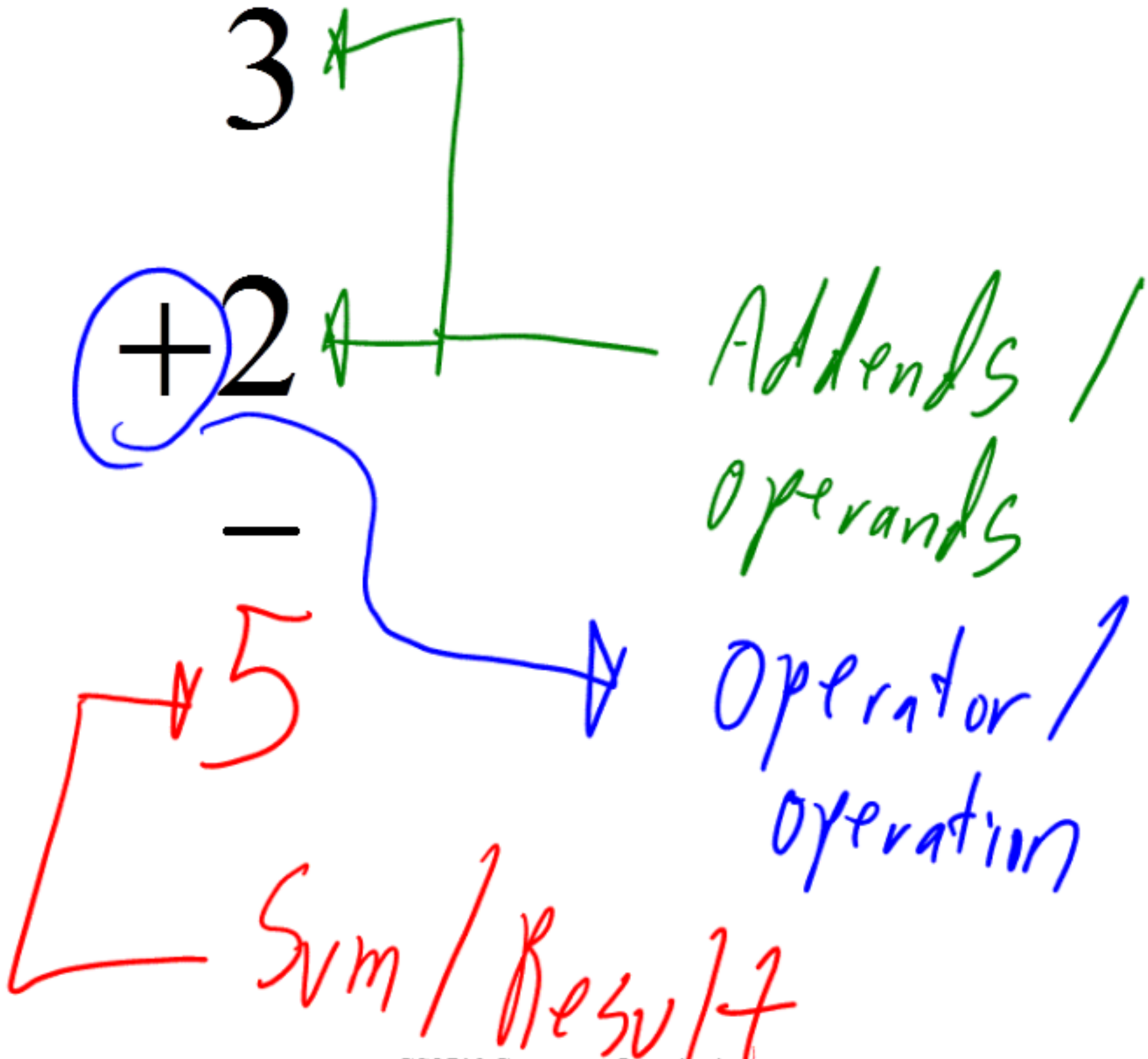
- Assign data and program instructions to different memory spaces.
- Each memory space has a separate bus.
- Allows:
  - Different timing, size, and structure for program instructions and data.
  - Concurrent access to data and instructions.
  - Clear partitioning of data and instructions (=> security)
  - Harder to program

Harvard Versus von Neumann Architecture

Figure 4-1: Harvard vs. von Neumann Block Architectures



Operands and Operations in Mathematics



Mathematics Operations

- Addition
- Subtraction
- Multiplication
- Division
- Power
- Square Root
- Etc.

adding

All those things from Java.

Definition of what an Instruction Set is for a given computer

- *The vocabulary of commands understood by a given computer architecture.*
  - The repertoire of instructions of a computer
- Different computers have different instruction sets
  - But with many aspects in common
- Early computers had very simple instruction sets
  - Simplified implementation
- Many modern computers also have simple instruction sets

Stored Program Concept

- The idea that instructions and data of many types can be stored in memory as numbers.

=> Hex values

(compiler .class

Code => "Assembly" => "Hex Code"

"Binary Code"

The MIPS Instruction Set

- Used as the example in textbook
- Will be used as example language for the course
- Stanford MIPS commercialized by MIPS Technologies (www.mips.com)
- Large share of embedded core market
  - Applications in consumer electronics, network/storage equipment, cameras, printers, …
- Typical of many modern ISAs
  - See MIPS Reference Data tear-out card, and Appendixes B and E

MIPS Assembly Language

Statement

```
add a, b, c # Adds the sum of b and c and places it in a.
```

Comment

Destination register on the MIPS

Operands

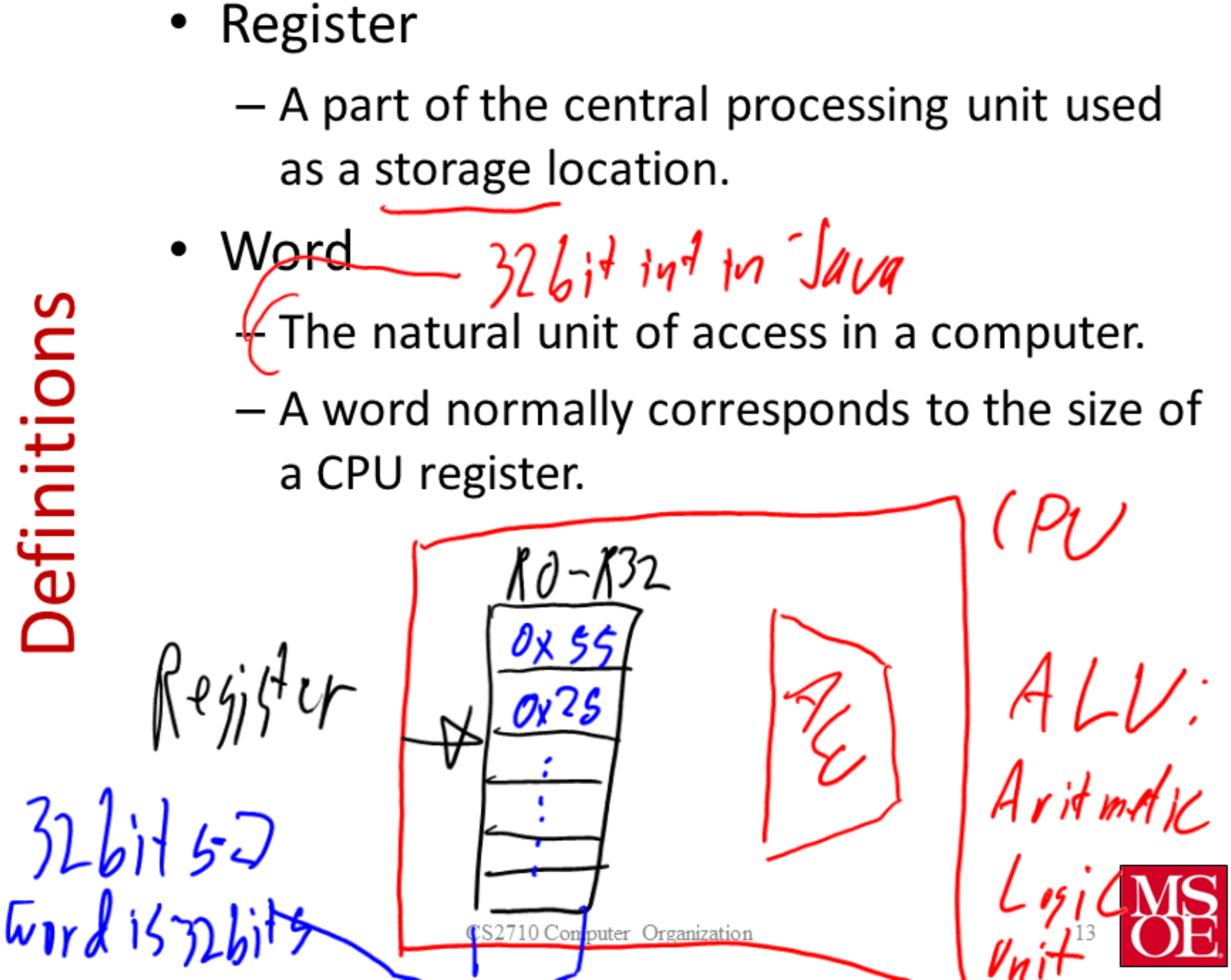
Operand

All arithmetic operations have this form.

Design principle 1: Simplicity favors regularity

Definitions

- Register
  - A part of the central processing unit used as a storage location.
- Word
  - The natural unit of access in a computer.
  - A word normally corresponds to the size of a CPU register.



An Arithmetic Example

- Initial C / Java code:

```
f = (g + h) - (i + j);
```

- Refactored C / Java Code

```
temp0 = g + h;
temp1 = I + j;
f = temp0 - temp1;
```

Re-arranged

- MIPS Assembly Code:

```
add t0, g, h   # temp t0 = g + h
add t1, i, j   # temp t1 = i + j
sub f, t0, t1  # f = t0 - t1
```

operands / Source registers

t0 / t1 => Destination Registers

MIPS Architecture

- Arithmetic instructions use register operands
- MIPS has a 32 × 32-bit register file
  - Use for frequently accessed data
  - Numbered 0 to 31
  - 32-bit data called a “word”

Fully worked register example

- C code:

```
f = (g + h) - (i + j);
```

  - f, …, j in $s0, …, $s4

- Compiled MIPS code:

```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

Register Operands

- Assembler names
  - $t0, $t1, …, $t9 for temporary values
  - $s0, $s1, …, $s7 for saved variables

Problem

- r = 6 + 3;

- How do we do it?