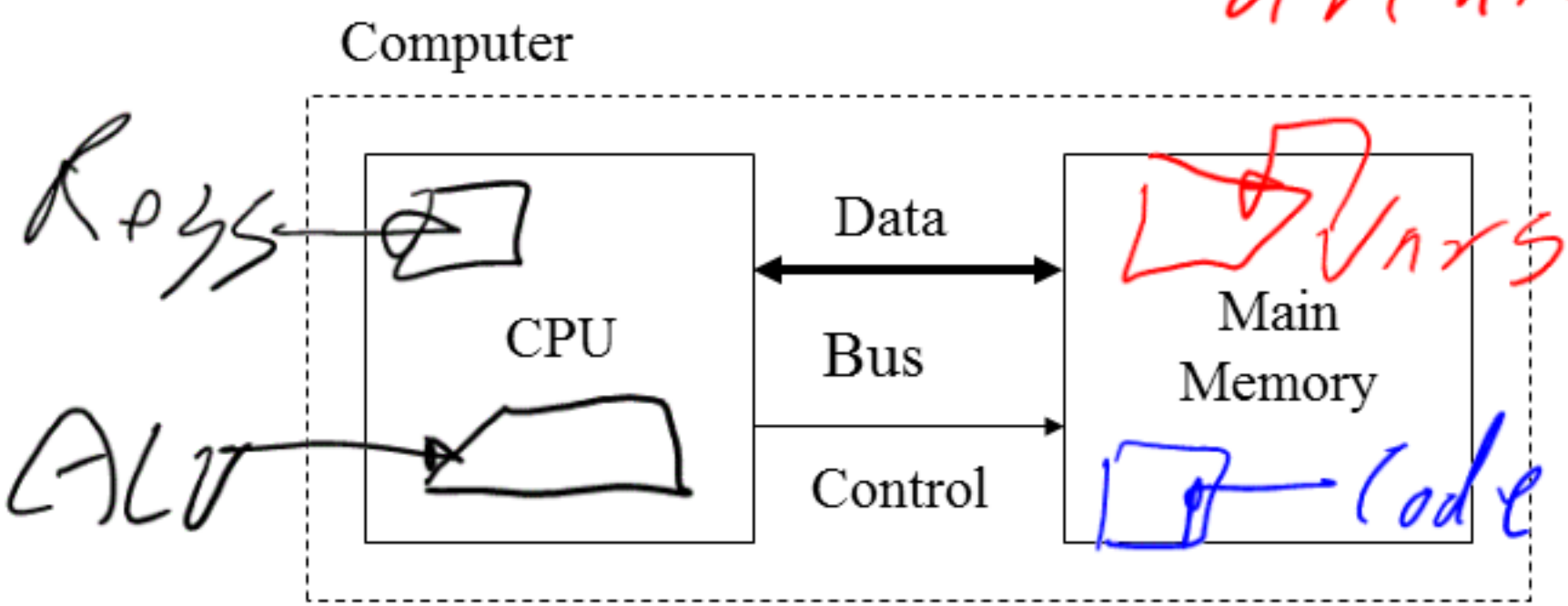
Instruction Set-Part 2

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 address
7) Define the term data transfer instruction
8) Using the MIPS instruction set, initialize a register to a fixed value.
9) Define alignment restriction
10) Define the terms "Big endian" and "little endian"

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

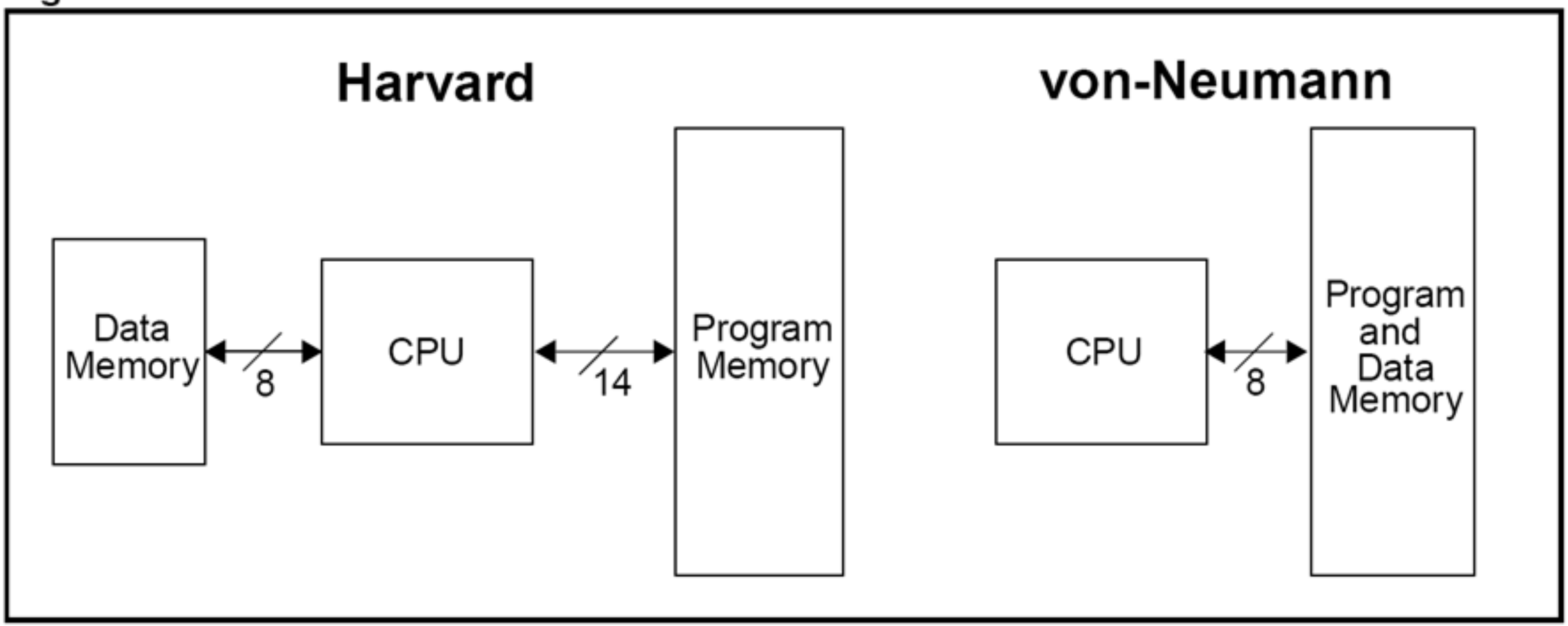
- Assign data and program instructions to different memory spaces.
- Each memory space has a separate bus.
- Allows:
  - Different timing, size, and structure for program instructions and data.
  - Concurrent access to data and instructions.
  - Clear partitioning of data and instructions (=> security)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

C Not as common

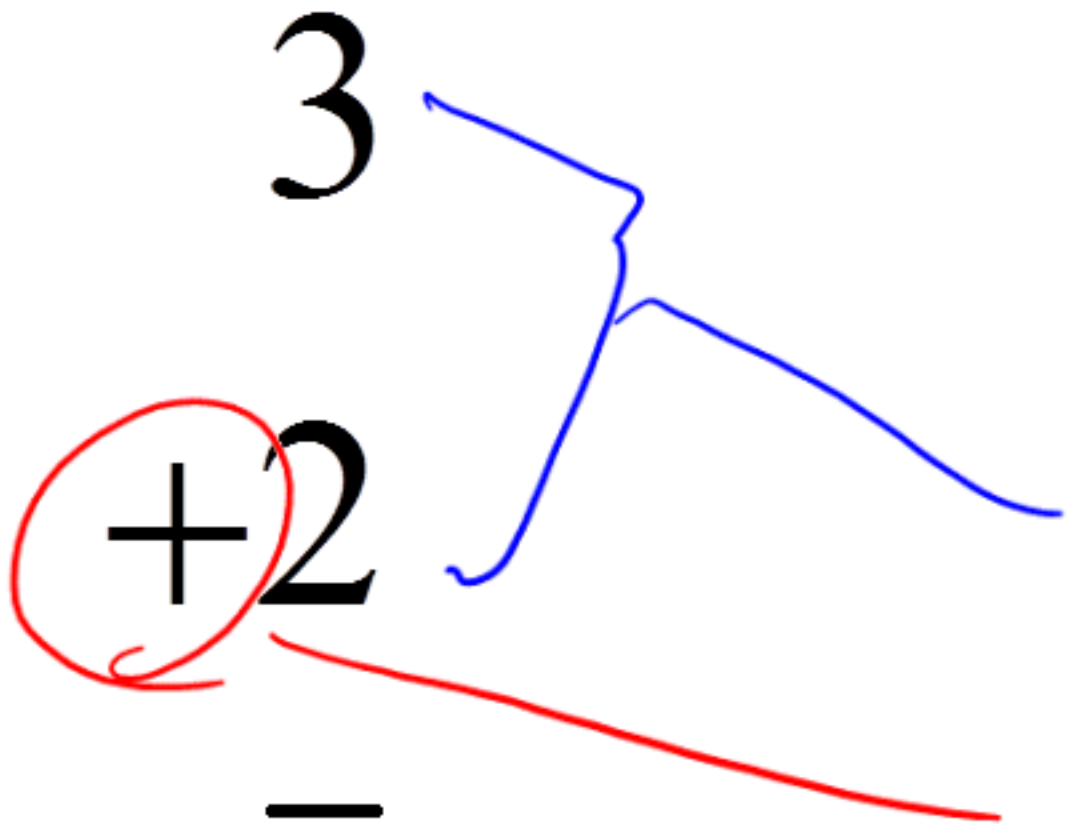
Harvard Versus von Neumann Architecture

ure

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



Operands and Operations in Mathematics



Operands

Operation

Result

Mathematics Operations

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

Same as Java.

Instruction Set

- *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.

Source code is encoded into hex #'s in memory.

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

Very Similar to ARM
⇒ Most common embedded micro.

MIPS Assembly Language
Statement

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

Instruction

operands

Destination

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.

on the mips

32 bits on a MIPS processor.

An Arithmetic Example

- Initial C / Java code:

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

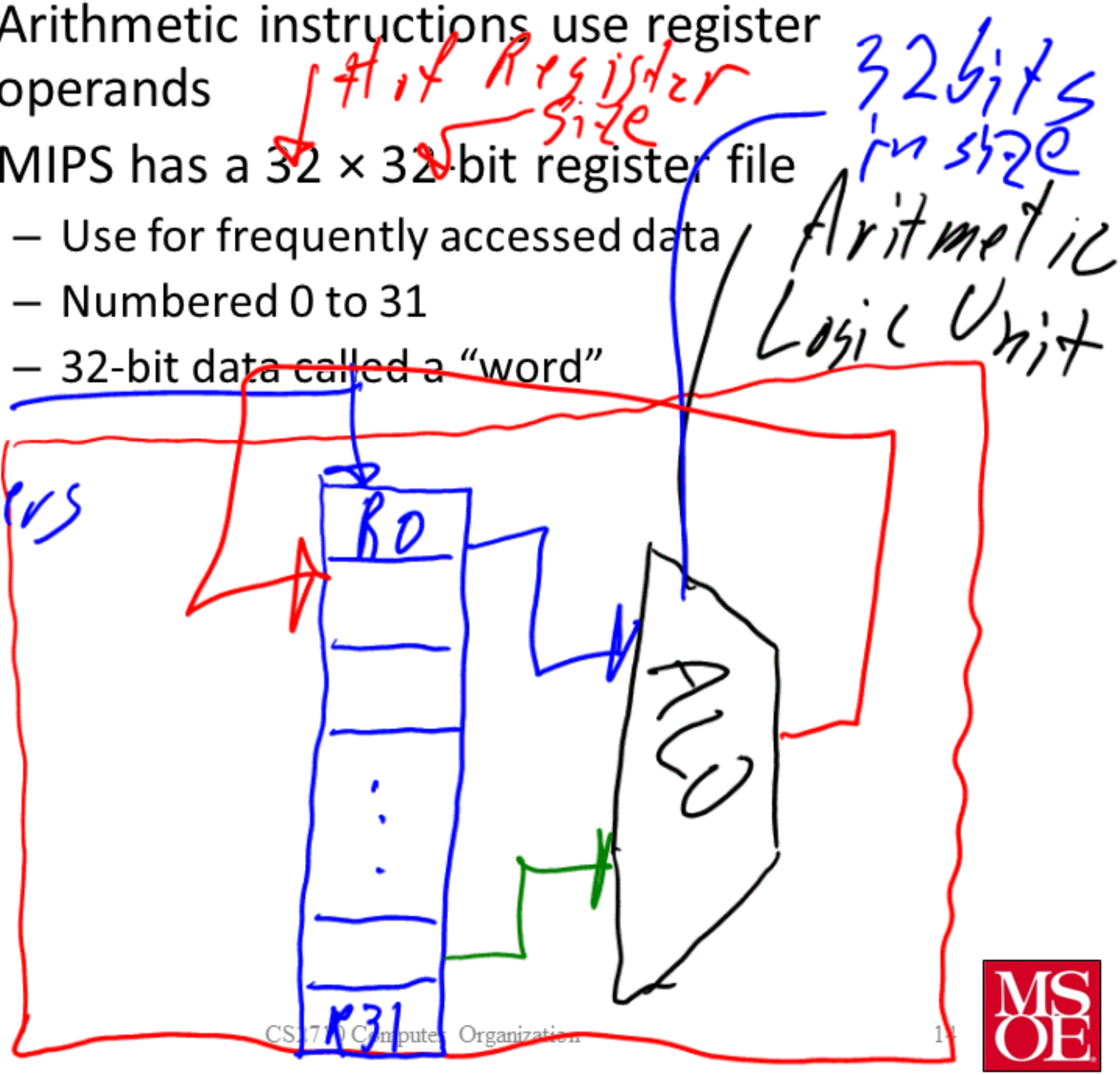
- Refactored C / Java Code

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

- 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
```

- 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"



Registers

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
```

Example code translation

Real

register 16 Code

$t7 => represents

temp registers

$s represents stores regs

Register Operands

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

Green Card in the text.

- r = 6 + 3;

temp register

temp Register

- How do we do it?

Add the two temps.

Problem

li $t0, 6 ⇐ Load immediate value of 6 into t0.

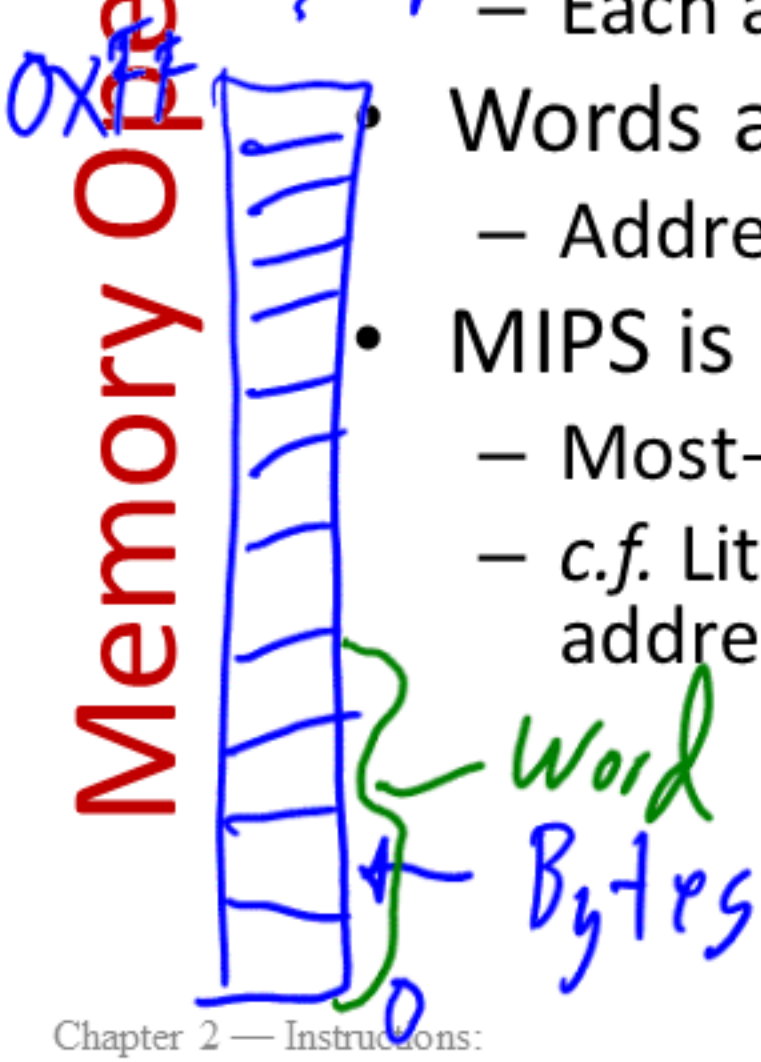
li $t1, 3 ⇐ Load second value

Add $s0, $t0, $t1

Register Operands

- 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”
- Assembler names
  - $t0, $t1, …, $t9 for temporary values
  - $s0, $s1, …, $s7 for saved variables
- *Design Principle 2:* Smaller is faster
  - c.f. main memory: millions of locations

- Main memory used for composite data
  - Arrays, structures, dynamic data
- To apply arithmetic operations
  - Load values from memory into registers
  - Store result from register to memory
- Memory is byte addressed
  - Each address identifies an 8-bit byte
- Words are aligned in memory
  - Address must be a multiple of 4
- MIPS is Big Endian
  - Most-significant byte at least address of a word
  - *c.f.* Little Endian: least-significant byte at least address



CPU [Registers] ∈ Fastest but few

Slower but Larger ⇒ [Main Memory (RAM)]

Little Endian

Big Endian

0x03 0x01 | 0x67

0x02 0x23 | 0x45

0x01 0x45 | 0x23

0x00 0x67 | 0x01

LSB

0x 01 23 45 67

Most Significant Byte

Least Significant Byte

Memory Operand Example 1

- C code:

  g = h + A[8];

  – g in $s1, h in $s2, base address of A in $s3

- Compiled MIPS code:

  – Index 8 requires offset of 32

    - 4 bytes per word

```
lw  $t0, 32($s3)   # load word
add $s1, $s2, $t0
```

offset — base register

Array

Base Address

Why 32?
↑ offset of 8 words.
Each word is 4 bytes.

Memory Operand Example 2

- C code:

```
A[12] = h + A[8];
```

  - h in $s2, base address of A in $s3

- Compiled MIPS code:

  - Index 8 requires offset of 32

```
lw  $t0, 32($s3)     # load word
add $t0, $s2, $t0
sw  $t0, 48($s3)     # store word
```

$s3 is base address of the array

Store Word: Puts a word from a register into memory.

MSOE

Registers vs. Memory

- Registers are faster to access than memory
- Operating on memory data requires loads and stores
  - More instructions to be executed
- Compiler must use registers for variables as much as possible
  - Only spill to memory for less frequently used variables
  - Register optimization is important!

Operands

- Constant data specified in an instruction

```
addi $s3, $s3, 4
```

$s3 += 4;

- No subtract immediate instruction
  - Just use a negative constant

```
addi $s2, $s1, -1
```

Immediate

- *Design Principle 3:* Make the common case fast
  - Small constants are common
  - Immediate operand avoids a load instruction

The Constant Zero

- MIPS register 0 ($zero) is the constant 0
  - Cannot be overwritten
- Useful for common operations
  - E.g., move between registers

```
add $t2, $s1, $zero
```