

---

# **C335**

## **Computer Structures**

### **MIPS Instructions (Part #2)**

Dr. Liqiang Zhang

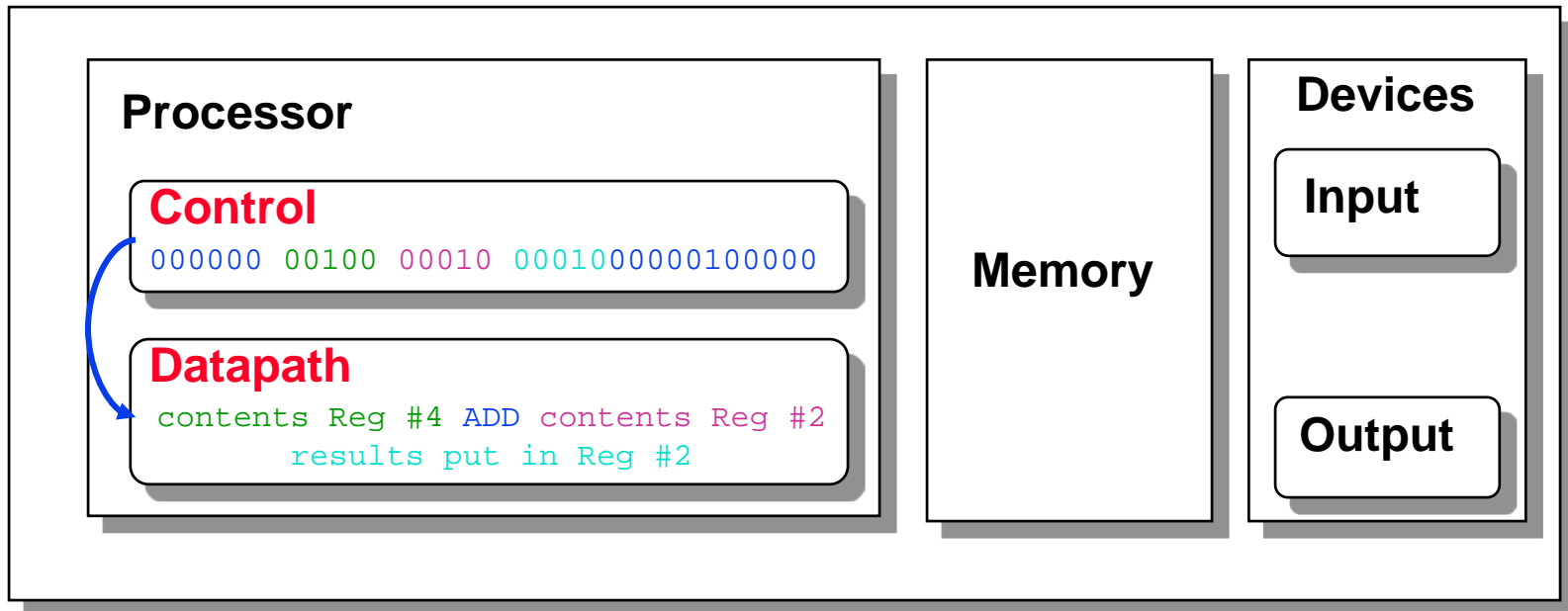
Department of Computer and Information Sciences

Adapted from Morgan Kaufmann and others

# Review: Execute Cycle

---

The datapath **executes** the instructions  
as directed by control



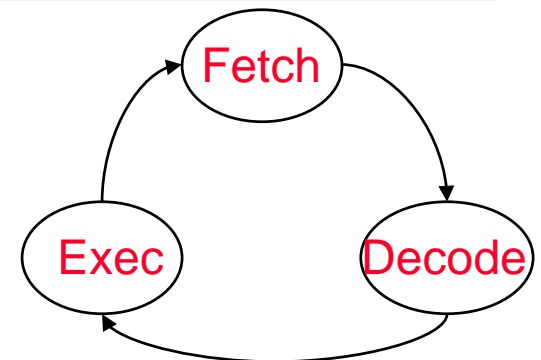
Memory stores **both** instructions and  
data

# Processor Organization

---

## ❑ Control needs to have circuitry to

- Decide which is the next instruction and input it from memory
- Decode the instruction
- Issue signals that control the way information flows between datapath components
- Control what operations the datapath's functional units perform



## ❑ Datapath needs to have circuitry to

- Execute instructions - functional units (e.g., adder) and storage locations (e.g., register file)
- Interconnect the functional units so that the instructions can be executed as required
- Load data from and store data to memory

# Review and more: RISC vs. CISC

---

- ❑ RISC philosophy
  - fixed instruction lengths
  - load-store instruction sets
  - limited number of addressing modes
  - limited number of operations
- ❑ MIPS, ARM, Sun SPARC, HP PA-RISC, IBM PowerPC ...
- ❑ Instruction sets are measured by how well compilers use them as opposed to how well assembly language programmers use them
  
- ❑ CISC (C for complex), e.g., Intel x86

# Review and More: ISA

---

- ❑ The language of the machine
  - Want an ISA that makes it easy to build the hardware and the compiler while maximizing **performance** and minimizing **cost**
- ❑ Stored program (von Neumann) concept
  - Instructions are stored in memory (as is the data)
- ❑ Our target: the MIPS ISA
  - similar to other ISAs developed since the 1980's
  - used by Broadcom, Cisco, NEC, Nintendo, Sony, ...

*Design goals: maximize performance, minimize cost, reduce design time (time-to-market), minimize memory space (embedded systems), minimize power consumption (mobile systems)*

# The Four Design Principles

---

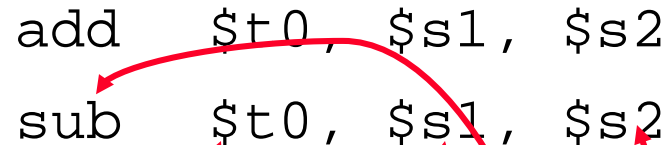


1. Simplicity favors regularity.
2. Smaller is faster.
3. Make the common case fast.
4. Good design demands good compromises.

# Review and More: MIPS Arithmetic Instruction

- ❑ MIPS assembly language arithmetic statement

add \$t0, \$s1, \$s2  
sub \$t0, \$s1, \$s2



- ❑ Each arithmetic instruction performs only **one** operation
- ❑ Each arithmetic instruction specifies exactly **three** operands

destination ← source1 **op** source2

- Operand order is fixed (the destination is specified first)

- ❑ The operands are contained in the datapath's **register file** (\$t0, \$s1, \$s2)

# Compiling More Complex Statements

---

- Assuming variable `b` is stored in register `$s1`, `c` is stored in `$s2`, and `d` is stored in `$s3` and the result is to be left in `$s0`, what is the assembler equivalent to the C statement

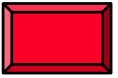
$$h = (b - c) + d$$

```
sub    $t0, $s1, $s2
```

```
add    $s0, $t0, $s3
```



# MIPS Register File



- ❑ Operands of arithmetic instructions must be from a limited number of special locations contained in the datapath's **register file**

- Thirty-two 32-bit registers
  - Two read ports
  - One write port

- ❑ Registers are

- Fast
  - **Smaller is faster & Make the common case fast**
- Easy for a compiler to use
  - e.g.,  $(A*B) - (C*D) - (E*F)$  can do multiplies in any order
- Improves code density
  - Since register are named with fewer bits than a memory location



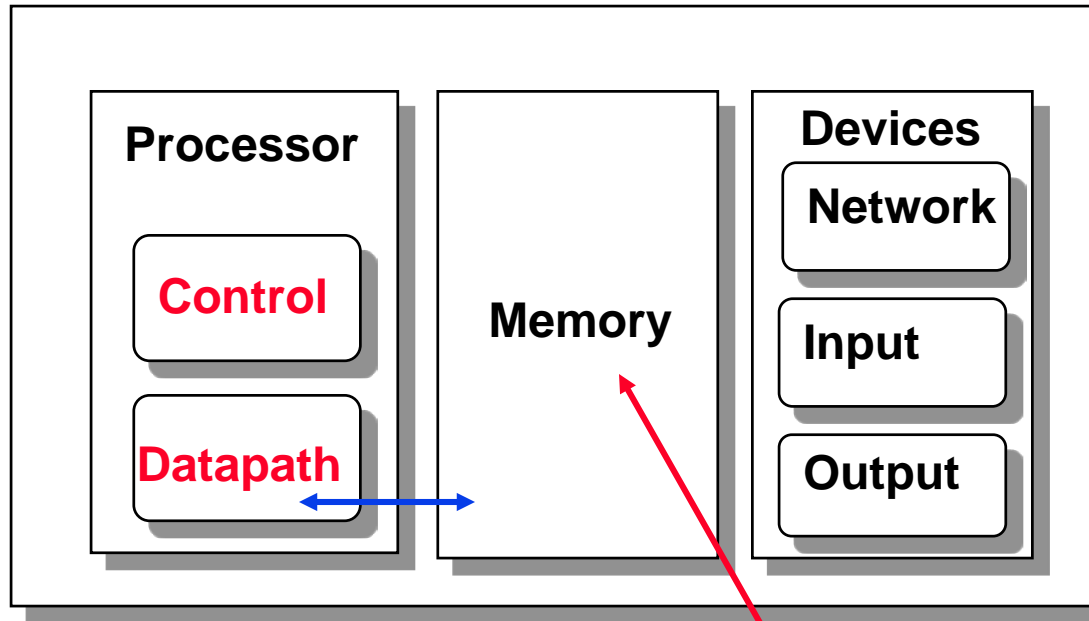
# Naming Conventions for Registers



|     |                                     |     |                                     |
|-----|-------------------------------------|-----|-------------------------------------|
| 0   | <b>\$zero</b> constant 0 (Hdware)   | 16  | <b>\$s0</b> callee saves            |
| 1   | <b>\$at</b> reserved for assembler  | ... | (caller can clobber)                |
| 2   | \$v0 expression evaluation &        | 23  | <b>\$s7</b>                         |
| 3   | \$v1 function results               | 24  | <b>\$t8</b> temporary (cont'd)      |
| 4   | <b>\$a0</b> arguments               | 25  | <b>\$t9</b>                         |
| 5   | <b>\$a1</b>                         | 26  | <b>\$k0</b> reserved for OS kernel  |
| 6   | <b>\$a2</b>                         | 27  | <b>\$k1</b>                         |
| 7   | <b>\$a3</b>                         | 28  | \$gp pointer to global area         |
| 8   | <b>\$t0</b> temporary: caller saves | 29  | \$sp stack pointer                  |
| ... | (callee can clobber)                | 30  | \$fp frame pointer                  |
| 15  | <b>\$t7</b>                         | 31  | <b>\$ra</b> return address (Hdware) |

# Registers vs. Memory

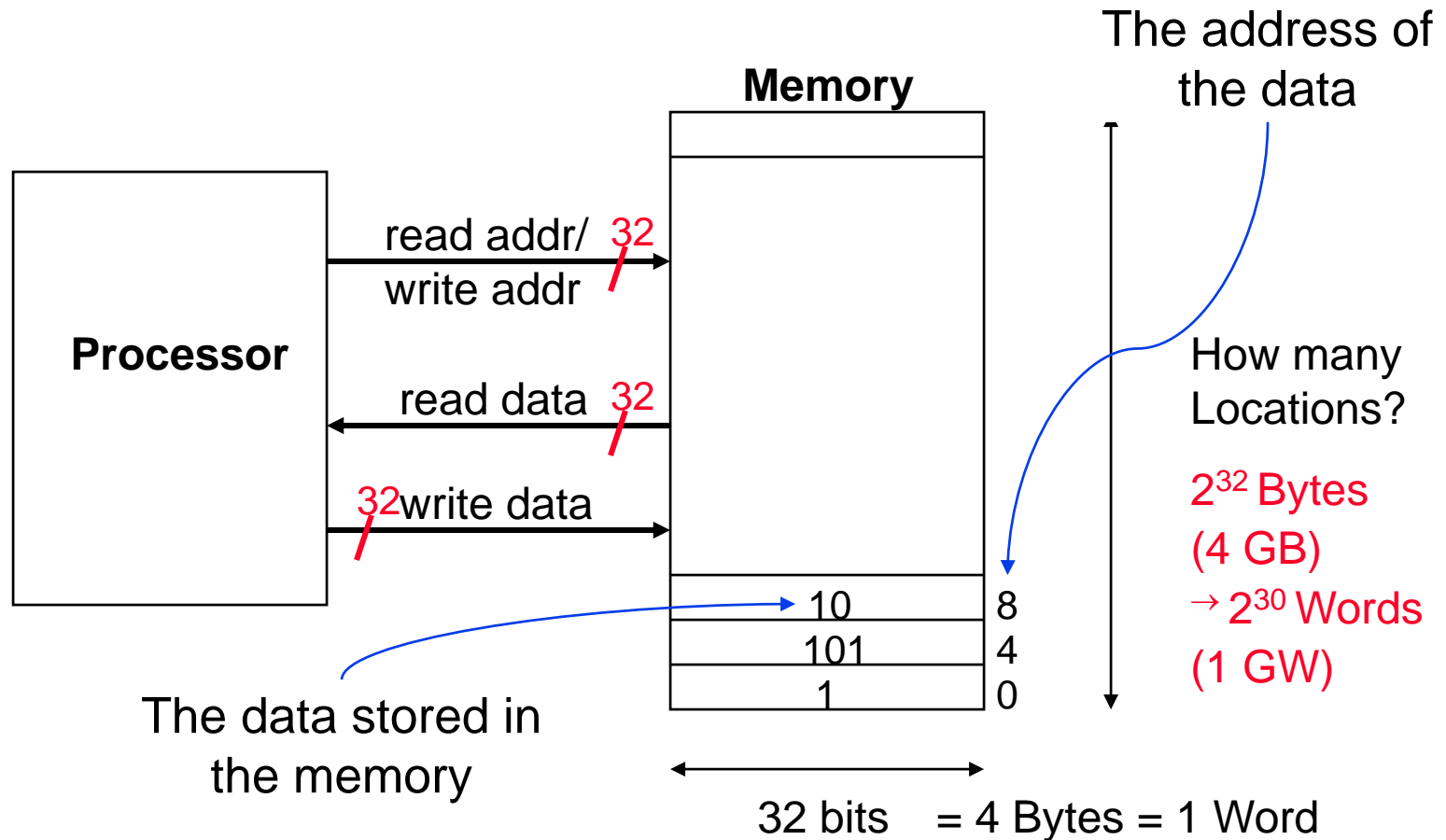
- ❑ Arithmetic instructions *operands* must be in registers
  - only thirty-two registers are provided



- ❑ Compiler associates variables with registers
- ❑ What about programs with *lots* of variables?

# Processor – Memory Interconnections

- ❑ Memory is a large, single-dimensional array
- ❑ An **address** acts as the index into the memory array





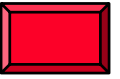
# Accessing Memory

- ❑ MIPS has two basic **data transfer** instructions for accessing memory (assume  $\$s3$  holds  $24_{10}$ )

```
lw      $t0, 4($s3)    #load word from memory
                28
sw      $t0, 8($s3)    #store word to memory
                32
```

- ❑ The data transfer instruction must specify
  - where in memory to read from (load) or write to (store) – **memory address**
  - where in the register file to write to (load) or read from (store) – **register destination (source)**
- ❑ The memory address is formed by **summing the constant portion of the instruction and the contents of the second register**

# MIPS Memory Addressing



- ❑ The memory address is formed by summing the constant portion of the instruction and the contents of the second (base) register

\$s3 holds 8

Memory

|             |         |         |
|-------------|---------|---------|
| ...         | 0 1 1 0 | 24      |
| ...         | 0 1 0 1 | 20      |
| ...         | 1 1 0 0 | 16      |
| ...         | 0 0 0 1 | 12      |
| ...         | 0 0 1 0 | 8       |
| ...         | 1 0 0 0 | 4       |
| ...         | 0 1 0 0 | 0       |
| 32 bit Data |         | Address |

... 0001

lw      \$t0, 4(\$s3)    #what? is loaded into \$t0

sw      \$t0, 8(\$s3)    # \$t0 is stored where?

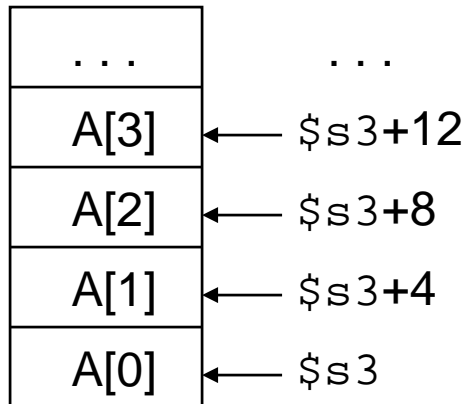
                             in location 16

... 0001

# Compiling with Loads and Stores

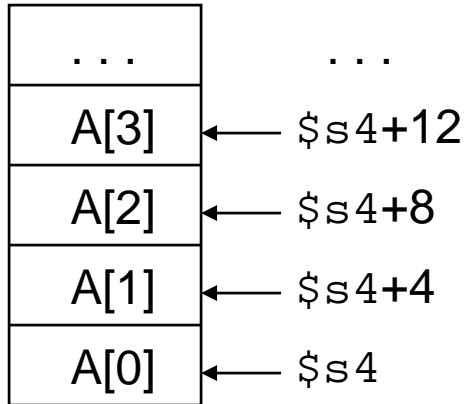
- Assuming variable `b` is stored in `$s2` and that the base address of array `A` is in `$s3`, what is the MIPS assembly code for the C statement

$$A[8] = A[2] - b$$



```
lw    $t0, 8($s3)
sub   $t0, $t0, $s2
sw    $t0, 32($s3)
```

# Compiling with a Variable Array Index



- Assuming that the base address of array A is in register \$s4, and variables b, c, and i are in \$s1, \$s2, and \$s3, respectively, what is the MIPS assembly code for the C statement

$c = A[i] - b$

```
add    $t1, $s3, $s3    #array index i is in $s3
add    $t1, $t1, $t1    #temp reg $t1 holds 4*i
add    $t1, $t1, $s4    #addr of A[i] now in $t1
lw     $t0, 0($t1)
sub    $s2, $t0, $s1
```