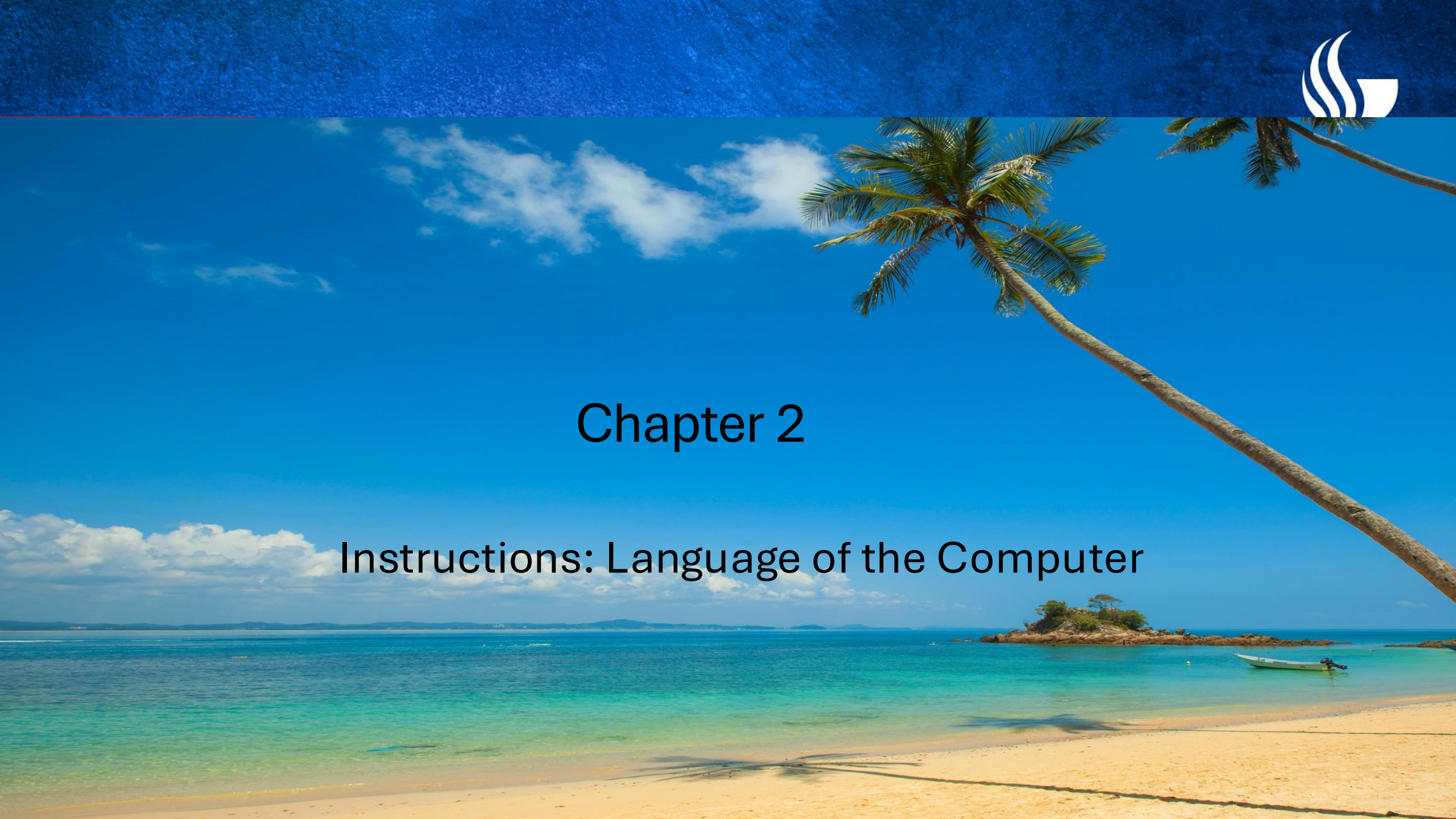
CSC 3210 Computer organization and programming

Chunlan Gao

Georgia State University

About Quiz for Chapter 1

- The quiz will be on Thursday, at the end of the class, online(30mins)
- No calculation in the quiz.
- Some basic knowledge we covered in the chapete1. Use slides as reference. You can have one cheat-sheet. As we have 7 Great Ideas, what does these mean, and so on.



Chapter 2

Instructions: Language of the Computer

Instruction Set

An **instruction** is a single operation that a CPU can perform, the word of a computer's language.
Instruction Set: The repertoire of instructions of a computer.

Different computers have different instruction sets, but with many aspects in common.
Why?

1. All computers are constructed from hardware technologies based on similar underlying principles and because there are a few basic operations that all computers must provide.
2. computer designers have a common goal: to find a language that makes it easy to build the hardware and the compiler while maximizing performance and minimizing cost and energy.

Which instruction set are we going to use?

The RISC-V Instruction Set

- Used as the example throughout the book
- Developed at UC Berkeley as open ISA
- Now managed by the RISC-V Foundation (riscv.org)
- Typical of many modern ISAs
  - See RISC-V Reference Data tear-out card https://www.cs.sfu.ca/~ashriram/Courses/CS295/assets/notebooks/RISCV/RISCV_CARD.pdf
- Similar ISAs have a large share of embedded core market
  - Applications in consumer electronics, network/storage equipment, cameras, printers,.....

Widely used **instruction sets (ISAs)**

1. RISC Instruction Sets

These follow the **Reduced Instruction Set Computer (RISC)** principles, with simple and fast instructions:

- **RISC-V**:
  - Open-source and modular.
  - Example instructions: ADD, SUB, LW (Load Word), SW (Store Word).
- **ARM**:
  - Commonly used in mobile and embedded systems.
  - Example instructions: ADD r0, r1, r2, MOV r0, #5.
- **MIPS**:
  - Used in embedded systems and educational contexts.
  - Example instructions: ADD $t1, $t2, $t3, LW $t1, 0($t2).

Widely used **instruction sets (ISAs)**

- **2. CISC Instruction Sets**

These follow the **Complex Instruction Set Computer (CISC)** principles, with more complex and powerful instructions:

- **x86 (Intel/AMD)**:
  - Dominates personal computers and servers.
  - Example instructions: MOV AX, BX, ADD AX, [address].
- **x86-64 (AMD64)**:
  - Extension of x86 for 64-bit computing.
  - Example instructions: MOV RAX, RBX, CMP RAX, 1.

RISC-V instruction set

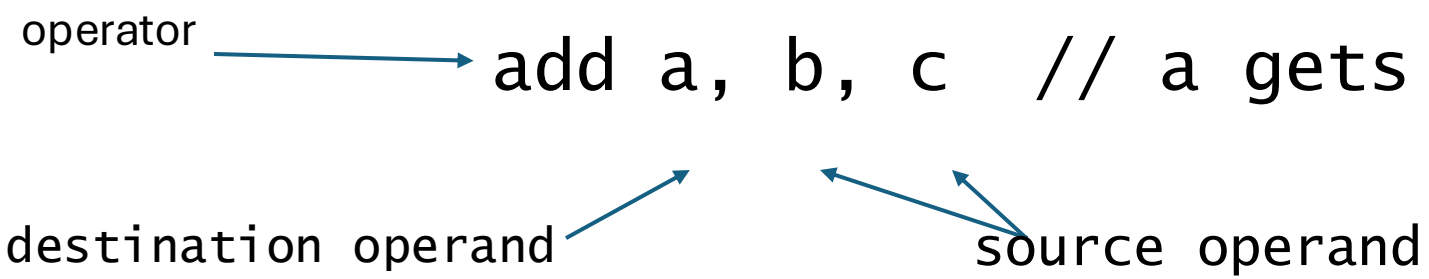
- RISC-V is an open architecture that is controlled by RISC-V International.
- not a proprietary architecture that is owned by a company like ARM, MIPS, or x86.
- In 2020, more than 200 companies are members of RISC-V International, and its popularity is growing rapidly

Different between Open Source and Proprietary

Aspect	Open Source	Proprietary (Closed Source)
Source Code Access	Publicly accessible	Restricted to company or authorized users
Modification	Allowed, encourages customization	Modification is prohibited
Cost	Typically free	Often requires licensing or subscription fees
Maintenance	Community-driven or non-profit organizations	Maintained by the owning company
Examples	Linux, RISC-V, Git, Python	Windows, macOS, ARM, x86

Arithmetic Operations

- Every computer must be able to perform arithmetic operation.
- Add, subtract, add immediate are the basic arithmetic operations! They all have three operands
  - Two sources and one destination



All arithmetic operations have this form

- *Design Principle*
- *1:* Simplicity favors regularity
  - Regularity makes implementation simpler
  - Simplicity enables higher performance at lower cost

Arithmetic Example

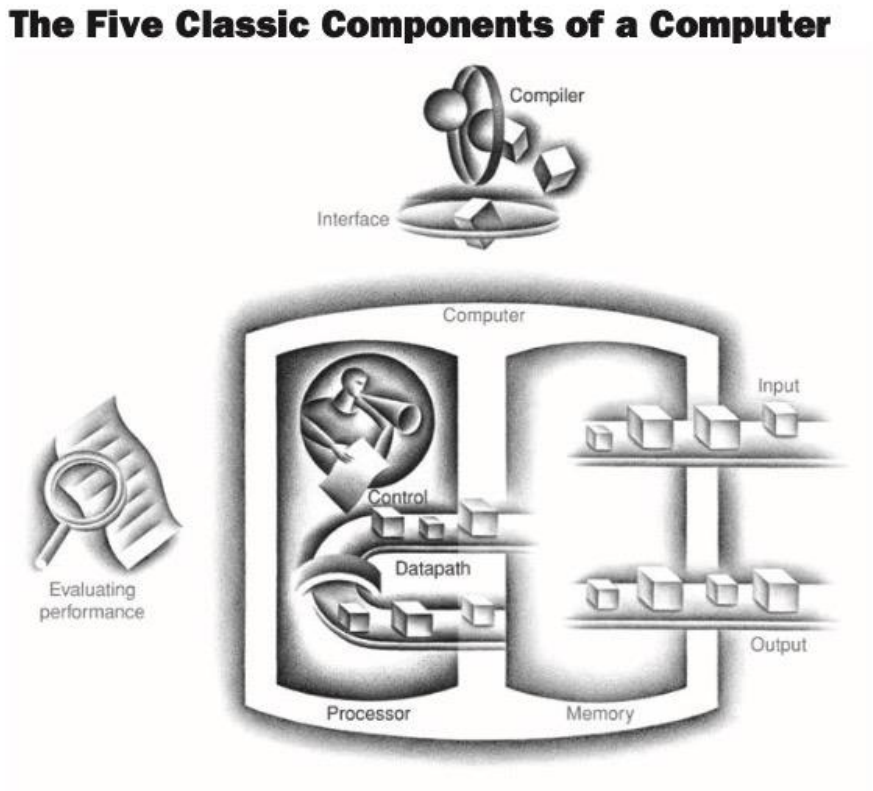
- C code:

result = (x +y) + (a - b);

- Compiled RISC-V code:

```
add t0, x, y          // temp t0 = x + y
sub t1, a, b          // temp t1 = a - b
add result, t0, t1 // result = t0 + t1
```

Computer components



Types of storage

Aspect	Register	Cache	Main Memory (RAM)	Secondary Memory
Purpose	Holds operands or instructions for immediate processing by the CPU.	Stores frequently accessed data/instructions to reduce access time to RAM.	Holds instructions and data for running programs.	Stores large volumes of data for long-term use.
Size	Very small (32–64 bits per register).	Small (kilobytes to megabytes).	Medium (gigabytes).	Large (terabytes or more).
Speed	Fastest storage available.	Faster than RAM but slower than registers.	Slower than cache but faster than secondary memory.	Slowest compared to registers, cache, and RAM.
Volatility	Volatile (loses data on power loss).	Volatile.	Volatile.	Non-volatile (data persists after power off).
Access Time	Few nanoseconds.	Few nanoseconds to microseconds.	Tens of nanoseconds.	Milliseconds (for HDD) or microseconds (for SSD).
Control	Fully controlled by the CPU.	Managed by hardware for optimal performance.	Accessed via the CPU and managed by the OS.	Accessed and managed via the OS and file systems.
Examples	Program Counter, Accumulator Register.	L1, L2, L3 Cache.	DRAM, SDRAM.	SSDs, HDDs.

Operands of the Computer Hardware

Register Operands

- Arithmetic instructions use register operands

- RISC-V has a 32 × 32-bit register file (32 ×64-bit)
  - Use for frequently accessed data
  - 32-bit data is called a “word”
    - 32 x 32-bit general purpose registers x0 to x31
  - 64-bit data is called a “doubleword”
- *Design Principle 2:* Smaller is faster
  - c.f. main memory: millions of locations

RISC-V Registers

RISC-V convention is x followed by the number of the register

- x0: the constant value 0
- x1: return address
- x2: stack pointer
- x3: global pointer
- x4: thread pointer
- x5 – x7, x28 – x31: temporaries
- x8: frame pointer
- x9, x18 – x27: saved registers
- x10 – x11: function arguments/results
- x12 – x17: function arguments

Register Operand Example

- **It is the compiler's job to associate program variables with register**
- C code:

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

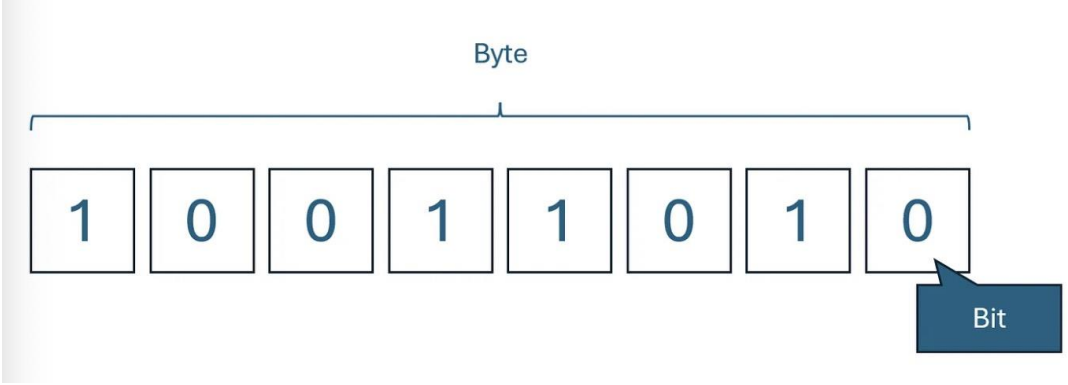
  - f, …, j in x19, x20, …, x23

- Compiled RISC-V code:

```
add x5, x20, x21
add x6, x22, x23
sub x19, x5, x6
```

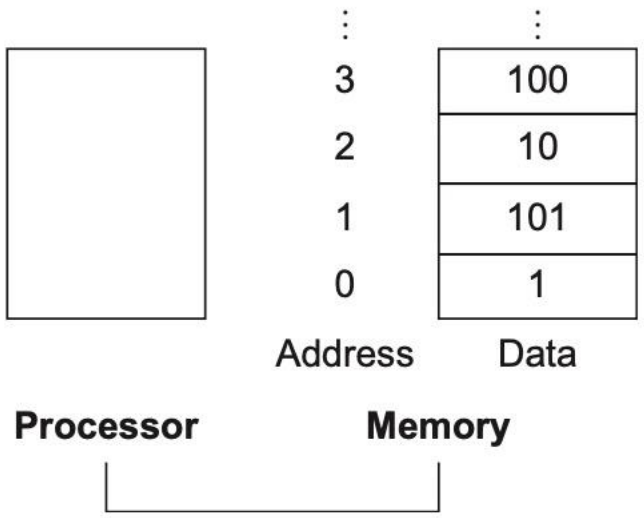
Memory Operands

- 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



Memory Operands

- RISC-V is Little Endian
  - Least-significant byte at least address of a word
  - *c.f.* Big Endian: most-significant byte at least address



Consider the 32-bit hexadecimal number 0x12345678

0X12 most significant byte , 0X78 least significant byte

78	56	45	12

Memory Operands

- RISC-V does not require words to be aligned in memory
  - Unlike ARM instruction set（must be aligned in memory）

32 bits memory

Address	Data (Bytes in Little Endian)				
	+0	+1	+2	+3	
0x1000	0x78	0x56	0x34	0x12	// A[0] = 0x12345678
0x1004	0xAB	0xCD	0xEF	0x01	// A[1] = 0x01EFCDAB
0x1008	0x45	0x67	0x89	0x23	// A[2] = 0x23896745
0x100C	0xDE	0xAD	0xBE	0xEF	// A[3] = 0xEFBEADDE

Aligned VS Unaligned

- Address Data
- 0x1000 0x78
- 0x1001 0x56
- 0x1002 0x34
- 0x1003 0x12
- 0x1004 0xAB
- 0x1005 0xCD
- 0x1006 0xEF
- 0x1007 0x01

Aligned:
lw x5, **4**(x10) // Load word at address 0x1004 into x5， the number should be 4, 8, 12,16 ……

0x 01EFCDAB //

Unaligned:
lw x5, 3(x10) // Load word at address 0x1003 into x5
0x EFCDAB12

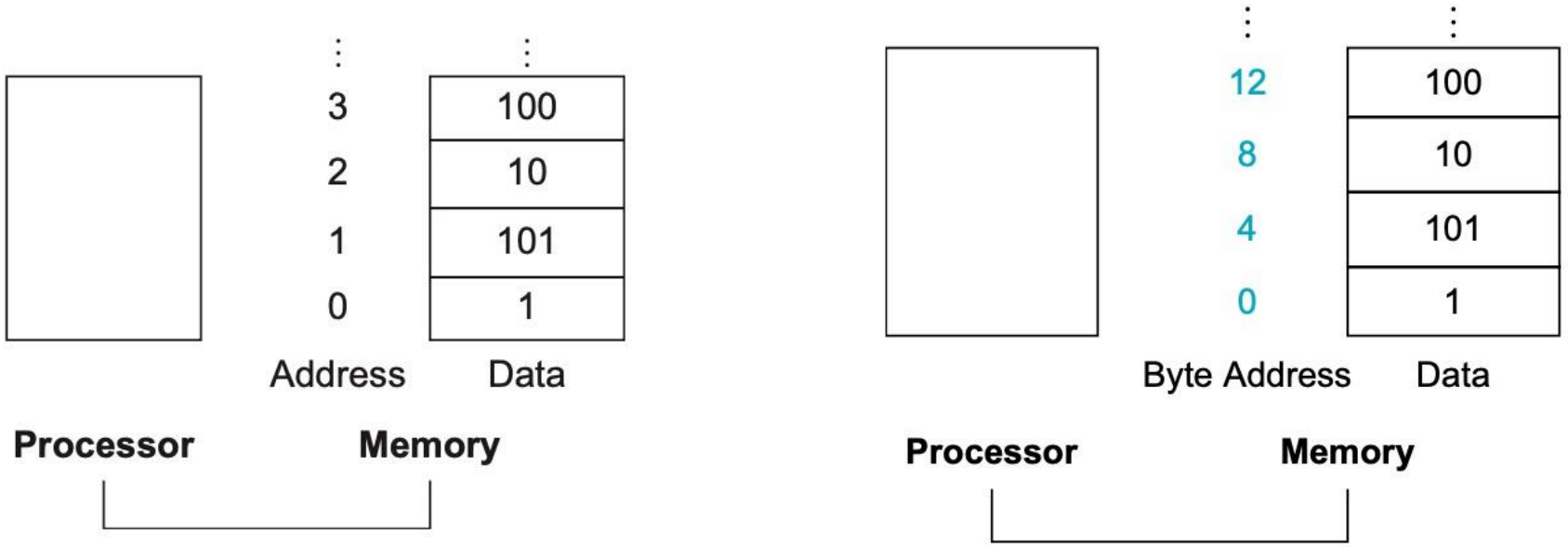
Memory Operand Example 1:

- Let's assume that A is an array of 100 words and that the compiler has associated the variables g and h with the registers x20 and x21 as before. Let's also assume that the starting address, or base address, of the array is in x22. Compile this C assignment statement.

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

```
lw x9, 8(x22)          // Temporary reg x9 gets A[8]
add x20, x21, x9       // g = h + A[8]
```

Memory Operand Example 1:



Memory Operand Example 2

- C code:

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

  - h in x21, base address of A in x22
- Compiled RISC-V code:
  - Index 8 requires offset of 32
    - 4 bytes per doubleword

```
lw      x9, 32(x22) \\Temporary reg x9 gets A[8]
add     x9, x21, x9  \\ Temporary reg x9 gets h + A[8]
sw      x9, 48(x22)   \\  Stores h + A[8] back into A[12]
```

Immediate Operands

- Constant data specified in an instruction

```
addi x22, x22, 4
```

- Make the common case fast
  - Small constants are common
  - Immediate operand avoids a load instruction

Attendance Number

0