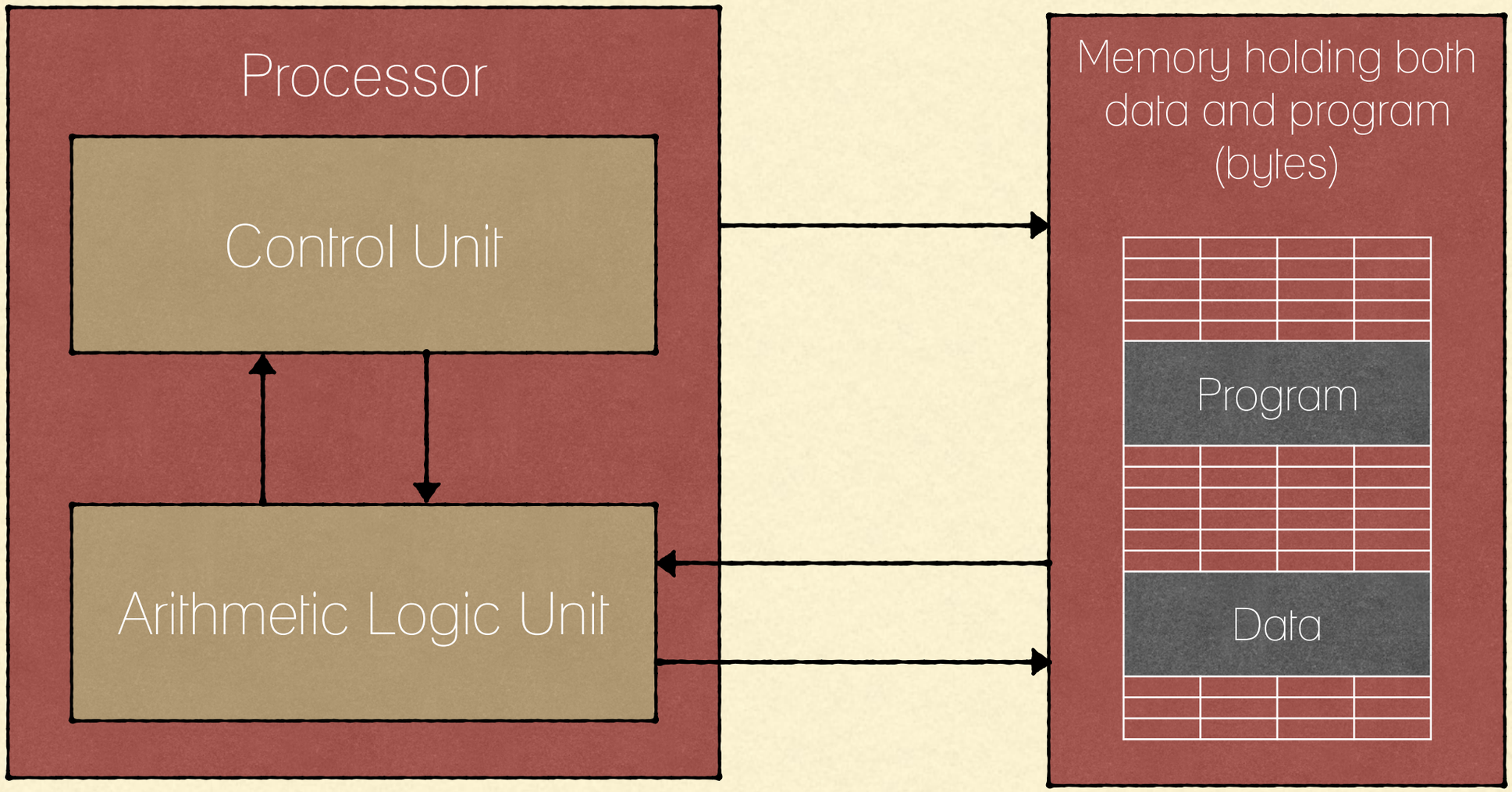
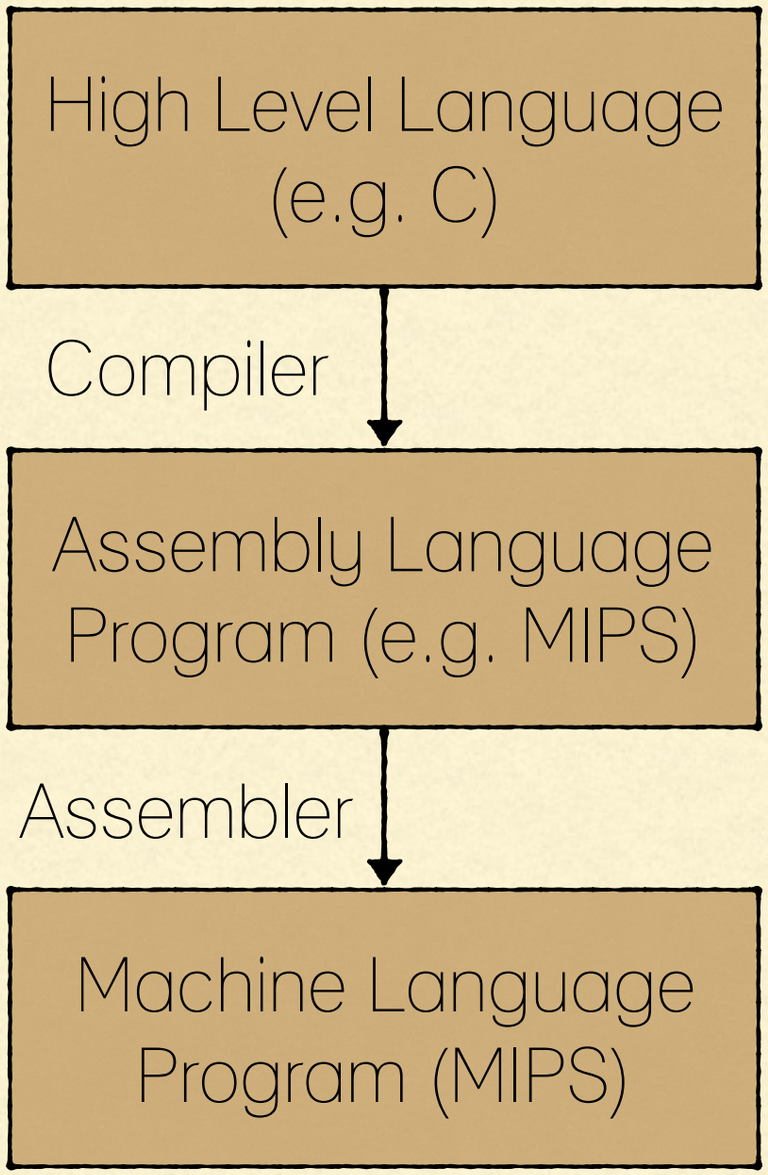
MIPS ASSEMBLY PROGRAMMING LANGUAGE PART I

Ayman Hajja, PhD

THE VON NEUMANN ARCHITECTURE



LEVELS OF REPRESENTATION




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw   $t0, 0($2)
lw   $t1, 4($2)
sw   $t1, 0($2)
sw   $t0, 4($2)
```

```
0000 1001 1001 0110 1010 1111 0101 1000
1111 1001 0000 1001 0000 1010 1111 0101
1001 1010 1111 0101 1000 1010 1111 0101
1001 1001 1010 1111 0101 1000 1010 1111
```

ASSEMBLY LANGUAGE

- Assembly language is a low-level programming language for a computer or other programmable device
- Each assembly language is specific to a particular computer architecture, which is not the case for high-level programming languages
  - Each family of processor chip (MIPS, PIC, SPARC, Alpha, Motorola, Intel, et al.) has its own architecture
  - Each type of processor has its own assembly language.

ASSEMBLY LANGUAGE

- Assembly language is converted into executable machine code by a program referred to as an assembler

- In pure assembly language, one assembly language statement corresponds to one basic operation of the processor.

MIPS ASSEMBLY LANGUAGE

- The MIPS chip was designed from the ground up in 1985.
- MIPS Technologies (formerly MIPS Computer Systems) is a semiconductor company that built one of the first commercial RISC architectures.
- Why MIPS instead of Intel x86?
  - MIPS is simple, elegant. Don't want to get bogged down in gritty details.

RISC: REDUCED INSTRUCTION SET COMPUTING

- The RISC Approach:
  - RISC processors only use simple instructions that can be executed within one (or few) clock cycle(s)
- The CISC Approach:
  - The primary goal of CISC architecture is to complete a task in as few lines of assembly as possible. This is achieved by building processor hardware that is capable of understanding and executing a series of operations.

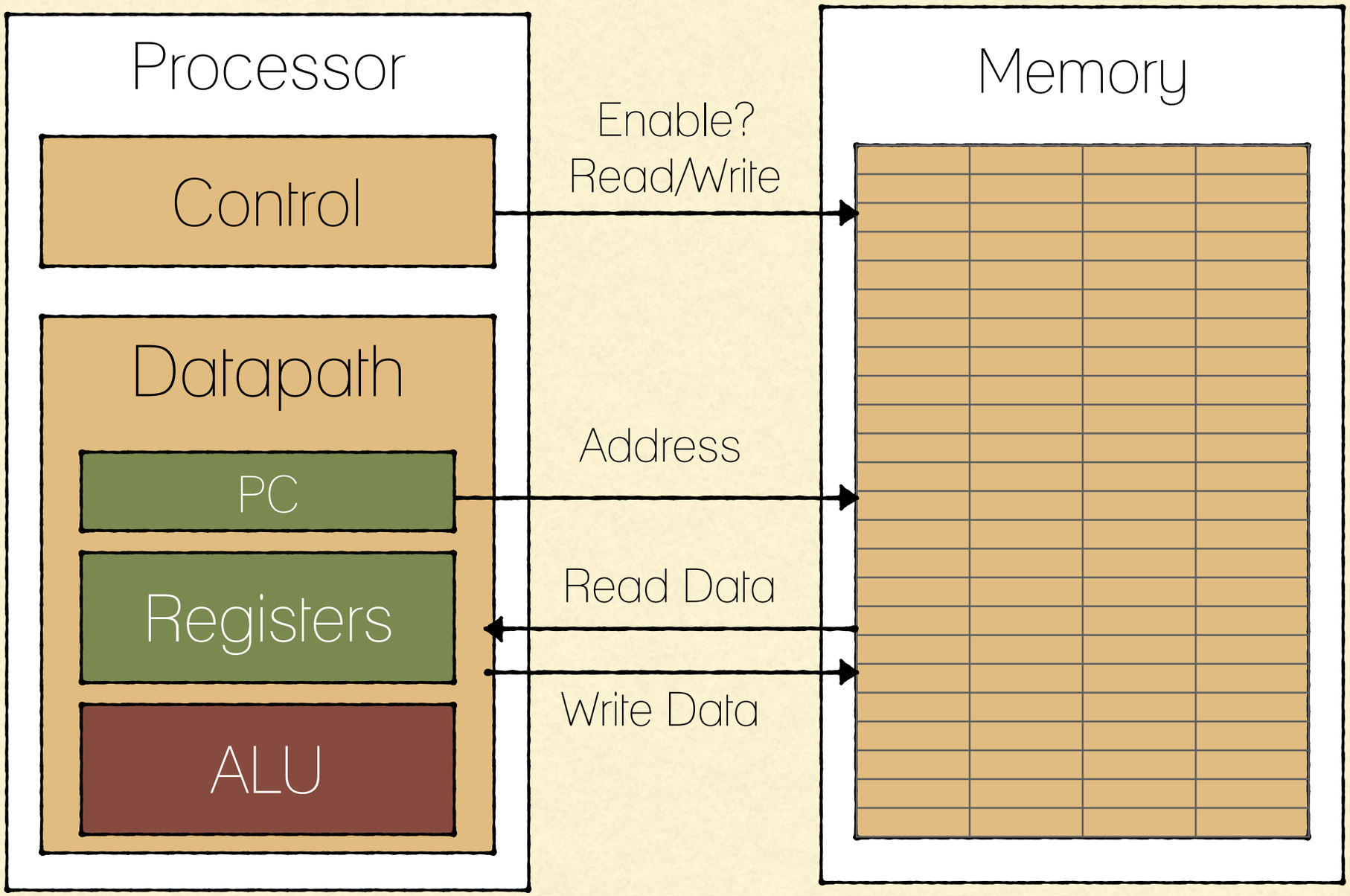
MEMORY ADDRESSES ARE IN BYTES

- 8 bit chunk of bits is called a byte
  - 1 word is 4 bytes
- Word addresses are 4 bytes apart (more on that in future)
- To say that memory is byte-addressable simply means that, given an "address", this address refers to a single block of 8 bits (or a byte)

BASIC MACHINE CODE

- The machine cycle of most processor chips looks like the following:
  1. Fetch the Instruction. The instruction is fetched from memory. The program counter contains the address of the instruction in memory.
  2. Increment the Program Counter. The program counter now points to the next instruction.
  3. Execute the Instruction. The operation asked for by the current machine instruction is performed.
     - On a 32-bit processor, memory addresses are 32 bits wide and so the program counter (PC) holds a 32 bit address.

MEMORY ADDRESSES ARE IN BYTES



REGISTERS

- Unlike C or Java, assembly does not use variables
- Assembly operands are registers:
  - Limited number of special locations built directly into the hardware
  - Operations can only be performed on registers
- Since registers are are directly built in the CPU, they are very fast (100 to 500 times faster than main memory)

REGISTERS

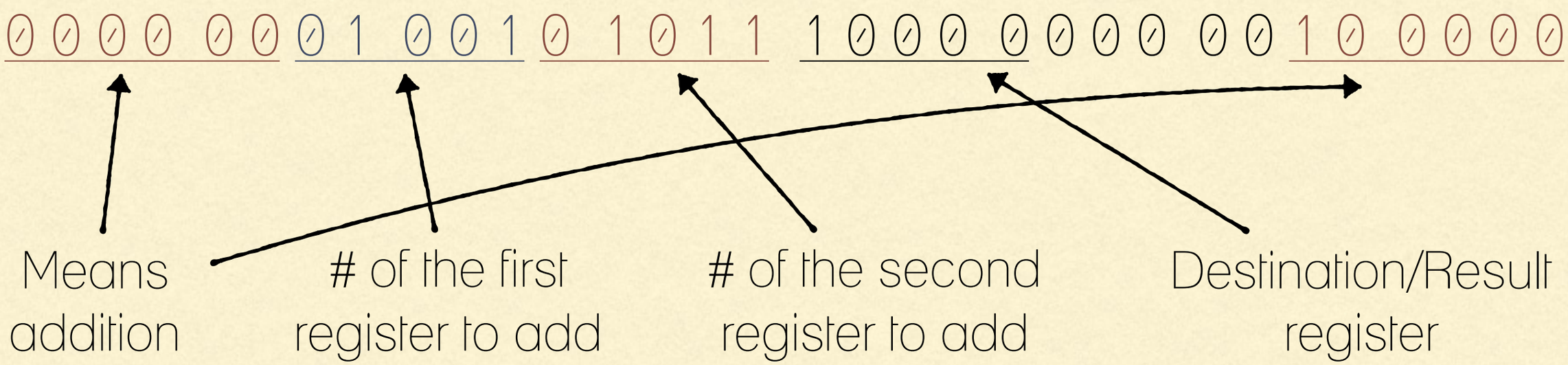
- Since registers are directly built in the CPU, there is a predetermined number of them:
    - In MIPS, we have 32 general-purpose registers, and few other special-purpose registers

REGISTERS

- Registers are numbered from 0 to 31
- Each register can be referred to by a number or name
- Number references:
  - $0, $1, $2, …, $30, $31
- For now:
  - $16 to $23 will be referred to by $s0 to $s7 (variables)
  - $8 to $15 will be referred to by $t0 to $t7 (temp variables)
- In general, use names to make your code more readable

MACHINE INSTRUCTIONS

- A machine instruction is a pattern of bits that directs the processor to perform one machine operation.
- Here is the machine instruction that directs the MIPS processor to add two 32-bit registers together (a register is a part of the processor that holds a bit pattern).



MIPS INSTRUCTIONS: ADD (REGISTER INSTRUCTION)

- Addition in Assembly:
  - Example 1: add $s0, $s1, $s2 (in MIPS)
    - Equivalent to a = b + c; (in C), assuming that:
      - $s1 contains the value of b
      - $s2 contains the value of c
      - and $s0 will be used to store the result (equivalent to 'a')