

# Bytes and words

---

- ❑ Remember to be careful with memory addresses when accessing words.
- ❑ For instance, assume an array of words begins at address 2000.
  - The first array element is at address 2000.
  - The second word is at address 2004, not 2001.
- ❑ For example, if \$a0 contains 2000, then

```
lw $t0, 0($a0)
```

accesses the first word of the array, but

```
lw $t0, 8($a0)
```

would access the *third* word of the array, at address 2008.

# Loading and storing bytes

---

- ❑ The MIPS instruction set includes dedicated load and store instructions for accessing memory.
- ❑ The main difference is that MIPS uses **indexed addressing**.
  - The address operand specifies a signed constant and a register.
  - These values are added to generate the effective address.
- ❑ The MIPS “load byte” instruction **lb** transfers one byte of data from main memory to a register.

```
lb $t0, 20($a0)      # $t0 = Memory[$a0 + 20]
```

question: what about the other 3 bytes in \$t0?

Sign extension!

- ❑ The “store byte” instruction **sb** transfers the lowest byte of data from a register into main memory.

```
sb $t0, 20($a0)      # Memory[$a0 + 20] = $t0
```

# Loading and storing words

---

- ❑ You can also load or store 32-bit quantities—a complete **word** instead of just a byte—with the **lw** and **sw** instructions.

```
lw $t0, 20($a0)      # $t0 = Memory[$a0 + 20]
sw $t0, 20($a0)      # Memory[$a0 + 20] = $t0
```

- ❑ Most programming languages support several 32-bit data types.
  - Integers
  - Single-precision floating-point numbers
  - Memory addresses, or pointers
- ❑ Unless otherwise stated, we'll assume words are the basic unit of data.

# Computing with memory

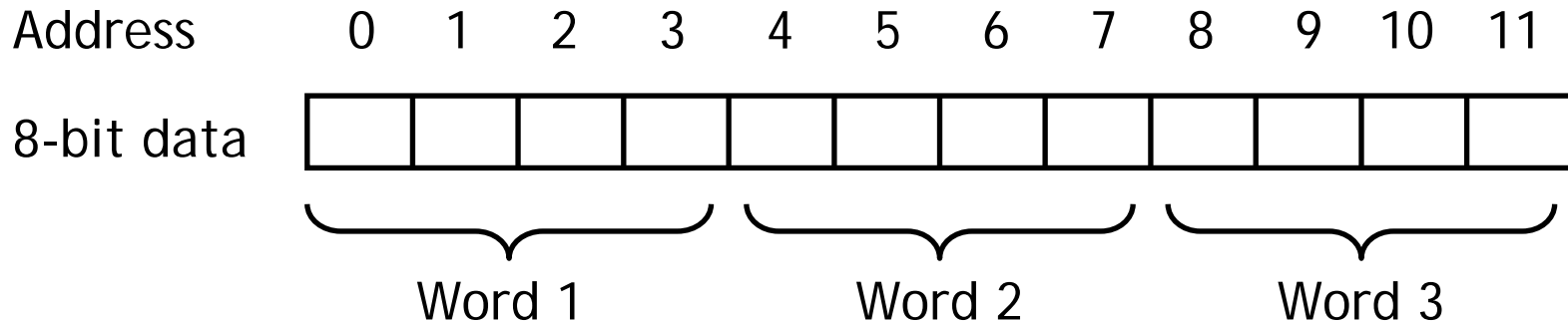
---

- ❑ **So, to compute with memory-based data, you must:**
  1. Load the data from memory to the register file.
  2. Do the computation, leaving the result in a register.
  3. Store that value back to memory if needed.
- ❑ **For example, let's say that you wanted to do the same addition, but the values were in memory. How can we do the following using MIPS assembly language using as few registers as possible? (Ex. on your own!)**

```
char A[4] = {1, 2, 3, 4};  
  
int result;  
  
result = A[0] + A[1] + A[2] + A[3];
```

# Memory alignment

- ❑ Keep in mind that memory is byte-addressable, so a 32-bit word actually occupies four contiguous locations (bytes) of main memory.



- ❑ The MIPS architecture requires words to be **aligned** in memory; 32-bit words must start at an address that is divisible by 4.
  - 0, 4, 8 and 12 are valid **word addresses**.
  - 1, 2, 3, 5, 6, 7, 9, 10 and 11 are *not* valid word addresses.
  - Unaligned memory accesses result in a **bus error**, which you may have unfortunately seen before.
- ❑ This restriction has relatively little effect on high-level languages and compilers, but it makes things easier and faster for the processor.

# Exercise

## □ Can we figure out the code?

```
swap(int v[], int k);      swap:          ; $5=k $4=v[0]
{ int temp;                sll $2, $5, 2; $2←k×4
    temp = v[k]            add $2, $4, $2; $2←v[k]
    v[k] = v[k+1];        lw $15, 0($2) ; $15←v[k]
    v[k+1] = temp;        lw $16, 4($2) ; $16←v[k+1]
}                           sw $16, 0($2) ; v[k]←$16
                           sw $15, 4($2) ; v[k+1]←$15
                           jr $31
```



Assuming  $k$  is stored in  $\$5$ , and the starting address of  $v[]$  is in  $\$4$ .

# Pseudo-instructions

- ❑ MIPS assemblers support **pseudo-instructions** that give the illusion of a more expressive instruction set, but are actually translated into one or more simpler, “real” instructions.
- ❑ In addition to the **la** (load address) we saw on last lecture, you can use the **li** and **move** pseudo-instructions:

```
li      $a0, 2000      # Load immediate 2000 into $a0
move    $a1, $t0       # Copy $t0 into $a1
```

- ❑ They are probably clearer than their corresponding MIPS instructions:

```
addi    $a0, $0, 2000  # Initialize $a0 to 2000
add     $a1, $t0, $0    # Copy $t0 into $a1
```

- ❑ We'll see lots more pseudo-instructions this semester.
  - A core instruction set is given in “Green Card” of the text (1<sup>st</sup> page).
  - Unless otherwise stated, you can always use pseudo-instructions in your assignments and on exams.