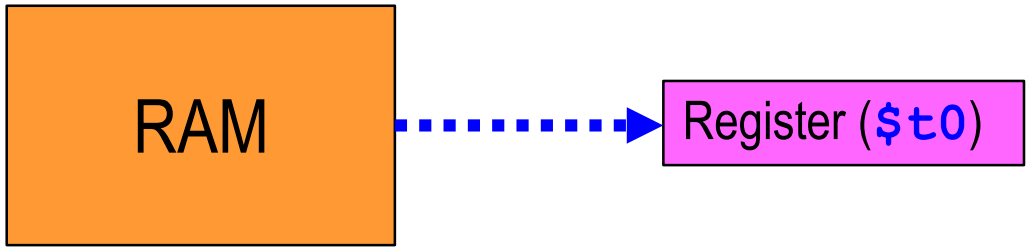
BYTE ADDRESING

MIPS Byte Addressing

byte endianness

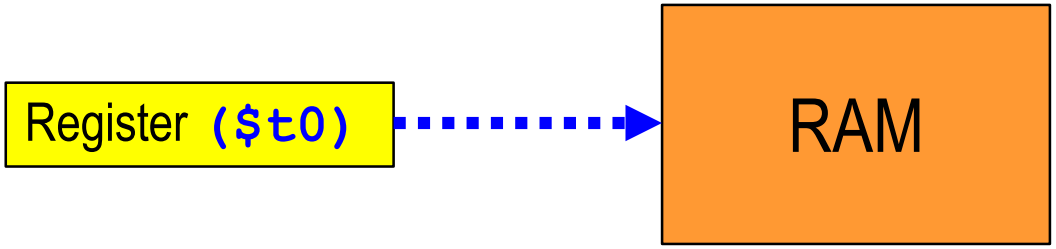
Accessing the RAM

- **Load instructions**: Read data from RAM and copy it to a register. (`lw $t0`, memory-address)



- **Store instructions**: Write data from a register to RAM. (`sw $t0`, memory-address)

.



MIPS memory access instructions

- **WORD [32 (64-bits)]**
- **BYTE [8-bits].**

MIPS memory access instructions

- **WORD (Address in memory must be word-aligned)**
  - **`lw`;** Loads a word from a location in memory to a register
  - **`sw`;** Store a word from a register to a location in memory
- **BYTE (Address not aligned-**Only one byte is loaded from memory)
  - **`lb`;** Loads a byte from a location in memory to a register. Sign extends this result in the register.
  - **`sb`;** Store the least significant byte of a register to a location in memory.

Memory alignment

MIPS requires that all words start at byte addresses and are multiples of 4 bytes (4 x 8 = 32-bits)

4	3	2	1
C	8	4	0

Aligned

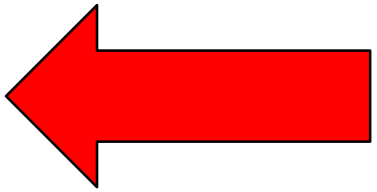
Address in 0,4,8, C in Hexadecimal

Load Word and Load Byte

- **`lw $rt, offset($rs)`**

EA ← $(s) + sign-ext$_{32}$(offset)

- **`lb $rt, offset($rs)`**



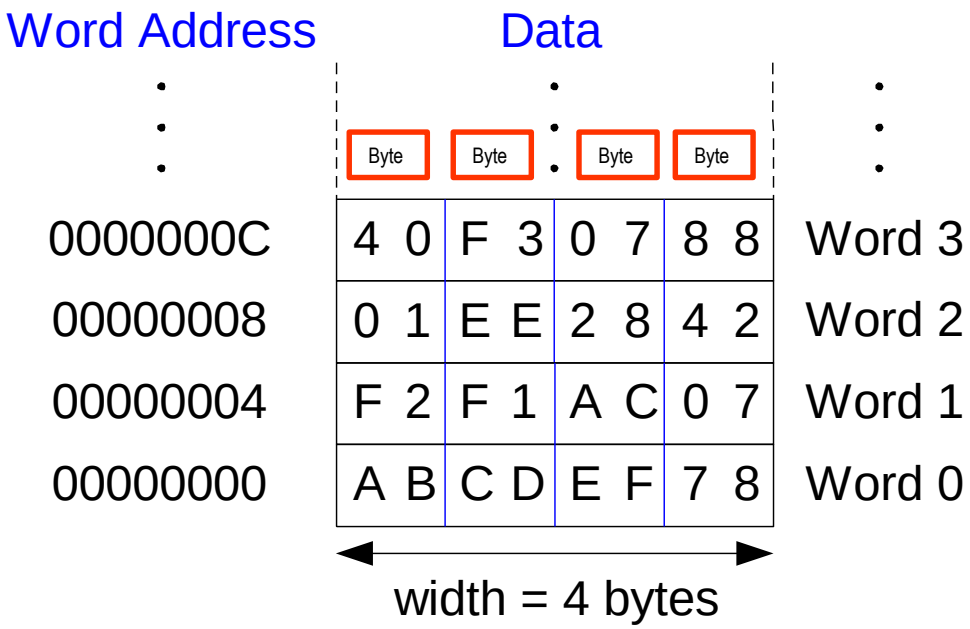
Load … Store >> Byte

- `lb $rt, offset($rs)`
  - `lb register(destination), RAM(source)`
  - copy byte at source RAM location to low-order byte of destination register

- `sb $rs, offset($rt)`
  - `sb register(source), RAM(destination)`
  - store byte (low-order) in source register into RAM destination.

Byte-Addressable Memory

- Each data byte has unique address
- Load/store words or single bytes: load byte (**`lb`**) and store byte (**`sb`**)
- 32-bit word = 4 bytes, so word address increments by: +4.



Load Byte (Memory Read)

```
lb
```

Reading Byte-Addressable Memory

- The address of a memory word must now be multiplied by 4. [4-bytes = 1-word MIPS = 32-bits]
- For example:
  - the address of memory word 0 is 0 × 4 = 0
  - the address of memory word 1 is 1 × 4 = 4
  - the address of memory word 2 is 2 × 4 = 8
  - the address of memory word 3 is 3 × 4 = 12
  - the address of memory word 4 is 4 × 4 = 16

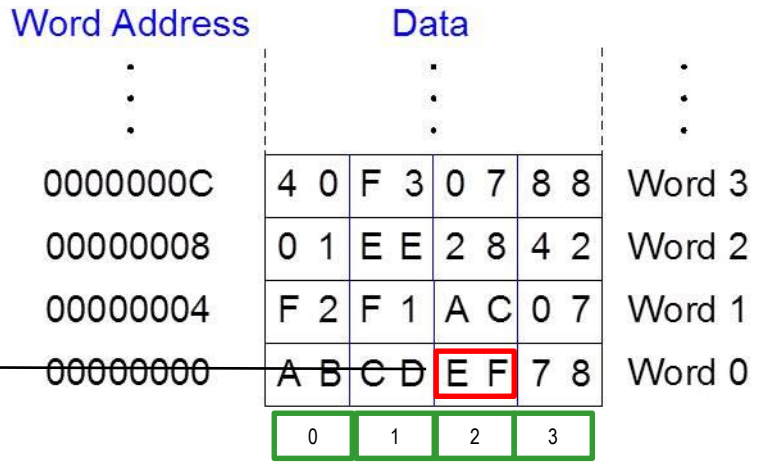
Reading Byte-Addressable Memory

- **Example:** Load a byte of data at memory address 2 into `$s3`.
- **Effectice-Address**:
  `$s3` ← `Mem[$t0+2]=0x00000000+2=0x00000002`
- `$s3` holds the value 0xEF after load; (with signExtension:`0xEEEEEEEF`)

MIPS assembly code

```
                                $t0 = 0x00000000 (base address)
lb $s3, 2($t0)  # read byte at address 2 into $s3
```

Register File	
Reg	value
$t0	
….	
$s3	0xEEEEEEEF



Store Byte (Memory Write)

sb

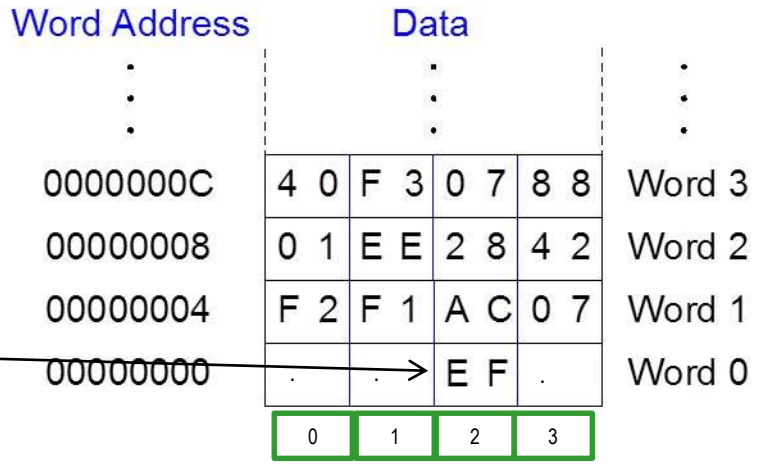
Writing Byte-Addressable Memory

- **Example:** Load the byte 2 of data from the register **$s3**, into the main memory.
- **Effective-Address**:
  Mem[$t0+2] ➔ **0x00000000+0x2 = 0x00000002**
- To the above address load the byte = **2**

MIPS assembly code

```
$t0 = 0x00000000 (base address)
sb $s3, 2($t0)    # Write byte to address 2 from $s3
```

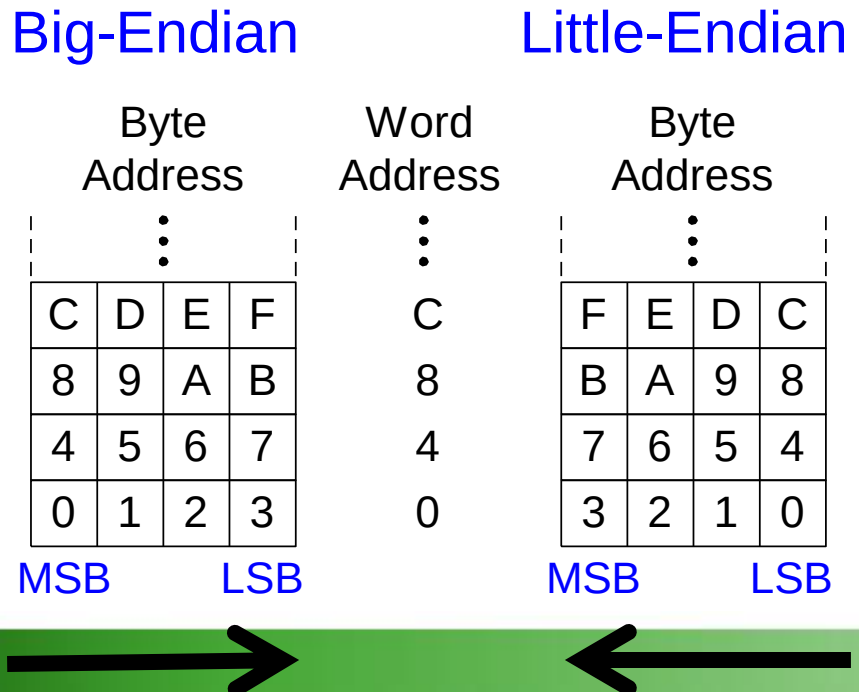
Register File	
Reg	values
$t0	
….	
$s3	0xABCDEF78



Left to Right (Big Endian) or Right to left?

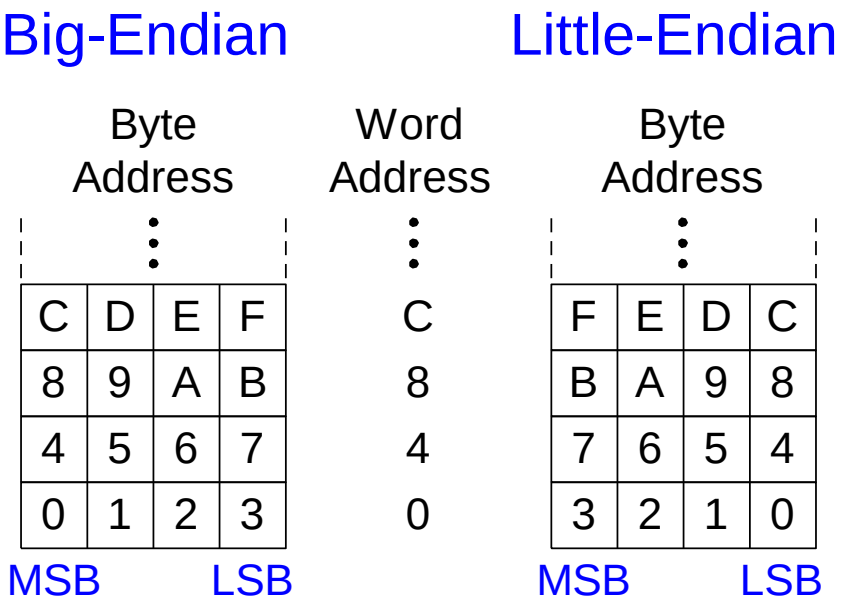
Big-Endian & Little-Endian Memory

- How to number bytes within a word?
- **Little-endian:** byte numbers start at the **little** (least significant) end
- **Big-endian:** byte numbers start at the **big** (most significant) end
- **Word address** is the **same** for big- or little-endian



Big-Endian & Little-Endian Memory

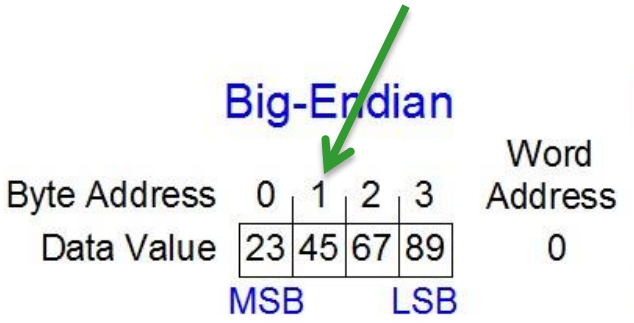
- Jonathan Swift's *Gulliver's Travels*: the Little-Endians broke their eggs on the little end of the egg and the Big-Endians broke their eggs on the big end …
- It doesn't really matter which addressing type used – except when the two systems need to share data!



Example …

Big-Endian Memory

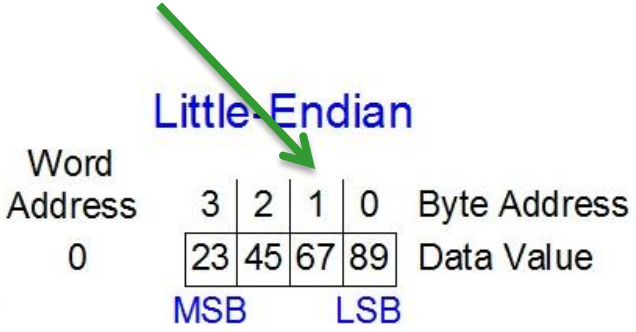
- Suppose `$s0` initially contains: `0x23456789`
- After the following code runs on a Big-Endian system, what value is in register `$s0`?
  - `sw $s0, 0($0)`
  - `lb $s0, 1($0)`
- The above instruction `lb $s0, 1($0),` loads the data at byte address `(1+$0) = 1,` which is `0x45`, into the LSB of `$s0.`



- Therefore after the instruction: `lb $s0,1($0)`
  `Register:` `$s0` would contain `0x00000045`

Little-Endian Memory

- Suppose `$s0` initially contains: `0x23456789`
- After the following code runs on a Little-Endian system, what value is in register `$s0`?
  - `sw $s0, 0($0)`
  - `lb $s0, 1($0)`
- The above instruction `lb $s0, 1($0)` loads the data at byte address `(1+$0) = 1,` which is `0x67`, into the LSB of `$s0.`



- Therefore after the instruction: `lb $s0,1($0).`
  Register: `$s0` would contain: `0x00000067`

Big and Little Endian systems

- Little-Endian: **CISC**
- Big-Endian: **RISC**
- **RISC/MIPS** can use **both** **byte** endianness.