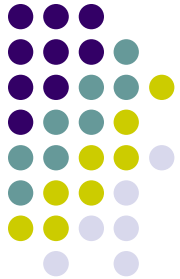
Little and Big endian



- Let us take that memory width on some processor is 8 bits, i.e. that every 8 bits have their own address.

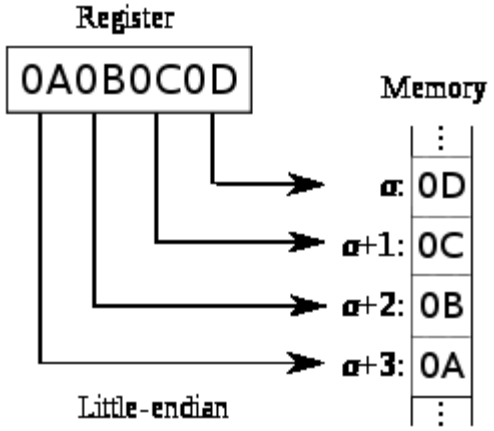
  Question: How are we going to store 32bit value in such memory?

  Answer: We will split 32bit value into 4 8bit values, e.g. value 0x0A0B0C0D could be split into these four parts:

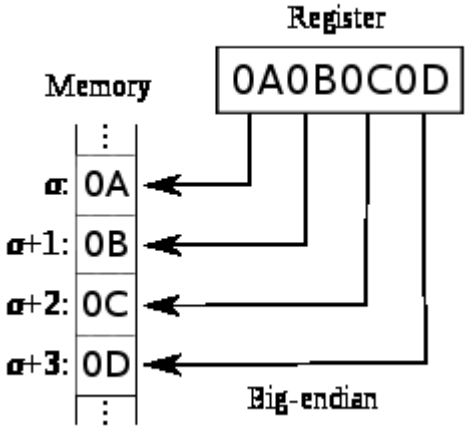
  0x0A, 0x0B, 0x0C and 0x0D, ordered from the most significant to the least significant part.
- There two equally logical ways to store those parts in the memory:
  - Parts of lower significance go on lower addresses – Little endian
  - Parts of higher significance go on lower addresses – Big endian
- Endianness mostly depends on the processor itself. For example:
  - MIPS – you can choose LE and BE
  - Intel – LE
  - ARM – LE or BE, depending on the build

Little and Big endian

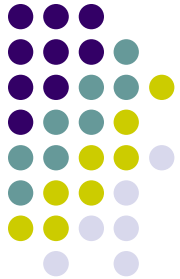
- Part of the **lowest** significance is written first = **little** endian



- Part of the **highest** significance is written first = **big** endian



Little and Big endian
Why it makes difference



1. Store 0x0A0B0C0D in memory as a 32bit value (for example, of int type)
2. Read from those memory locations as two consecutive 16bit values
3. Read from those memory locations as four consecutive 8bit values.
4. Read values:

   16bit access:
   - **LE**: 0x0C0D и 0x0A0B
   - **BE**: 0x0A0B и 0x0C0D

   8bit access:
   - **LE**: 0x0D, 0x0C, 0x0B и 0x0A
   - **BE**: 0x0A, 0x0B, 0x0C и 0x0D