Princípios da Computação

Status register.
Little endian and Big endian.

isep Instituto Superior de Engenharia do Porto

Status register

Status register

- A status register is a set of status flags that provide information about the status of the processor.

- Typically, flags are updated (set or cleared) upon an instruction executed by the Arithmetic Logic Unit.

- Status flags allow an instruction to act based on the result of the previous instruction.

Status register

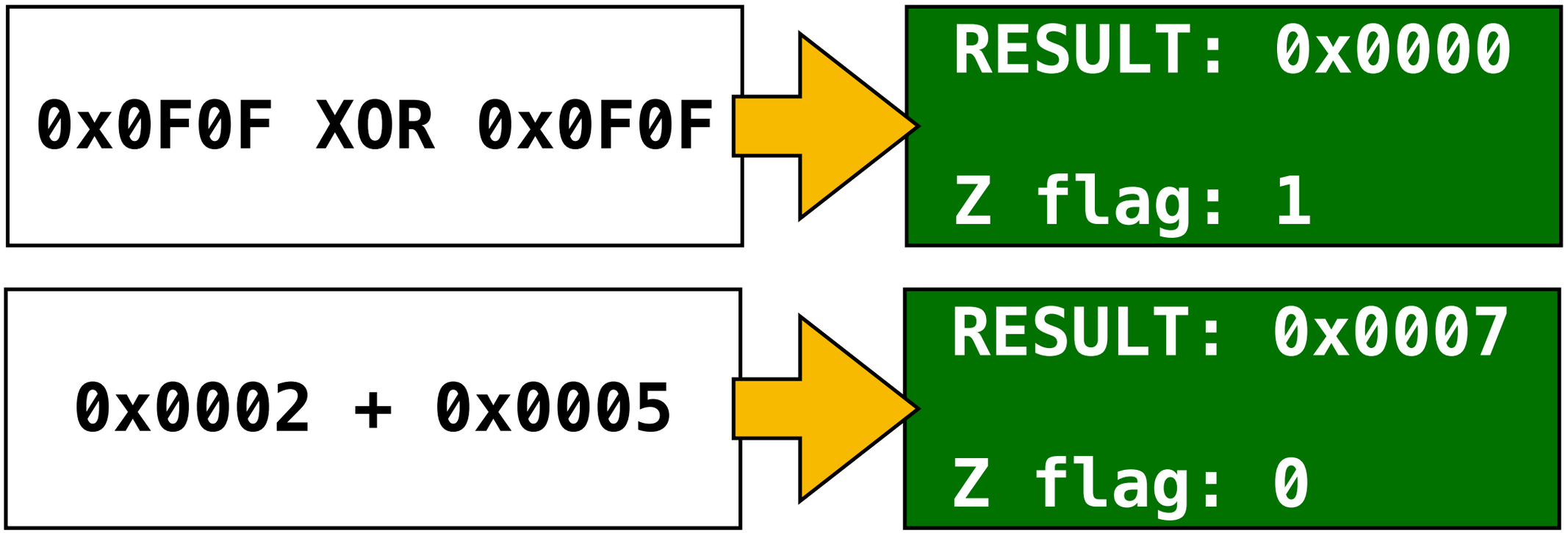
- A status register is present on several architectures, such as x86 and ARM.
    - Provides information about the last ALU instruction, directly by hardware.
- But other architectures (e.g. MIPS) do not support a status register.
    - Relevant information about the previous ALU instruction must be inspected by software.

Common status flags

FLAG	Name	Description
Z	*Zero*	The result of the last ALU instruction is zero. **Value: 0** if not zero; **1** if zero.
S	*Sign*	The result of the last ALU instruction is negative, if signed. **Value: 0** if positive; **1** if negative.
C	*Carry*	An unsigned arithmetic carry (sum) or borrow (subtraction) has been generated out of the most significant ALU bit position.
O	*Overflow*	The signed two's-complement result does not fit in the number of bits used for the result.

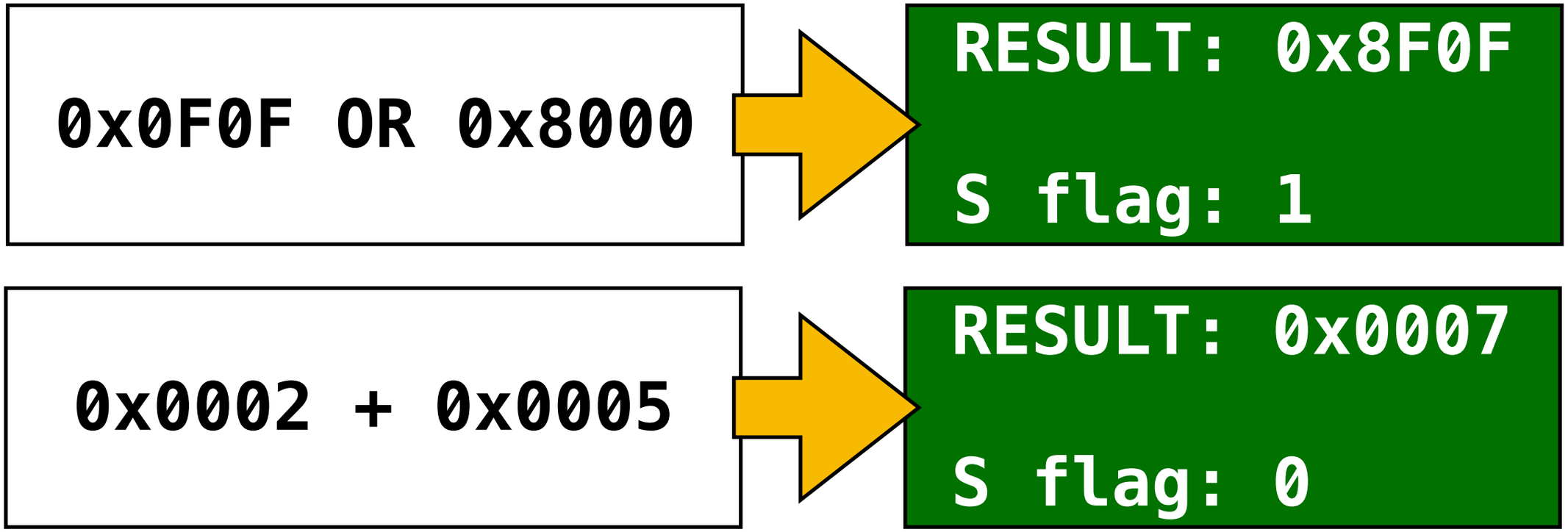
ZERO flag

- The Zero flag is set to 1 when the result of an ALU operation results in a set of exclusively zeros.



SIGN flag

- The Sign flag is set to 1 when the most significant bit of the result of an ALU operation is 1 (negative?).



When results exceed the size…

- Computers don't distinguish between signed and unsigned binary numbers.
  - This characteristic makes logic circuits fast!
- However, the burden of distinguishing between signed and unsigned lies on the programmers' side.

When results exceed the size…

- Arithmetic operations have the potential to run into a condition known as overflow.

- Overflow occurs with respect to the size of the data type that must accommodate the result.

- The correct approach to detect overflow is to consider two separate cases:

    - Overflow when adding unsigned numbers.

    - Overflow when adding signed numbers.

CARRY flag

- The Carry flag is useful when adding or subtracting unsigned integers.

- This flag informs that the size used to accommodate the resulting integer is not sufficient.

- However, it enables numbers larger than a single ALU width to be added/subtracted by adding a binary digit from a partial addition/subtraction to the least significant bit position of a more significant word.

CARRY flag

- The Carry flag is set to 1 when an unsigned carry/borrow is generated from the most significant bit position.

0x8000 + 0x8000

(32768 + 32768 = 65536)

RESULT: 0x0000

(0 + 2^{16} carry)

C flag: 1

0x0002 + 0x0005

RESULT: 0x0007

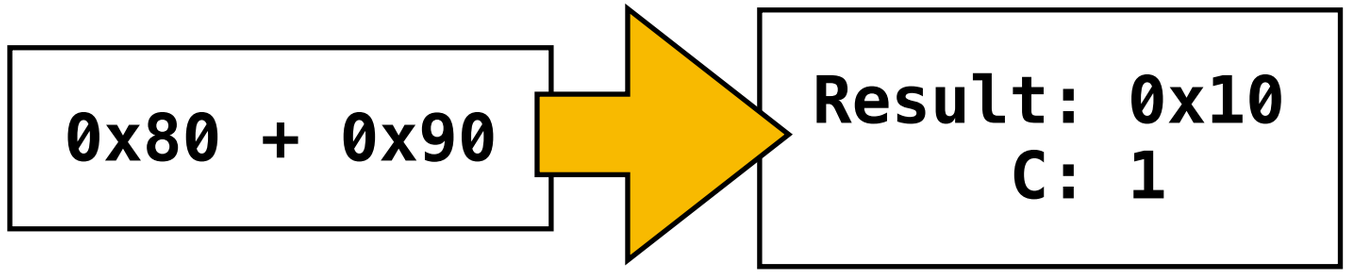
C flag: 0

CARRY flag - example of a partial addition

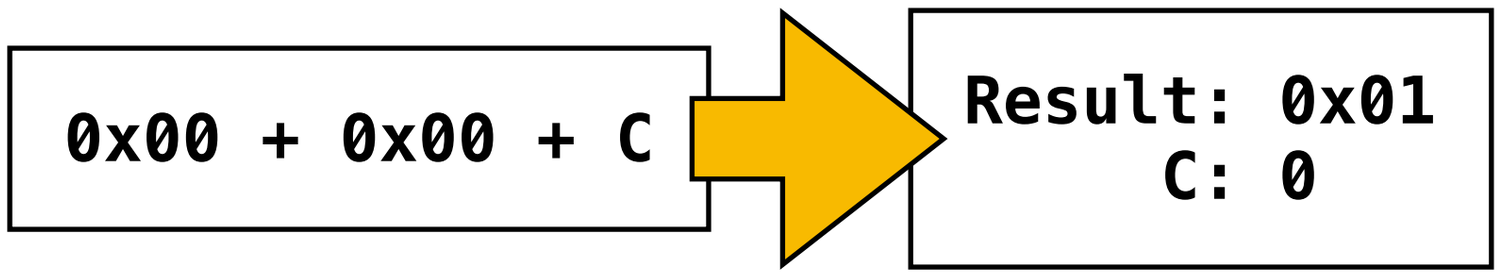
Addition of two 16-bit integers on an 8-bit ALU:

`0x0080 + 0x0090 = 0x0110`

1st step:
add low 8-bits



2nd step:
add high 8-bits with carry



OVERFLOW flag

- The Overflow flag is useful when adding or subtracting signed two's complement integers.
- The result may exceed the size used to accommodate the result:
    - Adding two positive (or two negative) integers may result in a negative (or positive) integer.

OVERFLOW flag

- The Overflow flag is set to 1 when the signed two's complement result exceeds the number of bits used for the result .

0x5000 + 0x5000

20480 + 20480

RESULT: 0xA000

(-24576 : negative!!)

O flag: 1

0x8000 + 0x8000

(-32768) + (-32768)

RESULT: 0x0000

(0 : not negative!!)

O flag: 1

Little endian and Big endian

isep Instituto Superior de Engenharia do Porto

Organising bits in memory cells

- Current memories store 8 bits in each cell.
    - Each cell is accessed through its unique address.
- However, programs usually process numbers with larger sizes:
    - 32-bits and 64-bits are currently the usual sizes for integers, matching the width of popular architectures.
- There is an apparent mismatch between memory and CPU widths!

Splitting bits by bytes

- The solution is to store every value in multiple memory cells:
  - A 32-bit value is stored in 4 memory cells (4 bytes).
  - A 64-bit value is stored in 8 memory cells (8 bytes).

Which chunk first?

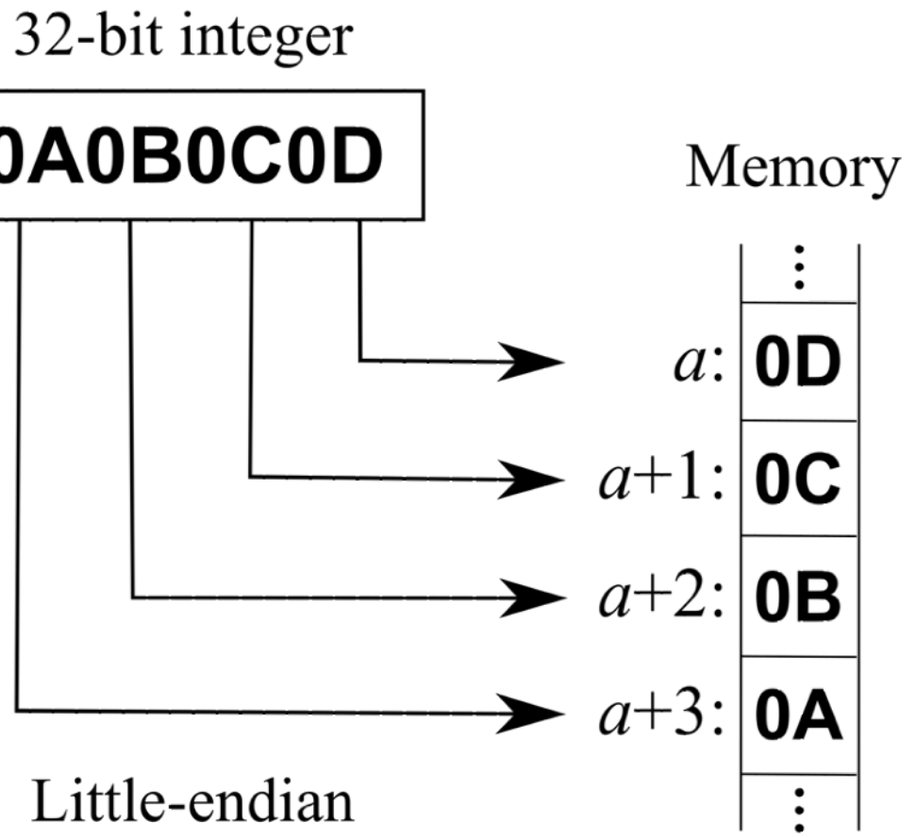
- While the approach is simple (split the value into 8-bit chunks)…
- … in which order should the chunks take in the memory cells?

Which chunk first?

- Two opposite approaches:
  - Little endian
  - Big endian

Little endian

- The Little endian approach stores:
  - the most significant byte of a word in the highest address, and
  - the least significant byte at the lowest address.
- The x86 architecture follows this technique.



Big endian

- The Big endian approach stores:
  - the most significant byte of a word in the lowest address, and
  - the least significant byte at the highest address.
- The 68000 architecture follows this technique.