# MIPS Software Training

## Porting Software

www.mips.com

## Porting C programs

- **Common Problems:**
  - Which end of the egg to eat?
  - Caches and Physical Memory Map
  - Data alignment
  - Short variables
  - Unsigned Characters
  - Bit Fields

# Which end of the egg to eat?

- **Endianness:**
  - MIPS Cores can be set to run in big-endian or little-endian.

MS Byte

| Big-endian | | | | |
|---|---|---|---|---|
| byte | 0 | 1 | 2 | 3 |
| bit | 31 24 | 23 16 | 15 8 | 7 0 |
| | integer | | | |
| | short[0] | | short[1] | |
| | char[0] | char[1] | char[2] | char[3] |

| Little-endian | | | | |
|---|---|---|---|---|
| byte | 3 | 2 | 1 | 0 |
| bit | 31 24 | 23 16 | 15 8 | 7 0 |
| | integer | | | |
| | short[1] | | short[0] | |
| | char[3] | char[2] | char[1] | char[0] |

MS Byte

## Caches and Physical Memory Map

- **Reminder of what we have already gone over:**
  - DMA (with the exception of a Coherent memory with IOCU)
    - Invalidate incoming
    - Flush out going
  - Self-Modifying code
    - Must be synced to instruction cached (synci instruction)
  - Cache Aliases
    - Way larger than the page size can lead to same physical stored twice in the cache (issue if writing data)
  - All Program Memory address are virtual

## Porting C programs

- **Unaligned addresses: will cause an ‘‘Address Error’’**
  - Normal loads and stores in the MIPS architecture must be aligned; half-words may be loaded only from 2-byte boundaries and words only from 4-byte boundaries.
  - Won’t effect most programs since the compiler correctly aligns data.
  - Beware when type-casting pointers of small types into pointers of larger types can cause problems (char to a word could be aligned to a byte boundary which would cause the word to be miss aligned)
    - Use –Wcast-align to catch errors
    - Use _mips_unaligned_init() to install exception handler to catch exceptions
      - Only to test code (too slow for normal use)

## Porting C programs

- **Avoid the use of short variables**
  - There are no MIPS arithmetic instructions which operate on sub 32-bit values, and they have to be synthesized into multiple instructions.
  - Particularly bad if used for, for loop counters and array indices. Although the compiler attempts to avoid excessive conversions, always use ‘‘int’’ for such purposes, unless you specifically need the semantics of 16-bit arithmetic.
  - Moving from 16-bit int: In most cases you can convert up to the MIPS int size of 32 bits, be aware of places where signed comparisons are used to catch 16-bit overflow.

## Porting C programs

- **MIPS compilers default Characters to unsigned!**
  - Example : You may get caught out by mistakes like assigning the integer result of getc() to a char variable, and then comparing that with EOF (which is defined as a −1).
  - If signed is required you need to specify signed char or use compiler option –fsigned-char to change the default

## Porting C programs

- **Bitfields do not default to unsigned in GCC**
  - GCC uses your type definition as written
  - Accessing signed bit fields generates slower code especially for MIPS16
  - To default to unsigned use the −funsigned−bitfields option.