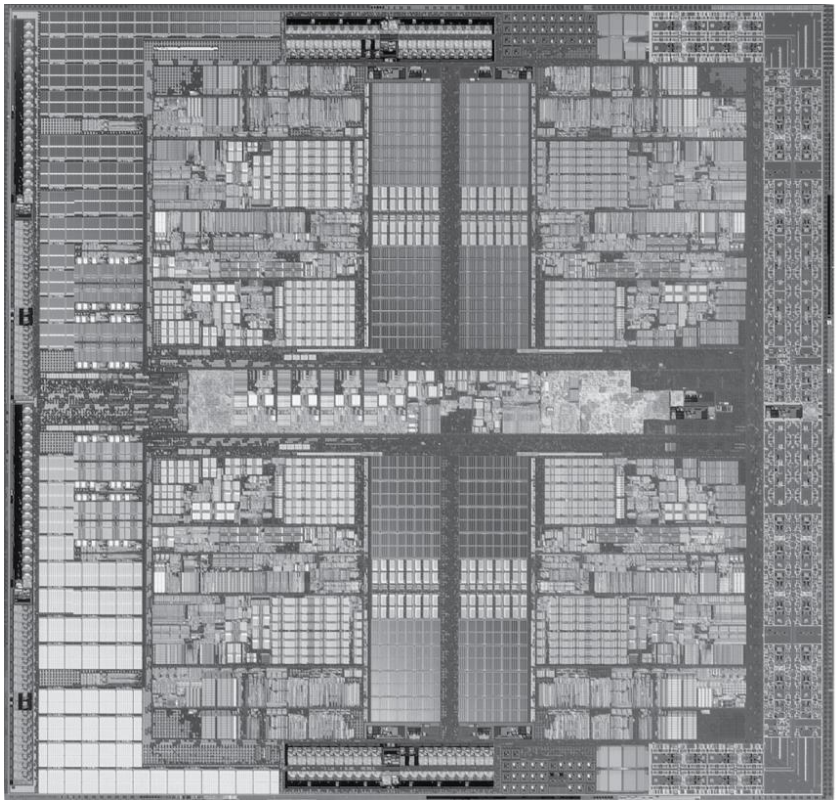
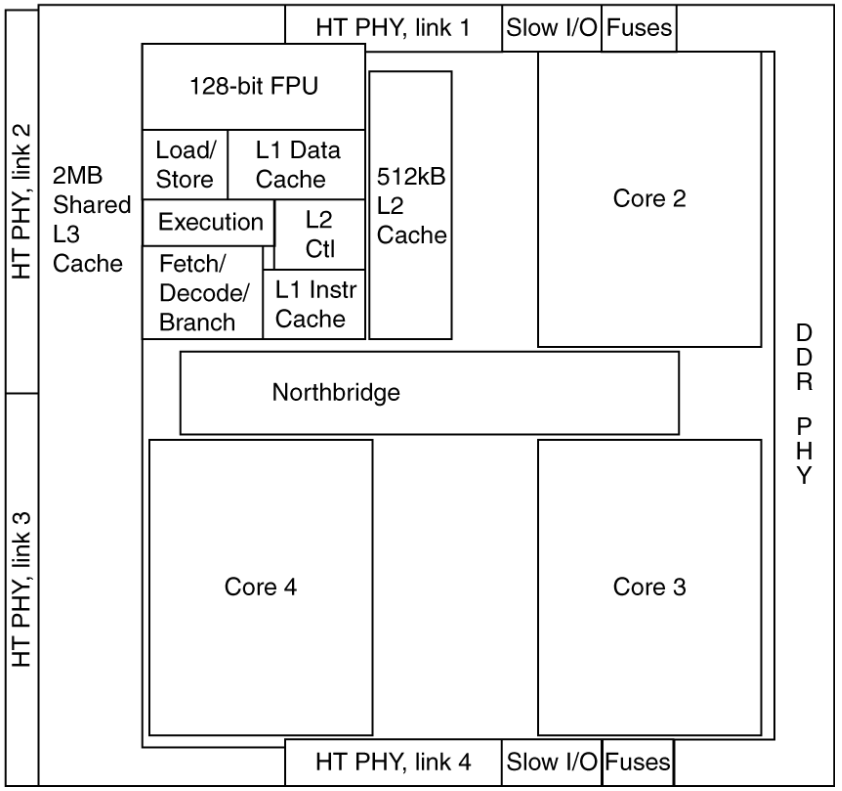
O que vamos estudar hoje?





(AMD Barcelona)

ISA: operações, operandos e representação de instruções

Introdução

- **A linguagem da máquina**
  - **Instruções: “palavras”**
  - **Conjunto de instruções: “vocabulário”**
- **ISA escolhido: MIPS**
  - **Simplicidade**
  - **RISC**
    - » **ISA típico de máquinas contemporâneas (após 1980)**
  - **Similar ao processador mais popular**
    - » **ARMv7 (9 bilhões de unidades fabricadas em 2011)**
  - **Diferente dos usados em PCs e *Cloud***
    - » **Intel x86 (~ 400 milhões de PCs vendidos em 2012)**

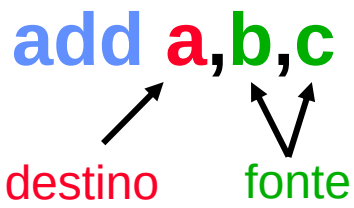
Suporte para operações

- **Operações aritméticas:**
  - **Exemplo: a = b + c + d + e;**

    add a,b,c

    add a,a,d

    add a,a,e



- **Princípio de Projeto 1**
  - **Simplicidade favorece regularidade.**
    - » **Número de operandos fixo no MIPS (3) ⇒**
    - » **HW: mais simples do que se número variável**
    - » **SW: Menor número de decisões tomadas pelo compilador**
- **Operando pode ser valor de variável**
  - **Como armazená-lo ?**

Suporte para operações

- **Exemplo: f = (g+h) - (i+j);**

```
add t0,g,h      # variável temporária t0 contém g+h
add t1,i,j      # variável temporária t1 contém i+j
sub f,t0,t1     # f recebe t0-t1, ou seja, (g+h)-(i+j)
```

- **Operando pode ser valor temporário**
  - **Como armazená-lo ?**
- **Armazenamento em HW**
  - **Registradores**
    - » **Valores temporários**
    - » **Valores de variáveis**

Suporte para operandos

- **Registradores x variáveis**
  - **SW: número ilimitado de variáveis**
    - » **Linguagens de programação**
  - **HW: número limitado de registradores**
- **Consequência:**
  - **Nem todas as variáveis são alocadas em registradores simultaneamente**
- **Conclusão:**
  - **Alocação de registradores**
    - » **Manualmente: programador em “assembly”**
    - » **Automaticamente: compilador!**

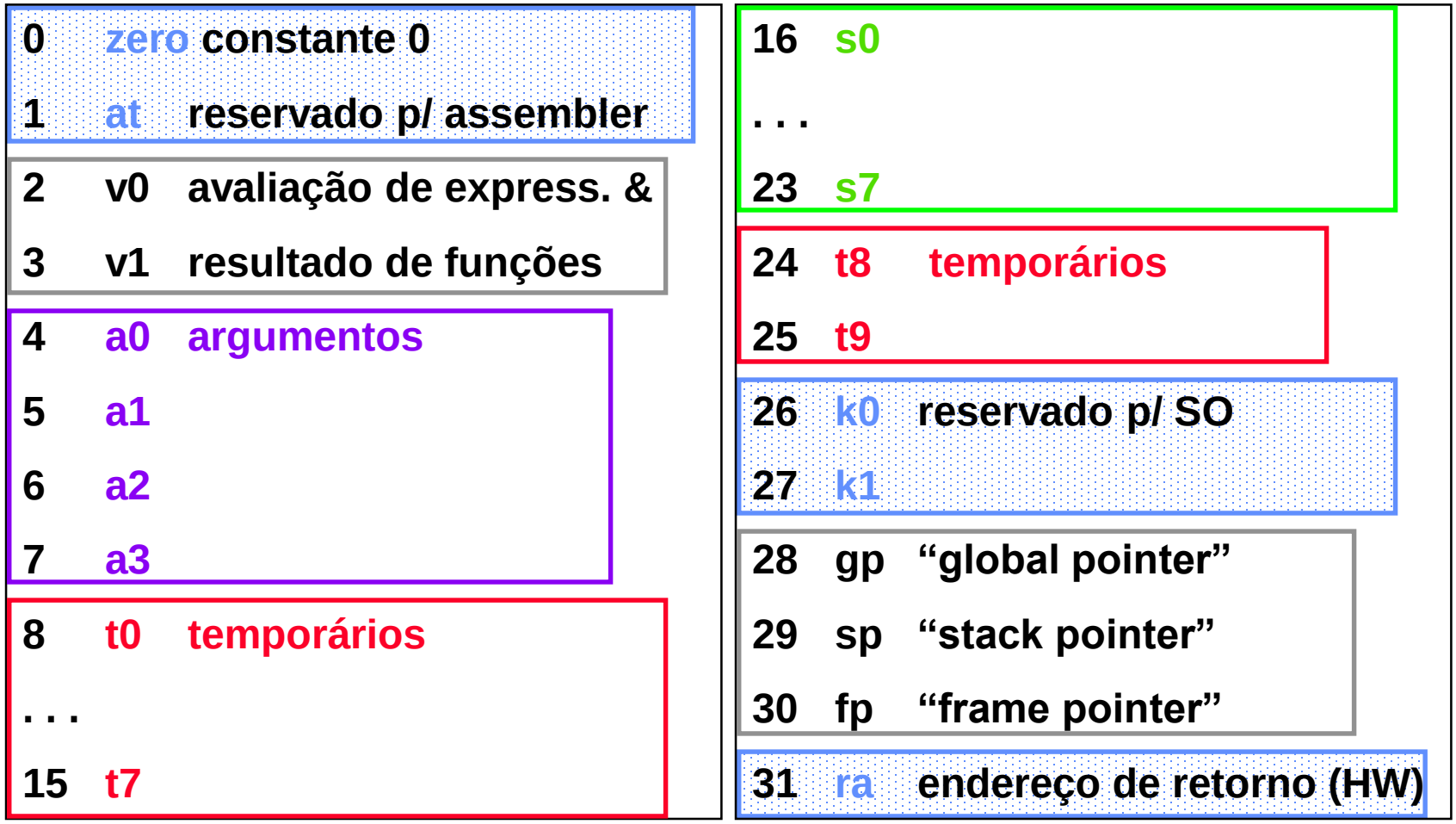
Suporte para operandos

- **Princípio de Projeto 2**
  - **Quanto menor, mais rápido.**
    - » **Menor a energia consumida (para mesma tecnologia)**
- **Registradores x memória principal**
  - **Memória: acesso mais lento (~50ns em DRAM)**
    - » **Milhares a milhões de elementos**
  - **Registradores: acesso mais rápido (~0.5ns em SRAM)**
    - » **Algumas dezenas de elementos**
- **Conclusão:**
  - **Alocação de registradores**
  - **Uma otimização para melhorar o desempenho**
    - » **Reduz também o consumo de energia**

Suporte para operandos

- **Exemplo: MIPS 32**
  - **32 registradores de uso geral**
    - » **Cada operando escolhido deve residir num deles**
  - **Sintaxe:**
    - » **$s0, $s1, ... (valores de variáveis)**
    - » **$t0, $t1, ... (valores temporários)**

MIPS: banco de registradores

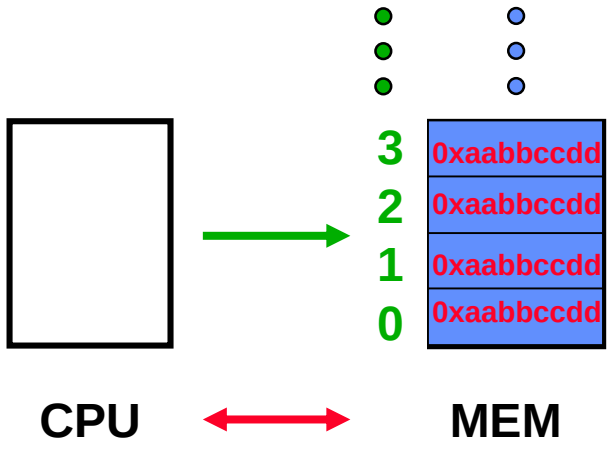


Suporte para operandos

- **Nem todos os operandos são escalares**
  - **Podem ser elementos de estruturas de dados**
    - » **Arranjos, records, listas, etc.**
- **Estruturas de dados**
  - **Não cabem no banco de registradores**
- **Solução**
  - **Estruturas de dados mantidas em memória**

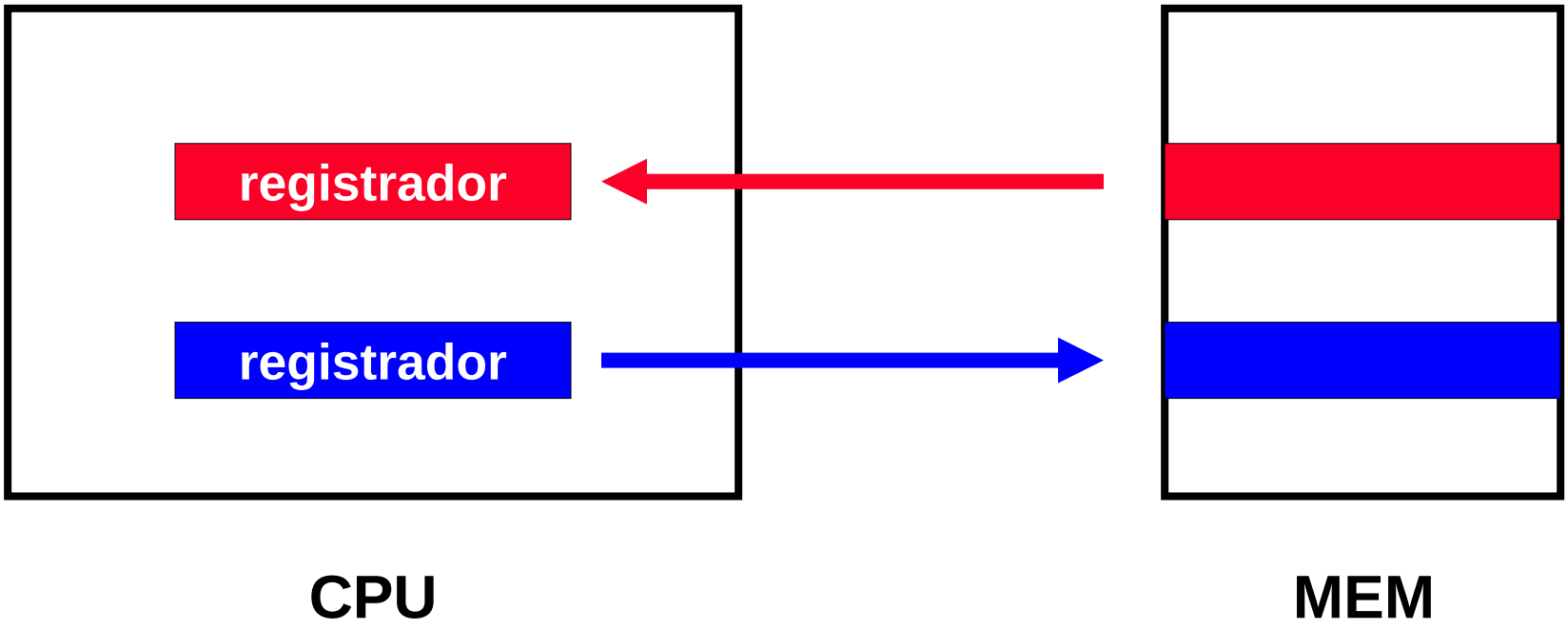
Suporte para operandos

- **Memória**
  - **"Arranjo unidimensional"**
    - » **Organizado em palavras**
    - » **Palavras armazenam os dados**
    - » **Cada palavra tem um endereço**



- **Operações de transferência de dados**
  - **Load (lw)**
  - **Store (sw)**

Load e Store



Máquina Load/Store

- **Exclusividade de acesso**
  - **Somente as instruções load e store podem acessar a memória.**
- **Corolário 1:**
  - **Outras instruções devem acessar seus operandos em registradores.**
- **Corolário 2:**
  - **Se uma instrução aritmética precisa manipular um elemento de estrutura de dados, este deve ser pré-carregado em registrador.**

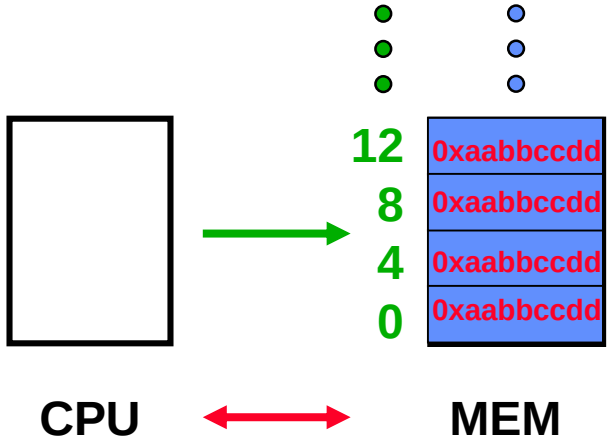
Suporte para operandos

- **Exemplo: g = h + A[8];**
  - **g → $s1, h → $s2**
  - **A[100], endereço-base → $s3**
  - **endereço = base + deslocamento**

```
lw $t0, 8($s3)        # temporário recebe A[8]
add $s1, $s2, $t0     # g = h + A[8]
```

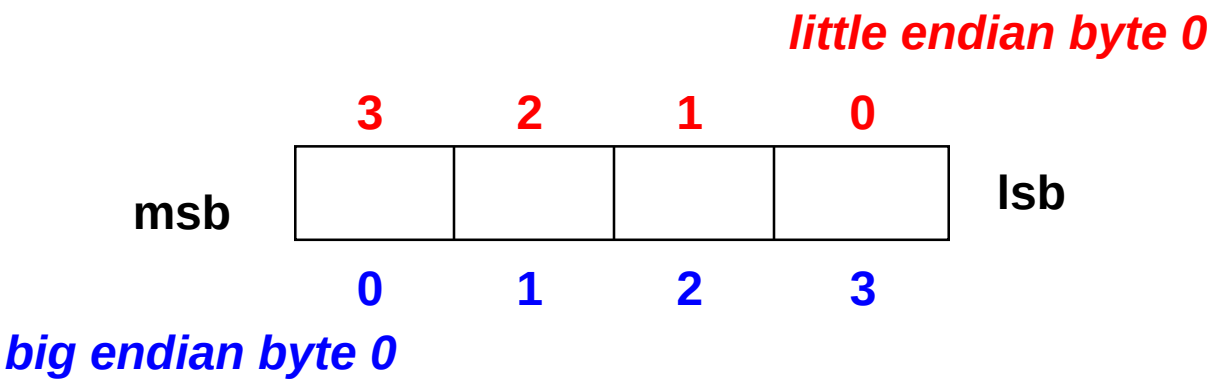
Suporte para operandos

- **Dados de diferentes tamanhos**
  - **Byte (char), half word (short), word (integer)**
- **Endereçamento de byte individual**
  - **1 word = 4 bytes**
    - » **Referenciada pelo endereço de um de seus bytes**
    - » **O endereço de menor valor!**
  - **Endereços de palavras sucessivas**
    - » **Incrementar de 4**

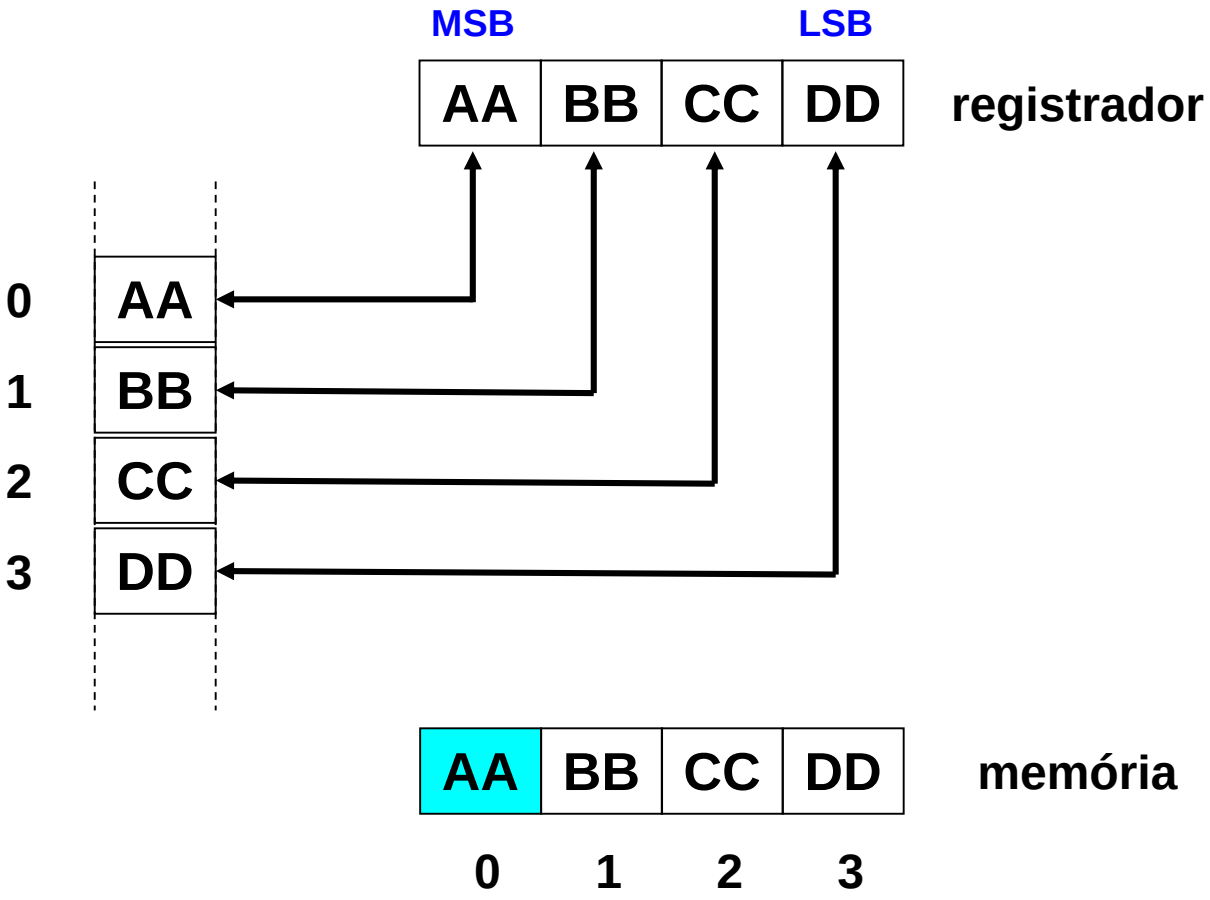


Endian = ordem dos bytes

- **Classificação pelo endereço que identifica palavra**
  - **Big Endian: endereço do byte mais significativo (msb)**
    - » **Exemplo: MIPS**
  - **Little Endian: endereço do byte menos significativo (lsb)**
    - » **Exemplo: Intel x86**

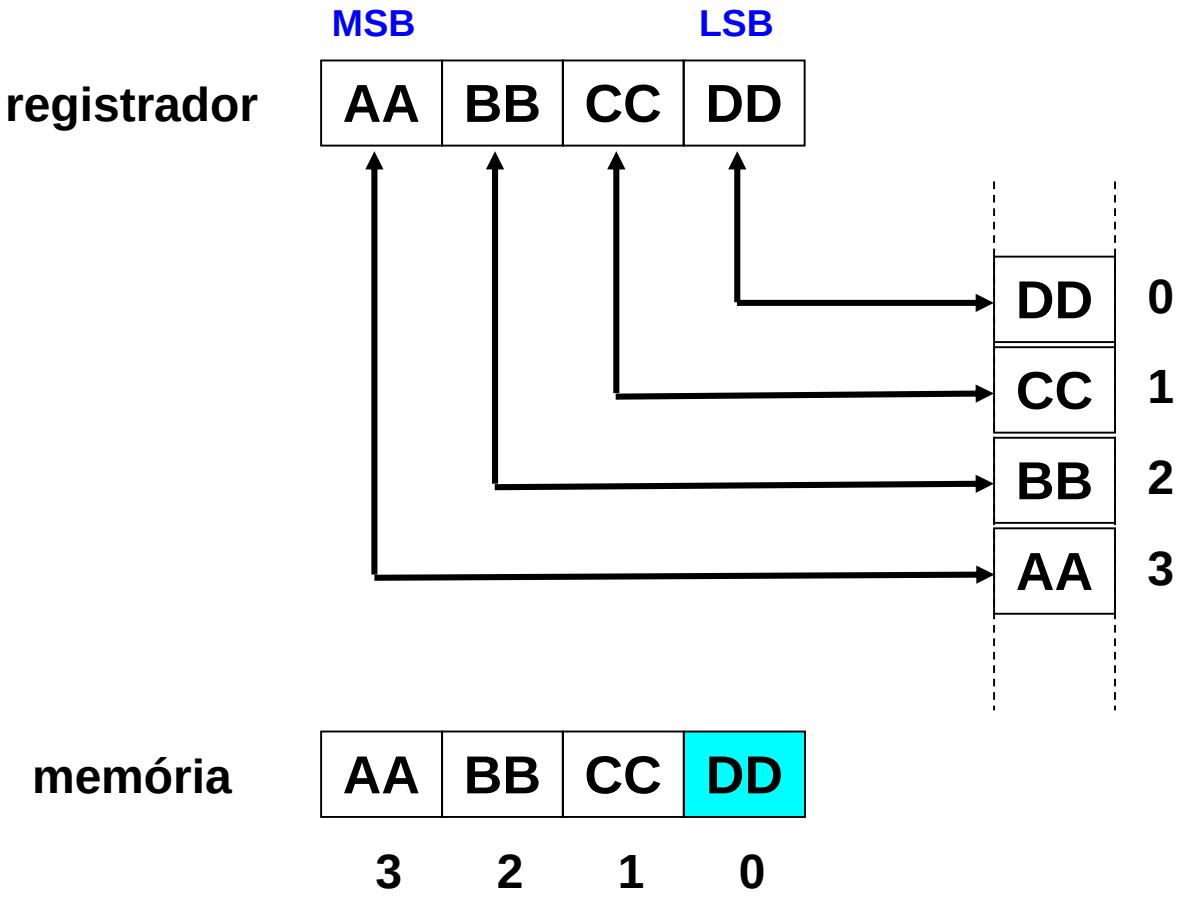


Big endian



Byte mais significativo (MSB) armazenado no menor endereço

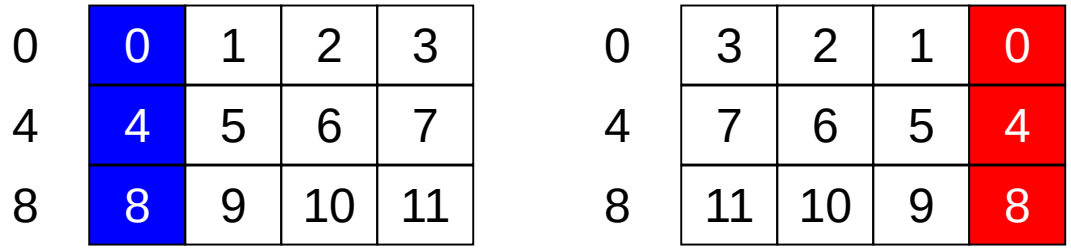
Little endian



Byte menos significativo (LSB) armazenado no menor endereço

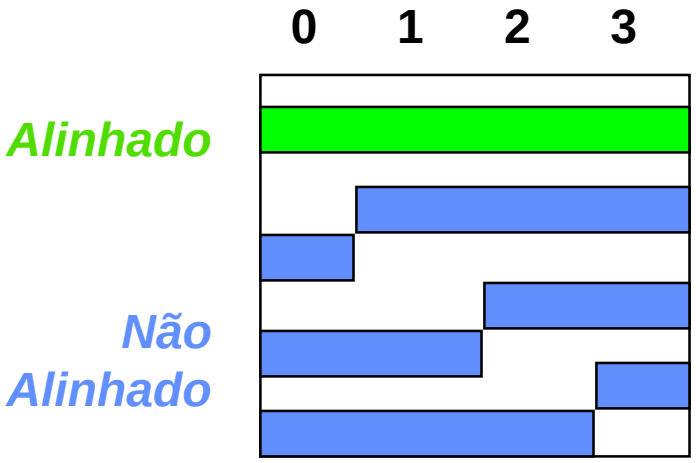
Endian = ordem dos bytes

- **Classificação pelo endereço que identifica palavra**
  - **Big Endian: endereço do byte mais significativo (msb)**
    - » **Exemplo: MIPS**
  - **Little Endian: endereço do byte menos significativo (lsb)**
    - » **Exemplo: Intel x86**



Alinhamento de objetos

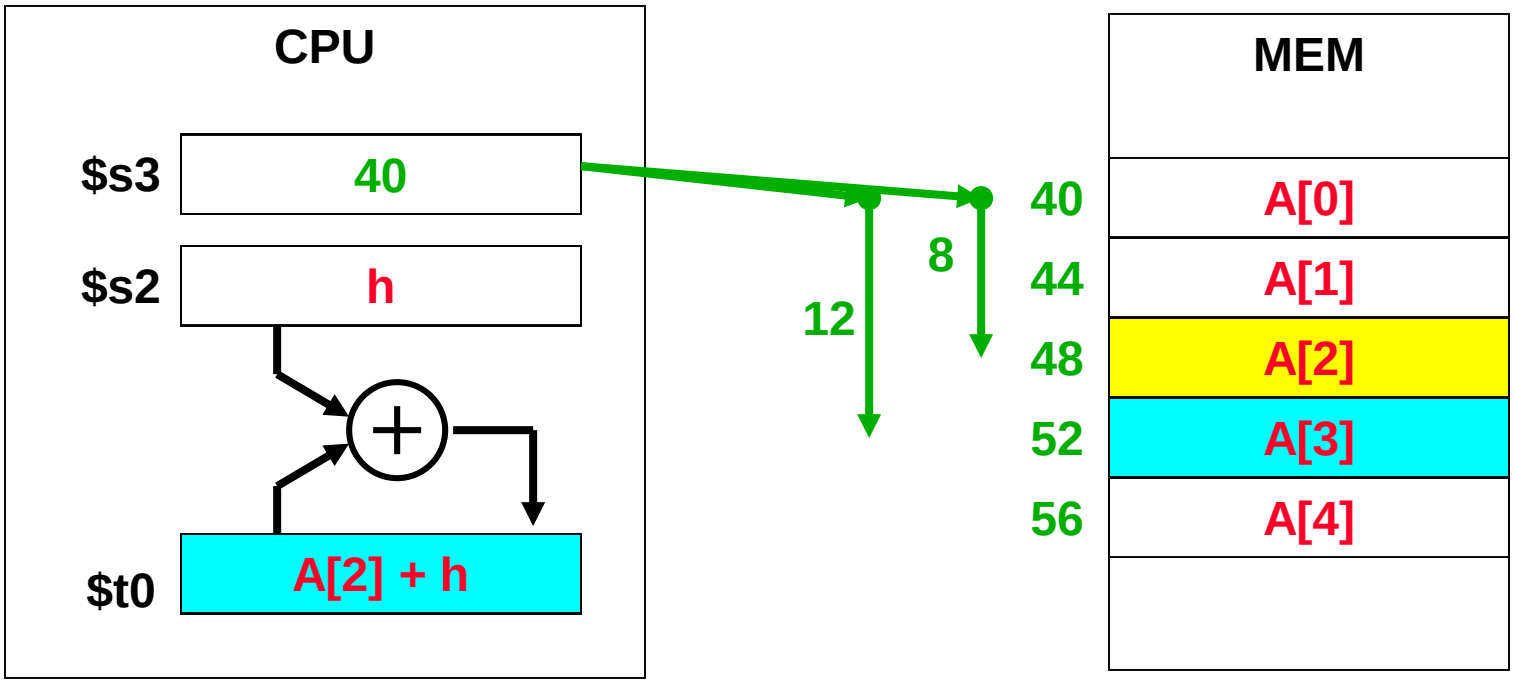
- **Objetos de diferentes tamanhos**
  - **Byte, half word, word**
- **Dado (instrução) está alinhado (a) se:**
  - **Endereço for múltiplo de seu tamanho**



Exemplo

A[3] = h + A[2]; **base de A → s3; h → $s2**

```
lw      $t0, 8($s3)      # temp. recebe A[2]
add     $t0, $s2, $t0    # temp. recebe h + A[2]
sw      $t0, 12($s3)     # armazena h+ A[2] em A[3]
```



Operandos constantes

- **Ocorrência de constantes é frequente**
  - **50% das instruções aritméticas**
- **Solução 1: constante mantida em memória**

  lw $t0, AddrConstant4 ($s1)

  add $s3, $s3, $t0
- **Solução 2: constante embutida na instrução**

  addi $s3, $s3, 4
- **Princípio de projeto 3**
  - **Acelere o caso frequente**
    - » **Instruções imediatas**

Interface HW/SW

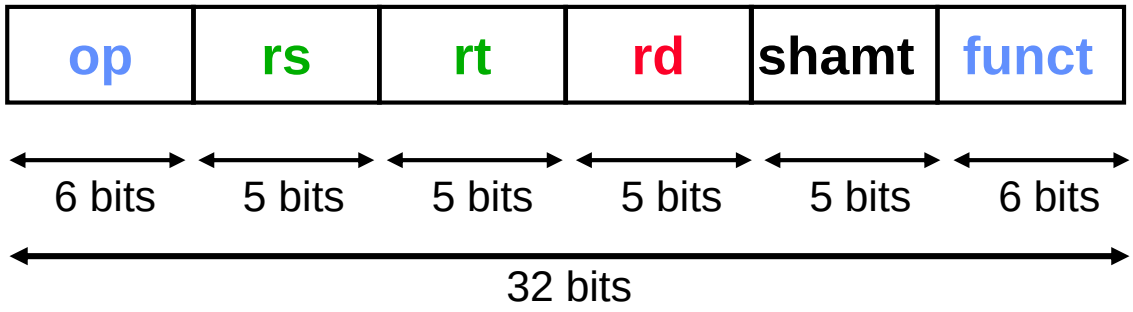
- **Registradores são mais eficientes**
  - **Menor tempo de acesso**
  - **Maior vazão de dados ("throughput")**
    - » **Até 3 operandos acessados por instrução (via registradores)**
    - » **Um único operando acessado em memória por instrução**
  - **Menor consumo de energia**
- **Como o compilador mantém as variáveis ?**
  - **As de uso mais frequente mantidas em registradores**
    - » ***Register allocation* = mapeamento: variáveis → registradores**
  - **As de uso menos frequente em memória**
    - » ***Register spilling* = mapeamento: registradores → memória**
  - **Load e store usadas para movê-las**

Componentes de uma instrução

- **Codificados como números**
  - **Operação**
    - » **Código operacional (“opcode”)**
  - **Endereço de operando**
    - » **Número de registrador ou endereço de memória**
  - **Valor de operando constante**
    - » **Imediato**
- **Representação de uma instrução**
  - **Formato de instrução**
    - » **Organização de seus componentes em campos**

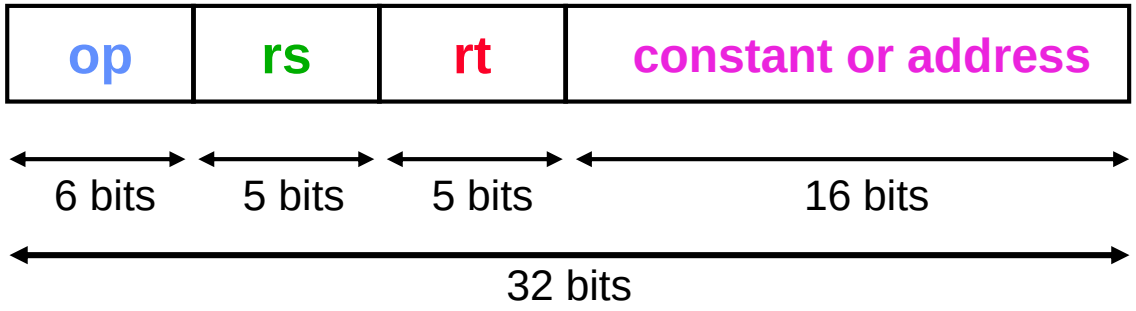
MIPS: formatos de instrução

- **Formato tipo R**



add $s1,$s2,$s3

- **Formato tipo I**



lw $t0,32($s3)

addi $s3, $s3, 4

Geração de código

- **Linguagem de alto nível**
  - **A[300] = h + A[300];**
    - » **Base de A → $t1; h →$s2**

- **Linguagem de montagem**

```
lw   $t0,1200($t1)
add  $t0,$s2,$t0
sw   $t0,1200($t1)
```

Geração de código

- **Linguagem de montagem**

```
lw      $t0,1200($t1)
add     $t0,$s2,$t0
sw      $t0,1200($t1)
```

- **Linguagem de máquina**

100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

Conceito de programa armazenado

- **Princípios-chave em computadores**
  - **Instruções são representadas como números**
  - **Programas são armazenados na memória**
    - » **Para serem lidos ou escritos como números**
  - **Processador pode executar vários algoritmos**
- **Computador = sistema digital programável**
  - **Contra-exemplo: ASIC**
    - » **“Application-specific integrated circuit”**
    - » **Sistema digital que executa um único algoritmo**

Conceito de programa armazenado