Estructura de Computadores

Objetivos

Objetivos

- Conocer la jerarquía de niveles de un computador

Objetivos

- Conocer la jerarquía de niveles de un computador
- Conocer la arquitectura (**ISA**) de un procesador RISC

Objetivos

- Conocer la jerarquía de niveles de un computador
- Conocer la arquitectura (**ISA**) de un procesador RISC
- Saber representar y operar con números reales y enteros

Objetivos

- Conocer la jerarquía de niveles de un computador
- Conocer la arquitectura (**ISA**) de un procesador RISC
- Saber representar y operar con números reales y enteros
- Saber cómo se almacenan y acceden datos estructurados

Objetivos

- Conocer la jerarquía de niveles de un computador
- Conocer la arquitectura (**ISA**) de un procesador RISC
- Saber representar y operar con números reales y enteros
- Saber cómo se almacenan y acceden datos estructurados
- Saber traducir programas de alto nivel a **lenguaje ensamblador**

Objetivos

- Conocer la jerarquía de niveles de un computador
- Conocer la arquitectura (**ISA**) de un procesador RISC
- Saber representar y operar con números reales y enteros
- Saber cómo se almacenan y acceden datos estructurados
- Saber traducir programas de alto nivel a **lenguaje ensamblador**
- Conocer la estructura y el funcionamiento de la **memoria cache**

Objetivos

- Conocer la jerarquía de niveles de un computador
- Conocer la arquitectura (**ISA**) de un procesador RISC
- Saber representar y operar con números reales y enteros
- Saber cómo se almacenan y acceden datos estructurados
- Saber traducir programas de alto nivel a **lenguaje ensamblador**
- Conocer la estructura y el funcionamiento de la **memoria cache**
- Entender el funcionamiento básico de la **memoria virtual**

Objetivos

- Conocer la jerarquía de niveles de un computador
- Conocer la arquitectura (**ISA**) de un procesador RISC
- Saber representar y operar con números reales y enteros
- Saber cómo se almacenan y acceden datos estructurados
- Saber traducir programas de alto nivel a **lenguaje ensamblador**
- Conocer la estructura y el funcionamiento de la **memoria cache**
- Entender el funcionamiento básico de la **memoria virtual**
- Conocer los conceptos de **excepción** e **interrupción**

Objetivos

- Conocer la jerarquía de niveles de un computador
- Conocer la arquitectura (**ISA**) de un procesador RISC
- Saber representar y operar con números reales y enteros
- Saber cómo se almacenan y acceden datos estructurados
- Saber traducir programas de alto nivel a **lenguaje ensamblador**
- Conocer la estructura y el funcionamiento de la **memoria cache**
- Entender el funcionamiento básico de la **memoria virtual**
- Conocer los conceptos de **excepción** e **interrupción**
- Adquirir competencias transversales (sostenibilidad)

Temario

1. Introducción
2. Ensamblador MIPS y tipos de datos básicos
3. Traducción de programas
4. Matrices
5. Aritmética de enteros y coma flotante
6. Memoria cache
7. Memoria virtual
8. Excepciones e interrupciones

Recursos

- Bibliografía
  - D. Patterson and J. L. Hennessy. “Estructura y Diseño de Computadores: La Interfaz Hardware/Software”, 2011.
- Recursos online:
  - http://docencia.ac.upc.edu/FIB/grau/EC/
  - https://raco.fib.upc.edu/
  - Problemas, apuntes, prácticas, exámenes de años anteriores...
  - Transparencias: http://jarnau.site.ac.upc.edu/EC/
- Consultas
  - jarnau@ac.upc.edu
  - Despacho C6-115

Teoría, Laboratorio y Exámenes

- 28 sesiones de teoría/problemas
  - Lunes y miércoles de 18:00 a 20:00 (A5102)
- 6 sesiones de laboratorio
  - Sesión 0: Introducción
  - Sesión 1: Ensamblador MIPS y tipos de datos básicos
  - Sesión 2: Traducción de programas
  - Sesión 3: Tipos de datos estructurados
  - Sesión 4: Codificación en coma flotante
  - Sesión 5: Memoria cache
- Exámenes:
  - Examen parcial: 07/11/2019
  - Examen de laboratorio: 19/12/2019
  - Examen final: 09/01/2020

Evaluación

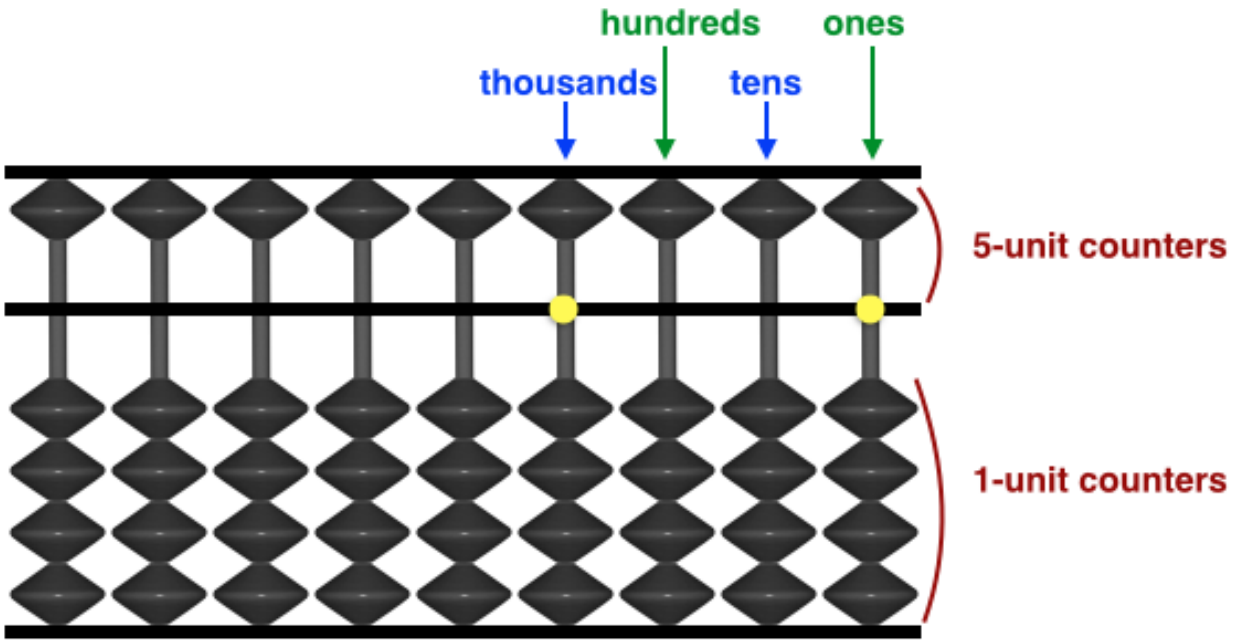
- 0.2*max(EP,EF) + 0.6*EF + 0.2*(EL*0.85 + EC*0.15)
  - EP = Examen Parcial (8/11/18)
  - EF = Examen Final (10/1/19)
  - EL = Examen de Laboratorio (20/12/18)
  - EC = Evaluación Continua de Laboratorio

- Evaluación continua en cada sesión de laboratorio
  - Estudio previo e individual
  - Actividades por parejas durante la sesión presencial

Tema 1. Introducción

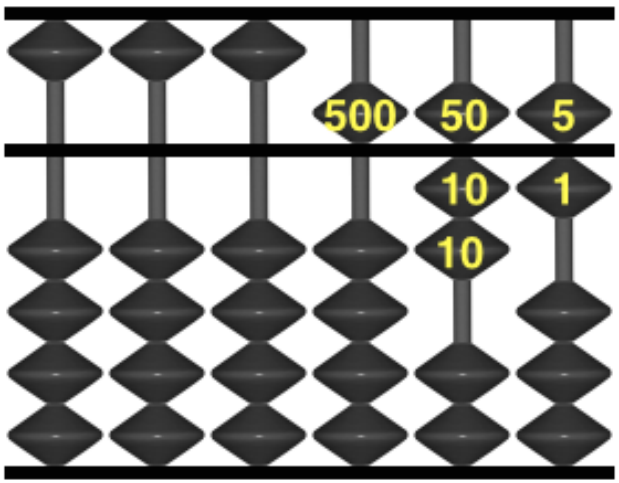
¿Cuál fue el primer dispositivo de cómputo?

¿Cuál fue el primer dispositivo de cómputo?



¿Cómo se realizan los cálculos?

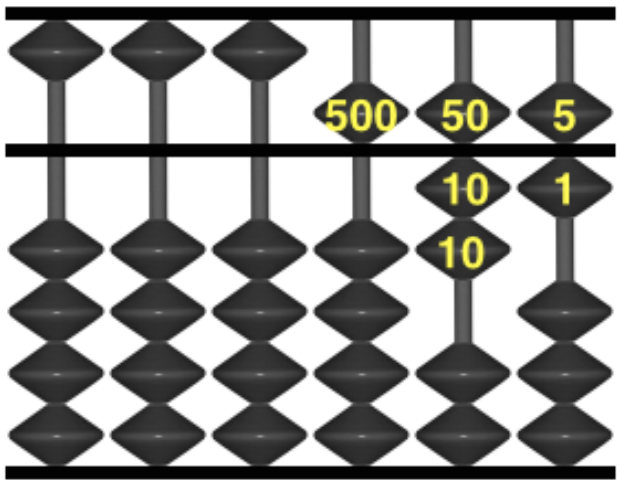
Ábaco



Movimiento de fichas

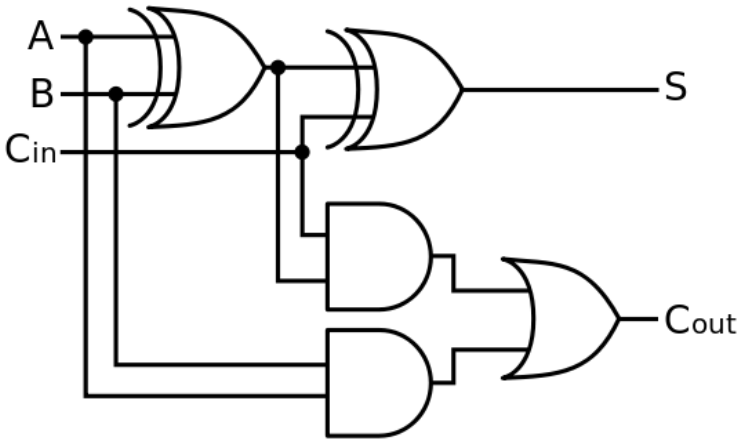
¿Cómo se realizan los cálculos?

Ábaco



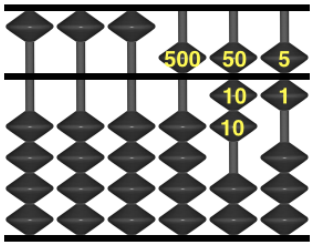
Movimiento de fichas

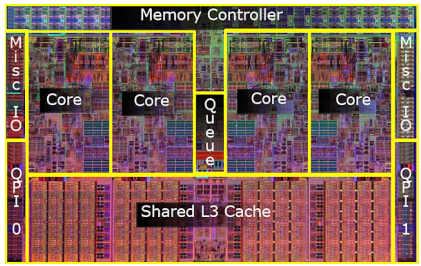
Computador Digital Moderno



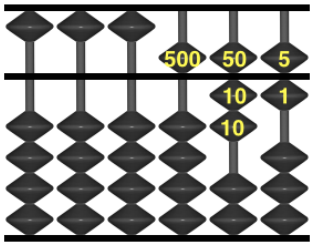
Movimiento de electrones

Ábaco vs Computador Digital

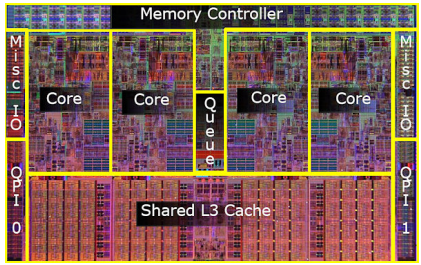




Ábaco vs Computador Digital

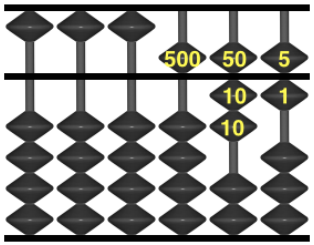


Mecánico

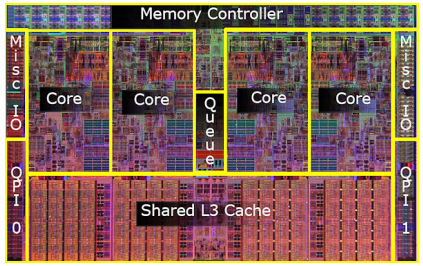


Electrónico

Ábaco vs Computador Digital

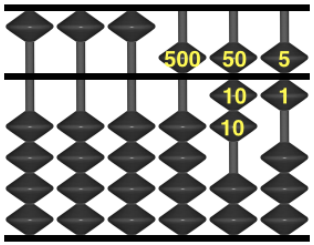


Mecánico
Manual

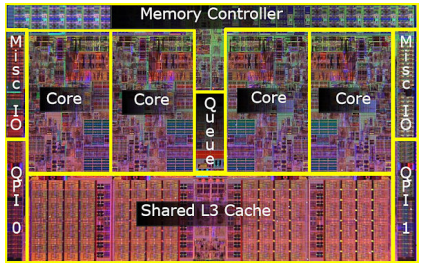


Electrónico
Automático/Programable

Ábaco vs Computador Digital

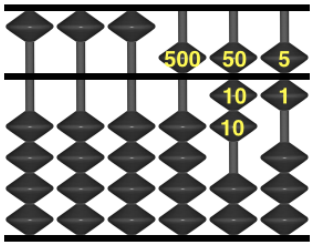


Mecánico
Manual
Escaso almacenamiento

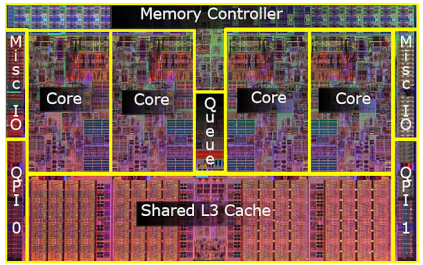


Electrónico
Automático/Programable
Gran almacenamiento (TBytes)

Ábaco vs Computador Digital

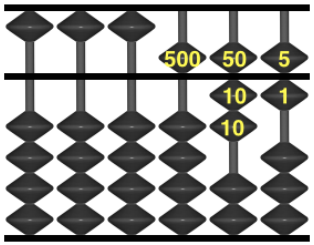


Mecánico
Manual
Escaso almacenamiento
Lento

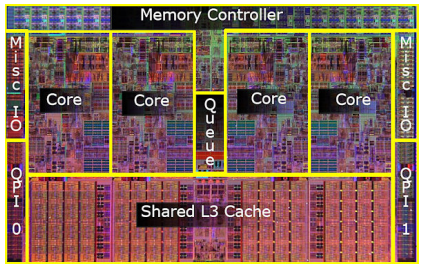


Electrónico
Automático/Programable
Gran almacenamiento (TBytes)
Muy rápido (TFLOPS)

Ábaco vs Computador Digital



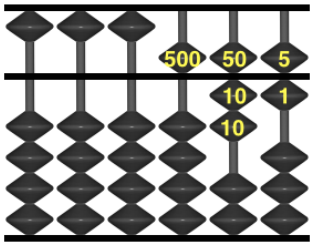
Mecánico
Manual
Escaso almacenamiento
Lento



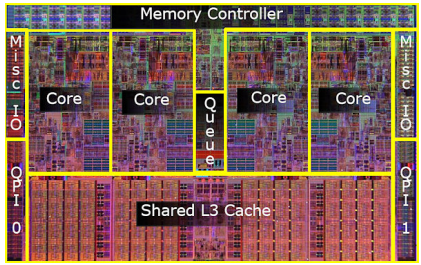
Electrónico
Automático/Programable
Gran almacenamiento (TBytes)
Muy rápido (TFLOPS)

- Computador digital
  - Dispositivo electrónico capaz de almacenar y procesar información de forma automática

Ábaco vs Computador Digital



Mecánico
Manual
Escaso almacenamiento
Lento



Electrónico
Automático/Programable
Gran almacenamiento (TBytes)
Muy rápido (TFLOPS)

- Computador digital
  - Dispositivo electrónico capaz de almacenar y procesar información de forma automática
  - Pero el usuario no puede interaccionar con los electrones de forma directa...

Niveles de abstracción

Interfaz de usuario

Menus, iconos, botones, copia/pega...

Lenguaje de alto nivel

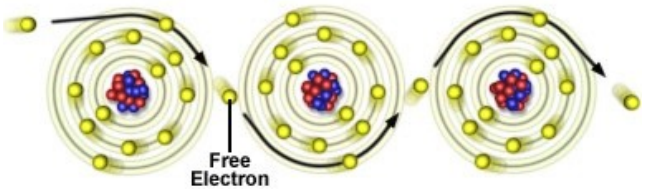
C/C++, Java, Fortran, Python... (Pro2)

Lenguaje máquina

MIPS, x86, ARM, RISC-V... (EC)

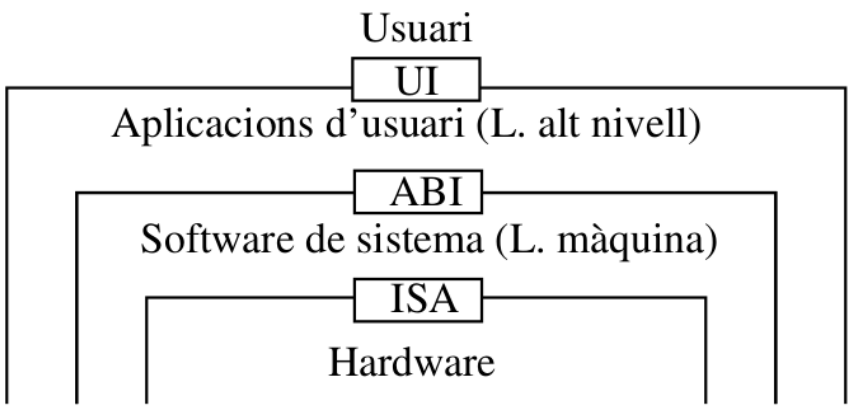
Hardware

Puertas lógicas, multiplexores, biestables... (IC)



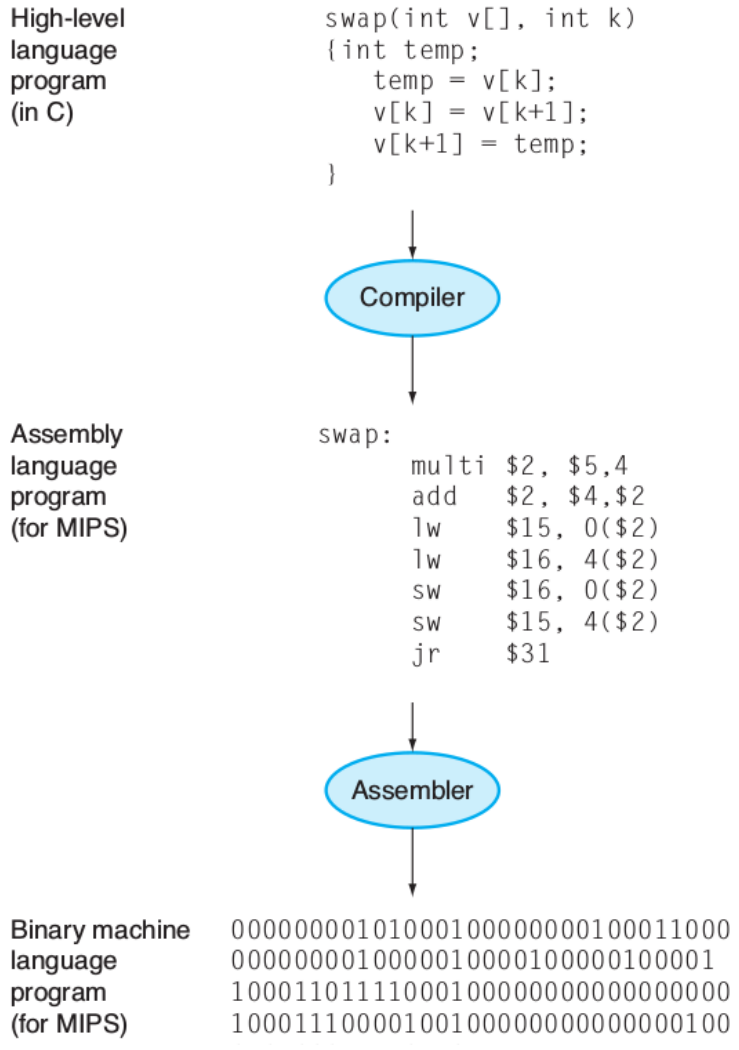
Interfaces entre niveles

- Application Binary Interface (ABI)
  - Funciones del sistema operativo (llamadas al sistema)
- Instruction Set Architecture (ISA)
  - Interfaz entre el hardware y el software
  - Describe los aspectos del procesador visibles al programador en lenguaje máquina:
    - Juego de instrucciones, modos de direccionamiento, excepciones, modelo de memoria...
  - Ejemplos: MIPS, x86, ARM, RISC-V



Traducción entre niveles

- Lenguaje de alto nivel
  - Abstracto
  - Portable
  - Rápido de escribir
- Lenguaje ensamblador
  - Representación textural de instrucciones
- Lenguaje máquina
  - Representación hardware
  - Dígitos binarios (bits)
  - Datos e instrucciones



Compilación vs Interpretación

- Compilación
  - Traducción del programa entero una sola vez (estático)
  - Programa generado es muy rápido
  - Recompilación para cada ISA y/o ABI (no portable)
  - Ejemplos: C/C++, Fortran, Pascal...
- Interpretación
  - Traducción dinámica en tiempo de ejecución
  - La ejecución es más lenta
  - Portabilidad
  - Ejemplos: Java, Python...
- Los lenguajes interpretados son mucho más productivos, portables y seguros... pero son demasiado lentos para aplicaciones donde el rendimiento es crítico

Tema 2. Ensamblador MIPS y tipos de datos básicos

CISC vs RISC

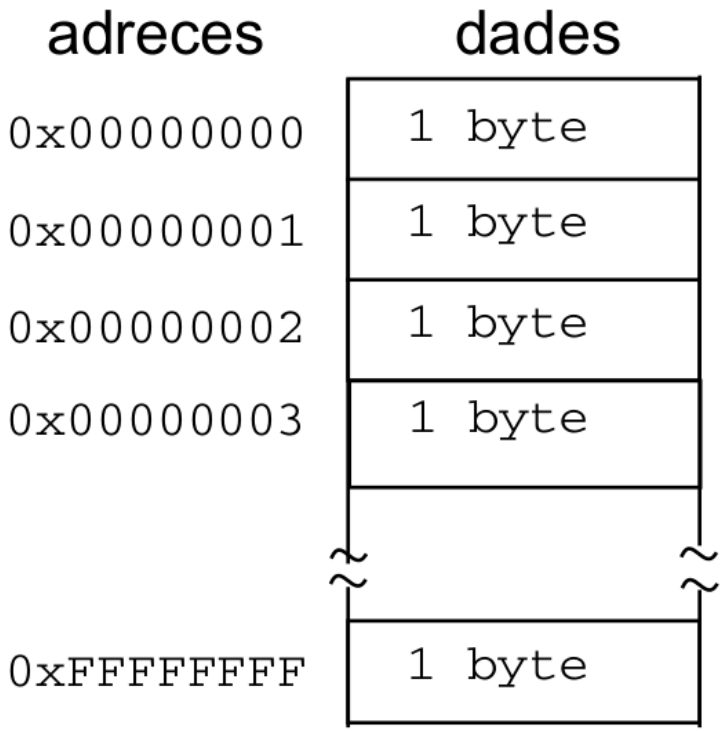
- Complex Instruction Set Computer (CISC)
  - Juego de instrucciones grande y complejo
  - Instrucciones de longitud variable
  - Cada instrucción se decodifica en múltiple microoperaciones
  - Ejemplos: x86
- Reduced Instruction Set Computer (RISC)
  - Juego de instrucciones pequeño y sencillo
  - Instrucciones de longitud fija
  - Formatos de instrucción y modos de direccionamiento sencillos
  - Ejecutadas directamente por hardware
  - Ejemplos: MIPS, ARM, RISC-V

MIPS

- Microprocessor without Interlocked Pipeline Stages
  - Diseñado en 1981-1985 por Henessy y Patterson
  - Juego de instrucciones sencillo (RISC)
  - Distintas implementaciones comerciales
    - Routers de Cisco y Linksys
    - Módems ADSL
    - Controladoras de impresora láser
    - Playstation (PSX, PS2 y PSP)
    - Nintendo 64
- Ampliamente utilizado para la docencia
  - Muchos de los conceptos son muy similares en otros ISAs RISC
  - MIPS32: ISA utilizado en la asignatura (teoría y laboratorio)

La Memoria

- Vector de bytes
  - Cada byte se identifica por una dirección

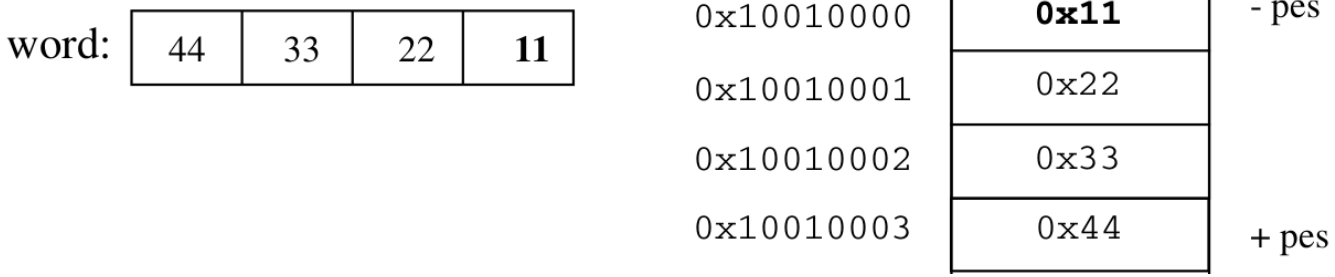


Palabra (Word)

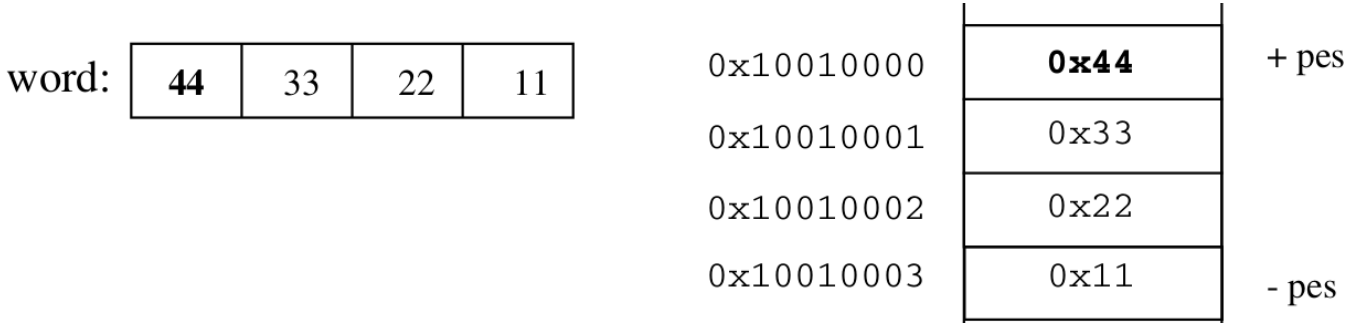
- Dato que tiene el tamaño nativo de la arquitectura
- Normalmente, tamaño de palabra = tamaño de registro
- MIPS32: 32 bits (4 bytes)
- Ejemplos: 0xAABBCCDD, 0x44332211
- ¿Cómo se almacenan los bytes de una palabra en memoria?

Endianness

- Little-endian: byte de **menor** peso en la dirección más baja



- Big-endian: byte de **mayor** peso en la dirección más baja



Declaración de variables

- Variables **globales**
  - Pueden ser accedidas desde cualquier función
  - Se mantienen en memoria durante toda la ejecución del programa
  - Se almacenan en una dirección de memoria fija
- Variables **locales**
  - Solo se pueden acceder dentro del bloque donde se declaran
  - Se crean al inicio de la ejecución del bloque y dejan de existir cuando finaliza
  - Se reserva espacio de almacenamiento de forma dinámica (memoria o registro)

Declaración de variables

- Lenguaje C

```
int a = 0x44332211;

int main(void) {
    int i = 7;
}
```

- Ensamblador MIPS

```
.data
a: .word 0x44332211

.text
main:
    li $t0, 7
```

Tamaño de las variables

Tamaño	Lenguaje C	Ensamblador MIPS
1 byte	char / unsigned char	.byte
2 bytes	short / unsigned short	.half
4 bytes	int / unsigned int	.word
8 bytes	long long / unsigned long long	.dword

Alineación en memoria

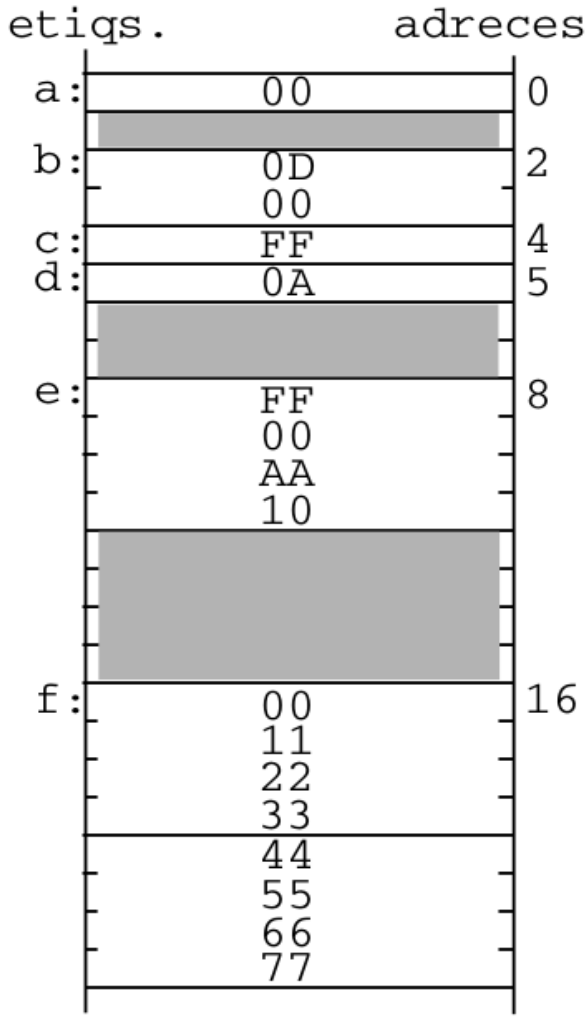
```
unsigned char a;
short b = 13;
char c = -1, d = 10;
int e = 0x10AA00FF;
long long f = 0x7766554433221100;
```

```
    .data
a:  .byte 0
b:  .half 13
c:  .byte -1
d:  .byte 10
e:  .word 0x10AA00FF
f:  .dword 0x7766554433221100
```

Alineación en memoria

```
unsigned char a;
short b = 13;
char c = -1, d = 10;
int e = 0x10AA00FF;
long long f = 0x7766554433221100;
```

```
    .data
a:  .byte 0
b:  .half 13
c:  .byte -1
d:  .byte 10
e:  .word 0x10AA00FF
f:  .dword 0x7766554433221100
```



Ejercicio

- Traduce a MIPS la siguiente declaración de variables en C e indica el contenido de la memoria.

```
char a = 0xFF;
char b = 0xEE;
char c = 0xDD;
unsigned long long d = 0x7766554433221100;
short e = 0xABCD;
unsigned int f = 0x40302010;
```

Declaración de vector

- Declaración con inicialización

```
short vec[5] = {2, -1, 3, 5, 0};
```

```
        .data
vec:    .half 2, -1, 3, 5, 0
```

Declaración de vector

- Declaración con inicialización

```
short vec[5] = {2, -1, 3, 5, 0};
```

```
      .data
vec:  .half 2, -1, 3, 5, 0
```

- Declaración sin inicialización (alineación explícita)

```
char a;
int v[100];
```

```
    .data
a:  .byte 0
    .align 2     # Alinear a multiplo de 4
v:  .space 400   # Vector de 100 enteros
```