UNIVERSIDAD POLITECNICA DE VALENCIA

etsinf

DISCA

Fundamentos de computadores

Tema 7. LENGUAJE ENSAMBLADOR

Objetivos

FCO

- Estudiar los conceptos fundamentales para la programación en lenguaje ensamblador del procesador MIPS R2000.
- Conocer la estructura básica de un procesador, que permita la ejecución de las diferentes instrucciones, a partir de circuitos digitales ya conocidos.
- Conocer los elementos de un programa (datos e instrucciones) en ensamblador y comprender su ubicación en la memoria.
- Conocer una muestra significativa del repertorio de instrucciones de un procesador actual.
- Entender la codificación los distintos tipos de instrucciones en lenguaje máquina del MIPS R2000.

DISCA

Índice

Bibliografía

FCO

- MIPS Assembly Language Programming. Robert Briton
- See MIPS Run. Dominic Sweetman
- Introducción a los computadores. Julio Sahuquillo

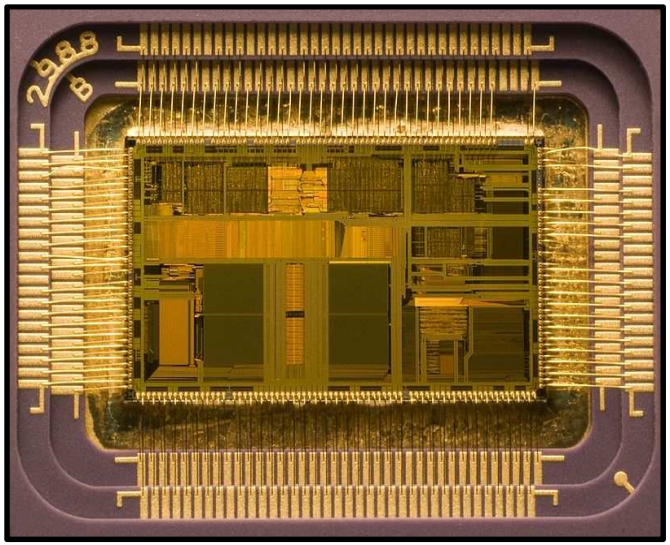
DISCA

Unidades Funcionales del Computador FCO

- Unidad funcional
  - Conjunto de circuitos que realizan una función específica diferenciada de otras funciones
- Unidades funcionales básicas del computador
  - Unidad Central de Proceso (UCP, CPU)
  - Unidad de Memoria
  - Unidad de Entrada/Salida (E/S, I/O)

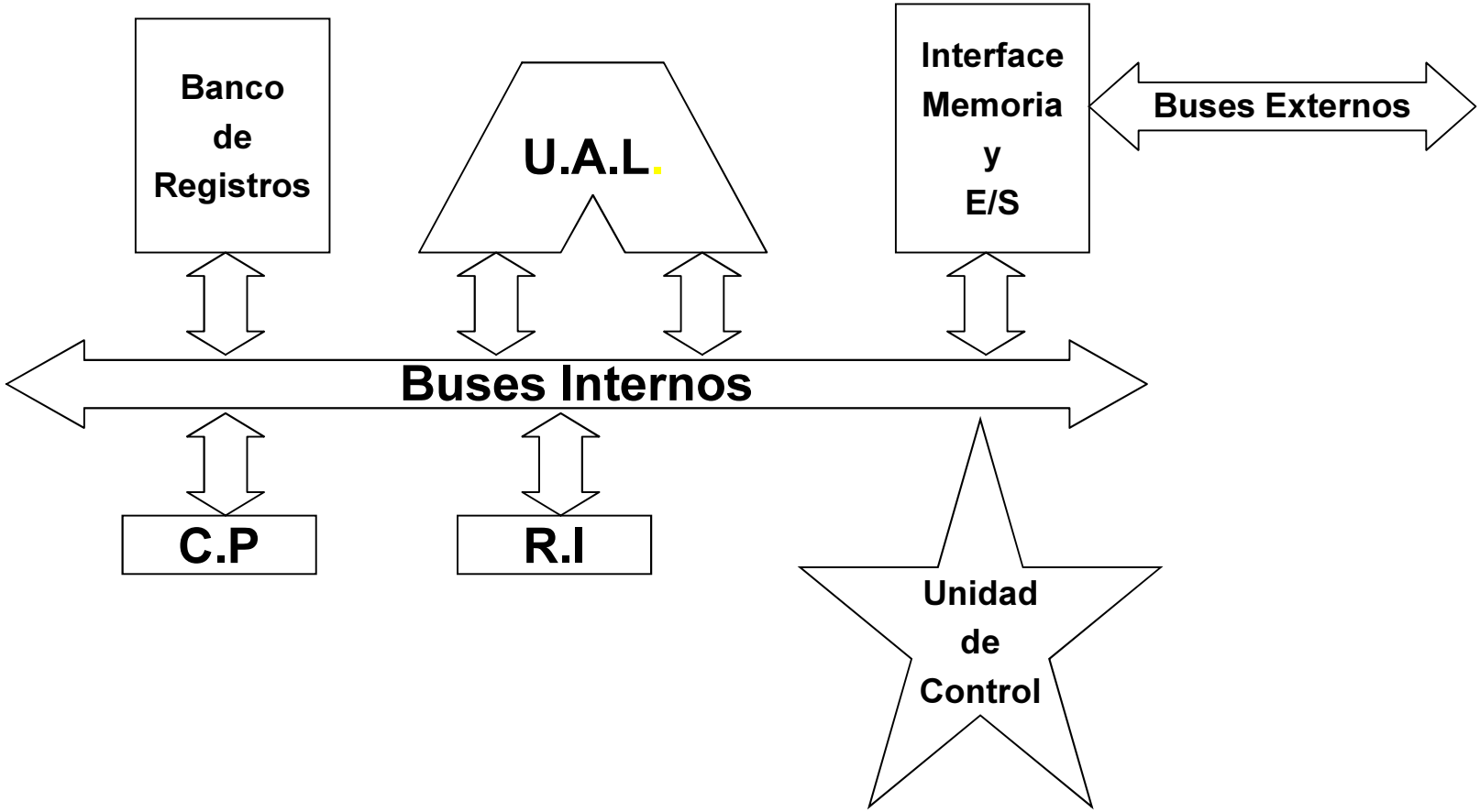
Unidades Funcionales del Computador FCO

- Unidad Central de Procesamiento o CPU
  - Es el componente en un ordenador, que interpreta las instrucciones y procesa los datos contenidos en los programas de la computadora.
  - Es decir, ejecuta instrucciones



Unidades Funcionales del Computador FCO

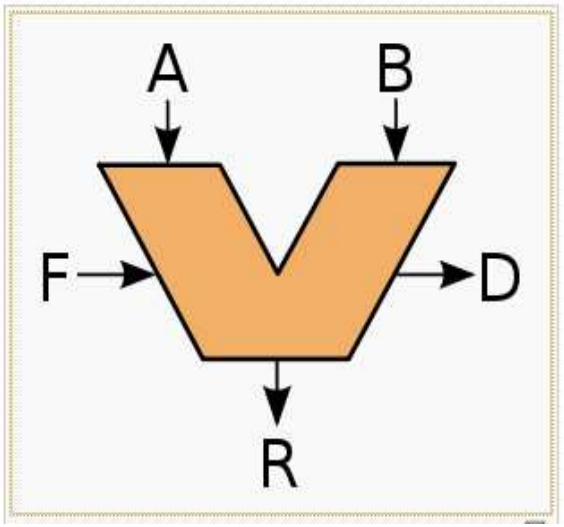
- Unidad Central de Proceso (UCP)



Unidades Funcionales del Computador FCO

- La Unidad Aritmético-Lógica (UAL), o Arithmetic Logic Unit (ALU), realiza operaciones aritméticas (como suma, resta, multiplicación, etc.) y operaciones lógicas (como igual a, menor que, mayor que, etc.), entre dos números.

  - A y B → Operandos
  - F → Operación a realizar
  - R → Resultado de la operación
  - D → Indicadores de resultado

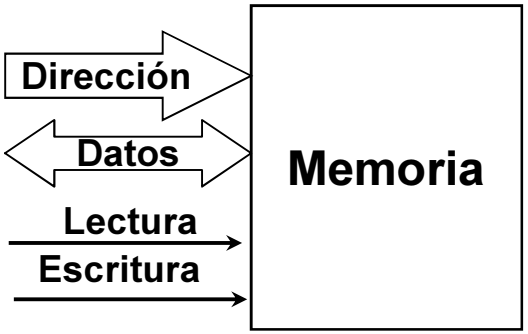


Unidades Funcionales del Computador FCO

- Banco de registros
  - Se usan para almacenar datos temporalmente
  - Memoria con un tiempo de acceso muy pequeño
  - Siempre se encuentra dentro del procesador
  - Numero de registros entre 8 y 512, depende del procesador
  - MIPS R2000 32 registros de 32 bits ($0 .. $31)
  - 1 puerto de escritura y 1 o 2 puertos de lectura
- Registros especiales
  - Contador de programa (C.P.) Contiene la dirección de memoria de la siguiente instrucción que se debe ejecutar.
  - Registro de instrucción (R.I.) Contiene el código de la instrucción que se está ejecutando.

Unidades Funcionales del Computador FCO

- Memoria: Dispositivo de almacenamiento (datos + instrucciones)

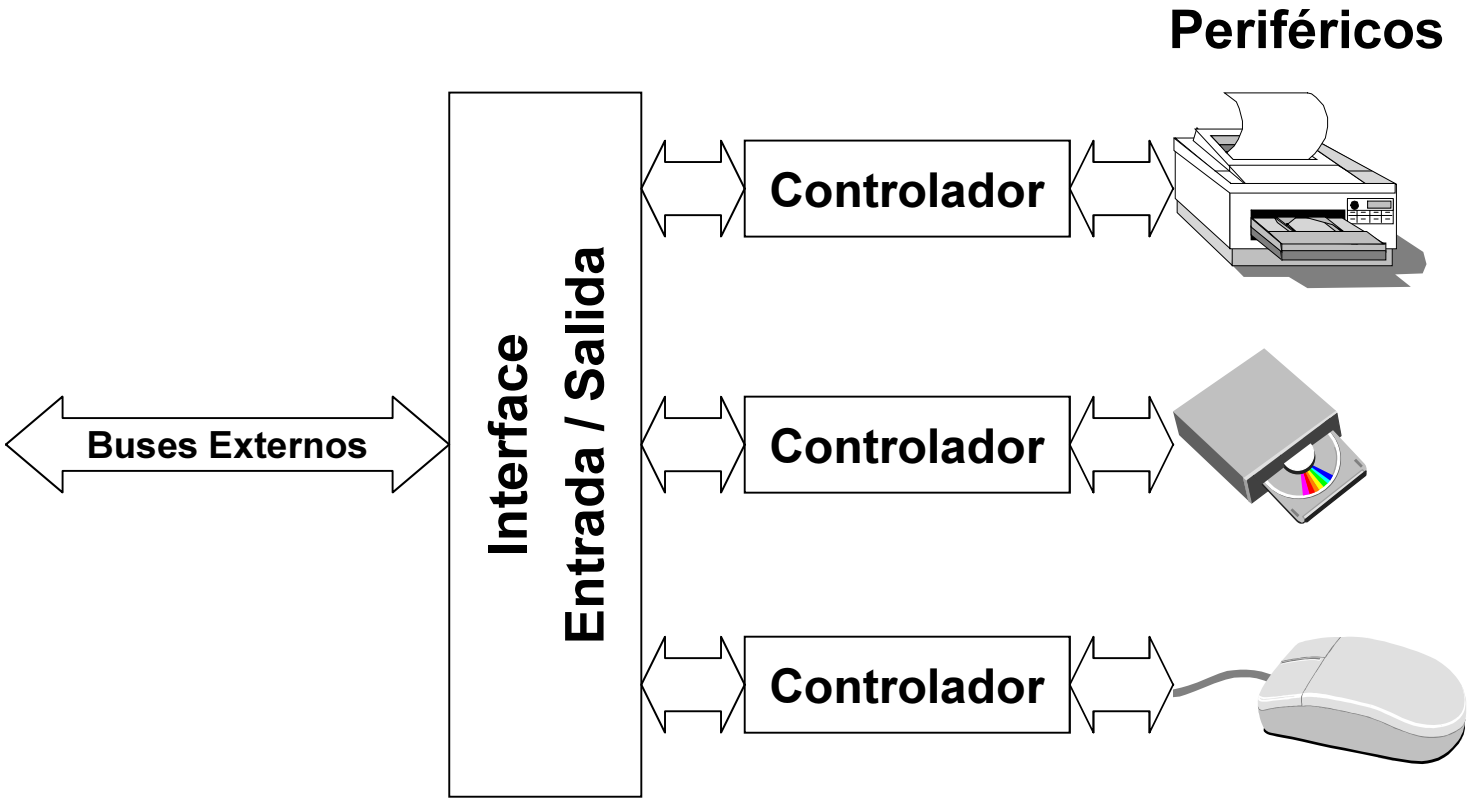


Unidades de capacidad

- **1K (kilo) = 2^{10} = 1024**
- **1M (mega) = 2^{10}K = 2^{20}**
- **1G (giga) = 2^{10}M = 2^{20}K = 2^{30}**
- **1T (tera) = 2^{10}G = 2^{20}M = 2^{30}K = 2^{40}**
- **1P (peta) = 2^{10}T = 2^{20}G = 2^{30}M = 2^{40}K =2^{50}**
- **1E (exa) = 2^{10}P = 2^{20}T = 2^{30}G = 2^{40}M = 2^{50}K =2^{60}**

Unidades Funcionales del Computador FCO

- Sistema de Entrada/Salida
  - Permiten la comunicación del sistema UCP-memoria con el exterior



Unidades Funcionales del Computador FCO

- Periférico
  - Dispositivo que permite la comunicación con el exterior
    - Memoria secundaria (almacenamiento)
    - Entrada/salida de datos (comunicación)

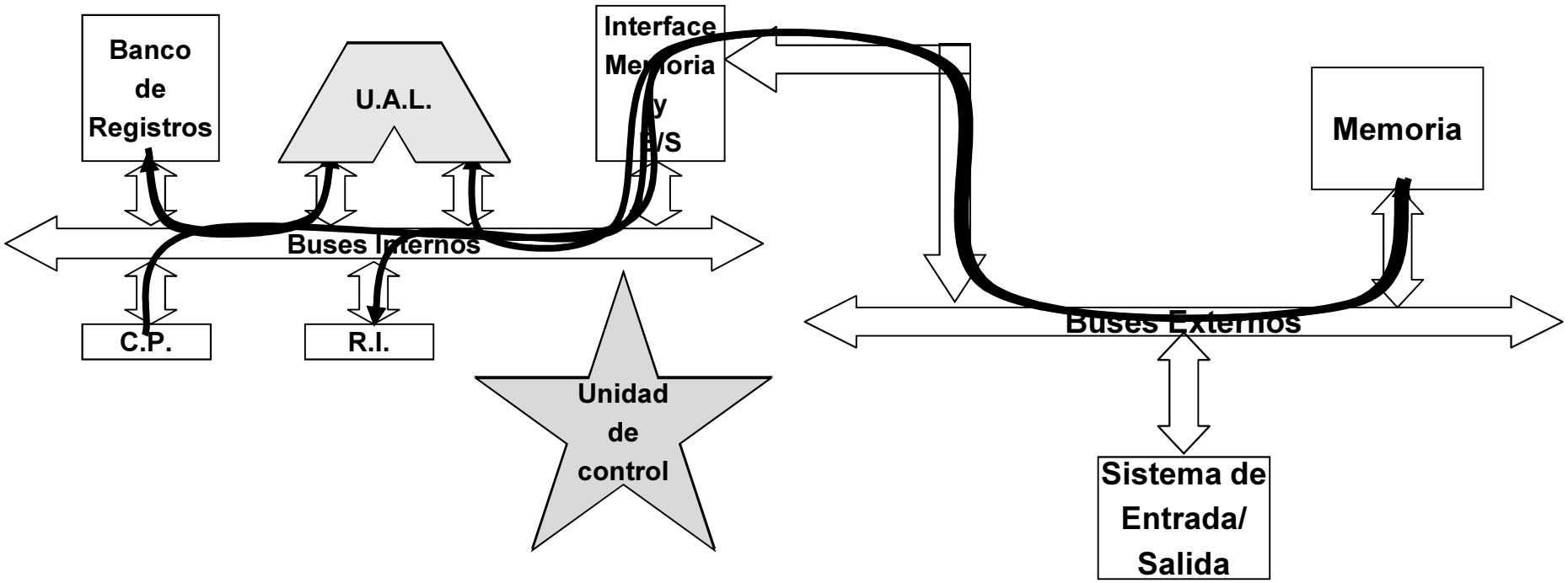
- Interfaz o controlador
  - Dispositivo hardware/software que permite la comunicación entre la UCP (o el sistema de memoria) y el periférico
  - Independiza la UCP del periférico
    - Puede estar en el periférico (ejemplo: discos IDE)

Unidades Funcionales del Computador FCO

- Unidad de control
  - Genera las señales necesarias para que la unidad de proceso ejecute las instrucciones de forma adecuada
  - Indica el tipo de operación que tiene que realizar la ALU
  - Indica que registros contienen los datos y dónde se debe almacenar el resultado
  - Genera señales de carga de todos los registros cuando estos tienen que almacenar información
  - Es un sistema secuencial. Su complejidad depende de la complejidad de la unidad de proceso y del número y tipo de instrucciones a ejecutar

Etapas en la ejecución de una instrucción FCO

- Búsqueda de la instrucción a ejecutar
- Decodificación de la instrucción
- Búsqueda de los operandos
- Realización de la operación
- Almacenamiento del resultado



Arquitectura del Juego de Instrucciones FCO

- Arquitectura del Juego de Instrucciones
  - Visión de la máquina que tiene el programador en lenguaje ensamblador
- La arquitectura del Juego de Instrucciones está determinada por:
  - Tipos de datos
  - Espacio lógico direccionable (organización de la memoria)
  - Conjunto de registros
  - Conjunto de instrucciones (Juego de instrucciones)
  - Modos de direccionamiento

MIPS R2000: Representación y tipos de datos FCO

- Tipos de datos soportados por el MIPS R2000

Tipo de datos	Representación	Tamaño	Nombre
Caracteres	Ascii	8 bits	Ascii
Enteros	Ca2	8 bits	Byte
		16 bits	Half
		32 bits	Word
Reales	IEEE 754	32 bits	Float
		64 bits	Double

- Float : Números reales representados según el estándar IEEE 754 de simple precisión
- Double: Números reales representados según el estándar IEEE 754 de doble precisión

MIPS R2000: Organización de la memoria FCO

- Memoria. Puede verse como un vector unidimensional.
  - Dirección de memoria. Índice dentro del vector.
  - Memoria direccionada byte a byte.
    - Los índices apuntan a cada byte de la memoria. El byte es la mínima unidad direccionable

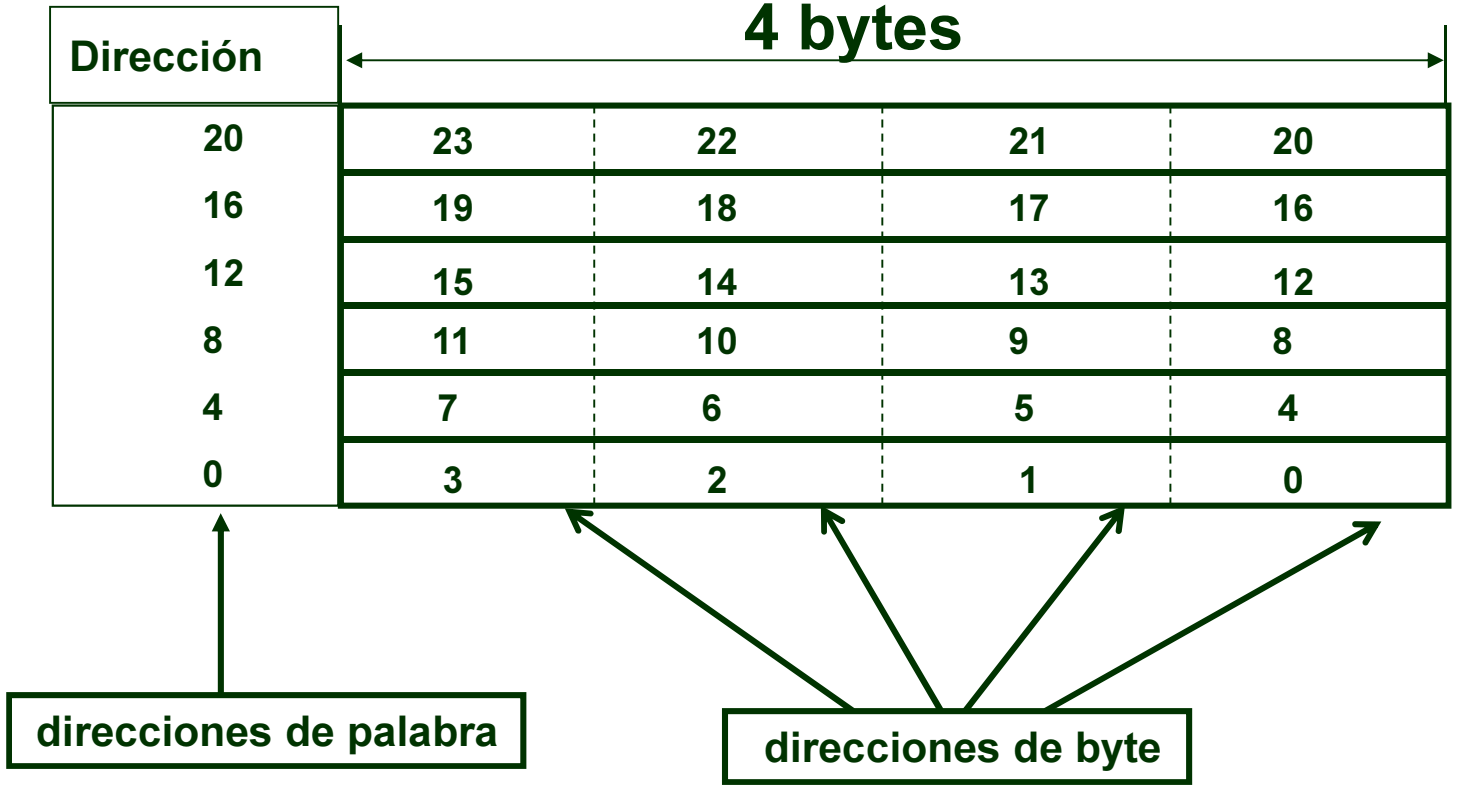
Dirección	Contenido
3	1 byte de datos
2	1 byte de datos
1	1 byte de datos
0	1 byte de datos

- En el MIPS R2000 la memoria se organiza en palabras de 4 bytes.
  - Cada palabra (word) ocupa 4 posiciones (bytes) o direcciones de memoria
  - Es posible acceder a la memoria de las siguientes formas:
    - Por Byte → Cualquier dirección
    - Por Word → Direcciones múltiplo de 4 (0, 4, 8, …)
    - Por media palabra o Half → Direcciones múltiplos de 2 (0, 2, 4, …)

MIPS R2000: Organización de la memoria

FCO

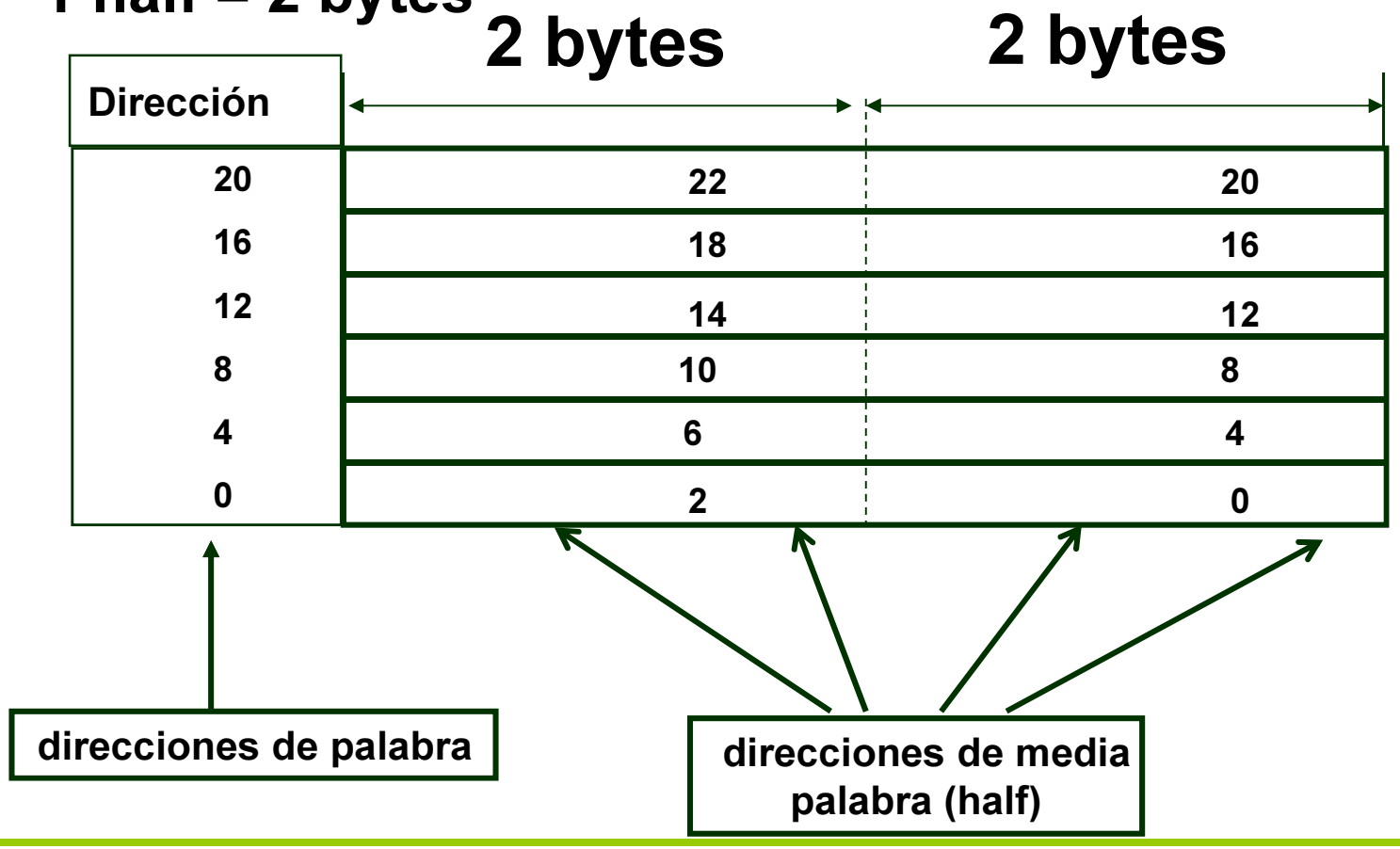
- **Acceso por palabra (word)**
  - **1 word = 4 bytes**



DISCA

MIPS R2000: Organización de la memoria FCO

- **Acceso por media palabra (half)**
  - **1 half = 2 bytes**



MIPS R2000: Organización de la memoria

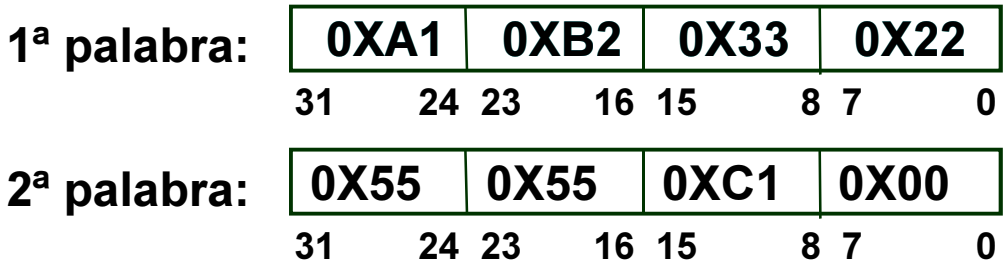
FCO

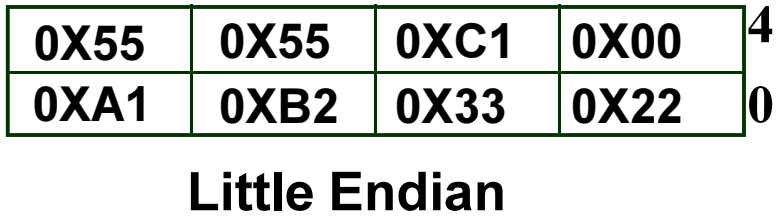
- **Existen 2 formas de almacenar la información en memoria:**
  - **Formato big endian**
    - **El byte de mayor peso del dato o instrucción se almacena en la dirección de memoria más baja.**
  - **Formato little endian**
    - **El byte de menor peso del dato o instrucción se almacena en la dirección de memoria más baja.**

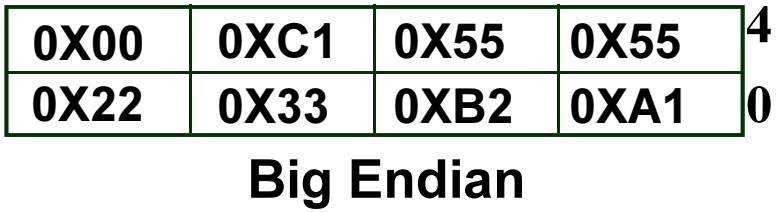
DISCA

MIPS R2000: Organización de la memoria FCO

- **Ejemplo 1. Almacenar en memoria las dos palabras siguientes utilizando ambos formatos:**



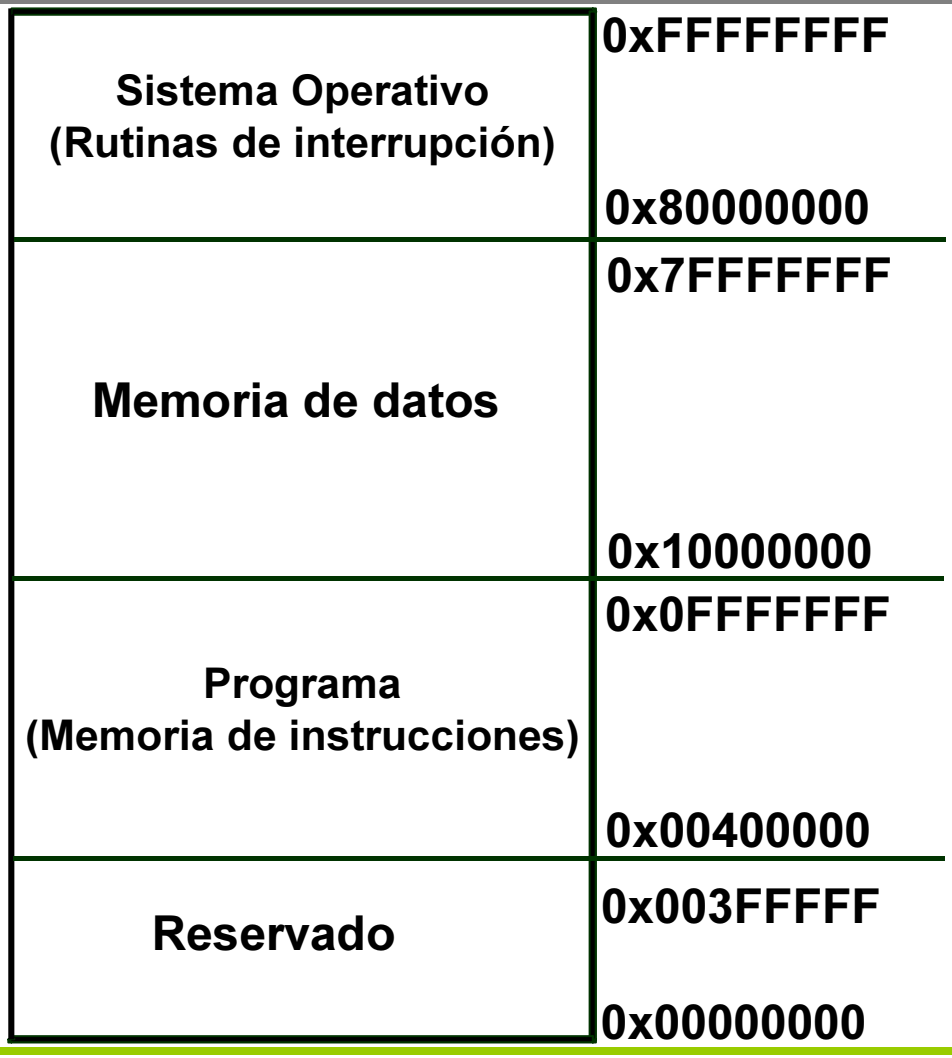




MIPS R2000: Organización de la memoria FCO

- **Espacio direccionable en el MIPS R2000**
  - **Bus de direcciones de 32 bits**
  - **4 Gbytes (4G x 8bits) = 2 Ghalf = (2G x 16bits) = 1 Gword = (1G x 32bits).**
  - **2^{32} bytes con direcciones desde 0 a $2^{32} - 1$**
  - **2^{30} palabras (4 bytes) con direcciones: 0, 4, 8, ..., $2^{32} - 4$**

Memoria accesible por el usuario se encuentra en el rango [0x00400000, 0x7FFFFFFF]



DISCA

Programas en ensamblador FCO

- Directivas
  - Las directivas del ensamblador sirven para ubicar los datos y las instrucciones en la memoria del procesador
  - Sus nombres empiezan con un punto y se emplean corchetes para indicar que uno o varios argumentos son opcionales
    - .nombre_directiva argumento1 [, argumento2]
  - Se agrupan en tres tipos:
    - Directivas para reserva de posiciones de memoria
    - Directivas para indicar el inicio del área de datos y de instrucciones
    - Directivas de propósito variado
  - NO son instrucciones, no ocupan memoria.

Programas en ensamblador FCO

- Directiva “.space”
  - Sintaxis: .space n
  - Reserva n bytes de memoria y los inicializa a 0x00
- Directivas “.ascii” y “.asciiz”
  - Sintaxis: .ascii ‘cadena1’ ,[‘cadena2’, …’cadena n’]
    .asciiz ‘cadena1’, [‘cadena2’, …’cadena n’]
  - Se emplea para almacenar cadenas de caracteres codificados en ASCII en posiciones consecutivas de memoria.
  - La directiva “.asciiz” añade un carácter NULL (0x00) al final de la cadena.

Programas en ensamblador FCO

- Directiva “.byte”
  - Sintaxis: “.byte b1, [b2, b3, …]”
  - Inicializa posiciones consecutivas de memoria con los enteros de 1 byte codificados en Ca2
- Directiva “.half”
  - Sintaxis: “.half h1, [h2, h3, …]”
  - Inicializa posiciones consecutivas de memoria con los enteros de 2 bytes codificados en Ca2
- Directiva “.word”
  - Sintaxis: “.word w1, [w2, w3, …]”
  - Inicializa posiciones consecutivas de memoria con los enteros de 4 bytes codificados en Ca2
- En todos los casos se pueden especificar los datos en decimal o hexadecimal..

Programas en ensamblador FCO

- Directiva “.float”
  - Sintaxis: “.float f1, [f2, f3, …]”
  - Inicializa posiciones consecutivas de memoria con los reales de 4 bytes codificados en IEEE 754 de simple precisión
- Directiva “.double”
  - Sintaxis: “.double d1, [d2, d3, …]”
  - Inicializa posiciones consecutivas de memoria con los reales de 4 bytes codificados en IEEE 754 de doble precisión
- En ambos casos se pueden especificar los datos en notación decimal o en notación científica.

Programas en ensamblador FCO

- Directiva “.data”
  - Sintaxis: “.data dirección”
  - Ubica los datos que se especifican en las directivas de datos a partir de la dirección especificada
  - Si no se especifica una dirección, se toma 0x10000000
- Directiva “.text”
  - Sintaxis “.text dirección”
  - Ubica las instrucciones a partir de la dirección especificada
  - Si no se especifica una dirección, se toma 0x00400000
- Directiva “.end”
  - Sintaxis “.end”
  - Indica el final de un programa

Programas en ensamblador FCO

- Alineación de datos en memoria
  - .space, .byte, .ascii y .asciiz reservan memoria a partir de cualquier dirección
  - .half reserva memoria a partir de direcciones múltiplos de 2
  - .word y .float reservan memoria a partir de direcciones múltiplos de 4
  - .double reserva memoria a partir de direcciones múltiplos de 8
- El simulador del MIPS almacena los datos en formato Little Endian

Programas en ensamblador FCO

Ejemplo alineación de datos:

```
.data 0x10000020
        .word  0x20000001
        .space 1
        .word 0x10
        .byte  0x1, 0x2, 0x3
        .double 1.0  #0x3FF0000000000000 IEEE doble precisión
```

Contenido en hexadecimal 31 ..... 24 23 ..... 16 15 ..... 8 7 0	Dirección mem
0x20000001	0x10000020
0x00	0x10000024
0x00000010	0x10000028
0x03 | 0x02 | 0x01	0x1000002C
0x00000000	0x10000030
0x3FF00000	0x10000034

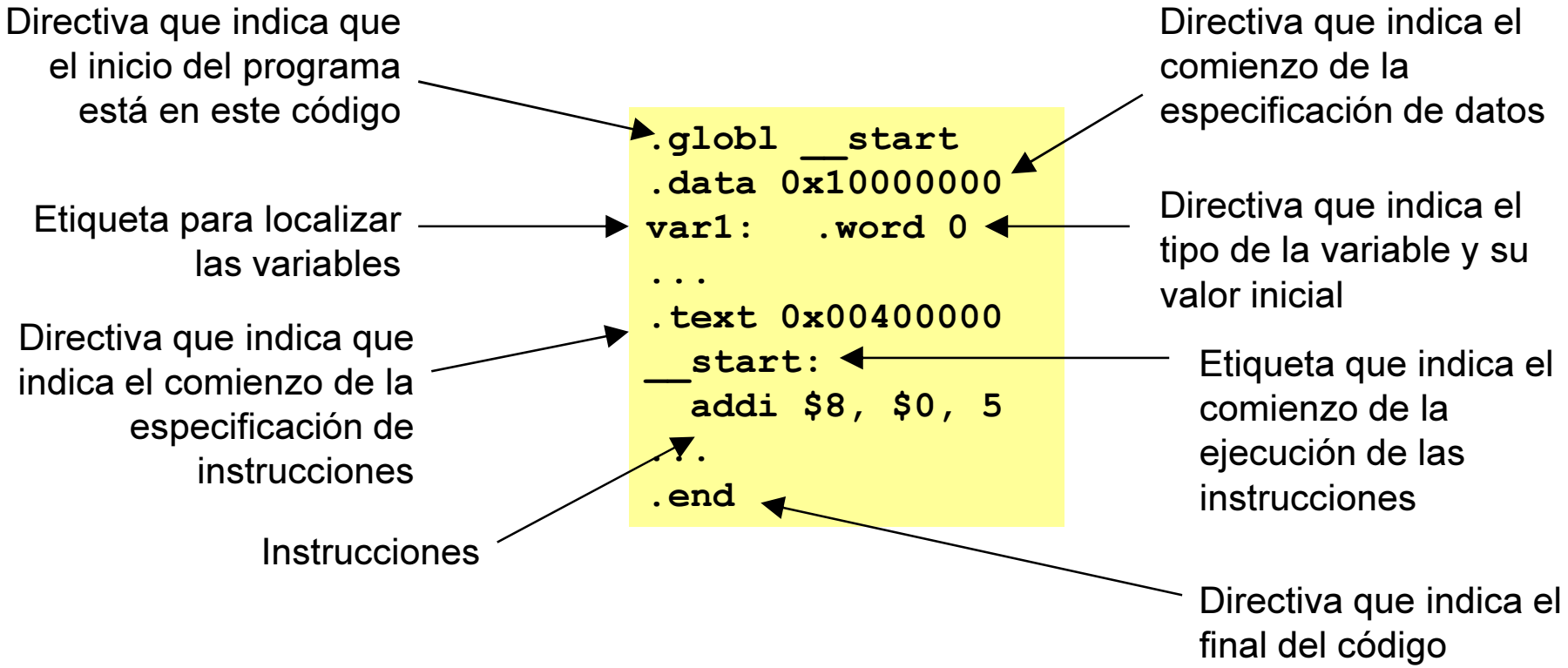
DISCA

Programas en ensamblador FCO

- Etiquetas
  - Sintaxis: “etiqueta:”
  - Indican una posición de memoria importante, por ejemplo variables en el segmento de datos o puntos a donde saltar en el segmento de instrucciones.
- “__start”
  - Etiqueta que indica el inicio del programa, es decir de la primera instrucción que comenzará a ejecutarse.
  - Sólo puede haber una etiqueta “__start” en un programa, para ello se emplea la directiva “.globl etiqueta” que indica que la etiqueta definida es común a todos los archivos del programa.

Programas en ensamblador FCO

- Estructura de un programa escrito en ensamblador del MIPS R2000



MIPS R2000: Banco de Registros (BR) FCO

- Características del banco de registros del MIPS R2000
  - Banco de registros de Enteros:
    - 32 Registros de 32 bits de propósito general
      - Se identifican como $0 a $31
      - Ejemplo: add $2, $3, $4 → ($2←$3+$4)
    - 2 Registros especiales de 32 bits HI y LO.
      - Almacenan los resultados de multiplicaciones y divisiones.
  - Banco de registros de Reales, que puede utilizarse como:
    - 32 Registros de 32 bits con formato IEEE 754 de simple precisión. Se identifican por $f0 a $f31
    - 16 registros de 64 bits con formato IEEE 754 de doble precisión. Se identifican por $f0, $f2, $f4, …, $f30
- El registro $0 tiene permanentemente el valor 0

Juego de instrucciones

FCO

- Instrucciones
  - Lenguaje del Computador
  - El conjunto de instrucciones del MIPS R2000 que se estudiará es similar al utilizado en otras arquitecturas como Sony o Nintendo
- Cada instrucción en lenguaje ensamblador tiene un formato según el cual se codificará en lenguaje máquina (0's y 1's)
- Tipos de instrucciones:
  - Instrucciones aritméticas
  - Instrucciones lógicas
  - Instrucciones de carga y almacenamiento
  - Instrucciones de movimiento
  - Instrucciones de comparación
  - Instrucciones de salto condicional
  - Instrucciones de salto incondicional

DISCA

Juego de instrucciones FCO

- En la descripción de las diferentes instrucciones se utilizará la siguiente nomenclatura:
  - rd, rs y rt representan registros de propósito general
  - $rt_{31..16}$: Parte alta del registro rt
  - $rt_{15..0}$: Parte baja del registro rt
  - desp+rs, da una dirección de memoria, resultado de sumar al contenido del registro rs una cantidad llamada desplazamiento
  - M[desp+rs] es el contenido de la posición de memoria desp+rs

datos	direcciones
0x00000032	0x00000010
0x00007654	0x0000000C
0x00000543	0x00000008
0x00000000	0x00000004
0x00000100	0x00000000

rs = 0x00000004
desp = 8

} Desp+rs = 0x0000000C
M[desp+rs] = 0x7654

Juego de instrucciones FCO

- Instrucciones aritméticas

Sintaxis	Formato	Descripción
add rd, rs, rt	R	rd ← rs + rt
addi rt, rs, inm	I	rt ← rs + inm (inmediato de 16 bits representado en Ca2)
sub rd, rs, rt	R	rd ← rs – rt
mult rs, rt	R	Multiplica rs por rt dejando los 32 bits de menor peso en el registro LO y los 32 bits de mayor peso en HI
div rs, rt	R	Divide rs entre rt dejando el cociente en el registro LO y el resto en el registro HI

- Instrucciones de movimiento de datos

Sintaxis	Formato	Descripción
mfhi rd	R	rd ← HI
mflo rd	R	rd ← LO
mthi rs	R	HI ← rs
mtlo rs	R	LO ← rs

Juego de instrucciones

FCO

Suma y resta

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 5
  addi $9, $0, 6
  add $10, $8, $9
  sub $11, $8, $9
.end
```

Multiplicación

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 2
  addi $9, $0, 3
  mult $8, $9
  mflo $10
  mfhi $11
.end
```

División

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 4
  addi $9, $0, 11
  div $9, $8
  mflo $10
  mfhi $11
.end
```

- ¿Cuál es el contenido de los registros $10 y $11 al finalizar la ejecución de cada código?
- ¿ Cuál es el contenido de los registros $10 y $11 si se sustituyen las siguientes instrucciones?

Suma y resta `sub $11, $8, $9` **por** `sub $11, $9, $8`

División `div $9, $8` **por** `div $8, $9`

DISCA

Juego de instrucciones FCO

- Instrucciones lógicas

Sintaxis	Formato	Descripción
and rd, rs, rt	R	rd ← rs and rt, la operación lógica indicada se realiza bit a bit
nor rd, rs, rt	R	rd ← rs nor rt
xor rd, rs, rt	R	rd ← rs xor rt
or rd, rs, rt	R	rd ← rs or rt
andi rt, rs, inm	I	rt← rs and inm, (el dato inmediato, es de 16 bits y se extiende con 16 ceros)
ori rt, rs, inm	I	rt← rs or inm
xori rt, rs, inm	I	rt← rs xor inm
sll rd, rt, desp	R	rd← rt << desp, desplazamiento a izquierdas, conforme desplaza se rellena con 0
srl rd, rt, desp	R	rd← rt >> desp, desplazamiento a derechas, conforme desplaza se rellena con 0

Juego de instrucciones

or y and

```
.globl __start
.text 0x00400000
__start:
  ori $8, $0, 0x000a
  ori $9, $0, 0x000c
  or $10, $8, $9
  and $11, $8, $9
.end
```

nor y xor

```
.globl __start
.text 0x00400000
__start:
  ori $8, $0, 0x000a
  ori $9, $0, 0x000c
  nor $10, $8, $9
  xor $11, $8, $9
.end
```

- Realizar las operaciones bit a bit
- ¿Cuál es el contenido de los registros $10 y $11 al finalizar la ejecución de cada código?
- ¿De qué forma puede realizarse la operación NOT?
- ¿De qué forma puede realizarse la operación NAND?

Juego de instrucciones

FCO

A izquierdas

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 6
  sll $9, $8, 1
  sll $10, $8, 2
.end
```

A derechas

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 6
  srl $9, $8, 1
  srl $10, $8, 2
.end
```

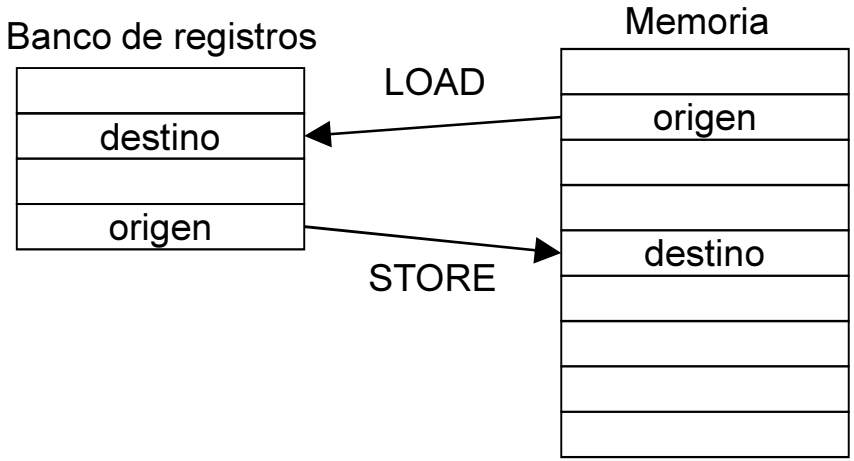
- ¿Cuál es el contenido de los registros $9 y $10 al finalizar la ejecución de cada código?
- ¿Qué operación aritmética está implementando el desplazamiento a izquierdas?
- ¿Qué operación aritmética está implementando el desplazamiento a derechas?

DISCA

Juego de instrucciones

FCO

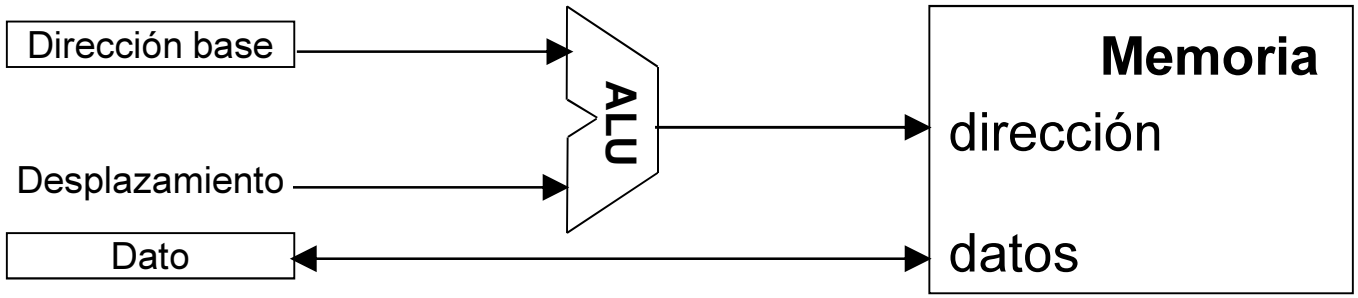
- Instrucciones de carga y almacenamiento
  - Son las únicas que acceden a la memoria
    - Load: lee el contenido que indica una dirección de memoria y lo almacena en un registro del banco de registros.
    - Store: lee el contenido de un registro del banco de registros y lo almacena en la memoria, en la dirección indicada.



DISCA

Juego de instrucciones FCO

- Instrucciones de carga y almacenamiento
  - Cálculo de la dirección, tanto para las instrucciones de carga como las de almacenamiento :
    - Dirección base: registro con la dirección de memoria base.
    - Desplazamiento: entero de 16 bits con el desplazamiento. Va incluido en la instrucción.
    - Dirección efectiva de memoria = Dirección base + Desplazamiento
    - Dato: registro con los datos para cargar (load) desde la memoria (extendiendo el signo) o almacenar (store) a la memoria



Juego de instrucciones FCO

- Instrucciones de carga y almacenamiento

Sintaxis	Formato	Descripción
lw rt, desp(rs)	I	rt ← M[desp+rs],
lh rt, desp(rs)	I	rt ← M[desp+rs], carga media palabra (16 bits) y extiende el signo
lb rt, desp(rs)	I	rt ← M[desp+rs], carga 1 byte (8 bits) y extiende el signo
sw rt, desp(rs)	I	M[desp+rs] ← rt
sh rt, desp(rs)	I	M[desp+rs] ← rt Almacena la parte baja de rt en memoria
sb rt, desp(rs)	I	M[desp+rs] ← rt Almacena el byte menos significativo en memoria
lui rt, inm	I	$rt_{31\,..\,16}$ ← inm $rt_{15\,..\,0}$ ←0

Juego de instrucciones

FCO

Lectura de memoria

```
.globl __start
.data 0x10000000
var_a: .byte 2
var_b: .half -4
var_c: .word 6

.text 0x00400000
__start:
  lui $8, 0x1000
  lb $9, 0($8)
  lh $10, 2($8)
  lw $11, 4($8)
.end
```

- ¿Cuál es el contenido de la memoria antes de lanzar el código?
- ¿En qué direcciones de memoria están almacenadas las variables “var_a”, “var_b” y “var_c”?
- ¿Cuál es el contenido de los registros $8, $9, $10 y $11 al finalizar la ejecución del código?
- ¿Qué se debería modificar (cambiar o ampliar) en el código si se sustituyese la directiva de datos `.data 0x10000000` por `.data 0x10000010` ?

DISCA

Juego de instrucciones

FCO

Escritura en memoria

```
.globl __start
.data 0x10000000
var_a: .space 1
var_b: .space 2
var_c: .space 4

.text 0x00400000
__start:
  addi $8, $0, -1
  addi $9, $0, 2
  addi $10, $0, 4
  lui $11, 0x1000
  sb $8, 0($11)
  sh $9, 2($11)
  sw $10, 4($11)
.end
```

- ¿Cuál es el contenido de la memoria antes de lanzar el código?
- ¿Cuál es el contenido de la memoria después de lanzar el código?
- ¿Cuál es el contenido de los registros $8, $9, $10 y $11 al finalizar la ejecución del código?

DISCA

Juego de instrucciones

- Instrucciones de comparación

Sintaxis	Formato	Descripción
slt rd, rs, rt	R	Si rs < rt entonces rd ← 1, si no rd ← 0
slti rt, rs, inm	I	Si rs < inm entonces rt ← 1, si no rt ← 0

Comparación inmediata

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 1
  addi $9, $0, 3
  slti $10, $8, 2
  slti $11, $9, 2
.end
```

Comparación

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 1
  addi $9, $0, 3
  slt $10, $8, $9
  slt $11, $9, $8
.end
```

- ¿Cuál es el contenido de los registros $10 y $11 al finalizar la ejecución de cada código?

Juego de instrucciones FCO

- Instrucciones de salto condicional

Sintaxis	Formato	Descripción
beq rs, rt, etiqueta	I	Si rs = rt entonces salta a la dirección etiqueta PC← etiqueta
bne rs, rt, etiqueta	I	Si rs <> rt entonces salta a la dirección etiqueta PC ← etiqueta

- Instrucciones de salto incondicional

Sintaxis	Formato	Descripción
j etiqueta	J	PC← etiqueta, salta a la dirección etiqueta
jal etiqueta	J	Salta a la dirección etiqueta guardándose previamente la dirección de retorno en $31 $31← PC actualizado PC← etiqueta
jr rs	R	PC ← rs, salta a ladirección contenida en el registro rs

Juego de instrucciones

FCO

Salto de igualdad

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 1
  addi $9, $0, -1
  beq $8,$0,fin
  addi $9, $0, 1
fin:
.end
```

Salto de desigualdad

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 1
  addi $9, $0, -1
  bne $8,$0,fin
  addi $9, $0, 1
fin:
.end
```

- ¿Cuál es la dirección de memoria a la que apunta la etiqueta “fin”?
- ¿Cuál es el contenido del registro $9 al finalizar la ejecución de los programas anteriores?

DISCA

Juego de instrucciones

FCO

Salto incondicional

```
.globl __start
.text 0x00400000
__start:
   addi $8, $0, 3
   addi $9, $0, 1
bucle:
   beq $8, $0, fin
   mult $9, $8
   mflo $9
   addi $8, $8, -1
   j bucle
fin:
.end
```

- ¿Cuál es la dirección de memoria a la que apunta la etiqueta “bucle”?
- ¿Cuál es el contenido de los registros $8 y $9 al finalizar la ejecución del programa?
- El programa ejemplo ¿realiza alguna operación conocida?¿cuál?
- En la operación que implementa el programa ejemplo ¿Qué representan $8 y $9?

DISCA

Pseudo instrucciones

FCO

- Definición:
  - Conjunto de instrucciones que el lenguaje ensamblador incorpora además de las instrucciones máquina del procesador.
  - Una pseudoinstrucción se traduce en un número de instrucciones: de 1 a 4 instrucciones máquina.
  - El programador puede usar las pseudoinstrucciones como si se tratase de instrucciones máquina soportadas por el procesador.
  - El programa ensamblador es el encargado de sustituir dichas pseudoinstrucciones por las instrucciones máquina correspondientes.
  - El SPIM reserva el registro $1, para almacenamiento auxiliar en caso de necesitar más de una instrucción
- Ejemplo:
  - Cargar la dirección de memoria de una etiqueta en un registro
  - Inicializar un dato en un registro
  - Cubrir todas las posibles comparaciones o saltos condicionales

DISCA

Pseudo instrucciones

FCO

- li: Load Inmediate:
  - Sintaxis: li R, inm
  - Descripción: Carga en el registro indicado por R el entero con signo en complemento a 2 de 32 bits inm
  - Traducción: puede traducirse en lui, lui y ori, u ori. Dependiendo del dato que se debe cargar en el registro.
  - Ejemplos
    - li $8, 1 → ori $8, $0, 1
    - li $9, 0x10000001 → lui $1, 0x1000; ori $9, $1, 0x0001
    - li $10, 0x10000000 → lui $10, 0x1000

DISCA

Pseudo instrucciones

FCO

- la: Load Address:
  - Sintaxis: la R, etiqueta
  - Descripción: Carga en el registro R la dirección de memoria a la que hace referencia etiqueta
  - Traducción: puede traducirse en lui, lui y ori. Dependiendo de la dirección de la etiqueta
  - Ejemplos
    - Ante esta definición de datos →

```
.data 0x10000000
var_a:  .word 1
var_b:  .word 2
```

    - la $8, var_a → lui $8, 0x1000
    - la $9, var_b → lui $1, 0x1000; ori $9, $1, 4

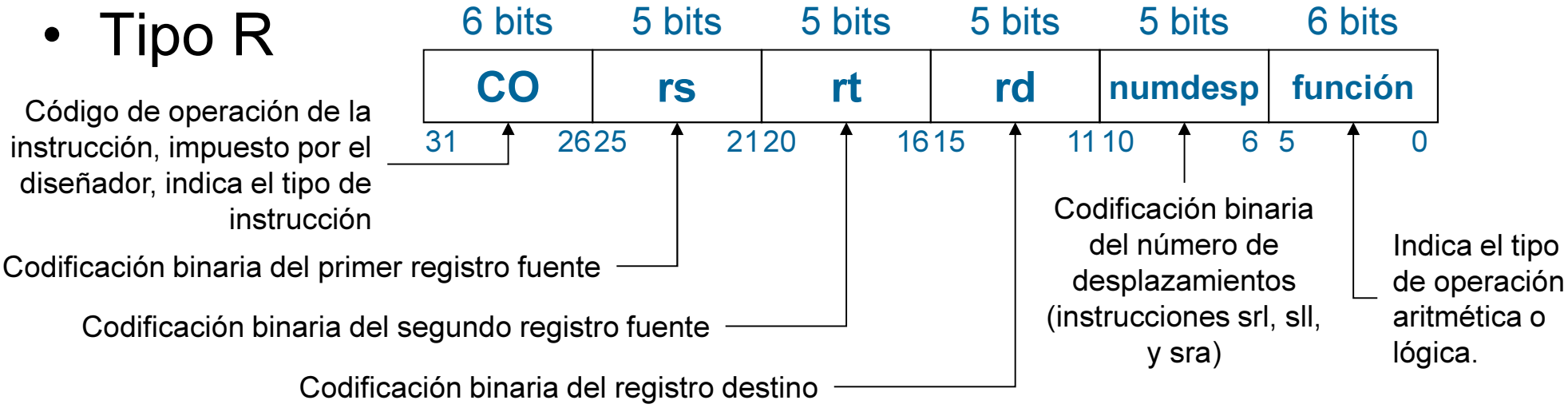
DISCA

Codificación de instrucciones FCO

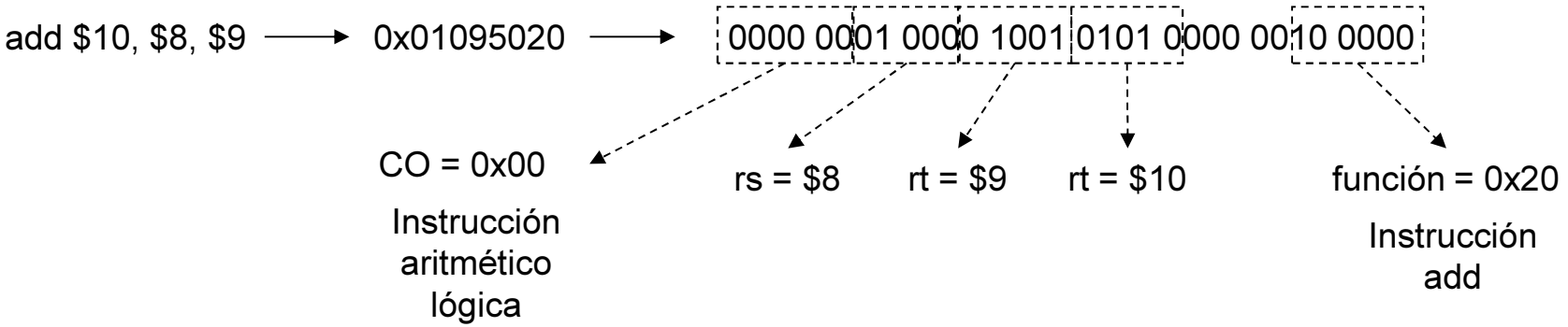
- El formato de las instrucciones indica la forma en que se codifican dichas instrucciones en código máquina.
- Todas las instrucciones del MIPS R2000 tienen un tamaño de 32 bits.
- En el MIPS R2000 se distinguen tres tipos de formatos en función de los componentes del procesador que utilizan las instrucciones:
  - Tipo R o de tipo registro.
  - Tipo I o de tipo inmediato.
  - Tipo J o de salto incondicional.
- Todos los tipos tienen en común los primeros seis bits, formando un campo conocido como CO (Código de Operación)

Codificación de instrucciones FCO

- Tipo R

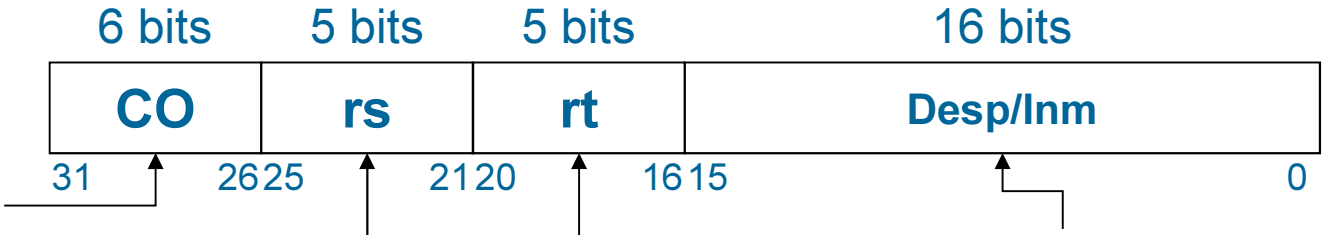


- Ejemplo



Codificación de instrucciones FCO

- Tipo I



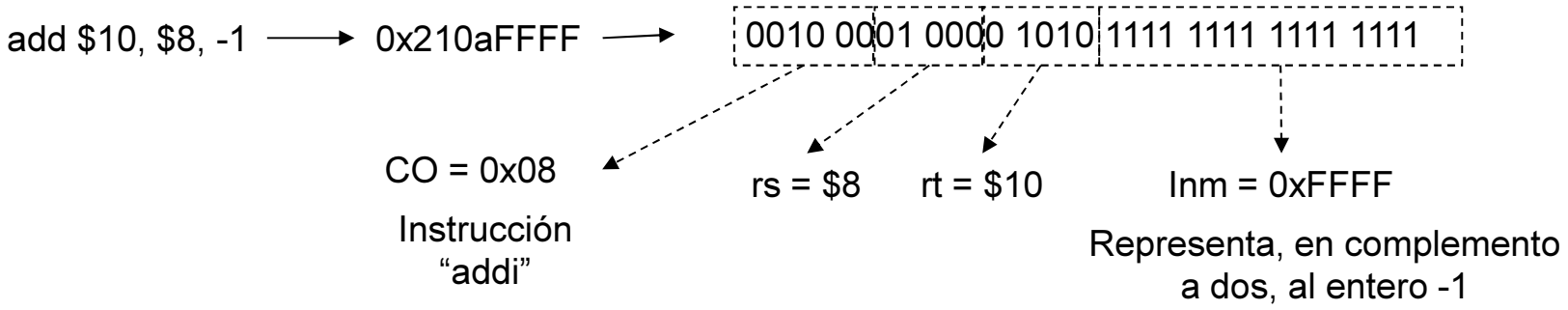
Código de operación de la instrucción, impuesto por el diseñador, indica el tipo de instrucción

Codificación binaria del primer registro fuente

Codificación binaria del segundo registro fuente

16 bits codificados en Complemento a dos:
- Desplazamiento expresado en número de palabras para las instrucciones BEQ y BNE
- Desplazamiento en número de bytes para las instrucciones de Carga y almacenamiento

- Ejemplo con un dato inmediato



Codificación de instrucciones FCO

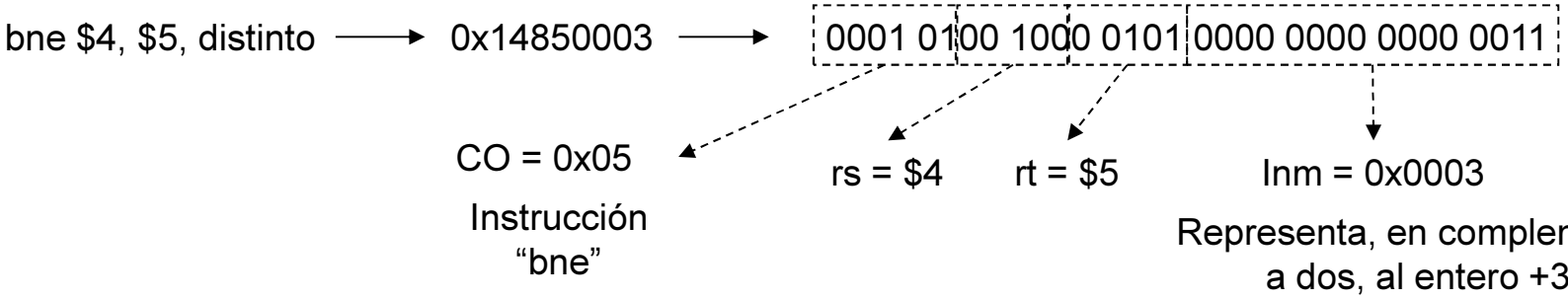
- Tipo I: Ejemplo con un desplazamiento condicional

```
.text 0x0040000
__start:
  lw $4, 8($10)
  bne $4, $5, distinto
  sub $4, $4, $5
  j salida
distinto: add $4, $4, $5
salida:   addi $5, $5, 1
```

El contador de programa (CP) apunta a la instrucción que se está ejecutando ("bne")

La etiqueta "distinto" apunta a la instrucción "add" que se encuentra a tres instrucciones del CP

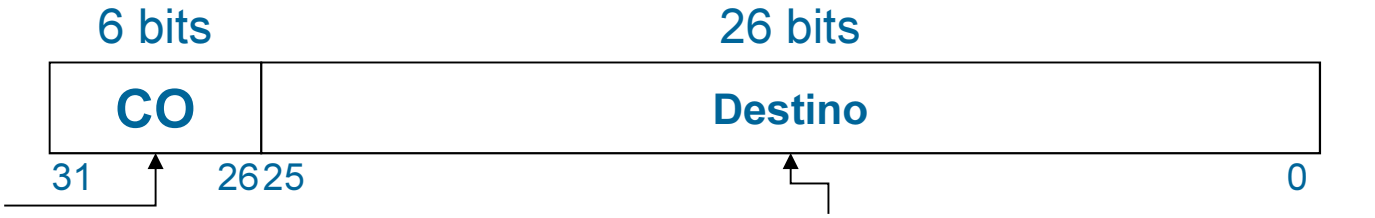
Cada instrucción se codifica en 32 bits, por lo que ocupa una palabra en memoria. Por tanto, las direcciones de memoria donde se encuentran las instrucciones siempre serán múltiplos de cuatro: 0, 4, 8, 12, 16, etc. El campo desplazamiento para la instrucción condicional ("bne" en este caso) expresa el salto en número de palabras de memoria (instrucciones) a saltar



Codificación de instrucciones FCO

- Tipo j

Código de operación de la instrucción, impuesto por el diseñador, indica el tipo de instrucción



Dirección de memoria a la que apunta la etiqueta destino del salto

- Ejemplo (del código de la transparencia anterior)

j salida → 0x08100005 → 0000 1000 0001 0000 0000 0000 0000 0101

CO = 0x02

Instrucción "j"

Inm = 00 0001 0000 0000 0000 0000 0101

Añadido = 0000 00

Resultado = 0000 0000 0100 0000 0000 0000 0001 0100

Dirección de salto = 0x00400014

Sólo 26 bits de los 32 del Contador de Programa (PC) son modificados por j permaneciendo los otros 6 sin cambios.
La dirección de salto (32 bits) debe codificarse en el campo destino de 26 bits, por lo que se prescinde de los dos bits de menor peso y de los 4 bits de mayor peso por ser siempre ceros

Lenguajes informáticos

FCO

- Un lenguaje de programación es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras.
  - Dependiente de la máquina
  - Independiente de la máquina

Lenguaje	Código
Lenguaje Java	a=b+c;
Lenguaje ensamblador del MIPS	add $3,$2,$9
Lenguaje máquina del MIPS	0000 0000 0100 1001 0001 1000 0010 0000

DISCA

Lenguajes informáticos FCO

- Software: es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.
- Tipos de Software Según el IEEE 729
  - Software de Sistema
  - Software de Programación
  - Software de Aplicación
- El Software adopta varias formas en distintos momentos de su ciclo de vida
  - Código Fuente
  - Código Objeto
  - Código Ejecutable

Sistemas operativos FCO

- ¿Qué es un Sistema Operativo?
  - Un programa que actúa como intermediario entre el usuario de un computador y el hardware del mismo
- Objetivos del sistema operativo:
  - Ejecutar programas y facilitar la solución de los problemas del usuario
  - Hacer un uso conveniente del computador
- Usar el computador de forma eficiente
- Proporcionar una máquina virtual extendida

Sistemas operativos FCO

- SO comerciales:
  - Linux
  - Windows XP, Vista, 7
  - Mac OS X

- PDAS
  - Linux
  - Symbian
  - Windows Mobile
  - Palm OS
  - Android
  - Iphone os

Sistemas operativos: Llamadas al sistema FCO

- Es el mecanismo usado por una aplicación para solicitar un servicio al sistema operativo.
- Interfaz entre aplicaciones y SO.
  - Generalmente disponibles como funciones en ensamblador.
- Servicios típicos del sistema operativo
  - Gestión de procesos
  - Gestión de procesos ligeros
  - Gestión de señales, temporizadores
  - Gestión de memoria
  - Gestión de ficheros y directorios
- Ejemplos de llamada
  - read: permite leer datos de un fichero
  - fork: permite crear un nuevo proceso

Procesamiento avanzado FCO

- Para aumentar la potencia y la eficiencia en el procesamiento se tienen diferentes modelos de procesamiento más avanzado.
- La optimización se fundamenta en diversos conceptos
  - Paralelización de procesos.
  - Utilización de recursos compartidos.
  - Especialización de componentes.
- Existen diversos modelos
  - Multiprocesadores.
  - Supercomputadores.
  - Redes de Computadores.
  - Sistemas Distribuidos.

Multiprocesadores y Supercomputadores FCO

- Multiprocesadores
  - Computador con dos o más procesadores.
  - Permiten el procesamiento en paralelo procesando simultáneamente varios “hilos” pertenecientes a un mismo proceso o bien de procesos diferentes.
  - Generan ciertas dificultades de coordinación especialmente en el acceso a memoria.
  - Precisan que el Sistema Operativo esté preparado para poder obtener toda la potencia que ofrecen.
- Supercomputadores
  - Sistemas con múltiples computadores conectados directamente por medio de conexiones dedicadas.
  - Proporcionan una potencia de cómputo mucho más elevada que los computadores comunes.

- Redes de Computadores
  - Conjunto de computadores conectados por medios de comunicación (cables, ondas, etc.) que les permite compartir recursos (unidades de almacenamiento, impresoras, etc.)
  - Existen diversos modelos de implementación: ISO/OSI, TCP/IP, ATM, etc.
- Sistemas Distribuidos
  - Sistemas basados en redes de computadores donde los computadores se comunican a través de servicios (modelo cliente-servidor).
  - Un servidor también puede ser cliente y viceversa
  - Deben ser confiables: si un componente falla, otro componente debe de ser capaz de reemplazarlo (tolerancia a fallos).

UNIVERSIDAD POLITECNICA DE VALENCIA

etsinf

DISCA

Fundamentos de computadores

Tema 7. LENGUAJE ENSAMBLADOR