Bloque Temático
Lenguaje Ensamblador

- El procesador
- Lenguaje ensamblador: Conceptos
- Programación con subrutinas

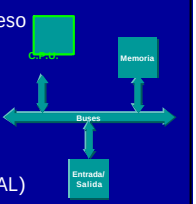
OBJETIVOS:

- **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.**
- **Obtener la codificación en lenguaje máquina de las diferentes instrucciones del lenguaje ensamblador del MIPS R2000**

Índice de Contenidos

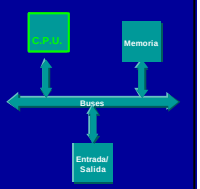
1. Introducción (I)

- Procesador o Unidad Central de Proceso (UCP)
  - Unidad Funcional que ejecuta las instrucciones almacenadas en memoria.
  - Formado principalmente por:
    - La Unidad Aritmético-Lógica (UAL)
    - La Unidad de Control (UC)
    - El Banco de Registros (BR)
  - Actualmente los procesadores también disponen de memorias internas



1. Introducción (II)

- Filosofías de diseño
  - **“**Computadores con un juego de instrucciones complejo” *(Complex Instruction Set Computer o CISC)*
  - “Computadores con un juego de Instrucciones Reducido” *(Reduced Instruction Set Computer o RISC)*

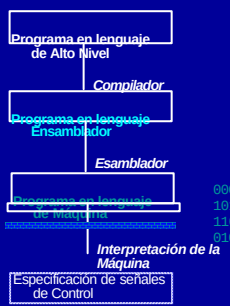


Hoy en día un procesador RISC no tiene un conjunto de instrucciones reducido, sino que es aquel que pretende ejecutar a la máxima velocidad las instrucciones más comunes, aún a costa de ralentizar las que menos se utilizan.

Introducción (III)

- ? instrucciones ? datos tienen que representarse mediante 0`s y 1's ⇨ interpretados por los diferentes circuitos del computador → Código Máquina
- La programación en Lenguaje Ensamblador intenta facilitar las cosas al programador sustituyendo los códigos numéricos (0`s y 1's) por códigos nemotécnicos.
  - **Lenguaje máquina: 0000 0000 0100 1001 0001 1000 0010 0000**
  - **Lenguaje ensamblador: add $3,$2,$9**

Niveles de Representación




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw    $15, 0($2)
lw    $16, 4($2)
sw    $16, 0($2)
sw    $15, 4($2)
```

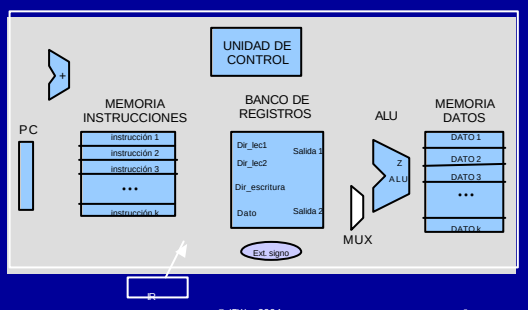
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

ALUOP[0:3] <= InstReg[9:11] & MASK

2. Estructura básica del Procesador

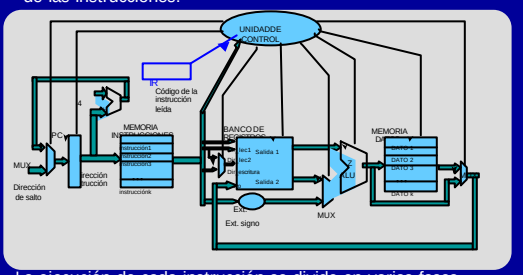
- Estructura de un Procesador
  - Forma en que se organizan e interconectan los diferentes elementos que lo forman (UAL, UC, BR)
- La estructura básica de un Procesador se construye a partir de los circuitos digitales ya estudiados
  - Unidad Aritmético-Lógica
  - Banco de registros
  - Memoria
  - Contadores
  - Codificadores/Decodificadores
  - Multiplexores/Demultiplexores
  - Dispositivos lógicos programables
- Ejemplo
  - Estructura del MIPS R2000 (Procesador RISC)

Estructura básica del MIPS R2000 (I)



Estructura básica del MIPS R2000 (II)

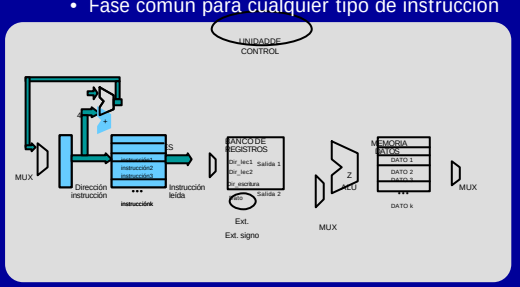
- Interconexión de los diferentes componentes para la ejecución de las instrucciones:



La ejecución de cada instrucción se divide en varias fases

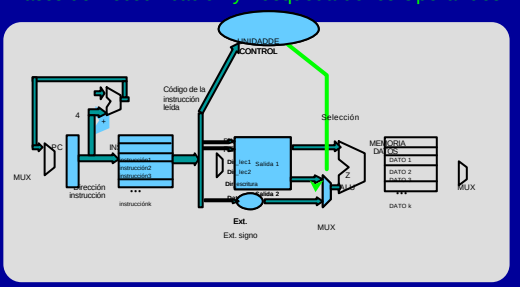
Fases de Ejecución en el MIPS R2000

- Fase de Búsqueda de la Instrucción
  - Fase común para cualquier tipo de instrucción



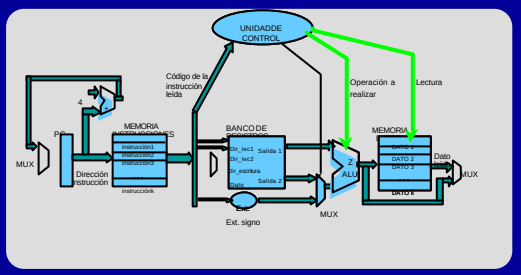
Fases de Ejecución en el MIPS R2000

- Fases de Decodificación y Búsqueda de los Operandos



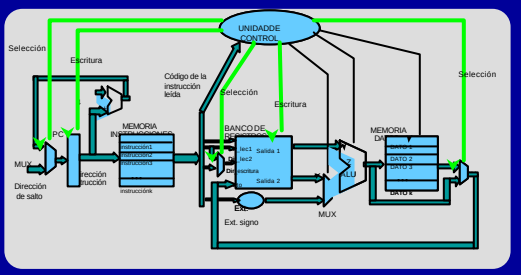
Fases de Ejecución en el MIPS R2000

- Fases de Ejecución y Acceso a Memoria



Fases de Ejecución en el MIPS R2000

- Fase de Almacenamiento



3. Arquitectura del Juego de Instrucciones.

- 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:
  - Conjunto de instrucciones
  - Conjunto de registros
  - Tipos de datos
  - Modos de direccionamiento
  - Espacio lógico direccionable (organización de la memoria)

3.1. Representación y tipos de datos

- Tipos de datos soportados por el MIPS R2000

Tipo de datos	Tamaño de palabra	Datos representados
Ascii	8 bits	Caracteres
Byte	8 bits	Números enteros
Half	16 bits	
Word	32 bits	
Float	32 bits	Números reales de simple precisión
Double	64 bits	Números reales de doble precisión

- Half y Word. El procesador dispone de instrucciones para el manejo de tipos de datos Half y Word sin signo y con signo en notación complemento a dos.
- Float. Representa números reales de simple precisión según el estandar IEEE-754

3.2. Banco de Registros (BR) (I)

- Características del banco de registros del MIPS R2000
  - Banco de registros de **Enteros**, constituido por:
    - 32 Registros de 32 bits de propósito general para operaciones con enteros. Se identifican por el carácter especial **$** seguido de un número de 0 a 31.
      - Ejemplo: add $2, $3, $4
    - 2 Registros especiales de 32 bits HI y LO.
      - Almacenan los resultados de multiplicaciones y divisiones.
      - Se utilizan para operaciones de transferencia de datos.

3.2. Banco de Registros (BR) (II)

- Banco de registros de **Reales**, que puede utilizarse como:
  - 32 Registros de 32 bits de propósito general para operaciones en coma flotante de simple precisión según el formato IEEE 754. Se identifican por $f0 a $f31
  - 16 registros de 64 bits para operaciones en coma flotante de doble precisión
- El registro $0 tiene permanentemente el valor 0

3.3. Organización de la Memoria (I)

- Memoria. Puede verse como un vector (array) unidimensional.
- Dirección de memoria. Índice dentro del vector.
- Memoria direccionada byte a byte. Los índices apuntan a cada byte de la memoria.

- En el MIPS R2000 la memoria se organiza en palabras de 4 bytes.
  - Cada palabra ocupa 4 posiciones o direcciones de memoria
  - Direcciones de palabras en memoria: 0 - 4 - 8 - 12 - .... etc.
- Es posible acceder a la memoria por:
  - Bytes: cualquier dirección 0 - 1 - 2 - 3 - .... etc.
  - Half: sólo direcciones pares 0 - 2 - 4 - 6 - .... etc.
  - Words: sólo direcciones múltiplos de cuatro: 0 - 4 - 8 - 12 - .... etc.

Organización de la Memoria (II)

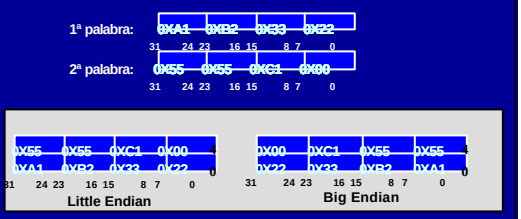
- Espacio direccionable en el MIPS R2000
  - 4 Gbytes = 2 Ghalf = 1 Gword
    - 2^{32} bytes con direcciones que van desde 0 a 2^{32} - 1
    - 2^{30} palabras (4 bytes) con direcciones: 0, 4, 8, ..., 2^{32} - 4
- Memoria accesible por el usuario se encuentra en el rango [0x00400000, 0x7FFFFFFF]

Organización de la Memoria (III)

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

Organización de la Memoria (IV)

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



3.4. Juego de Instrucciones (I)

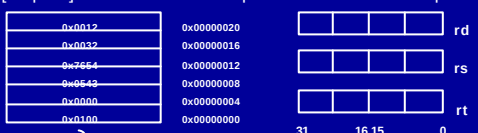
- Instrucciones
  - Lenguaje del Computador
  - El conjunto de instrucciones del MIPS R2000 que se estudiará es similar al utilizado en otras arquitecturas como Silicon Graphics, Sony , Nintendo ...
- Cada instrucción en lenguaje ensamblador tiene un formato según el cual se codificará finalmente la instrucción en lenguaje máquina (0's y 1's)

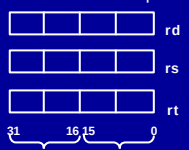
3.4. Juego de Instrucciones (II)

- 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

Juego de instrucciones (III)

- 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





rs = 0x00000004 } Desp+rs = 0x00000008
desp = 4 } M[desp+rs] = 0x0543

Juego de instrucciones (IV)

- Instrucciones Aritméticas

Sintaxis	Formato	Descripción
add rd, rs, rt	R	rd ← rs + rt
addi rt, rs, inm	I	rt ← rs + inm
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 HI y los 32 bits de mayor peso en LO
div rs, rt	R	Divide rs entre rt dejando el cociente en el registro LO y el resto en el registro HI

- Ejercicio:
  - Almacénese en los registros 8, 9 y 10 los valores 0x65, 0x65 y 0x54
  - Súmese el contenido de los registros 8, 9 y 10 almacenando el resultado en el registro 2

Juego de instrucciones (V)

- 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 (VI)

- Ejercicio:
  - La información almacenada en el registro $8 representa
    - Parte alta: color de un determinado pixel de la pantalla
    - Parte baja: brillo del mismo pixel
  - Se desea:
    - Obtener en el registro $2 el brillo de dicho pixel
    - Obtener en el registro $3 el color del mismo
- Solución:
  - andi $2, $8, 0x00001111
  - srl $3, $8, 16

Juego de instrucciones (VII)

- 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 (VIII)

- 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

- Ejercicio:
  - Multiplíquese los valores 0x78 y 0x41
    - Obteniendo la parte baja del resultado en el registro $2 y la parte alta en el registro $3.
    - Obteniendo el resultado a partir de la posición de memoria 0x10001000

Juego de instrucciones (IX)

- 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

Juego de instrucciones (X)

- 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

- Ejercicio:
  - Incrementar el contenido del registro $3 en tres unidades en el caso de que el registro $3 sea menor que el registro $4.

Juego de instrucciones (XI)

- 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+4 PC← etiqueta
jr rs	J	PC ← rs, salta a ladirección contenida en el registro rs

3.5. Formato de las Instrucciones (I)

- 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:
  - Formato **R** o de tipo registro
  - Formato **I** o de tipo inmediato
  - Formato **J** o de salto incondicional
- Cada uno de los componentes que utiliza la instrucción se especifica mediante una serie de bits denominada “campo” .

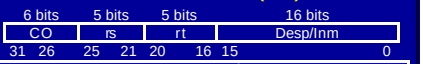
Formato de la Instrucciones (II)

- Formato tipo R

Campos	Nº de bits	Descripción
CO	6	Código de operación de la instrucción, impuesto por el diseñador, indica el tipo de instrucción.
rs	5	Codificación binaria del primer registro fuente.
rt	5	Codificación binaria del segundo registro fuente.
rd	5	Codificación binaria del registro destino que se utilice.
Numdesp	5	Codificación del número de desplazamientos. Se utiliza en operaciones de desplazamiento.
Función	6	Indica el tipo de operación aritmética o lógica.

Formato de la Instrucciones (III)

- Formato tipo I

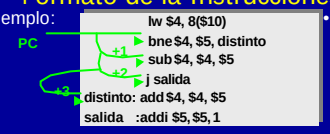


Campos	Nº de bits	Descripción
CO	6	Código de operación de la instrucción, impuesto por el diseñador, indica la operación que realiza la instrucción.
rs	5	Codificación binaria del primer registro fuente que se utilice
rt	5	Codificación binaria del segundo registro.
Desp/Inm	16	Dato de 16 bits codificado en complemento a 2.

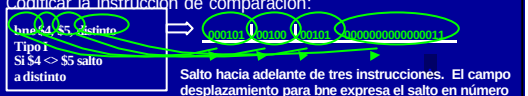
- El campo desplazamiento indica:
  - Desplazamiento expresado en número de palabras para las instrucciones BEQ y BNE
  - Desplazamiento en número de bytes para las instrucciones de Carga y Almacenamiento

Formato de la Instrucciones (IV)

- Ejemplo:



- En el simulador del MIPS, utilizado en prácticas, el registro PC contiene la dirección de memoria de la instrucción en ejecución
- Cada instrucción (32 bits) 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.
- Codificar la instrucción de comparación:

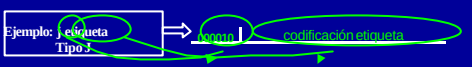


Salto hacia adelante de tres instrucciones. El campo desplazamiento para bne expresa el salto en número de palabras de memoria (instrucciones) a saltar

Formato de la Instrucciones (V)

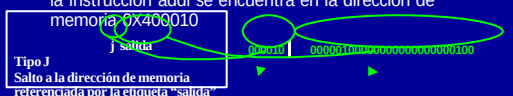
- Formato tipo J

Campos	Nº de bits	Descripción
CO	6	Código de operación de la instrucción, impuesto por el diseñador, indica la operación que realiza la instrucción.
Destino	26	Destino del salto a ejecutar.



Formato de la Instrucciones (VI)

- Ejemplo: (cont. ejemplo pág. 32)
  - Codificar la instrucción de salto j teniendo en cuenta que la instrucción addi se encuentra en la dirección de memoria 0X400010



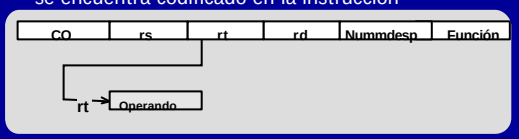
- Codificación de la etiqueta:
  - Sólo 26 bits de los 32 del Contador de Programa (PC) son modificados por j permaneciendo los otros 6 sin cambios
  - Salida = 0X400010 = 0000 0000 0100 0000 0000 0000 0001 0000 = dirección de salto
  - La dirección de salto (32 bits) debe codificarse en el campo destino de 26 bits 0000 0000 0100 0000 0000 0000 0001 0000
  - Se prescinde de los dos bits de menor peso y de los 4 bits de mayor peso por ser siempre ceros :

3.6. Modos de Direccionamiento (I)

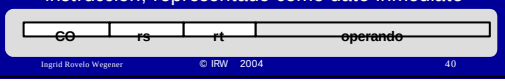
- Los modos de direccionamiento permiten localizar un operando que puede estar ubicado en:
  - En la propia instrucción (Direccionamiento inmediato)
  - En un registro (Direccionamiento a registro)
  - En la memoria (Direccionamiento relativo)
- El objetivo de los modos de direccionamiento es especificar el lugar donde se encuentra el operando.
- *La dirección efectiva* es la dirección física donde se encuentra un dato en memoria.
  - Para acceder a un dato determinado en memoria la instrucción debe proporcionar su dirección efectiva.

Modos de Direccionamiento (II)

- Direccionamiento a registro
  - El operando está en un registro cuyo identificador se encuentra codificado en la instrucción



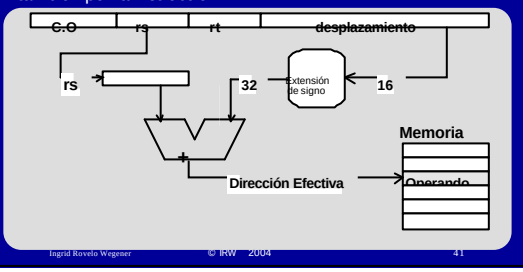
- Direccionamiento inmediato
  - El operando se encuentra codificado en la propia instrucción, representado como dato inmediato



Modos de Direccionamiento (III)

Direccionamiento relativo.

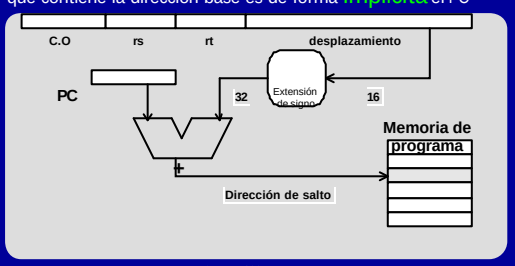
- La dirección efectiva del operando se obtiene sumando un desplazamiento que se encuentra en la instrucción con una dirección base que se encuentra en un registro especificado también por la instrucción



Modos de Direccionamiento (IV)

Direccionamiento relativo al contador de programa

- Es un caso particular del direccionamiento relativo donde el registro que contiene la dirección base es de forma implícita el PC



Modos de Direccionamiento (IV)

- Ejercicio.
  - Dadas las siguientes instrucciones

```
lb $19, 0($20)
beq $0, $19, fin     # La etiqueta fin hace referencia
                     # a la dirección de tres palabras abajo
addi $20, $20, 1
```

- Indíquese el modo de direccionamiento utilizado en cada una de las instrucciones

4. Codificación en Lenguaje Máquina (I)

- El programa ensamblador es el encargado de codificar en código máquina las instrucciones de un programa escrito en ensamblador
- La codificación de una instrucción depende del tipo de formato (R, I o J) al que pertenece

Codificación en Lenguaje Máquina (II)

- Codificación de las instrucciones tipo R

Instrucción	CO	rs	rt	rd	Num desp	Función
add	0	*rs*	*rt*	*rd*	0	0x20
and	0	*rs*	*rt*	*rd*	0	0x24
div	0	*rs*	*rt*	0	0	0x1a
jr	0	*rs*	0	0	0	8
mfhi	0	0	0	*rd*	0	0x10
mflo	0	0	0	*rd*	0	0x12
mthi	0	*rs*	0	0	0	0x11
mtlo	0	*rs*	0	0	0	0x13
mult	0	*rs*	*rt*	0	0	0x18
nor	0	*rs*	*rt*	*rd*	0	0x27
or	0	*rs*	*rt*	*rd*	0	0x25
slt	0	*rs*	*rt*	*rd*	0	0x2a
sub	0	*rs*	*rt*	*rd*	0	0x22
xor	0	*rs*	*rt*	*rd*	0	0x26
sll	0	*rs*	*rt*	*rd*	desp	0x00
srl	0	*rs*	*rt*	*rd*	desp	0x02
	6	*5*	*5*	*5*	5	6

Codificación en Lenguaje Máquina (III)

- Codificación de las instrucciones tipo I

Instrucción	CO	rs	rt	desp/Inm
addi	8	*rs*	*rt*	Inm
andi	0xc	*rs*	*rt*	Inm
beq	4	*rs*	*rt*	Desplazamiento en palabras
bne	5	*rs*	*rt*	Desplazamiento en palabras
lw	0x23	*rs*	*rt*	Desplazamiento en bytes
lui	0xf	0	*rt*	Inm
ori	0xd	*rs*	*rt*	Inm
sw	0x2b	*rs*	*rt*	Desplazamiento en bytes
xori	0xe	*rs*	*rt*	Inm
	6	*5*	*5*	16

Codificación en Lenguaje Máquina (IV)

- Codificación de las instrucciones tipo J

Instrucción	C.O.	Dirección
j	2	destino del salto
jal	3	destino del salto
	6	26

- Ejercicio:
  - Codifíquense las siguientes instrucciones
    - add $2, $10, $9
    - mult $10, $14
    - addi $20, $20, 1
    - lw $2, 0($8)
  - Codifíquense las instrucciones del programa ejemplo de la página 36