# CONJUNTO DE INSTRUCCIONES (SET DE INSTRUCCIONES): CLASIFICACION

## Instrucciones Tipo-R

### (Definición de los campos)

| opcode | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| | | | | | |

**opcode**: Código de operación de la familia de operaciones

**funct**: Código de la operación de la operación particular

**rs**: referencia al primer operando fuente (register source 1)

**rt**: referencia al segundo operando fuente (register source 2)

**rd**: referencia al registro destino. Referencia al registro donde se almacena el resultado

## CONJUNTO DE INSTRUCCIONES (SET DE INSTRUCCIONES): CLASIFICACION

| opcode | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| | | | | | |

**shamt**: Usado sólo en operaciones de desplazamiento. Cantidad de desplazamiento (*shift amount*).

**Formato de las instrucciones:** tamaño fijo, variable o híbrido.

# CONJUNTO DE INSTRUCCIONES (SET DE INSTRUCCIONES): CLASIFICACION

| opcode | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| add | $s2 | $s3 | $s1 | 0 | addition (with overflow) |
| 000000 | rs | rt | rd | 0 | 0x20 |
| 000000 | 18 | 19 | 17 | 0 | 100000 |
| 000000 | $18 | $19 | $17 | 0 | 32 |

add rd, rs, rt

add $s1, $s2, $s3 ⟶ $s1 = $s2 + $s3

add $17, $18, $19 ⟶ $17 = $18 + $19

Pattersson-Hennessy, page 64, A-49, A-51

# CONJUNTO DE INSTRUCCIONES (SET DE INSTRUCCIONES): CLASIFICACION

| opcode | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| addu | $s2 | $s3 | $s1 | 0 | addition (without overflow) |
| 000000 | rs | rt | rd | 0 | 0x21 |
| 000000 | 18 | 19 | 17 | 0 | 100001 |
| 000000 | $18 | $19 | $17 | 0 | 33 |

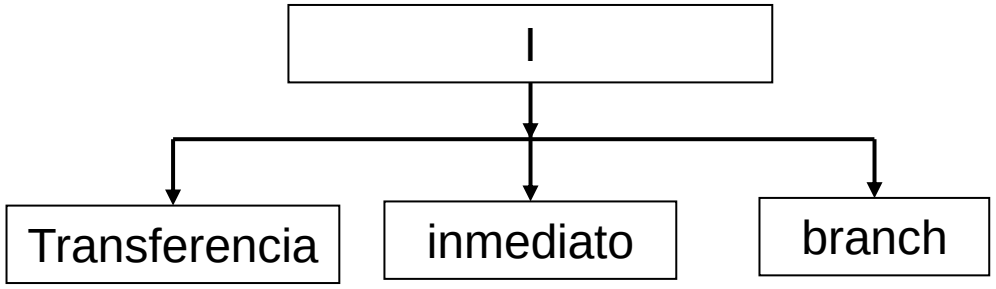
addu rd, rs, rt

addu $s1, $s2, $s3 ⟶ $s1 = $s2 + $s3

addu $17, $18, $19 ⟶ $17 = $18 + $19

Pattersson-Hennessy, page 64, A-49, A-51

**Instrucciones MIPS:** (Instrucciones Tipo-I: immediate-type)




Utilizado por las instrucciones de:

a).- Transferencia (load, store, swap, move)

b).- De tipo inmediato

c).- Branch [Saltos condicionales (beq, bnq)]

# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)

## Instrucciones de transferencia

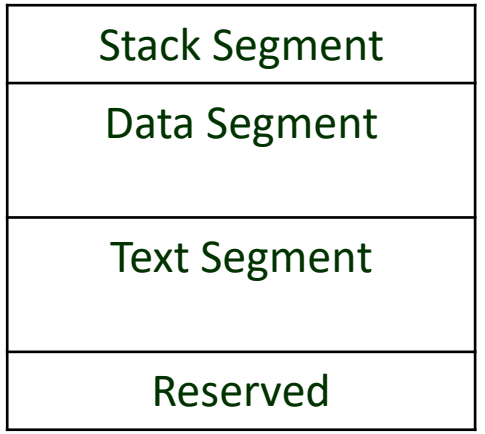
Conocimientos previos para las operaciones de transferencia

La memoria cache se organiza en tres tipos de segmentos llamados:

segmento de código, CS (memoria de programa, text)

segmento de datos, DS (memoria de datos)

segmento de pila, SS




Organización de la memoria interna del procesador

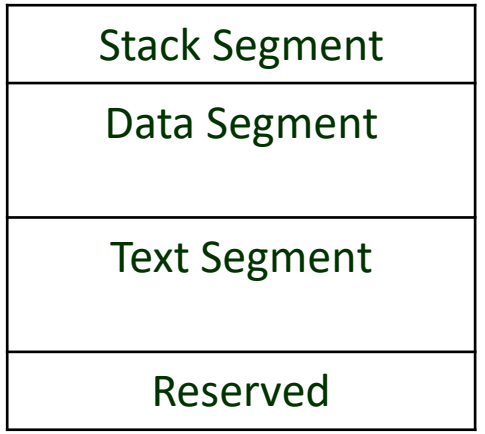
# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)

## Instrucciones de transferencia

Conocimientos previos para las operaciones de transferencia

La memoria cache se organiza en tres tipos de segmentos llamados:

La memoria que se presenta en el datapath de la arquitectura MIPS-32, es sustituida, en un diseño final por la memoria cache




Organización de la memoria interna del procesador

Patterson-Hennessy: Page 383, 384, 389

# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)

## Instrucciones de transferencia

Conocimientos previos para las operaciones de transferencia

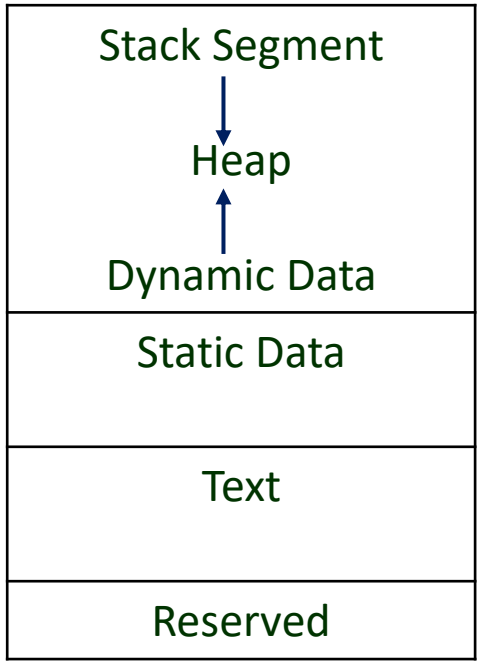
La memoria cache se organiza en tres tipos de segmentos llamados:

segmento de código, CS (memoria de programa, text)

segmento de datos, DS (memoria de datos)

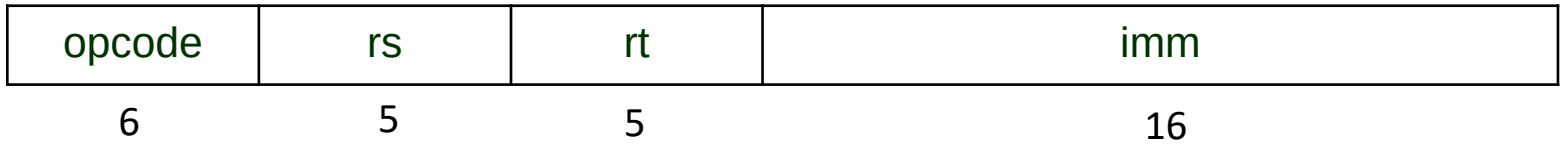
segmento de pila, SS

Organización de la memoria interna del procesador



# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)

## Instrucciones de transferencia




rs: registro base

rt: registro destino

imm = nombre del campo de 16 bit para instrucciones tipo I (offset)

# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)

## Instrucciones de transferencia

| opcode | rs | rt | imm |
|---|---|---|---|
| 6 | 5 | 5 | 16 |

rs: registro base: contiene la dirección del inicio del segmento de datos

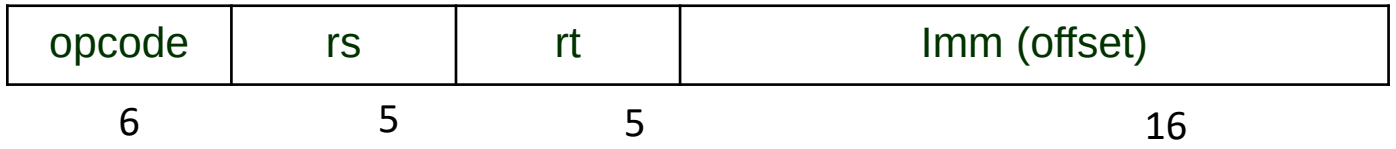
rt: registro destino

imm = nombre del campo de 16 bit para instrucciones tipo I

## CONJUNTO DE INSTRUCCIONES (SET DE INSTRUCCIONES): CLASIFICACION

# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)

**Instrucciones de transferencia: load**




rs: hace referencia al registro base

rt: registro destino: registro donde se depositará el dato

imm = offset

# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)

## Instrucciones de transferencia: load

| 0x23 | rs | rt | Imm (offset) |
|---|---|---|---|
| 6 | 5 | 5 | 16 |

lw rt, address

Carga la cantidad, que es una palabra de 32 bit, en la dirección contenida en el registro rt

lw $rt, imm($rs) $rt, memory[$rs + imm]

lw $s1, 20($s2) $s1, memory[$s2 + 20]

## CONJUNTO DE INSTRUCCIONES (SET DE INSTRUCCIONES): CLASIFICACION

# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)

### Instrucciones de transferencia: lui

| 0xf | rs | rt | Imm (offset) |
|---|---|---|---|
| 6 | 5 | 5 | 16 |

lui rt, imm

Carga la halfword inferior del campo inmediato imm en la halfword superior del registro rt. Los bits inferiores del registro son escritos con ‘0’s.

## CONJUNTO DE INSTRUCCIONES (SET DE INSTRUCCIONES): CLASIFICACION

# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)

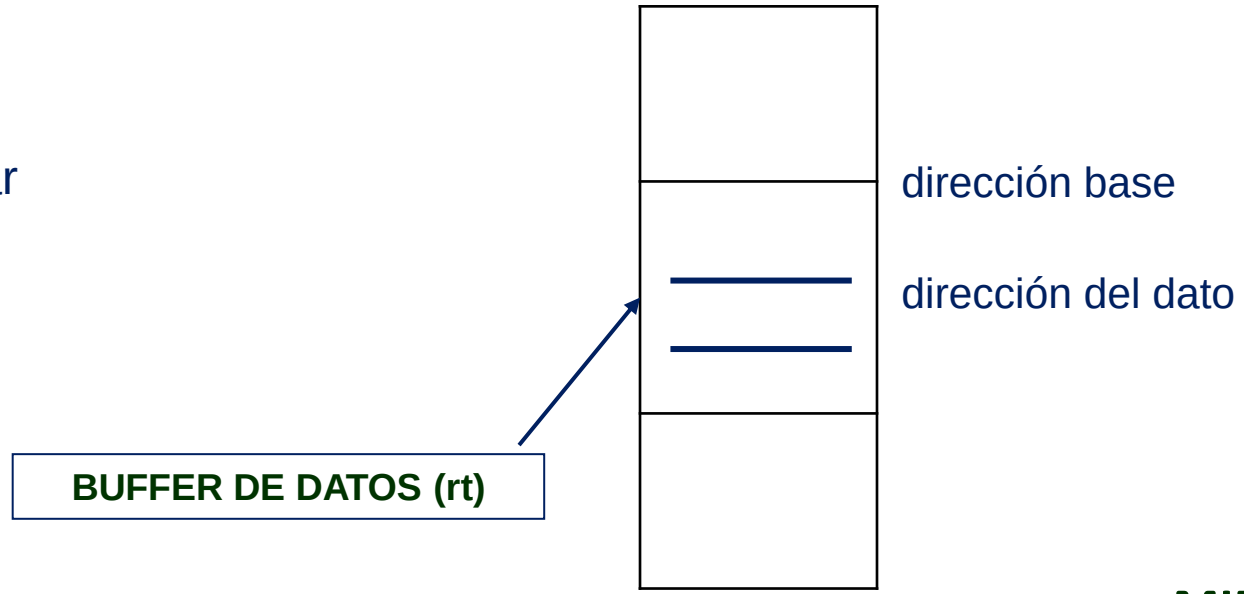
**Instrucciones de transferencia: store**

| opcode | rs | rt | Imm (offset) |
|---|---|---|---|
| 6 | 5 | 5 | 16 |

rs: registro base

rt: registro con el dato a guardar

El campo de 16, para instrucciones de transferencia, es el offset



# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)

## Instrucciones de transferencia: store

| opcode | rs | rt | Imm (offset) |
|---|---|---|---|
| 6 | 5 | 5 | 16 |

sw rt, address

salva la cantidad, que es una palabra de 32 bit, contenida en el registro rt en la posición address

sw $rt, imm($rs) $rt, memory[$rs + imm]

sw $s1, 20($s2) $s1, memory[$s2 + 20]

## CONJUNTO DE INSTRUCCIONES (SET DE INSTRUCCIONES): CLASIFICACION

# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)

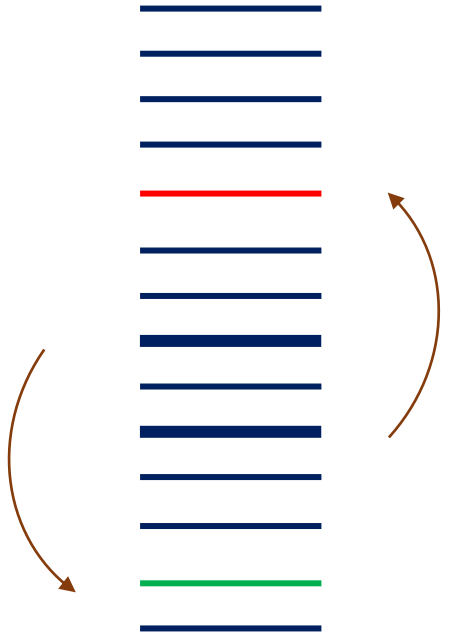
**Instrucciones Branch**

| opcode | rs | rt | Imm (address) |
|---|---|---|---|
| 6 | 5 | 5 | 16 |

rs: es uno de los registros que se ha de comparar

rt: es otro de los registros que se ha de comparar

Branch: Son saltos condicionales



# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)

**Instrucciones para operaciones Inmediatas**

| opcode | rs | rt | Imm (constant) |
|---|---|---|---|
| 6 | 5 | 5 | 16 |

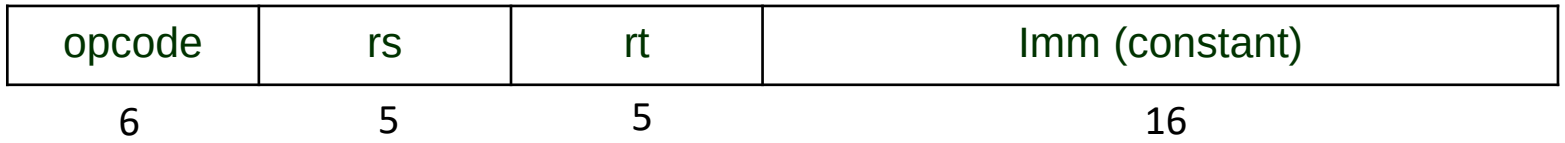
rs: registro fuente

rt: registro destino

**imm = constant:** Para el caso de una instrucción de suma inmediata, la constante a sumar está limitada a valores de 16 bits

La constante es un número signado

# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)




rs: registro base

En el modelo MIPS al registro base también se le llama **registro índice**

Al valor del registro base deberá sumarse un offset para obtener el valor de la dirección donde se ubica el dato a ser cargado en el registro **rt**

# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)

| opcode | rs | rt | Imm (constant) |
|---|---|---|---|
| 6 | 5 | 5 | 16 |

rs: registro base

En el procesador ARM DDI 0201D

El registro base conserva el significado de la posición de inicio del segmento de datos

El valor de offset, en la dirección (address) es definido como **registro índice**

3CM12: 18 de octubre de 2021

3CM14: 18 de octubre de 2021

3CM11: 19 de octubre de 2021

ARM: Procesador

ARM946E-S

ARM DDI 0201D

Glossary-5

Pagina 213

# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)

| opcode | rs | rt | Imm (address) |
|---|---|---|---|
| 6 | 5 | 5 | 16 |

**rs: registro base**

(En Instrucciones de saltos condicionales)

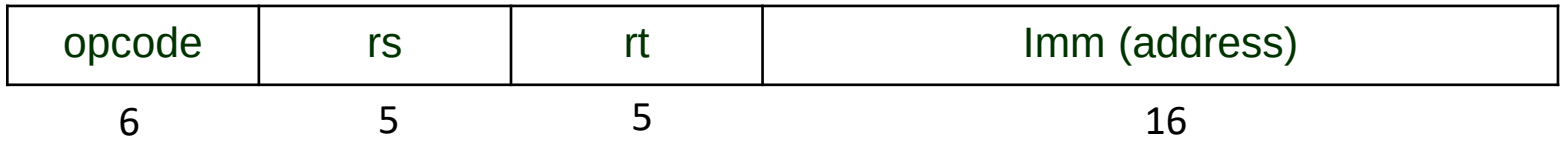
Hacer un salto condicional, en nivel alto es con una condición “if” – “then”-”else”

En lenguaje ensamblador es con “beq”, “branch if equal”

ó “bne”, “branch if not equal”

## CONJUNTO DE INSTRUCCIONES (SET DE INSTRUCCIONES): CLASIFICACION

# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)




(En Instrucciones de saltos condicionales)

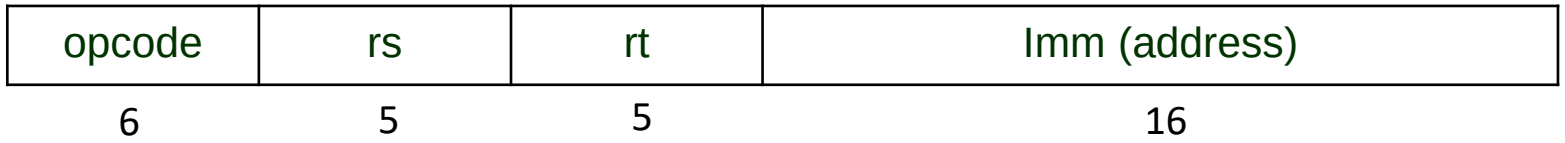
**rs: registro base**

Hacer un salto condicional, en nivel alto es con una condición “if” – “then”-”else”

Para la instrucción “beq”, “branch if equal, “rs” y “rt” son registros que se han de comparar

El valor en el campo “constant or address” es la dirección a donde ha de saltar el contador de programa si los registros son iguales

# Instrucciones MIPS: (Instrucciones Tipo-I: immediate-type)




(En Instrucciones de saltos condicionales)

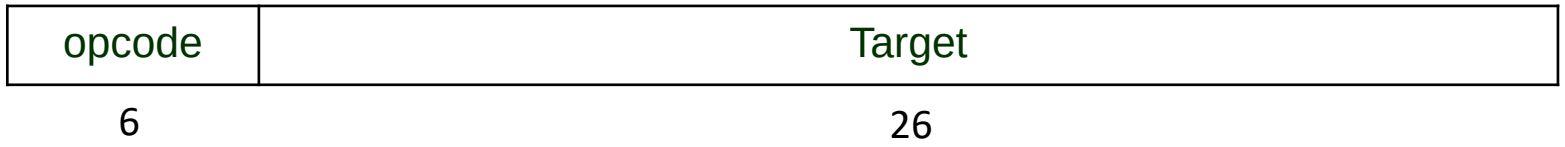
**rs: registro base**

Hacer un salto condicional, en nivel alto es con una condición “if” – “then”-”else”

Para la instrucción “bne”, “branch if not equal, “rs” y “rt” son registros que se han de comparar

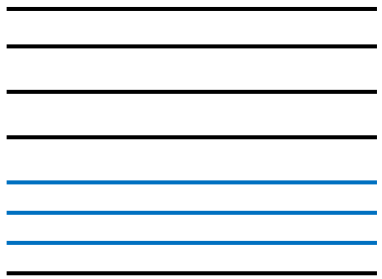
El valor en el campo “constant or address” es la dirección a donde ha de saltar el contador de programa si los registros **no** son iguales

# Instrucciones MIPS: (Instrucciones Tipo-J: Jump-type)




**Target** : Definidas para ser usadas como instrucciones de salto y/o e Instrucciones de bifurcación

Es una constante que identifica una posición de la lista de instrucciones de un programa.



## CONJUNTO DE INSTRUCCIONES (SET DE INSTRUCCIONES): CLASIFICACION

# Instrucciones MIPS: (Instrucciones Tipo-J: Jump-type)

| opcode | Target |
|---|---|
| 6 | 26 |
| 2 | Target |

j target

Salto incondicional a la instrucción con la etiqueta “tarjet”

# Instrucciones MIPS: (Instrucciones Tipo-J: Jump-type)

| opcode | Target |
|---|---|
| 6 | 26 |

| 3 | Target |
|---|---|

jal target

Salto incondicional a la instrucción con la etiqueta “tarjet”. Guarda la dirección de la siguiente instrucción en el registro $ra

Primero carga en el registro $ra la dirección siguiente del programa que se está ejecutando. Después carga en el contador de programa la dirección “target”, la cual es la dirección a la cual debe saltar

# En lenguaje ensamblador MIPS: Archivo de registros

| mnemónico | Registro |
|---|---|
| $t0 | 8 |
| $t1 | 9 |
| $t2 | 10 |
| $t3 | 11 |
| $t4 | 12 |
| $t5 | 13 |
| $t6 | 14 |
| $t7 | 15 |

| mnemónico | Registro |
|---|---|
| $s0 | 16 |
| $s1 | 17 |
| $s2 | 18 |
| $s3 | 19 |
| $s4 | 20 |
| $s5 | 21 |
| $s6 | 22 |
| $s7 | 23 |

```
add  $t0,  $s1,  $s2;
```

# En lenguaje ensamblador MIPS: Archivo de registros

add $t0, $s1, $s2 $t0 = $s1 + $s2

| opcode | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| **000000** | **17** | **18** | **8** | **0** | **32** |
| arith | $s1 | $s2 | $t0 | | 100000 |
| arith | 17 | 18 | 8 | | add |
| | 1er OP fuente | 2do OP fuente | Reg. Destino | | |

En combinación, los campos “opcode” y “funct” le dicen al procesador MIPS que la operación a realizar es una suma

# En lenguaje ensamblador MIPS: Archivo de registros

add $t0, $s1, $s2; $t0 = $s1 + $s2

| opcode | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| **000000** | **10001** | **10010** | **01000** | 00000 | **100000** |
| arith | $s1 | $s2 | $t0 | | add |
| | 17 | 18 | 8 | | |
| | 1er OP fuente | 2do OP fuente | Reg. Destino | | |

En combinación, los campos “opcode” y “funct” le dicen al procesador MIPS que la operación a realizar es una suma

# En lenguaje ensamblador MIPS: Archivo de registros

Suponer que la variable h está asociada al registro $s2

Suponer que la dirección base del arreglo A está en $s3

Suponer que el siguiente enunciado está en lenguaje C

A[12] = h + A[8]

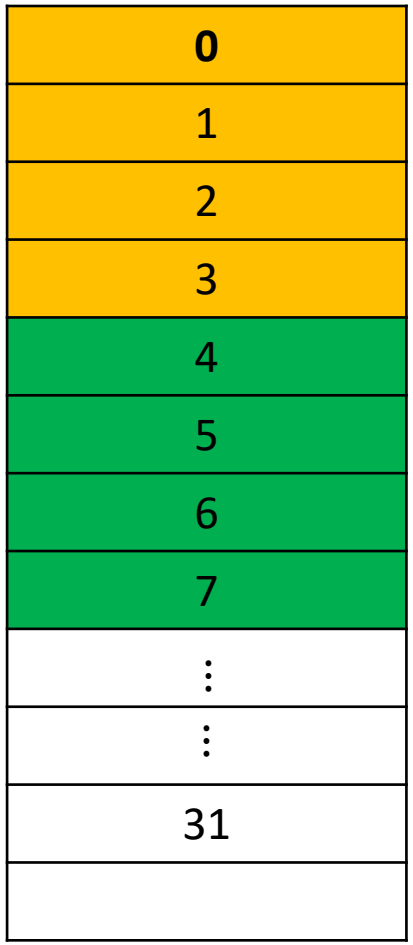
Cuál sería su correspondiente codificación en lenguaje ensamblador MIPS?

```
add  $t0,  $s1,  $s2
```

RISC MIPS

# CONJUNTO DE INSTRUCCIONES (SET DE INSTRUCCIONES): CLASIFICACION

## En lenguaje ensamblador MIPS: Archivo de registros




Patterson-Hennessy, página 150, 252

# CONJUNTO DE INSTRUCCIONES (SET DE INSTRUCCIONES): CLASIFICACION

## Formato de lectura de los byte de un registro

| | | |
|---|---|---|
| **0** | **BYTE 0** | **DATO 0** |
| 1 | BYTE 1 | |
| 2 | BYTE 2 | |
| 3 | BYTE 3 | |
| 4 | BYTE 4 | DATO 1 |
| 5 | BYTE 5 | |
| 6 | BYTE 6 | |
| 7 | BYTE 7 | |
| 8 | BYTE 8 | DATO 2 |
| 9 | | |
| 10 | | |
| 11 | BYTE 111 | |

# En lenguaje ensamblador MIPS: Archivo de registros

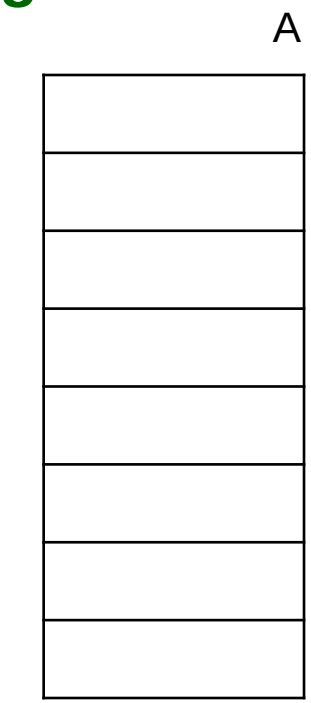
Suponer que la variable h está asociada al registro $s2

Suponer que la dirección base del arreglo A está en $s3

Suponer que el siguiente enunciado está en lenguaje C

A[12] = h + A[8]

Cuál sería su correspondiente codificación en lenguaje ensamblador MIPS?




Primero se carga, en un registro, el dato localizado en la posición

[4x8 = 32] + $s3 El resultado se carga en el registro $t0

offset Dirección base

# En lenguaje ensamblador MIPS: Archivo de registros

Procesadores load-store

La instrucción en ensamblador MIPS es

lw $t0, 32($s3) (Load Word)

A continuación se hace la suma en ensamblador MIPS

add $t0, $s2, $t0 Se lleva a cabo la suma de el dato cargado en $t0 con la variable h = $s2 y el resultado se guarda temporalmente de nuevo en $t0

A continuación se guarda el resultado en memoria con la instrucción “store word”

sw $t0, 48($s3) En la posición [4x12 = 48] + $s3 (Store Word)

## Instrucciones MIPS: (Instrucciones Tipo-J: Jump-type)

| opcode | Target |
|---|---|
| Definición de campos en una instrucción MIPS | |

```
Loop:   sll $t1,$s3,2          # Temp reg $t1 = 4 * i
        add $t1,$t1,$s6        # $t1 = address of save[i]
        lw $t0,0($t1)          # Temp reg $t0 = save[i]
        bne $t0,$s5, Exit      # go to Exit if save[i] ≠ k
        addi $s3,$s3,1         # i = i + 1
      j   Loop                 # go to Loop
Exit:
```

# Instrucciones MIPS: Registros de Propósito General

En esta lámina se introduce el concepto de “Registros de Propósito General” para el caso del modelo MIPS

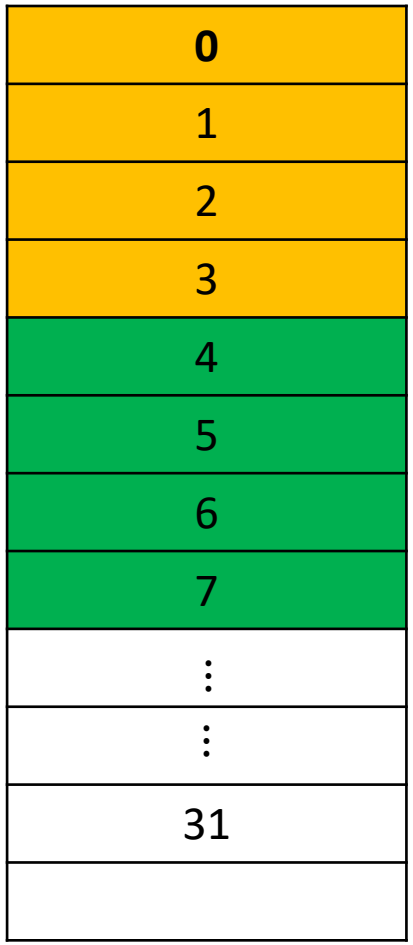
En otras láminas, más adelante, se abordará nuevamente el tema.

**Un archivo de registro es una colección de registros en los que se puede leer o escribir cualquier registro especificando el número del registro en el archivo. Además, obsérvese que todo se considera formando parte del procesador**

RISC MIPS

# CONJUNTO DE INSTRUCCIONES (SET DE INSTRUCCIONES): CLASIFICACION

## En lenguaje ensamblador MIPS: Archivo de registros




Patterson-Hennessy, página 150, 252

# Instrucciones MIPS: Registros de Propósito General

**La CPU MIPS contiene 32 registros de propósito general, numerados de 0 a 31**

**El registro 0 ($0) siempre contiene el valor 0 por defecto**

**Los registros 1 ($at) 26 ($k0) y 27 ($k1) son reservados para el ensamblador y el sistema operativo y no deben ser usados por el programador**

**Los registros del 4 al 7 (del $a0 al $a3) son usados para pasar cuatro argumentos a subrutinas. Otros argumentos de subrutinas son pasados al registro stack.**

**Los registros 2 ($v0) y 3 ($v1) son usados para regresar valores de funciones**