

Grupo ARCOS

**uc3m** | Universidad **Carlos III** de Madrid

# Tema 3 (III)

## Fundamentos de la programación en ensamblador

Estructura de Computadores  
Grado en Ingeniería Informática



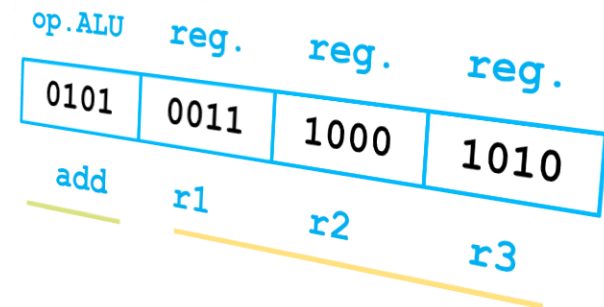
# Contenidos

- ▶ Fundamentos básicos de la programación en ensamblador
- ▶ Ensamblador del RISC-V 32, modelo de memoria y representación de datos
- ▶ Formato de las instrucciones y modos de direccionamiento
- ▶ Llamadas a procedimientos y uso de la pila

# Información de una instrucción

## ► Las instrucciones:

- Su tamaño se **ajusta** a **una** o **varias palabras**
- Están **divididas en campos**:
  - Operación a realizar
  - Operandos a utilizar
    - Puede haber operando implícitos

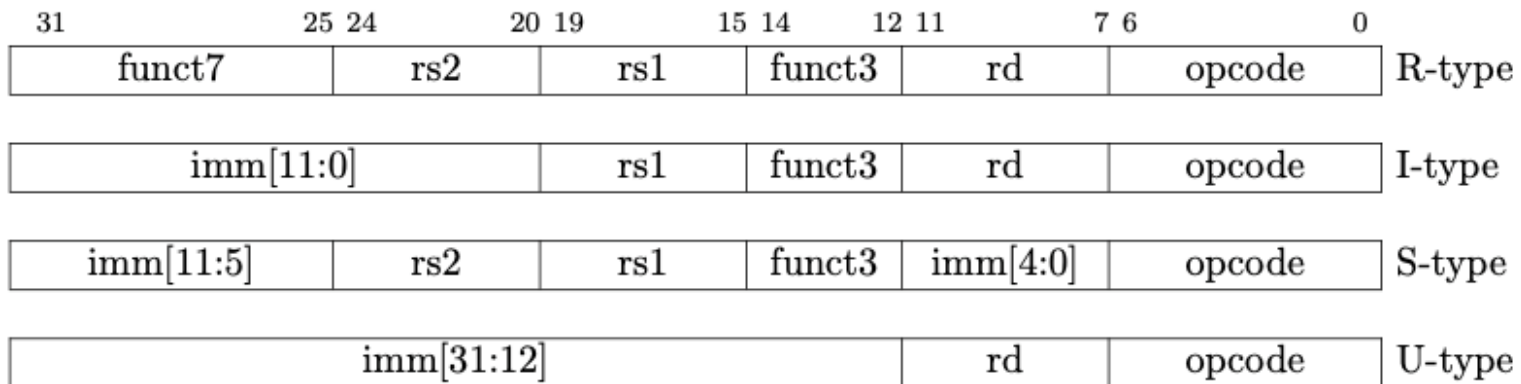


## ► El **formato de una instrucción**:

- Forma de representación de una instrucción compuesta de campos de números binarios:
  - El tamaño de los campos limita el número de valores a codificar

# Información de una instrucción

- ▶ Se utiliza unos pocos formatos:
  - ▶ Cada instrucción pertenece a un formato
  - ▶ Según el código de operación se conoce el formato asociado
- ▶ Ejemplo: formatos básicos en RISC-V



# Instrucciones y pseudoinstrucciones del RISC-V<sub>32</sub>

- ▶ Una instrucción en ensamblador se corresponde con una instrucción máquina
  - ▶ Ejemplo: `addi t1, t1, 2`
- ▶ Una pseudoinstrucción en ensamblador se corresponde con una o varias instrucciones de ensamblador
  - ▶ Ejemplo 1:
    - ▶ La instrucción: `mv reg2, reg1`
    - ▶ Equivale a: `add reg2, zero, reg1`
  - ▶ Ejemplo 2:
    - ▶ La instrucción: `li t1, 0x00800010`
    - ▶ No cabe en 32 bits, pero puede usarse como pseudoinstrucción
    - ▶ Es equivalente a:
      - `lui t1, 0x0080`
      - `ori t1, t1, 0x0010`

# Campos de una instrucción

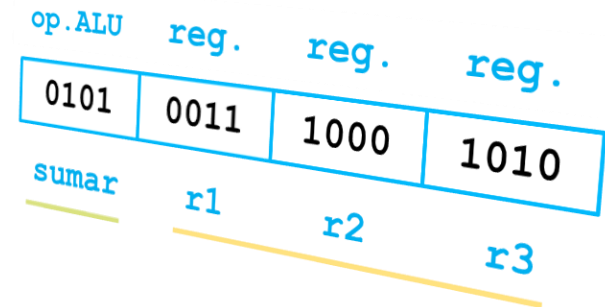
## ► En los campos se codifica:

### ► Operación a realizar (código Op.)

- Instrucción y formato de la misma

### ► Operandos a utilizar

- Ubicación de los operandos
- Ubicación del resultado
- Ubicación de la siguiente instrucción (*si op. salto*)
  - Implícito:  $PC \leftarrow PC + '4'$  (apuntar a la siguiente instrucción)
  - Explícito: j 0x01004 (modifica el PC)



# Ubicaciones posibles para los operandos

## 1. En la propia instrucción

li t0 0x123

## 2. En registros (CPU)

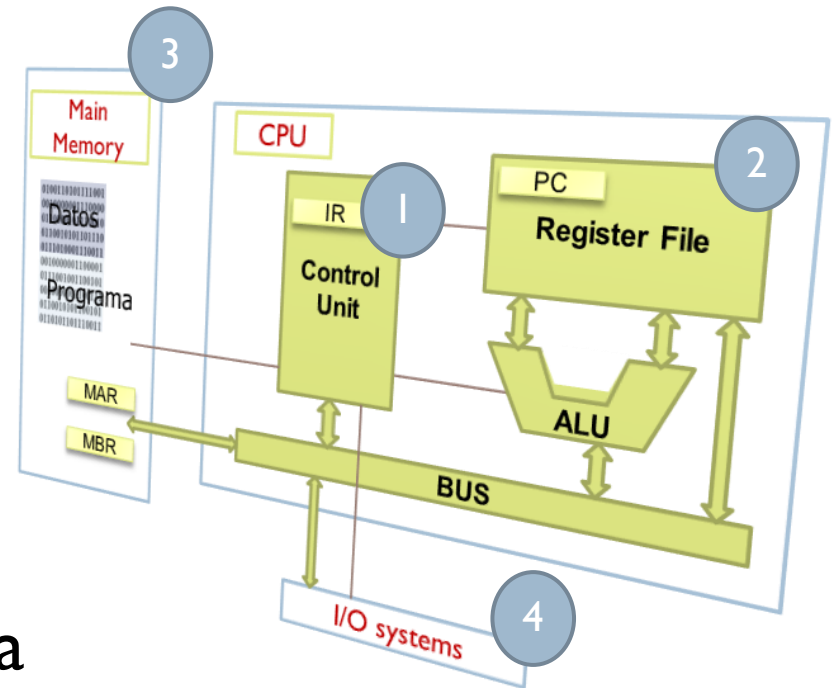
li t0 0x123

## 3. Memoria principal

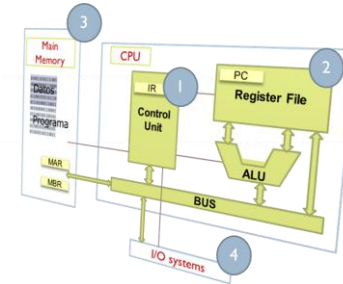
lw t0 address(x0)

## 4. Unidades de Entrada/Salida

in t0 0xFEB



# Formas de indicar la ubicación de operandos: modos de direccionamiento



## 1. En la propia instrucción

li t0 0x123

## 2. En registros (CPU)

li t0 0x123

## 3. Memoria principal

lw t0 **address(x0)**

- **num(registro)**: representa la dirección que se obtiene de sumar num con la dirección almacenada en el registro

## 4. Unidades de Entrada/Salida

in t0 0xFEB






# Contenidos

- ▶ Fundamentos básicos de la programación en ensamblador
- ▶ Ensamblador del RISC-V<sub>32</sub>, modelo de memoria y representación de datos
- ▶ Formato de las instrucciones y modos de direccionamiento
- ▶ Llamadas a procedimientos y uso de la pila

# Modos de direccionamiento

- ▶ El **modo de direccionamiento** es un procedimiento que permite determinar la **ubicación** de un operando, un resultado o una instrucción




- ▶ Implícito
- ▶ Inmediato
- ▶ Directo 
  - a registro
  - a memoria
- ▶ Indirecto 
  - a registro
  - a memoria
- ▶ Relativo 
  - a registro índice
  - a registro base
  - a PC
  - a Pila

# Modos de direccionamiento en RISC-V

- ▶ Inmediato value
- ▶ Directo
  - ▶ A memoria address
  - ▶ A registro xr
- ▶ Indirecto
  - ▶ A memoria
  - ▶ A registro (xr)
- ▶ Relativo a
  - registro offset(xr)
  - pila offset(sp)
  - PC beq ... label

# Modos de direccionamiento

- ▶ El **modo de direccionamiento** es un procedimiento que permite determinar la **ubicación** de un operando, un resultado o una instrucción

- ▶ **Implícito**
- ▶ **Inmediato**
- ▶ **Directo** 
  - a registro
  - a memoria
- ▶ **Indirecto** 
  - a registro
  - a memoria
- ▶ **Relativo** 
  - a registro índice
  - a registro base
  - a PC
  - a Pila

# Direccionamiento implícito

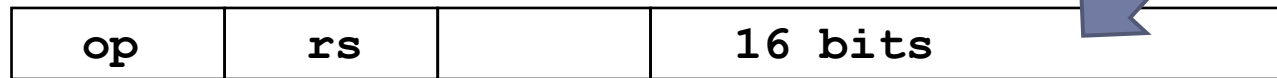
- ▶ El operando no está codificado en la instrucción, pero forma parte de esta
- ▶ Ejemplo: **auipc a0 0x12345**
  - ▶  $a0 = PC + (0x12345 \ll 12)$ .
  - ▶ a0 es un operando, PC es el otro (implícito)



- ▶ V/I (Ventajas/Inconvenientes)
  - ✓ Es rápido: no es necesario acceder a memoria.
  - ✗ Pero solo es posible en unos pocos casos.

# Direccionamiento inmediato

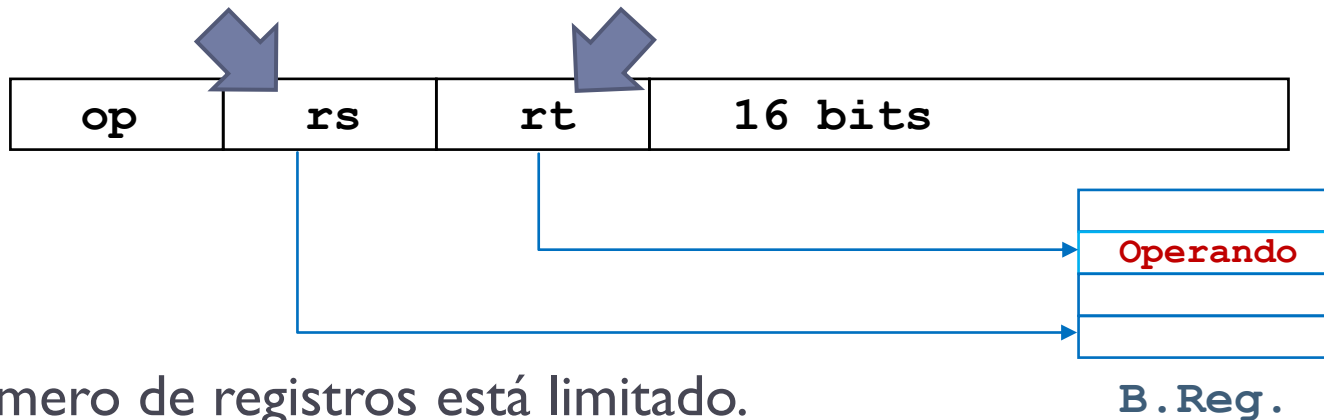
- ▶ El operando forma parte de la instrucción.
- ▶ Ejemplo: `li a0 0x4f5 l`
  - ▶ Carga en el registro a0 el valor inmediato `0x4f5 l`.
  - ▶ El valor `0x00004f5 l` está codificado en la propia instrucción.



- ▶ V/I
  - ✓ Es rápido: no es necesario acceder a memoria.
  - ✗ No siempre cabe el valor en una palabra:
    - ▶ No cabe en 32 bits, es equivalente a:
      - `lui t1, 0x87654`
      - `ori t1, t1, 0x321`

# Direccionamiento directo a registro (direccionamiento de registro)

- ▶ El operando se encuentra en el registro.
- ▶ Ejemplo: `mv a0 a1`
  - ▶ Copia en el registro `a0` el valor que hay en el registro `a1`.
  - ▶ El identificador de `a0` y `a1` está codificado en la instrucción.



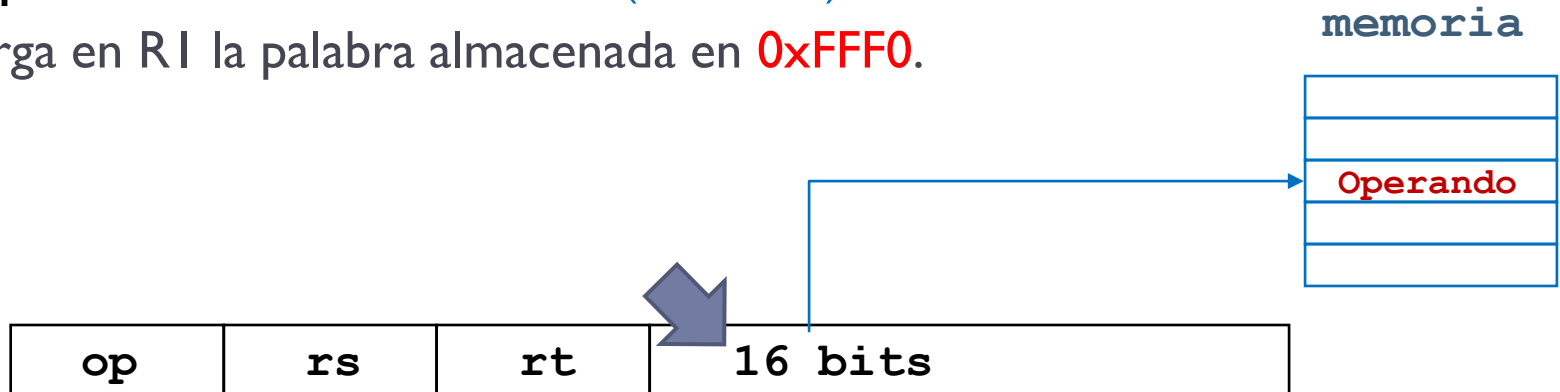
- ▶ V/I
  - ✗ El número de registros está limitado.
  - ✓ Acceso a registros es rápido
  - ✓ El número de registros es pequeño => pocos bits para su codificación, instrucciones más cortas

# Direccionamiento directo a memoria

- ▶ El operando se encuentra en memoria, y la dirección está codificada en la instrucción.

- ▶ Ejemplo: **LD RI #0xFFF0** (IEEE 694)

- ▶ Carga en RI la palabra almacenada en **0xFFF0**.



- ▶ V/I

- ✗ Acceso a memoria es más lento comparado con los registros
  - ✗ Direcciones largas => instrucciones más largas
  - ✓ Acceso a un gran espacio de direcciones (capacidad > B.R.)



# Modos de direccionamiento

- ▶ El **modo de direccionamiento** es un procedimiento que permite determinar la **ubicación** de un operando, un resultado o una instrucción

- ▶ Implícito
- ▶ Inmediato
- ▶ Directo
  - a registro
  - a memoria
- ▶ Indirecto
  - **a registro**
  - **a memoria**
- ▶ Relativo
  - a registro índice
  - a registro base
  - a PC
  - a Pila

# Direccionamiento directo vs. indirecto

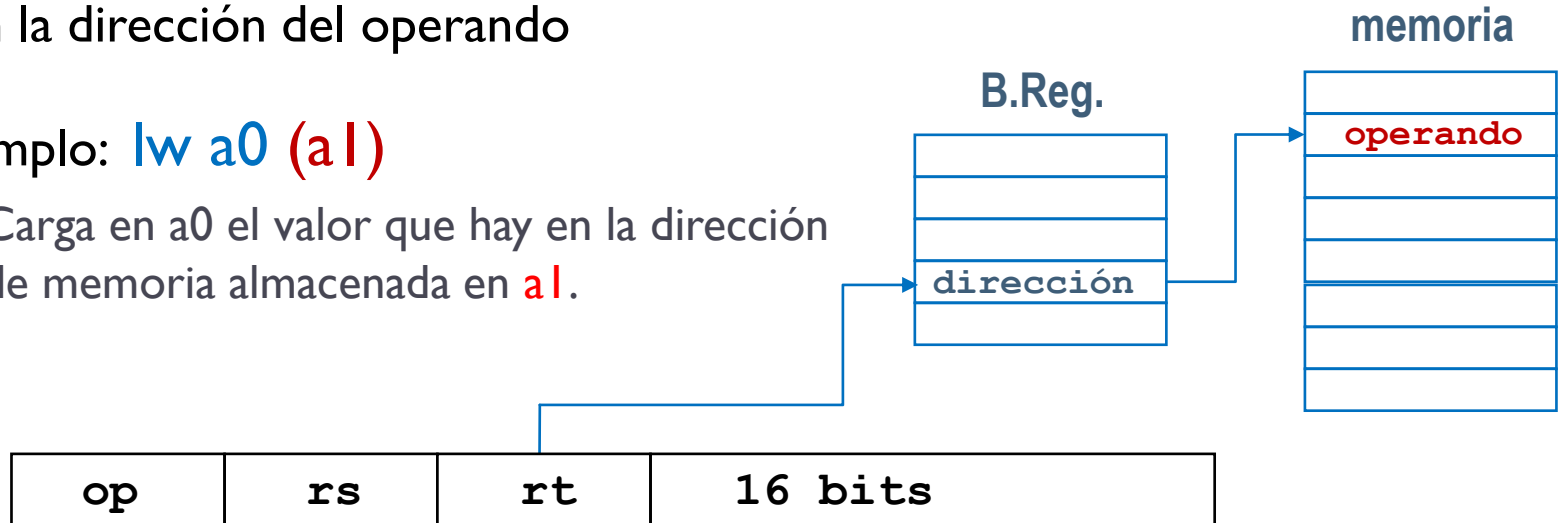
- ▶ En el direccionamiento directo se indica **dónde está el operando**:
  - ▶ En qué registro o en qué posición de memoria
- ▶ En el direccionamiento indirecto se indica **dónde está la dirección del operando**:
  - ▶ Hay que acceder a esa dirección en memoria
  - ▶ Se incorpora un nivel (o varios) de direccionamiento

# Direccionamiento indirecto de registro

- ▶ Se indica en la instrucción el registro con la dirección del operando

- ▶ Ejemplo: **lw a0 (a1)**

- ▶ Carga en a0 el valor que hay en la dirección de memoria almacenada en **a1**.



- ▶ V/I

- ✓ Amplio espacio de direcciones, instrucciones cortas
  - ▶ Pseudo-instrucción equivalente a `lw a0 0(a1)`

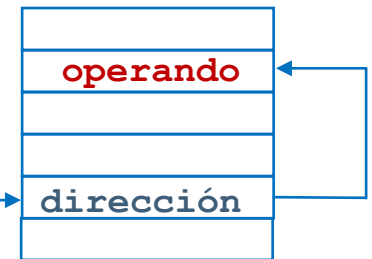
# Direccionamiento indirecto a memoria

- Se indica en la instrucción la dirección donde está la de la dirección del operando (no disponible en RISC-V)

- Ejemplo: **LD RI [DIR]** (IEEE 694)

- Carga en RI el valor que hay en la dirección de memoria que está almacenada en la dirección de memoria **DIR**.
- .

memoria






- V/I

- ✓ Amplio espacio de direcciones
- ✓ El direccionamiento puede ser anidado, multinivel o en cascada
  - Ejemplo: LD RI [[[.RI]]]
- ✗ Puede requerir varios accesos memoria
- ✗ instrucciones más lentas de ejecutar

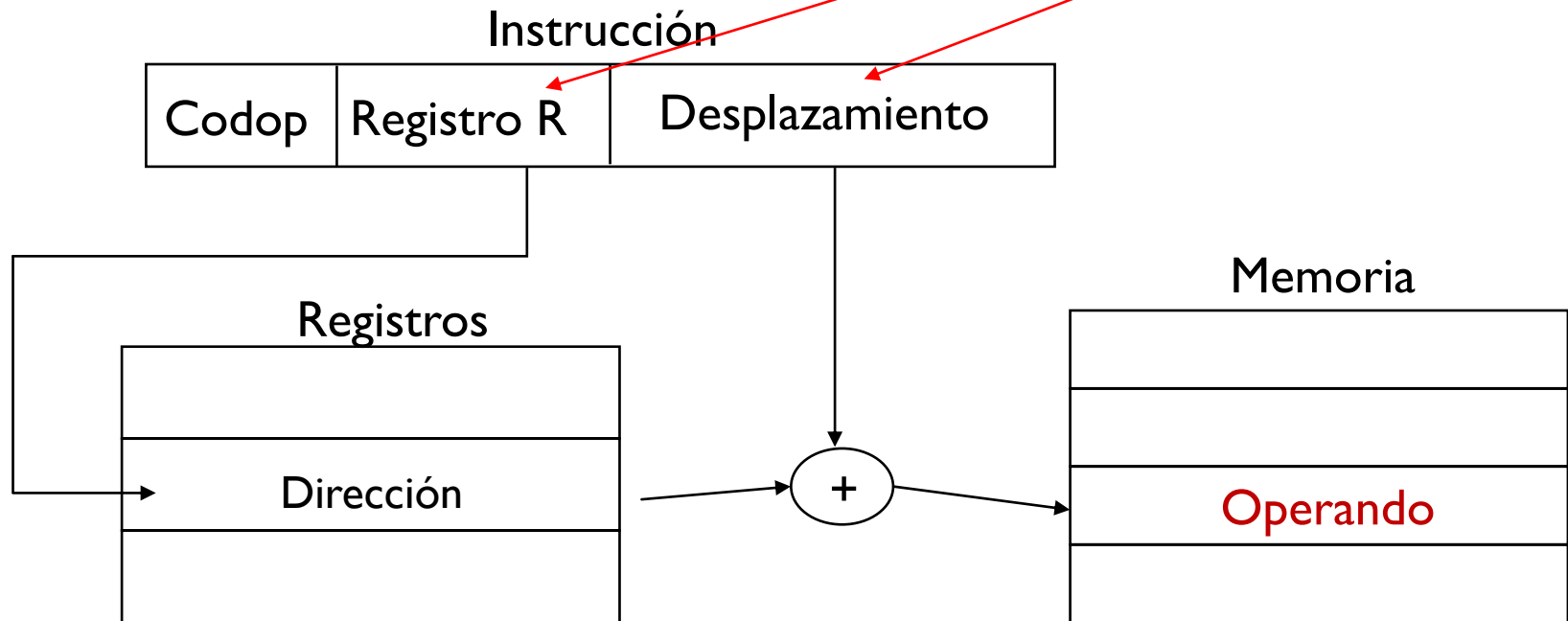
# Modos de direccionamiento

- ▶ El **modo de direccionamiento** es un procedimiento que permite determinar la **ubicación** de un operando, un resultado o una instrucción

- ▶ Implícito
- ▶ Inmediato
- ▶ Directo 
  - a registro
  - a memoria
- ▶ Indirecto 
  - a registro
  - a memoria
- ▶ Relativo 
  - **a registro índice**
  - **a registro base**
  - **a PC**
  - **a Pila**

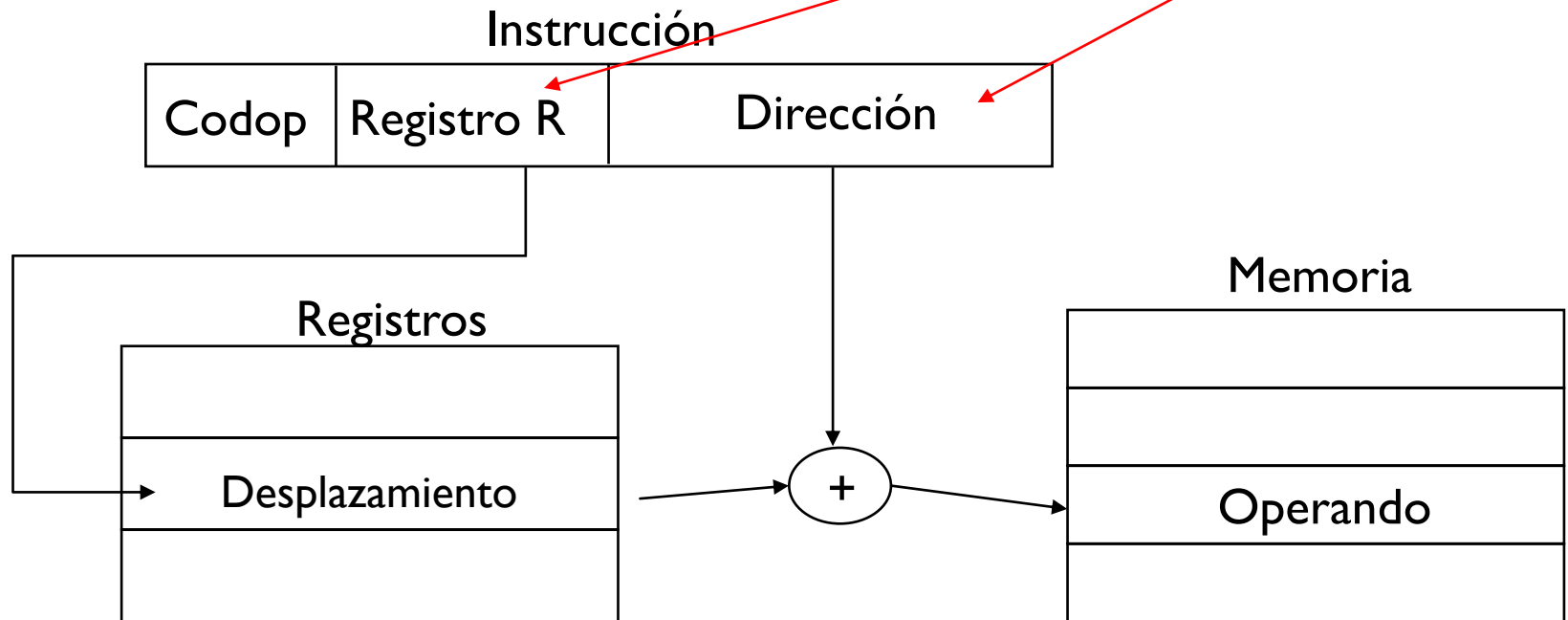
# Direccionamiento relativo a registro base

- Ejemplo: `lw a0 12(tl)`
  - Carga en a0 el contenido de la posición de memoria dada por  $tl + 12$
  - Utiliza dos campos de la instrucción, `tl` tiene la dirección base



# Direccionamiento relativo a registro índice

- Ejemplo: `lw a0 dir(tl)`
  - Carga en a0 el contenido de la posición de memoria dada por  $tl + dir$
  - Utiliza dos campos: tl representa el desplazamiento (índice) respecto a la dirección dir



# Utilidad: acceso a vectores

```
int v[5] ;
```

```
main ( )
```

```
{
```

```
    v[3] = 5 ;
```

```
    v[4] = 8 ;
```

```
}
```

```
.data
```

```
    .align 2#siguiente alineado a 4 byte
```

```
v: .space 20    # 5int*4bytes/int
```

```
.text
```

```
main:
```

```
la    t0 v
```

```
li    t1 5
```

```
sw    t1 12(t0)
```

```
li    t0 16
```

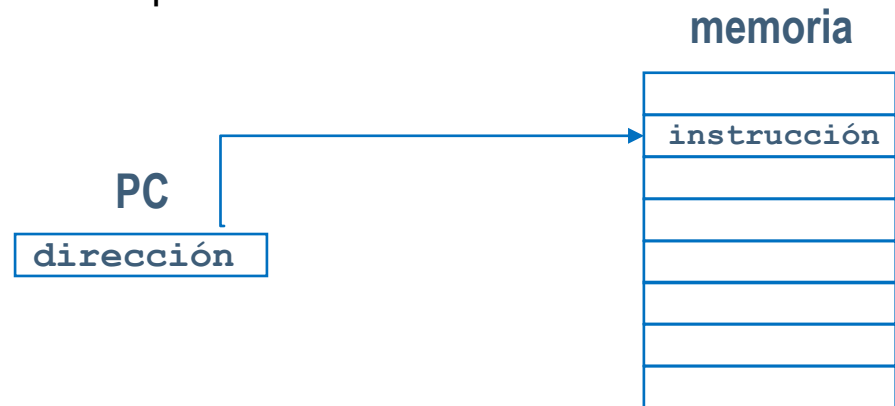
```
li    t1 8
```

```
sw    t1 v(t0)
```



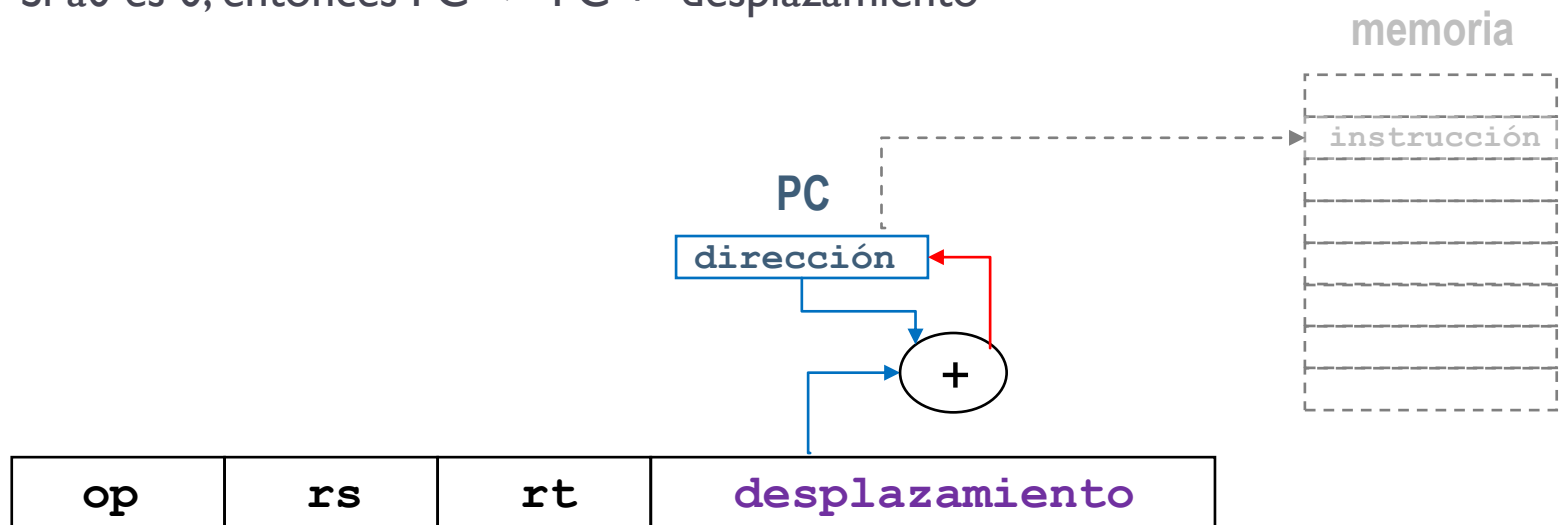
# Direcccionamiento relativo al contador de programa

- ▶ El contador de programa PC:
  - ▶ Es un registro de 32 bits (4 bytes) en un computador de 32-bits
  - ▶ Almacena la dirección de la siguiente instrucción a ejecutar
    - ▶ Apunta a una palabra (4 bytes) con la instrucción a ejecutar
    - ▶ PC en un computador de 32-bits se actualiza por defecto como  $PC = PC + 4$



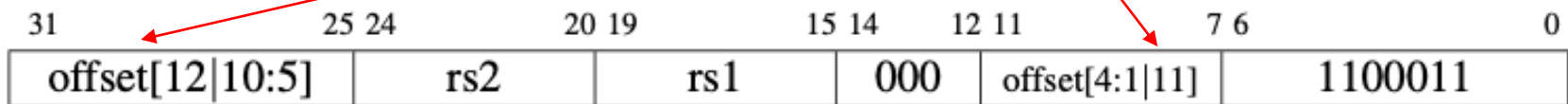
# Direcccionamiento relativo al contador de programa

- ▶ Ejemplo: **beq a0 x0 etiqueta**
  - ▶ Se codifica etiqueta como el desplazamiento desde la dirección de memoria donde está esta instrucción, hasta la posición de memoria indicada en **etiqueta**.
    - ▶ Etiqueta se codifica como desplazamiento (dirección  $\rightarrow$  # instrucciones a saltar)
  - ▶ Si a0 es 0, entonces  $PC \leq PC + \text{"desplazamiento"}$



# Direccionamiento relativo a PC en el RISC-V

- ▶ La instrucción `beq t0, x1, ofsset` se codifica en la instrucción:



- ▶ Etiqueta tiene que codificarse en el campo “ofsset”
- ▶ ¿Cómo se actualiza el PC si `t0 == x1` y cuánto vale fin cuando se genera código máquina?

```
bucle:    beq    t0, x1, fin
          add    t8, t4, t4
          addi   t0, x0, -1
          j      bucle
fin:      . . .
```

# Direccionamiento relativo a PC en el RISC-V

- ▶ Si se cumple la condición
  - ▶  $PC = PC + \text{offset}$
  - ▶ el valor de offset puede ser positivo o negativo
- ▶ Por tanto en:

```
bucle:    beq    $t0, $1,  fin
          add    $t8, $t4, $t4
          addi   $t0, $0,  -1
          j      bucle
fin:      . . .
```

- ▶ `fin == 12`
  - ▶ Cuando se ejecuta una instrucción, el PC apunta a la siguiente

# Utilidad: desplazamientos en bucles

```
        li    t0 8
        li    t1 4
        li    t2 1
        li    t4 0
while:   bge   t4 t1 fin
        mul   t2 t2 t0
        addi  t4 t4 1
        j     while
fin:     mv    t2 t4
```

- ▶ **fin** representa la dirección donde se encuentra la instrucción `mv`
- ▶ **while** representa la dirección donde se encuentra la instrucción `bge`

# Utilidad: desplazamientos en bucles

```
        li    $t0 8
        li    $t1 4
        li    $t2 1
        li    $t4 0
while:   bge   $t4 $t1 fin
        mul   $t2 $t2 $t0
        addi  $t4 $t4 1
        j     while
fin:     move  $t2 $t4
```

Dirección	Contenido
0x0000100	li    \$t0    8
0x0000104	li    \$t1    4
0x0000108	li    \$t2    1
0x000010C	li    \$t4    0
0x0000110	bge   \$t4 \$t1 fin
0x0000114	mul   \$t2 \$t2 \$t0
0x0000118	addi  \$t4 \$t4 1
0x000011C	j     while
0x0000120	move  \$t2 \$t4

# Utilidad: desplazamientos en bucles

```
        li    $t0 8
        li    $t1 4
        li    $t2 1
        li    $t4 0
while:   bge   $t4 $t1 fin
        mul   $t2 $t2 $t0
        addi  $t4 $t4 1
        j     while
fin:     move  $t2 $t4
```

En `bge $t4 $t1 fin`

`fin` representa un desplazamiento  
respecto al PC actual => **12**

PC = PC + 12

En `j while`

`while` representa un desplazamiento  
respecto al PC actual => **-16**

PC = PC - 16

Dirección	Contenido
0x0000100	li \$t0 8
0x0000104	li \$t1 4
0x0000108	li \$t2 1
0x000010C	li \$t4 0
0x0000110	bge \$t4 \$t1 fin
0x0000114	mul \$t2 \$t2 \$t0
0x0000118	addi \$t4 \$t4 1
0x000011C	j while
0x0000120	move \$t2 \$t4

# Utilidad: desplazamientos en bucles

```
        li    $t0 8
        li    $t1 4
        li    $t2 1
        li    $t4 0
while:   bge   $t4 $t1 fin
        mul   $t2 $t2 $t0
        addi  $t4 $t4 1
        j     while
fin:     move  $t2 $t4
```

En `bge $t4 $t1 fin`

`fin` representa un desplazamiento  
respecto al PC actual => **12**

$PC = PC + 12$

En `j while`

`while` representa un desplazamiento  
respecto al PC actual => **-16**

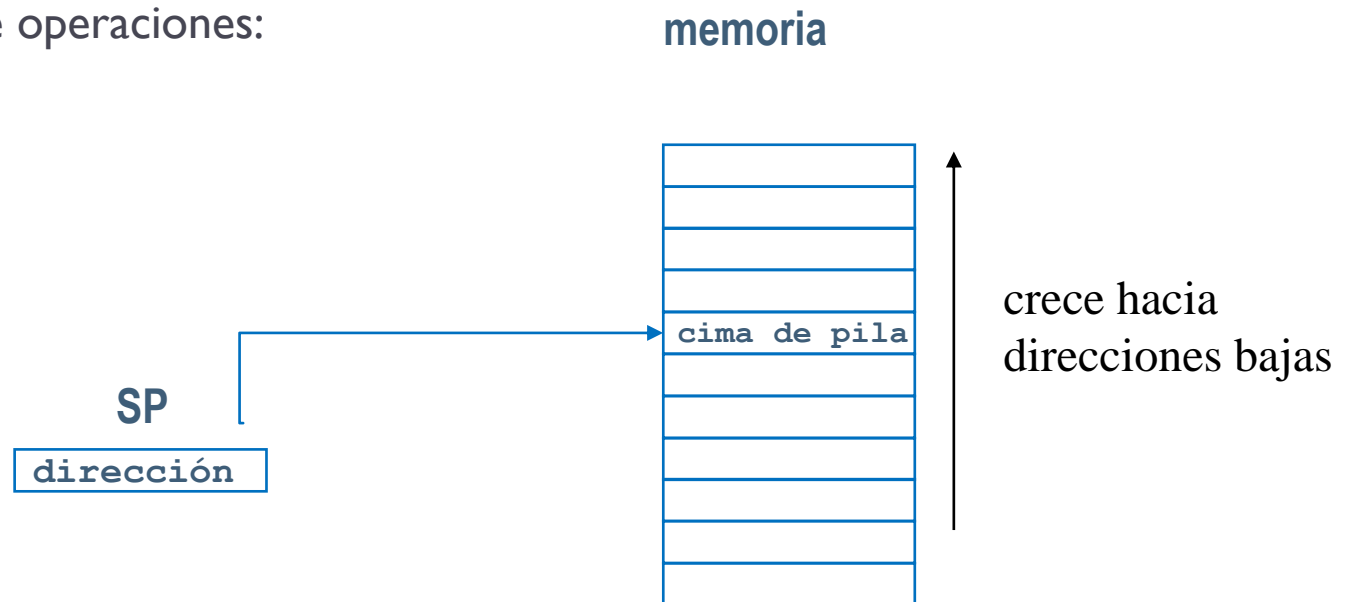
$PC = PC - 16$

Dirección	Contenido
0x0000100	li \$t0 8
0x0000104	li \$t1 4
0x0000108	li \$t2 1
0x000010C	li \$t4 0
0x0000110	bge \$t4 \$t1 <b>12</b>
0x0000114	mul \$t2 \$t2 \$t0
0x0000118	addi \$t4 \$t4 1
0x000011C	j <b>-16</b>
0x0000120	move \$t2 \$t4



# Direccionamiento relativo a pila

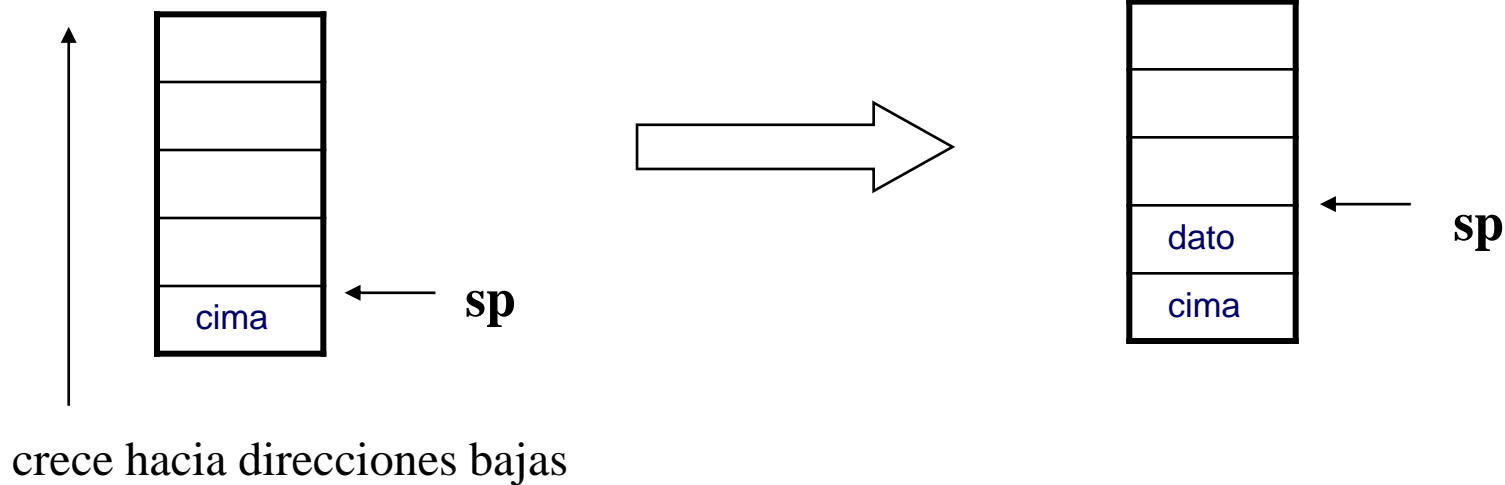
- ▶ El puntero de pila SP (*Stack Pointer*):
  - ▶ Es un registro de 32 bits (4 bytes) en el RISC-V<sub>32</sub>
  - ▶ Almacena la dirección de la cima de pila
    - ▶ Apunta a una palabra (4 bytes)
- ▶ Dos tipos de operaciones:
  - ▶ push
  - ▶ pop



# Operación PUSH

## **PUSH Reg**

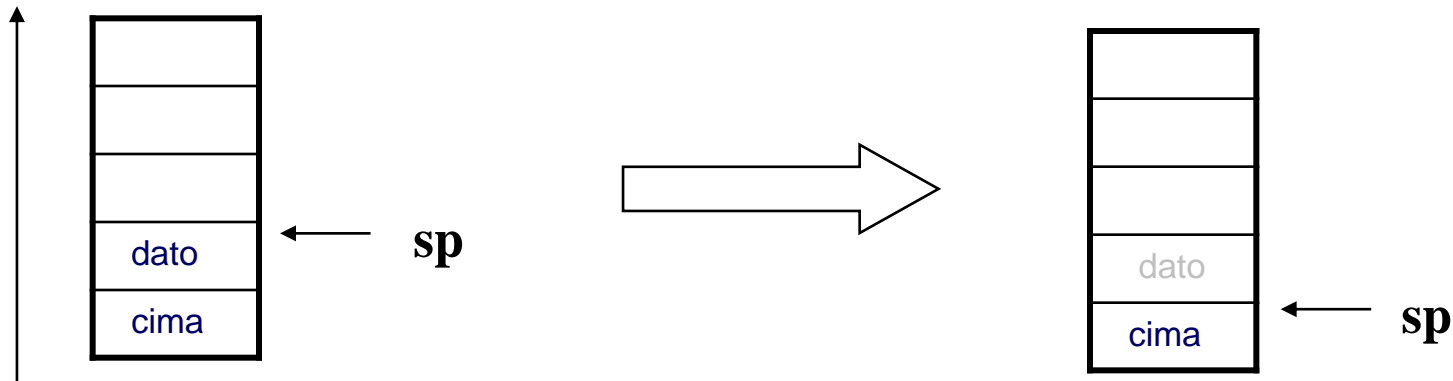
Apila el contenido del registro (dato)



# Operación POP

## POP Reg

Desapila el contenido del registro (dato)  
Copia dato en el registro Reg



crece hacia direcciones bajas

# Direcccionamiento de pila en el RISC-V

- ▶ RISC-V no dispone de instrucciones PUSH o POP.
- ▶ El registro puntero de pila (sp) es visible al programador.
  - ▶ Se va a asumir que el puntero de pila apunta al último elemento de la pila

## PUSH t0

```
addi sp, sp, -4  
sw    t0, 0(sp)
```

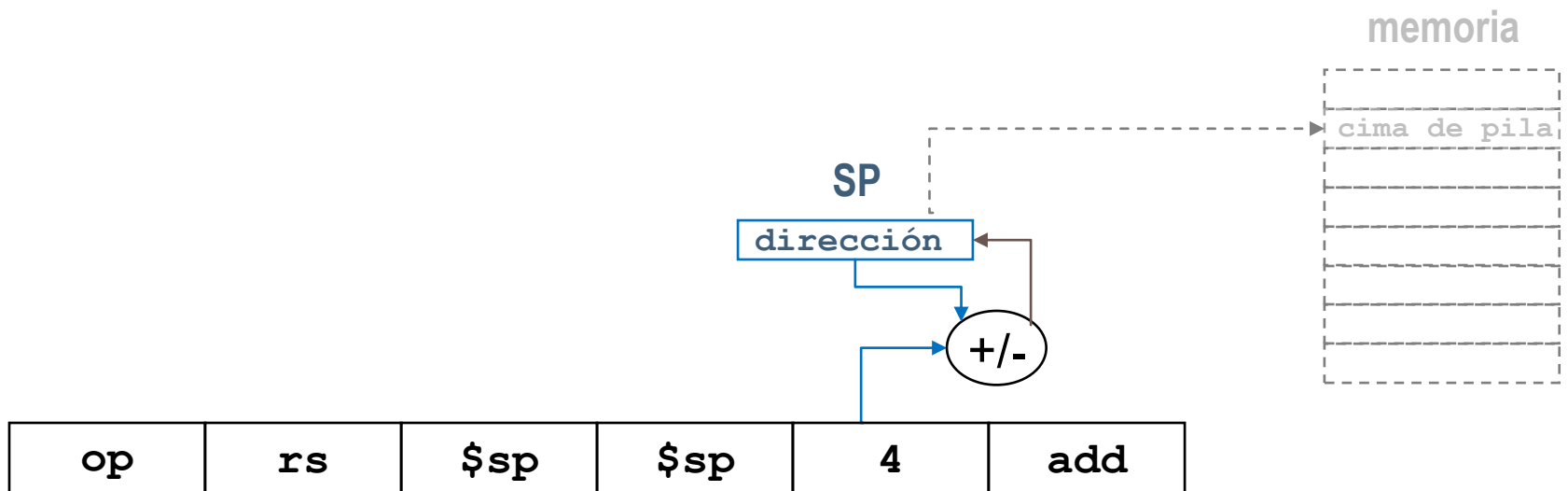
## POP t0

```
lw    t0, 0(sp)  
addi sp, sp, 4
```

# Operación PUSH en RISC-V

## ► Ejemplo: **push a0**

- `addi sp sp -4` #  $SP = SP - 4$
- `sw a0 0(sp)` # `memoria[SP] = a0`



# Ejercicio

- Indique el tipo de direccionamiento usado en las siguientes instrucciones RISC-V:

1. `li t1 4`
2. `lw t0 4(a0)`
3. `bne x0 a0 etiqueta`

# Ejercicio (solución)

1. `li t1 4`

- ▶ `t1` -> directo a registro
- ▶ `4` -> inmediato

2. `lw t0 4(a0)`

- ▶ `t0` -> directo a registro
- ▶ `4(a0)` -> relativo a registro base

3. `bne x0 a0 etiqueta`

- ▶ `a0` -> directo a registro
- ▶ `etiqueta` -> relativo a contador de programa

# Ejemplos de tipos de direccionamiento

## ► la t0 **label** inmediato

- El segundo operando de la instrucción es una dirección
- PERO no se accede a esta dirección, la propia dirección es el operando

## ► lw t0 **label** directo a memoria (!)

- El segundo operando de la instrucción es una dirección
- Hay que acceder a esta dirección para tener el valor con el que trabajar

## ► bne t0 t1 **label** relativo a registro PC

- El tercer operando de la instrucción es desplazamiento respecto al PC
- label se codifica como un número en complemento a dos que representa el desplazamiento (como palabras) relativo al registro PC



# Contenidos

- ▶ Fundamentos básicos de la programación en ensamblador
- ▶ Ensamblador del RISC-V<sub>32</sub>, modelo de memoria y representación de datos
- ▶ **Formato de las instrucciones** y modos de direccionamiento
- ▶ Llamadas a procedimientos y uso de la pila

# Formato de instrucciones

- ▶ Una instrucción se divide en **campos**
- ▶ Una instrucción máquina es autocontenida e incluye:
  - ▶ Código de operación
  - ▶ Operandos
    - ▶ Tipos de representación de los operandos
  - ▶ Resultado
  - ▶ Dirección de la siguiente instrucción
- ▶ Ejemplo de campos en una instrucción del RISC-V:



# Formato de una instrucción

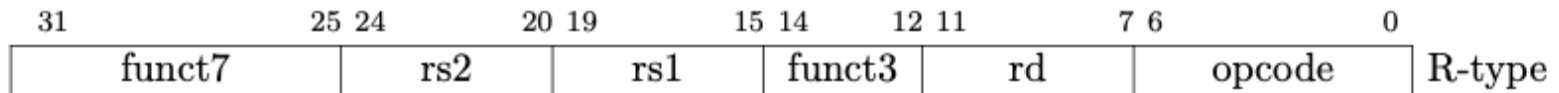
- ▶ El **formato** especifica, por cada campo de la instrucción:
  - ▶ El significado de cada campo
  - ▶ El número de bits de cada campo
  - ▶ Cómo se codifica cada campo
    - ▶ Binario, complemento a uno, a dos, etc.
- ▶ Normalmente:
  - ▶ Una arquitectura ofrece unos pocos formatos de instrucción.
    - ▶ Simplicidad en el diseño de la unidad de control.
  - ▶ Campos del mismo tipo siempre igual longitud.
  - ▶ Selección mediante **código de operación**.
    - ▶ **Normalmente el primer campo.**

# Longitud de formato

- ▶ La **longitud del formato** es el número de bits para codificar la instrucción
  - ▶ El tamaño habitual es una palabra (o múltiples palabras)
  - ▶ En RISC-V<sub>32</sub> el tamaño de todas las instrucciones es una palabra.
- ▶ Dos tipos:
  - ▶ **Longitud única:**
    - ▶ Todas las instrucciones tienen la misma longitud de formato.
    - ▶ Ejemplos:
      - MIPS32: 32 bits, PowerPC: 32 bits, ...
  - ▶ **Longitud variable:**
    - ▶ Distintas instrucciones tienen distinta longitud de formato.
    - ▶ ¿Cómo se sabe la longitud? → Código de operación
    - ▶ Ejemplos:
      - IA32 (Procesadores Intel): Número variable de bytes.

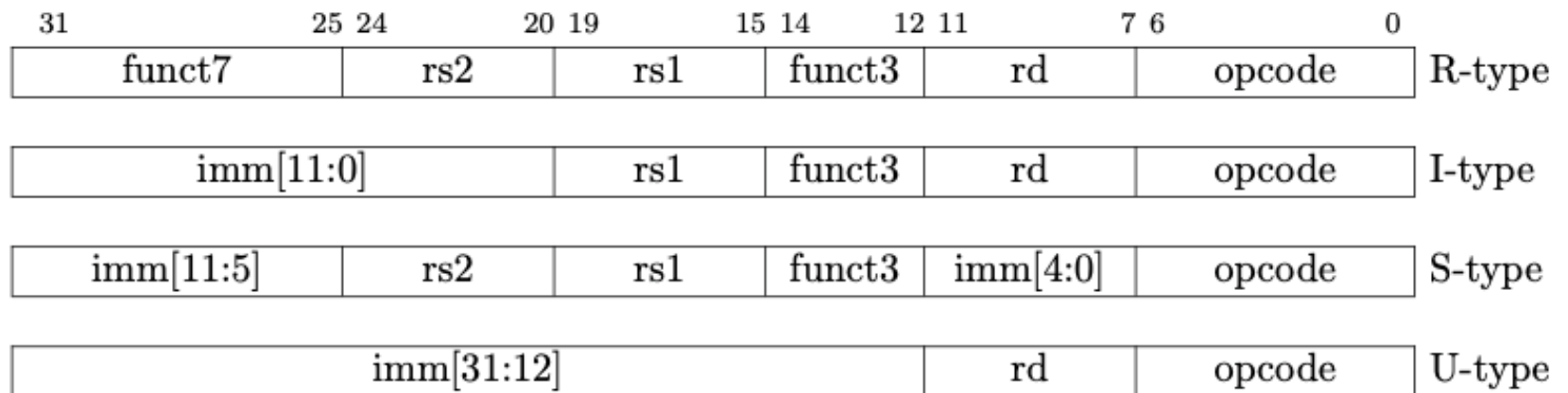
# Código de operación

- ▶ **Tamaño fijo:**
  - ▶  $n$  bits  $\rightarrow 2^n$  códigos de operación.
  - ▶  $m$  códigos de operación  $\rightarrow \log_2 m$  bits.
- ▶ **Campos de extensión**
  - ▶ RISC-V (instrucciones aritméticas-lógicas)
  - ▶  $Op = 0$ ; la instrucción está codificada en `fun.X`



- ▶ **Tamaño variable:**
  - ▶ Instrucciones más frecuentes = Tamaños más cortos.

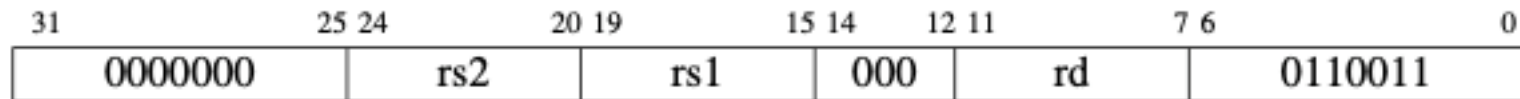
# Ejemplo: Formato de las instrucciones del RISC-V



# Ejemplo de formato en el RISC-V

## ► RISC-V Instruction:

► `add rd rs1 rs2`



# Ejercicio

- ▶ Sea un computador de 16 bits de tamaño de palabra, que incluye un repertorio con 60 instrucciones máquina y con un banco de registros que incluye 8 registros.

Se pide:

Indicar el formato de la instrucción **ADDx RI R2 R3**, donde RI, R2 y R3 son registros.



# Ejercicio (solución)

palabra -> 16 bits

60 instrucciones

8 registros (en BR)

ADDx R1(reg.), R2(reg.), R3(reg.)

- ▶ Palabra de 16 bits define el tamaño de la instrucción

16 bits



# Ejercicio (solución)

palabra -> 16 bits

60 instrucciones

8 registros (en BR)

ADDx R1(reg.), R2(reg.), R3(reg.)

- ▶ Para 60 instrucciones se necesitan 6 bits (mínimo)



# Ejercicio (solución)

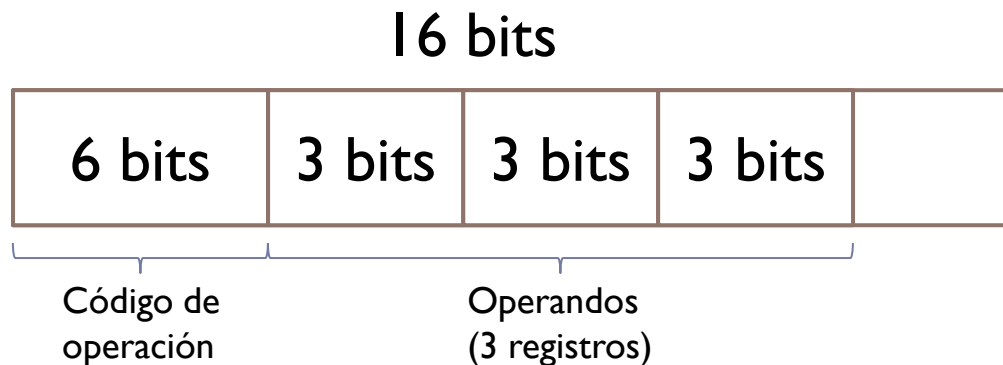
palabra -> 16 bits

60 instrucciones

8 registros (en BR)

ADDx R1(reg.), R2(reg.), R3(reg.)

- ▶ Para 8 registros se necesitan 3 bits (mínimo)



# Ejercicio (solución)

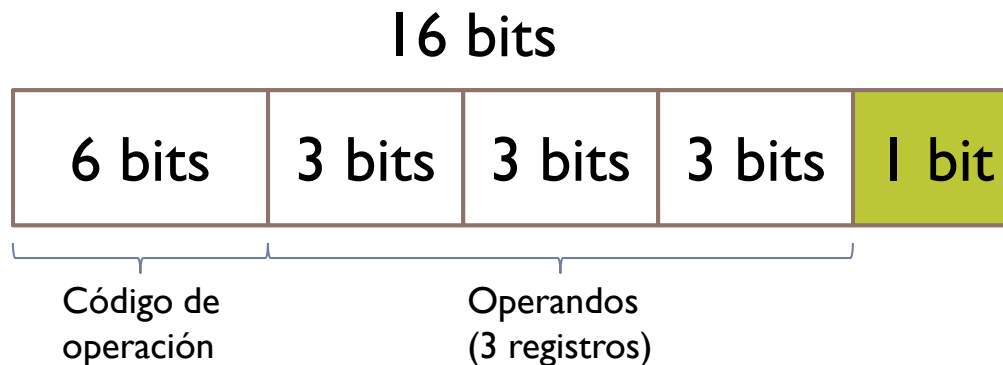
palabra -> 16 bits

60 instrucciones

8 registros (en BR)

ADDx R1(reg.), R2(reg.), R3(reg.)

- Sobra 1 bit ( $16 - 6 - 3 - 3 - 3 = 1$ ), usado de relleno



# Juego de instrucciones

- ▶ Queda **definido** por:
  - ▶ Conjunto de instrucciones
  - ▶ Formato de las instrucciones
  - ▶ Registros
  - ▶ Modos de direccionamiento
  - ▶ Tipos de datos y formatos

# Juego de instrucciones

- ▶ Distintas formas para la **clasificación** de un juego de instrucciones:
  - ▶ Complejidad del juego de instrucciones
    - ▶ CISC vs RISC
  - ▶ Modo de ejecución
    - ▶ Pila
    - ▶ Registro
    - ▶ Registro-Memoria, Memoria-Registro, ...

# CISC vs RISC

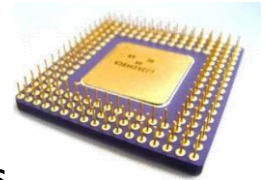
## ▶ *Complex Instruction Set Computer*

- ▶ Muchas instrucciones
- ▶ Instrucciones complejas
  - ▶ Más de una palabra
  - ▶ Unidad de control más compleja
  - ▶ Mayor tiempo de ejecución
- ▶ Diseño irregular

## ▶ *Reduced Instruction Set Computer*

- ▶ Instrucciones simples y ortogonales
  - ▶ Ocupan una palabra
  - ▶ Instrucciones sobre registros
  - ▶ Uso de los mismos modos de direccionamiento para todas las instrucciones (alto grado de ortogonalidad)
- ▶ Diseño más compacto:
  - ▶ Unidad de control más sencilla y rápida
  - ▶ Espacio sobrante para más registros y memoria caché

- ▶ Alrededor del 20% de las instrucciones ocupa el 80% del tiempo total de ejecución de un programa
- ▶ El 80% de las instrucciones no se utilizan casi nunca
- ▶ 80% del silicio infrautilizado, complejo y costoso



# Modos de ejecución

- ▶ Los modos de ejecución indican el número de operandos y el tipo de operandos que pueden especificarse en una instrucción.
  - ▶ 0 direcciones → Pila.
    - PUSH 5; PUSH 7; ADD
  - ▶ 1 dirección → Registro acumulador.
    - ADD RI →  $AC \leftarrow AC + RI$
  - ▶ 2 direcciones → Registros, Registro-memoria, Memoria-memoria.
    - ADD .R0, .RI ( $R0 \leftarrow R0 + RI$ )
  - ▶ 3 direcciones → Registros, Registro-memoria, Memoria-memoria.
    - ADD .R0, .RI, .R2



# Ejercicio

- ▶ Sea un computador de 16 bits, que direcciona la memoria por bytes y que incluye un repertorio con 60 instrucciones máquina. El banco de registros incluye 8 registros. Indicar el formato de la instrucción `ADDV R1, R2, M`, donde R1 y R2 son registros y M es una dirección de memoria.

# Ejercicio

- ▶ Sea un computador de 32 bits, que direcciona la memoria por bytes. El computador incluye 64 instrucciones máquina y 128 registros. Considere la instrucción SWAPM dir1, dir2, que intercambia el contenido de las posiciones de memoria dir1 y dir2. Se pide:
  - ▶ Indicar el espacio de memoria direccionable en este computador.
  - ▶ Indicar el formato de la instrucción anterior.
  - ▶ Especifique un fragmento de programa en ensamblador del RISC-V 32 equivalente a la instrucción máquina anterior.
  - ▶ Si se fuerza a que la instrucción quepa en una palabra, qué rango de direcciones se podría contemplar considerando que las direcciones se representan en binario puro.

Grupo ARCOS

**uc3m** | Universidad **Carlos III** de Madrid

# Tema 3 (III)

## Fundamentos de la programación en ensamblador

Estructura de Computadores  
Grado en Ingeniería Informática

