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 MIPS 32, modelo de memoria y representación de datos
- Formato de las instrucciones y modos de direccionamiento
- Llamadas a procedimientos y uso de la pila

Formatos de las instrucciones de acceso a memoria (Repaso)

```
lw
sw
lb
sb
lbu
```

```
Registro, dirección de memoria
```

Número que representa una dirección

Etiqueta simbólica que representa una dirección

(registro): representa la dirección almacenada en el registro

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

etiqueta + num: representa la dirección que se obtiene de sumar etiqueta con num

Instrucciones y pseudoinstrucciones del MIPS 32

- Una instrucción en ensamblador se corresponde con una instrucción máquina
  - Ocupa 32 bits
  - `addi  $t1, $t1,  2`
- Una pseudoinstrucción en ensamblador se corresponde con varias instrucciones máquina.
  - `li $t1,  0x00800010`
    - No cabe en 32 bits, pero se puede utilizar como pseudoinstrucción.
    - Es equivalente a:
      - `lui $t1, 0x0080`
      - `ori $t1, $t1, 0x0010`

Otro ejemplo de pseudoinstrucción del MIPS 32

- La pseudoinstrucción `move`

```
move      reg2, reg1
```

- Se convierte en:

```
add       reg2,$zero,reg1
```

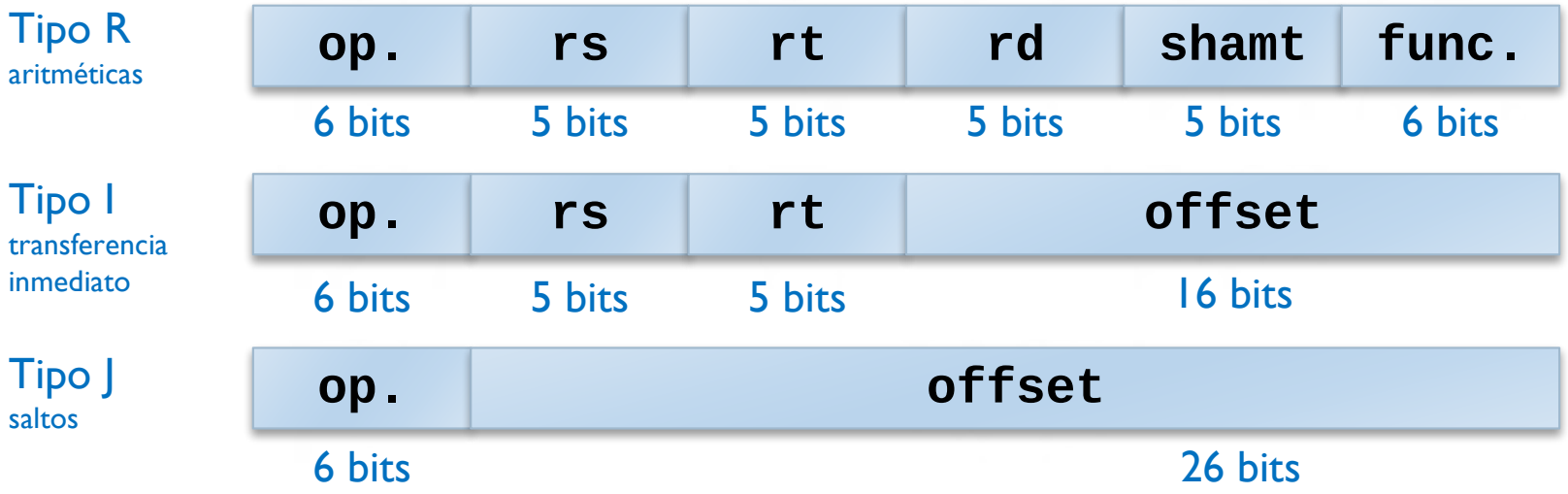
Información de una instrucción

- El tamaño de la instrucción se ajusta al de palabra (o múltiplo)
- Una instrucción máquina se divide en campos:
  - Operación a realizar
  - Operandos a utilizar
    - Puede haber operando implícitos
- El formato de una instrucción indica los campos y su tamaño:
  - Uso de formato sistemático
  - Tamaño de un campo limita los valores que codifica
    - Un campo de n bits permite codificar 2^n valores distintos

Información de una instrucción

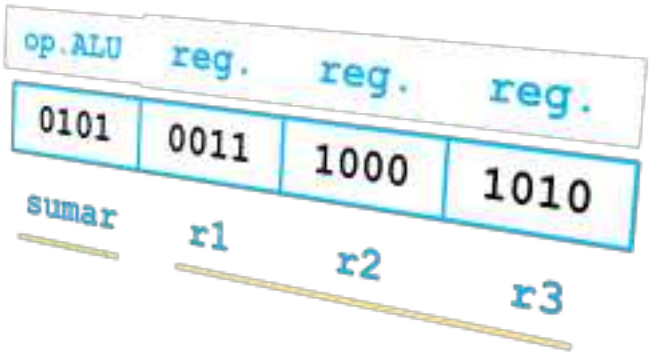
- Se utiliza unos pocos formatos:
  - Cada instrucción pertenecen a un formato
  - Según el código de operación se conoce el formato asociado

- Ejemplo: formatos en MIPS



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 ⟵ PC + '4' (apuntar a la siguiente instrucción)
      - Explícito: j 0x01004 (modifica el PC)

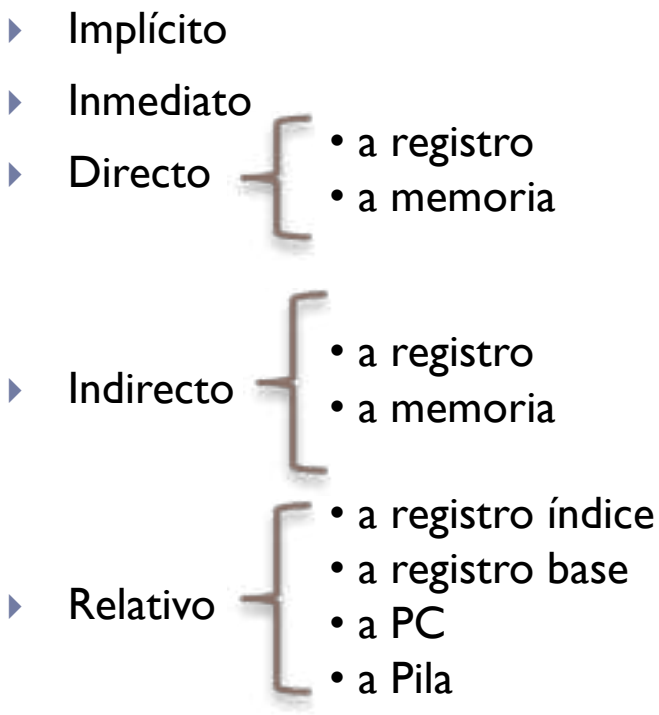


Ubicaciones posibles para los operandos

- En la propia instrucción
- En los registros del procesador
- En memoria principal
- En unidades de Entrada/Salida (I/O)

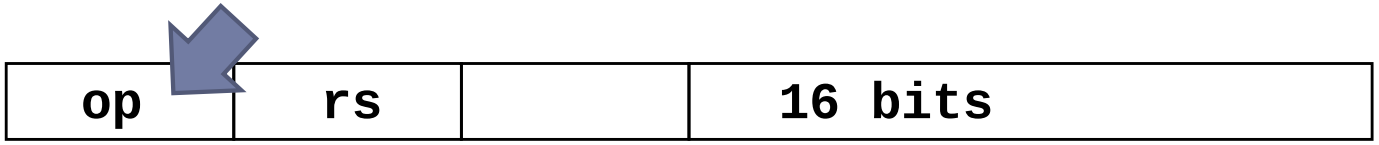
Modos de direccionamiento

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



Direccionamiento implícito

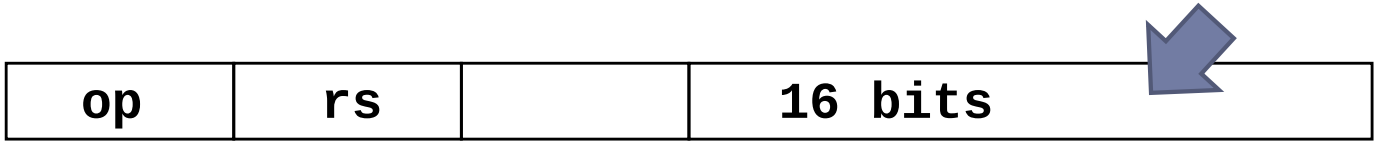
- El operando no está codificado en la instrucción, pero forma parte de esta
- Ejemplo: beqz $a0 etiqueta
  - Si registro $a0 es cero, salta a etiqueta.
  - $a0 es un operando, $zero es el otro (implícito)



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

Direccionamiento inmediato

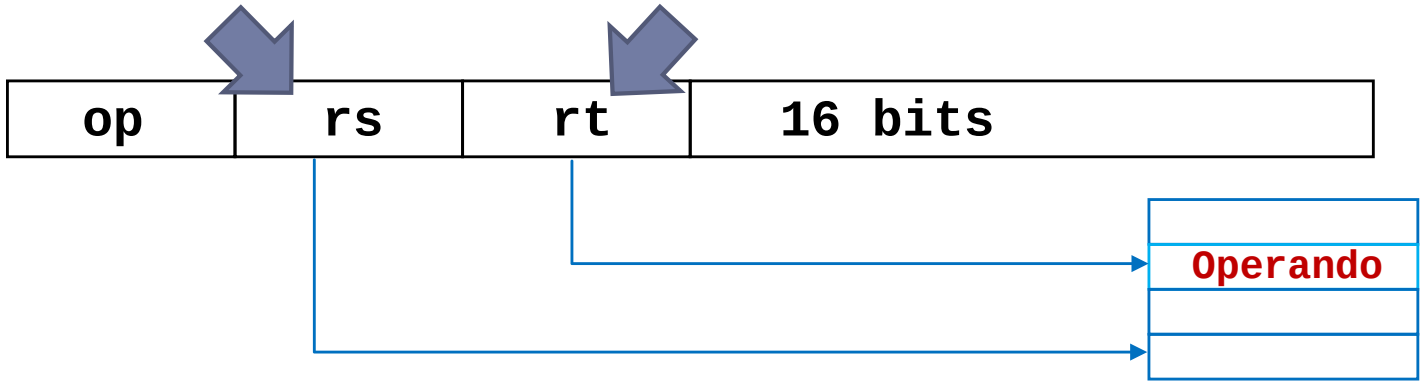
- El operando forma parte de la instrucción.
- Ejemplo: li $a0 0x4f51
  - Carga en el registro $a0 el valor inmediato 0x4f51.
  - El valor 0x00004f51 está codificado en la propia instrucción.



- 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, 0x0080`
      - `ori $t1, $t1, 0x0010`

Direccionamiento directo a registro (direccionamiento de registro)

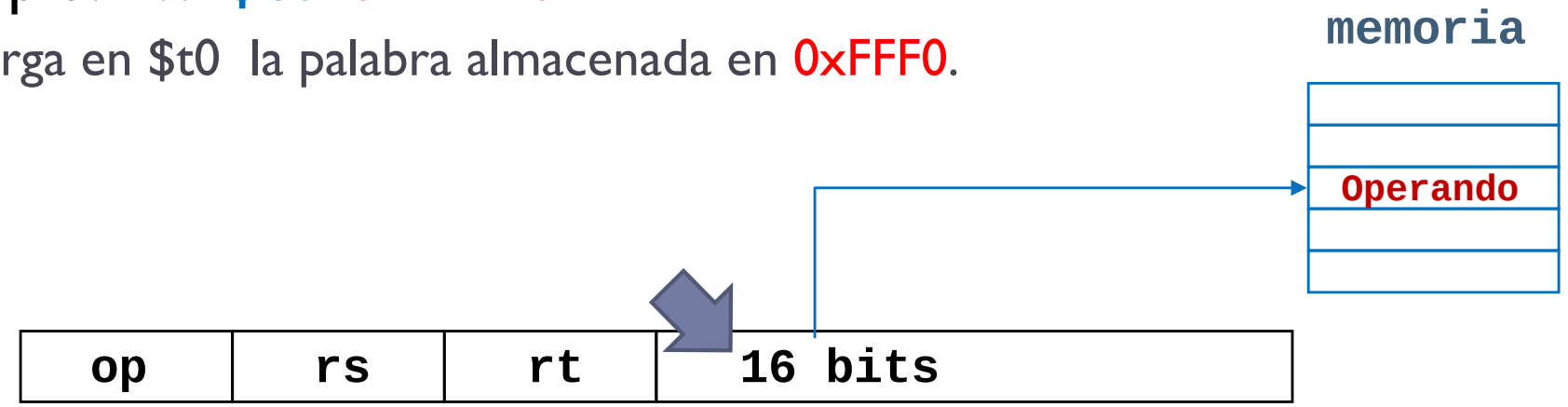
- El operando se encuentra en el registro.
- Ejemplo: move $a0 $a1
  - Copia en el registro $a0 el valor que hay en el registro $a1.
  - El identificador de $a0 y $a1 está codificado en la instrucción.



- 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: lw $t0 0xFFF0
  - Carga en $t0 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.)

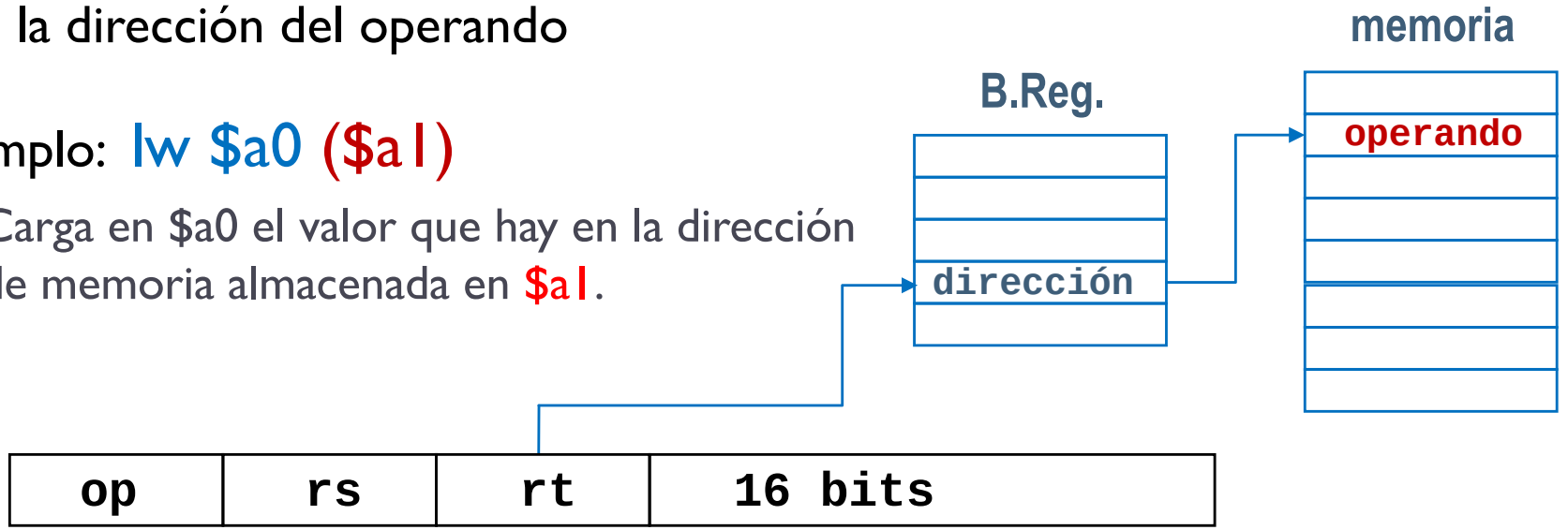
ARCOS @ UC3M
Félix García Carballeira, Alejandro Calderón Mateos

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

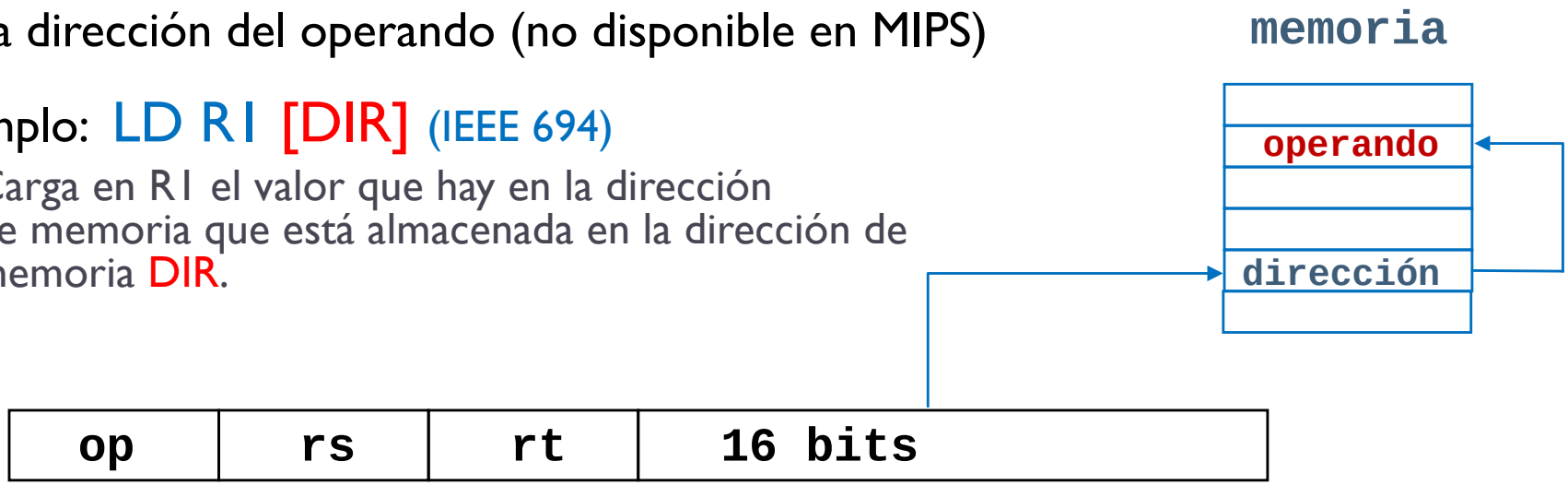
- Se indica en la instrucción el registro con la dirección del operando
- Ejemplo: lw $a0 ($a1)
  - Carga en $a0 el valor que hay en la dirección de memoria almacenada en $a1.



- V/I
  - ✓ Amplio espacio de direcciones, instrucciones cortas

Direccionamiento indirecto a memoria

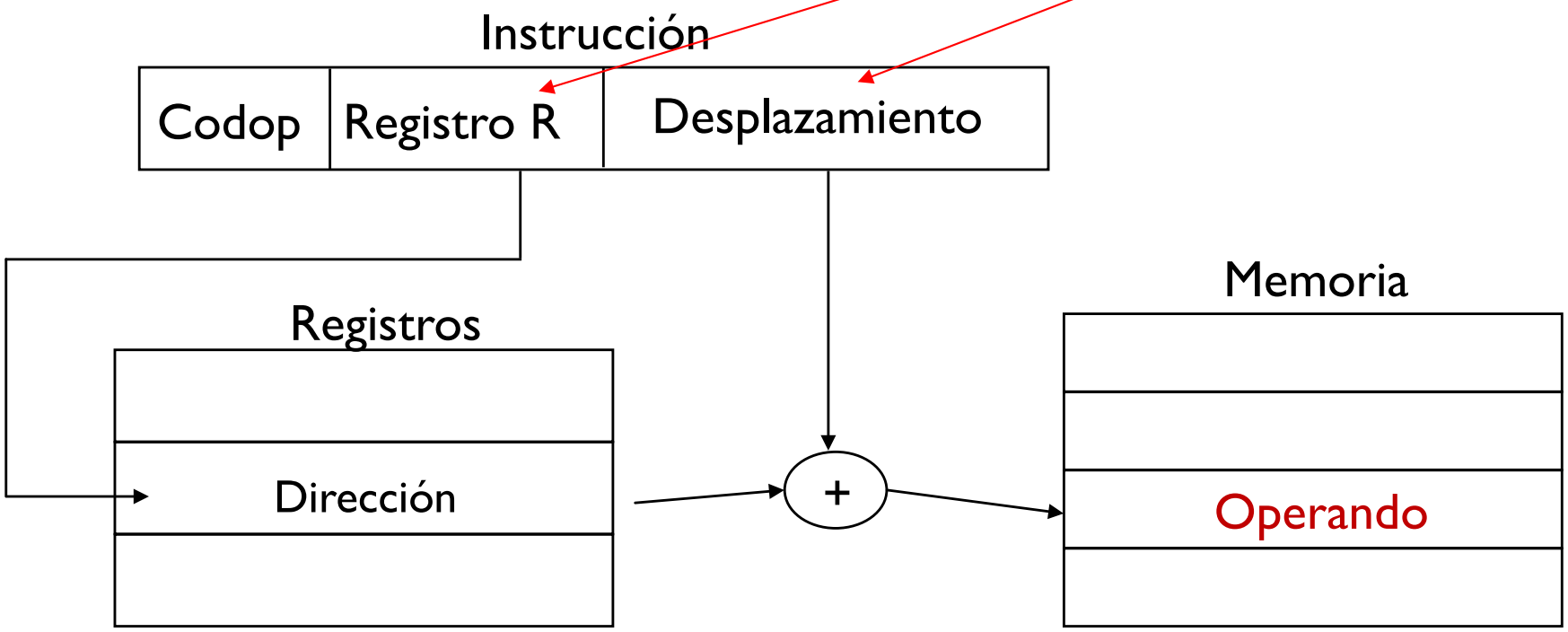
- Se indica en la instrucción la dirección donde está la de la dirección del operando (no disponible en MIPS)
- Ejemplo: LD R1 [DIR] (IEEE 694)
  - Carga en R1 el valor que hay en la dirección de memoria que está almacenada en la dirección de memoria DIR.
  - .



- V/I
  - ✓ Amplio espacio de direcciones
  - ✓ El direccionamiento puede ser anidado, multinivel o en cascada
    - Ejemplo: LD R1 [[[.R1]]]
  - ✗ Puede requerir varios accesos memoria
  - ✗ instrucciones más lentas de ejecutar

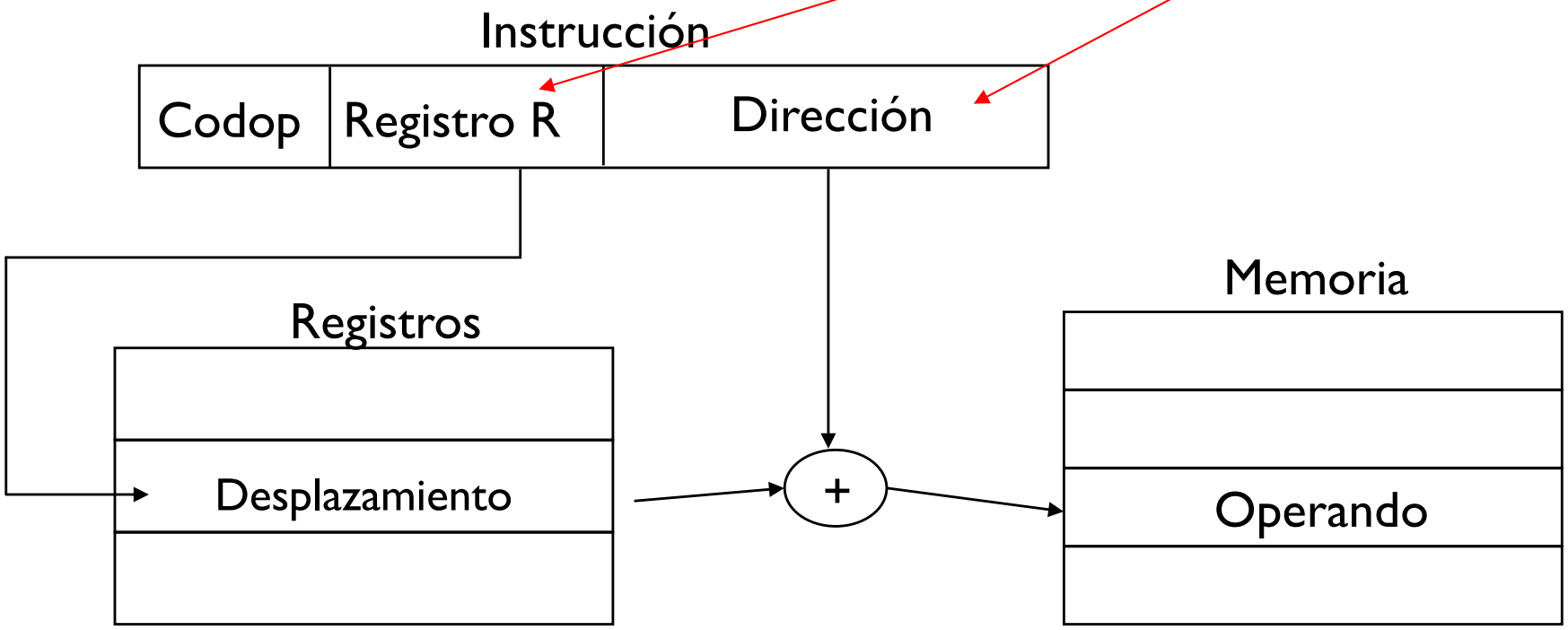
Direccionamiento relativo a registro base

- Ejemplo: lw $a0 12($t1)
  - Carga en $a0 el contenido de la posición de memoria dada por $t1 + 12
  - Utiliza dos campos de la instrucción, $t1 tiene la dirección base



Direccionamiento relativo a registro índice

- Ejemplo: lw $a0 dir($t1)
  - Carga en $a0 el contenido de la posición de memoria dada por $t1 + dir
  - Utiliza dos campos: $t1 representa el desplazamiento (índice) respecto a la dirección dir



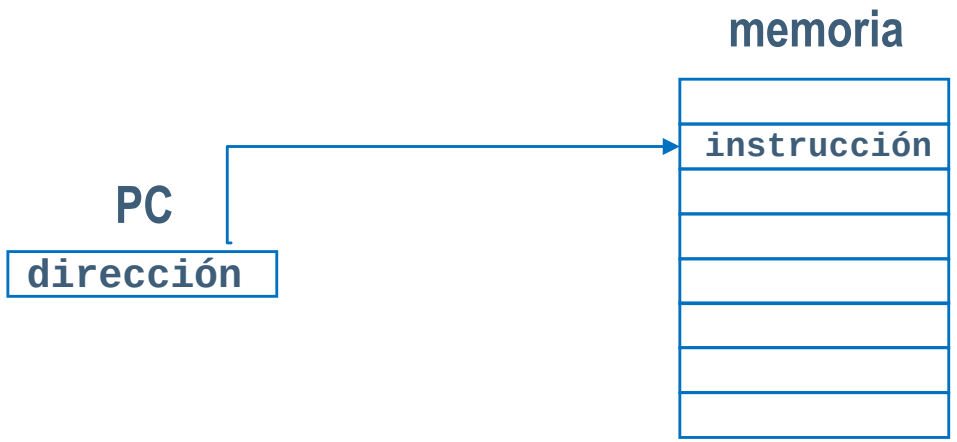
Utilidad: acceso a vectores

```
int v[5] ;

main ( )
{
  v[3] = 5 ;

  v[4] = 8 ;
}
```

```
.data
    .align 2#siguiente dato alineado a 4

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

.text
.globl main
main:
        la $t0 v
        li $t1 5
        sw $t1 12($t0)

        la $t0 16
        li $t1 8
        sw $t1 v($t0)
```

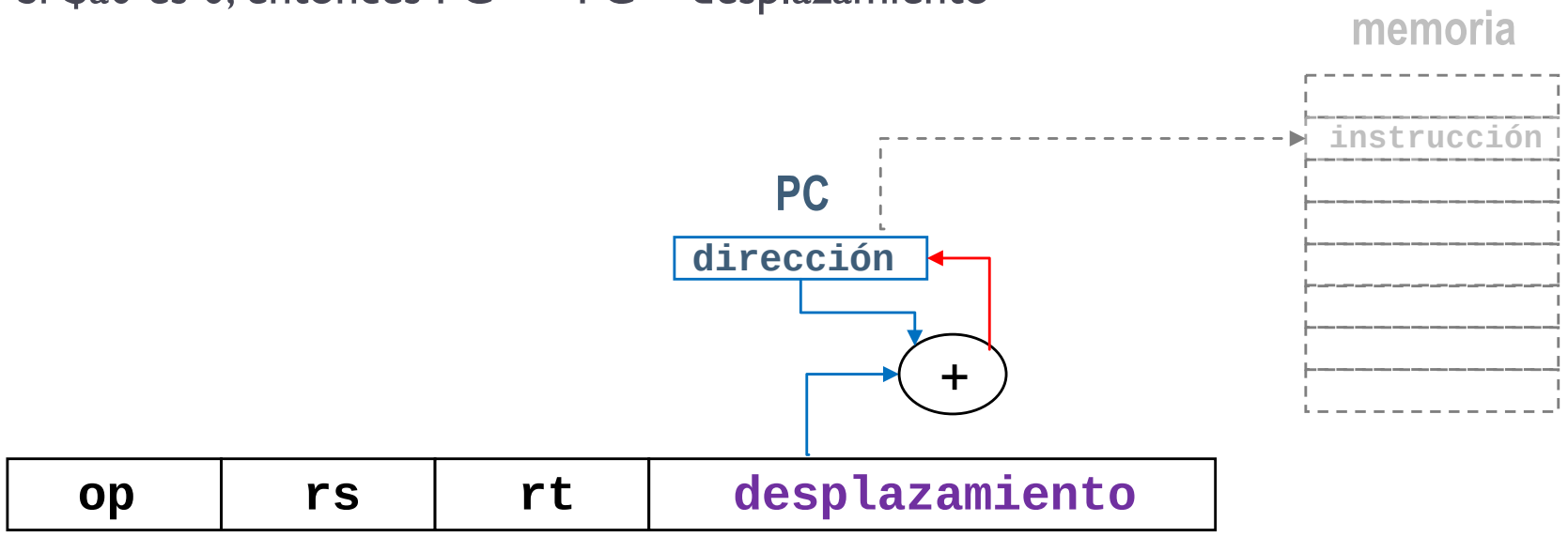
Direccionamiento relativo al contador de programa

- El contador de programa PC:
  - Es un registro de 32 bits (4 bytes)
  - Almacena la dirección de la siguiente instrucción a ejecutar
    - Apunta a una palabra (4 bytes) con la instrucción a ejecutar



Direccionamiento relativo al contador de programa

- Ejemplo: beqz $a0 etiqueta
  - La instrucción codifica etiqueta como el desplazamiento desde la dirección de memoria donde está esta instrucción, hasta la posición de memoria indicada en etiqueta.
  - Si $a0 es 0, entonces PC <= PC + desplazamiento



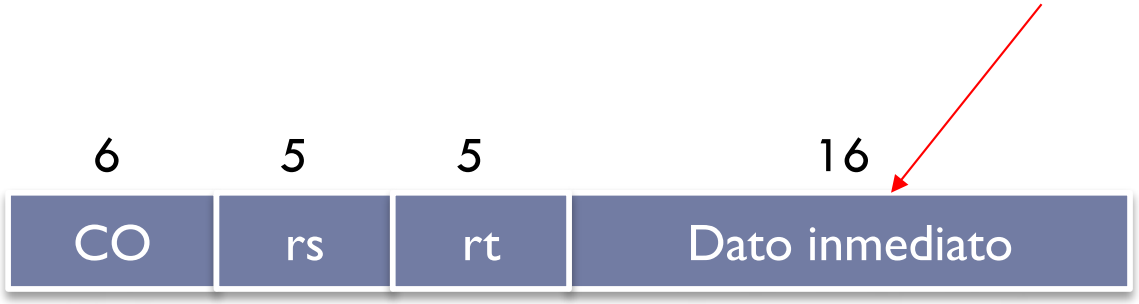
Contador de programa en el MIPS 32

- Los registros tienen 32 bits
- El contador de programa tiene 32 bits
- Las instrucciones ocupan 32 bits (una palabra)
- El contador de programa almacena la dirección donde se encuentra una instrucción
- La siguiente instrucción se encuentra 4 bytes después.
- Por tanto el contador de programa se actualiza:
  - PC = PC + 4

Dirección:	Instrucción:
0x00400000	or $2,$0,$0
0x00400004	slt $8,$0,$5
0x00400008	beq $8,$0,3
0x0040000c	add $2,$2,$4
0x00400010	addi $5,$5,-1
0x00400014	j 0x100001

Direccionamiento relativo a PC en el MIPS

- La instrucción `beq $t0, $1, etiqueta` se codifica en la instrucción:



- Etiqueta tiene que codificarse en el campo "Dato inmediato"
- ¿Cómo se actualiza el PC si $t0 == $1 y cuánto vale fin cuando se genera código máquina?

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

Direccionamiento relativo a PC en el MIPS

- Si se cumple la condición
  - PC = PC + (etiqueta* 4)
- Por tanto en:

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

- fin == 3
  - Cuando se ejecuta una instrucción, el PC apunta a la siguiente

ARCOS @ UC3M
Félix García Carballeira, Alejandro Calderón Mateos

Utilidad: desplazamientos en bucles

- fin representa la dirección donde se encuentra la instrucción move

```
            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
            b    while
fin:        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
          b    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	b 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
          b    while
fin:      move $t2  $t4
```

fin representa un desplazamiento respecto al PC actual => 3

PC = PC + 3 * 4

while representa un desplazamiento respecto al PC actual =>-4

PC = PC + (-4)*4

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	b 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
          b    while
fin:      move $t2  $t4
```

fin representa un desplazamiento respecto al PC actual => 3

PC = PC + 3 * 4

while representa un desplazamiento respecto al PC actual =>-4

PC = PC + (-4)*4

Dirección	Contenido
0x0000100	li $t0 8
0x0000104	li $t1 4
0x0000108	li $t2 1
0x000010C	li $t4 0
0x0000110	bge $t4 $t1 3
0x0000114	mul $t2 $t2 $t0
0x0000118	addi $t4 $t4 1
0x000011C	b -4
0x0000120	move $t2 $t4

Diferencia entre las instrucción b y j

Instrucción j direccion

Dirección de salto => PC = direccion

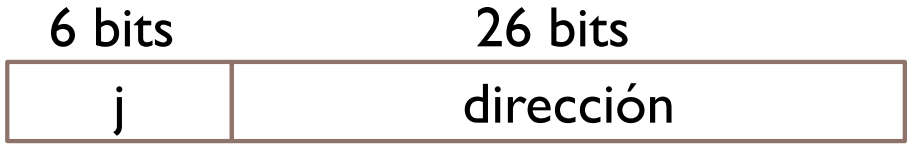
Instrucción b desplazamiento

Dirección de salto => PC = PC + desplazamiento * 4

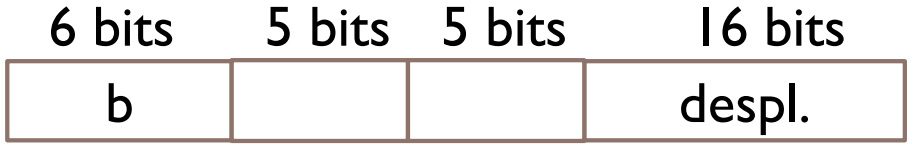
permite que el código sea reubicable en memoria

Ejercicio

- Dadas estas 2 instrucciones para realizar un salto incondicional:
  - 1) j etiqueta1



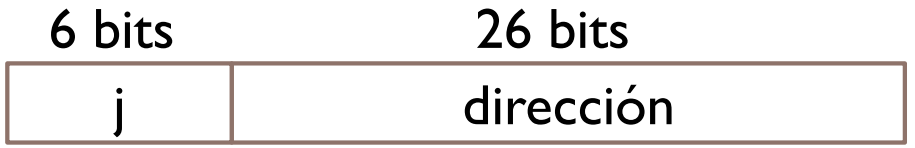
  - 2) b etiqueta2



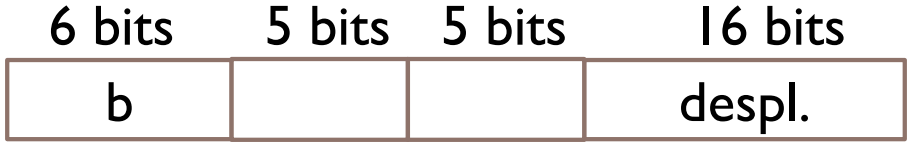
- Donde en la primera se carga la dirección en PC y en la segunda se suma el desplazamiento a PC (siendo este un número en complemento a dos)
- Se pide:
  - Indique razonadamente cual de las dos opciones es más apropiada para bucles pequeños.

Ejercicio (solución)

- Ventajas de la opción 1:
  - El cálculo de la dirección es más rápido, solo cargar
  - El rango de direcciones es mayor, mejor para bucles grandes



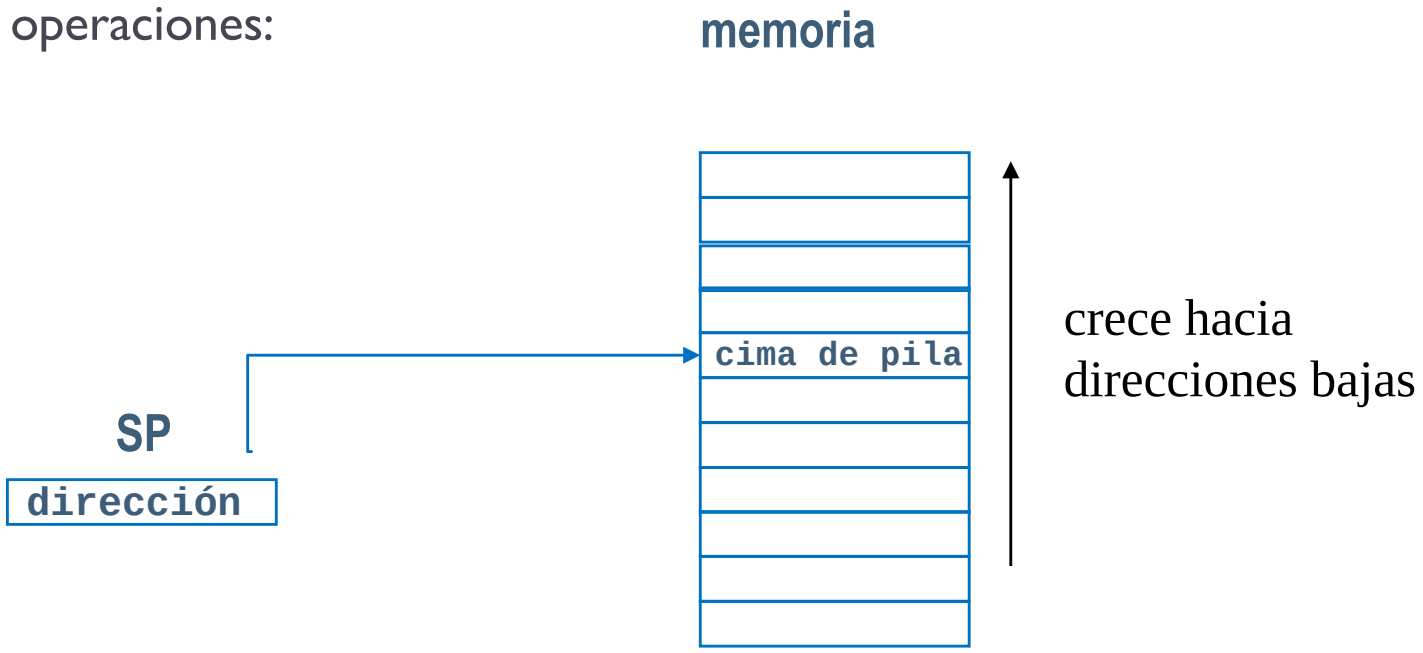
- Ventajas de la opción 2:
  - El rango de direcciones a las que se puede saltar es menor (bucles pequeños)
  - Permite un código reubicable



- La opción 2 sería más apropiada

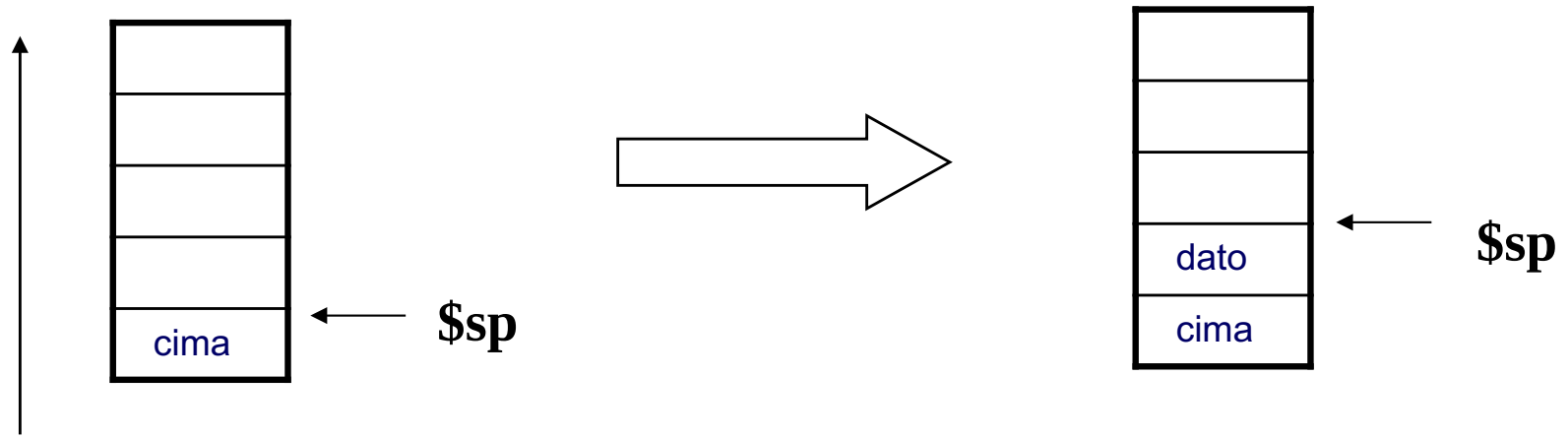
Direccionamiento relativo a pila

- El puntero de pila SP (*Stack Pointer*):
  - Es un registro de 32 bits (4 bytes) en el MIPS
  - 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)

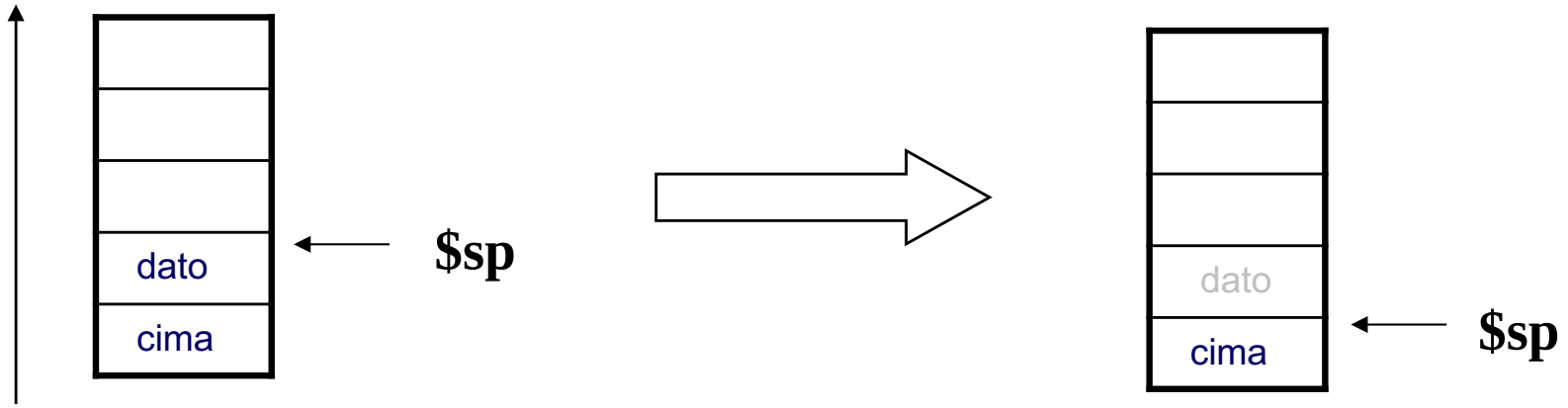


crece hacia direcciones bajas

Operción POP

POP Reg

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



crece hacia direcciones bajas

Direccionamiento de pila en el MIPS

- MIPS 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

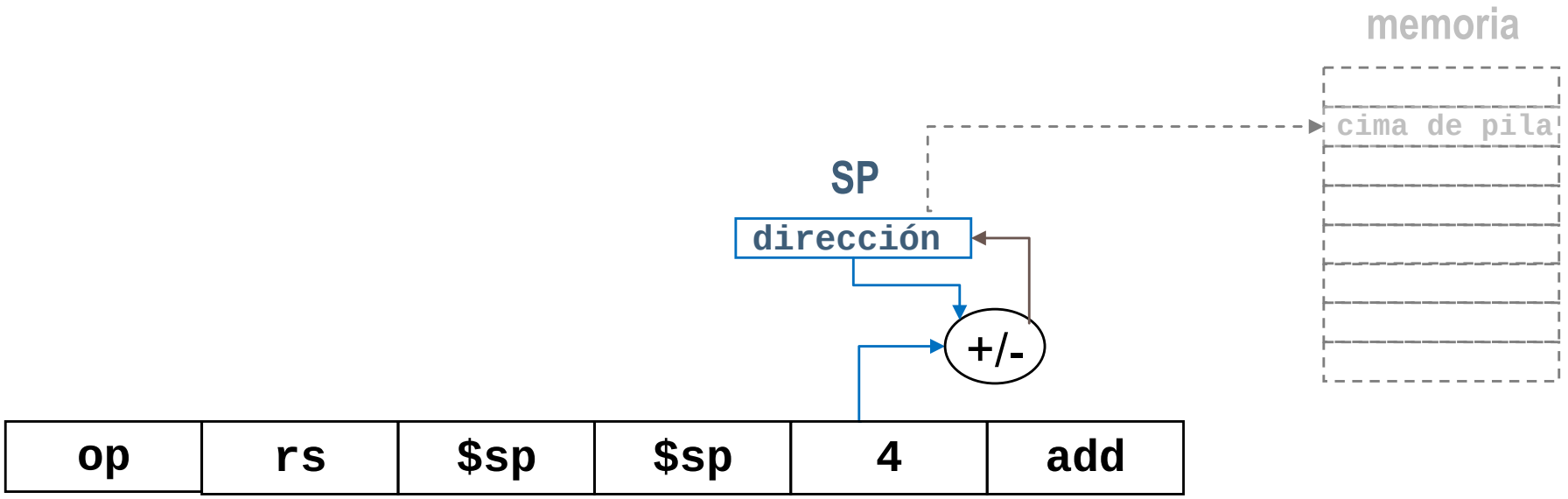
```
addu $sp, $sp, -4
sw   $t0, ($sp)
```

POP $t0

```
lw   $t0, ($sp)
addu $sp, $sp, 4
```

Operación PUSH en MIPS

- Ejemplo: push $a0
  - addu $sp $sp -4 # $SP = $SP - 4
  - sw $a0 ($sp) # memoria[$SP] = $a0



Modos de direccionamiento en MIPS

- Direccionamientos:
  - Inmediato valor
  - De registro $r
  - Directo dir
  - Indirecto de registro ($r)
  - Relativo a registro valor($r)
    - valor puede representar una dirección (registro base)
    - valor puede representar un desplazamiento (registro índice)
  - Relativo a PC beq etiqueta
  - Relativo a pila desplazamiento($sp)

Ejercicio

- Indique el tipo de direccionamiento usado en las siguientes instrucciones MIPS:
  1. li $t1 4
  2. lw $t0 4($a0)
  3. bnez $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. bnez $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

Juego de instrucciones

- Queda definido por:
  - Conjunto de instrucciones
  - Formato de la 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, …

Formato de instrucciones

- Una instrucción máquina es autocontenida e incluye:
  - Código de operación
  - Dirección de los operandos
  - Dirección del resultado
  - Dirección de la siguiente instrucción
  - Tipos de representación de los operandos
- Una instrucción se divide en campos
- Ejemplo de campos en una instrucción del MIPS:

Formato de instrucciones

- Una instrucción normalmente ocupa una palabra pero puede ocupar más en algunos computadores
  - En el caso del MIPS todas las instrucciones ocupan una palabra
- Campo de código:
  - Con n bits se pueden codificar 2^n instrucciones
  - Si se quiere codificar más se utiliza un campo de extensión
  - Ejemplo: en el MIPS las instrucciones aritméticas tienen como código de op = 0. La función concreta se codifica en el campo func.

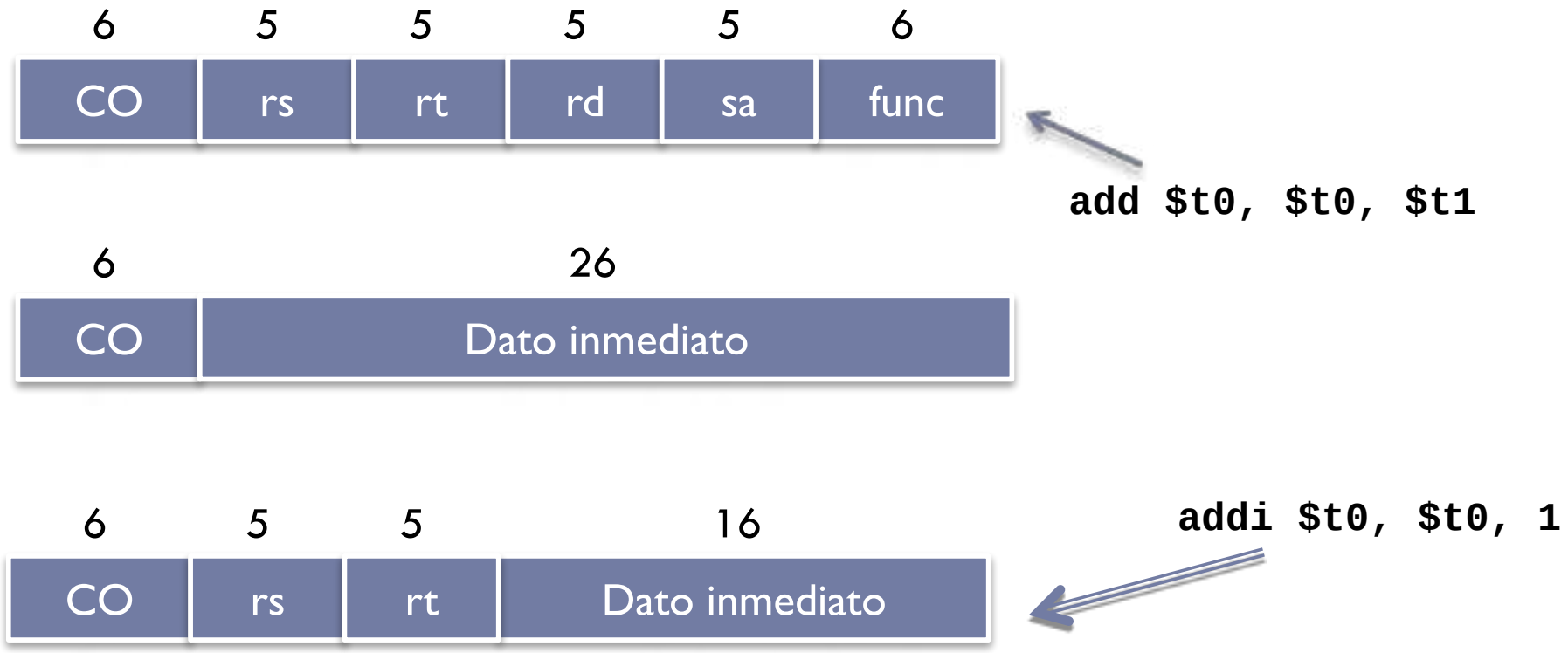
Formato de una instrucción

- Especifica el significado de cada uno de los bits que forma la instrucción.
- Longitud del formato: Número de bits que componen la instrucción.
- La instrucción se divide en campos.
- Normalmente una arquitectura ofrece unos pocos formatos de instrucción.
  - Simplicidad en el diseño de la unidad de control.
- Uso sistemático:
  - Campos del mismo tipo siempre igual longitud.
  - Selección mediante código de operación.
    - Normalmente el primer campo.

Longitud de formato

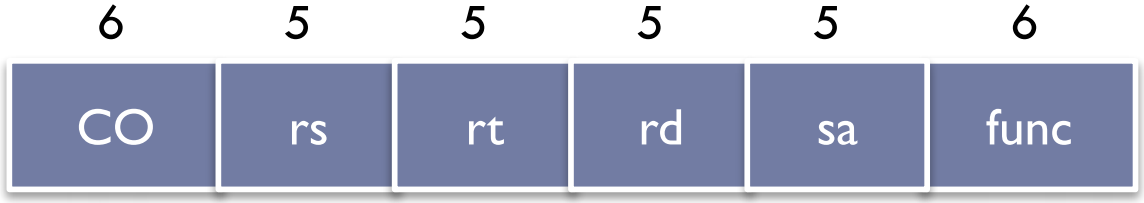
- Alternativas:
  - Longitud única: Todas las instrucciones tienen la misma longitud de formato.
    - MIPS32: 32 bits
    - PowerPC: 32 bits
  - Longitud variable: Distintas instrucciones tienen distinta longitud de formato.
    - ¿Cómo se sabe la longitud de la instrucción? → Cod. Op.
    - IA32 (Procesadores Intel): Número variable de bytes.

Ejemplo: Formato de las instrucciones del MIPS



Ejemplo de formato en el MIPS

- MIPS Instruction:
  - `add $8,$9,$10`
  - Formato a utilizar :



- Representación decimal de cada campo:

0	9	10	8	0	32

- Representación binaria de cada campo:

000000	01001	01010	01000	00000	100000

Ejemplo de formato en el MIPS

- MIPS Instruction:
  - `addi  $21,$22,-50`
  - Formato a utilizar :

Representación decimal de cada campo:

8	22	21	-50

Representación binaria de cada campo

001000	10110	10101	1111111111001110

¿Cómo utilizar addi con un valor de 32 bits?

- ¿Qué ocurre si se utiliza desde el ensamblador?
  - `addi  $t0,$t0, 0xABABCDCD`
  - El valor inmediato es de 32 bits. Esta instrucción no se puede codificar en una palabra de 32 bits.

¿Cómo utilizar addi con un valor de 32 bits?

- ¿Qué ocurre si se utiliza desde el ensamblador?
  - `addi  $t0,$t0, 0xABABCDCD`
  - El valor inmediato es de 32 bits. Esta instrucción no se puede codificar en una palabra de 32 bits.
- Solución:
  - Desde el ensamblador se puede utilizar, pero al final se traduce en:

```
lui    $at, 0xABAB
ori    $at, $at, 0xCDCD
add    $t0, $t0, $at
```

  - El registro `$at` está reservado para el ensamblador por convenio

Ejercicio

- ¿Cómo sabe la unidad de control el formato de la instrucción que está ejecutando?
- ¿Cómo sabe la unidad de control el número de operandos de una instrucción?
- ¿Cómo sabe la unidad de control el formato de cada operación?

Código de operación

- Tamaño fijo:
  - n bits ➔ 2^n códigos de operación.
  - m códigos de operación ➔ $\log_2 m$ bits.
- Campos de extensión
  - MIPS (instrucciones aritméticas-lógicas)
  - Op = 0; la instrucción está codificada en `func`

Tipo R
aritméticas

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

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 R1 R2 R3, donde R1, R2 y R3 son registros.

Ejercicio (solución)

palabra -> 16 bits
60 instrucciones
8 registros (en BR)
ADDx R1(reg.), R2(reg.), R3(reg.)

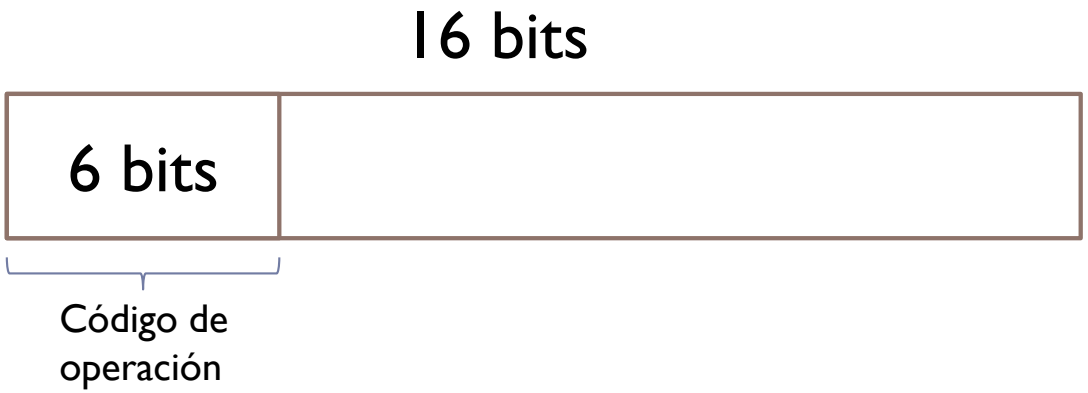
- Palabra de 16 bits define el tamaño de la instrucción

16 bits

Ejercicio (solución)

palabra -> 16 bits
60 instrucciones
8 registros (en BR)
ADDx R1(reg.), R2(reg.), R3(reg.)

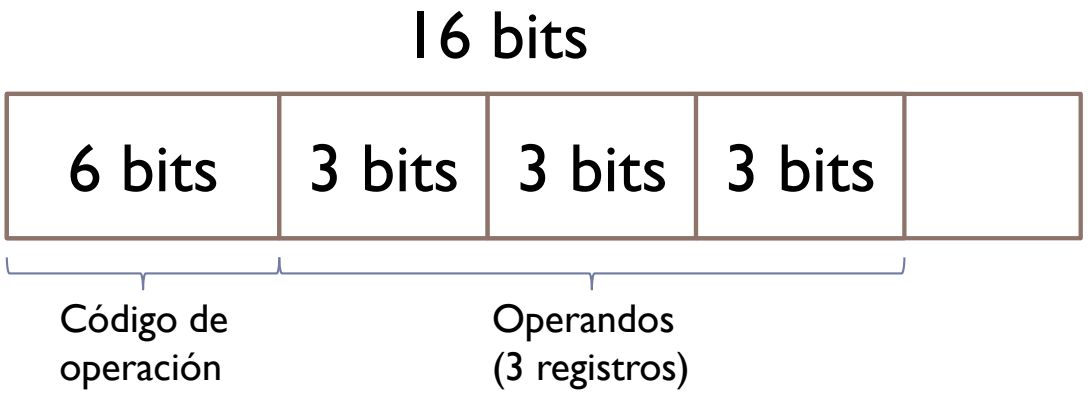
- Para 60 instrucciones se necesitan 6 bits (mínimo)



Ejercicio (solución)

palabra -> 16 bits
60 instrucciones
8 registros (en BR)
ADDx R1(reg.), R2(reg.), R3(reg.)

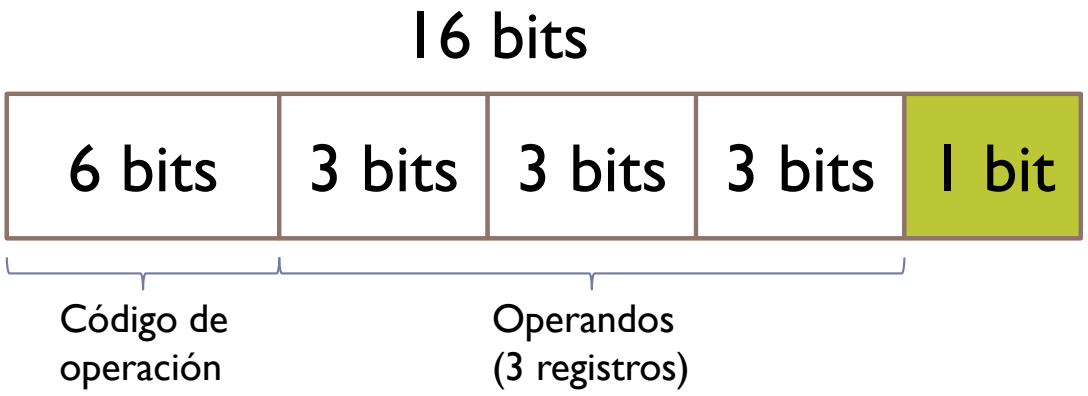
- Para 8 registros se necesitan 3 bits (mínimo)



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



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 MIPS 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.

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 MIPS 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.

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, …

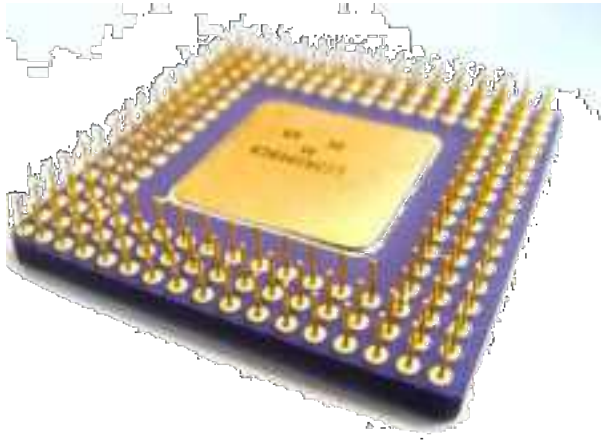
CISC

- *Complex Instruction Set Computer*

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

CISC vs RISC

- Observación:
  - 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



RISC

- *Reduced Instruction Set Computer*

- Juegos de instrucciones reducidos
- 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

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, …

Modelo de ejecución

- Una máquina tiene un modelo de ejecución asociado.
  - Modelo de ejecución indica el número de direcciones y tipo de operandos que se pueden especificar en una instrucción.

- Modelos de ejecución:
  - 0 direcciones → Pila
  - 1 dirección → Registro acumulador
  - 2 direcciones → Registros, Registro-Memoria y Memoria-Memoria
  - 3 direcciones → Registros, Registro-Memoria y Memoria-Memoria

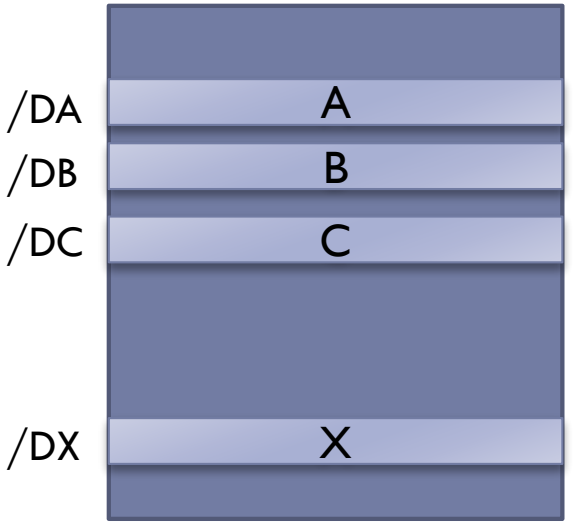
Modelo de 3 direcciones

- Registro-Registro:
  - Los 3 operandos son registros.
  - Requiere operaciones de carga/almacenamiento.
  - **ADD .R0, .R1, .R2**
- Memoria-Memoria:
  - Los 3 operandos son direcciones de memoria.
  - **ADD /DIR1, /DIR2, /DIR3**
- Registro-Memoria:
  - Híbrido.
  - **ADD .R0, /DIR1, /DIR2**
  - **ADD .R0, .R1, /DIR1**

Ejercicio

- Sea la siguiente expresión matemática:
  - **X = A + B * C**

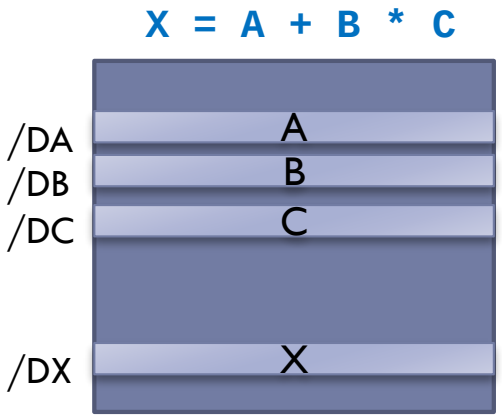
Donde los operandos están en memoria tal y como se describe en la figura:



Para los modelos R-R y M-M, indique:

- El número de instrucciones
- Accesos a memoria
- Accesos a registros

Ejercicio (solución)



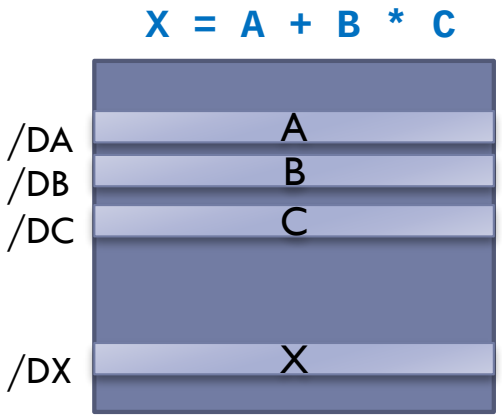
- Memoria-Memoria:

```
MUL /DX, /DB, /DC
ADD /DX, /DX, /DA
```

- Registro-Registro:

```
LOAD  R0, /DB
LOAD  R1, /DC
MUL   R0, R0, R1
LOAD  R2, /DA
ADD   R0, R0, R2
STORE R0, /DX
```

Ejercicio (solución)



- Memoria-Memoria:
  - 2 instrucciones
  - 6 accesos a memoria
  - 0 accesos a registros

```
MUL /DX, /DB, /DC
ADD /DX, /DX, /DA
```

- Registro-Registro:
  - 6 instrucciones
  - 4 accesos a memoria
  - 10 accesos a registros

```
LOAD  R0, /DB
LOAD  R1, /DC
MUL   R0, R0, R1
LOAD  R2, /DA
ADD   R0, R0, R2
STORE R0, /DX
```

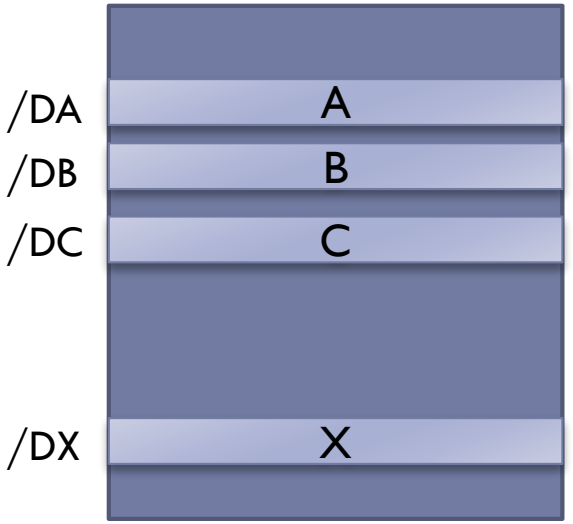
Modelo de 2 direcciones

- Registro-Registro:
  - Los 2 operandos son registros.
  - Requiere operaciones de carga/almacenamiento.
  - **ADD R0, R1 (R0 <- R0 + R1)**
- Memoria-Memoria:
  - Los 2 operandos son direcciones de memoria.
  - **ADD /DIR1, /DIR2 (MP[DIR1] <- MP[DIR1] + MP[DIR2])**
- Registro-Memoria:
  - Híbrido.
  - **ADD R0, /DIR1 (R0 <- R0 + MP[DIR1])**

Ejercicio

- Sea la siguiente expresión matemática:
  - **X = A + B * C**

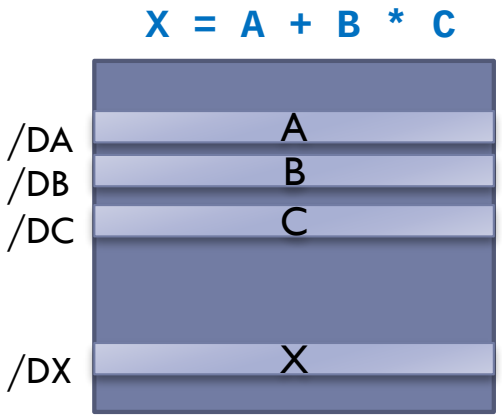
Donde los operandos están en memoria tal y como se describe en la figura:



Para los modelos R-R y M-M, indique:

- El número de instrucciones
- Accesos a memoria
- Accesos a registros

Ejercicio (solución)



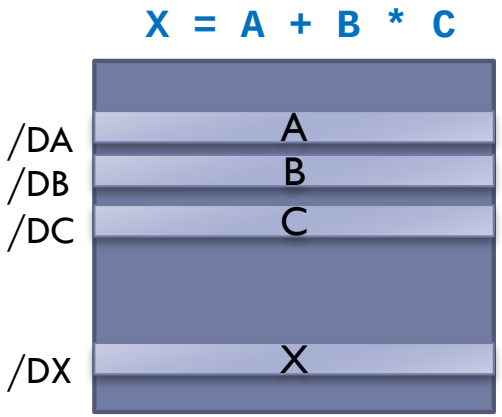
- Memoria-Memoria:

```
MOVE   /DX, /DB
MUL    /DX, /DC
ADD    /DX, /DA
```

- Registro-Registro:

```
LOAD   R0, /DB
LOAD   R1, /DC
MUL    R0, R1
LOAD   R2, /DA
ADD    R0, R2
STORE  R0, /DX
```

Ejercicio (solución)



- Memoria-Memoria:
  - 3 instrucciones
  - 6 accesos a memoria
  - 0 accesos a registros

```
MOVE   /DX, /DB
MUL    /DX, /DC
ADD    /DX, /DA
```

- Registro-Registro:
  - 6 instrucciones
  - 4 accesos a memoria
  - 8 accesos a registros

```
LOAD   R0, /DB
LOAD   R1, /DC
MUL    R0, R1
LOAD   R2, /DA
ADD    R0, R2
STORE  R0, /DX
```

Modelo de 1 dirección

- Todas las operaciones utilizan un operando implícito:
  - Registro acumulador
  - **ADD R1 (AC <- AC + R1)**

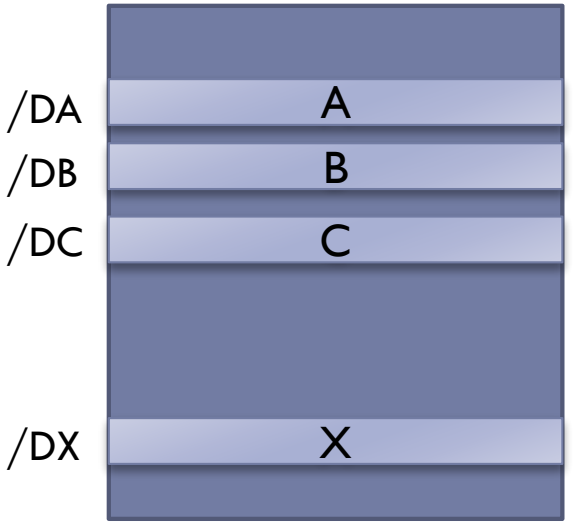
- Operaciones de carga y almacenamiento siempre sobre el acumulador.

- Posibilidad de movimiento entre el registro acumulador y otros registros

Ejercicio

- Sea la siguiente expresión matemática:
  - **X = A + B * C**

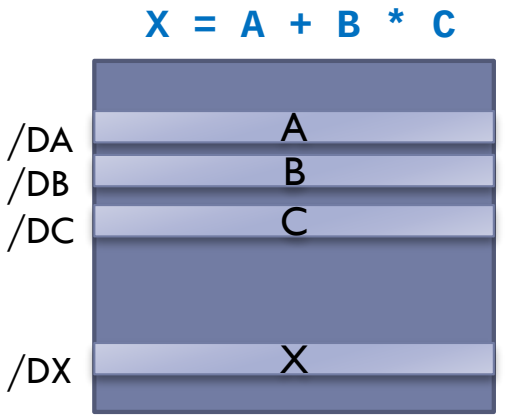
Donde los operandos están en memoria tal y como se describe en la figura:



Para el modelo de 1 dirección, indique:

- El número de instrucciones
- Accesos a memoria
- Accesos a registros

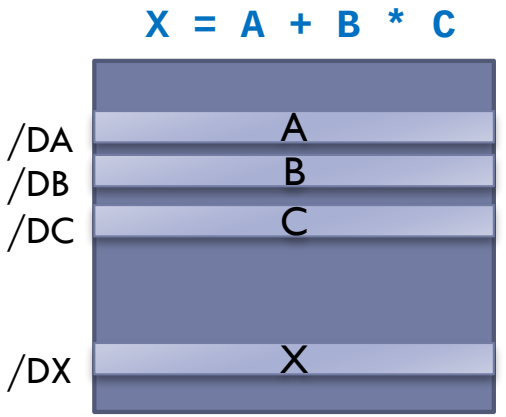
Ejercicio (solución)



- Modelo de 1 sola dirección:

```
LOAD   /DB
MUL    /DC
ADD    /DA
STORE  /DX
```

Ejercicio (solución)



- Modelo de 1 sola dirección:
  - 4 instrucciones
  - 4 accesos a memoria
  - 0 accesos a registros

```
LOAD   /DB
MUL    /DC
ADD    /DA
STORE  /DX
```

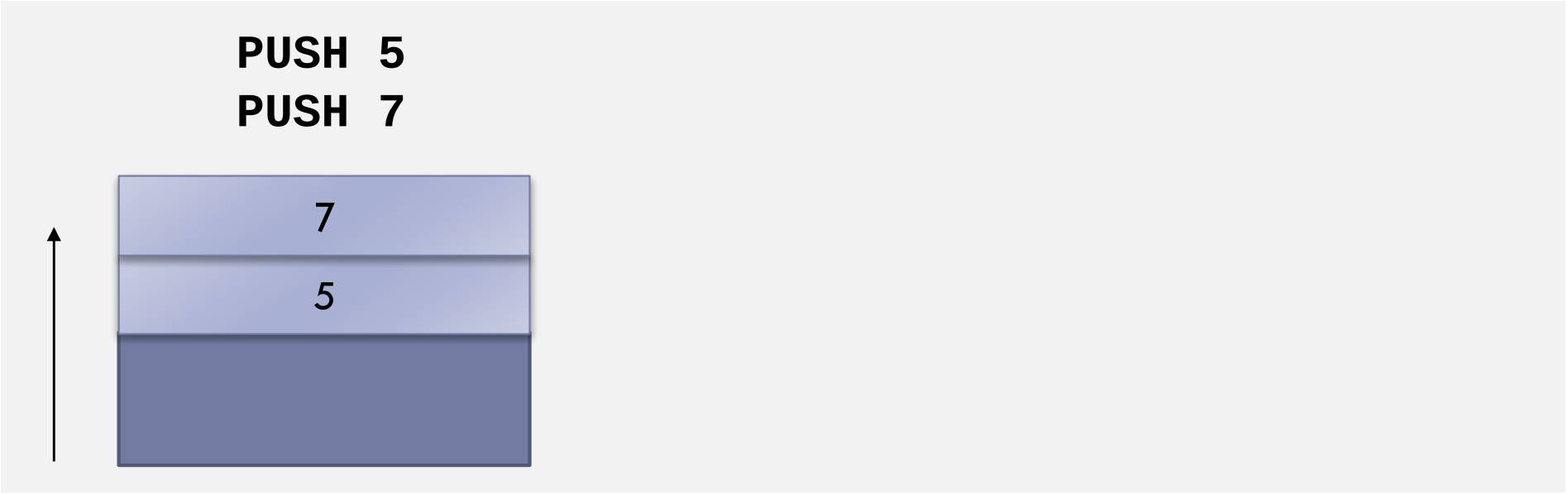
Modelo de 0 direcciones

- Todas las operaciones referidas a la pila:
  - Los operandos están en la cima de la pila.
    - Al hacer la operación se retiran de la pila.
  - El resultado se coloca en la cima de la pila.
  - **ADD**

- Dos operaciones especiales:
  - PUSH
  - POP

Ejemplo

```
push 5
push 7
```



Ejemplo

```
push 5
push 7
add
```

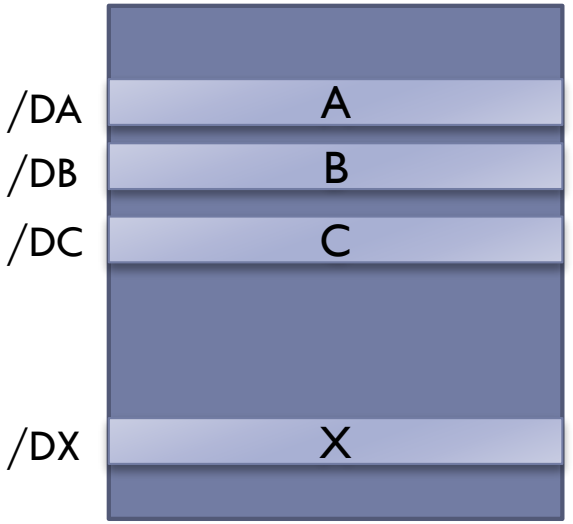
Ejemplo

```
push 5
push 7
add
pop /dx
```

Ejercicio

- Sea la siguiente expresión matemática:
  - **X = A + B * C**

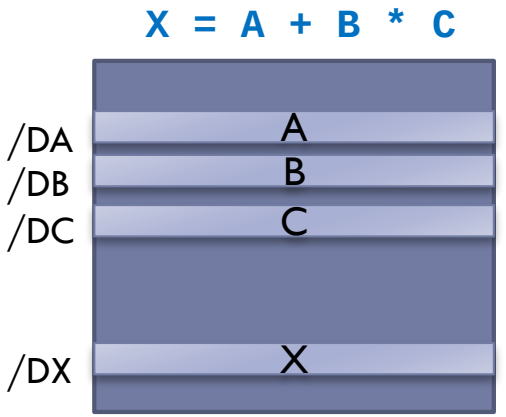
Donde los operandos están en memoria tal y como se describe en la figura:



Para el modelo de 0 dirección, indique:

- El número de instrucciones
- Accesos a memoria
- Accesos a registros

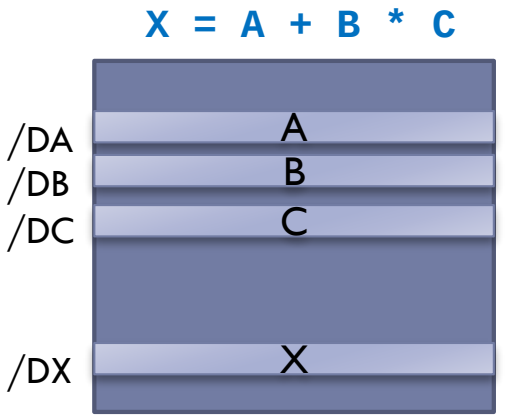
Ejercicio (solución)



- Modelo de 0 direcciones:

```
PUSH /DB
PUSH /DC
MUL
PUSH /DA
ADD
POP  /DX
```

Ejercicio (solución)



- Modelo de 0 direcciones:
  - 6 instrucciones
  - 4 accesos a memoria (datos)
  - 10 accesos a memoria (pila)
  - 0 accesos a registros

```
PUSH /DB
PUSH /DC
MUL
PUSH /DA
ADD
POP  /DX
```