

# MIPS

## Microprocessor without Interlocked Pipeline Stages



Surgió a comienzos de los 80 en Stanford.  
Sintetiza las principales ideas de RISC.  
Arquitectura eficiente y simple.

7/08/17



Guillermo Aguirre

1

## Procesador MIPS - Registros

32 registros de 32 bits de propósito general (GPR).

32 registros de 32 bits de punto flotante (FPR).

R0 siempre tiene el valor 0.

Instrucciones diferenciadas para GPR y FPR.

7/08/17



Guillermo Aguirre

2

# Convenciones de registros en MIPS

Name	Register number	Usage	Preserved on call?
\$zero	0	The constant value 0	n.a.
\$v0-\$v1	2-3	Values for results and expression evaluation	no
\$a0-\$a3	4-7	Arguments	no
\$t0-\$t7	8-15	Temporaries	no
\$s0-\$s7	16-23	Saved	yes
\$t8-\$t9	24-25	More temporaries	no
\$gp	28	Global pointer	yes
\$sp	29	Stack pointer	yes
\$fp	30	Frame pointer	yes
\$ra	31	Return address	yes

## MIPS - tipos de datos

Bytes de 8 bits.

Half word - Media palabra 16 bits.

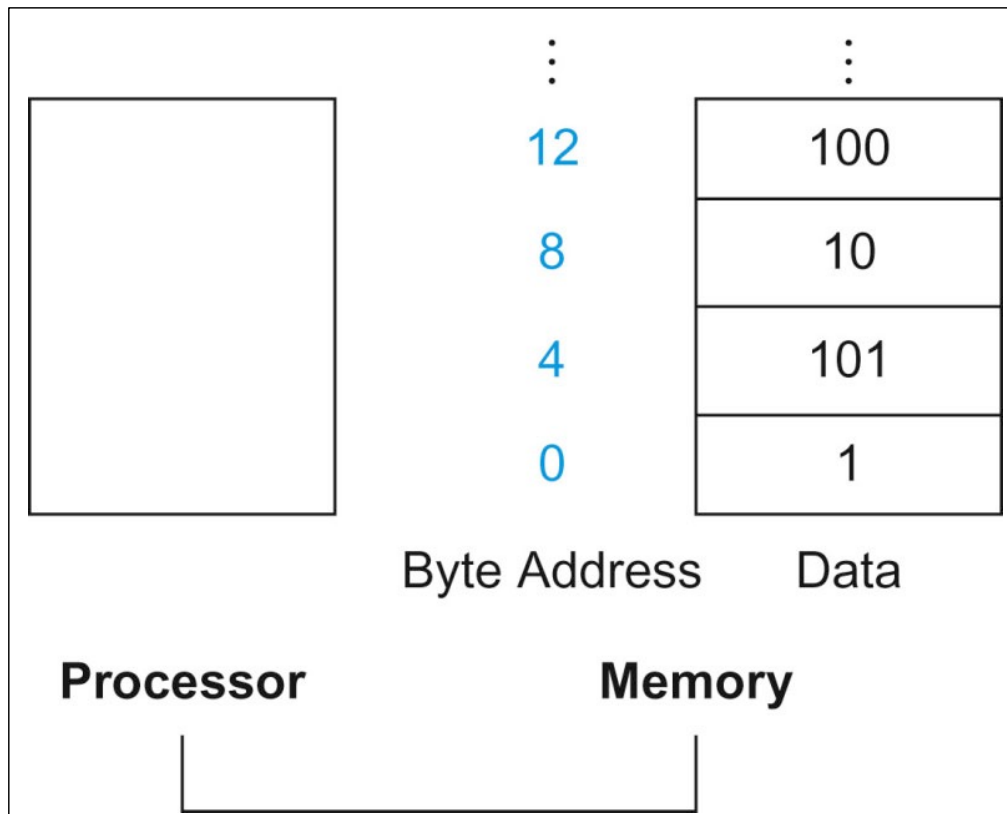
Word - Palabra de 32 bits.

Double Word - Doble palabra de 64 bits.

Simple precisión 32 bits.  
Doble precisión 64 bits. } punto flotante

Los bytes, half-word y word son cargados en GPR's, completando con 0 o el signo.

# Direcciones de memoria



7/08/17

 Guillermo Aguirre

5

# Formato de instrucciones

I-type instruction



Encodes: Loads and stores of bytes, half words, words, double words. All immediates ( $rt \leftarrow rs \text{ op immediate}$ )

Conditional branch instructions (rs is register, rd unused)

En el código de operación se codifica el tipo de dato

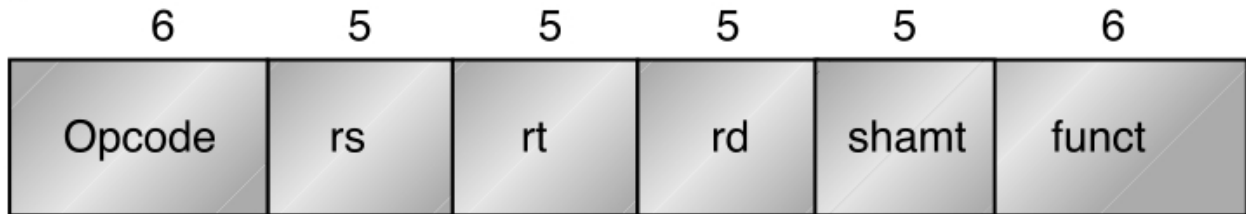
7/08/17

 Guillermo Aguirre

6

# Formato de instrucciones

R-type instruction



Register-register ALU operations:  $rd \leftarrow rs \text{ funct } rt$   
Function encodes the data path operation: Add, Sub, . . .  
Read/write special registers and moves

7/08/17

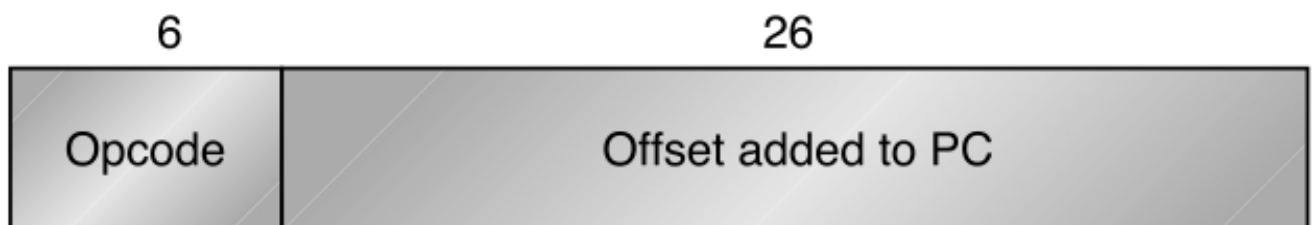


Guillermo Aguirre

7

# Formato de instrucciones

J-type instruction



Jump and jump and link  
Trap and return from exception

7/08/17



Guillermo Aguirre

8

# Los 3 formatos de instrucciones

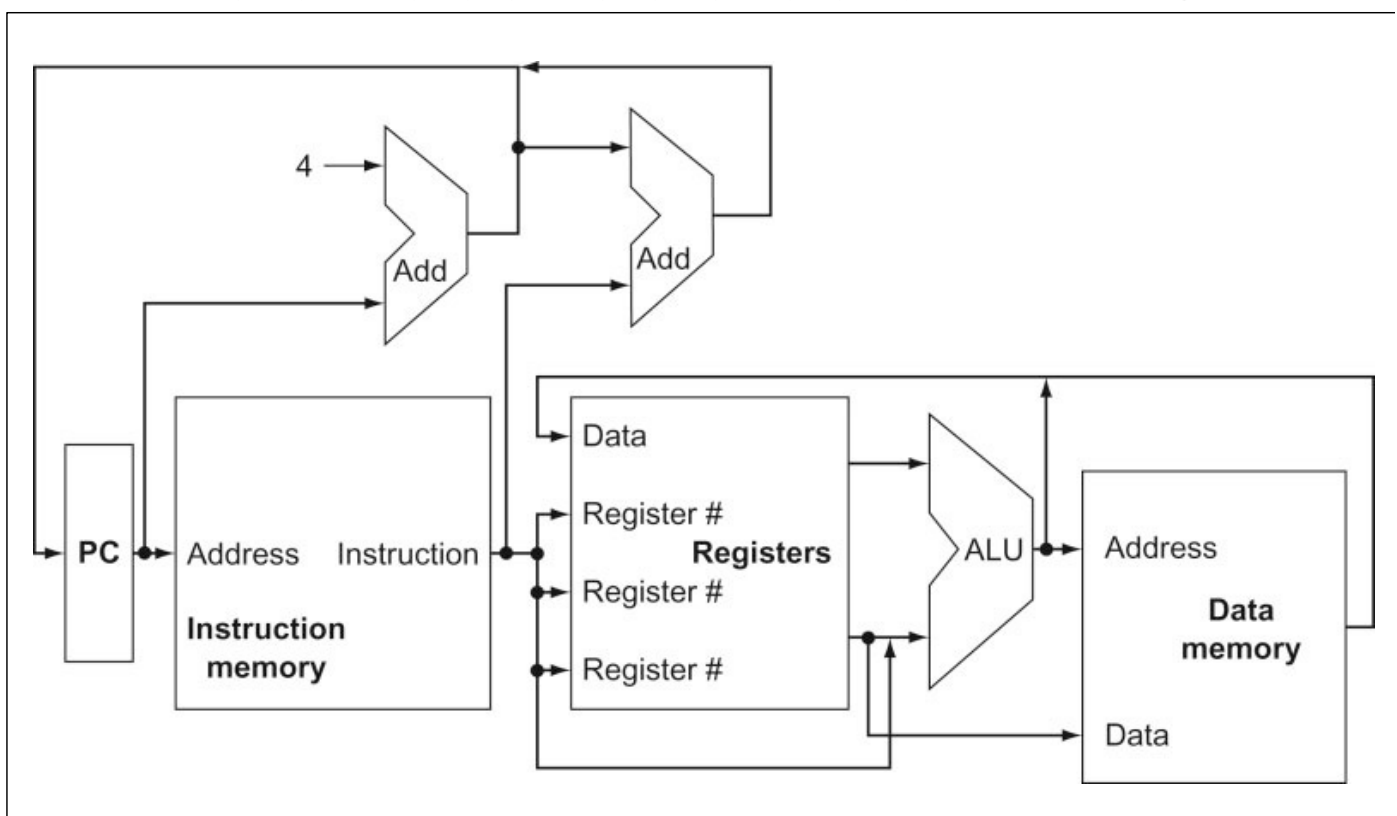
Name	Fields					
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R-format	op	rs	rt	rd	shamt	funct
I-format	op	rs	rt	address/immediate		
J-format	op	target address				

7/08/17

 Guillermo Aguirre

9

## Visión abstracta de un MIPS

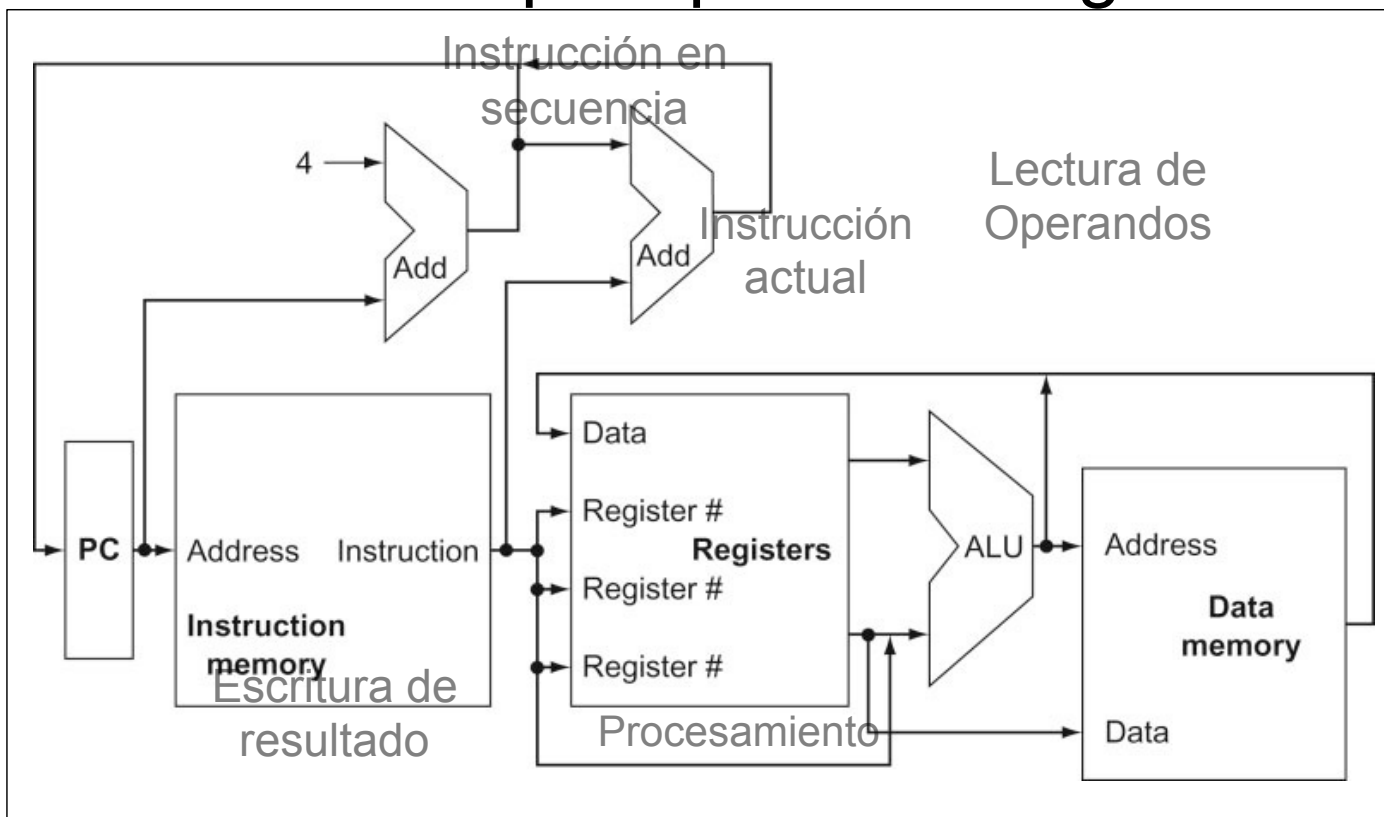


7/08/17

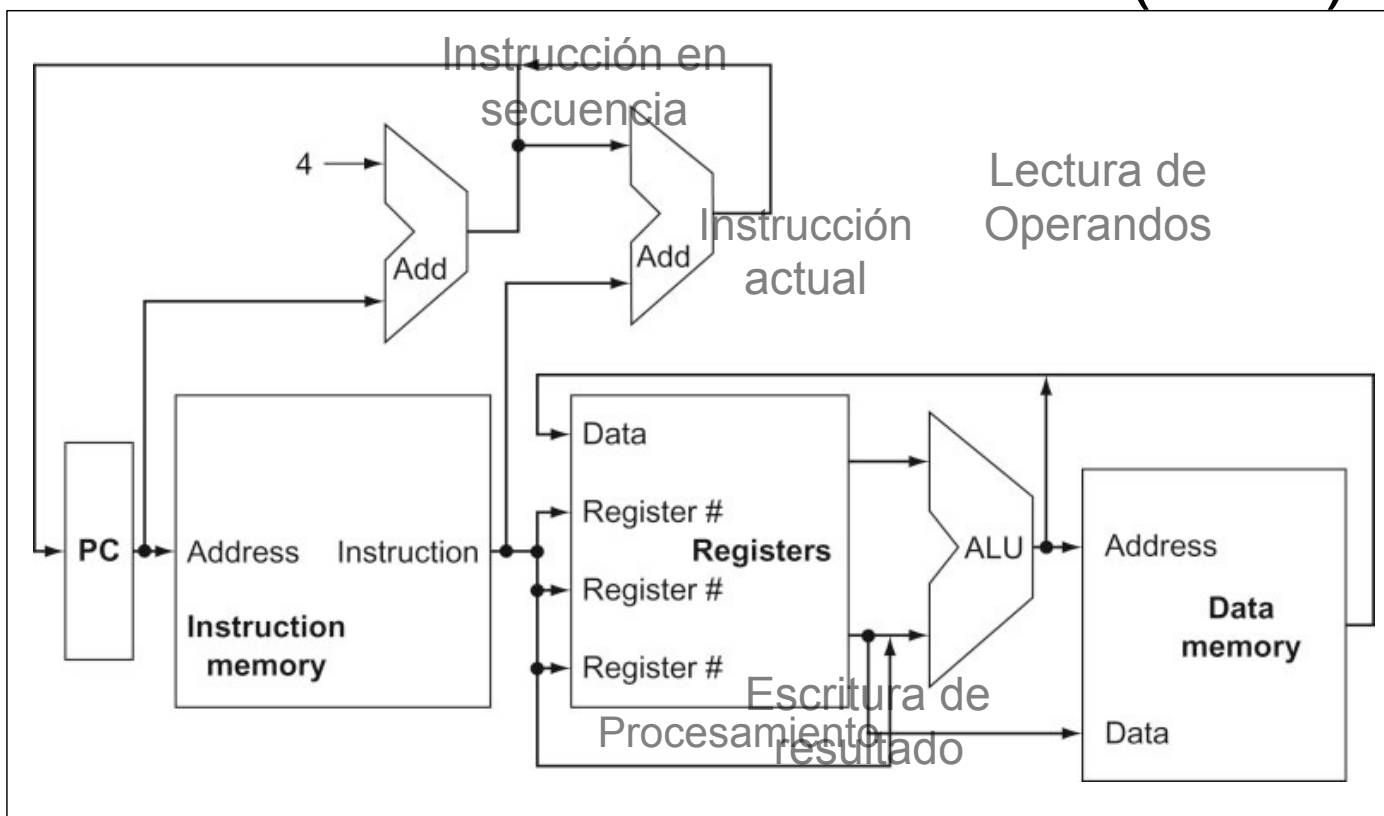
 Guillermo Aguirre

10

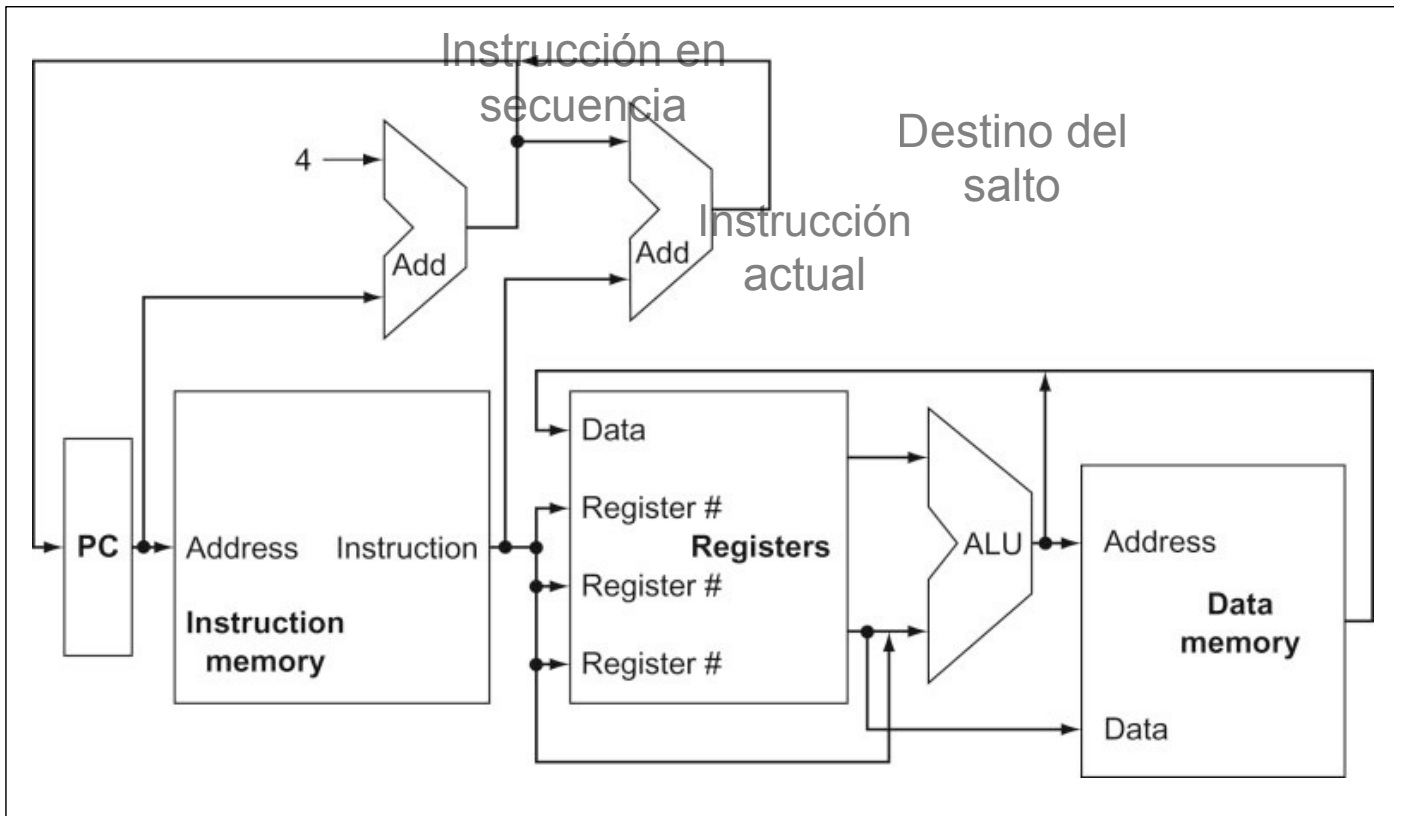
# Instrucción que opera con registros



# Instrucción de almacenamiento(store)



# Instrucción de salto Incondicional

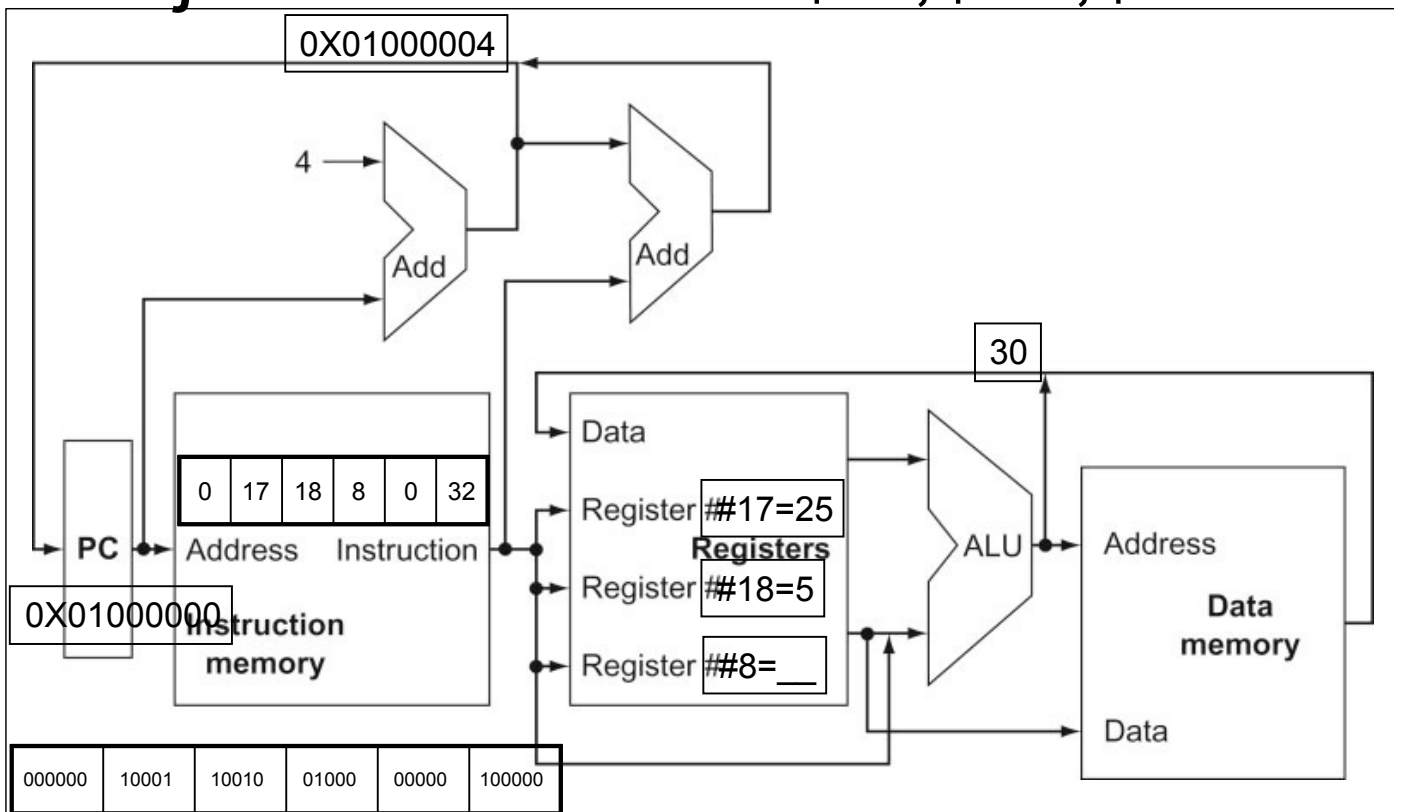


7/08/17

Guillermo Aguirre

13

# Ejecución de add \$t0,\$s1,\$s2



7/08/17

Guillermo Aguirre

14

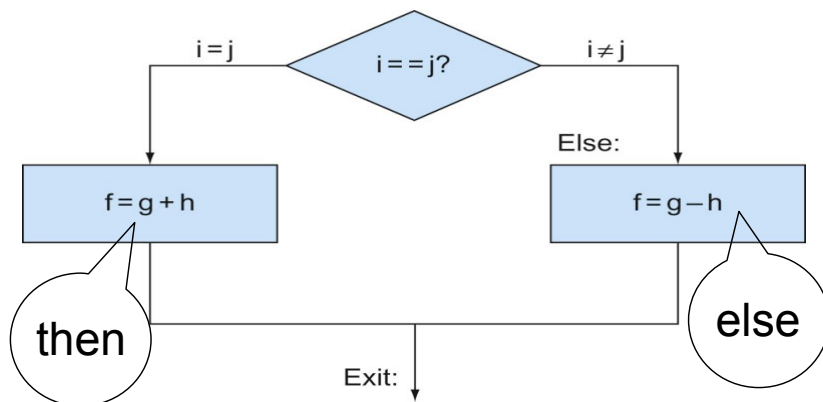
# Instrucciones de salto condicional

beq reg1, reg2, rótulo1      (Branch if equal)  
 ir a la sentencia en rótulo1, si (reg1)=(reg2)

bne reg1, reg2, rótulo1      (Branch if not equal)  
 ir a la sentencia en rótulo1, si (reg1)≠(reg2)

## Estructuras de control

if(i==j) f=g+h;  
 else f=g-h;



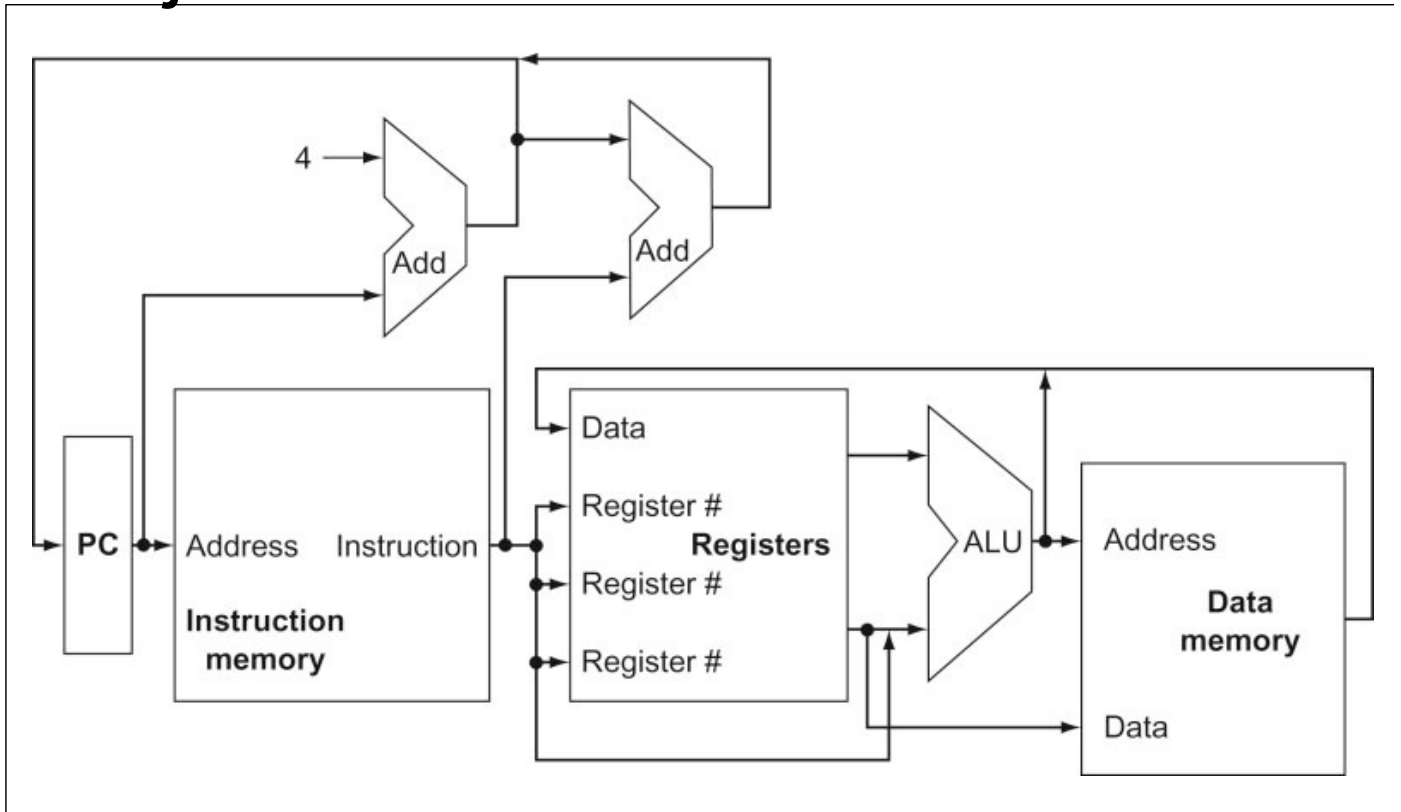
bne \$s3,\$s4,0x2	00	16930002
add \$s0,\$s1,\$s2	04	02328020
j 0x1	08	08000001
sub \$s0,\$s1,\$s2	0C	02328022
add \$t8,\$t8,\$t9	10	0319C020

lenguaje de  
ensamble

lenguaje  
de máquina



# Ejecución de salto condicional



7/08/17

 Guillermo Aguirre

17

## MIPS - Ejemplo de Operaciones

`lw $s0,40($t0)`      load word

$\text{Regs}[16] \leftarrow_{32} \text{Mem}[40 + \text{Regs}[8]]$

`sll $a0,$t2,5`      Shift izquierda lógico

$\text{Regs}[4] \leftarrow_{32} \text{Regs}[10] \ll 5$

7/08/17

 Guillermo Aguirre

18

# Operaciones de control de secuencia

Instrucciones de comparación. `slt $t1,$t2,$t3`

Si ( $\text{Regs}[t2] < \text{Regs}[t3]$ )

$\text{Regs}[t1] \leftarrow 1$  Else  $\text{Regs}[t1] \leftarrow 0$

jump	jump and link	jump register
$\text{PC} \leftarrow \text{Off} \ll 2 : \text{PC}$	$\$ra = \text{PC} + 4$ $\text{PC} \leftarrow \text{Off} \ll 2$	$\text{PC} \leftarrow \text{Regs}[rs]$

26 bits

Los Branches son condicionales. Testean dos registros fuente.  $\text{PC} \leftarrow \text{Inm} \ll 2 + \text{PC}$

7/08/17



Guillermo Aguirre

16 bits

19

## Invocación a procedimientos

Poner los parámetros donde el procedimiento pueda accederlos

Transferir el control al procedimiento.

Adquirir los recursos de memoria que necesita el procedimiento

Realizar la tarea deseada.

Poner el resultado donde el programa pueda accederlo.

Retornar el control al punto original.

### Convenciones en MIPS

$\$a0-\$a3$  Registros para cuatro argumentos

$\$v0, \$v1$  Registros para retornar valores

$\$ra$  Registro para la dirección de retorno. `jal $ra`

7/08/17



Guillermo Aguirre

20

# Uso de la pila (lifo)

Resguardo de registros del llamador.

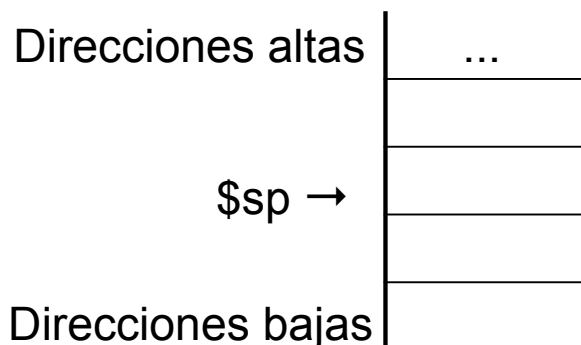
El puntero al tope de la pila es \$sp (29)

La pila crece hacia las direcciones bajas.

\$t0-\$t9 no preservados por el procedimiento

\$s0-\$s7, \$ra, \$sp son preservados.

Anidamiento de llamadas y variables locales



7/08/17



Guillermo Aguirre

21

hoja:

```
addi $sp,$sp,-12
sw   $t1,8($sp)
sw   $t0,4($sp)
sw   $s0,0($sp)
```

} Resguarda registros

```
add $t0,$a0,$a1
add $t1,$a2,$a3
sub $s0,$t0,$t1
```

} Operación

```
add $v0,$zero,$s0
```

} Valor de retorno

```
lw   $s0,0($sp)
lw   $t0,4($sp)
lw   $t1,8($sp)
addi $sp,$sp,12
```

} Recupera registros

```
jr   $ra
```

} Retorno al llamador

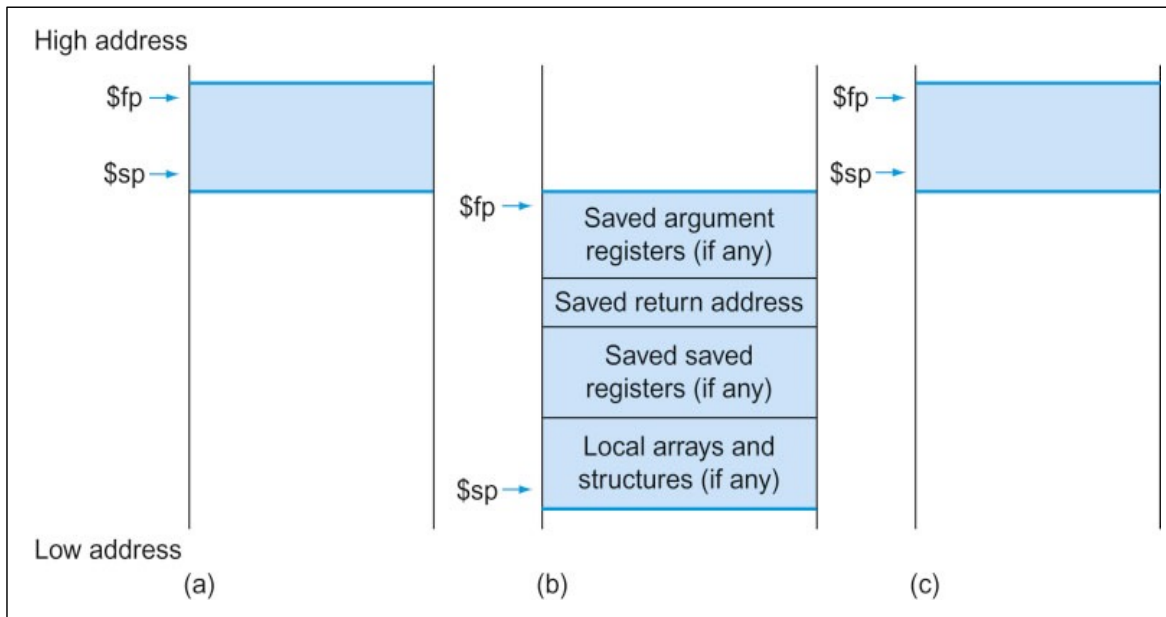
7/08/17



Guillermo Aguirre

22

# Registro de activación



Puntero al marco \$fp (30)

Base estable para las referencias locales

7/08/17



Guillermo Aguirre

23

fact:

```
addi $sp,$sp,-8
sw    $ra,4($sp)
sw    $a0,0($sp)
```

Resguarda dir de retorno y argumento

```
slti    $t0,$a0,1
beq    $t0,$zero,L1
```

Salta si es mayor que 1

```
addi $v0,$zero,1
addi $sp,$sp,8
jr    $ra
```

Retorna al programa principal

L1:

```
addi $a0,$a0,-1
jal    fact
```

Nueva invocación

```
lw    $a0,0($sp)
lw    $ra,4($sp)
addi $sp,$sp,8
```

Desde el stack recupera argumento y dir de ret.

```
mul   $v0,$a0,$v0
```

Multiplica

```
jr    $ra
```

Retorno recursivo

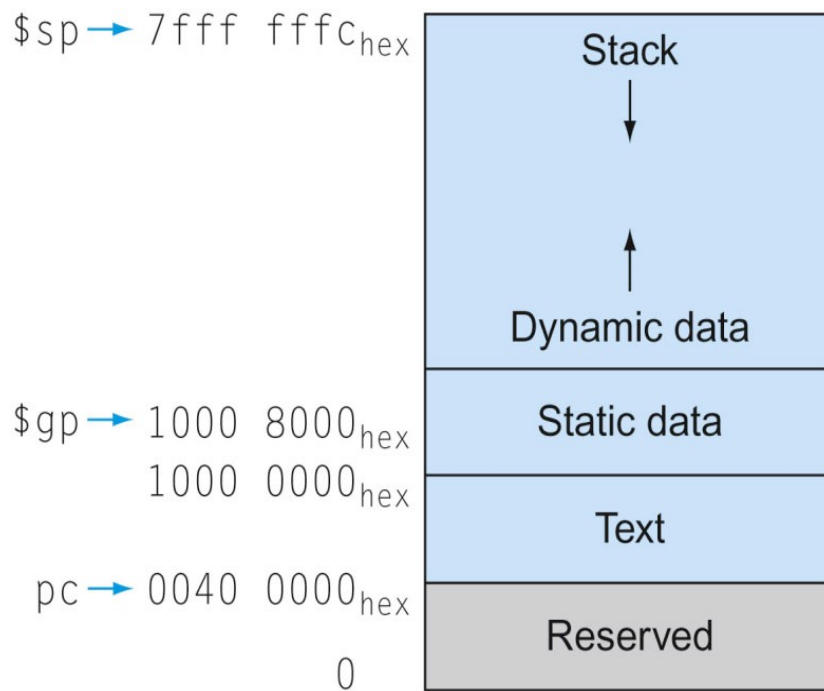
7/08/17



Guillermo Aguirre

24

# Memoria para código y datos



7/08/17



Guillermo Aguirre

25

## MIPS - modos de direccionamiento

Inmediato. Campo de 16 bits.

`addi $t2,$t3,64`

Desplazamiento. Campo de 16 bits.

`load $s1,8($s3)`

Direcciona a byte. Tamaño de dirección 32 bits.

Transferencia desde y hacia memoria con load y store.

Acceso a datos alineados.

7/08/17



Guillermo Aguirre

26

# Modos de direccionamiento

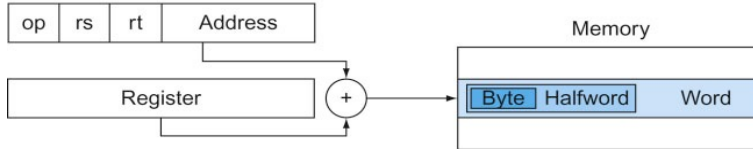
1. Immediate addressing



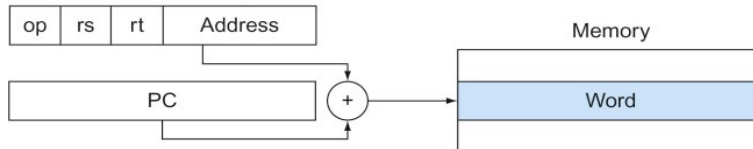
2. Register addressing



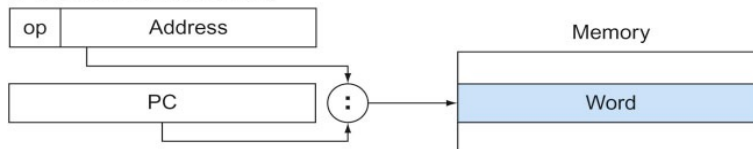
3. Base addressing



4. PC-relative addressing



5. Pseudodirect addressing



7/08/17

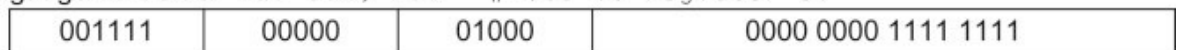


Guillermo Aguirre

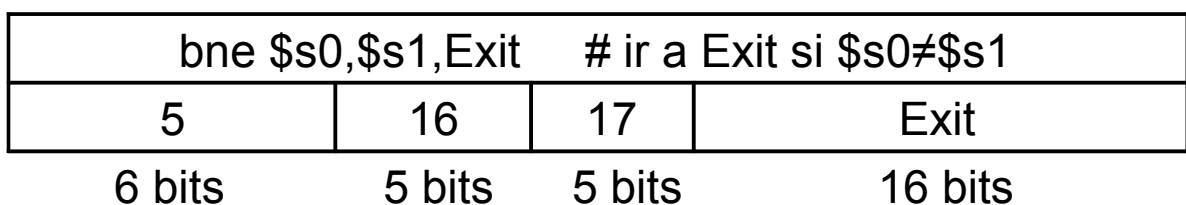
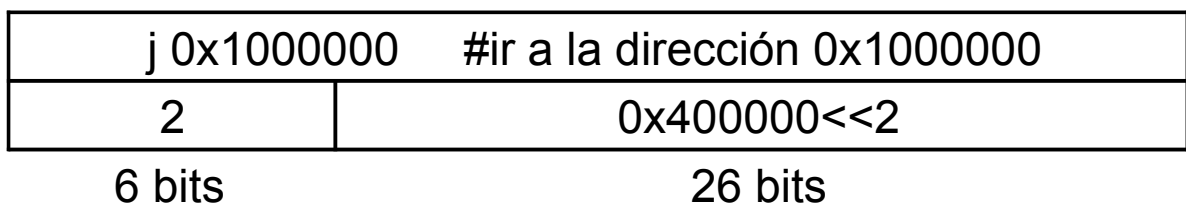
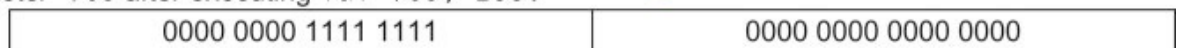
27

## Direcciones e inmediatos de 32 bits

The machine language version of `lui $t0, 255` # \$t0 is register 8:



Contents of register `$t0` after executing `lui $t0, 255`:



7/08/17



Guillermo Aguirre

28

Pseudoinstrucción	Traducción
<b>mov \$rt, \$rs</b>	addi \$rt, \$rs, 0
<b>li \$rs, small</b>	addi \$rt, \$rs, small
<b>li \$rs, big</b>	lui \$rs, upper( big ) ori \$rs, \$rs, lower( big )
<b>la \$rs, big</b>	lui \$rs, upper( big ) ori \$rs, \$rs, lower( big )
<b>lw \$rt, big(\$rs)</b>	lui \$t0, upper( big ) ori \$t0, \$t0, lower( big ) add \$t0, \$rs, \$t0 lw \$rt, 0(\$t0)

7/08/17



Guillermo Aguirre

29

#Recorre un arreglo de palabras hasta  
#que encuentra una distinta a cero

```

main:      .text
           la $s6,arr
           add $s3,$zero,$zero
           add $s5,$zero,$zero
loop:     sll $t1,$s3,2
           add $t1,$t1,$s6
           lw $t0,0($t1)
           bne $t0,$s5,exit
           addi $s3,$s3,1
           j loop
exit:     jr $ra
           .data
arr:      .word 0,0,1

```

base del arreglo

índice \*4

dirección del elemento

carga la palabra

incremento del índice

```

while(arr[i]==k)
    i++;

```

7/08/17



Guillermo Aguirre

30

# ¿Qué vimos?

## Características de los procesadores RISC.

Instrucciones simples.

Direccionamientos sencillos.

Ejecución eficiente.

## Procesador MIPS

Registros y tipos de datos.

Modos de direccionamientos

Formatos de instrucciones.

Ejemplos de operaciones.

