

MIPS

Microprocessor without Interlocked Pipeline Stages



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

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.

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

6/08/18



Guillermo Aguirre

3

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.

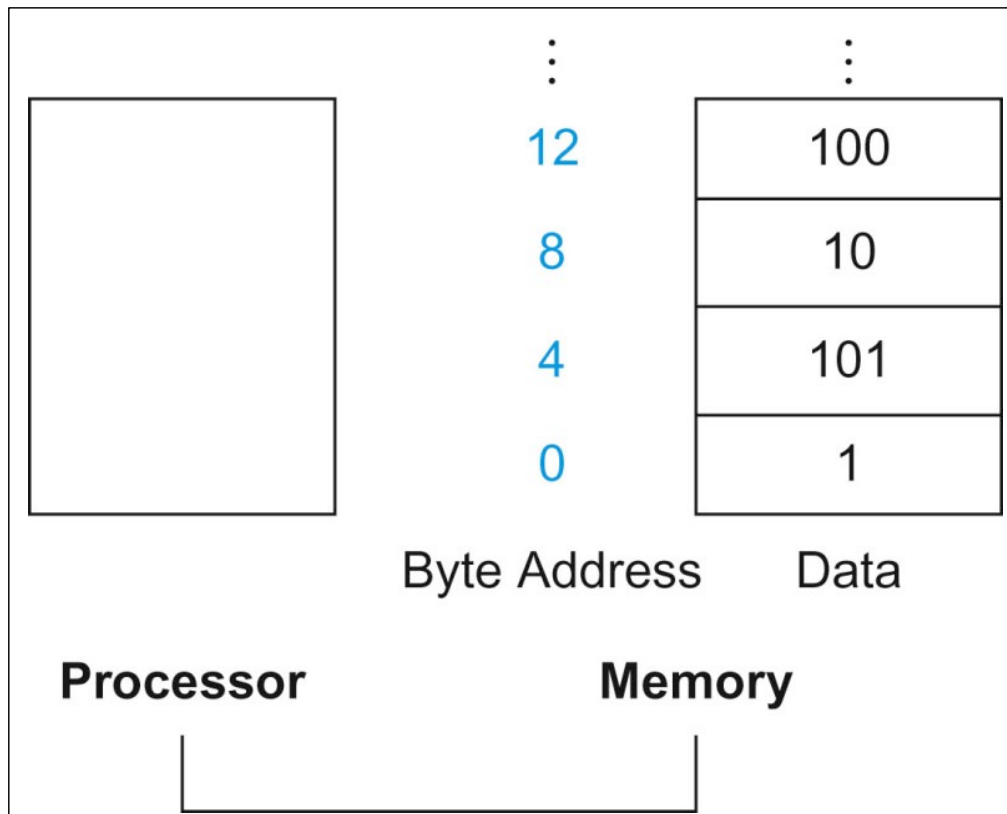
6/08/18



Guillermo Aguirre

4

Direcciones de memoria



6/08/18

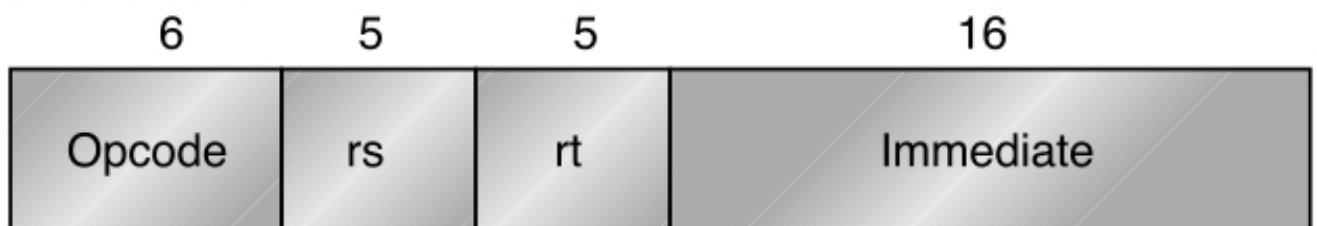


Guillermo Aguirre

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

6/08/18

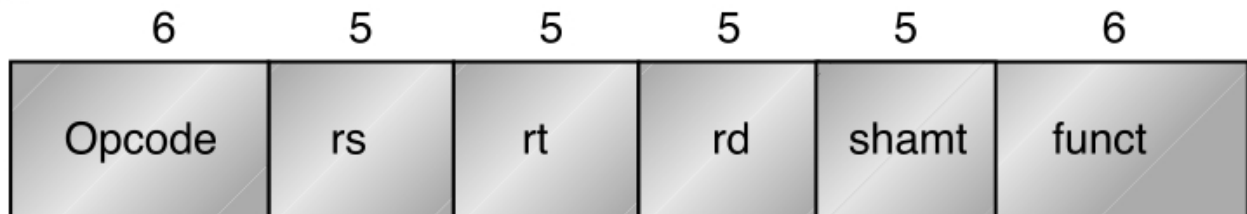


Guillermo Aguirre

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

6/08/18

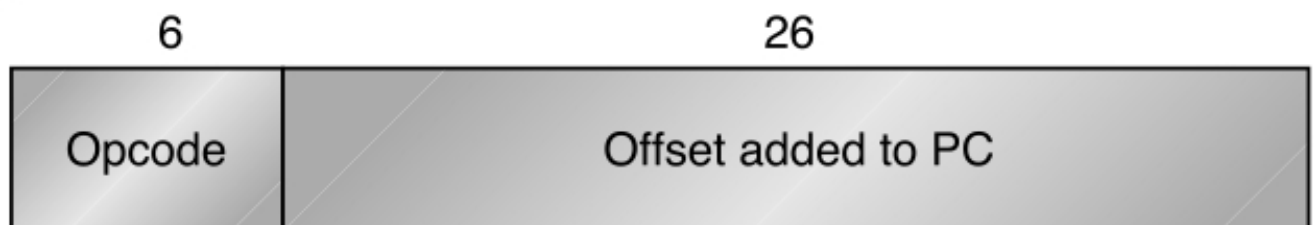


Guillermo Aguirre

7

Formato de instrucciones

J-type instruction



Jump and jump and link

Trap and return from exception

6/08/18



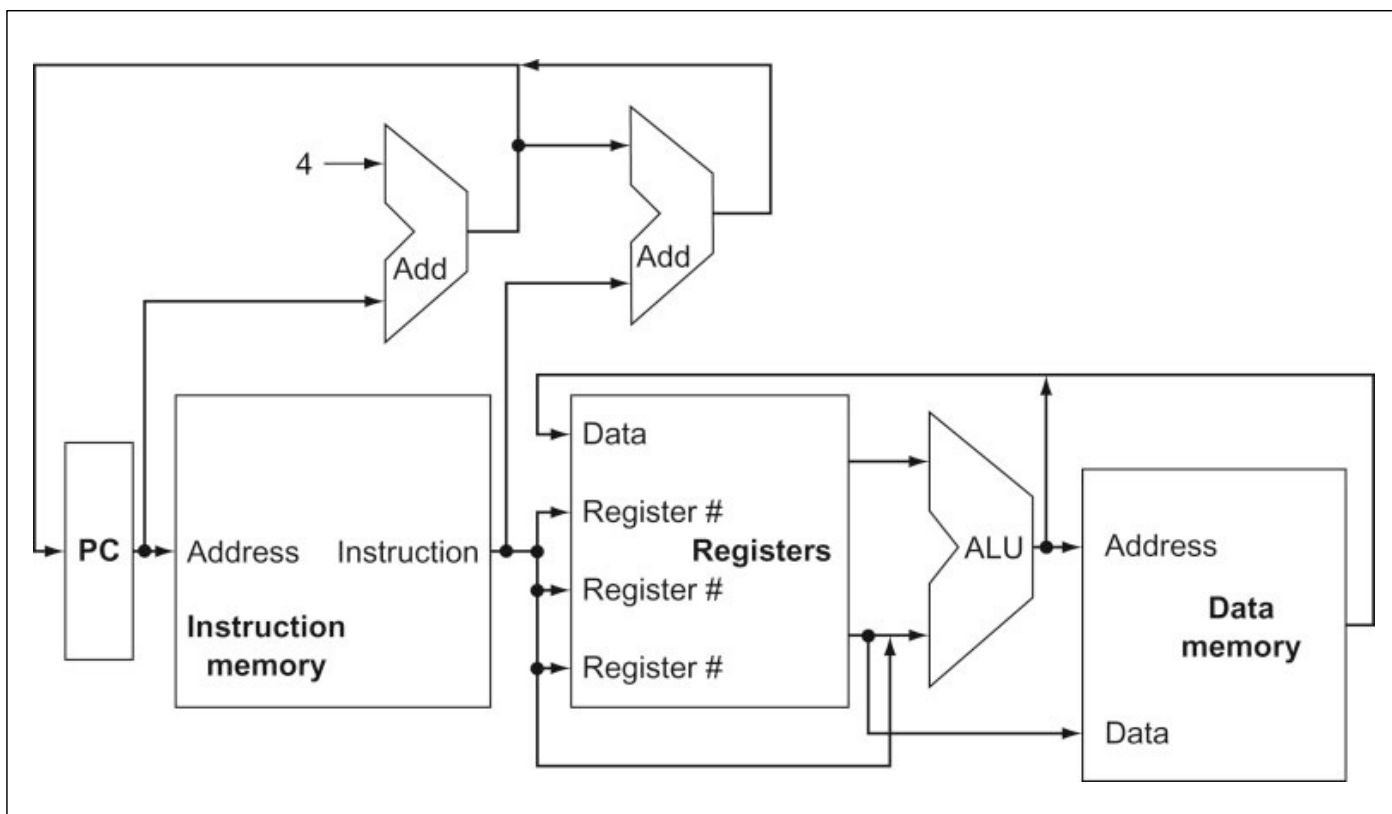
Guillermo Aguirre

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				

Visión abstracta de un MIPS



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 $(reg1) \neq (reg2)$

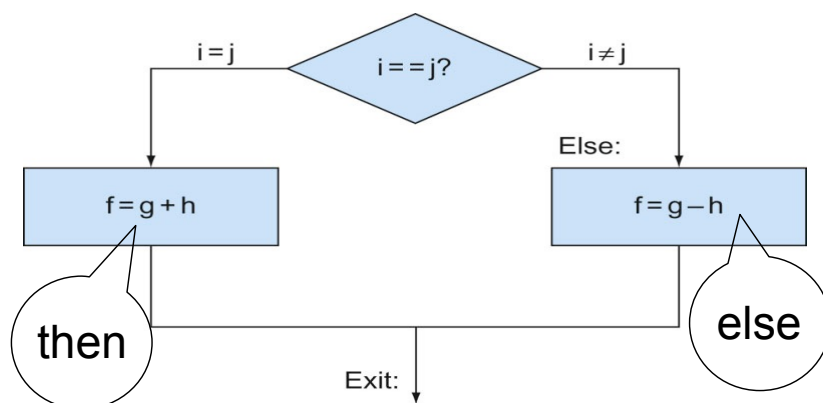
6/08/18

 Guillermo Aguirre

11

Estructuras de control

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



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

lenguaje de
ensamble

lenguaje
de máquina

6/08/18

 Guillermo Aguirre

12

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

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



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

26 bits

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



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



Uso de la pila (lifo)

Resguardo de registros del llamador.

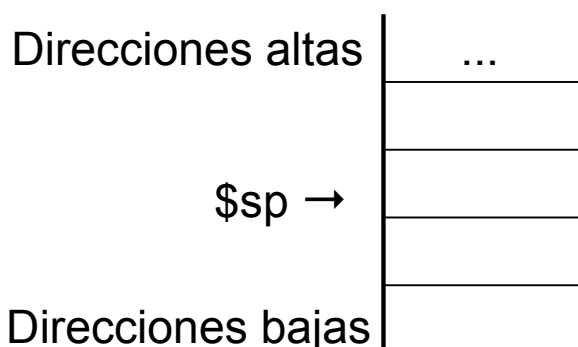
El puntero al tope de la pila es \$sp (29)

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



hoja:

<code>addi \$sp,\$sp,-12</code>	}	Resguarda registros
<code>sw \$t1,8(\$sp)</code>		
<code>sw \$t0,4(\$sp)</code>		
<code>sw \$s0,0(\$sp)</code>		
 <code>add \$t0,\$a0,\$a1</code>	}	Operación
<code>add \$t1,\$a2,\$a3</code>		
<code>sub \$s0,\$t0,\$t1</code>		
 <code>add \$v0,\$zero,\$s0</code>	}	Valor de retorno
 <code>lw \$s0,0(\$sp)</code>	}	Recupera registros
<code>lw \$t0,4(\$sp)</code>		
<code>lw \$t1,8(\$sp)</code>		
<code>addi \$sp,\$sp,12</code>		
 <code>jr \$ra</code>	}	Retorno al llamador

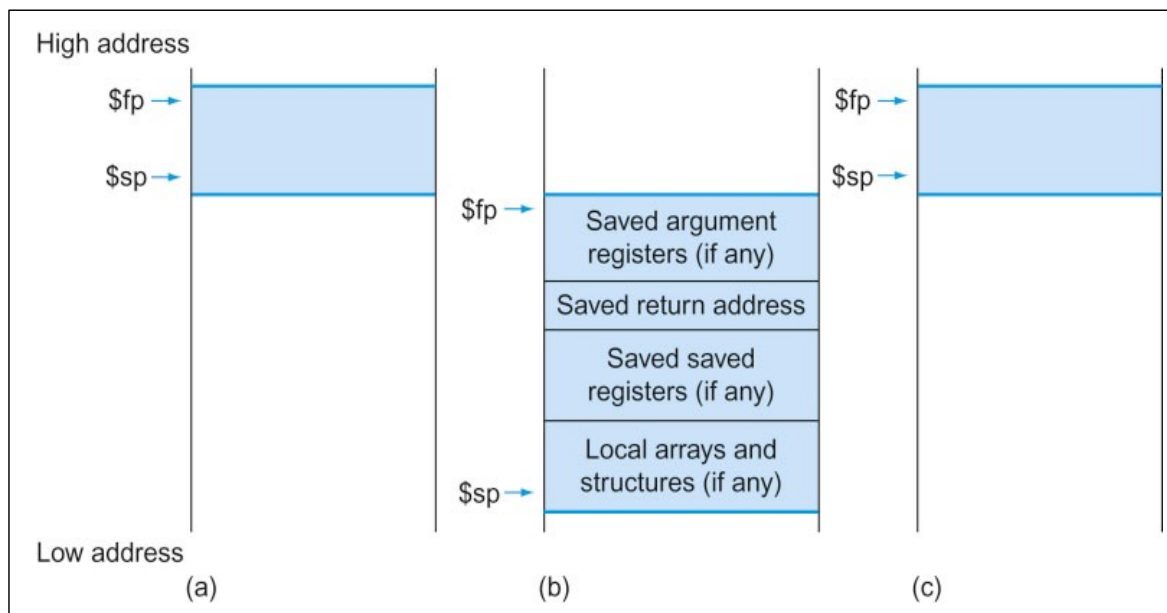
6/08/18



Guillermo Aguirre

17

Registro de activación



Puntero al marco `$fp` (30)

Base estable para las referencias locales

6/08/18



Guillermo Aguirre

18

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

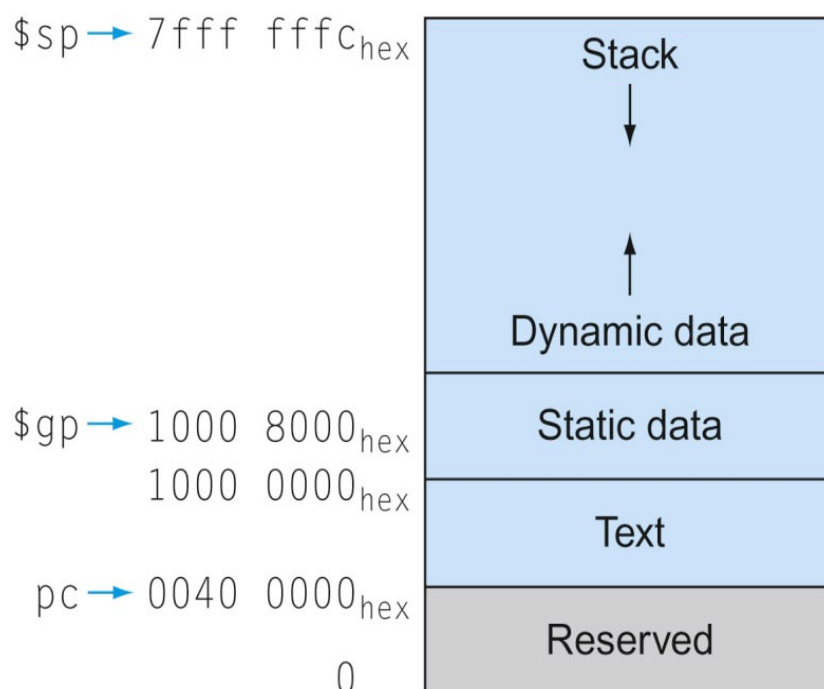
6/08/18



Guillermo Aguirre

19

Memoria para código y datos



6/08/18



Guillermo Aguirre

20

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.



Modos de direccionamiento

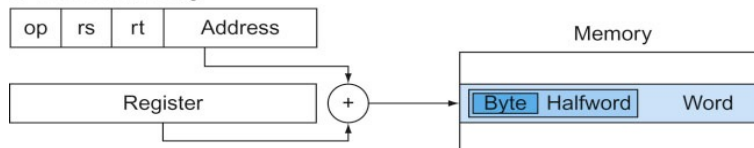
1. Immediate addressing



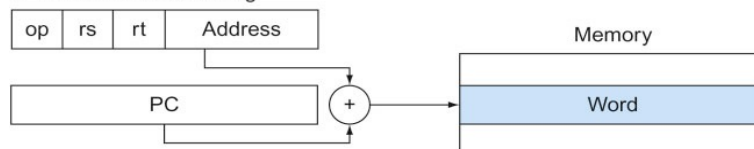
2. Register addressing



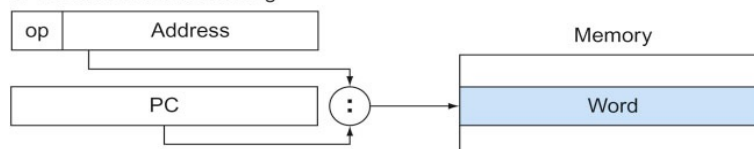
3. Base addressing



4. PC-relative addressing



5. Pseudodirect addressing



Direcciones e inmediatos de 32 bits

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

001111	00000	01000	0000 0000 1111 1111
--------	-------	-------	---------------------

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

0000 0000 1111 1111	0000 0000 0000 0000
---------------------	---------------------

j 0x1000000 #ir a la dirección 0x1000000	
2	0x400000<<2
6 bits	26 bits

bne \$s0,\$s1,Exit # ir a Exit si \$s0≠\$s1			
5	16	17	Exit
6 bits	5 bits	5 bits	16 bits

6/08/18



Guillermo Aguirre

23

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

6/08/18



Guillermo Aguirre

24

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

main:	.text la \$s6,arr	base del arreglo
	add \$s3,\$zero,\$zero	
	add \$s5,\$zero,\$zero	índice *4
loop:	sll \$t1,\$s3,2	
	add \$t1,\$t1,\$s6	dirección del elemento
	lw \$t0,0(\$t1)	
	bne \$t0,\$s5,exit	carga la palabra
	addi \$s3,\$s3,1	incremento del índice
exit:	j loop	
	jr \$ra	
	.data	
arr:	.word 0,0,1	

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

6/08/18



Guillermo Aguirre

25

¿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.

6/08/18



Guillermo Aguirre

26