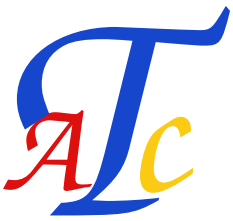

Universidad de Córdoba

Arquitectura y Tecnología de Computadores

Práctica 2

ISA de MIPS

Arquitecturas Avanzadas de Procesadores

D. Miguel Ángel Montijano Vizcaíno (el1movim@uco.es)
D. Héctor Martínez Pérez (el2mapeh@uco.es)

Curso 2025/2026

1 Introducción

En la práctica anterior se tuvo la primera toma de contacto con el entorno MARS y el ISA de MIPS. En esta segunda práctica se profundizará en el ISA de MIPS hasta adquirir los conocimientos básicos de las operaciones más relevantes del conjunto de instrucciones, así como de los conceptos básicos a la hora de implementar un programa.

En concreto, los objetivos principales de esta sesión de prácticas consistirán en:

- Conocimiento de las principales etiquetas reservadas que puede contener un programa MIPS.
- Entender el funcionamiento de las instrucciones de carga y almacenamiento: *lui, lw, lb, sw y sb.*
- Conocimiento de las principales instrucciones para operaciones aritméticas.
- Comprensión de las llamadas al sistema *syscall* y sus distintos tipos.
- Manejo de las instrucciones de salto condicional e incondicional.
- Manejo de otras instrucciones de uso común como es: *slt, sll y srl.*

2 Principales etiquetas reservadas en MIPS

En MIPS, el desarrollador tiene la capacidad de crear sus propias etiquetas para poder llevar a cabo saltos, ya sean bien condicionales o incondicionales, a través del código fuente. Esto pudo verse en el ejemplo de la reducción de los elementos de un vector práctica anterior, a la hora de implementar el bucle encargado de la reducción. No obstante, el usuario debe tener en cuenta que existe una serie de etiquetas reservadas en la ISA de MIPS que permiten definir secciones de código, y otras que permiten reservar espacios en memoria.

En primer lugar, se listan las principales etiquetas que permiten la delimitación de secciones de código:

- *.data* Inicio de la sección de datos estáticos.
- *.text* Inicio de la sección de código ejecutable.
- *.glob* Declara una etiqueta como símbolo global (ej. main). En MARS no es obligatorio, pero en otros tipos de entornos sí puede ser necesario.

A continuación, se presenta la lista de las etiquetas que permiten la reserva de espacios en memoria:

- *.word* Reserva de palabras (32 bits) en memoria.
- *.half* Reserva de media palabra (16 bits) en memoria.
- *.byte* Reserva de bytes (8 bits).

- *.ascii/.asciiz* Ambas reservan cadenas de caracteres. La opción *.asciiz* termina la cadena en *"\0"*.
- *.align* Alinea los datos en memoria. Existen arquitecturas en las que los datos en memoria deben estar alineados a cierto tamaño.

3 Instrucciones destinadas a la carga/almacenamiento de los datos

Este apartado se centra en las instrucciones destinadas a la carga/almacenamiento de los datos en memoria y los registros del procesador. Generalmente, las instrucciones de carga de un dato de memoria a registro comienzan con la letra *"l"* (procedente de *load* en inglés) y las de almacenamiento de registro a memoria con la letra *"s"* (procedente de *store* en inglés), seguidos por la letra inicial correspondiente al tamaño del dato que se va a mover, *"b"* para bytes, *"h"* para media palabra (*half word*) y *"w"* para palabra (*word*).

Hay que tener en cuenta que las instrucciones de carga como *lb* y *lh*, donde únicamente se almacena un byte o dos (media palabra), usan extensión de signo, en su defecto, si no quiere aplicar esta extensión debe usarse *lbu* o *lhu*.

3.1 *lui:* Carga de datos inmediatos

Para empezar a conocer este tipo de instrucciones del ISA de MIPS, el Código 1 muestra un ejemplo de carga de un dato inmediato. La primera línea muestra la directiva *.text*, la cual sirve para indicar el comienzo de la zona de memoria dedicada a las instrucciones, tal y como se ha comentado en el apartado anterior. Por defecto, esta zona comienza en la dirección 0x00400000, en ella pueden verse las instrucciones que ha introducido el simulador para ejecutar de forma adecuada el programa. Seguidamente, en la línea 2, aparece la etiqueta *main*, la cual indica al simulador el inicio del programa. Por defecto, esta etiqueta hace referencia a la dirección 0x00400000, tal y como puede verse en la Figura 1. A partir de dicha dirección, el simulador cargará el código de nuestro programa en el segmento de memoria de instrucciones.

La instrucción *lui* es la única instrucción de carga inmediata real, y almacena la media palabra que indica el dato inmediato de 16 bits en la parte alta del registro especificado, en este caso *$s0*. Hay que tener en cuenta que la parte baja del registro correspondiente a los bits de menor peso se fija a cero.

Código 1: Ejemplo de instrucción *lui* del ISA MIPS.

```
1       .data
2       .text
3 main: lui $s0, 0x8692
```

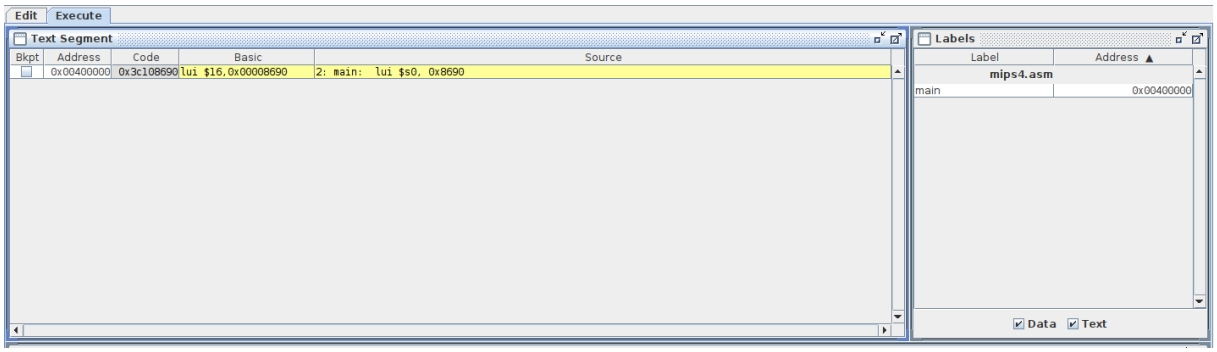

Figura 1: Ejemplo de carga de una instrucción MIPS y del direccionamiento de etiquetas en MARS.

3.2 *lw:* Carga de palabras de memoria a registro

La instrucción *lw* carga una palabra almacenada en una dirección de memoria (Código 2). Esta dirección de memoria se especifica en el segundo parámetro de la instrucción, y se le aplica el desplazamiento indicado en el registro, en este caso *$0*, para obtener la dirección final. El valor contenido en dicha posición de memoria es cargado en el registro indicado en el primer argumento de la instrucción, en este caso *$s0*.

Código 2: Ejemplo de instrucción *lw* del ISA MIPS.

```
1           .data
2 palabra:  .word 0x34226514
3           .text
4 main:     lw $s0, palabra($0)
```

3.3 *lb:* Carga de bytes

La instrucción *lb* carga el byte de una dirección de memoria en un registro (Código 3). De igual manera que la anterior instrucción, la dirección de memoria se indica en el segundo parámetro de esta instrucción, y se le aplica el desplazamiento indicado en el registro, en este caso *$0*. Almacena el contenido en el registro indicado en el primer parámetro, *$s0*.

Código 3: Ejemplo de instrucción *lb* del ISA MIPS.

```
1          .data
2 octeto:  .byte 0xf3
3          .text
4 main:    lb $s0, octeto($0)
```

3.4 *sw:* Almacenamiento de palabras desde registro a memoria

La instrucción *sw* almacena la palabra contenida en el registro indicado en la primera posición de esta instrucción, *$s0*, en la dirección de memoria especificada en la segunda posición (Código 4). El cálculo de la posición final de memoria se lleva a cabo de la misma manera que las instrucciones anteriores.

Código 4: Ejemplo de instrucción *sw* del ISA MIPS.

```
1           .data
2 palabra0: .word 0xffffffff
3 palabra1: .word 0x10203040
4           .text
5 main:     lw $s0, palabra0($0)
6           sw $s0, palabra1($0)
```

3.5 *sb:* Almacenamiento de bytes desde registro a memoria

La instrucción *sb* almacena el byte de menor peso del registro indicado en la primera posición de esta instrucción, *$s0*, en la dirección de memoria especificada en la segunda posición (Código 5). El cálculo de la posición final de memoria se lleva a cabo de la misma manera que las instrucciones anteriores.

Código 5: Ejemplo de instrucción *sb* del ISA MIPS.

```
1           .data
2 palabra0: .word 0x10203040
3 palabra1: .space 2
4           .text
5 main:     lw $s0, palabra0($0)
6           sb $s0, palabra1($0)
```

4 Instrucciones destinadas a operaciones aritméticas

Este apartado se centra en las principales instrucciones existentes en la ISA de MIPS orientadas a operaciones aritméticas. En general, existen cuatro operaciones aritméticas principales soportadas por instrucciones MIPS: suma, resta, multiplicación y división. A todas ellas, existe una característica en común: si la instrucción acaba en *.s*, se trata de una operación en coma flotante de simple precisión; si la instrucción acaba en *.d*, se trata de una operación en coma flotante de doble precisión; finalmente, si la instrucción no contiene ninguna terminación, se trata de una operación simple.

Cabe remarcar que la instrucción de multiplicación sí sufre una ligera modificación al ser acompañada por esta terminación. La instrucción base es *mult* y acompañada *mul.s*.

4.1 *add:* Operación de suma

La instrucción *add* suma el contenido de los registros indicados en la segunda y tercera posición de la instrucción, en el Código 6 *$t1 y $t2*, almacenando el resultado de la operación en el registro *$t0*.

Código 6: Ejemplo de instrucción *add* del ISA MIPS.

```
1       .data
2       .text
3 main: add $t0, $t1, $t2
```

addi: Variante de la instrucción *add*, donde el tercer operando es un dato inmediato en vez de un registro. Ej: *addi $t0, $t1, -5*.

4.2 *sub:* Operación de resta

La instrucción *sub* resta el contenido de los registros indicados en la segunda y tercera posición de la instrucción (segundo registro menos tercero), en el Código 7 *$t1 y $t2*, almacenando el resultado de la operación en el registro *$t0*.

Código 7: Ejemplo de instrucción *sub* del ISA MIPS.

```
1       .data
2       .text
3 main: sub $t0, $t1, $t2
```

subbi: Variante de la instrucción *sub*, donde el tercer operando es un dato inmediato en vez de un registro. Ej: *subbi $t0, $t1, 5*.

4.3 *mult:* Operación de multiplicación

La instrucción *mult* multiplica el contenido del primer y el segundo registro, tal y como muestra el Código 8. En MIPS, esta instrucción se trata de manera distinta a la de otras arquitecturas; el resultado se almacena siempre en los registros especiales *HI* y *LO*. Esto se debe a que el producto puede causar un desbordamiento; por lo tanto, la parte baja del resultado pasa al registro *LO*, y la parte alta al registro *HI* del procesador. Para almacenar el resultado en el registro destino, es necesario acompañar esta instrucción con:

- *mflo:* Mueve el contenido de *LO* al registro destino.
- *mfhi:* Mueve el contenido de *HI* al registro destino.

Si el producto no va a causar desbordamiento, moviendo la parte baja del resultado al registro destino sería suficiente; en caso contrario, si por ejemplo se tratase del producto de dos *INT32*, el producto debería almacenarse en un *INT64* moviendo tanto la parte baja como la alta. Un ejemplo de uso de esta instrucción puede verse en el Código 8.

Código 8: Ejemplo de instrucción *mult* del ISA MIPS.

```
1       .data
2       .text
3 main: mult $t1, $t2
4       mflo $t0
```

4.4 *div:* Operación de división

La instrucción *div* funciona similar a la instrucción *mult*. Divide el contenido del primer registro por el del segundo, como muestra el Código 9 *$t1 y $t2*. El cociente de la división se almacena en el registro *LO* y el residuo en el *HI*.

Código 9: Ejemplo de instrucción *div* del ISA MIPS.

```
1       .data
2       .text
3 main: div  $t1, $t2
4       mflo $t0
```

5 *syscall:* Llamadas al sistema

Las llamadas al sistema son peticiones que hace nuestro programa al S.O. para llevar a cabo determinadas acciones. Normalmente, estas acciones consisten en operaciones de entrada-salida, donde la llamada al sistema es similar a una llamada a una función, con la diferencia de que el cuerpo de la función forma parte del S.O. *Syscall* no se trata propiamente de una instrucción MIPS, sino de una pseudoinstrucción. Además, observando el Código 10, puede verse que se almacena información, tanto en el registro $a0, como en $v0. Más concretamente, en $v0 se almacena el código de la operación que se va a llevar a cabo (en el ejemplo se almacena el valor uno, es decir, mostrar un entero por pantalla). En la Tabla 1 se muestra el listado de los distintos códigos disponibles y sus correspondientes servicios. Por último, el registro $a almacena el valor a ser mostrado por pantalla.

Código	**Servicio**	**Argumentos**	**Resultado**
1	print int	$a0	-
2	print float	$f12	-
3	print double	$f12	-
4	print string	$a0	-
5	read int	-	int in $v0
6	read float	-	float in $f0
7	read double	-	double in $f0
8	read string	$a0=buffer, $a1=length	-
9	sbrk	$a0=amount	address in $v0
10	exit	-	-

Tabla 1: Tabla de códigos para la llamada *syscall*.

Código 10: Ejemplo de llamada al S.O. mediante la instrucción *syscall* del ISA MIPS.

```
1       .text
2 main: addi $a0, $zero, 12
3       addi $v0, $zero, 1
4       syscall
```

5.1 Instrucciones de salto incondicional/condicional

Empezando con las instrucciones de salto incondicional, podemos encontrar:

- ***j*** *(**j**ump)*: Salta a la dirección asociada a la etiqueta especificada junto a la instrucción.

- ***jr*** *(**j**ump **r**egister)*: Salta a la dirección que contiene el registro especificado junto a la instrucción.

Estas instrucciones, como se ha especificado, saltan incondicionalmente. Además de este tipo de instrucciones de salto, el ISA de MIPS también contiene un conjunto de pseudoinstrucciones de salto condicional, las cuales permiten comparar dos variables almacenadas en registros, y según el resultado de esta comparación, saltan o no a la instrucción que referencia la etiqueta. Estas psudoinstrucciones son:

- ***bge*** *(**b**ranch if **g**reater or **e**qual)*: Salta si mayor o igual.
- ***bgt*** *(**b**ranch if **g**reater **t**han)*: Salta si mayor que.
- ***ble*** *(**b**ranch if **l**ess or **e**qual)*: Salta si menor o igual.
- ***blt*** *(**b**ranch if **l**ess **t**han)*: Salta si menor que.

Además de estas pseudoinstrucciones, incluye dos instrucciones básicas de toma de decisiones que permiten implementar estructuras de control muy sencillas. Estas instrucciones son:

- ***beq*** *(**b**ranch if **eq**ual)*: Salta si igual.
- ***bne*** *(**b**ranch if **n**o **e**qual)*: Salta si no igual.

Un ejemplo de uso de ellas puede verse en el Código 11.

Código 11: Ejemplo de la instrucción de salto condicional *bne* del ISA MIPS.

```
1         .text
2 main:   addi $a0, $zero, 0
3         addi $a1, $zero, 1
4         bne  $a0, $a1, salto
5 salto:  add  $a0, $a0, $a1
```

Finalmente, en el Código 12 puede verse un caso común del uso de saltos condicionales, un bucle. Inicialmente, las líneas 6 y 7 cargan las palabras contenidas en las direcciones de memoria *dato1* y *dato2*, seguidamente, se forma el bucle hasta que *$t0 == $t1* llevando a cabo *"n"* acumulaciones sobre *$t2*.

Código 12: Implementación de bucle mediante instrucciones del ISA MIPS.

```
1          .data
2          dato1: .word 30
3          dato2: .word 40
4
5          .text
6 main:    lw   $t0, dato1($0)  # Carga una palabra ($t0=30)
7          lw   $t1, dato2($0)  # Carga una palabra ($t1=40)
8          addi $t2, $zero, 0   # Inicializa $t2 a cero
9 bucle:   addi $t2, $t2, 2     # Suma un valor inmediato ($t2+=2)
10         addi $t0, $t0, 1     # Suma un valor inmediato ($t0++)
11         bne  $t0, $t1, bucle # Salta si ambos registros no son iguales
12                              # (salto si: $t0 != $t1)
```

Resulta familiar... ¿Verdad?

Código 13: Ejemplo de bucle en C.

```c
// Cuidado, depende del compilador, las optimizaciones que este aplique
// y la propia arquitectura del procesador, la similitud con el
// ejemplo mostrado anteriormente.

#define DATO1 30
#define DATO2 40

int main() {
  int i     = DATO1;
  int j     = DATO2;
  int total = 0;
  for (; i < j; i++) {total += 2;}
}
```

5.2 slt: Instrucción de comparación

Esta instrucción perteneciente al ISA de MIPS sigue el siguiente formato *slt rd, rs, rt*. Concretamente, compara el registro *rs* frente a *rt*; si el contenido de *rs* es menor que el de *rt*, almacena el valor uno en *rd*, en caso contrario, almacena un cero. El Código 14 muestra un ejemplo de uso de *slt*.

Código 14: Ejemplo de la instrucción comparativa *slt* del ISA MIPS.

```
      .text
      addi $t0, $zero, 10
      addi $t1  $zero, 20
main: slt  $t2, $t0, $t1 #$t2=1, ya que $t0 < $t1
```

5.3 Instrucciones de desplazamiento de bits *sll y srl*

Las últimas instrucciones que se abordan en esta práctica son las encargadas de los desplazamientos de bits. A priori, pueden resultar de menor relevancia, ya que en los desarrollos de códigos cotidianos no se usan con una gran frecuencia. No obstante, en el desarrollo de códigos donde la optimización es un factor crítico, toman una gran importancia.

Las dos instrucciones disponibles para llevar a cabo desplazamiento de bits son:

- ***sll*** *(**s**hift **l**eft **l**ogical)*: Desplaza *"n"* bits a la izquierda. Equivalente a multiplicar por potencia de dos.
- ***srl*** *(**s**hift **r**ight **l**ogical)*: Desplaza *"n"* bits a la derecha. Equivalente a dividir por potencia de dos.

El Código 15 muestra un ejemplo de uso de la instrucción *sll* aplicando un desplazamiento a la izquierda de un bit.

Código 15: Ejemplo de la instrucción de desplazamiento *sll* del ISA MIPS.

```
      .text
      addi $t0, $zero, 10
main: sll  $t1, $t0, 1     #$t1 = $t0 << 1; (10 * 2)
```

Ejercicio 1

Cree un fichero con el Código 16, seguidamente ensamble el código y responda las siguientes preguntas.

Código 16: Código MIPS ejercicio 1.

```
1            .data
2 octeto:    .byte 0xf3
3 siguiente: .byte 0x20
4            .text
5 main:      lb $s0, octeto($0)
```

1.1. ¿En qué instrucciones máquina ha traducido la instrucción MIPS *lb*? ¿Es una instrucción o una pseudoinstrucción?

1.2. ¿Qué valor acaba teniendo *$s0*? ¿Porque?

1.3. Cambie la instrucción *lb* por *lbu*. ¿Qué valor acaba teniendo ahora *$s0*? ¿Porque?

1.4. Cambie el valor de *octeto* por *.word 0x10203040*. ¿Cuál es el valor de *$s0* ahora? ¿Porque?

1.5. Dejando el valor de octeto como en el ejercicio anterior, modifique la instrucción *lb* por la siguiente: *lb $s0, octeto+1($0)*. ¿Qué valor contiene $s0? ¿Por qué ha funcionado?

Ejercicio 2

Recupere el ejercicio de la sesión anterior que lleva a cabo la reducción de los elementos de un vector. Lleve a cabo las siguientes modificaciones.

2.1. En primer lugar, inicialice un nuevo registro para llevar a cabo otra reducción de los elementos. A continuación, añada un nuevo bucle que lleve a cabo otra reducción de los elementos en esta nueva variable creada.

2.2. Inmediatamente después del nuevo bucle, añada un bloque de código que compruebe si el resultado de la nueva reducción coincide con el original. En caso afirmativo, que muestre por pantalla el mensaje "Resultado correcto"; en caso contrario, que muestre por pantalla el mensaje "Resultado erróneo". Compruebe su correcto funcionamiento.

Ejercicio 3

Cree un fichero con el Código 17, seguidamente ensamble el código, ejecútelo y responda las siguientes preguntas.

Código 17: Código MIPS ejercicio 3

```
1          .data
2  dato1:  .word 30
3  dato2:  .word 40
4  res:    .space 1
5          .text
6  main:   lw  $t0, dato1($0)
7          lw  $t1, dato2($0)
8          slt $t2, $t0, $t1
9          bne $t0, $t1, fineval
10         ori $t2,  $0, 1
11 fineval: sb $t2, res($0)
```

3.1. ¿Qué valor se carga en la posición de memoria res? ¿Cuál es la dirección de dicha posición?

3.2. Inicialice ahora las posiciones de memoria *dato1* y *dato2* con los valores 50 y 20, respectivamente. Ejecute de nuevo el programa. ¿Qué valor se carga en la posición de memoria res?

3.3. Inicialice ahora las posiciones de memoria *dato1* y *dato2* con los valores 20 y 20, respectivamente. Ejecute de nuevo el programa. ¿Qué valor se carga en la posición de memoria res?

3.4. ¿Qué comparación se ha evaluado entre dato1 y dato2?

3.5. Añada un *breakpoint* en la instrucción *ori*. Ahora ejecute el programa completamente. ¿Qué ha ocurrido? ¿Cuánto vale el contador de programa en este punto? ¿A que corresponde esta dirección?

3.6. Localice la dirección de memoria donde se almacenan *dato1* y *dato2*. ¿En qué direcciones de memoria están almacenadas? ¿Qué valor contienen dichas direcciones? ¿Porque?