Asignatura de Organización de Computadoras

ARQUITECTURA MIPS

SEGUNDO TRABAJO

Alberto Hernández Cerezo

Cristian Tejedor García

Rodrigo Alonso Iglesias

Universidad de Valladolid

Segundo de Ingeniería Técnica Informática de Sistemas

ETSII

1.-Introducción a los formatos de instrucción de MIPS:

Introducida ya los conceptos más elementales de la arquitectura MIPS ahora pasamos a analizar en profundidad el formato de instrucciones presente en ésta arquitectura. Como recuerdo del anterior trabajo, uno de los factores que más por los que más destaca MIPS es por ser una arquitectura de tipo RISC con un conjunto de instrucciones bastante claro.

2.-Formato de instrucciones de MIPS:

Para entrar de lleno en el formato de las instrucciones MIPS veamos un ejemplo de instrucción representada simbólicamente:

add $t0, $s1, $s2

Esta instrucción se representa en el lenguaje MIPS como campos de números binarios de la siguiente forma:

000000	10001	10010	01000	00000	100000

Que en decimal se corresponde con:

0	17	18	8	0	32

Esta distribución de la instrucción se denomina *formato de instrucción*. Es importante saber que en MIPS, el compromiso elegido por los diseñadores de la arquitectura fue el de guardar todas las instrucciones con la misma longitud. Como consecuencia de ello, el número de bits de una instrucción MIPS es siempre de 32, el mismo tamaño que el de una palabra.

A los campos MIPS se les da una serie de nombres para su rápida identificación:

- *Op:* operación básica de la instrucción, tradicionalmente llamada código de operación.
- *Rs:* primer registro operando fuente.
- *Rt:* segundo registro operando fuente.
- *Rd:* registro operando destino, donde se almacena el resultado de la operación.
- *Shamt:* tamaño de desplazamiento (*shift amount*).
- *Funct:* función. Este campo selecciona la variante específica de la operación del campo *op*, y a veces se le denomina código de función.

Cuando una instrucción necesita campos más largos que los mostrados anteriormente aparece un problema.

Por ejemplo, la instrucción de carga debe especificar dos registros y una constante. Si la dirección usara uno de los campos de 5 bits del formato anterior, la constante dentro de la instrucción de carga estaría limitada a solo 32 o 2^5. Esta constante se usa para seleccionar elementos de tablas grandes o estructuras de datos, y frecuentemente necesita ser mucho mayor que 32, por lo que este campo de 5 bits es demasiado pequeño para ser útil.

Volviendo con la idea de **diseño de MIPS**, de que todas las instrucciones se guarden con misma longitud, se da el caso de requerirse diferentes clases de formatos de instrucción para diferentes clases de instrucción. En MIPS se distinguen tres: tipo R, tipo I y tipo J:

- **Formato tipo R**: utilizado por las instrucciones aritméticas y lógicas.

Tipo R (shamt: *shift amount* en instrucciones de desplazamiento)	Cód. Op.	Registro fuente 1	Registro fuente 2	Registro destino		Funct
	xxxxxx	rs	rt	rd	shamt	funct
	6	5	5	5	5	6
	31-26	25-21	20-16	15-11	10-6	5-0

- **Formato tipo I**: utilizado por las instrucciones de transferencia, las de salto condicional y las instrucciones con operandos inmediatos.

Tipo I (carga o almacenamiento, ramificación condicional)	Cód. Op.	Registro base	Registro destino	Desplazamiento
	xxxxxx	rs	rt	Inmediato
	6	5	5	16
	31-26	25-21	20-16	15-0

- **Formatos tipo J**: utilizado por las instrucciones de bifurcación.

Tipo J (salto incondicional)	Cód. Op.	Dirección destino
	xxxxxx	dirección
	6	26
	31-26	25-0

Aunque tener múltiples formatos complica la circuitería, se puede reducir la complejidad guardándolos de forma similar. Por ejemplo, los tres primeros campos de los formatos de tipo R e I son del mismo tamaño y tienen los mismos nombres.

Los formatos se distinguen por el valor del primer campo: a cada formato se le asigna un conjunto de valores distintos en el primer campo y por lo tanto la circuitería sabe si ha de tratar la última mitad de la instrucción como tres campos, es decir, como tipo R, o como un campo simple, tipo I, o si la instrucción es tipo J.

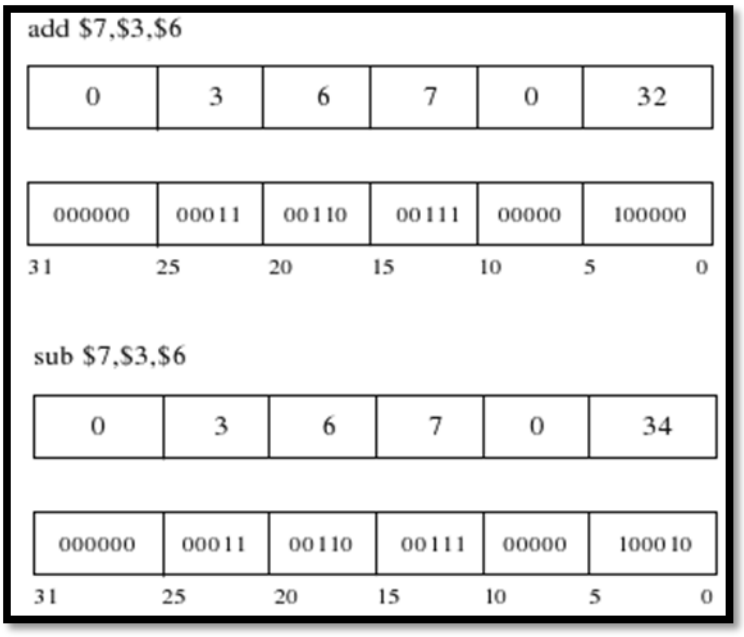
3.-Juego de instrucciones de MIPS:

El conjunto de instrucciones que especificaremos permite realizar operaciones de carga y almacenamiento desde y hacia memoria, tendrá capacidad de desarrollar programas aritméticos y lógicos y ofrecerá la posibilidad de controlar el flujo de la ejecución del programa mediante instrucciones de comparación y salto, tanto condicionales, como incondicionales. En resumidas cuentas, tendremos:

- Instrucciones aritméticas
- Instrucciones lógicas
- Instrucciones de carga/almacenamiento (o de transferencia)
- Instrucciones de comparación
- Instrucciones de salto condicional
- Instrucciones de salto incondicional

- **Instrucciones aritmético-lógicas**

El tipo de formato de las instrucciones aritméticas y lógicas es de tipo R y el número de operandos en una operación de este tipo es siempre tres. Estos operandos son siempre registros, el modo de direccionamiento empleado es, por tanto, de registro.



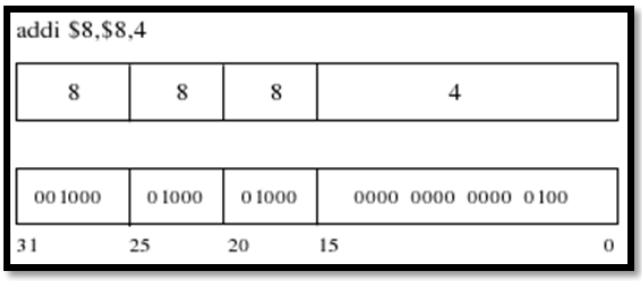
En el ejemplo de la figura se describe en estructura de lenguaje máquina MIPS las operaciones aritméticas de suma y resta respectivamente.

Muchas veces los programas usan constantes en las operaciones para, por ejemplo, incrementar un índice y apuntar al siguiente elemento de una tabla. De hecho, como dato curioso, en el compilador de C *gcc*, usado en la facultad, el 52% de las operaciones aritméticas utilizan

constantes. Si en las instrucciones anteriormente mencionadas deseásemos usar una constante habría que cargarla de memoria al registro para posteriormente sumarla.

Una alternativa que evita los accesos a memoria es la de ofrecer versiones de las instrucciones aritméticas en las cuales un operando es constante, con la nueva restricción de que esta constante se almacena dentro de la misma instrucción. Se usa en este caso el formato de instrucción de tipo I y el modo de direccionamiento es direccionamiento inmediato.

Los operandos constantes aparecen con frecuencia, y situarlos dentro de las instrucciones aritméticas hace que se ejecuten mucho más rápido.



- **<u>Instrucciones de transferencia o de carga/almacenamiento</u>**

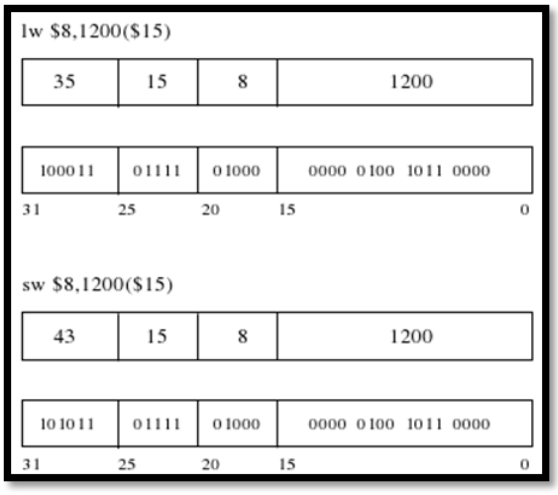
Los operandos para todas las operaciones aritméticas y lógicas se contienen en registros. Para operar con datos en memoria antes deben haberse metido estos datos en los registros.

Una operación de carga (*load*) copia información desde memoria principal a los registros.

Una operación de almacenamiento (*store)* copia información de un registro a memoria principal.

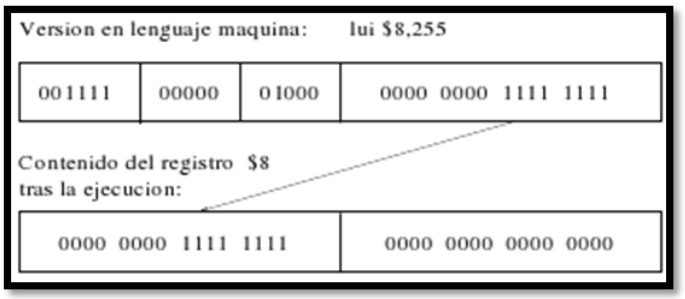
Cuando una palabra (4 bytes) se carga desde memoria en un registro o se pasa a memoria desde un registro, la dirección de memoria involucrada ha de ser múltiplo de 4. Esto es lo que se denomina restricción de alineamiento. Las direcciones que son múltiplo de cuatro se llaman direcciones alineadas. Esta restricción hace que el hardware sea más simple y rápido.

Las instrucciones de transferencia son instrucciones de tipo I. En la siguiente figura se muestran las estructuras de MIPS para dos ejemplos de operaciones de transferencia muy comunes, la carga y el almacenamiento de una palabra:



Los 16 bits de dirección significan que una instrucción, por ejemplo una instrucción de carga, puede alargar cualquier palabra dentro de la región 215 o 32768 bytes de la dirección del registro base *rs* (el primer registro del operando fuente). El direccionamiento usado en este tipo de instrucciones es direccionamiento con desplazamiento (registro-base).

Existe también otra instrucción de transferencia que implementa MIPS, la llamada instrucción *load upper inmediate* (lui) que sirve específicamente para almacenar los 16 bits de la parte alta de una constante en un registro.



- **<u>Instrucciones de salto condicional</u>**

Lo que distingue a un computador de una simple calculadora es la habilidad de tomar decisiones. Basándose en los datos de entrada y los valores creados durante la computación, el computador ejecuta diferentes instrucciones. La toma de decisiones se representa comúnmente en los lenguajes de programación usando la sentencia *if* (si condicional), combinada a veces con sentencias *go to* (ir a) y etiquetas. El lenguaje ensamblador del MIPS incluye dos instrucciones de toma de decisiones, similares a una sentencia *if* con un *go to*. Estas instrucciones se muestran en la figura 2.9.

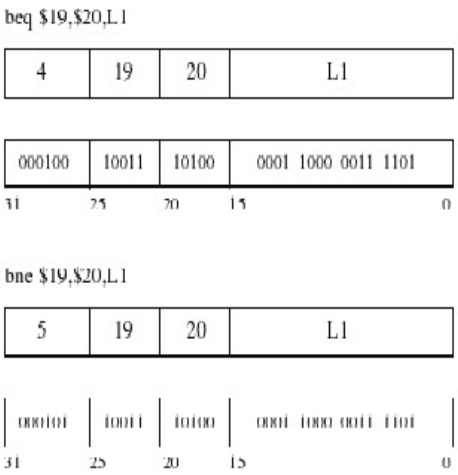


Figura 2.9: Estructura en lenguaje MIPS de las instrucciones de salto condicional

La instrucción *beq* (*branch if equal*) significa ir a la sentencia etiquetada con L1 si el valor del registro *rs* es igual al valor del registro *rt*.

La instrucción *bne* (*branch if nt equal*) significa ir a la sentencia etiquetada con L1 si el valor de *rs* no es igual al valor en *rt.*

Estas dos instrucciones se conocen tradicionalmente como saltos condicionados.

Las instrucciones de salto condicionado son de tipo-I. La prueba de igualdad y desigualdad es probablemente la más habitual, pero a veces es útil establecer comparaciones del tipo "menor que". Para ello se dispone de la instrucción MIPS *set on less than* (activar si es menor que). También existe la versión de esta instrucción utilizando operandos inmediatos. La estructura de ambas instrucciones se muestra en la figura 2.10.

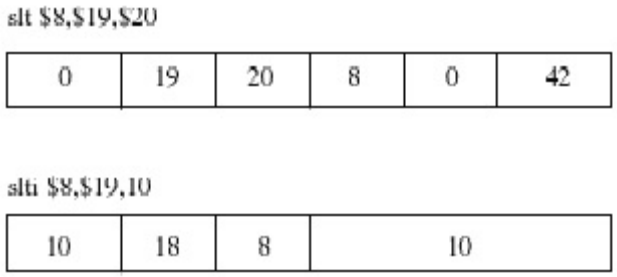


Figura 2.10: Estructura de las instrucciones *slt* y *slti* en el MIPS

- **<u>Instrucciones de comparación</u>**

Permiten poner a 1 o 0 el valor de un registro en función del cumplimiento o no de una condición.

(Ver instrucciones de comparación en el Anexo A)

- **Instrucciones de bifurcación (salto incondicional)**

Una bifurcación se puede ver como un salto incondicional, es decir, la instrucción obliga a la máquina a seguir siempre el salto. Para distinguir entre saltos condicionales e incondicionales, el nombre MIPS para este tipo de instrucción es *jump*. En la figura 2.11 se muestra esta instrucción y su formato.

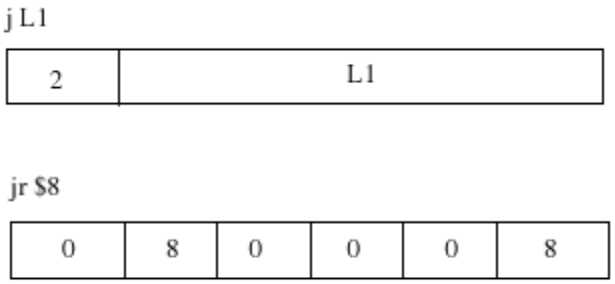


Figura 2.11: Estructura de las instrucciones *jump* y *jump register* en el MIPS

La instrucción de bifurcación *jump* es de tipo J y su modo de direccionamiento es pseudo-directo.

Muchos lenguajes de programación tienen una sentencia alternativa *case* o *switch*, que permite al programador seleccionar una de las muchas alternativas dependiendo de un único valor. Una forma de realizar *switch* es a través de una secuencia de pruebas condicionales, convirtiendo la sentencia *switch* en una cadena de sentencias if-then-else. A veces las alternativas se pueden codificar eficientemente como una tabla de direcciones de secuencias de instrucciones alternativas, llamadas tabla de direcciones de saltos, y el programa necesita solo acceder a la tabla y luego saltar a la secuencia apropiada.

La tabla de saltos es entonces simplemente una tabla de palabras, que contiene direcciones que se corresponden con etiquetas en el código. Para permitir este tipo de situaciones, los computadores como el MIPS, incluyen una instrucción denominada *jump register* (*jr*), que significa un salto incondicional a la dirección especificada en el registro. El programa carga previamente la entrada apropiada de la tabla de saltos en un registro, y luego salta a la dirección indicada usando un registro de salto.

En la figura 2.11 se muestra la estructura de esta instrucción. La instrucción *jump register* es de tipo-R y utiliza modo de direccionamiento indirecto con registro.

ANEXO A: Tablas de instrucciones MIPS

Tabla A.5. Instrucciones de movimiento de datos

Instrucción	Descripción
`li rd, inm`	Carga inmediato (Load inmediate)
`lui rd, inm`	Carga inmediato superior (Load upper inmediate)
Carga el valor inmediato `inm` de 16 bits en la parte baja (`li`) o en la parte alta (`lui`, que pone la parte baja a cero) del registro `rd`. Con `li` también es posible usar un valor inmediato de 32 bits, sustituyéndose en ese caso la instrucción por varias instrucciones máquina.	
`la rd, dir`	Carga dirección (Load Address)
Carga en el registro `rd` la dirección `dir`. Observe que esta instrucción carga la dirección y no el valor almacenado en dicha dirección.	
`lb rd, dir`	Carga byte (Load byte)
`lbu rd, dir`	Carga byte sin signo (Load byte unsigned)
Carga en el registro `rd` el valor almacenado en la dirección `dir`. El valor se copia realizando extensión de signo (`lb`) o sin realizarla (`lbu`).	
`sb ro, dir`	Almacena byte (Store byte)
Almacena en la dirección `dir` el byte almacenado en la dirección `dir`.	
`lh rd, dir`	Carga semipalabra (Load halfword)
`lhu rd, dir`	Carga semipalabra sin signo (Load halfword unsigned)
Carga en el registro `rd` el valor de 2 bytes almacenado a partir de la dirección `dir`. El valor se copia realizando extensión de signo (`lh`) o sin realizarla (`lhu`). Si la dirección `dir` no está alineada a nivel de semipalabra (dirección múltiplo de 2) se produce una excepción.	
`sh ro, dir`	Almacena semipalabra (Store halfword)
Almacena a partir de la posición de memoria `dir` el valor de 2 bytes que se encuentra en la parte inferior del registro `ro`. Si la dirección `dir` no está alineada a nivel de semipalabra (dirección múltiplo de 2) se produce una excepción.	
`lw rd, dir`	Carga palabra (Load word)
Carga el valor de 4 bytes almacenado a partir de la posición de memoria `dir` en el registro `rd`. Si la dirección `dir` no está alineada a nivel de palabra (dirección múltiplo de 4) se produce una excepción.	
`sw ro, dir`	Almacena palabra (Store word)
Almacena a partir de la posición de memoria `dir` el valor de 4 bytes que se encuentra en el registro `ro`. Si la dirección `dir` no está alineada a nivel de palabra (dirección múltiplo de 4) se produce una excepción.	
`ll rd, dir`	Carga palabra enlazada (Load linked word)
Carga en el registro `rd` el valor de 4 bytes almacenado a partir de la dirección `dir`. Esta instrucción inicia una operación atómica de lectura-modificación-escritura[2].	
`sc ro, dir`	Almacena palabra condicionalmente (Store word conditional)
Almacena en la posición de memoria `dir` el valor del registro `ro` e indica el éxito (`1`) o fracaso (`0`) en dicho registro. Esta instrucción completa una operación atómica de lectura-modificación-escritura[3].	
`ld rd, dir`	Carga doble palabra (Load double word)
Carga en el registro `rd` y en el registro siguiente a éste el valor de 8 bytes almacenado a partir de la dirección `dir`. Si la dirección `dir` no está alineada a nivel de palabra (dirección múltiplo de 4) se produce una excepción.	
`sd ro, dir`	Almacena doble palabra (Store double word)
Almacena a partir de la dirección `dir` la doble palabra almacenada en el registro `ro` y en el registro siguiente. Si la dirección `dir` no está alineada a nivel de palabra (dirección múltiplo de 4) se produce una excepción.	
`move rd, ro`	Mueve (Move)
Copia el valor del registro general `ro` en el registro general `rd`.	
`movn rd, ro1, ro2`	Mueve si distinto de cero (Move conditional on not zero)
Comprueba si el registro `ro2` es distinto de cero y, en ese caso, copia el valor del registro `ro1` al registro `rd`.	
`mfhi rd`	Mueve desde registro HI (Move from HI register)
`mflo rd`	Mueve desde registro LO (Move from LO register)
Copia el valor del registro especial `HI` (`mfhi`) o `LO` (`mflo`) al registro general `rd`.	
`mthi rd`	Mueve a registro HI (Move to HI register)
`mtlo rd`	Mueve a registro LO (Move to LO register)
Copia el valor del registro general `rd` en el registro especial `HI` (`mthi`) o `LO` (`mtlo`).	

Tabla A.6. Instrucciones de movimiento de datos no alineado

Instrucción	Descripción
`ulh rd, dir`	Carga semipalabra desalineada (Unaligned load halfword)
`ulhu rd, dir`	Carga semipalabra desalineada sin signo (Unaligned load halfword unsigned)
Carga el valor de 16 bits que se encuentra a partir de la posición de memoria `dir`, en el registro `rd`, realizando extensión de signo (`ulh`) o sin realizarla (`ulhu`).	
`ush ro, dir`	Almacena semipalabra desalineada (Unaligned store halfword)
Almacena los 16 bits menos significativos del valor del registro `ro` a partir de la posición de memoria `rd`.	
`ulw rd, dir`	Carga palabra desalineada (Unaligned load word)
Carga el valor de 32 bits que se encuentra a partir de la posición de memoria `dir` en el registro `rd`.	
`usw ro, dir`	Almacena palabra desalineada (Unaligned store word)
Almacena el valor del registro `ro` a partir de la posición de memoria `rd`.	
`lwl rd, dir`	Carga palabra izquierda (Load word left)
`lwr rd, dir`	Carga palabra derecha (Load word right)
Carga los bytes que van desde la posición de memoria `dir` hasta el anterior límite de palabra (`lwl`) o el siguiente límite de palabra (`lwr`) en los bytes más significativos (`lwl`) o menos significativos (`lwr`) del registro `rd`.	
`swl ro, dir`	Almacena palabra izquierda (Store word left)
`swr ro, dir`	Almacena palabra derecha (Store word right)
Almacena en los bytes que van desde la posición de memoria `dir` hasta el anterior límite de palabra (`swl`) o hasta el siguiente límite de palabra (`swr`), el valor de los bytes más significativos (`swl`) o menos significativos (`swr`) del registro `ro`.	

Tabla A.7. Instrucciones aritméticas del procesador

Instrucción	Descripción
`abs rdest, rorig`	Valor absoluto (Absolute value)
Pone en el registro `rdest` el valor absoluto del registro `rorig`.	
`neg rd, ro`	Niega palabra (Negate word)
`negu rd, ro`	Niega palabra sin signo (Negate word unsigned)
Pone en el registro `rd` el valor del registro `ro`, cambiado de signo. La instrucción `neg` provoca una excepción si se produce desbordamiento. La instrucción `negu` ignora el desbordamiento.	
`add rd, ro1, ro2`	Suma palabra (Add word)
`addi rd, ro1, inm`	Suma palabra inmediata (Add word inmediate)
`addu rd, ro1, ro2`	Suma palabra sin signo (Add word unsigned)
`addiu rd, ro1, inm`	Suma palabra inmediata sin signo (Add word inmediate unsigned)
Suma el valor del registro `ro1` y el del registro `ro2` (o el valor inmediato de 16 bits `inm`) y deja el resultado en el registro `rd`. Si la instrucción es *unsigned* (`addu` o `addiu`) se consideran las cantidades como números en binario puro y se ignora el desbordamiento realizándose una suma módulo 2^{32}. En otro caso (`add` o `addi`) se consideran los operandos en complemento a dos, y cuando se produce un desbordamiento no se modifica el registro `rd` y se provoca una excepción del procesador.	
`sub rd, ro1, ro2`	Resta palabra (Subtract word)
`subu rd, ro1, ro2`	Resta palabra sin signo (Subtract unsigned word)
Pone en el registro `rd` el resultado de calcular la resta entre los valores de los registros `ro1` y `ro2`. Si la instrucción es *unsigned* (`subu`) se consideran las cantidades como números en binario puro y se ignora el desbordamiento realizándose una resta módulo 2^{32}. En otro caso (`sub`) se consideran los operandos en complemento a dos, y cuando se produce desbordamiento no se modifica el registro `rd` y se provoca una excepción del procesador.	
`mul rd, ro1, ro2`	Multiplica palabra (Multiply word)
`mulo rd, ro1, ro2`	Multiplica con desbordamiento (Multiply with overflow)
`mulou rd, ro1, ro2`	Multiplica sin signo con desbordamiento (Multiply unsigned with overflow)
Multiplica el valor de los registros `ro1` y `ro2` y deja los 32 bits menos significativos del resultado en el registro `rd`. Si la instrucción es con *overflow* (`mulo` o `mulou`) se provoca una excepción del procesador en caso de desbordamiento. Las instrucciones `mul` y `mulo` realizan multiplicación en complemento a dos, mientras que `mulou` realiza multiplicación en binario puro.	
`mult ro1, ro2`	Multiplica palabra (Multiply word)
`multu ro1, ro2`	Multiplica palabra sin signo (Multiply unsigned word)
Multiplica los valores de los registros `ro1` y `ro2` y deja el resultado en el par de registros especiales formado por `HI` y `LO`. Esta instrucción nunca provoca una excepción. La operación puede realizarse sin signo (`mult`) o con signo (`multu`).	
`madd ro1, ro2`	Multiplica y suma (Multiply add)
`maddu ro1, ro2`	Multiplica y suma sin signo (Multiply add unsigned)
`msub ro1, ro2`	Multiplica y resta (Multiply subtract)
`msubu ro1, ro2`	Multiplica y resta sin signo (Multiply subtract unsigned)
Multiplica el valor del los registros `ro1` y `ro2` y suma (instrucciones `madd` y `maddu`) o resta (instrucciones `msub` y `msubu`) el resultado al par de registros formados por `HI` y `LO`, dejando el resultado en dicho par de registros. Esta instrucción nunca provoca una excepción. Las operación se puede realizar interpretando los valores como números en complemento a dos (`madd` y `msub`) o como números en binario puro (`maddu` y `msubu`).	
`div ro1, ro2`	Divide palabra (Divide word)
`divu ro1, ro2`	Divide palabra sin signo (Divide unsigned word)
Divide el valor del registro `ro1` entre el valor del registro `ro2`. Ambos valores se consideran cantidades enteras sin signo (`divu`) o con signo (`div`). El cociente de la división se almacena en el registro `LO` y el resto de la división en el registro `HI`. Esta instrucción nunca provoca una excepción.	
`rem rd, ro1, ro2`	Resto (Remainder)
`remu rd, ro1, ro2`	Resto sin signo (Remainder unsigned)
Pone en el registro `rd` el resultado de calcular el resto de la división entre los valores de los registros `ro1` y `ro2`. Todos los operandos se consideran números enteros en complemento a dos (`rem`) o en binario puro (`remu`).	

Tabla A.8. Instrucciones lógicas del procesador

Instrucción	Descripción
`not rd, ro`	No (Not)
Pone en el registro `rd` el resultado de realizar una operación *not* a nivel de bit sobre el registro `ro`.	
`and rd, ro1, ro2`	Y (And)
`andi rd, ro, inm`	Y inmediato (And inmediate)
Pone en el registro `rd` el resultado de realizar una operación *and* a nivel de bit entre los registros `ro1` y `ro2` (en el caso de la instrucción and) o entre el registro `ro` y el valor de 16 bits `inm` extendido con ceros (en el caso de la instrucción `andi`).	
`or rd, ro1, ro2`	O (Or)
`ori rd, ro1 inm`	O inmediato (Or inmediate)
Pone en el registro `rd` el resultado de realizar una operación *or* a nivel de bit entre los registros `ro1` y `ro2` (en el caso de la instrucción or) o entre el registro `ro` y el valor de 16 bits `inm` extendido con ceros (en el caso de la instrucción `ori`).	
`nor rd, ro1, ro2`	O negado (Not or)
Pone en el registro `rd` el resultado de realizar una operación *nor* a nivel de bit entre los registros `ro1` y `ro2`.	
`xor rd, ro1, ro2`	O exclusivo (Exclusive or)
`xori rd, ro1 inm`	O exclusivo inmediato (Exclusive or inmediate)
Pone en el registro `rd` el resultado de realizar una operación *xor* entre los registros `ro1` y `ro2` (en el caso de la instrucción xor) o entre el registro `ro` y el valor de 16 bits `inm` extendido con ceros (en el caso de la instrucción `xori`).	

Tabla A.9. Instrucciones de desplazamiento del procesador

Instrucción	Descripción
`rol rd, ro1, ro2`	Rota a la izquierda (Rotate left)
`ror rd, ro1, ro2`	Rota a la derecha (Rotate right)
Pone en el registro `rd` el resultado de rotar el valor del registro `ro1` tantos bits a la izquierda (`rol`) o a la derecha (`ror`) como indique el valor del registro `ro2`.	
`sll rd, ro1, inm`	Desplazamiento lógico a izquierda (Shift left logical)
`sllv rd, ro1, ro2`	Desplazamiento variable lógico a izquierda (Shift left logical variable)
`srl rd, ro1, inm`	Desplazamiento lógico a derecha (Shift right logical)
`srlv rd, ro1, ro2`	Desplazamiento variable lógico a derecha (Shift right logical variable)
Pone en el registro `rd` el resultado de desplazar el valor del registro `ro1` tantos bits a la izquierda (`sll` o `sllv`) o a la derecha (`srl` o `srlv`) como indique el valor inmediato `inm` o el valor del registro `ro2`. El desplazamiento se realiza insertando ceros por el extremo opuesto.	
`sra rd, ro1, inm`	Desplazamiento aritmético a derecha (Shift right arithmetic)
`srav rd, ro1, ro2`	Desplazamiento aritmético a derecha variable (Shift right arithmetic variable)
Pone en el registro `rd` el resultado de desplazar el valor del registro `ro1` tantos bits a la derecha como indique el valor inmediato `inm` o el valor del registro `ro2`. El desplazamiento se realiza insertando el valor del bit de signo original (desplazamiento aritmético) por la izquierda.	
`clo rd, ro`	Cuenta unos en palabra (Count leading ones)
`clz rd, ro`	Cuenta ceros en palabra (Count leading zeros)
Cuenta el número de bits seguidos que están activados (`clo`) o desactivados (`clz`) en el registro `ro`, comenzando a contar por el bit más significativo, y almacena dicho número en el registro `rd`.	

Tabla A.10. Instrucciones de comparación

Instrucción	Descripción
`seq rd, ro1, ro2`	Activa si igual (Set on equal)
`sne rd, ro1, ro2`	Activa si distinto (Set on not equal)
`sge rd, ro1, ro2`	Activa si mayor o igual (Set on greater or equal)
`sgt rd, ro1, ro2`	Activa si mayor (Set on greater than)
`sle rd, ro1, ro2`	Activa si menor o igual (Set on less or equal)
`slt rd, ro1, ro2`	Activa si menor (Set on less than)
`slti rd, ro, inm`	Activa si menor inmediato (Set on less than inmediate)
Pone el valor `1` en el registro `rd`, si los valores de los registros `ro1` y `ro2` cumplen la correspondiente condición. En otro caso pone el registro `rd` a cero. Si la comparación implica algún tipo de orden se.considera que los números se interpretan en complemento a dos.	
`sgeu rd, ro1, ro2`	Activa si mayor o igual sin signo (Set on greater or equal unsigned)
`sgtu rd, ro1, ro2`	Activa si mayor sin signo (Set on greater than unsigned)
`sleu rd, ro1, ro2`	Activa si menor o igual sin signo (Set on less or equal unsigned)
`sltu rd, ro1, ro2`	Activa si menor sin signo (Set on less than unsigned)
`sltiu rd, ro, inm`	Activa si menor inmediato sin signo (Set on less than inmediate unsigned)
Pone el valor `1` en el registro `rd`, si el valor del registro `ro1` y el registro `ro2` cumplen la correspondiente condición. En otro caso pone el registro `rd` a cero. Se considera que los números se interpretan en binario puro.	

Tabla A.11. Instrucciones de salto

Instrucción	Descripción
`j etiq`	Salto (Jump)
Salta a la dirección marcada con `etiq`.	
`b etiq`	Bifurcación incondicional (Unconditionally branch)
Salta incondicionalmente a la instrucción marcada con `etiq`.	
`beqz ro, etiq`	Bifurcación si igual a cero (Branch on equal zero)
`bnez ro, etiq`	Bifurcación si distinto de cero (Branch on not equal zero)
`bgtz ro, etiq`	Bifurcación si mayor que cero (Branch on greater than zero)
`bgez ro, etiq`	Bifurcación si mayor o igual que cero (Branch on greater or equal than zero)
`bltz ro, etiq`	Bifurcación si menor que cero (Branch on less than zero)
`blez ro, etiq`	Bifurcación si menor o igual que cero (Branch on less or equal than zero)
Salta, si el valor del registro `ro` cumple una relación con respecto al valor cero, a la instrucción marcada con `etiq`.	
`bgtzl ro, etiq`	Bifurcación si mayor que cero con retardo (Branch on greater than zero likely)
`bgezl ro, etiq`	Bifurcación si mayor o igual que cero con retardo (Branch on greater or equal than zero likely)
`bltzl ro, etiq`	Bifurcación si menor que cero con retardo (Branch on less than zero likely)
`blezl ro, etiq`	Bifurcación si menor o igual que cero con retardo (Branch on less or equal than zero likely)
Salta, si el valor del registro `ro` cumple una relación con respecto al valor cero, a la instrucción marcada con `etiq`. Estas instrucciones son retardadas.	
`beq ro1, ro2, etiq`	Bifurcación si igual (Branch on equal)
`bne ro1, ro2, etiq`	Bifurcación si no igual (Branch on not equal)
`bgt ro1, ro2, etiq`	Bifurcación si mayor que (Branch on greater than)
`bgtu ro1, ro2, etiq`	Bifurcación si mayor que sin signo (Branch on greater than unsigned)
`bge ro1, ro2, etiq`	Bifurcación si mayor o igual (Branch on greater or equal)
`bgeu ro1, ro2, etiq`	Bifurcación si mayor o igual sin signo (Branch on greater or equal unsigned)
`blt ro1, ro2, etiq`	Bifurcación si menor que (Branch on less than)
`bltu ro1, ro2, etiq`	Bifurcación si menor que sin signo (Branch on less than unsigned)
`ble ro1, ro2, etiq`	Bifurcación si menor o igual que (Branch on less or equal)
`bleu ro1, ro2, etiq`	Bifurcación si menor o igual que sin signo (Branch on less or equal unsigned)
Salta, si los valores de los registros `ro1` y `ro2` cumplen una relación, a la instrucción marcada con `etiq`.	
`beql ro1, ro2, etiq`	Bifurcación si igual con retardo (Branch on equal likely)
`bnel ro1, ro2, etiq`	Bifurcación si distinto con retardo (Branch on not equal likely)
Salta, si los valores de los registros `ro1` y `ro2` cumplen una relación, a la instrucción marcada con `etiq`. Estas instrucciones son retardadas.	
`jal etiq`	Salta y enlaza (Jump and link)
`bal etiq`	Bifurca y enlaza (Branch and link)
Salta incondicionalmente a la instrucción marcada con `etiq`, guardando la dirección de la siguiente instrucción en `$ra`.	
`bltzal ro, etiq`	Bifurca y enlaza si menor que cero (Branch on less than zero and link)
`bgezal ro, etiq`	Bifurca y enlaza si mayor que cero (Branch on greater than zero and link)
Salta, si el valor del registro `ro` cumple la condición, a la instrucción marcada con `etiq`, salvando la dirección de la siguiente instrucción en `$ra`.	
`bltzall ro, etiq`	Bifurca y enlaza si menor que cero con retardo (Branch on less than zero and link likely)
`bgezall ro, etiq`	Bifurca y enlaza si mayor que cero con retardo (Branch on greater than zero and link likely)
Salta, si el valor del registro `ro` cumple la condición, a la instrucción marcada con `etiq`, salvando la dirección de la siguiente instrucción en `$ra`. Esta instrucción es una instrucción con retardo.	
`jalr rd`	Salta y enlaza a registro (Jump and link register)
`jalr rd, ro`	Salta y enlaza a registro (Jump and link register)
Salta a la dirección especificada por el valor del registro `rd`, guardando la dirección de la siguiente instrucción en el registro `$ra` o en el registro `ro`.	
`jr rd`	Salta a registro (Jump register)
Salta a la dirección especificada por el valor del registro `rd`.	

5.-Bibliografía y referencias orientativas:

- Libro: *Estructura De Computadores, Procesadores Mips Y Su Ensamblador, José Antonio Álvarez. Ra-Ma.*

- Libro: *Problemas resueltos de estructura de computadores / Félix García Carballeira. Paraninfo.*

- http://docs.google.com/viewer?a=v&q=cache:L3am8MHddZcJ:dac.escet.urjc.es/docencia/ETC-ITIG_LADE/teoria-cuat1/tema12_ensamblador_MIPS.pdf+mips+funciones+i+r+j&hl=es&gl=es&pid=bl&srcid=ADGEESi1yfb_7UVPKQkrY8EeloNUl6fRzyvEBRT2GYxicYF6xd1Op9HzVY85xr2Fgb5VA0YnAWo78akMMnAbM4CJ79SFJZF9UdJTDyNSrpV2zCM2n8k-AP-yISsINug12o0gOizV8T2w&sig=AHIEtbRFEd6_QXwdRrU_rt79JK3qqX9xCA

- http://docs.google.com/viewer?a=v&q=cache:WpfcnP2ahE0J:quegrande.org/apuntes/EI/2/ECm1/teoria/09-10/tema_2.pdf+formato+de+instruccion+mips&hl=es&gl=es&pid=bl&srcid=ADGEEShYbTW-c6_5lFGBRWo64A6nv38-W217S7ykdYo2VBBAGYxOQkz_suM2pvo6fol86n3VWd-u-a5TEQRB_HzVFmWGkcVM6KgGalLZojaTh4tej_ljB2Pty4_zhRYsAnSLOaA248cB&sig=AHIEtbS8AmPQujzmZjrRiHChk8eUEz1Xsg

- http://books.google.es/books?id=vPmSWr-wDt8C&pg=PA273&lpg=PA273&dq=arquitectura+MIPS+registros&source=bl&ots=zjcMLwABTz&sig=gGU0yQqPar_GsX9a4Ap9HoTKvSY&hl=es&ei=y8_BTM-CGNSA4Qae4Nn9Cw&sa=X&oi=book_result&ct=result&resnum=1&ved=0CBkQ6AEwADgK#v=onepage&q=arquitectura%20MIPS%20registros&f=false

6.- Miembros del grupo

- ***Alberto Hernández Cerezo***
- ***Rodrigo Alonso Iglesias***
- ***Cristian Tejedor García***