SESIÓN 1 – PROCESADOR MIPS I

INTRODUCCIÓN A LA PROGRAMACIÓN DEL MIPS Y AL SIMULADOR VISUALMIPS32

ARQUITECTURA DE COMPUTADORES. 2º CURSO

1. OBJETIVOS

El objetivo de esta práctica es familiarizar al alumno con la programación en ensamblador de procesadores con arquitectura RISC (como es el procesador MIPS) y con el entorno de simulación VisualMIPS32 0.20.304.0. Para ello, el alumno realizará una serie de ejercicios de programación que irán aumentando de dificultad gradualmente, partiendo de un código escrito en el lenguaje de programación de alto nivel C.

2. INTRODUCCIÓN TEÓRICA. ARQUITECTURA DEL JUEGO DE INSTRUCCIONES MIPS

2.1. CARACTERÍSTICAS DE LA ARQUITECTURA

Los procesadores MIPS forman parte de una gran familia de procesadores RISC desarrollados por MIPS Technologies. Esta asignatura se centra principalmente en el procesador R2000 (primer procesador comercial de esta familia) con una unidad de coma flotante R2010. La sencillez de su arquitectura y el hecho de que muchos procesadores actuales se basan en su diseño, lo convierte en un procesador adecuado para el estudio de la arquitectura de los procesadores.

Algunas características de este procesador son:

- Procesador RISC (arquitectura de carga y almacenamiento) de 32 bits
- Buses de datos y de direcciones de 32 bits.
- Almacenamiento en memoria tanto en modo *Little Endian* como en *Big Endian*, aunque en el simulador **siempre emplea *Big Endian*.**
- 32 registros de propósito general de 32 bits (el Anexo 1 muestra la nomenclatura y las características de cada registro)
- Unidades funcionales independientes para el producto y la división, que contienen además los registros *HI* y *LO* para almacenar el resultado de operaciones estos tipos (véase *figura 1*).
- Coprocesador de coma flotante (identificado como coprocesador 1) con un banco de 32 registros de 32 bits (*figura 1*). Estos registros pueden:
  - Contener números en coma flotante de simple precisión (32 bits).
  - O agruparse por pares para almacenar números en coma flotante de doble precisión (64 bits).

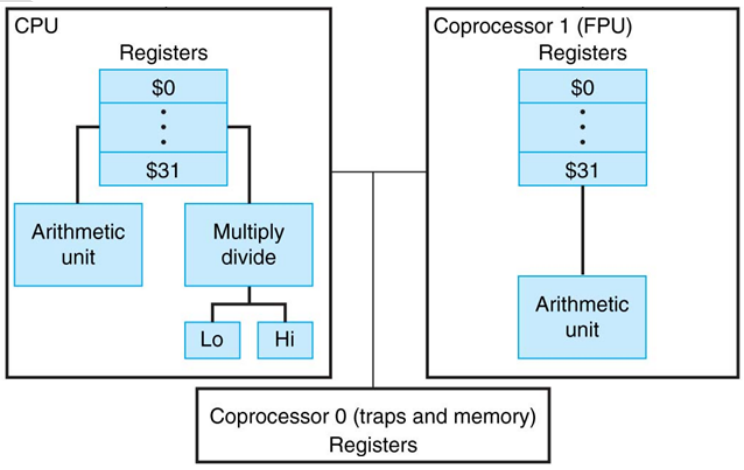

Figura 1. Unidades de procesamiento del MIPS R2000

2.2. MODOS DE DIRECCIONAMIENTO.

Los modos de direccionamiento hacen alusión a las diferentes nomenclaturas para especificar un operando dentro de una instrucción en lenguaje ensamblador. En las arquitecturas RISC, cada tipo de instrucción sólo admite un conjunto reducido de modos de direccionamientos. A continuación, se muestran todos los modos de direccionamientos del MIPS con ejemplos de algunas instrucciones que los usan.

A. Registro. El operando está en el registro (especificado en el anexo 2 como *reg* para registros de enteros y *freg* para registros en FP). Para especificar los registros de enteros en ensamblador puede utilizarse dos nomenclaturas. El símbolo '$' seguido del número del registro…

Ej. *add $1,$2,$3* ; $1 ← $2 + $3

… o bien el símbolo '$' seguido de una palabra que define su utilización (véase Anexo 1)

Ej. *add $at,$v0,$v1* ; Equivalente a la instrucción del ejemplo anterior

B. Inmediato. El operando es una constante de 16 bits almacenada en la propia instrucción (especificado como *imm16* en la lista de instrucciones del Anexo 2).

Ej. *addi $1,$2,* ***2000*** ; $1 ← $2 + 2000

C. Registro base más desplazamiento. Direccionamiento a memoria en el que la dirección efectiva viene determinada por el contenido de un registro (registro base) más una constante (desplazamiento). Su sintaxis es *inm(reg)*, siendo *reg* el registro base e *inm* el desplazamiento.

Ej. *lw $1,* ***24($2)*** ; $1 ← Mem32([$2]+24)
sb $5, ***20($6)*** ; Mem8([$6]+20) ← $5

Cabe recordar que, como en todas las arquitecturas RISC, este modo de direccionamiento sólo se usa en instrucciones de acceso a memoria (cargas y almacenamientos) por lo que nunca puede ser utilizado para realizar operaciones ALU en memoria.

D. Relativo al Contador de Programa (PC). Modo de direccionamiento empleado exclusivamente en los saltos condicionales (como instrucciones *beq*, *bne*, *beqz*, *bnez*) en el que la dirección destino de salto se obtiene sumando una constante de 16 bits en complemento A2 al registro contador de programa (PC).

A efectos prácticos, este modo queda oculto en los entornos de programación mediante el uso de etiquetas.

Ej. *beqz $1,* ***sigue***
…
sigue: …

E. Pseudodirecto: Utiliza una constante de 26 bits para especificar bits menos significativos de la dirección. Los 6 bits más significativos que faltan de los 32 que posee una dirección no cambian (permanecen intactos en el registro PC y es por ello por lo que este modo se considera *pseudo*).

Este modo de direccionamiento se emplea exclusivamente para indicar la dirección de saltos en las instrucciones de salto incondicional sin registro (instrucciones *j* y *jal*). Al igual que en modo de direccionamiento anterior, los entornos de programación ocultan este detalle mediante etiquetas:

Ej. *j* ***sigue***
…
sigue*:* …

2.3. INSTRUCCIONES:

La estructura general de una instrucción en MIPS es:

<etiqueta> <instrucción> <operandos> <comentarios>

A. Etiqueta: identifica simbólicamente la dirección de memoria donde se encuentra la instrucción. Resulta de interés especialmente en las instrucciones de salto.

Las etiquetas pueden contener:

- Cualquier secuencia de letras, números y guion bajo ('_')
- No pueden comenzar por un número.
- Debe finalizar con dos puntos (':')

Ej. *repite:* *_bucle:* *bucle_ppal:* ~~*3error:*~~

B. Comentarios: añade información útil al usuario pero irrelevante para el compilador (el compilador lo elimina durante la compilación). Los comentarios deben comenzar con punto y coma (';') y finalizar con el fin de línea (esto es, son comentarios de línea). Este campo es opcional.

Ej. *add $1,$2,$3* ***; calcula el total***

C. Operandos: el número de operandos depende del tipo de instrucción y su número puede oscilar entre 0 y 3. Salvo excepciones, el primero de ellos actúa como destino y el resto como fuente.

2.4. REPERTORIO DE INSTRUCCIONES

El Anexo 2 muestra el juego de instrucciones MIPS soportado por el simulador. A continuación, se muestra una clasificación de las instrucciones atendiendo al tipo de operación que realizan:

A. Instrucciones ALU: realizan operaciones aritméticas y lógicas (siempre de 32 bits). Estas instrucciones suelen emplear dos o tres operandos según sea una operación unaria o binaria, y los modos de direccionamiento registro e inmediato con ciertas restricciones pero en ningún caso operandos en memoria.

Para una misma operación (por ejemplo, *sumar*) suele haber varias instrucciones que realicen la misma operación pero con ciertas variaciones:

- Con o sin signo: Las instrucciones sin signo emplean el sufijo u (de *unsigned*).
- Entre registros o con una contante: Las que emplean una constante utilizan el sufijo i (de *immediate*)

Por ejemplo, para la suma se dispone de las instrucciones:

add $1,$2,$3 ; $1 ← $2 + $3 Puede provocar desbordamiento
*add**u** $1,$2,$3* ; $1 ← $2 + $3 Nunca provoca desbordamiento
*add**i** $1,$2,25* ; $1 ← $2 + 25 La constante de 16 bits extiende el signo hasta 32 bits
*add**iu** $1,$2,25*; $1 ← $2 + 25 También extiende el signo pero no provoca desbordamiento

Pero hay muchas operaciones ALU que no tienen todas las variantes:

- En las operaciones lógicas (OR, XOR, AND) no existen versiones sin signo, pues no tiene sentido en operaciones lógicas. En este caso, las versiones que operan con constantes de 16 bits (ORI, XORI, ANDI) extienden con ceros hasta 32 bits.
- En multiplicaciones y divisiones no existe el direccionamiento inmediato por razones de implementación. Además, la mayoría de ellas guardan el resultado sobre los registros HI y LO.

La instrucción de suma conjuntamente con el registro $0 (que siempre vale 0) pueden utilizarse para inicializar el contenido de los registros a una determinada constante de 16 bits o para realizar transferencias entre registros. Para inicializar con una constante de 32 bits utilice la *pseudoinstrucción li* que se describe más adelante.

Ej. *addiu $10,$0,365* ; $10 ← 365

add $10,$0,$15 ; $10 ← $15

También se dispone de instrucciones de comparación de operandos. Estas instrucciones establecen el registro destino a "1" (verdadero) o "0" (falso) en función del resultado de la comparación de dos operandos fuente. Estas instrucciones suelen utilizarse conjuntamente con las de salto condicional para definir cualquier condición de salto (se verá más adelante).

Por ejemplo, la instrucción de comparación "menor que" *slt* (del inglés: *set if less than*) sería

Ej. *slt $10,$11,$12* ; if ($11<$12) $10=1; else $10=0;

De ésta también hay diferentes versiones según compare con o sin signo, o si compara dos registros o un registro con un inmediato (*slt, sltu, slti, sltiu*)

Finalmente, el procesador también dispone de instrucciones de desplazamiento y rotación. Tales instrucciones pueden ser a su vez lógicas o aritméticas (ésta última sólo utilizada en desplazamientos a la derecha).

Ej. *sll $1,$2,5* ; Desplaza $2 a la izquierda 5 bits insertando ceros por la derecha.
sllv $1,$2,$3 ; Como el anterior pero el nº de desplazamiento viene dado por $3
sra $1,$2,5 ; Desplaza $2 a la derecha 5 bits insertando el signo por la izquierda

B. Instrucciones de carga y almacenamiento*:* son instrucciones para la transferencia de datos entre registro y memoria. Las instrucciones de carga (*load*) leen de memoria y almacenan en un registro y las de almacenamiento (*store*) leen de un registro y lo almacena en memoria. Todas ellas utilizan dos operandos, uno de ellos emplea el modo de direccionamiento a registro *reg* y otro el modo de direccionamiento a memoria mediante registro base más desplazamiento *inm(reg)*.

Tanto de carga como de almacenamiento hay múltiples versiones según el tamaño de la transferencia de datos. El tamaño del dato puede ser de 8, 16, 32 ó 64 bits, por lo que se tienen los siguientes especificadores de longitud de acceso:

DoubleWord: transferencias de 64 bits (una doble palabra)
Word: transferencias de 32 bits (una palabra. La palabra del procesador es de 32 bits)
HalfWord: " de 16 bits (media palabra)
Byte: " de 8 bits (un byte).

Además, las instrucciones de carga de menos de 32 bits pueden ser con o sin signo según se extiendan en el registro destino con el signo o con ceros (versión sin signo)

Algunos ejemplos:

ldc1 $f2, 32($2)	**(load doubleword coprocessor 1)** Carga en los registros $f2 y $f3 los 64 bits leídos de la dirección de memoria $2+32
lw $1, 32($2)	**(load word)** Carga en $1 una palabra de 32 bits leída de la dirección de memoria $2+32
lbu $1, 32($2)	**(load byte unsigned)** Lee byte leído de la dirección de memoria $2+32, extiende sin signo el byte a 32 bits y lo guarda en $1
sh $1, 32($2)	**(store halfword)** Guarda a partir de la dirección $2+32 los 16 bits más bajos del registro $1

Los accesos a memoria deben estar alineados al tamaño de la transferencia, esto es, los accesos de tipo word deben estar alineados a 32 bits o los de tipo half a 16 bits.

C. Instrucciones de salto condicional: son instrucciones que saltan (modifican el flujo de ejecución del programa) si se cumple cierta condición entre dos registros o con el cero. Las más utilizadas son:

beq $1,$2,etiqueta ; si $1 es igual a $2 salta a etiqueta
bne $1,$2,etiqueta ; si $1 es distinto de $2 salta a etiqueta
beqz $1, etiqueta ; si $1 es igual a 0 $2 salta a etiqueta (es una pseudoinstrucción)
bnez $1, etiqueta ; si $1 no es igual a 0 $2 salta a etiqueta (es una pseudoinstrucción)

Existen otras para comparaciones de desigualdad con cero: *bgez,bgtz,blez,bltz*

Y otras más que son *pseudoinstrucciones* formadas por una instrucción de comparación tipo *set* y una instrucción de salto condicional. Hay disponibles versiones con signo (*bge,bgt,ble,blt*) y sin signo (*bgeu,bgtu,bleu,bltu*).

Ej. *bgeu $1,$2,etiqueta* → sltu $1,$1,$2
*beq $1,$0,e*tiqueta

D. Instrucciones de salto incondicional: aquellas en las que el salto no está sometido a condiciones. De éstas se distinguen varios tipos según el modo de direccionamiento empleado para especificar el destino del salto (registro o pseudodirecto) y si el salto guarda la dirección de retorno (instrucciones de salta y enlaza *jump and link*). Los entornos de programación utilizan etiquetas para el direccionamiento *pseudodirecto*.

```
Ej.   j etiqueta        ; Salto a etiqueta.
      jal etiqueta      ; Salto a etiqueta con salva de la dirección de retorno en $31
      jr $1             ; Salto mediante registro.
      jalr $1           ; Salto mediante registro con salva de la dirección de retorno en $31
```

E. Otras instrucciones:

la: (de las siglas de *load address*) es una *pseudoinstrucción* que almacena la dirección de memoria referenciada por una etiqueta en un registro.

```
Ej.   .data
      valores:    .word  10,-20,30,40
      .text
                  .la $1, valores    ; carga en $1 la dirección apuntada por valores
```

li: (de las siglas *load immediate*) *pseudoinstrucción* que guarda una constante de 32 bits (*inm32*) en un registro.

Ej. *li $1, 0xAABBCCDD*

2.5. ESTRUCTURA DE UN PROGRAMA EN ENSAMBLADOR PARA MIPS

En esencia, un programa en ensamblador se encarga de describir qué debe cargarse en la sección de la memoria reservada para instrucciones (programa a ejecutar) y qué datos (con sus respectivos valores) tendrá almacenada inicialmente la parte de la memoria reservada para datos. A cada una de estas sesiones se las denomina segmento; teniendo entonces dos segmentos:

- El segmento de código (dirección de memoria donde se cargarán las instrucciones). Se marca inicialmente con la directiva ".text".
- El segmento de datos (establecer los valores iniciales de datos almacenados en memoria). Se marca inicialmente con la directiva ".data".

Además, los compiladores disponen de un conjunto de directivas que permiten configurar ciertas características y acciones del proceso de compilación. El Anexo 3 muestra una tabla con las directivas del compilador más empleadas. A continuación, se muestra una estructura típica de un programa en ensamblador con un ejemplo:

Ej.

```
.data

valores:  .word 23, 65, -320, 125, 0xFF, -10, 299, 322, 237,-12
res:      .space 4
.text
          addiu $1,$0,10
          add $10,$0,$0
          la $2, valores    ; carga la dirección
sigue:    lw $5, 0($2)
          add $10,$10,$5
          addiu $2,$2,4
          addi $1,$1,-1
          bne $1,$0, sigue
          sw $10, 0($2)
```

3. FUNCIONAMIENTO BÁSICO DEL SIMULADOR VISUALMIPS

VisualMips32 es un simulador de MIPS para Windows que permite fundamentalmente visualizar la ejecución segmentada, la memoria y los registros del procesador durante la simulación de un programa en ensamblador. La aplicación se encuentra disponible en la plataforma de enseñanza virtual y no precisa instalación alguna. En esta sesión de laboratorio sólo se introducirán aquellas secciones del simulador necesarias para el desarrollo de la práctica.

El simulador se compone fundamentalmente de siete ventanas que el usuario puede mantener de forma independiente de la ventana principal, acoplarla a la ventana principal agrupadas en forma de pestañas o subdividiendo el área de otras ya acopladas. Al iniciar la aplicación las ventanas se encuentran distribuidas como muestra la figura 2.

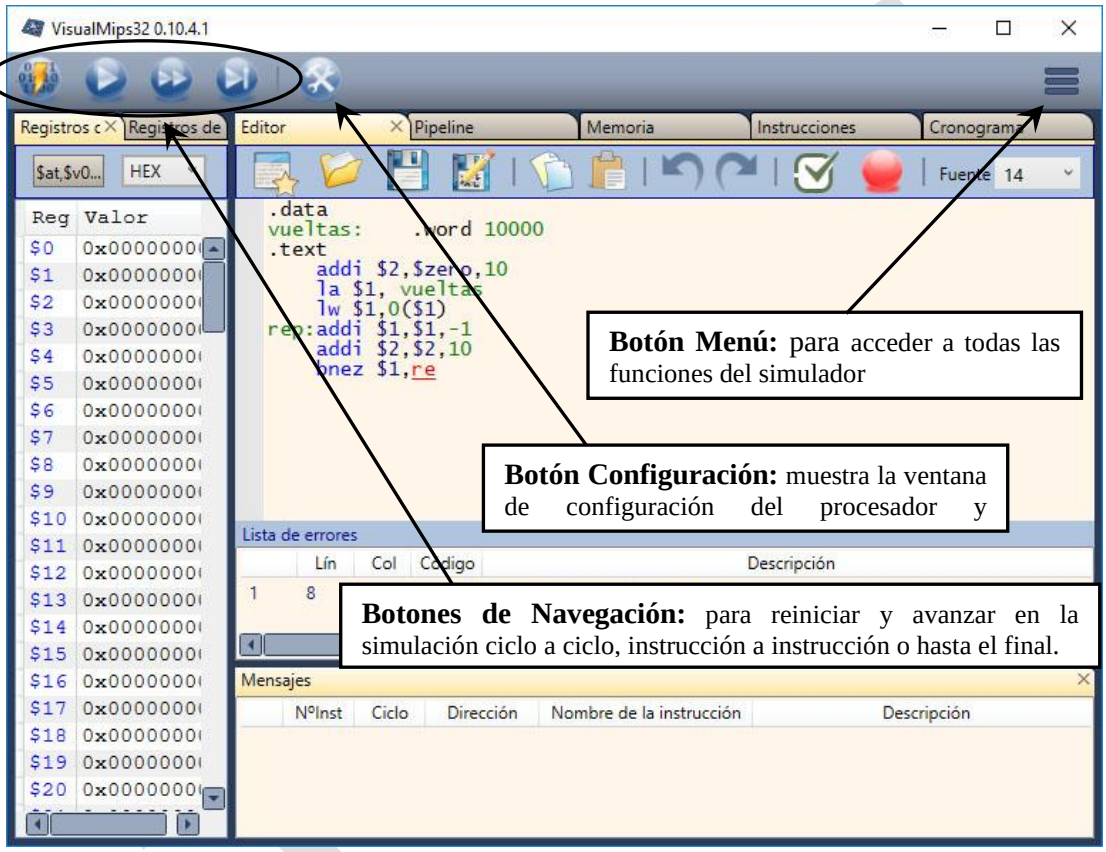

Figura 2. Visión general del simulador

A. Ventana *Editor*: utilizada principalmente para cargar, editar y compilar el código en ensamblador a simular. Los programas se guardan con la extensión *.asm*

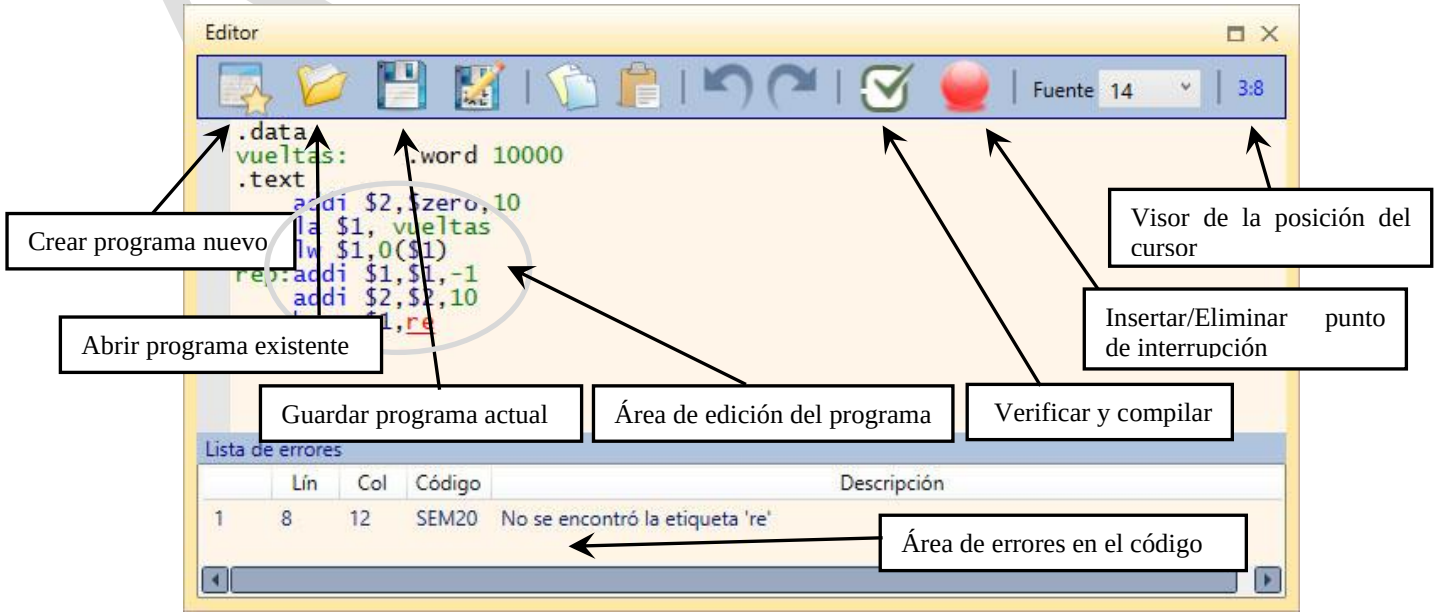

Figura3. Ventana Editor

El botón ***Verificar y Compilar*** detecta los errores del código, si la opción *Autocompilar* de la ventana de configuración está activa (por defecto lo está) los errores se mostrarán automáticamente en la ventana *mensajes*. Esta ventana permite además establecer los puntos de interrupción (*breakpoints*) antes de comenzar la simulación. Al comenzar la simulación, los puntos de interrupción se transfieren a la ventana *Instrucciones*.

B. Ventana *Mensajes*: lista los errores del código durante su compilación y las excepciones generadas por el procesador durante la ejecución.

C. Ventana *Memoria*: muestra y permite editar el contenido de la memoria tanto del segmento de código como del segmento de datos (la región destinada al segmento del *kernel* no es accesible). Cuando se escribe en memoria, bien sea por el usuario o el procesador, la ventana muestra en rojo el último dato que ha sido modificado.

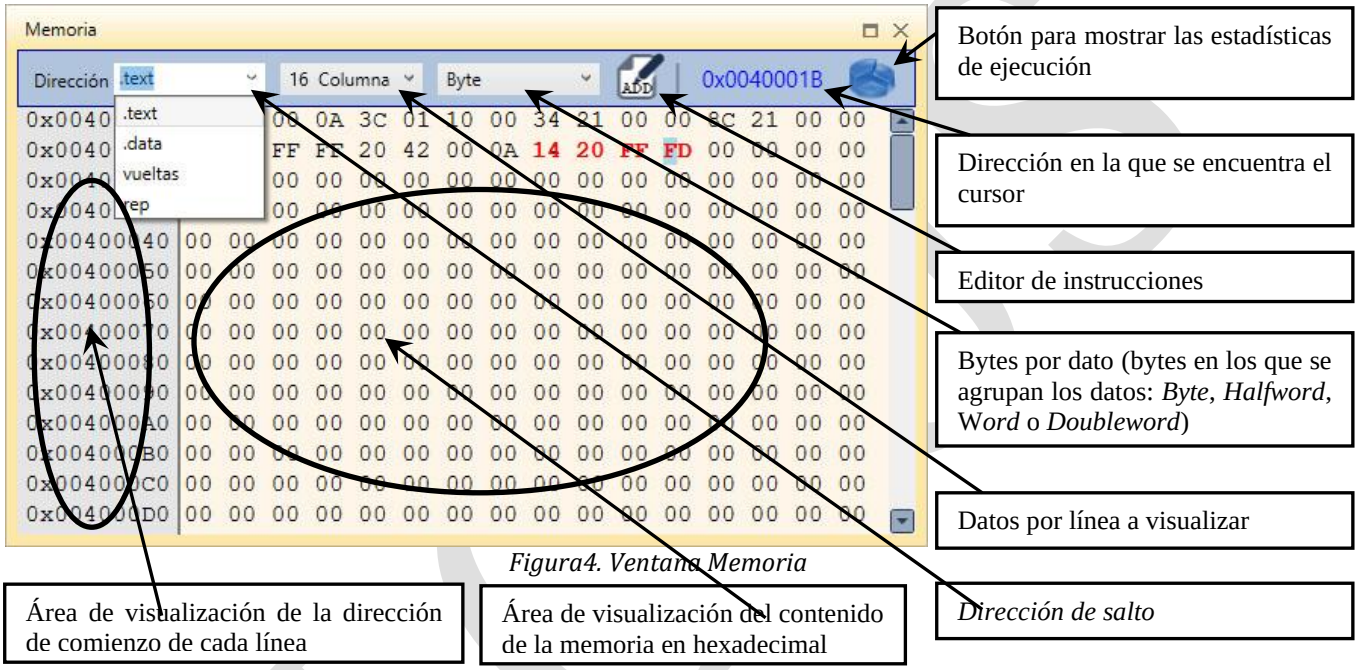

Figura4. Ventana Memoria

El elemento *Dirección de salto* que aparece en la *figura 4* permite al usuario acceder rápidamente a una zona determinada de la memoria. En ella puede indicarse:

- Una dirección de memoria en hexadecimal. Ej. *0x00400000*
- El nombre de un segmento. Por ejemplo, para ir al comienzo del segmento de código hay que escribir *.text* o para el de datos especificar *.data*
- El nombre de una variable definida en el programa. En el ejemplo de la *figura 4 vueltas* o *rep* .

D. Ventanas de *Registros*: consiste en dos ventanas para visualizar y modificar el contenido de los registros del banco de registros de enteros y los del banco de coma flotante.

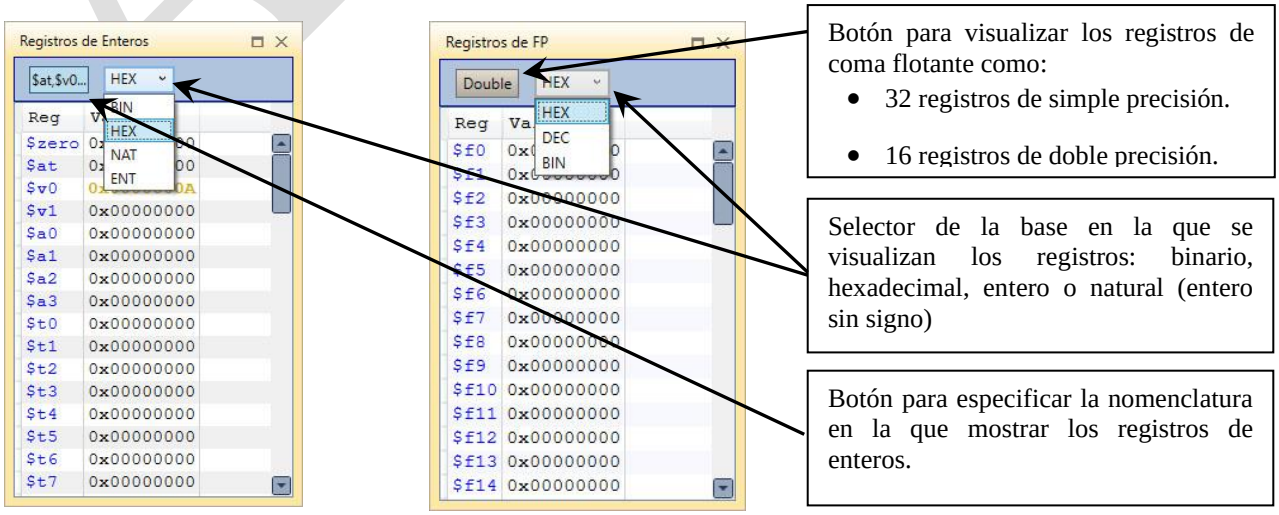

Figura5. Ventanas de Registros

E. Ventana *Instrucciones*: muestra la codificación y la interpretación de las instrucciones almacenadas en el segmento de código así como las fases en las que se encuentra cada instrucción. Esta ventana permite además editar tales instrucciones y establecer puntos de interrupción durante la ejecución.

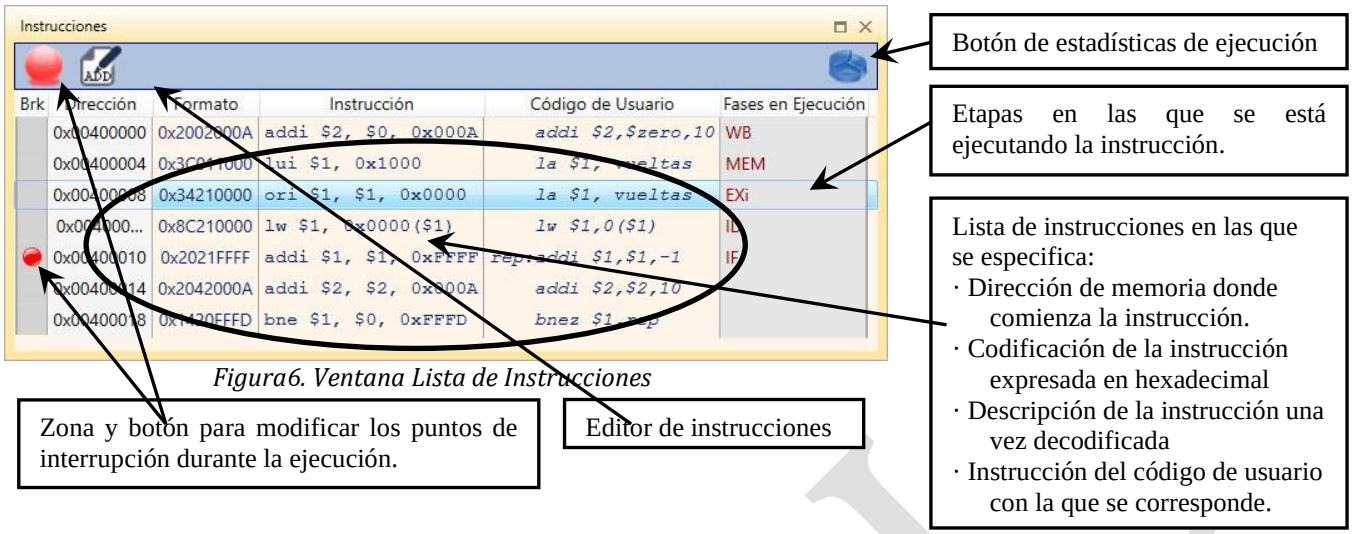

Figura6. Ventana Lista de Instrucciones

Algunas características de las ventanas anteriores, así como las ventanas *Pipeline* y *cronograma* que se describen a continuación, se verán con más detalle en el boletín de la próxima práctica.

F. Ventana *Pipeline*: representa un esquema de las etapas del procesador y muestra en cada ciclo de ejecución las instrucciones que se encuentran en cada etapa y su estado (si la instrucción ha ejecutado la fase, qué tipo de bloqueo se ha producido, etc.). No es motivo de uso durante esta sesión, así que se verá en profundidad en próximas sesiones.

G. Ventana *Cronograma*: muestra un cronograma de ejecución, el estado de cada etapa al final de cada ciclo. No es motivo de uso durante esta sesión, así que se verá en profundidad en próximas sesiones.

4. EJERCICIOS A REALIZAR EN ESTA SESIÓN:

En esta sesión de laboratorio, el alumno tendrá a su disposición una hoja de ejercicios que le permitirá profundizar en los conceptos iniciales del MIPS, además de aprender a manejar el entorno de simulación.

Es importante indicar que NO ES OBLIGATORIO realizar estos ejercicios, aunque es SUMAMENTE RECOMENDABLE y el alumno puede acudir a tutoría de su profesor de práctica para consultar cualquier duda relacionada con esta sesión.

De igual forma, para la segunda sesión se necesitará que el alumno tenga un manejo básico de la herramienta y conozca el lenguaje ensamblador del MIPS. De hecho, en el estudio previo de la segunda sesión, se realizarán consultas acerca de los conceptos básicos que el alumno ha debido adquirir durante esta primera sesión.

5. ANEXOS

ANEXO 1: Nomenclatura y características de los registros de enteros

Nombre	Nombre alternativo	Registro	Uso	¿Se conserva?*
$zero	$0	0	Constante 0	---
$at	$1	1	Reservado para el ensamblador	---
$v0-$v1	$2-3	2-3	Valores de retorno	no
$a0-$a3	$4-7	4-7	Argumentos	sí
$t0-$t7	$8-15	8-15	Temporales	no
$s0-$s7	$16-23	16-23	Valores Guardados	sí
$t8-$t9	$24-25	24-25	Temporales	no
$k0-$k1	$16-27	16-27	Reservados por el kernel del Sistema Operativo	---
$gp	$28	28	Puntero Global	sí
$sp	$29	29	Puntero de Pila	sí
$fp	$30	30	Puntero de Marco	sí
$ra	$31	31	Dirección de retorno	sí

al realizar una llamada a subrutina

ANEXO 2: Repertorio de instrucciones soportadas por el simulador

IMPORTANTE: Las instrucciones que trabajen con coma flotante no serán utilizadas en esta asignatura (denotadas en color rojo), pero se detallan porque el simulador lo soporta.

Instrucciones de transferencia de datos		
Instrucción	**F***	**Descripción**
lb reg,inm16(reg)	I	Carga un byte desde memoria a un registro de enteros (extiende signo)
lbu reg,inm16(reg)	I	Carga un byte desde memoria a un registro de enteros (extiende ceros)
lh reg,inm16(reg)	I	Carga 16 bits desde memoria a un registro de enteros (extiende signo)
lhu reg,inm16(reg)	I	Carga 16 bits desde memoria a un registro de enteros (extiende ceros)
lw reg,inm16(reg)	I	Carga 32 bits desde memoria a un registro de enteros
sb reg,inm16(reg)	I	Almacena un byte en memoria desde un registro de enteros
sh reg,inm16(reg)	I	Almacena 16 bits en memoria desde un registro de enteros
sw reg,inm16(reg)	I	Almacena 32 bits en memoria desde un registro de enteros
lwc1 freg,inm16(reg)	I	Carga 32 bits desde memoria a un registro de coma flotante
ldc1 freg,inm16(reg)	I	Carga 64 bits desde memoria a un registro de coma flotante doble precisión
swc1 freg,inm16(reg)	I	Almacena 32 bits en memoria desde un registro de coma flotante
sdc1 freg,inm16(reg)	I	Almacena 32 bits en memoria desde un registro FP doble precisión
mfc1 reg,freg	FR	Copia 32 bits desde un registro de coma flotante a un registro de enteros
mtc1 reg,freg	FR	Copia 32 bits desde un registro de enteros a un registro de coma flotante
mflo reg	R	Copia 32 bits desde el registro especial LO a un registro de enteros
mfhi reg	R	Copia 32 bits desde el registro especial HI a un registro de enteros
lui reg,inm16	I	Carga una constante de 16 bits en la parte alta de un registro de enteros
li reg,imm32	P	Carga una constante de 32 bits en un registro de enteros
la reg,etiq	P	Carga la dirección especificada por una etiqueta en un registro de enteros

Instrucciones lógicas y aritméticas para enteros		
Instrucción	**F***	**Descripción**
add reg,reg,reg	R	Suma de enteros con signo (puede provocar desbordamiento)

addu reg,reg,reg	R	Suma de enteros (sin desbordamiento)
addi reg,reg,inm16	I	Suma de enteros con inmediato (puede provocar desbordamiento)
addiu reg,reg,inm16	I	Suma de enteros con inmediato (sin desbordamiento)
sub reg,reg,reg	R	Resta de enteros (puede provocar desbordamiento)
subu reg,reg,reg	R	Resta de enteros (sin desbordamiento)
mult reg,reg	R	Multiplicación de enteros con signo. Guarda el resultado de 64 bits en HI y LO
multu reg,reg	R	Multiplicación de enteros sin signo. Guarda el resultado de 64 bits en HII y LO
div reg,reg	R	División de enteros con signo. Guarda el cociente en LO y el resto en HI
divu reg,reg,reg	R	División de enteros sin signo. Guarda el cociente en LO y el resto en HI
and reg,reg,reg	R	Operación lógica AND entre registros
andi reg,reg,inm16	I	Operación lógica AND entre registro y constante
nor reg,reg,reg	R	Operación lógica NOR entre registros.
or reg,reg,reg	R	Operación lógica OR entre registros.
ori reg,reg,inm16	I	Operación lógica OR entre registro y constante
xor reg,reg,reg	R	Operación lógica OR exclusiva entre registros.
xori reg,reg,inm16	I	Operación lógica OR exclusiva entre registro y constante
sll reg,reg,inm16	R	Desplazamiento lógico a la izquierda
srl reg,reg,inm16	R	Desplazamiento lógico a la derecha (inserta ceros por la izquierda)
sra reg,reg,inm16	R	Desplazamiento aritmético a la derecha (inserta el signo por la izquierda)
sllv reg,reg,reg	R	Desplazamiento lógico a la izquierda de una cantidad variable por registro
srlv reg,reg,reg	R	Desplazamiento lógico a la derecha de una cantidad variable por registro
srav reg,reg,reg	R	Desplazamiento aritmético a la derecha de una cantidad variable por registro
rol reg,reg,reg	P	Rotación a la izquierda de una cantidad variable por registro
ror reg,reg,reg	P	Rotación a la derecha de una cantidad variable por registro
slt reg,reg,reg	R	Comparación "menor que" con signo entre registros. "1" si *true*, "0" si *false*
sltu reg,reg,reg	R	Comparación "menor que" sin signo entre registros. "1" si *true*, "0" si *false*
slti reg,reg,inm16	I	Comparación "menor que" con signo con constante. "1" si *true*, "0" si *false*
sltiu reg,reg,inm16	I	Comparación "menor que" sin signo con constante. "1" si *true*, "0" si *false*
nop	P	No operación. Representa a la instrucción *sll $0,$0,0*

Instrucciones de Control		
Instrucción	**F***	**Descripción**
beq reg,reg,etiq	I	Salto de 16 bits relativo al PC (*etiq*) si ambos registros son iguales
bne reg,reg,etiq	I	Salto de 16 bits relativo al PC (*etiq*) si ambos registros son distintos
beqz reg,etiq	P	Salto de 16 bits relativo al PC (*etiq*) si el registro es igual a "0"
bnez reg,etiq	P	Salto de 16 bits relativo al PC (*etiq*) si el registro es distinto de "0"
bltz reg,etiq	I	Salto de 16 bits relativo al PC (*etiq*) si el registro es menor que"0"
blez reg,etiq	I	Salto de 16 bits relativo al PC (*etiq*) si el registro es menor o igual a "0"
bgtz reg,etiq	I	Salto de 16 bits relativo al PC (*etiq*) si el registro es mayor que "0"
bgez reg,etiq	I	Salto de 16 bits relativo al PC (*etiq*) si el registro es mayor o igual a "0"
bgt reg,reg,etiq	P	Salto de 16 bits relativo al PC (*etiq*) si es mayor con signo
bge reg,reg,etiq	P	Salto de 16 bits relativo al PC (*etiq*) si es mayor o igual con signo
blt reg,reg,etiq	P	Salto de 16 bits relativo al PC (*etiq*) si es menor con signo
ble reg,reg,etiq	P	Salto de 16 bits relativo al PC (*etiq*) si es menor o igual con signo
bgtu reg,reg,etiq	P	Salto de 16 bits relativo al PC (*etiq*) si es mayor sin signo
bgeu reg,reg,etiq	P	Salto de 16 bits relativo al PC (*etiq*) si es mayor o igual sin signo
bltu reg,reg,etiq	P	Salto de 16 bits relativo al PC (*etiq*) si es menor sin signo
bleu reg,reg,etiq	P	Salto de 16 bits relativo al PC (*etiq*) si es menor o igual sin signo
j etiq	J	Salto incondicional a una dirección constante de 26 bits (*etiq*)
jal etiq	J	Salto incondicional a una dirección constante de 26 bits (*etiq*) con retorno.
jr reg	R	Salto incondicional a una dirección especificada por registro
jalr reg	R	Salto incondicional a una dirección especificada por registro con retorno.

Instrucciones en coma flotante		
Instrucción	**F***	**Descripción**
add.s freg,freg,freg	FR	Suma en coma flotante de simple precisión
sub.s freg,freg,freg	FR	Resta en coma flotante de simple precisión
mul.s freg,freg,freg	FR	Multiplicación en coma flotante de simple precisión
div.s freg,freg,freg	FR	División en coma flotante de simple precisión
abs.s freg,freg	FR	Valor absoluto en coma flotante de simple precisión
add.s freg,freg,freg	FR	Suma en coma flotante de doble precisión
sub.s freg,freg,freg	FR	Resta en coma flotante de doble precisión
mul.s freg,freg,freg	FR	Multiplicación en coma flotante de doble precisión
div.s freg,freg,freg	FR	División en coma flotante de doble precisión
abs.d freg,freg	FR	Valor absoluto en coma flotante de doble precisión

* La columna F especifica el formato de la instrucción (tipo I, R, J o FR) o si es pseudoinstrucción (P).

ANEXO 3: Directivas más comunes del compilador

Directiva	Descripción	Ejemplo
.text <dirección>	Comienzo de una zona de código en el programa. La dirección es opcional y si no se especifica comenzará al principio del segmento de código o a continuación de la última zona de código definida	*.text* *.text 0x004500000*
.data <dirección>	Como el anterior pero para la zona de datos.	*.data* *.data 0x12000000*
.space <nbytes>	Reserva un espacio de <nbytes> bytes en memoria.	*.space 25*
.byte <val1>,<val2>,…	Almacena en memoria una lista de valores de enteros de 8 bits (con o sin signo)	*.byte 20, -128, 255*
.half <val1>,<val2>,…	Igual que el anterior de 16 bits	*.half -32768, 65535*
.word <val1>,<val2>,…	Igual que el anterior de 32 bits	*.word 50, 0xFFFFFFFF*
.float <val1>,<val2>,…	Almacena en memoria una lista de valores en coma flotante de simple precisión	*.float 3.1415*
.double <val1>,<val2>,…	Almacena en memoria una lista de valores en coma flotante de doble precisión	*.double 2.728182*
.ascii <cadena>	Almacena en memoria una cadena de caracteres encerrados entre comillas dobles	*.ascii "Hola mundo"*
.asciiz <cadena>	Almacena en memoria una cadena de caracteres siendo el último carácter la marca de fin de cadena (carácter ascii 0)	*.asciiz "Mi mensaje"*