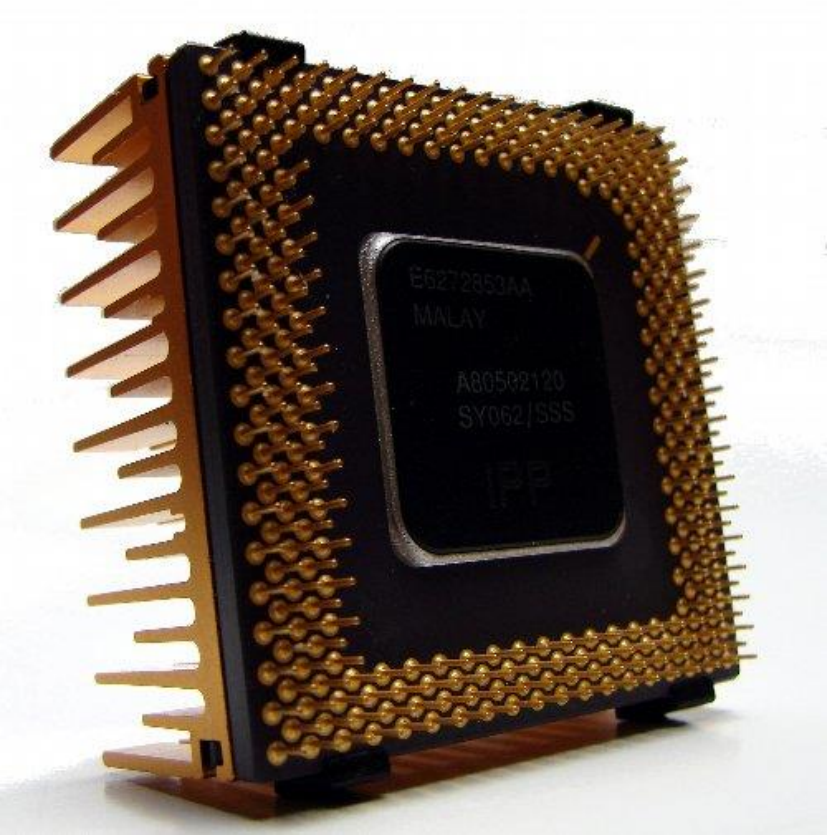
ANEXOS MATERIAL INDIVIDUAL F-III

PROYECTO



2020

Anexos: Simulador WinMIPS64

Arquitectura de los Computadores
Grado en Ingeniería Informática
Dpto. Tecnología Informática y Computación
Universidad de Alicante

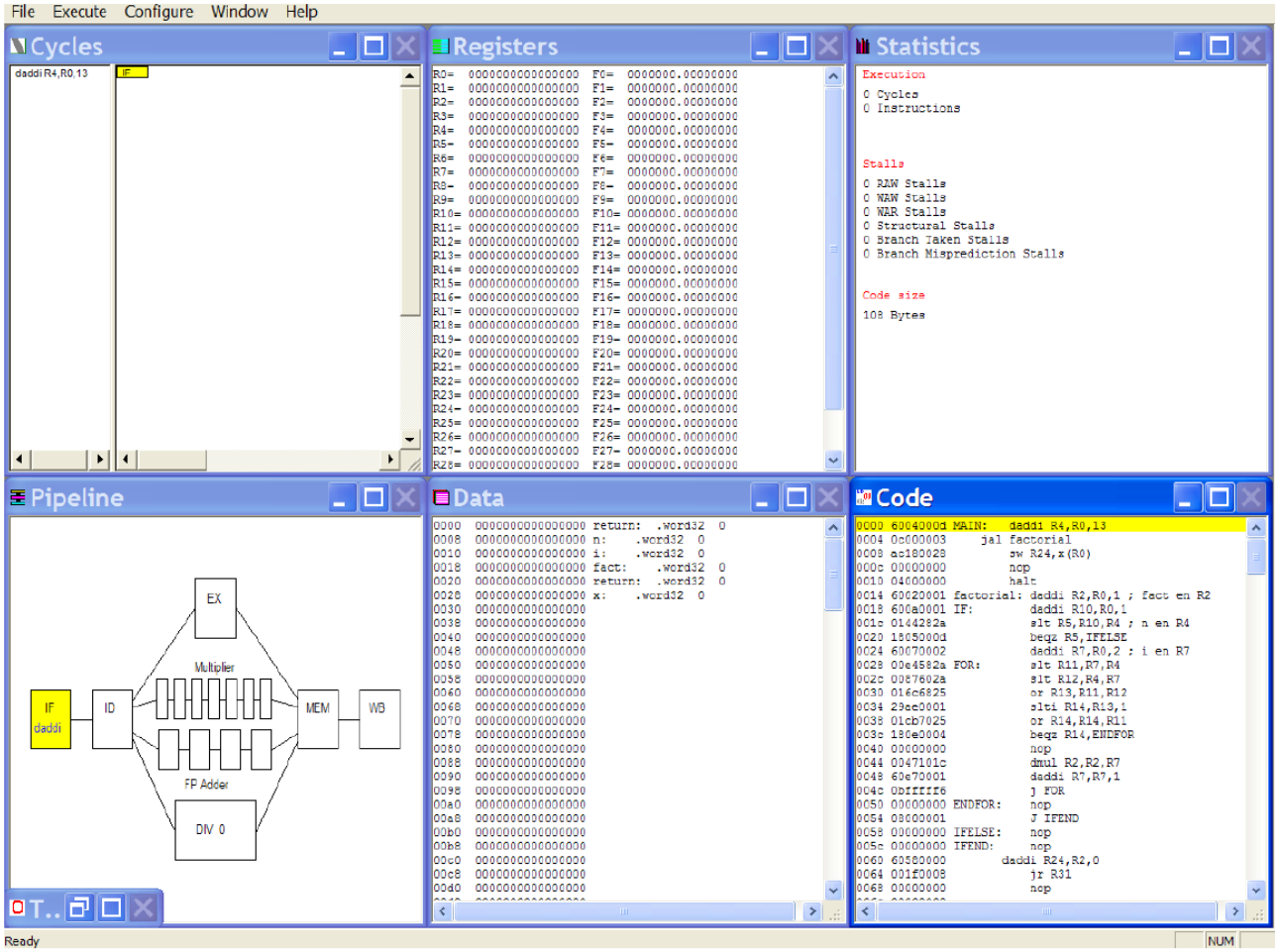
Anexos Material individual de la Fase III

SIMULADOR WINMIPS64

I. ANEXO 1

Sobre el simulador

El simulador WinMIPS64 únicamente funciona en entornos Microsoft Windows. Para arrancar el simulador únicamente hay que clicar sobre el icono correspondiente. El aspecto del simulador es:



Consta de siete ventanas y la barra de entado:

1. **Pipeline**: En esta ventana se encuentra la estructura del cauce con las etapas y las diferentes unidades funcionales disponible. Cuando se ejecuta un programa muestra la instrucción que se halla en cada etapa.

2. **Code**: Esta ventana muestra el contenido del segmento de código de memoria. La primera columna indica la dirección de memoria, en la segunda columna aparece el código máquina de la instrucción y en la tercera se muestra la instrucción en ensamblador. Si no se indicó otra cosa, el simulador carga el programa a partir de la dirección 0 del segmento de código.

3. **Data**: En esta ventana se muestra el contenido del segmento de datos. El contenido de la memoria se muestra en palabras de 64 bits. El simulador utiliza ordenación little-endian.

4. **Register:** Esta ventana muestra el contenido de los registros de propósito general (R0 a R31) y de coma flotante (F0-F31). El contenido de los registros puede modificarse en cualquier momento al clicar sobre su nombre.

5. **Cycles:** Esta ventana muestra la evolución del flujo del programa a lo largo del tiempo dentro del cauce. Cada etapa tiene asignado un color y un nombre.

6. **Statistics**: En esta ventana se muestra información sobre el último programa ejecutado (Ciclos de reloj consumidos por la CPU, instrucciones ejecutadas, paradas clasificadas por tipo de riesgo, etc.).

7. **Terminal:** Esta ventana simula el comportamiento de un terminal de E/S y permite a los programas leer y escribir información desde/hacia el exterior.

8. **Barra de estado:** Durante la ejecución de un programa proporciona información útil sobre el estado de la simulación.

Todos los valores numéricos mostrados en las anteriores ventanas están expresados en hexadecimal.

Cuando se tenga editado un programa en un fichero, se puede proceder a su carga en el simulador con la opción ***Open*** del menú ***File***. Una vez cargado, las ventanas de código (Code) y datos (Data) aparecen rellenas a partir de la información que contenía el fichero que se acaba de cargar

La ejecución de un programa puede realizarse de tres formas: Ciclo-a-Ciclo (Single Cycle), Multi-Ciclo (Multi-Cycle) y Breakpoint (Run-to).

Ciclo-a-Ciclo: Realiza la ejecución del programa ciclo a ciclo, lo que permite ver con gran detalle el funcionamiento del cauce de segmentación. Se ha de utilizar la opción ***Single cycle*** del menú ***Execute*** o mediante la tecla ***F7***.

Multi-ciclo: La ejecución del programa avanza cinco ciclos de reloj (valor por defecto que se puede modificar con la opción ***Multi Step*** del menú ***Configure*** o bien al pulsar la combinación de teclas ***ctrl..+T***). Se ha de utilizar la opción ***Multi Cycle*** del menú ***Execute*** o la tecla ***F8***.

El modo Breakpoint ejecuta el programa hasta que encuentra un punto de parada o hasta el final del programa (la instrucción *halt* normalmente es la última y tiene por objeto detener el procesador). Los puntos de parada se insertan al clicar sobre una instrucción en la ventana de código. La instrucción se colorea de azul para indicar que se ha insertado un punto de parada. Clicar de nuevo sobre la instrucción elimina el punto de parada. Para utilizar este modo de ejecución hay que acceder a la opción ***Run to*** del menú ***Execute*** o a la tecla de ***F4***.

En cualquier momento de la simulación podemos reiniciar el simulador mediante las opciones ***Reset MIPS64*** (simula un reset hardware) y ***Full Reset*** (además borra el contenido de memoria) del menú ***File***.

Instrucciones en coma flotante

El simulador WinMIPS64 permite ejecutar instrucciones en coma flotante. Si las operaciones en coma flotante se ejecutaran en un único ciclo de reloj, entonces se necesitaría un ciclo de reloj con una duración muy larga. Para evitar esta situación, la segmentación permite para las operaciones en coma flotante una latencia mayor. Se entiende como latencia al número de ciclos que hay entre una instrucción que produce un resultado y una instrucción que lo utiliza. Con esta definición una operación ALU entera tiene una latencia 0 puesto que el resultado puede utilizarse en el siguiente ciclo de reloj. La latencia de la segmentación será esencialmente un ciclo menos que la profundidad del cauce de ejecución, aunque en el simulador WinMIPS64 simplifica esta definición y asimila la latencia como número de ciclos de reloj del cauce de ejecución.

En el WinMIPS64 hay cuatro unidades funcionales separadas:

1. La unidad principal entera que manipula todas las cargas y almacenamientos, todas las operaciones ALU enteras (excepto la multiplicación y división) y los saltos.
2. El multiplicador de coma flotante y entero.
3. El sumador en coma flotante que ejecuta operaciones de suma en coma flotante, resta y operaciones de conversión
4. El divisor en coma flotante y entero.

En la figura 1 se muestra una estructura segmentada que soporta varias operaciones en coma flotante similar a la del simulador WinMips64.

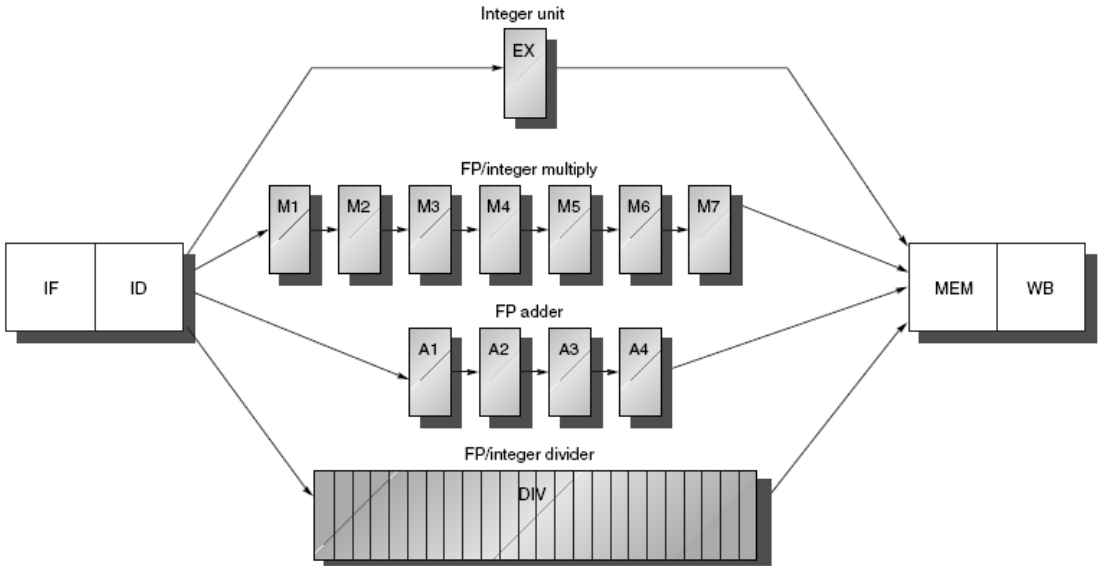


Figura 1. Estructura segmentada con varias unidades de ejecución.

Esta estructura segmentada permite la salida de múltiples operaciones de coma flotante. El multiplicador y sumador de coma flotante están completamente segmentados y tiene tienen un profundidad de siete y cuatro etapas respectivamente. El divisor en coma flotante no está segmentado pero requiere 24 ciclos de reloj para completarse. La latencia en instrucciones entre la emisión de una operación en coma flotante y la utilización del resultado de la operación sin incurrir en una parada por dependencia de datos, se determinará por el número de ciclos que se gastan en las etapas ejecución. Por ejemplo, la cuarta instrucción después de una suma en coma flotante puede utilizar el resultado de una suma en coma flotante. Para las operaciones ALU enteras, la profundidad de la segmentación de ejecución es siempre uno y la siguiente instrucción puede utilizar el resultado.

En la segmentación de esta ruta de datos, se han tenido en cuenta algunos aspectos adicionales debido a la presencia de cauces con grandes latencias:

- Puesto que la unidad de división no está completamente segmentada, pueden ocurrir riesgos estructurales. La ruta de datos segmentada requiere de mecanismos para detectar estos riesgos y detener las instrucciones a emitir.
- Debido a que las instrucciones pueden variar en los tiempos de ejecución, el número de escrituras en registros en un ciclo pueden ser mayores que uno.
- Pueden aparecer riesgos por dependencias de datos WAW (write after write), es decir existe la posibilidad de riesgos por realizar escrituras en orden incorrecto. Esto se debe a que las instrucciones más cortas no alcanzan la etapa WB en el orden que debería ser el correcto.
- Cabe la posibilidad de que se terminen instrucciones en diferente orden al que fueron emitidas causando problemas.
- Debido a las latencias mas largas de las operaciones, las paradas por riesgos de dependencias de datos RAW (read after write), es decir, riesgos por intentar leer datos antes que se escriban, serán más frecuentes.

Configuración del simulador

El simulador WinMIPS64 puede configurarse de muchas maneras. Se puede cambiar la estructura y los requerimientos de tiempo de la unidad segmentada de coma flotante y el tamaño de la memoria para los datos y para el código. Para ver o cambiar los valores estándar o por defecto, hay seleccionar la opción ***Architecture*** del menú ***Configure*** **,** aparecerá entonces la ventana de la figura 2.:

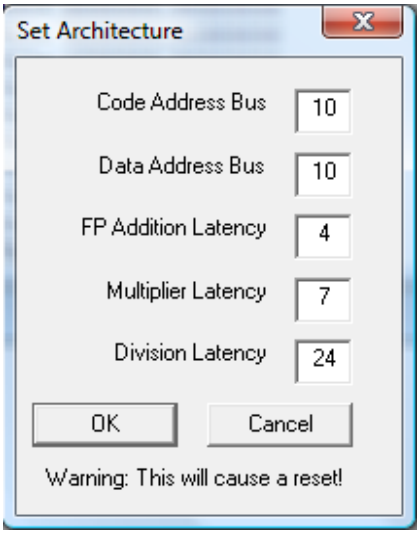


Figura 2. Ventana que aparece con la opción ***Architecture*** del menú ***Configure***.

Se pueden cambiar las opciones pinchando y editando el campo deseado. Cualquier cambio que se realice en las latencias de la coma flotante se verá reflejado en la ventana de pipeline. El ***Code Address Bus*** y el ***Data Address Bus*** se refieren al número de bits del bus de direcciones y el bus de datos respectivamente. Cualquier cambio en los bits de los buses se verá reflejado en la ventana ***Code Widow*** o ***Data Windows***. Al finalizar cualquier cambio se debe pulsar sobre loa opción OK para volver a la ventana principal.

Otras opciones del menú ***Configure*** permiten modificar aspectos de ejecución en la máquina segmentada: Por ejemplo se puede permitir o no la anticipación de datos mediante la activación o desactivación de la opción ***Enable forwarding*** y se puede permitir o no el salto retardado activando o desactivando la opción ***Enable Delay Slot***.

Sobre el programa ensamblador.

El programa en ensamblador consta de dos secciones: en la primera se definen los datos (.data) y en la segunda se definen las instrucciones (.code).

La definición de los datos consiste en reservar espacio de memoria para las variables y asignarles un nombre simbólico para referenciarlas de manera más sencilla en el código. El nombre de la variable irá seguida de dos puntos. El ensamblador proporciona el siguiente conjunto de directivas para ello:

```
.data                    - comienzo del segmento de datos
.text                    - comienzo del segmento de código.
```

```
.code                   - comienzo del segmento de código (igual que .text)
.org <n>                - comienzo de una dirección
.space <n>              - reserva n bytes vacíos
.asciiz <s>             - introduce una cadena ascii terminada con cero
.ascii <s>              - introduce una cadena ascii
.align <n>              - alinea a un límite de n bytes
.word <n1>,<n2>..       - introduce palabras de datos (64 bits)
.byte <n1>,<n2>..       - introduce bytes
.word32 <n1>,<n2>..     - introduce números de 32 bits
.word16 <n1>,<n2>..     - introduce números de 16 bits
.double <n1>,<n2>..     - introduce números en coma flotante
```

Ejemplo:

```
      .data
A:    .word 10
B:    .word 8
C:    .word 0
CR:   .word32 0x10000
DR:   .word32 0x10008
```

La directiva ".data" indica el comienzo de la declaración de variables. Por ejemplo la directiva .word32 indica que la variable ocupara 32 bits de memoria. El identificador que precede a la directiva es nombre simbólico asignado a la variable seguido de ":".

Las constantes numéricas pueden expresarse en base decimal o bien en hexadecimal anteponiendo al valor los caracteres (0x).

El código (las instrucciones) comienza tras la directiva ".text" o ".code"). Las primeras columnas de la línea se reservan para situar una etiqueta como posible destino de una instrucción de salto.

Ejemplo:

```
      .data
i:    .word32 0

      .text

      daddi R2,R0,0 ;
      daddi R5,R0,5 ; comentario

WHILE: slt R6,R2,R5
      beqz R6,ENDWHILE
       daddi R2,R2,1
       sw R2,i(R0)
      j WHILE
ENDWHILE: nop
halt
```

También pueden aparecer comentarios, se considera como tal cualquier texto que aparezca a la derecha del carácter ";".

Terminal de salida

El simulador soporta un dispositivo sencillo de entrada/salida que funciona como un terminal con capacidades simples de gráficos. Para este propósito el simulador WinMIPS64 utiliza entrada/salida mapeada en memoria. Utilizando esta capacidad, podemos mostrar la salida de un programa en esta ventana.

La acción a realizar en el terminal se determina escribiendo la función de control (*CONTROL*) deseada en una determinada posición de memoria. Se utiliza otra posición de memoria para escribir o leer un dato en o desde la pantalla (*DATA*).

Las direcciones para las funciones de control (*CONTROL*) y para el dato (*DATA*) son:

```
CONTROL: .word32 0x10000
DATA: .word32 0x10008
```

Los valores de *CONTROL* y su función son:

CONTROL	FUNCIÓN
1	Poner en DATA el entero sin signo a aparecer en pantalla.
2	Poner en DATA el entero con signo a aparecer en pantalla.
3	Poner en DATA el valor en coma flotante a aparecer en pantalla.
°4	Poner en DATA la dirección de comienzo de la cadena a aparecer en pantalla.
5	Poner en DATA+5 la coordenada X, en DATA+4 la coordenada Y y en DATA el color RGB a aparecer en pantalla.
6	Borra toda la pantalla del terminal.
7	Borra la ventana de gráficos.
8	Lee del teclado y pone en DATA el valor (un entero o un valor en coma flotante))
9	Lee un byte desde DATA, sin caráter echo.

Ejemplo:

```
.data
     A: .word 10
     B: .word 8
     C: .word 0
     CR: .word32 0x10000
     DR: .word32 0x10008

.text
     main:
     ld r4,A(r0)
     ld r5,B(r0)
     dadd r3,r4,r5
     sd r3,C(r0)
     lwu r1,CR(r0) ;Registro de control
     lwu r2,DR(r0) ;Registro de datos
     daddi r10,r0,1
     sd r3,(r2) ;Salida del registro r3
     sd r10,(r1) ;en el terminal de salida
     halt
```

Después de ejecutarse el programa se mostrara el resultado de la suma en decimal en la ventana terminal.

II. ANEXO 2

Repertorio de instrucciones soportado por el simulador WinMIPS64:

Instrucciones de Transferencia de Datos		
lb	r_d, Inm(r_i)	Copia en r_d un byte (8 bits) desde la dirección (Inm+r_i) (con extensión del signo)
lbu	r_d, Inm(r_i)	Copia en r_d un byte (8 bits) desde la dirección (Inm+r_i) (sin extensión del signo)
sb	r_f, Inm(r_i)	Guarda los 8 bits menos significativos de r_f en la dirección (Inm+r_i)
lh	r_d, Inm(r_i)	Copia en r_d un half-word (16 bits) desde la dir. (Inm+r_i) (con extensión del signo)
lhu	r_d, Inm(r_i)	Copia en r_d un half-word (16 bits) desde la dir. (Inm+r_i) (sin extensión del signo)
sh	r_f, Inm(r_i)	Guarda los 16 bits menos significativos de r_f a partir de la dirección (Inm+r_i)
lw	r_d, Inm(r_i)	Copia en r_d un word (32 bits) desde la dir. (Inm+r_i) (con extensión del signo)
lwu	r_d, Inm(r_i)	Copia en r_d un word (32 bits) desde la dir. (Inm+r_i) (sin extensión del signo)
sw	r_f, Inm(r_i)	Guarda los 32 bits menos significativos de r_f a partir de la dirección (Inm+r_i)
ld	r_d, Inm(r_i)	Copia en r_d un double word (64 bits) desde la dirección (Inm+r_i)
sd	r_f, Inm(r_i)	Guarda r_f a partir de la dirección (Inm+r_i)
l.d	f_d, Inm(r_i)	Copia en r_d un valor en punto flotante (64 bits) desde la dirección (Inm+r_i)
s.d	f_f, Inm(r_i)	Guarda f_f a partir de la dirección (Inm+r_i)
mov.d	f_d, f_f	Copia el valor del registro f_f al registro f_d
mtc1	r_f, f_d	Copia los 64 bits del registro entero r_f al registro f_d de punto flotante
mfc1	r_d, f_f	Copia los 64 bits del registro f_f de punto flotante al registro r_d entero
cvt.d.l	f_d, f_f	Convierte a punto flotante el valor entero copiado al registro f_f, dejándolo en f_d
cvt.l.d	f_d, f_f	Convierte a entero el valor en punto flotante contenido en f_f, dejándolo en f_d

Instrucciones Aritméticas		
dadd	r_d, r_f, r_g	Suma r_f con r_g, dejando el resultado en r_d (valores con signo)
daddi	r_d, r_f, N	Suma r_f con el valor inmediato N, dejando el resultado en r_d (valores con signo)
daddu	r_d, r_f, r_g	Suma r_f con r_g, dejando el resultado en r_d (valores sin signo)
daddui	r_d, r_f, N	Suma r_f con el valor inmediato N, dejando el resultado en r_d (valores con signo)
add.d	f_d, f_f, f_g	Suma f_f con f_g, dejando el resultado en f_d (en punto flotante)
dsub	r_d, r_f, r_g	Resta r_g a r_f, dejando el resultado en r_d (valores con signo)
dsubu	r_d, r_f, r_g	Resta r_g a r_f, dejando el resultado en r_d (valores sin signo)
sub.d	f_d, f_f, f_g	Resta f_g a f_f, dejando el resultado en f_d (en punto flotante)
dmul	r_d, r_f, r_g	Mutiplica r_f con r_g, dejando el resultado en r_d (valores con signo)
dmulu	r_d, r_f, r_g	Mutiplica r_f con r_g, dejando el resultado en r_d (valores sin signo)
mul.d	f_d, f_f, f_g	Multiplica f_f con f_g, dejando el resultado en f_d (en punto flotante)
ddiv	r_d, r_f, r_g	Divide r_f por r_g, dejando el resultado en r_d (valores con signo)
ddivu	r_d, r_f, r_g	Divide r_f por r_g, dejando el resultado en r_d (valores sin signo)
div.d	f_d, f_f, f_g	Divide f_f por f_g, dejando el resultado en f_d (en punto flotante)
slt	r_d, r_f, r_g	Compara r_f con r_g, dejando r_d=1 si r_f es menor que r_g (valores con signo)
slti	r_d, r_f, N	Compara r_f con el valor inmediato N, dejando r_d=1 si r_f es menor que N (valores signo)
c.lt.d	f_d, f_f	Compara f_f con f_g, dejando flag FP=1 si f_f es menor que f_g (en punto flotante)
c.le.d	f_d, f_f	Compara f_f con f_g, dejando flag FP=1 si f_f es menor o igual que f_g (en punto flotante)
c.lq.d	f_d, f_f	Compara f_f con f_g, dejando flag FP=1 si f_f es igual que f_g (en punto flotante)

Instrucciones Lógicas		
and	r_d, r_f, r_g	Realiza un AND entre r_f y r_g (bit a bit), dejando el resultado en r_d
andi	r_d, r_f, N	Realiza un AND entre r_f y el valor inmediato N (bit a bit), dejando el resultado en r_d
or	r_d, r_f, r_g	Realiza un OR entre r_f y r_g (bit a bit), dejando el resultado en r_d
ori	r_d, r_f, N	Realiza un OR entre r_f y el valor inmediato N (bit a bit), dejando el resultado en r_d
xor	r_d, r_f, r_g	Realiza un XOR entre r_f y r_g (bit a bit), dejando el resultado en r_d
xori	r_d, r_f, N	Realiza un XOR entre r_f y el valor inmediato N (bit a bit), dejando el resultado en r_d

Instrucciones de desplazamiento de bits		
dsll	r_d, r_f, N	Desplaza a izquierda N veces los bits del registro r_f, dejando el resultado en r_d
dsllv	r_d, r_f, r_N	Desplaza a izquierda r_N veces los bits del registro r_f, dejando el resultado en r_d
dsrl	r_d, r_f, N	Desplaza a derecha N veces los bits del registro r_f, dejando el resultado en r_d
dsrlv	r_d, r_f, r_N	Desplaza a derecha r_N veces los bits del registro r_f, dejando el resultado en r_d
dsra	r_d, r_f, N	Igual que **dsrl** pero mantiene el signo del valor desplazado
dsrav	r_d, r_f, r_N	Igual que **dsrlv** pero mantiene el signo del valor desplazado

Instrucciones de Transferencia de Control		
j	offN	Salta a la dirección rotulada offN
jal	offN	Salta a la dirección rotulada offN y copia en r_{31} la dirección de retorno
jr	r_d	Salta a la dirección contenida en el registro r_d
beq	r_d, r_f, offN	Si r_d es igual a r_f, salta a la dirección rotulada offN
bne	r_d, r_f, offN	Si r_d no es igual a r_f, salta a la dirección rotulada offN
beqz	r_d, offN	Si r_d es igual a 0, salta a la dirección rotulada offN
bnez	r_d, offN	Si r_d no es igual a 0, salta a la dirección rotulada offN
bc1f	offN	Salta a la dirección rotulada offN si flag FP=1 (ó true) (en punto flotante)
bc1t	offN	Salta a la dirección rotulada offN si flag FP=0 (ó false) (en punto flotante)

Instrucciones de Control	
nop	Operación nula
halt	Detiene el simulador