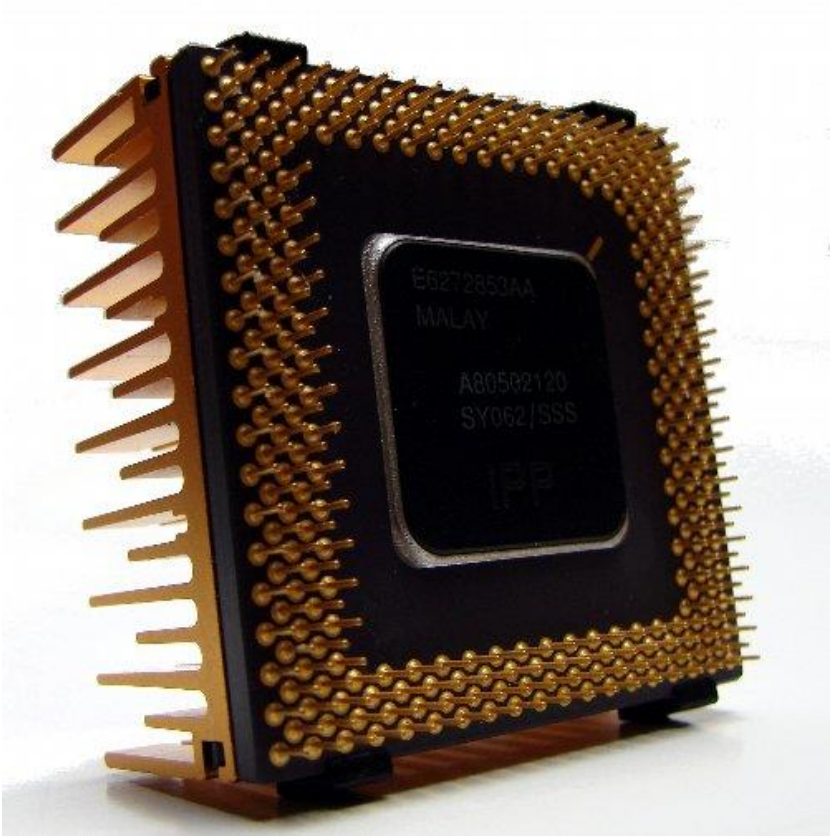
ANNEXOS MATERIAL INDIVIDUAL F-III

PROJECTE



2020

Annexos: Simulador WinMIPS64

Arquitectura dels Computadors
Grau en Enginyeria Informàtica
Dpto. Tecnologia Informàtica i Computació
Universitat d'Alacant

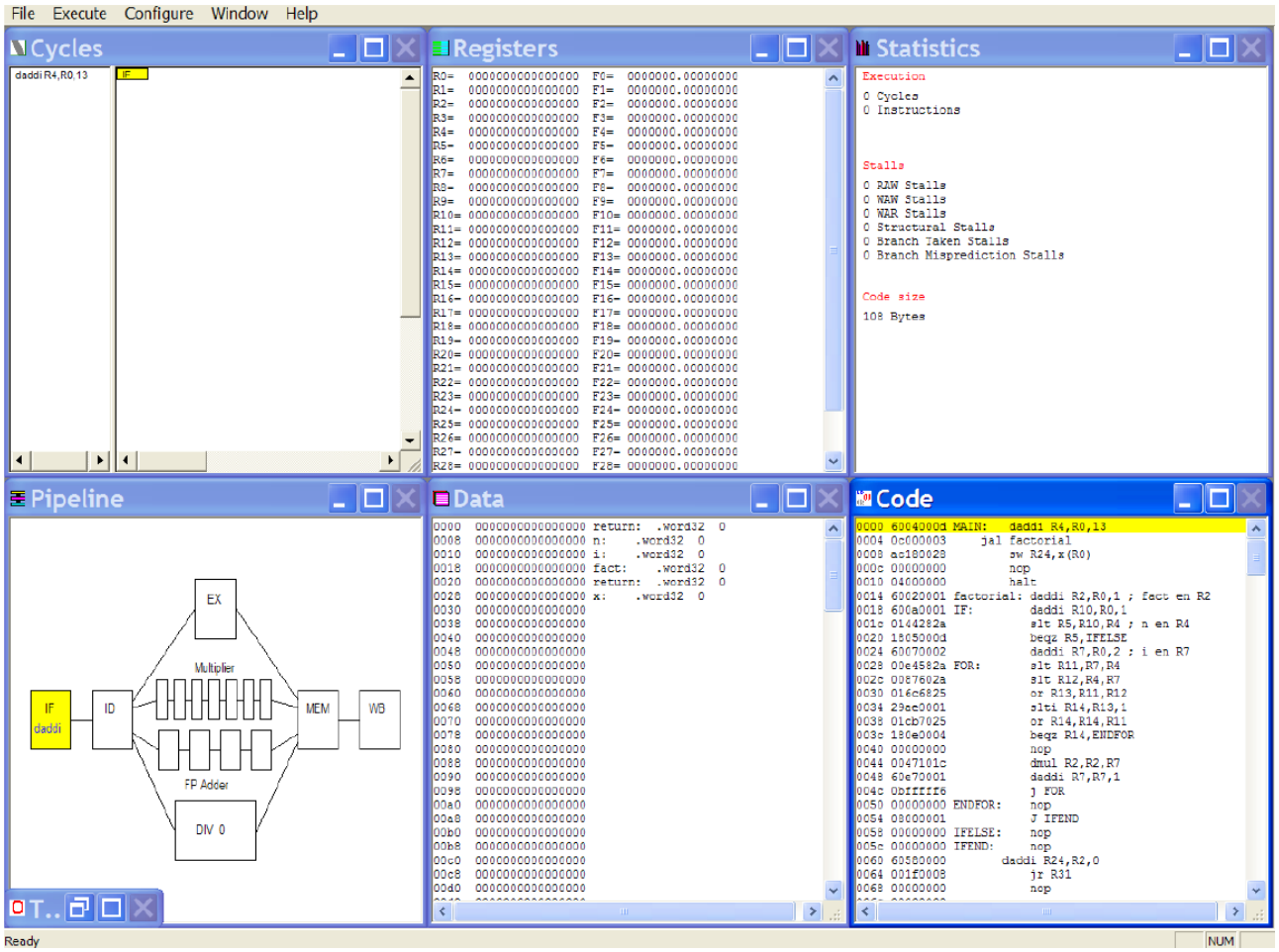
Annexos Material individual de la Fase III

SIMULADOR WINMIPS64

I. ANNEX 1

Sobre el simulador

El simulador WinMIPS64 únicament funciona en entorns Microsoft Windows. Per arrencar el simulador únicament cal clicar sobre la icona corresponent. L'aspecte del simulador és:



Consta de set finestres i la barra d'estat:

1. **Pipeline**: En aquesta finestra es troba l'estructura de la llera amb les etapes i les diferents unitats funcionals disponible. Quan s'executa un programa mostra la instrucció que es troba en cada etapa.

2. **Code**: Aquesta finestra mostra el contingut del segment de codi de memòria. La primera columna indica l'adreça de memòria, en la segona columna apareix el codi màquina de la instrucció i en la tercera es mostra la instrucció en assemblador. Si no es va indicar una altra cosa, el simulador carrega el programa a partir de l'adreça 0 del segment de codi.

3. **Data**: En aquesta finestra es mostra el contingut del segment de dades. El contingut de la memòria es mostra en paraules de 64 bits. El simulador utilitza ordenació little-endian.

4. **Register:** Aquesta finestra mostra el contingut dels registres de propòsit general (R0 a R31) i de coma flotant (F0-F31). El contingut dels registres pot modificar-se a qualsevol moment en clicar sobre el seu nom.

5. **Cycles:** Aquesta finestra mostra l'evolució del flux del programa al llarg del temps dins de la llera. Cada etapa té assignat un color i un nom.

6. **Statistics**: En aquesta finestra es mostra informació sobre l'últim programa executat (Cicles de rellotge consumits per la CPU, instruccions executades, parades classificades per tipus de risc, etc.).

7. **Terminal:** Aquesta finestra simula el comportament d'un terminal d'E/E i permet als programes llegir i escriure informació des de/cap a l'exterior.

8. **Barra d'estat:** Durant l'execució d'un programa proporciona informació útil sobre l'estat de la simulació.

Tots els valors numèrics mostrats en les anteriors finestres estan expressats en hexadecimal.

Quan es tingui editat un programa en un fitxer, es pot procedir a la seva càrrega en el simulador amb l'opció ***Open*** del menú ***File***. Una vegada carregat, les finestres de codi (Code) i dades (Data) apareixen emplenades a partir de la informació que contenia el fitxer que s'acaba de carregar

L'execució d'un programa pot realitzar-se de tres formes: Cicle-a-Cicle (Single Cycle), Multi-Cicle (Multi-Cycle) i Breakpoint (Run-to).

Cicle-a-Cicle: Realitza l'execució del programa cicle a cicle, la qual cosa permet veure amb gran detall el funcionament de la llera de segmentació. S'ha d'utilitzar l'opció ***Single cycle*** del menú ***Execute*** o mitjançant la tecla ***F7***.

Multi-cicle: L'execució del programa avança cinc cicles de rellotge (valor per defecte que es pot modificar amb l'opció ***Multi Step*** del menú ***Configure*** o bé en prémer la combinació de tecles ***ctrl..+T***). S'ha d'utilitzar l'opció ***Multi Cycle*** del menú ***Execute*** o la tecla ***F8***.

El mode Breakpoint executa el programa fins que troba un punt de parada o fins al final del programa (la instrucció *halt* normalment és l'última i té per objecte detenir el processador). Els punts de parada s'insereixen en clicar sobre una instrucció en la finestra de codi. La instrucció s'acoloreix de blava per indicar que s'ha inserit un punt de parada. Clicar de nou sobre la instrucció elimina el punt de parada. Per utilitzar aquest mode d'execució cal accedir a l'opció ***Run to*** del menú ***Execute*** o a la tecla ***de F4***.

En qualsevol moment de la simulació podem reiniciar el simulador mitjançant les opcions ***Reset MIPS64*** (simula un reset maquinari) i ***Full Reset*** (a més esborra el contingut de memòria) del menú ***File***.

Instruccions en coma flotant

El simulador WinMIPS64 permet executar instruccions en coma flotant. Si les operacions en coma flotant s'executessin en un únic cicle de rellotge, llavors es necessitaria un cicle de rellotge amb una durada molt llarga. Per evitar aquesta situació, la segmentació permet per a les operacions en coma flotant una latència major. S'entén com a latència al nombre de cicles que hi ha entre una instrucció que produeix un resultat i una instrucció que ho utilitza. Amb aquesta definició una operació ALU sencera té una latència 0 ja que el resultat pot utilitzar-se en el següent cicle de rellotge. La latència de la segmentació serà essencialment un cicle menys que la profunditat de la llera d'execució, encara que en el simulador WinMIPS64 simplifica aquesta definició i assimila la latència com a nombre de cicles de rellotge de la llera d'execució.

En el WinMIPS64 hi ha quatre unitats funcionals separades:

1. La unitat principal sencera que manipula totes les càrregues i emmagatzematges, totes les operacions ALU senceres (excepte la multiplicació i divisió) i els salts.
2. El multiplicador de coma flotant i sencer.
3. El sumador en coma flotant que executa operacions de suma en coma flotant, resta i operacions de conversió
4. El divisor en coma flotant i sencer.

En la figura 1 es mostra una estructura segmentada que suporta diverses operacions en coma flotant similar a la del simulador WinMips64.

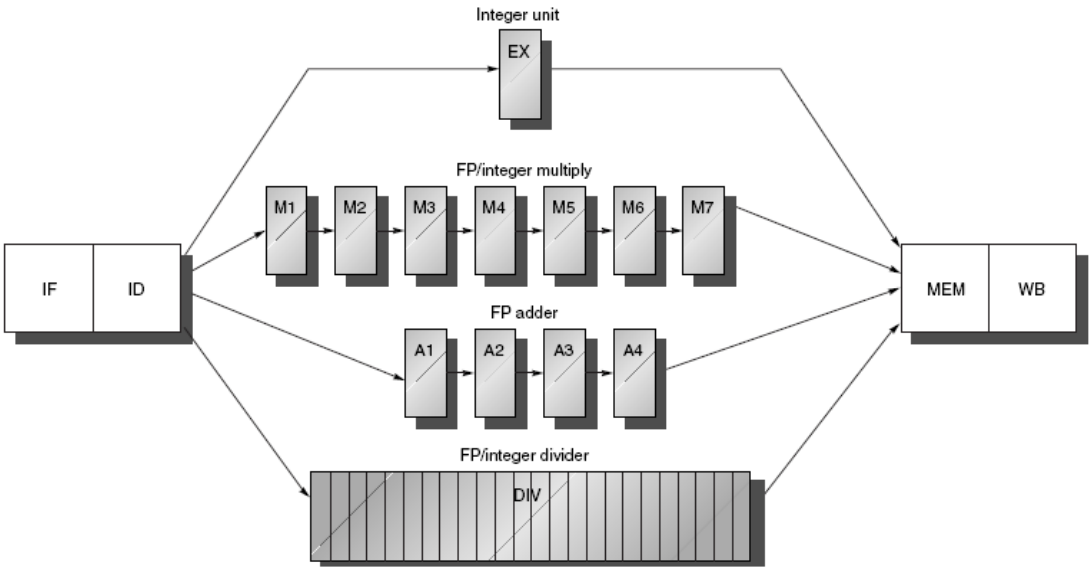


Figura 1. Estructura segmentada amb diverses unitats d'execució.

Aquesta estructura segmentada permet la sortida de múltiples operacions de coma flotant. El multiplicador i sumador de coma flotant estan completament segmentats i té tenen un profunditat de set i quatre etapes respectivament. El divisor en coma flotant no està segmentat però requereix 24 cicles de

rellotge per completar-se. La latència en instruccions entre l'emissió d'una operació en coma flotant i la utilització del resultat de l'operació sense incórrer en una parada per dependència de dades, es determinarà pel nombre de cicles que es gasten en les etapes execució. Per exemple, la quarta instrucció després d'una suma en coma flotant pot utilitzar el resultat d'una suma en coma flotant. Per a les operacions ALU senceres, la profunditat de la segmentació d'execució és sempre un i la següent instrucció pot utilitzar el resultat.

En la segmentació d'aquesta ruta de dades, s'han tingut en compte alguns aspectes addicionals a causa de la presència de lleres amb grans latències:

- ja que la unitat de divisió no està completament segmentada, poden ocórrer riscos estructurals. La ruta de dades segmentada requereix de mecanismes per detectar aquests riscos i detenir les instruccions a emetre.
- A causa que les instruccions poden variar en els temps d'execució, el nombre d'escriptures en registres en un cicle poden ser majors que un.
- Poden aparèixer riscos per dependències de dades WAW (write after write), és a dir existeix la possibilitat de riscos per realitzar escriptures en ordre incorrecte. Això es deu al fet que les instruccions més curtes no aconsegueixen l'etapa WB en l'ordre que hauria de ser el correcte.
- Cap la possibilitat que s'acabin instruccions en diferent ordre al que van ser emeses causant problemes.
- A causa de les latències mes llargues de les operacions, les parades per riscos de dependències de dades RAW (read after write), és a dir, riscos per intentar llegir dades abans que s'escriguin, seran més freqüents.

Configuració del simulador

El simulador WinMIPS64 pot configurar-se de moltes maneres. Es pot canviar l'estructura i els requeriments de temps de la unitat segmentada de coma flotant i la grandària de la memòria per a les dades i per al codi. Per veure o canviar els valors estàndard o per defecte, hi ha seleccionar l'opció ***Architecture*** del menú ***Configure*** , apareixerà llavors la finestra de la figura 2.:

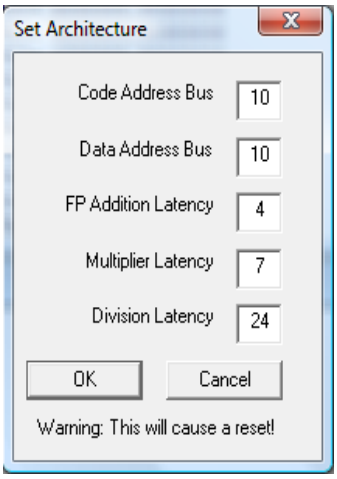


Figura 2. Finestra que apareix amb l'opció ***Architecture*** del menú ***Configure***.

Es poden canviar les opcions punxant i editant el camp desitjat. Qualsevol canvi que es realitzi en les latències de la coma flotant es veurà reflectit en la finestra de pipeline. El ***Code Address Bus*** i el ***Data Address Bus*** es refereixen al nombre de bits del bus d'adreces i el bus de dades respectivament. Qualsevol canvi en els bits dels busos es veurà reflectit en la finestra ***Code Widow*** o ***Data Windows***. En finalitzar qualsevol canvi s'ha de prémer sobre lloa opció OK per tornar a la finestra principal.

Altres opcions del menú ***Configure*** permeten modificar aspectes d'execució en la màquina segmentada: Per exemple es pot permetre o no l'anticipació de dades mitjançant l'activació o desactivació de l'opció ***Enable forwarding*** i es pot permetre o no el salt retardat activant o desactivant l'opció ***Enable Delay Slot***.

Sobre el programa assemblador.

El programa en assemblador consta de dues seccions: en la primera es defineixen les dades (.data) i en la segona es defineixen les instruccions (.code).

La definició de les dades consisteix a reservar espai de memòria per a les variables i assignar-los un nom simbòlic per referenciar-les de manera més senzilla en el codi. El nom de la variable anirà seguida de dos punts. L'assemblador proporciona el següent conjunt de directives per a això:

```
.data                   - començament del segment de dades
.text                   - començament del segment de codi.
.code                   - començament del segment de codi (igual que .text)
.org <n>                - començament d'una adreça
.space <n>              - reserva n bytes buits
.asciiz <s>             - introdueix una cadena ascii acabada amb zero
.ascii <s>              - introdueix una cadena ascii
.align <n>              - alinea a un límit de n bytes
.word <n1>,<n2>..       - introdueix paraules de dades (64 bits)
.byte <n1>,<n2>..       - introdueix bytes
.word32 <n1>,<n2>..     - introdueix nombres de 32 bits
.word16 <n1>,<n2>..     - introdueix nombres de 16 bits
.double <n1>,<n2>..     - introdueix nombres en coma flotant
```

Exemple:

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

La directiva ".data" indica el començament de la declaració de variables. Per exemple la directiva .word32 indica que la variable ocupés 32 bits de memòria. L'identificador que precedeix a la directiva és nom simbòlic assignat a la variable seguit de ":".

Les constants numèriques poden expressar-se en base decimal o bé en hexadecimal anteposant al valor els caràcters (0x).

El codi (les instruccions) comença després de la directiva ".text" o ".code"). Les primeres columnes de la línia es reserven per situar una etiqueta com a possible destinació d'una instrucció de salt.

Exemple:

```
      .data
i:    .word32 0

      .text

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

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

També poden aparèixer comentaris, es considera com tal qualsevol text que aparegui a la dreta del caràcter ";".

Terminal de sortida

El simulador suporta un dispositiu senzill d'entrada/sortida que funciona com un terminal amb capacitats simples de gràfics. Per a aquest propòsit el simulador WinMIPS64 utilitza entrada/sortida mapejada en memòria. Utilitzant aquesta capacitat, podem mostrar la sortida d'un programa en aquesta finestra.

L'acció a realitzar en el terminal es determina escrivint la funció de control (*CONTROL*) desitjada en una determinada posició de memòria. S'utilitza una altra posició de memòria per escriure o llegir una dada en o des de la pantalla (*DATA*).

Les adreces per a les funcions de control (*CONTROL*) i per a la dada (*DATA*) són:

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

Els valors de *CONTROL* i la seva funció són:

CONTROL	FUNCIÓ
1	Posar en DATA l'enter sense signe a aparèixer en pantalla.
2	Posar en DATA l'enter amb signe a aparèixer en pantalla.
3	Posar en DATA el valor en coma flotant a aparèixer en pantalla.
°4	Posar en DATA l'adreça de començament de la cadena a aparèixer en pantalla.
5	Posar en DATA+5 la coordenada X, en DATA+4 la coordenada I i en DATA el color RGB a aparèixer en pantalla.

6	Esborra tota la pantalla del terminal.
7	Esborra la finestra de gràfics.
8	Llegeix del teclat i posa en DATA el valor (un sencer o un valor en coma flotant))
9	Llegeix un byte des de DATA, sense caràcter echo.

Exemple:

```
.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) ;Registre de control
     lwu r2,DR(r0) ;Registre de dades
     daddi r10,r0,1
     sd r3,(r2) ;Sortida del registro r3
     sd r10,(r1) ;en el terminal de sortida
     halt
```

Després d'executar-se el programa es mostrés el resultat de la suma en decimal en la finestra terminal.

II.ANNEX 2

Repertori d'instruccions suportat pel 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