ESTRUCTURA DE COMPUTADORS
Grau en Enginyeria Informàtica

Sessió de laboratori número 7

ARITMÈTICA DE COMA FLOTANT

Introducció

En esta pràctica es treballa amb l'aritmètica de coma flotant del MIPS R2000. La ferramenta de treball és el simulador del processador MIPS R2000 denominat PCSpim.

Objectius

- Entendre els fonaments del processament de nombres reals en un computador.
- Manipular nombres reals codificats per mitjà de l'estàndard IEEE 754 de simple i de doble precisió.
- Conéixer com llegir de la memòria principal els nombres reals.
- Entendre el funcionament de programes en assemblador que processen nombres reals.

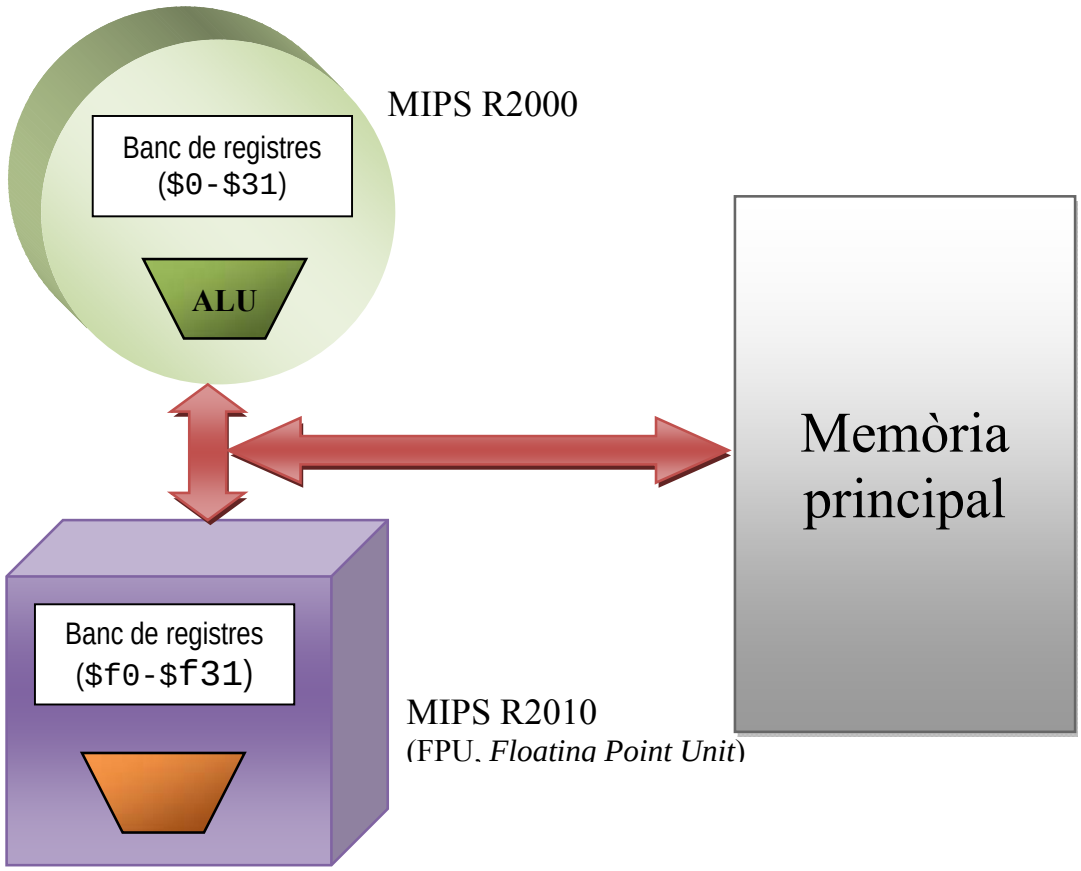
Material

El material es pot obtindre de la carpeta de recursos de PoliformaT.

- Simulador PCSpim del MIPS R2000.
- Arxius font (`formatos.s`, `promedio.s`, `pi-leibniz.s`).

L'aritmètica real en el processador MIPS R2000

El MIPS R2000 està dissenyat per a treballar amb una unitat de coma flotant (FPU, *floating point unit*) externa denominada MIPS R2010. La Il.lustració 1 mostra gràficament la connexió d'ambdós dispositius així com la seua relació amb la memòria.



Il.lustració 1. La unitat de coma flotant del processador MIPS R2000

La unitat de coma flotant té un banc de 32 registres de 32 bits anomenats `$f01`, `$f1`, `$f2`,…, `$f31`. No obstant això, des del punt de vista del programador, estos registres es veuen només com 16 registres, bé de 64 o de 32 bits; en tot cas, es fa ús exclusivament dels registres parells (`$f0`, `$f2`, `$f4`,…, `$f30`).

Els valors codificats en el format IEEE 754 de doble precisió (DP) s'emmagatzemen en una parella de registres, mentres que els valors de simple precisió (SP) s'ubiquen en un únic registre del banc. Si, per exemple, diem que el registre `$f0` conté un valor real de doble precisió, llavors els seus 32 bits de major pes s'emmagatzemen en `$f1` i els 32 de menor pes en `$f0`. En definitiva i, com s'ha dit, quan es dissenyen programes només es fa ús explícit dels registres parells.

Com ocorre amb el banc de registres de la unitat aritmeticològica, el patró d'utilització dels registres de coma flotant per part del programador no és arbitrari i ve establit en la taula següent:

Nom del registre	Utilització
$f0	Retorn de funció (part real)
$f2	Retorn de funció (part imaginària)
$f4,$f6,$f8,$f10	Registres temporals
$f12,$f14	Pas de paràmetres a funcions
$f16,$f18	Registres temporals
$f20,$f22,$f24,$f26,$f28,$f30	Registres a preservar entre crides

El processador *MIPS R2000 disposa de les següents instruccions per llegir o escriure nombres reals en la memòria principal:

- `lwc1 FPdst, Despl (Rsrc)`
- `swc1 FPsrc, Despl (Rsrc)`

On `FPsrc` i `FPdst` són registres del coprocessador de coma flotant ($f0..$f31) i `Rsrc` és un registre del processador base ($0..$31).

Per exemple, la instrucció `lwc1 $f4, 0($t0)` llegeix el contingut de l'adreça de memòria [$t0 + 0] i ho deixa en el registre `$f4`, mentre que `swc1 $f8, 0($t0)` escriu el contingut de `$f8` en memòria. Quan les variables són de doble precisió les operacions de lectura o escriptura necessiten utilitzar dues instruccions `lwc1` o `swc1`, respectivament.

El llenguatge assemblador també permet usar pseudoinstrucciones que faciliten l'escriptura dels programes. Algunes d'aquestes pseudoinstruccions permeten introduir nombre reals directament en els registres de la *FPU:

```
li.s FPdst, Num_float      # Load inmediate
li.d FPdst, Num_double
```

Per exemple, la pseudoinstrucció `li $f4, 2.7539` carregarà en el registre `$f4` el valor 2.7539 codificat en simple precisió (32 bits). Altres pseudoinstruccions permeten llegir o escriure de la memòria principal:

- `l.s FPdst, Address  # Load float from memory Address to FPdst`
- `l.d FPdst, Address  # Load double from memory Address to FPsrc|*FPsrc+1`
- `s.s FPsrc, Address  # Store float (FPsrc) to memory Address`
- `s.d FPsrc, Address  # Store double (FPsrc|FPsrc+1) to memory Address`

Per exemple, la pseudoinstrucció l.d $f4, A llegeix un nombre de doble precisió (8 bytes) de l'adreça de la variable en memòria '`A`' i ho emmagatzema en el parell `$f4|$f5`. La variable '`A`' haurà d'estar declarada com:

```
A: .double 2753.9I-3 # o qualsevol altre valor inicial
```

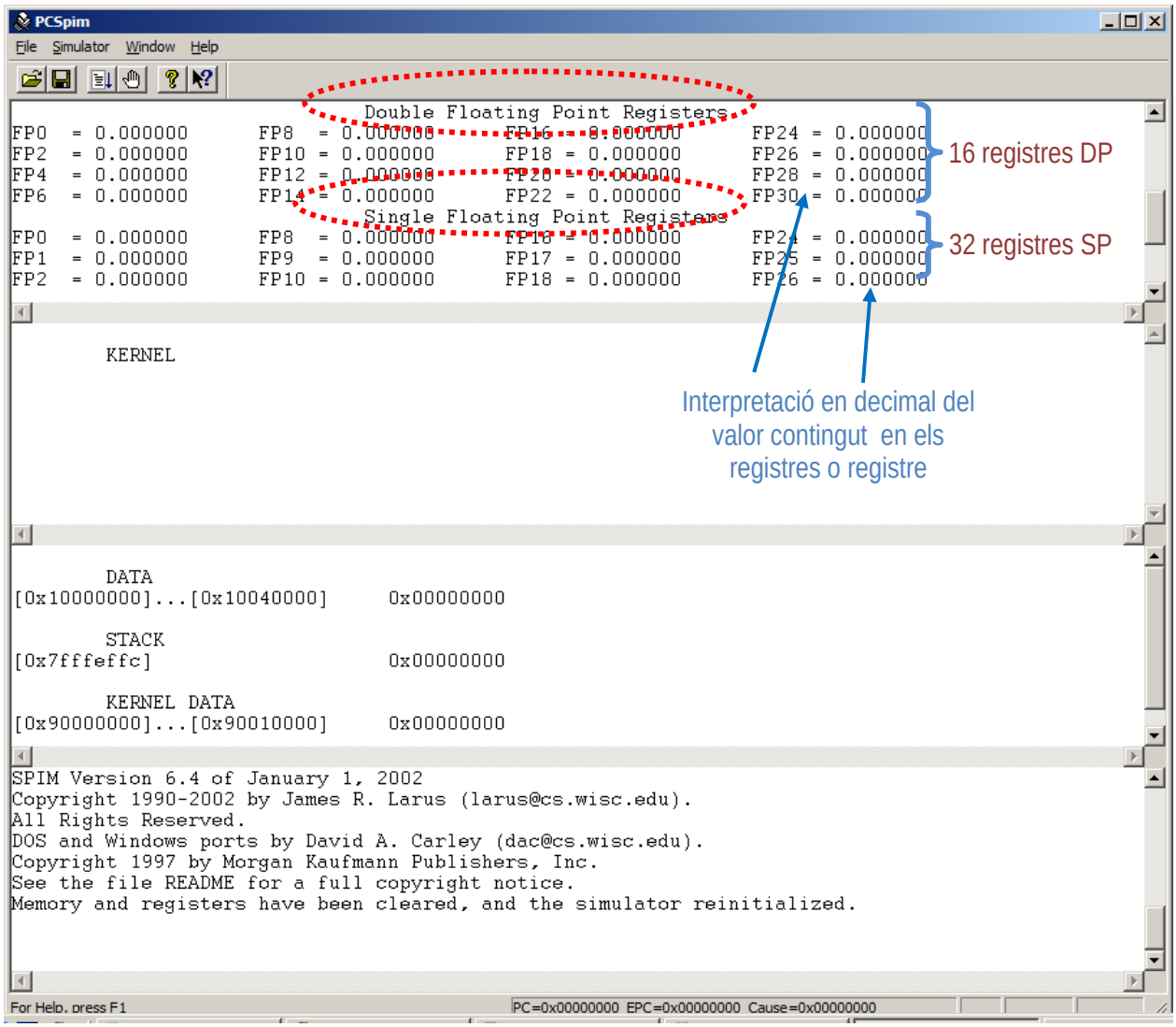
O també: `A: .space 8`

Però en aquest cas cal assegurar-se que la variable està correctament alineada en una adreça múltiple de 8.

Recordi que les pseudoinstruccions són traduïdes pel programa assemblador a instruccions executables pel processador.

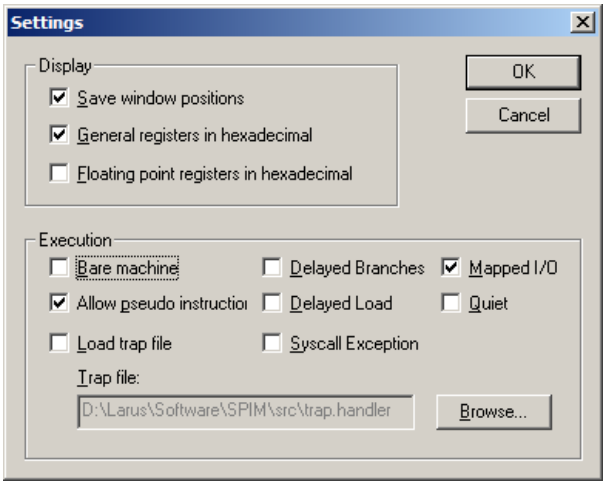
Configuració del simulador PCSpim

El simulador PCSpim permet l'execució de programes escrits en assemblador del processador MIPS R2000 per al tractament de números reals. Com es pot veure en la Il.lustració 2, en la part superior de la pantalla es mostra el contingut dels registres de la unitat de coma flotant. Es poden visualitzar com a números de doble precisió (64 bits) o números de simple precisió (32 bits). El que canvia d'un cas a un altre és la interpretació del contingut dels registres (doble o simple precisió).



Il.lustració 2. Els registres de la unitat de coma flotant

La forma en què es visualitza el contingut de cada registre es pot seleccionar en el menú *Simulator|Settings.* En este cas, segons s'aprecia en la Il.lustració 3, si es marca la casella assenyalada el contingut dels registres es mostra en hexadecimal. Si no es marca, com ocorre en el nostre cas, el contingut que es mostra és el seu valor d'acord amb la interpretació dels bits segons l'estàndard IEEE 754.



Il.lustració 3. Elecció de la visualització del contingut dels registres de coma flotant

Representació dels números en coma flotant

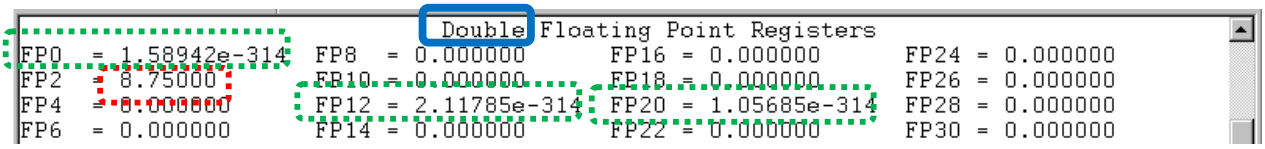
Començarem esta sessió pràctica amb un xicotet exemple per a il·lustrar la manera en què el simulador PCSpim visualitza els nombres de coma flotant. Considera el codi següent:

```
                .globl __start
                .text 0x00400000

__start:        li.s $f0, -1.5          # Constant -1.5
                li.d $f2, 8.75          # Constant 8.75

                li $t0, 0xFF800000      # Menys infinit (-∞)
                mtc1 $t0, $f12          # Enviament a $f12
                li $t1, 0x7F8003A0      # Not a Number (NaN)
                mtc1 $t1, $f20          # Enviament a $f20
```

El programa prepara quatre constants reals. Les dos primeres constants, –1.5 i 8.75, s'especifiquen per mitjà de dos pseudoinstruccions (*load immediate* per a *simple* i per a *double*) i es codifiquen en simple i doble precisió, respectivament. La grandària de cada variable és important per a interpretar bé el que el simulador ens mostra en la pantalla. En tot cas, no oblides que el contingut dels registres és únic: el que canviarà serà la interpretació que fem del seu contingut.

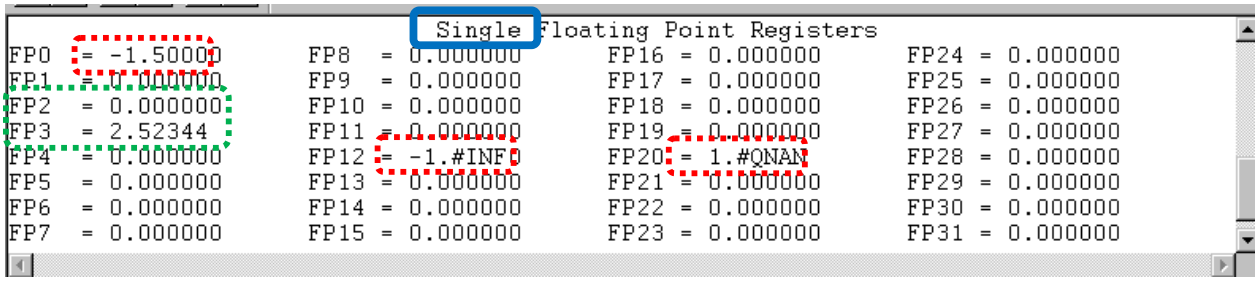
Les dos últimes constants s'especifiquen directament, també per mitjà de pseudoinstruccions (*load immediate* per a enters), en la seua codificació directa en l'estàndard IEEE 754 i serveixen, respectivament, per a representar el valor infinit amb signe negatiu (–∞) i un NaN (*Not a Number*) empleat per a posar de manifest situacions anòmales de càlcul (per exemple, indeterminacions del tipus ±0/±0, ±0× ±∞, etc.).

La constant –1.5 es guarda en `$f0`, però la constant 8.75 utilitza els registres `$f3|$f2`. El simulador ens oferix dos vistes complementàries del banc de registres de la unitat de coma flotant. En primer lloc ens mostra la interpretació que fa suposant que les variables són de doble precisió i cada una d'elles ocupa dos registres; en conseqüència, només veurem 16 valors:



En la figura veiem que, en efecte, 8.75 ocupa dos registres. No obstant això, la resta de valors emmagatzemats com a valors de simple precisió no s'interpreten correctament (veja els registres `$f0`, `$f12` i `$f20`).

Un poc més avall ens mostra la informació suposant que els registres contenen variables de simple precisió. Així doncs, veurem 32 valors:



En este cas la interpretació del contingut dels registres `$f0`, `$f12` i `$f20` és correcta: veiem que el primer conté −1.5, el segon −∞ i el tercer NaN (*Not a Number*).

►Carrega el programa anterior (fitxer `formats.s`) i executa-ho en el simulador. Comprova que els resultats obtinguts coincideixen amb els mostrats en les figures anteriors.

►Per què apareix el valor 2.52344 com contingut del registre `$f3`? Pots ajudar-te amb el simulador visualitzant el contingut dels registres en hexadecimal.

perque el valor 8.75 ha sigut guardat en doble precisio en $f2 potant ha utilitzat els dos registres el $f2 i el $f3. en hexadecimal (402180000)

►Quantes representacions possibles hi ha per al valor real 0.0 en l'estàndard IEEE 754 de simple precisió? Quines són eixes representacions? Expressa-les en hexadecimal.

2 representacions amb el signe a 0 o a 1

►Quantes representacions hi ha per al valor infinit (∞) en l'estàndard IEEE 754 de simple precisió? Quines són eixes representacions? Expressa-les en hexadecimal.

amb el signe 0 i mantisa f soia ∞
amb el signe 1 i mantisa f seia -∞

hexa { 0x7fc00000 = INF
0xffc00000 = INF

►Indica en quines instruccions ha traduït el programa assemblador la pseudoinstrucció del programa li.d `$f2, 8.75`. Interpreta el codi generat.

ori $1, $0, 0 → fica 0 la part baixa
mtc1 $1, $f2 → guarda la part baixa en $f2
lui $1, 16417 → carrega la part alta
ori $1, $1, -32768 → carrega la part baixa
mtc1 $1, $f3 → guarda el numero en coma flotant en $f3

►Indica en hexadecimal la representació en simple i doble precisió de la constant 78.325. Ajuda't del simulador per a obtindre les dos representacions.

SP → 0x429ca666
DP → 0xcccccccd
0x40539acc

►Quantes paraules diferents existeixen en el format de l'estàndard IEEE 754 de simple precisió per a representar el valor NaN? 0x7fffffff

1.#QNAN

7f800030

$2 \cdot 2^{24} - 1$ el cero de la mantisa no pot ser
1 signe, 8 exponent, 23 mantisa
24 bits
x/111111...11 ≠0

►Per què no hi ha una instrucció de suma de nombres reals semblant a `addi`?

porque necessitaria más de 32 bits para asignar el numero y el codigo de instruccion.

Càlcul de la mitjana aritmètica

A continuació es presenta un programa escrit en assemblador que calcula la mitjana aritmètica d'un conjunt de valors reals. Donats *n* nombres a_0, a_1,…, a_{n-1}, la seva mitjana es defineix per mitjà de la fórmula:

$$\frac{1}{n}\sum_{i=0}^{n-1} a_i$$

Els nombres reals es codifiquen per mitjà de variables de simple precisió (*float*) i s'ubiquen en memòria a partir de l'etiqueta `valors`. El valor de la mitjana es calcula tant en simple precisió (`mitjana_s`) com en doble precisió (`mitjana_d`) i s'emmagatzema en el segment de dades.

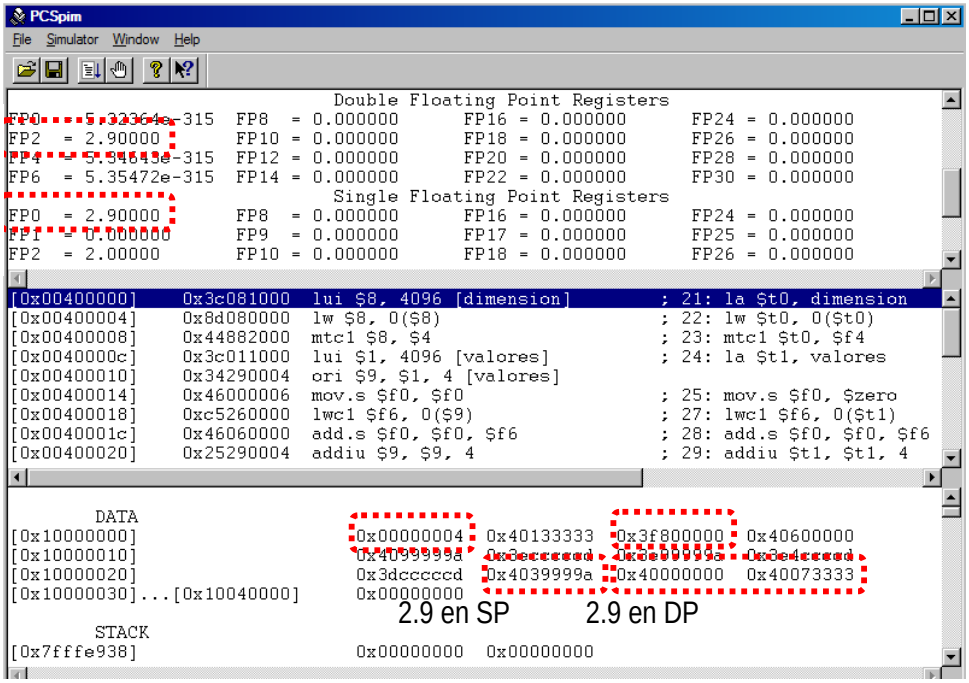
```
                ############################################################
                # Segment de dades
                ############################################################

                .data 0x10000000
dimensió:       .word 4
valors:         .float 2.3, 1.0, 3.5, 4.8
pesos:          .float 0.4, 0.3, 0.2, 0.1
mitjana_s:      .float  0.0
mitjana_d:      .double 0.0
```

```
                ############################################################
                # Segment de codi
                ############################################################

                .globl __start
                .text 0x00400000

__start:        la $t0, dimensió        # Adreça de la dimensió
                lw $t0, 0($t0)          # Lectura de la dimensió
                mtc1 $t0, $f4           # Porta la dimensió a $f4
                la $t1, valors          # Adreça dels valors
                mtc1 $zero, $f0         # Porta 0.0 a $f0

bucle:          lwc1 $f6, 0($t1)         # Llig valor[i]
                add.s $f0, $f0, $f6      # Suma del valor
                addiu $t1, $t1, 4        # Adreça de valor[i+1]
                addiu $t0, $t0, -1       # Decrementa comptador
                bgtz $t0, bucle

                cvt.s.w $f4, $f4         # Convertix dimensió a real
                div.s $f0, $f0, $f4      # Calcula mitjana aritmètica
                cvt.d.s $f2, $f0         # Convertix a doble precisió
                la $t0, mitjana_s        # Adreça del resultat mitjana_s
                swc1 $f0, 0($t0)         # Escriu resultat simple precisió
                la $t0, mitjana_d        # Adreça del resultat mitjana_d
                swc1 $f2, 0($t0)         # Escriu part baixa doble precisió
                swc1 $f3, 4($t0)         # Escriu part alta doble precisió
                .end
```

►El programa anterior es troba en el fitxer `mitjana.s`. Carrega-ho i executa-ho en el simulador. La figura següent mostra l'estat del simulador després d'executar el programa. Comprova que els resultats que has obtingut coincideixen amb els mostrats en la figura.

Resultat SP en `$f0`

Resultat DP en `$f3|$f2`



En la figura anterior s'ha destacat la ubicació del resultat en el banc de registres de la unitat de coma flotant, així com la seua localització en el segment de dades. Torna a examinar el codi i analitza el seu

funcionament, en especial la ubicació de les dades de partida en el segment de dades, la seua codificació, així com la localització dels resultats.

►Explica per a què serveix la instrucció `cvt.s.w` que apareix en el codi.

Serveix per a convertir una paraula a coma flotant en simple precisió

►Indica en la taula següent la codificació IEEE 754 per a la mitjana calculat tant en simple com en doble precisió. En el cas de la variable de doble precisió: la part baixa s'emmagatzema en l'adreça de memòria més baixa, i la part alta en la més alta.

Variable	Valor decimal	Codificació IEEE 754
`mitjana_s`	2'9	0x4039999a
`mitjana_d`	2'9	0x4007333340000000

►Indica quantes operacions aritmètiques de coma flotant i de quina classe (suma, resta, conversió de tipus, etc.) s'executen en el programa. 7 operaciones

4x add.s → suma
cvt.s.w → pasar a coma flotante
div.s → division
cvd.d.s → pasar de doble a simple precision

►Si el programa s'executa en un processador real en 0.5 microsegons, calcula el nombre d'operacions en coma flotant per segon aconseguits pel processador (FLOPS, *floating point operations per second*). Indica el resultat en milions d'operacions per segon (MFLOPS).

$$\frac{7 \text{ operacions}}{0'5 \times 10^{-6}} = 14 \times 10^6 = 14 \text{ MFLOPS}$$

Càlcul del número π

El número π (pi) és la relació entre la longitud d'una circumferència i el seu diàmetre. És un número irracional i una de les constants matemàtiques més importants. S'empra sovint en matemàtiques, física i enginyeria. El valor numèric de π, truncat a les seues primeres xifres, és el següent:

$$\pi \approx 3{,}14159265358979323846...$$

El valor de π s'ha obtingut amb diverses aproximacions al llarg de la història, sent una de les constants matemàtiques que més apareix en les equacions de la física, junt amb el número *e*.

El matemàtic alemany Gottfried Leibniz va idear en 1682 un mètode per al càlcul del número π. El dit mètode realitza una aproximació a π/4 a través de la sèrie infinita següent:

$$\frac{\pi}{4} = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

A continuació es presenta un programa que calcula el valor de π per mitjà de la sèrie anterior. El codi que presentem desenrotlla la sèrie fins al valor especificat per l'usuari per a la variable del programa `n`. El programa principal utilitza la subrutina `leibniz` per al dit càlcul. L'entrada i eixida de dades es du a terme per mitjà de crides al sistema.

```
                ############################################################
                # Segment de dades
                ############################################################

                .data 0x10000000
                cad_entrada:    .asciiz "\nDime el nombre d'iteracions: "
                cad_eixida:     .asciiz "El valor calculat de pi és: "

                ############################################################
                # Segmento de codi
                ############################################################

                .globl __start
                .text 0x00400000
__start:
                ############################################################
                # Lectura del nombre d'iteracions
                ############################################################

                la $a0, cad_entrada     # Cadena a imprimir
                li $v0, 4               # Funció print_string
                syscall

                li $v0, 5               # Funció read_int
                syscall

                move $a0, $v0           # Paràmetre de la subrutina
                jal leibniz             # Bot a la subrutina

                ############################################################
                # Impresió del resultat
                ############################################################

                la $a0, cad_eixida      # Cadena a imprimir
                li $v0, 4               # Funció print_string
                syscall

                li $v0, 2               # Funció print_float
                mfc1 $t0, $f0           # Valor a imprimir
                mtc1 $t0, $f12
                syscall

                ############################################################
                # Finalització del programa
                # Croda al sistema denominada "exit"
                ############################################################=

                li $v0, 10
                syscall

                ############################################################
                # Càlcul de pi amb el mètode de Leibniz
```

```
                # $a0 = Nombre d'iteracions de la sèrie
                ######################################################

leibniz:        li.s $f0, 0.0           # Constant 0.0
                li.s $f4, 1.0           # Constant 1.0
                li.s $f6, 2.0           # Constant 2.0
                move $t0, $a0           # Comptador nombre d'iteracions

bucle:          mtc1 $t0, $f8           # Porta n a la FPU
                cvt.s.w $f8, $f8        # Convertix n en número real

                mul.s $f8, $f8, $f6     # Calcula 2.0*n
                add.s $f8, $f8, $f4     # Calcula 2.0*n + 1.0
                div.s $f8, $f4, $f8     # Calcula 1.0/(2.0*n + 1.0)

                andi $t1, $t0, 0x0001   # Extrau bit LSB de n
                bne $t1, $zero, resta   # Bota si és imparell (LSB==1)
                add.s $f0, $f0, $f8     # El terme se suma
                j contínua
resta:          sub.s $f0, $f0, $f8     # El terme es resta
contínua:       addi $t0, $t0, -1       # Decrementa nombre d'iteracions
                bgez $t0, bucle         # Torna si queden iteracions

                li.s $f4, 4.0           # Constant 4.0
                mul.s $f0, $f0, $f4     # Torna en $f0 el càlcul de pi
                jr $ra
                .end
```

Analitza amb deteniment el codi anterior. Podràs comprovar que tot el càlcul de la sèrie es du a terme dins de la subrutina `leibniz`.

►Indica com fa el programa per a calcular si el terme de la sèrie se suma (n parell) o es resta (n senar).

andi $t1, $t0, 0x0001 → extrau LSB
bne $t1, $zero, resta → si LSB = 1 → es imparell i fa la resta
si no fa la suma.

►Expressa el nombre d'operacions de coma flotant que es duen a terme en el programa anterior en funció del nombre *n* d'iteracions.

5n + 1

►Carrega en el simulador el programa anterior (fitxer `pi-leibniz.s`) i executa-ho per als diferents desenrotllaments de la sèrie que s'especifiquen més avall. Completa la següent taula indicant els deu primers nombres decimals calculats del número π. Arrodoneix el valor del desé dígit.

Iteracions (n)	Valor calculat de π
10^3	3.1425914764
10^4	3.1416926333
10^5	3.1416025161
10^6	3.1415934562

L'arquitectura del MIPS R2000 ens ofereix instruccions de moviment de dades entre els bancs de registres enters i de coma flotant (`mtc1`, `mfc1`). També existeixen instruccions específiques de

moviment entre registres de coma flotant: `mov.s` i `mov.d`. Per exemple, `mov.s $f4,$f2` copia el contingut del registre `$f2` en `$f4`.

► Imagineu per un moment que la instrucció `mov.s` no estiguera disponible en l'arquitectura del processador. Quines instruccions alternatives es podrien emprar per tal de moure el contingut del registre `$f2` a `$f4`?

swc1 $f2
lwc1 $f4,

mfc1 $f2, $f4

► Per tal de moure el contingut d'un registre enter a un altre es pot emprar la pseudoinstrucció `move`. Per exemple, `move $t0,$t1` du el contingut de `$t1` a `$t0`. Per què creieu que `move` no s'ha inclòs en el processador com a instrucció màquina?

perque no fa falta

addu $t1, $zero, $t2

►Adapta el programa a nombres reals codificats en l'estàndard IEEE 754 de doble precisió (variables reals de tipus *double*) i crida al fitxer `pi-leibniz-d.s`. Pren atenció amb el trasllat del resultat final ubicat en la parella de registres `$f1|$f0` per a la seua impressió en la consola. Entre altres coses hauràs de modificar la crida al sistema que imprimeix aquest tipus de variables (l'índex de `print_double` és 3. Indica breument els canvis realitzats respecte de la versió original en simple precisió.

►Executa el programa i completa la taula següent:

Iteracions (n)	Valor calculat de π
10^3	3.1425916543
10^4	3.1416926435
10^5	3.1416026534
10^6	3.1415936535