



PRINCIPIOS DE COMPUTADORES

TUTORÍA ACADÉMICA NÚMERO 3:

- Operaciones aritméticas en punto flotante.
- Llamadas al sistema.

INSTRUCCIONES EN PUNTO FLOTANTE

- MIPS tiene un coprocesador de punto flotante (coprocesador 1 de la máquina) que opera en precisión simple (32 bits) y doble precisión (64 bits).
- El coprocesador tiene sus propios registros de 32 bits que que van desde el \$f0 al \$f31. Para operaciones de doble precisión se requerirán dos de estos registros, por eso de forma práctica solo se usan los pares. \$f0-\$f1 almacenan un doble, \$f2-\$f3 otro y así hasta \$f30-\$f31
- Los registros de punto flotante tienen igualmente un convenio que se ha de respetar:

Registro	Uso
\$f0-\$f2	Valores de retorno en punto flotante a subprogramas
\$f4-\$f10	Registros temporales. No se preservan entre llamadas
\$f12-\$f14	Primeros 2 parámetros en punto flotante. No preservados
\$f16-\$f18	Registros temporales. No se preservan entre llamadas
\$f20-\$f30	Registros salvados. Se preservan entre llamadas.

INSTRUCCIONES ARITMÉTICAS EN PUNTO FLOTANTE

Ejemplo	Significado	Comentario
add.s \$f2,\$f4,\$f6	$\$f2 = \$f4 + \$f6$	Suma PF. Precisión simple
sub.s \$f2,\$f4,\$f6	$\$f2 = \$f4 - \$f6$	Resta PF. Precisión simple
mul.s \$f2,\$f4,\$f6	$\$f2 = \$f4 * \$f6$	Multiplicación PF. Precisión simple
div.s \$f2,\$f4,\$f6	$\$f2 = \$f4 / \$f6$	División PF. Precisión simple
abs.s \$f2,\$f4	$\$f2 = \$f4 $	Valor absoluto en simple precisión
neg.s \$f2,\$f4	$\$f2 = -\$f4$	Cambio de signo simple precisión
add.d \$f2,\$f4,\$f6	$\$f2 = \$f4 + \$f6$	Suma PF. Precisión doble
sub.d \$f2,\$f4,\$f6	$\$f2 = \$f4 - \$f6$	Resta PF. Precisión doble
mul.d \$f2,\$f4,\$f6	$\$f2 = \$f4 * \$f6$	Multiplicación PF. Precisión doble
div.d \$f2,\$f4,\$f6	$\$f2 = \$f4 / \$f6$	División PF. Precisión doble
abs.d \$f2,\$f4	$\$f2 = \$f4 $	Valor absoluto en doble precisión
neg.d \$f2,\$f4	$\$f2 = -\$f4$	Cambio de signo doble precisión

CARGA Y ALMACENAMIENTO EN PUNTO FLOTANTE

Ejemplo	Significado	Comentario
lwc1 \$f0,etiqueta l.s \$f0,etiqueta	\$f0=Mem[etiqueta]	Carga en \$f0 (simple precisión) en la dir. Especificada.
swc1 \$f0,etiqueta s.s \$f0,etiqueta	Mem[etiqueta]=\$f0	Almacena en la dir. especificada el registro \$f0(simple precisión)
ldc1 \$f0,etiqueta l.d \$f0,etiqueta	\$f0=Mem[etiqueta]	Carga en \$f0 (doble precisión) en la dir. Especificada.
sdc1 \$f0,etiqueta s.d \$f0,etiqueta	Mem[etiqueta]=\$f0	Almacena en la dir. especificada el registro \$f0(doble precisión)
li.s \$f0,3.4	\$f0=3.4	Carga inmediata simple precisión
li.d \$f0,3.4	\$f0=3.4	Carga inmediata doble precisión
mov.s \$f4,\$f6	\$f4 = \$f6	Mueve datos entre registros punto flotante de precisión simple
mov.d \$f4,\$f6	\$f4 = \$f6	Mueve datos entre registros punto flotante de precisión doble
mtc1 \$t0,\$f0	\$f0 = \$t0	Es una copia "cruda" no pasa a flotante. Ojo porque el orden de los operandos no es el de convenio.
mfc1 \$t0,\$f0	\$t0 = \$f0	Igual que antes es copia cruda. Ojo con el orden.

COPIA Y CONVERSIÓN EN PUNTO FLOTANTE

Ejemplo	Significado	Comentario
<code>mtc1 \$t0,\$f0</code>	<code>\$f0 = \$t0</code>	Es una copia "cruda" no pasa a flotante. Ojo porque el orden de los operandos no es el de convenio.
<code>mfc1 \$t0,\$f0</code>	<code>\$t0 = \$f0</code>	Igual que antes es copia cruda. Ojo con el orden.
<code>cvt.s.w \$f2,\$f0</code>	<code>\$f2 = \$f0</code>	Supone que \$f0 tiene la copia cruda de un entero (no formato IEEE754) y lo almacena en \$f2 en formato IEEE754 simple precisión
<code>cvt.s.d \$f2,\$f0</code>	<code>\$f2 = \$f0</code>	Supone que \$f0 está en IEEE754 doble precisión y lo almacena en \$f2 en simple precisión.
<code>cvt.d.w \$f2,\$f0</code>	<code>\$f2 = \$f0</code>	Supone que \$f0 tiene la copia cruda de un entero (no formato IEEE754) y lo almacena en \$f2 en formato IEEE754 doble precisión
<code>cvt.d.s \$f2,\$f0</code>	<code>\$f2 = \$f0</code>	Supone que \$f0 está en IEEE754 simple precisión y lo almacena en \$f2 en doble precisión.

Ejemplo (mira los registros en el QTSpim:)

```
li    $t0,32      # $t0 = 32
mtc1 $t0,$f2      # $f2 = 32
cvt.s.w $f4,$f2   # $f4 = 32.0
```



PUNTO FLOTANTE. COMPARACIÓN Y SALTO

Ejemplo	Significado	Comentario
<code>c.eq.s \$f2,\$f4 (eq,lt,le)</code>	<code>If (\$f2==\$f4) cond =1 else cond =0</code>	Comparación en precisión simple. Pone cond a 1 o a 0.
<code>c.eq.d \$f2,\$f4 (eq,lt,le)</code>	<code>If (\$f2==\$f4) cond =1 else cond =0</code>	Comparación en precisión doble. Pone cond a 1 o a 0.
<code>bclt etiqueta</code>	<code>If (cond==1) go to etiqueta</code>	Salto condicional si cond es cierto
<code>bclf etiqueta</code>	<code>If (cond==0) go to etiqueta</code>	Salto condicional si cond es falso

○ Ejemplo:

```
c.eq.s $f2,$f4 # salta a la etiqueta exit si $f2==$f4
bclt  exit
exit:
```



ENTRADA Y SALIDA. LLAMADAS AL SISTEMA

- El simulador nos permite leer y escribir valores de entrada y salida en la ventana de la consola y hacer una salida correcta del programa.
- Usa para ello la llamada al sistema operativo **syscall**
- Se le pasa a la llamada syscall valores en los registros \$v0 y adicionalmente en \$a0 y \$a1.
- Si syscall tiene valor de salida quedará registrado en \$v0



ENTRADA Y SALIDA. LLAMADAS AL SISTEMA

Servicio	\$v0	Argumentos	Resultados
print_int	1	\$a0 = entero a imprimir	
print_float	2	\$f12 = flotante a imprimir	
print_double	3	\$f12-13 = double a imprimir	
print_string	4	\$a0 = direccion de memoria de la cadena a imprimir	
read_int	5		\$v0 = entero que se leyó
read_float	6		\$f0 = flotante que se leyó
read_double	7		\$f0-\$f1 = double que se leyó
read_string	8	\$a0 = dirección memoria donde se almacenará la cadena a leer. \$a1 = número. tamaño de la cadena	
sbrk	9	\$a0 = n° bytes requeridos de memoria dinámica	\$v0 = dirección de memoria que contendrá los bytes
exit	10		

ENTRADA Y SALIDA. LLAMADAS AL SISTEMA

- La función `print_string` siempre espera que la cadena termine un carácter nulo. Con la directiva `.asciiz` se crea una cadena terminada en un nulo (ver ejemplo)
- Las funciones de lectura `read_int`, `read_float`, y `read_double` siempre esperan hasta que se introduzca un retorno de carro en la entrada.
- La función `read_string` (recordar que `$a0` contiene la dirección del buffer y `$a1` contiene el número de caracteres) se comporta de la siguiente manera:
 - Si introducimos caracteres hasta llenarlo, los mete en los `n-1` primeros bytes, y a continuación introduce un carácter nulo (suponiendo que el tamaño del buffer es de `n` bytes)
 - Si introducimos menos caracteres de los que teníamos reservados, termina la cadena con un `newline` (retorno de carro) y al final un carácter nulo.
- La función `sbrk` sirve para solicitar memoria de forma dinámica. Busca `n` bytes (indicados por `$a0`) libres consecutivos y devuelve la dirección del primer byte libre en `$v0`.
- La función `exit` hace un que un programa termine de forma inmediata.



ENTRADA Y SALIDA. EJEMPLO.

```
.data          # Seccion declaracion de datos
cadpet: .asciiz "Introduzca un entero: ";
cadsuma: .asciiz "La suma es: ";
cadrc: .asciiz "\n";

.text          # Seccion de codigo de usuario

main:          # La etiqueta main es el inicio

# la funcion print_string es $v0=4
li $v0,4       # $v0 = 4 funcion print string
la $a0,cadpet  # $a0 = direccion de la cadena a imprimir
syscall
# leo el primer entero y lo almaceno en $t0
li $v0, 5      # $v0 = 5 funcion leer un entero.
syscall        # el entero leido se queda en $v0
move $t0,$v0   # almaceno en $t0 el primer entero leido.
# la funcion print_string es $v0=4
li $v0,4       # $v0 = 4 funcion print string
la $a0,cadpet  # $a0 = direccion de la cadena a imprimir
syscall
# leo el segundo entero y lo almaceno en $t1
li $v0, 5      # $v0 = 5 funcion leer un entero.
syscall        # el entero leido se queda en $v0
move $t1,$v0   # almaceno en $t1 el segundo entero leido.

add $t3,$t0,$t1 # almaceno en $t3 la suma de los dos enteros
# la funcion print_int es $v0 = 1 y la funcion print_string es $v0=4
li $v0,4       # $v0 = 4 funcion print string
la $a0,cadsuma # $a0 = direccion de la cadena a imprimir
syscall
li $v0,1       # $v0 = 4 funcion print_int
move $a0,$t3   # $a0 = entero a imprimir
syscall
li $v0,4       # $v0 = 4 funcion print string
la $a0,cadrc   # $a0 = direccion de la cadena a imprimir
syscall
# se hace una salida limpia del sistema (exit es codigo 10)
li $v0, 10
syscall
# END
```



EJEMPLO. CONVERSIÓN PTS-€

```
# Programa que convierte de pesetas a euros
.data
    msgentrada: .asciiz "Introduzca el importe en pesetas: "
    msgsalida: .asciiz "\nImporte en euros: "
.text
main:
    la $a0, msgentrada
    li $v0,4
    syscall # escribo el mensaje por pantalla

    li $v0,5
    syscall # leo un entero y lo deja en $v0

    # realizo una copia cruda del entero en un registro flotante
    mtcl $v0,$f4
    # $f6 = convierto $f4 al formato flotante IEEE754
    cvt.s.w $f6,$f4
    li.s $f8,166.386 # $f8 1 € en pesetas
    div.s $f10,$f6,$f8 # $f10 importe en €

    la $a0, msgsalida
    li $v0,4
    syscall # escribo el mensaje de salida por pantalla

    mov.s $f12,$f10
    li $v0,2
    syscall # escribo el flotante simple por pantalla

    li $v0,10
    syscall
```

