Introducción

- Lectura de memoria: *l.d f1, desp(r5)*
- Escritura a memoria: *s.d f3, res(r0)*
- Copiar valor de un registro a otro: *mov.d fd, ff ; fd = ff*

No hay carga de valores inmediatos!

Operaciones básicas

- *add.d fd, ff, fg*: Suma ff con fg, dejando el resultado en fd
- *sub.d fd, ff, fg*: Resta fg a ff, dejando el resultado en fd
- *mul.d fd, ff, fg*: Multiplica ff con fg, dejando el resultado en fd
- *div.d fd, ff, fg*: Divide ff por fg, dejando el resultado en fd

Conversión Numero Entero / Punto Flotante

Las copias se deben hacer en 2 pasos.

- Copia de bits desde/hasta punto flotante
  - En este paso se copian bits, no se copian valores!
- Conversión de los bits en número entero / punto flotante.

Las conversiones no siempre respetan el número original (por cuestiones de redondeo y representación). Nosotros no vamos a tener en cuenta esto en la práctica.

De registro entero a punto flotante

Para copiar el valor que tengo en un registro entero (r0 a r31) a uno de punto flotante (f0 a f31):

- Copiar los 64 bits del registro entero rf al registro fd de punto flotante
  - *mtc1 rf, fd*
- Conviertir a punto flotante el valor entero copiado al registro ff, dejándolo en fd
  - *cvt.d.l fd, ff*

Importante: los números muy grandes serán redondeados en su mejor representación de punto flotante.

Copiar el valor de r5 a f10

```
mtc1 r5, f10
cvt.d.l f10, f10
```

De punto flotante a registro entero

Para copiar el valor que tengo en un registro de punto flotante (f0 a f31) a un registro entero (r1 a r31):

- Conviertir a entero el valor en punto flotante contenido en ff, dejándolo en fd
  - *cvt.l.d fd, ff*
- Copiar los 64 bits del registro ff de punto flotante al registro rd entero
  - *mfc1 rd, ff*

Importante: El número se trunca, no se redondea

Copiar el valor de f3 a r8

```
cvt.l.d f4, f3
mfc1 r8, f4
```

Saltos condicionales en Punto Flotante

La unidad de punto flotante posee un flag (FP):

- *c.lt.d fd, ff*: Compara fd con ff, dejando FP=1 si fd es menor que ff, sino FP=0
- *c.le.d fd, ff*: Compara fd con ff, dejando flag FP=1 si fd es menor o igual que ff, sino FP=0
- *c.eq.d fd, ff*: Compara fd con ff, dejando flag FP=1 si fd es igual que ff, sino FP=0

Podemos usar el flag FP con las siguientes instrucciones:

- *bc1t etiqueta*: Salta a *etiqueta* si flag FP=1
- *bc1f etiqueta*: Salta a *etiqueta* si flag FP=0

Subrutinas

- El soporte de la arquitectura para la invocación de subrutinas es menor que el soporte que existe en otras arquitecturas (Pila, CALL, RET)
- La instrucción encargada de realizar un llamado a una subrutina es *jal* (jump and link):
  - La instrucción guarda en el registro *r31* la dirección de retorno de la subrutina
  - Cambia el PC por la dirección de la subrutina
- Para retornar, basta retornar a la posición apuntada por el registro *r31*
  - Esto podemos hacerlo con la instrucción *jr r31*

Subrutinas

Acción	MSX88	WinMIPS64
Llamada a subrutina	CALL *subrutina* Se apila la dirección de retorno Se actualiza IP con el valor de la subrutina	jal *subrutina* Se guarda en el registro *r31* la dirección de retorno Se actualiza PC con el valor de la subrutina
Retorno de la subrutina	RET Se desapila la dirección de retorno y se actualiza IP con ese valor	jr *r31* Se actualiza PC con el valor del registro *r31*, que tiene la dirección de retorno de la subrutina

- ¿Qué pasa si una subrutina invoca a otra subrutina?

Ejemplo

```
                .data
                base: .word 16
                exponente: .word 4
                result: .word 0

                .text
                ld r1, base(r0)
                ld r2, exponente(r0)
                jal a_la_potencia
                sd r5, result(r0)
                halt

a_la_potencia:  daddi r5, r0, 1
lazo:           beqz r2, terminar
                daddi r2, r2, -1
                dmul r5, r5, r1
                j lazo
terminar:       jr r31
```

Normalización de registros

En lugar de usar r0-r31 es más conveniente darle nombre más significativos a los registros:

Registro	Nombre	Uso	Preservado
r0	*$zero*, $0	Siempre tiene el valor 0 y no se puede cambiar.	No
r1	*$at*	Assembler Temporary – Reservado para ser usado por el ensamblador.	No
r2 - r3	*$v0 - $v1*	Valores de retorno de la subrutina llamada.	No
r4 - r7	*$a0 - $a3*	Argumentos pasados a la subrutina llamada.	No
r8 - r15	*$t0 - $t7*	Registros temporarios. No son conservados en el llamado a subrutinas.	No
r16 - r23	*$s0 - $s7*	Registros salvados durante el llamado a subrutinas.	Si
r24 - r25	*$t8 - $t9*	Registros temporarios. No son conservados en el llamado a subrutinas.	No
r26 - r27	*$k0 - $k1*	Para uso del kernel del sistema operativo.	No
r28	*$gp*	Global Pointer – Puntero a la zona de la memoria estática del programa.	Si
r29	*$sp*	Stack Pointer – Puntero al tope de la pila.	Si
r30	*$fp*	Frame Pointer – Puntero al marco actual de la pila.	Si
r31	*$ra*	Return Address – Dirección de retorno en un llamado a una subrutina.	Si

- Preservado implica que los registros deben devolverse *inalterados* desde una subrutina.
- *$s0-$s7* representan las variables locales de la subrutina / programa.
- *$t0-$t9* son registros para almacenar resultados auxiliares.
- Este uso de registros es el recomendado y esta normalizado
- Si una subrutina altera el valor de algún registro que debe ser preservado, debe conservar el valor original para poder restaurarlo.

Ejemplo

```
                .data
                base: .word 16
                exponente: .word 4
                result: .word 0

                .text
                ld $a0, base($0)
                ld $a1, exponente($0)
                jal a_la_potencia
                sd $v0, result($0)
                halt

a_la_potencia:  daddi $v0, $0, 1
lazo:           beqz $a1, terminar
                daddi $a1, $a1, -1
                dmul $v0, $v0, $a0
                j lazo
terminar:       jr $ra
```