ISTRUZIONI E PSEUDO-ISTRUZIONI MIPS

ARITMETICA

add	$1, $1, $3	$1 := $2 + $3	addizione
addu	$1, $1, $3	$1 := $2 + $3	addizione naturale
addi	$1, $2, cost	$1 := $2 + cost	addizione di costante
addiu	$1, $2, cost	$1 := $2 + cost	addizione naturale di costante
sub	$1, $2, $3	$1 := $2 − $3	sottrazione
subu	$1, $2, $3	$1 := $2 − $3	sottrazione naturale
mult	$1, $2	hi|lo := $1 × $2	moltiplicazione (risultato a 64 bit)
div	$1, $2	lo := $1 / $2; hi := $1 % $2	divisione (quoziente in lo; resto in hi)

ARITMETICA – pseudo-istruzioni

mul	$1, $2, $3	$1 := $2 × $3	moltiplicazione (lo in $1)
muli	$1, $2, imm	$1 := $2 × imm	moltiplicazione per imm (lo in $1)
div	$1, $2, $3	$1 := $2 / $3	divisione (lo in $1)

CONFRONTO

slt	$1, $2, $3	**if** $2 < $3 **then** $1 := 1 **else** $1 := 0	poni a 1 se strettamente minore
sltu	$1, $2, $3	**if** $2 < $3 **then** $1 := 1 **else** $1 := 0	poni a 1 se strettamente minore naturale
slti	$1, $2, cost	**if** $2 < cost **then** $1 := 1 **else** $1 := 0	poni a 1 se strettamente minore di costante
sltiu	$1, $2, cost	**if** $2 < cost **then** $1 := 1 **else** $1 := 0	poni a 1 se strettamente minore di costante naturale

LOGICA

or	$1, $2, $$3	$1 := $2 **or** $3	somma logica bit a bit
and	$1, $2, $$3	$1 := $2 **and** $3	prodotto logico bit a bit
ori	$1, $2, cost	$1 := $2 **or** cost	somma logica bit a bit con costante
andi	$1, $2, cost	$1 := $2 **and** cost	prodotto logico bit a bit con costante
nor	$1, $2, $$3	$1 := $2 **nor** $3	somma logica negata bit a bit
sll	$1, $2, cost	$1 := $2 << cost	scorrimento a sinistra (left) del numero di bit specificato da cost
srl	$1, $2, cost	$1 := $2 >> cost	scorrimento a destra (right) del numero di bit specificato da cost

LOGICA – pseudo-istruzione

not	$1, $2	s1 := **not** s2	negazione logica (not bit a bit)

SALTO INCONDIZIONATO: ASSOLUTO, INDIRETTO E CON COLLEGAMENTO

j	indir	pc := indir (28 bit)	salto incondizionato assoluto
jr	$1	pc := $1 (32 bit)	salto incondizionato indiretto da registro
jal	indir	pc := indir (28 bit) e collega il registro $ra	salto incondizionato assoluto con collegamento

SALTO CONDIZIONATO

beq	$1, $2, spi	**if** $1 = $2 salta relativo a PC	salto condizionato di uguaglianza
bne	$1, $2, spi	**if** $1 ≠ $2 salta relativo a PC	salto condizionato di disuguaglianza

SALTO CONDIZIONATO – pseudo-istruzioni

blt	$1, $2, spi	**if** $1 < $2 salta relativo a PC	salta se strettamente minore
bgt	$1, $2, spi	**if** $1 > $2 salta relativo a PC	salta se strettamente maggiore
ble	$1, $2, spi	**if** $1 ≤ $2 salta relativo a PC	salta se minore o uguale
bge	$1, $2, spi	**if** $1 ≥ $2 salta relativo a PC	salta se maggiore o uguale

TRASFERIMENTO TRA PROCESSORE E MEMORIA

lw	$1, spi ($2)	$1 := mem ($2 + spi)	carica parola (a 32 bit)
sw	$1, spi ($2)	mem ($2 + spi) := $1	memorizza parola (a 32 bit)
lh, *lhu*	$1, spi ($2)	$1 := mem ($2 + spi)	carica mezza parola (a 16 bit)
sh	$1, spi ($2)	mem ($2 + spi) := $1	memorizza mezza parola (a 16 bit)
lb, *lbu*	$1, spi ($2)	$1 := mem ($2 + spi)	carica byte (a 8 bit)
sb	$1, spi ($2)	mem ($2 + spi) := $1	memorizza byte (a 8 bit)

TRASFERIMENTO TRA PROCESSORE E MEMORIA – pseudo-istruzioni (vedi nota 1 sotto)

lw	$1, etichetta	$1 := mem ($gp + spi di etichetta)	carica parola (a 32 bit)
sw	$1, etichetta	mem ($gp + spi di etichetta) := $1	memorizza parola (a 32 bit)

TRASFERIMENTO TRA REGISTRI (non referenziabili)

mflo	$1	$1 := lo	copia registro lo
mfhi	$1	$1 := hi	copia registro hi

TRASFERIMENTO TRA REGISTRI – pseudo-istruzione

move	$1, $2	$1 := $2	copia registro

CARICAMENTO DI COSTANTE IN REGISTRO

lui	$1, cost	$1 (16 bit più signif.) := cost	carica cost (in 16 bit più signif. di $1) (16 bit meno signif. di $1 posti a 0)

CARICAMENTO DI COSTANTE / INDIRIZZO IN REGISTRO – pseudo-istruzioni (vedi nota 2 sotto)

li	$1, cost	$1 := cost (32 bit)	carica costante a 32 bit
la	$1, indir	$1 := indir (32 bit)	carica indirizzo a 32 bit

REGISTRI MIPS

REGISTRI REFERENZIABILI

0	*0*	costante 0 (denotabile anche come $zero)
1	*at*	uso riservato all'assembler-linker (per espandere pseudo-istruzioni e macro)
2 - 3	*v0 - v1*	valore restituito da funzione (v0 per dati di tipo scalare, si aggiunge v1 per numeri reali di tipo double)
4 - 7	*a0 - a3*	argomenti in ingresso a funzione (max quattro)
8 - 15	*t0 - t7*	registri per valori temporanei (p.es. calcolo delle espressioni)
16 - 23	*s0 - s7*	registri usabili (se possibile) per variabili locali scalari di sottoprogramma
24 - 25	*t8 - t9*	registri per valori temporanei (in aggiunta a t0 - t7), come i precedenti tx
26 - 27	*k0 - k1*	registri riservati per il nucleo (kernel) del Sistema Operativo
28	*gp*	global pointer (puntatore all'area dati globale)
29	*sp*	stack pointer (puntatore alla pila)
30	*fp*	frame pointer (puntatore all'area di attivazione di sottoprogramma)
31	*ra*	return address (indirizzo di rientro da chiamata a sottoprogramma)

REGISTRI NON REFERENZIABILI

	pc	program counter (contatore di programma)
	hi	registro per risultato di moltiplicazione e divisione (32 bit più significativi)
	lo	registro per risultato di moltiplicazione e divisione (32 bit meno significativi)

Nota 1: anche le pseudo-istruzioni *lh*, *lhu*, *sh*, *lb*, *lbu* e *sb* con etichetta sottintendono il registro $gp.

Nota 2: entrambe le pseudo-istruzioni *li* e *la* caricano un valore a 32 bit in un registro, ma *li* va usata per caricare una costante e *la* per caricare un indirizzo.

Uso dello stack

Riempimento dello Stack

Per effettuare operazioni di **Push** viene utilizzata l'istruzione **sw** (o *store word*).

Esempio: È necessario memorizzare nello Stack il contenuto dei registri $t0, $t1, $t2.

```
addiu $sp, $sp, -12  #Decremento il valore dello Stack Pointer di 12,
                      allocando spazio per 12 / 4 = 3 word.
sw $t0, 8($sp)       #Memorizzo il valore di $t0 all'indirizzo $sp + 8.
sw $t1, 4($sp)       #Memorizzo il valore di $t1 all'indirizzo $sp + 4.
sw $t2, 0($sp)       #Memorizzo il valore di $t2 all'indirizzo $sp + 0.
```

Svuotamento dello stack

Per effettuare operazioni di **Pop** viene utilizzata l'istruzione **lw** (o *load word*).

Esempio: È necessario rimuovere dallo stack i dati memorizzati nell'esempio precedente.

```
lw $t0, 8($sp)      #Carico il valore all'indirizzo $sp + 8 in $t0.
lw $t1, 4($sp)      #Carico il valore all'indirizzo $sp + 4 in $t1.
lw $t2, 0($sp)      #Carico il valore all'indirizzo $sp + 0 in $t2.
addiu $sp, $sp, 12  #Incremento il valore dello Stack Pointer di 12,
                     deallocando lo spazio occupato dalle 12 / 4 = 3 word.
```

Salvataggio dei registri

- $t0-$t9: registri temporanei, salvati dal programma chiamante.
- $s0-$s7: registri che devono essere preservati, salvati dal programma chiamato.
- $a0-$a3: registri che non sono preservati.
- $ra è un registro preservato.

SYSCALL

Service	Code in $v0	Arguments	Result
print integer	1	$a0 = integer to print	
print float	2	$f12 = float to print	
print double	3	$f12 = double to print	
print string	4	$a0 = address of null-terminated string to print	
read integer	5		$v0 contains integer read
read float	6		$f0 contains float read
read double	7		$f0 contains double read
read string	8	$a0 = address of input buffer $a1 = maximum number of characters to read	*See note below table*
exit (terminate execution)	10		
print character	11	$a0 = character to print	*See note below table*
read character	12		$v0 contains character read