

# MIPS Instruction Set 2

Laboratorio di Architettura

18 Maggio 2012

1 Architettura Mips

2 Chiamata a Funzione

3 Esercitazione

# Registri

## MIPS reference cards:

<http://refcards.com/docs/waetzigj/mips/mipsref.pdf>

<http://www.mips.com/media/files/MD00565-2B-MIPS32-QRC-01.01.pdf>

- 32 registri general purpose a 32bit:

Numero	Nome	Uso	Preservati dal chiamato?
\$0	\$zero	costante 0	N/A
\$1	\$at	temp assembler	No
\$2-\$3	\$v0-\$v1	retval funzioni e expr eval	No
\$4-\$7	\$a0-\$a3	args funzione	No
\$8-\$15	\$t0-\$t7	temp	No
\$16-\$23	\$s0-\$s7	temp da salvare	Si
\$24-\$25	\$t8-\$t9	temp	No
\$26-\$27	\$k0-\$k1	riservati kernel	No
\$28	\$gp	global pointer	Si
\$29	\$sp	stack pointer	Si
\$30	\$fp	frame pointer	Si
\$31	\$ra	return address	N/A

# Chiamata a Funzione

- Dato un semplice programma C che esegue una chiamata a funzione, traduciamolo in assembly e simuliamone il funzionamento.

```
/* variabili globali */  
int res;  
  
int double ( int a ) {  
    return ( a + a );  
}  
  
void main ( void ) {  
    res = double( 100 );  
}
```

# Regole per la Chiamata a Funzione

- La chiamata si esegue con l'istruzione **jal <label>**, che imposta anche l'indirizzo di ritorno
- Si possono specificare fino a 4 parametri tramite i registri **\$4-\$7**
- Ulteriori parametri si specificano utilizzando lo *stack*
- La funzione inizia con:

```
.ent <nome_funzione>  
<nome_funzione>:
```

e termina con:

```
.end <nome_funzione>
```

- Il valore di ritorno deve trovarsi nel registro **\$2** (e **\$3** se necessario)
- Per il ritorno dalla funzione si esegue **jr \$31**

# Regole Registri

Nell'utilizzo di chiamate a funzione e' importante attenersi alle regole di utilizzo dei registri:

- **\$2-\$3** modificabili per metterci i valori di ritorno dalla funzione
- **\$4-\$7** modificabili, contengono i parametri di chiamata alla funzione
- **\$8-\$15** modificabili, sono i temporanei
- **\$16-\$23** modificabili solo dopo averli salvati sullo stack
- **\$24-\$25** modificabili, sono i temporanei
- **\$28-\$31** modificabili solo dopo averli salvati sullo stack

# Traduzione Assembly

- MAIN

```

        .globl main
main:
        li      $4, 100 # carico il parametro per la funzione
        jal     double # setto il ret addr e salto alla funzione
        sw      $2, res # scrivo in memoria il risultato
        jr      $31    # ritorno al chiamante
        .end main

```

- DOUBLE

```

        .ent double
double:
        add     $2, $4, $4 # raddoppio il parametro della funzione
        jr      $31    # ritorno al chiamante
        .end double

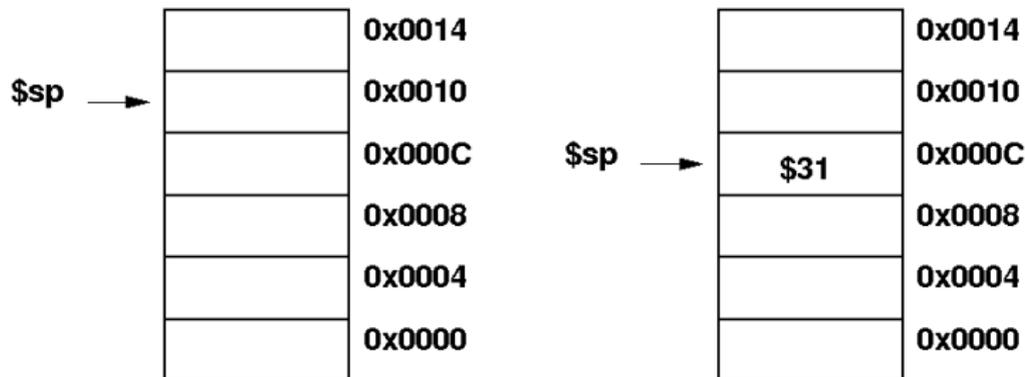
```

- Notate qualche problema? L'istruzione **jal** del *main* setta il return address per continuare con la **sw** successiva al ritorno da *double*, perdendo così il return address del chiamante del *main*.

## Lo Stack 1/3

- Il problema descritto precedentemente si risolve salvando il return address del chiamante di *main* sullo stack.
- Lo stack cresce dagli indirizzi alti verso quelli bassi quindi per allocare spazio per una variabile devo “decrementare” lo *stack pointer*.
- Prima di eseguire la chiamata a funzione (**jal**) salvo il valore di **\$31** sullo stack

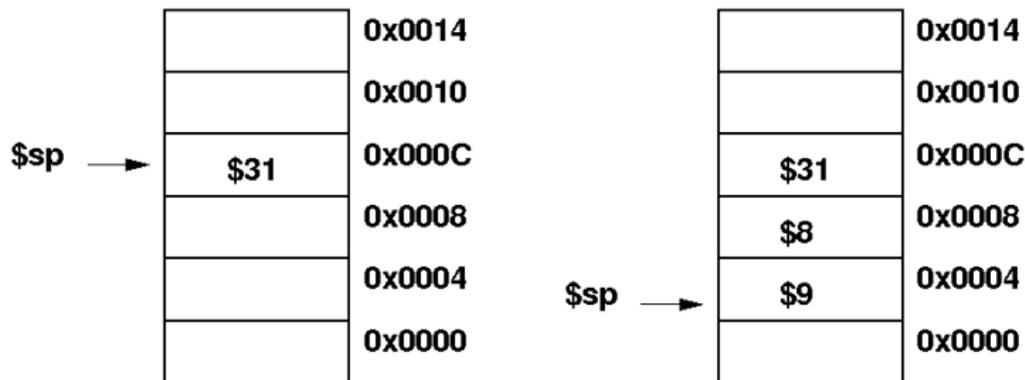
```
addi $sp,$sp,-4 # Alloco spazio per un int nello stack
sw   $31,0($sp) # Salvo ret addr sullo stack
```



## Lo Stack 2/3

- Le regole sui registri fissano \$4-\$7 per il passaggio di parametri ad una funzione, se devo passare alla funzione piu' di 4 parametri utilizzo lo stack per contenere i parametri eccedenti

```
addi $sp,$sp,-8 # Alloco spazio per due int nello stack
sw   $9,0($sp)  # Salvo parametro 6 sullo stack
sw   $8,4($sp)  # Salvo parametro 5 sullo stack
```



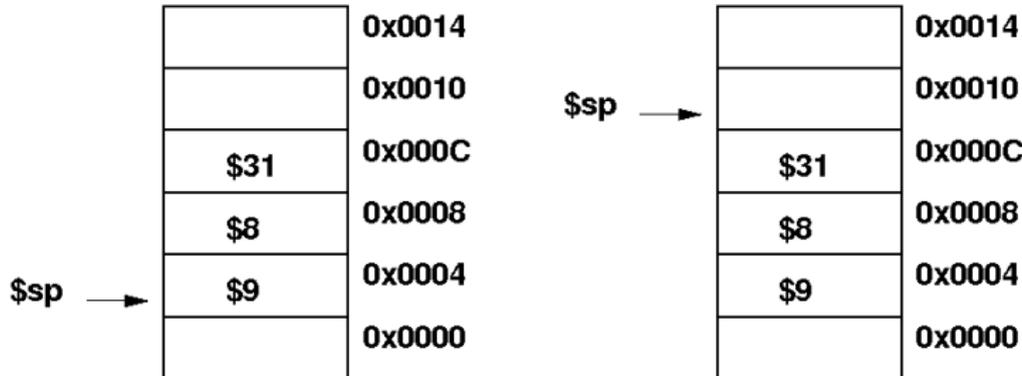
## Lo Stack 3/3

- Recupero i parametri che mi occorrono dallo stack

```
lw    $8,4($sp)  # Recupero parametro 5 dallo stack
lw    $9,0($sp)  # Recupero parametro 6 dallo stack
addi  $sp,$sp,8  # Rimuovo lo spazio per due int dallo stack
```

- Dopo la chiamata a funzione devo ripristinare il registro **\$31** recuperandolo dallo stack

```
lw    $31,0($sp) # Recupero ret addr dallo stack
addi  $sp,$sp,4  # Rimuovo lo spazio per un int dallo stack
```



- Qualunque funzione che ne richiami altre al suo interno e' tenuta a salvare sullo stack il return address e tutti i registri che vuole preservare.

# Traduzione Assembly con Stack

- MAIN

```

        .globl main
main:
    addi    $sp,$sp,-4 # Alloco spazio per un int nello stack
    sw     $31,0($sp) # Salvo ret addr sullo stack
    li     $4, 100    # carico il parametro per la funzione
    jal    double    # setto il ret addr e salto alla funzione
    sw     $2, res    # scrivo in memoria il risultato
    lw     $31,0($sp) # Recupero ret addr dallo stack
    addi    $sp,$sp,4 # Rimuovo lo spazio per un int dallo stack
    jr     $31        # ritorno al chiamante
    .end main

```

- DOUBLE

```

        .ent double
double:
    add     $2,$4,$4 # raddoppio il parametro della funzione
    jr     $31        # ritorno al chiamante
    .end double

```

## Codice Completo

```

        .text
        .align 2

        .ent double
double:
    add    $2,$4,$4    # raddoppio il parametro della funzione
    jr     $31         # ritorno al chiamante
        .end double

        .globl main
main:
    addi   $sp,$sp,-4 # Alloco spazio per un int nello stack
    sw     $31,0($sp) # Salvo ret addr sullo stack
    li     $4, 100    # carico il parametro per la funzione
    jal    double     # setto il ret addr e salto alla funzione
    sw     $2, res    # scrivo in memoria il risultato
    lw     $31,0($sp) # Recupero ret addr dallo stack
    addi   $sp,$sp,4  # Rimuovo lo spazio per un int dallo stack
    jr     $31         # ritorno al chiamante
        .end main

        .data    0x10002000
        .align 2
res:    .space 4

```

## Esercitazione 1/2

```
#define NUM 4

int A[NUM], prod, tmp;

// Funzione di shift a sinistra
int shift( int a, int b ) {
    return ( a << b );
}

// Funzione di produttoria ricorsiva
int prod_rec( int * data, int size ) {
    if ( size == 0 ) {
        tmp = 1;
        return tmp;
    } else {
        tmp = prod_rec( data, size - 1 ) * data[ size - 1 ];
        return tmp;
    }
}
```

## Esercitazione 2/2

```
int main(void) {  
    register int i, j;  
  
    // Inizializza array A  
    j = 2;  
    for( i = 0; i < NUM; i++ ) {  
        A[i] = shift( i+1, j);  
        j++;  
    }  
  
    prod = prod_rec( A, NUM );  
  
    return prod;  
}
```

# Scheletro di file assembly

```
.text                # PROLOGO
.align 2

.ent < nome_funz >   # FUNZIONI
< nome_funz >
  < codice funzione >
.end < nome_funz >

.globl main          # MAIN
main:
  < codice main >
  .end    main

.data                # DATI
.align 2

< variabili >
```

# Requisiti Fondamentali

- 1 Una volta scritto il file `es4.s` contenente l'assembler MIPS corrispondente alle funzioni `shift()`, `prod_rec()` e `main` **lo si commenta riga per riga** in modo tale che sia chiaro il funzionamento generale del programma.
- 2 Gli elementi dell'array `A` e il valore di `prod` **vanno mantenuti in memoria**, quindi se il loro valore viene **modificato** anche la memoria **deve essere aggiornata**.
- 3 Durante la simulazione con SPIM si catturi nel file `es4.txt` lo **stato dello stack** (valori) alla fine di ogni chiamata alla funzione `prod_rec()`, spiegando i valori modificati nei registri e nella memoria.
- 4 Al termine della simulazione con SPIM, si catturi nel file `es4.out` il **valore di tutti i registri** (finestra in alto di SPIM, tralasciare i registri floating point) **della memoria dati e dello stack** (finestra "data segment") spiegando i valori contenuti nei registri e nella memoria.

# Consegna

La relazione da consegnare e' formata da:

- 1 un file di testo es4.s contenente la traduzione in assembler MIPS del programma C
  - 2 un file di testo es4.txt con la copia dei segmenti DATI e STACK di SPIM per ogni attivazione di funzione
  - 3 un file di testo es4.out con la copia dei segmenti DATI e STACK di SPIM alla fine del programma
- NON usate la cattura di immagini da schermo: copiate ed incollate usando un editor di testo
  - NON includete file BINARI o creati con WORD, EXCEL...
  - NON verranno considerate mail con piu' o meno di 3 file allegati!
  - Evitate i file compressi.

Spedite tutto a [arc1@fe.infn.it](mailto:arc1@fe.infn.it) entro le ore 23:59:59 di giovedì 7 Giugno.

L'oggetto della mail **deve** essere nella forma:

LAB1-N#**esercitazione**-#**gruppo**

( es: LAB1-N**4-99**)