

## 6' Esercitazione

<https://politecnicomilano.webex.com/meet/gianenrico.conti>

Gian Enrico Conti  
ASM and functions

## Esercizi:

(2 H)

1) Frame pointer: concetto ed uso:

- Salvare i FP
- Recuperare FP
- Usare FP

2) esempi call a funzioni più complesse (con frame di attivazione completo)

3) esecuzione delle istruzioni nella architettura a singolo ciclo (altre slides)

# Registri e convenzioni: recap x Call

---

- Convenzioni sui registri
  - \$t0 - \$t9 ( = \$8 - \$15, \$24, \$25) are general use registers; need **not be preserved** across procedure calls
  - \$s0 - \$s7 ( = \$16 - \$23) are general use registers; **should be preserved** across procedure calls
  - \$sp ( = \$29) is stack pointer
  - \$fp ( = \$30) is frame pointer
  - \$ra ( = \$31) is return address storage for subroutine call (**R**eturn **A**ddress)
  - \$a0 - \$a3 ( = \$4 - \$7) are used to pass arguments to subroutines
  - \$v0, \$v1 ( = \$2, \$3) are used to hold return values from subroutine

# Registri e convenzioni (II)

- Sono Convenzioni adottate per il codice MIPS (non imposte da HW)
- Se tutto codice vs, uso "libero"
- Un programmatore MIPS appena arrivato difficoltà' ad adottarle

- Caller: SALVA
  - \$t0 - \$t9
  - \$a0 - \$a3 (params)
  - \$v0, \$v1 (return values)

- Calle ("funzione") SALVA
  - \$s0 - \$s7

- ATTENZIONE A" \$ra ( = \$31)
  - Contiene il return address (**R**eturn **A**ddress)
  - Se piu chiamate viene sovrascritto!

- 

\$t0 - \$t9 ( = \$8 - \$15, \$24, \$25) are general use registers;  
need **not be preserved** across procedure calls

\$s0 - \$s7 ( = \$16 - \$23) are general use registers;  
**should be preserved** across procedure calls

\$sp ( = \$29) is stack pointer

\$fp ( = \$30) is frame pointer

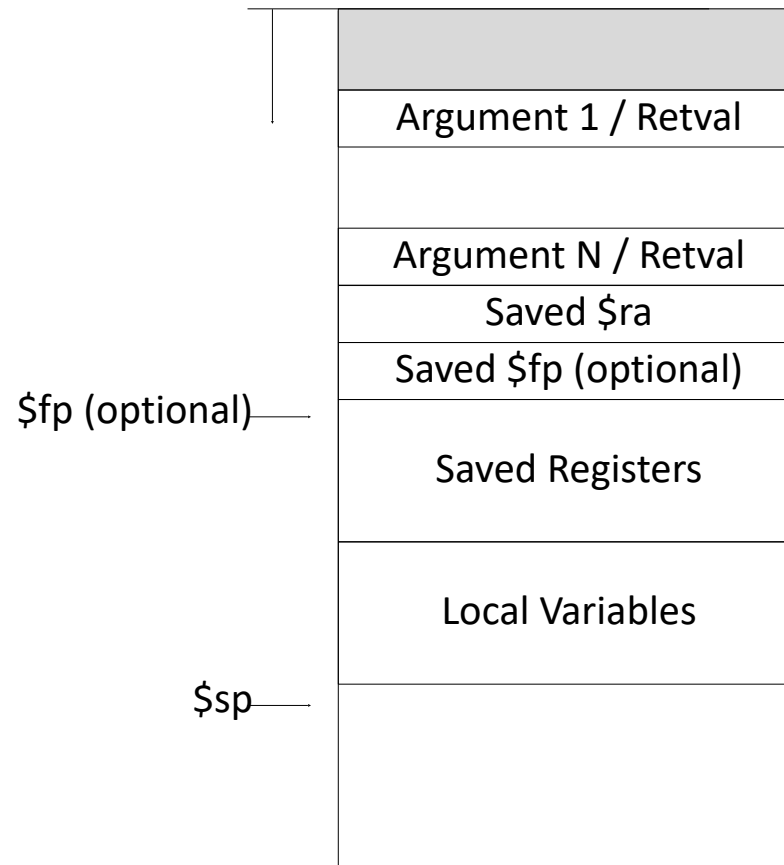
\$ra ( = \$31) is return address storage for subroutine call  
(**R**eturn **A**ddress)

\$a0 - \$a3 ( = \$4 - \$7) are used to pass arguments to  
subroutines

\$v0, \$v1 ( = \$2, \$3) are used to hold return values from  
subroutine

# Salvataggio di contesto

- Durante la chiamata a funzione può rendersi necessario salvare alcuni registri (contesto) per far sì che il chiamante continui a funzionare correttamente



# Alcuni esempi di codice

---

## Note:

- Attenzione alle f. annidate / ricorsive sono sia Caller che Callee !
- Se non bastano i registri, va usato STACK
- PROLOG / EPILOG (i.e. "entrata" / "uscita") (more about later)

## Sum up:

We call functions using **jal**, passing arguments in registers **\$a0-\$a3**.

Functions place results in **\$v0-\$v1** and return using **jr \$ra**.

# Chiamata a funzione: EX06\_es1\_CALL\_F.asm

---

```
int main( void )
{
    int a = leaf( 1, 2, 3, 4 );
}

int leaf( int g, int h, int i, int j )
{
    int f;

    f = (g + h) - (i + j);
    return f;
}
```

Note:

4 parametri, bastano i registri temp?  
Devo salvare registri (me ne bastano 3..)

In uscita devo ripristinare ..

# EX06\_es1\_CALL\_F.asm

```
int main( void )
{
    int a = leaf( 1, 2, 3, 4 );
}

int leaf(int g,int h,int i,int j )
{
    int f;
    f = (g + h) - (i + j);
    return f;
}
```

```
        .text
main:   addi $a0, $zero, 1
        addi $a1, $zero, 2
        addi $a2, $zero, 3
        addi $a3, $zero, 4
        jal  leaf
        ..

leaf:   addi $sp, $sp, -12

        sw   $t1, 8($sp)
        sw   $t0, 4($sp)
        sw   $s0, 0($sp)
```

Uso i registri x param.

Salto alla label "leaf"

Faccio "spazio" x t1, t0, s0  
(s0-s7 should be preserved across..  
salvo t1  
salvo t0  
salvo s0



# EX06\_es1\_CALL\_F.asm

```
int main( void )
{
    int a = leaf( 1, 2, 3, 4 );
}

int leaf(int g,int h,int i,int j )
{
    int f;
    f = (g + h) - (i + j);
    return f;
}
```

```
.text
main: addi $a0, $zero, 1
      addi $a1, $zero, 2
      addi $a2, $zero, 3
      addi $a3, $zero, 4
      jal  leaf
      ...

leaf: addi $sp, $sp, -12

      sw   $t1, 8($sp)
      sw   $t0, 4($sp)
      sw   $s0, 0($sp)

      add  $t0, $a0, $a1
      add  $t1, $a2, $a3
      sub  $s0, $t0, $t1
      add  $v0, $s0, $zero
      lw   $s0, 0($sp)
      lw   $t0, 4($sp)
      lw   $t1, 8($sp)
      addi $sp, $sp, 12
      jr   $ra
```

Uso i registri x param.

Salto alla label "leaf"

Faccio "spazio" x t1, t0, s0  
(s0-s7 should be preserved across calls)  
salvo t1  
salvo t0  
salvo s0

Add ...

Salvo ris. (f) in v0  
Ripristino regs.

Ripristino stack  
ret

# EX06\_es1\_CALL\_F.asm

```
int main( void )
{
    int a = leaf( 1, 2, 3, 4 );
}

int leaf(int g,int h,int i,int j )
{
    int f;
    f = (g + h) - (i + j);
    return f;
}
```

```
.text
main: addi $a0, $zero, 1
      addi $a1, $zero, 2
      addi $a2, $zero, 3
      addi $a3, $zero, 4
      jal  leaf
      add  $s0, $v0, $zero

      addi $v0, $zero, 10
      syscall

leaf: addi $sp, $sp, -12

      sw   $t1, 8($sp)
      sw   $t0, 4($sp)
      sw   $s0, 0($sp)

      add  $t0, $a0, $a1
      add  $t1, $a2, $a3
      sub  $s0, $t0, $t1
      add  $v0, $s0, $zero
      lw   $s0, 0($sp)
      lw   $t0, 4($sp)
      lw   $t1, 8($sp)
      addi $sp, $sp, 12
      jr   $ra
```

Uso i registri x param.

Salto alla label "leaf"  
\$v0 contiene il risultato,  
Lo copio in \$s0

param. X exit.  
exit

Faccio "spazio" x t1, t0, s0  
(s0-s7 should be preserved across  
salvo t1  
salvo t0  
salvo s0

Add ...

Salvo ris. (f) in v0  
Ripristino regs.

Ripristino stack  
ret

# EX06\_es1.asm

---

Traduciamo:

```
int x,y;
```

```
int sum(int a, int b){  
    return a+b;  
}
```

```
int main() {  
    x = 10;  
    y = sum(x, 20);  
    printf("%d", y);  
    return 0;  
}
```

# EX06\_es1.asm

---

Traduciamo:

```
int x,y;
```

```
int sum(int a, int b){  
    return a+b;  
}
```

```
int main() {  
    x = 10;  
    y = sum(x, 20);  
    printf("%d", y);  
    return 0;  
}
```

Note:

2 globali

Carico le globali nei registri \$a0 - \$a3  
Non mi servono altri regs.

# EX06\_es1.asm

Traduciamo:

```
int x,y;
```

```
int main() {  
    x = 10;
```

```
    y = sum(x, 20);
```

```
    printf("%d", y);  
    return 0;
```

```
}
```

```
int sum(int a, int b){  
    return a+b;  
}
```

```
x:      .word      0  
y:      .word      0
```

```
main:  
    la      $s0, x  
    li      $s1, 10  
    sw      $s1, ($s0)
```

```
#Caller prolog:  
    addi    $a0, $zero, $s1 # x  
    addi    $a1, $zero, 20  
    jal     sum
```

```
(Syscall. Not now..)
```

# EX06\_es2\_CALL\_sum.asm

Traduciamo:

```
int x,y;
```

```
int main() {  
    x = 10;  
    y = sum(x, 20);  
    printf("%d", y);  
    return 0;  
}
```

```
int sum(int a, int b){
```

```
    return a+b;  
}
```

```
.data:  
x:    .word    0  
y:    .word    0  
  
.text  
main:  
    la    $s0, x  
    li    $s1, 10  
    sw    $s1, ($s0)  
  
#Caller prolog:  
    add   $a0, $zero, $s1 # x  
    add   $a1, $zero, 20  
    jal   sum  
  
#(Syscall. Not now..)
```

```
sum:  
    add   $v0, $a0, $a1  
    jr    $ra
```

# EX06\_es2\_CALL\_sum.asm

Traduciamo:

```
int x,y;
```

```
int main() {  
    x = 10;  
    y = sum(x, 20);
```

```
    printf("%d", y);  
    return 0;
```

```
}
```

```
int sum(int a, int b){  
    return a+b;  
}
```

```
.data:  
x:    .word    0  
y:    .word    0
```

```
.text  
main:  
    la    $s0, x  
    li    $s1, 10  
    sw    $s1, ($s0)
```

```
#Caller prolog:  
    add   $a0, $zero, $s1 # x  
    add   $a1, $zero, 20  
    jal   sum
```

#save in Y:

```
    la    $s0, y  
    sw    $v0, ($s0)
```

```
sum:  
    add   $v0, $a0, $a1  
    jr    $ra
```

# EX06\_es2\_CALL\_sum.asm

Traduciamo:

```
int x,y;
```

```
int main() {  
    x = 10;  
    y = sum(x, 20);
```

```
    printf("%d", y);  
    return 0;
```

```
}
```

```
int sum(int a, int b){  
    return a+b;  
}
```

```
.data:  
x:    .word    0  
y:    .word    0  
  
.text  
main:  
    la    $s0, x  
    li    $s1, 10  
    sw    $s1, ($s0)  
  
#Caller prolog:  
    add   $a0, $zero, $s1 # x  
    add   $a1, $zero, 20  
    jal   sum  
  
#save in Y:  
    la    $s0, y  
    sw    $v0, ($s0)
```

#place sys call! 0therwise..

```
sum:  
    add   $v0, $a0, $a1  
    jr    $ra
```



# EX06\_es2\_CALL\_sum.asm

Traduciamo:

```
int x,y;
```

```
int main() {  
    x = 10;  
    y = sum(x, 20);
```

```
    printf("%d", y);
```

```
    return 0;
```

```
}
```

```
int sum(int a, int b){  
    return a+b;  
}
```

```
.data:  
x:    .word    0  
y:    .word    0  
  
.text  
main:  
    la    $s0, x  
    li    $s1, 10  
    sw    $s1, ($s0)
```

```
#Caller prolog:  
    add   $a0, $zero, $s1 # x  
    add   $a1, $zero, 20  
    jal   sum
```

```
#save in Y:  
    la    $s0, y  
    sw    $v0, ($s0)
```

```
li $v0,10 # selettore syscall  
main_:    syscall
```

```
sum:  
    add   $v0, $a0, $a1  
    jr    $ra
```

# EX06\_es2\_CALL\_sum.asm

Traduciamo:

```
int x,y;
```

```
int main() {  
    x = 10;  
    y = sum(x, 20);
```

```
    printf("%d", y);
```

```
    return 0;
```

```
}
```

```
int sum(int a, int b){  
    return a+b;  
}
```

```
.data:  
x:    .word    0  
y:    .word    0  
  
.text  
main:  
    la    $s0, x  
    li    $s1, 10  
    sw    $s1, ($s0)
```

```
#Caller prolog:  
    add   $a0, $zero, $s1 # x  
    add   $a1, $zero, 20  
    jal   sum
```

```
#save in Y:  
    la    $s0, y  
    sw    $v0, ($s0)
```

```
#printf: sys call: selettore: $v0 = 1, arg $a0
```

```
    move  $a0, $v0 # save Y  
    li    $v0, 1  
    syscall
```

```
li $v0,10 # selettore syscall  
main_:  syscall
```

```
sum:  
    add   $v0, $a0, $a1  
    jr    $ra
```

# EX06\_es2\_CALL\_sum.asm: cross compiler

Traduciamo:

```
mips-linux-gnu-gcc test.c -S -o temp.asm
```

```
int x,y;
```

```
int main() {  
    x = 10;  
    y = sum(x, 20);  
    printf("%d", y);  
    return 0;  
}
```

```
int sum(int a, int b){  
    return a+b;  
}
```

```
sum:  
        .frame    $fp,8,$31  
vars= 0, regs= 1/0, args= 0, gp= 0  
        .mask     0x40000000,-4  
        .fmask    0x00000000,0  
        .set      noreorder  
        .set      nomacro  
        addiu     $sp,$sp,-8  
        sw        $fp,4($sp)  
        move      $fp,$sp  
        sw        $4,8($fp)  
        sw        $5,12($fp)  
        lw        $3,8($fp)  
        lw        $2,12($fp)  
        addu      $2,$3,$2  
        move      $sp,$fp  
        lw        $fp,4($sp)  
        addiu     $sp,$sp,8  
        j         $31
```

# EX06\_es3\_CALL\_fattoriale.asm

---

Traduciamo:

```
int main( void ){  
    int f = fattoriale( 5 );  
}  
  
int fattoriale( int n ){  
    if( n < 1 )  
        return 1;  
    else  
        return n * fattoriale(n-1);  
}
```

# EX06\_es3\_CALL\_fattoriale.asm

---

## Traduciamo:

```
int main( void ){
    int f = fattoriale( 5 );
}

int fattoriale( int n ){
    if( n < 1 )
        return 1;
    else
        return n * fattoriale(n-1);
}
```

## Note:

2 push in PROLOGO  
2 pop in EPILOGO  
Devo salvare n e ra

## In MIPS ASM:

- abbasso la stack di  $4+4 = 8$  byte
- sw di \$ra sullo stack
- sw di \$a0 sullo stack

## EPILOGO:

- lw di \$ra
- lw di \$s0
- alzo stack di 8

# EX06\_es3\_CALL\_fattoriale.asm

---

```
.data
```

```
.text
```

```
main:    addi $a0, $zero, 5 #n = 5
```

```
...
```

```
li $v0,10 # selettore syscall x return
```

```
main_:   syscall
```

# EX06\_es3\_CALL\_fattoriale.asm

---

```
.data
.text
main:    addi $a0, $zero, 5 # n = 5
        jal  fattoriale
        li  $v0, 10 # selettore syscall x return
main_:   syscall

fattoriale:
    addi $sp, $sp, -8 # CALLEE PROLOG
    sw   $ra, 4($sp)
    sw   $a0, 0($sp)
```

...

Aggiungiamo anche epilogo e proviamo ...

# EX06\_es3\_CALL\_fattoriale.asm

---

```
.data
.text
main:    addi $a0, $zero, 5 # n = 5
        jal  fattoriale
        li  $v0, 10 # selettore syscall x return
main_:   syscall

fattoriale:
    addi $sp, $sp, -8 # CALLEE PROLOG
    sw   $ra, 4($sp)
    sw   $a0, 0($sp)

    lw   $a0, 0($sp) # CALLEE EPILOG
    lw   $ra, 4($sp)
    addi $sp, $sp, 8
    jr   $ra
```



# EX06\_es3\_CALL\_fattoriale.asm

---

#0ra la f. Vera propria"

```
.text
main:  addi    $a0, $zero, 5 # n = 5
        jal    fattoriale
        li     $v0, 10 # selettore syscall x return
main_:  syscall
```

```
fattoriale:
    addi    $sp, $sp, -8 # CALLEE PROLOG
    sw      $ra, 4($sp)
    sw      $a0, 0($sp)

    slti    $t0, $a0, 1    # set if less of 1, if( n < 1)
    beq     $t0, $zero, L1
```

#load 1 and return:

...

```
lw      $a0, 0($sp) # CALLEE EPOLOG
lw      $ra, 4($sp)
addi    $sp, $sp, 8
jr      $ra
```

# EX06\_es3\_CALL\_fattoriale.asm

---

```
.text
main:  addi    $a0, $zero, 5 # n = 5
        jal    fattoriale
        li     $v0, 10 # selettore syscall x return
main_:  syscall
```

```
fattoriale:
    addi    $sp, $sp, -8 # CALLEE PROLOG
    sw      $ra, 4($sp)
    sw      $a0, 0($sp)

    slti    $t0, $a0, 1    # set if less of 1, if( n < 1)
    beq     $t0, $zero, L1
```

```
    li      $v0, 1        # return 1
    addi    $sp, $sp, 8
    jr      $ra
```

**L1:**

..

```
    lw      $a0, 0($sp) # CALLEE EPOLOG
    lw      $ra, 4($sp)
    addi    $sp, $sp, 8
    jr      $ra
```

# EX06\_es3\_CALL\_fattoriale.asm

---

```
.text
main:  addi    $a0, $zero, 5 # n = 5
        jal    fattoriale
        li     $v0, 10 # selettore syscall x return
main_:  syscall
```

```
fattoriale:
    addi    $sp, $sp, -8 # CALLEE PROLOG
    sw      $ra, 4($sp)
    sw      $a0, 0($sp)

    slti    $t0, $a0, 1    # set if less of 1, if( n < 1)
    beq     $t0, $zero, L1

    li      $v0, 1        # return 1
    addi    $sp, $sp, 8
    jr      $ra
```

```
L1:  addi    $a0, $a0, -1 # decrement, recursive pass...
```

```
    jal     fattoriale
```

```
    lw      $a0, 0($sp) # CALLEE EPOLOG: get back a0
```

```
    lw      $ra, 4($sp)
```

```
    addi    $sp, $sp, 8
```

```
    mul     $v0, $a0, $v0 # v0 already set in recursive pass..
```

```
    jr      $ra
```

# EX06\_es3\_CALL\_fattoriale.asm

---

```
.text
main:  addi  $a0, $zero, 5 # n = 5
       jal   fattoriale

       move $a0, $v0 # call printf
       li   $v0, 1
       syscall

       li $v0, 10 # selettore syscall x return
main_: syscall

fattoriale:
       addi  $sp, $sp, -8 # CALLEE PROLOG
       sw    $ra, 4($sp)
       sw    $a0, 0($sp)

       slti  $t0, $a0, 1 # set if less of 1, if( n < 1)
       beq   $t0, $zero, L1

       li    $v0, 1 # return 1
       addi  $sp, $sp, 8
       jr    $ra

L1:    addi  $a0, $a0, -1 # decrement, recursive pass...
       jal   fattoriale
       lw    $a0, 0($sp) # CALLEE EPOLOG: get back a0
       lw    $ra, 4($sp)
       addi  $sp, $sp, 8
       mul   $v0, $a0, $v0 # v0 already set in recursive pass..
       jr    $ra
```

# EX06\_es3\_CALL\_fattoriale.asm

---

2 ultime note

- provate con  $n = 20$
- \$ra va salvato sempre o no?