Tutorato Architettura degli Elaboratori 06

Alberto Paparella[1]

15 Maggio 2025

[1]Dipartimento di Matematica e Informatica, Università degli studi di Ferrara

Funzioni

Insieme dei Registri del MIPS

Nome	N°	Utilizzo
`$0`	0	Il valore costante 0
`$at`	1	Temporaneo riservato all'assemblatore (assembler temporary)
`$v0-$v1`	2-3	Valori di ritorno delle funzioni
`$a0-$a3`	4-7	Argomenti delle funzioni
`$t0-$t7`	8-15	Temporanei
`$s0-$s7`	16-23	Variabili salvate
`$t8-$t9`	24-25	Altri temporanei
`$k0-$k1`	26-27	Temporanei riservati al sistema operativo
`$gp`	28	Puntatore globale (global pointer)
`$sp`	29	Puntatore allo stack (stack pointer)
`$fp`	30	Puntatore al frame (frame pointer)
`$ra`	31	Indirizzo di ritorno della funzione (return address)

Chiamate a funzioni

```c
int main() {
    simple();
    a = b + c;
}

void simple() {
    return;
}
```

Listing 1: Codice C.

```
0x00400200  main:   jal simple
0x00400204          add $s0, $s1, $s1
...
0x00401020  simple: jr $ra
```

Listing 2: Codice assembly MIPS.

Chiamate a funzioni

```
1 0x00400200  main:   jal simple
2 0x00400204          add $s0, $s1, $s1
3 ...
4 0x00401020  simple: jr $ra
```

Listing 3: Codice assembly MIPS.

- `jal` salta a `simple` e setta `$ra` = PC + 4 = 0x00400204
- `jr $ra` salta all'indirizzo contenuto in `$ra` (0x00400204)

Argomenti in ingresso e valore di ritorno

- **Convenzioni MIPS:**
  - valore degli argomenti: $a0-$a3
  - valore di ritorno: $v0-$v1

```c
int main() {
    int y;
    ...
    y = diffofsums(2, 3, 4, 5); // 4 argomenti
    ...
}

int diffofsums(int f, int g, int h, int i) {
    int result;
    result = (f + g) - (h + i)
    return result;  // Valore di ritorno
}
```

Listing 4: Codice C.

Argomenti in ingresso e valore di ritorno

```
1  # $s0 = y
2  main:
3      ...
4      addi    $a0, $0, 2      # Argomento 0 = 2
5      addi    $a1, $0, 3      # Argomento 1 = 3
6      addi    $a2, $0, 4      # Argomento 2 = 4
7      addi    $a3, $0, 5      # Argomento 3 = 5
8      jal     diffofsums      # Chiamata a funzione
9      add     $s0, $v0, $0    # y = valore di ritorno
10     ...
11
12 # $s0 = result
13 diffofsums:
14     add $t0, $a0, $a1   # $t0 = f + g
15     add $t1, $a2, $a3   # $t1 = h + i
16     sub $s0, $t0, $t1   # result = (f + g) - (h + i)
17     add $v0, $s0, $0    # Valore di ritorno in $v0
18     jr  $ra             # Ritorna al chiamante
```

Listing 5: Codice assembly MIPS.

Argomenti in ingresso e valore di ritorno

```
1 # $s0 = result
2 diffofsums:
3     add $t0, $a0, $a1    # $t0 = f + g
4     add $t1, $a2, $a3    # $t1 = h + i
5     sub $s0, $t0, $t1    # result = (f + g) - (h + i)
6     add $v0, $s0, $0     # Valore di ritorno in $v0
7     jr  $ra              # Ritorna al chiamante
```

Listing 6: Codice assembly MIPS.

- `diffofsums` sovrascrive 3 registri: `$t0, $t1, $s0`
- Questo viene chiamato **effetto collaterale** o **side effect**
- `diffofsums` può usare lo **stack** per salvare tali registri

Salvataggio e ripristino del codice dallo stack

```
1  # $s0 = result
2  diffofsums:
3      addi    $sp, $sp, -12   # Allocare spazio sullo stack
       per memorizzare 3 registri
4      sw      $s0, 8($sp)     # Salva $s0 sullo stack
5      sw      $t0, 4($sp)     # Salva $t0 sullo stack
6      sw      $t1, 0($sp)     # Salva $t1 sullo stack
7      add     $t0, $a0, $a1   # $t0 = f + g
8      add     $t1, $a2, $a3   # $t1 = h + i
9      sub     $s0, $t0, $t1   # result = (f + g) - (h + i)
10     add     $v0, $s0, $0    # Valore di ritorno in $v0
11     lw      $t1, 0($sp)    # Ristora $t1 dallo stack
12     lw      $t0, 4($sp)    # Ristora $t0 dallo stack
13     lw      $s0, 8($sp)    # Ristora $s0 dallo stack
14     addi    $sp, $sp, 12    # Dealloca spazio dallo stack
15     jr  $ra                 # Ritorna al chiamante
```

Listing 7: Codice assembly MIPS.

Convenzioni sui registri

Preservati dalla procedura chiamata	**Non preservati, eventualmente salvati e ripristinati dal chiamante**
`$s0-$s7`	`$t0-$t9`
`$ra`	`$a0-$a3`
`$sp`	`$v0-$v1`
stack sopra `$sp`	stack sotto `$sp`

Salvataggio di registri da preservare sullo stack

- Seguendo la convenzione, si può salvare solo $s0

```
1  proc1:
2      addi    $sp, $sp, -4     # Alloca spazio sullo stack per
       salvare un registro
3      sw      $s0, 0($sp)      # Salva $s0 sullo stack
4      # Non c'e' bisogno di salvare $t0 o $t1
5      add     $t0, $a0, $a1    # $t0 = f + g
6      add     $t1, $a2, $a3    # $t1 = h + i
7      sub     $s0, $t0, $t1    # result = (f + g) - (h + i)
8      add     $v0, $s0, $0     # Valore di ritorno in $v0
9      lw      $s0, 0($sp)     # Ristora $s0 dallo stack
10     addi    $sp, $sp, 4     # Dealloca spazio dallo stack
11     jr  $ra                  # Ritorna al chiamante
```

Listing 8: Codice assembly MIPS.

Chiamate a funzioni innestate

```
1 proc1:
2     addi    $sp, $sp, -4      # Alloca spazio sullo stack
3     sw      $ra, 0($sp)       # Salva $ra sullo stack
4     jal     proc2
5     ...
6     lw      $ra, 0($sp)       # Ristora $ra dallo stack
7     addi    $sp, $sp, 4       # Dealloca spazio dallo stack
8     jr      $ra               # Ritorna al chiamante
```

Listing 9: Codice assembly MIPS.

Chiamate di funzioni ricorsive

```c
int sum(int n) {
    if (n <= 1)
        return 1;
    else
        return (n + sum(n-1));
}
```

Listing 10: Codice C.

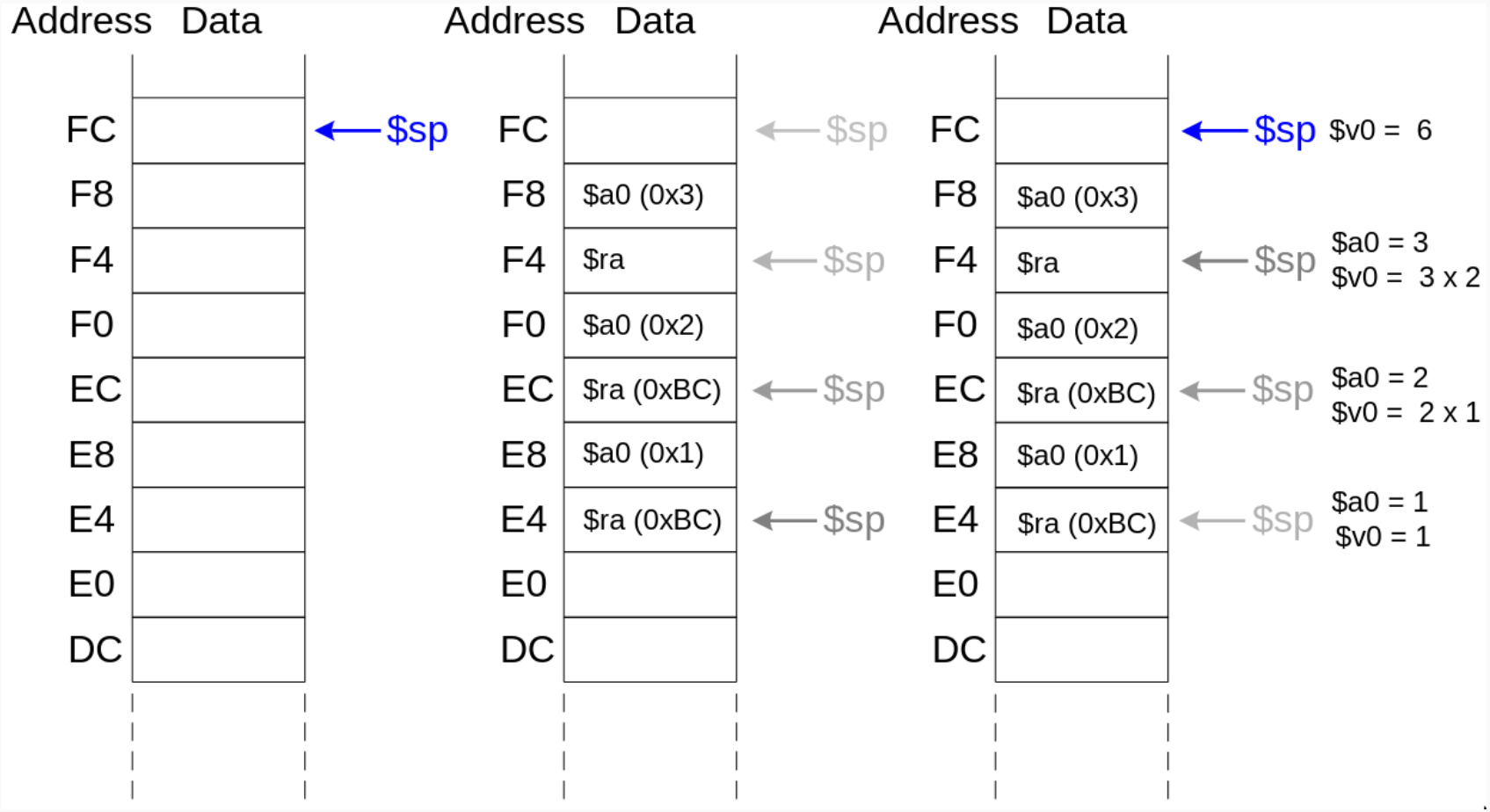
Chiamate di funzioni ricorsive

```
1  sum:
2      addi    $sp, $sp, -8     # Alloca spazio sullo stack
3      sw      $a0, 4($sp)      # Salva $a0
4      sw      $ra, 0($sp)      # Salva $ra
5      addi    $t0, $0, 2
6      slt     $t0, $a0, $t0    # a <= 1 ?
7      beq     $t0, $0, else    # no: salta ad else
8      addi    $v0, $0, 1       # yes: return 1
9      addi    $sp, $sp, 8      # Ristora $sp
10     jr      $ra              # Ritorna
11 else:
12     addi    $a0, $a0, -1     # n = n - 1
13     jal     sum              # Chiamata ricorsiva
14     lw      $ra, 0($sp)      # Ristora $ra
15     lw      $a0, 4($sp)      # Ristora $a0
16     addi    $sp, $sp, 8      # Ristora $sp
17     add     $v0, $a0, $v0    # n + sum(n-1)
18     jr      $ra              # Ritorna
```

Listing 11: Codice assembly MIPS.

Stack durante la chiamata ricorsiva

- Procedura chiamata inizialmente con il valore 3



Scansione di un array lineare alla ricerca del massimo e del minimo

Esercizio 1 (calcolo del massimo)

Tradurre il seguente codice C per la scansione di un array lineare alla ricerca dell'elemento maggiore in ASM (continua alla slide successiva):

```c
#include <stdio.h>

// Calcolo del massimo
int max(int a[], int dim) {
    int max = NULL;
    int i;

    for (i = 0; i < dim; i++) {
        if (max < a[i]) {
            max = a[i];
        }
    }

    return max;
}
```

Listing 12: Codice C per il calcolo del massimo (1).

Esercizio 1 (calcolo del massimo)

```c
// Programma principale per verificare le funzioni
int main() {
    int massimo;
    int a[10] = {10,11,13,14,17,9,7,1,4,99};

    massimo = max(a, 10);
    printf("\nIl massimo dell'array e' $d\n", massimo);

    return 0;
}
```

Listing 13: Codice C per il calcolo del massimo (2).

Soluzione

```
.data
    array:   .word    10, 11, 13, 14, 17, 9, 7, 1, 4, 99
    size:    .word    10
    SDR:     .asciiz "Il maggiore vale \n"

.text
    Main:
        lw  $a0, size
        la  $a1, array
        jal trovamaggiore
        add $t0, $v0, $zero # Mi salvo il valore di ritorno perche' lo vado a sovrascrivere per chiamare una syscall
        la  $a0, SDR
        li  $v0, 4
        syscall
        add $a0, $t0, $zero
        li  $v0, 1
        syscall
        li  $v0, 10
        syscall
```

Listing 14: Codice assembly MIPS per il calcolo del massimo (1).

Soluzione

```
1  trovamaggiore:
2      li   $t3, 0  # Indice in byte dell'array e riutilizzo per l'
       indirizzo assoluto del primo elemento
3      li   $t2, 0  # Memorizza il maggiore
4      li   $t1, 0  # Memorizza gli elementi dell'array
5      li   $t0, 0  # Indice di scorrimento dell'array
6  Forloop:
7      beq $t0, $a0, Endforloop    # Se l'indice corrisponde al numero
       di elementi, esco
8      sll $t3, $t0, 2     # Formo l'indice per byte
9      add $t3, $a1, $t3   # Indirizzo elemento
10     lw  $t1, 0($t3)
11     # if-then
12     slt $t4, $t1, $t2   # $t4 vale 1 se temp < maggiore
13     bne $t4, $zero, nulladafare
14     add $t2, $t1, $zero # Aggiorna il max
15 nulladafare:
16     addi    $t0, $t0, 1
17     j       Forloop
18 Endforloop:
19     add $v0, $t2, $zero
20     jr  $ra
```

Listing 15: Codice assembly MIPS per il calcolo del massimo (2).

Momenti salienti

Incremento dell'indice di scansione di un array di 4 byte alla volta **moltiplicando l'indice incrementale tramine "sll"**.

- In assembler non basta l'indice *i* per muoverci fra gli elementi di un array: sappiamo infatti che ogni elemento sarà allocato in modo sequenziale ed occuperò esattamente 4 byte.
- Per questo dobbiamo non solo tenere traccia del valore dell'indice *i*, in modo da incrementarlo ad ogni ciclo di 1, ma dovremo poi moltiplicarlo per 4 prima di sommarlo all'indirizzo di memoria a cui eravamo arrivati, partendo dall'indirizzo del primo elemento dell'array.
- Ciò può essere fatto facilmente con l'istruzione "sll", che "shifta" il valore contenuto nel registro di due posizioni, aggiungendo due zeri come bit meno significativi, effettuando di fatto una moltiplicazione per 4.

Momenti salienti

- **Nel nostro esercizio**, *i* è contenuto nel registro `$t0` ed è inizializzato a 0. Ad ogni ciclo, prima di tutto controllo se *i* è uguale al numero di elementi dell'array; nel caso sia vero significa infatti che sono arrivato alla fine dell'array, e devo quindi uscire dal ciclo for.
- Dopodichè, la prima cosa che faccio è moltiplicare il valore di *i* per 4 utilizzando l'istruzione "sll", salvando il risultato in un altro registro, in questo caso `$s3`.
- Nella riga successiva vado poi ad aggiungere questo valore all'indirizzo del primo elemento dell'array: in questo modo posso calcolare efficientemente ad ogni ciclo dove si trova in memoria l'elemento dell'array di cui ho bisogno, scorrendo ad ogni ciclo l'array di una posizione.
- Infine, prima di effettuare la jump all'inizio del Forloop e quindi ripetere il ciclo, vado ad aggiornare il valore di *i* in `$t0`, che non è stato toccato.

Momenti salienti


```
# allocato all'indirizzo 0x10010000 (base)
# allocato all'indirizzo 0x10010004 (base + 4)
# allocato all'indirizzo 0x10010008 (base + 8)
# ...
# allocato all'indirizzo base + (n-1)*4
```

Momenti salienti

Realizzazione del ciclo “if-then” in Assembler.

- Una volta caricato il nuovo elemento dell’array in `$t1`, lo confronto con quello che al momento considero il maggiore, contenuto in `$t2`, ed inizializzato a 0, utilizzando l’istruzione “slt” e salvando il risultato in `$s4`.
- Questo sarà 1 se il valore del primo registro (`$t1`) è minore del valore contenuto nel secondo (`$t2`), e quindi semanticamente se l’elemento corrente è minore di quello considerato massimo.
- Alla riga successiva indico infatti che se `$t4` non contiene 0 (e quindi se contiene 1, in pratica se l’elemento corrente non è il nuovo massimo), allora salta all’etichetta “nulladafare”

Momenti salienti

- In pratica, sto saltando l'istruzione di aggiornamento del massimo ed andando direttamente alla fine del ciclo for, aggiornando l'indice e ricominciando un nuovo ciclo.
- Se invece `$t4` contiene proprio 0, allora `$t1` è il nuovo massimo, e devo salvare il suo contenuto in `$t2` prima di aggiornare l'indice e cominciare un nuovo ciclo.
- **Nota bene:** dopo l'aggiornamento del massimo, il programma entrerà esattamente in "nulladafare"!

Esercizio 2 (calcolo del minimo) - per casa

Tradurre il seguente codice C per la scansione di un array lineare alla ricerca dell'elemento minore in ASM (continua alla slide successiva):

```c
#include <stdio.h>

// Calcolo del minimo
int min(int a[], int dim) {
    int min = NULL;
    int i;

    for (i = 0; i < dim; i++) {
        if (min > a[i]) {
            min = a[i];
        }
    }

    return min;
}
```

Listing 16: Codice C per il calcolo del minimo (1).

Esercizio 2 (calcolo del minimo) - per casa

```c
// Programma principale per verificare le funzioni
int main() {
    int minimo;
    int a[10] = {10,11,13,14,17,9,7,1,4,99};

    minimo = min(a, 10);
    printf("\nIl minimo dell'array e' $d\n", minimo);

    return 0;
}
```

Listing 17: Codice C per il calcolo del minimo (2).