Tutorato di Informatica

Foglio 3+ - Assembly

Matematici I Anno

25 Marzo 2026

"Me: I'm so sorry, my dog ate my homework.
Computer Science Professor: your dog ate your coding assignment?
Me:
Prof:
Me: it took him a couple bytes."

Esercizi di Ripasso e Approfondimento su Assembly

Esercizio 01: Ricerca di un elemento in un Array

Completare il programma `ricerca_in_array.asm` che cerca il valore **5** all'interno di un array di 10 numeri interi. Caricare il file su QtSpim, eseguirlo passo passo e verificarne il funzionamento. **Test:** Per verificare la robustezza del codice, provare a modificare il contenuto del vettore nel segmento `.data` affinché contenga (o non contenga affatto) il valore cercato.

Spiegazione (File: ricerca_in_array.asm): L'algoritmo utilizza un ciclo per scorrere la memoria. È fondamentale ricordarsi di incrementare il puntatore all'indirizzo di 4 byte ad ogni iterazione, poiché ogni `.word` occupa 4 indirizzi di memoria.

```
.data
array:           .word 3, 7, 1, 4, 9, 6, 8, 2, 0, 5  # Vettore di test
str_trovato:     .asciiz "Valore trovato!\n"
str_non_trovato: .asciiz "Valore non presente nell'array.\n"

.text
.globl main

main:
    li $t0, 0           # Indice i = 0
    li $t1, 10          # Dimensione array
    li $t2, 5           # Valore da cercare
    la $t3, array       # Carico indirizzo base array

loop:
    beq $t0, $t1, fine_not_found  # Se i == 10, fine ciclo (non trovato)

    lw $t4, 0($t3)      # Leggo l'elemento corrente
    beq $t4, $t2, fine_found      # Se elemento == 5, salta a trovato

    addi $t3, $t3, 4    # Incremento puntatore memoria (+4 byte)
    addi $t0, $t0, 1    # i++
    j loop

fine_found:
    li $v0, 4
    la $a0, str_trovato
    syscall
```

```
    j exit

fine_not_found:
    li $v0, 4
    la $a0, str_non_trovato
    syscall

exit:
    li $v0, 10
    syscall
```

Esercizio 02: Interazione Utente e Confronto

Scrivere un programma in assembly che esegua le seguenti operazioni:

1. Leggere due numeri interi (`num1` e `num2`) usando l'apposita **syscall**.
2. Confrontare i due valori inseriti.
3. Stampare a schermo il più **piccolo** (il minimo) dei due.
4. Caricare il codice su QtSpim e testarne la correttezza.

Extra: Aggiungere delle stampe a video (istruzioni per l'utente) prima di ogni lettura e prima del risultato finale (usando le direttive `.data` e `.asciiz`).

Spiegazione (File: interazione_utente.asm): Si noti come il valore restituito da `syscall 5` (read_int) debba essere spostato da `$v0` a un altro registro prima di effettuare una nuova chiamata di sistema, per evitare di sovrascriverlo.

```
.data
prompt1:    .asciiz "Inserisci il primo numero: "
prompt2:    .asciiz "Inserisci il secondo numero: "
msg_res:    .asciiz "Il numero piu' piccolo e': "

.text
.globl main

main:
    # Lettura primo numero
    li $v0, 4
    la $a0, prompt1
    syscall
    li $v0, 5
    syscall
    move $t0, $v0       # Salvo num1 in $t0

    # Lettura secondo numero
    li $v0, 4
    la $a0, prompt2
    syscall
    li $v0, 5
    syscall
    move $t1, $v0       # Salvo num2 in $t1

    # Confronto
    blt $t0, $t1, t0_minore  # Se $t0 < $t1 salta
    move $a0, $t1            # Altrimenti il minimo e' $t1
    j stampa

t0_minore:
    move $a0, $t0            # Il minimo e' $t0

```

```
stampa:
    move $t2, $a0           # Salvo temporaneamente il minimo
    li $v0, 4
    la $a0, msg_res
    syscall

    li $v0, 1
    move $a0, $t2           # Ripristino il minimo per la stampa
    syscall

    li $v0, 10
    syscall
```

Esercizio 03: Passaggio di parametri (Per Valore vs Per Indirizzo)

Testo: L'obiettivo di questo esercizio è comprendere la differenza tra il passaggio di parametri per *valore* e per *riferimento (o indirizzo)*.

1. Scaricare il file `somma1.asm`. Leggere il codice, caricarlo su QtSpim ed eseguirlo passo passo (usando il tasto F10 o "Step"). Osservare cosa viene caricato nei registri `$a0` e `$a1` prima della chiamata a procedura (`jal`).

2. Scaricare il file `somma2.asm`. Fare la stessa cosa: caricarlo, eseguirlo passo passo e osservare il contenuto dei registri `$a0` e `$a1`.

3. Qual è la differenza fondamentale tra i due approcci dal punto di vista dell'accesso alla memoria all'interno della procedura?

Spiegazione: Nel file `somma1.asm`, i parametri vengono passati **per valore**. L'istruzione `lw $a0, num1` va a leggere direttamente il *contenuto* della memoria (il numero 50) e lo inserisce nel registro. La procedura riceve quindi i numeri pronti per essere sommati.

Nel file `somma2.asm`, i parametri vengono passati **per indirizzo**. L'istruzione `la $a0, num1` (Load Address) inserisce nel registro non il numero 50, ma *l'indirizzo di memoria* in cui si trova il 50. Pertanto, all'interno della procedura `somma2`, è necessario fare una lettura in memoria (`lw $t0, 0($a0)`) per "dereferenziare" il puntatore e ottenere il valore effettivo prima di poter fare l'addizione.

```
# --- ESTRATTO DA somma1.asm (PER VALORE) ---
    lw $a0, num1        # a0 = 50 (VALORE)
    lw $a1, num2        # a1 = 14 (VALORE)
    jal somma1
    # ...
somma1:
    move $t0, $a0       # Copio direttamente i valori passati
    move $t1, $a1
    add $v0, $t0, $t1   # Sommo i valori
    jr $ra              # Ritorno

# --- ESTRATTO DA somma2.asm (PER INDIRIZZO) ---
    la $a0, num1        # a0 = Indirizzo di memoria di num1
    la $a1, num2        # a1 = Indirizzo di memoria di num2
    jal somma2
    # ...
somma2:
    lw $t0, 0($a0)      # Devo LEGGERE la memoria all'indirizzo a0 per avere 50
    lw $t1, 0($a1)      # Devo LEGGERE la memoria all'indirizzo a1 per avere 14
```

```
    add $v0, $t0, $t1   # Sommo i valori
    jr $ra              # Ritorno
```

Esercizio 04: Moduli Multipli e Visibilità delle Etichette

Testo: In progetti grandi, il codice assembly viene spesso diviso in più file. QtSpim permette di caricare più file consecutivamente nello stesso spazio di memoria.

1. Caricare su QtSpim **prima** il file `vector_mean.asm` e **poi** il file `mean_lib.asm`. Provare ad eseguire il programma.

2. Ora resettare completamente l'emulatore (File → Reinitialize and Load File). Caricare i file nell'ordine inverso: **prima** `mean_lib.asm` e **poi** `vector_mean.asm`. Eseguire il programma. Cosa succede e perché?

Spiegazione: Nel primo caso il programma funziona regolarmente. Nel secondo caso, QtSpim (che funge anche da linker di base) mostrerà un errore di risoluzione dei simboli e il programma non funzionerà. **Perché?** Nel file `vector_mean.asm` viene chiamata l'etichetta `mean` (`jal mean`), che si trova fisicamente in un altro file. Se l'etichetta `mean` non è dichiarata come globale (`.globl`), essa è "invisibile" all'esterno del proprio file. Se de-commentiamo la direttiva `.globl mean` all'inizio del file `mean_lib.asm`, rendiamo la procedura pubblica e risolviamo l'errore, permettendo al linker di collegare la chiamata `jal mean` alla giusta porzione di memoria indipendentemente dall'ordine di caricamento.

Esercizio 05: Costruzione Array e Media (Procedure Annidate)

Testo: Scrivere un programma completo in assembly MIPS che:

1. Dato un numero n definito in memoria, allochi uno spazio sufficiente per contenere un array di n interi (word).

2. Tramite un ciclo, popoli l'array con i numeri interi sequenziali da 1 a n (es. se $n = 5$, l'array conterrà `1, 2, 3, 4, 5`).

3. Chiami una procedura `mean` per calcolare la media dell'intero array, e la stampi a video.

Requisito: Si utilizzino le logiche delle procedure `sum` e `mean` viste precedentemente, **modificandole** affinché operino su dati di tipo `.word` (4 byte) invece che su `.byte` (1 byte). Caricare il file risolutivo su QtSpim e verificarne l'output.

Spiegazione (File: costruttore_array.asm): *Nota metodologica:* Rispetto alla libreria `mean_lib.asm` originale (che lavorava su array di `.byte` avanzando la memoria di 1 byte per volta tramite `lb`), qui stiamo salvando numeri interi grandi, quindi usiamo `.word`. Dobbiamo obbligatoriamente modificare l'incremento dell'indirizzo di memoria nella procedura `sum` a **+4 byte** e usare l'istruzione `lw` (Load Word). Inoltre, allochiamo un blocco vuoto in RAM usando `.space`.

```
.data
n:       .word 10           # Dimensione dell'array (n)
array:   .space 40          # Alloco 40 byte (10 interi * 4 byte ciascuno)
msg_ris: .asciiz "La media calcolata e': "

.text
.globl main

main:
    # 1. Inizializzazione variabili per il ciclo di costruzione
    lw $t0, n               # $t0 = n (valore limite, es. 10)
```

```
    la $t1, array           # $t1 = puntatore all'indirizzo base dell'array
    li $t2, 1               # $t2 = contatore (i = 1, partiamo dal numero 1)

ciclo_costruzione:
    bgt $t2, $t0, fine_costruzione  # Se contatore > n, esci dal ciclo

    sw $t2, 0($t1)          # Salvo il contatore (1, poi 2, ecc.) nell'array
    addi $t1, $t1, 4        # Mi sposto di 4 byte (una word) per la cella successiva
    addi $t2, $t2, 1        # Incremento il contatore (i++)
    j ciclo_costruzione     # Ripeti

fine_costruzione:
    # 2. Preparazione dei parametri per la procedura 'mean'
    la $a0, array           # Primo parametro ($a0): indirizzo base dell'array
    lw $a1, n               # Secondo parametro ($a1): numero di elementi (n)

    # Salvataggio su stack prima della chiamata a procedura
    addi $sp, $sp, -4
    sw   $ra, 0($sp)

    jal mean                # Chiamata alla procedura (il risultato tornerà in $v0)

    # Ripristino stack pointer e $ra
    lw $ra, 0($sp)
    addi $sp, $sp, 4

    # 3. Output dei risultati
    move $t3, $v0           # Salvo il risultato in $t3 per non perderlo

    li $v0, 4               # Syscall per stampare la stringa
    la $a0, msg_ris
    syscall

    li $v0, 1               # Syscall per stampare l'intero
    move $a0, $t3           # Metto il risultato da stampare in $a0
    syscall

    # Terminazione del programma
    li $v0, 10
    syscall

# ---------------------------------------------------------
# Procedura: mean
# Parametri: $a0 = indirizzo base array, $a1 = lunghezza array
# Ritorno:   $v0 = media matematica
# ---------------------------------------------------------
mean:
    # Poiche' mean chiama un'altra procedura (sum), DEVE salvare $ra nello stack
    addi $sp, $sp, -4
    sw   $ra, 0($sp)

    jal sum                 # Chiamo sum (parametri $a0 e $a1 gia' impostati)

    lw $ra, 0($sp)          # Ripristino $ra
    addi $sp, $sp, 4

    div $v0, $v0, $a1       # Media = Somma ($v0) / n ($a1)
    # L'istruzione 'div' mette il quoziente direttamente nel primo registro indicato

    jr $ra                  # Ritorno al main

# ---------------------------------------------------------
# Procedura: sum (MODIFICATA PER ARRAY DI WORD)
```

```
# Parametri: $a0 = indirizzo base array, $a1 = lunghezza array
# Ritorno:   $v0 = somma degli elementi
# ------------------------------------------------------
sum:
    move $t0, $a0           # $t0 = Puntatore scorrevole array
    li $t1, 0               # $t1 = Contatore del ciclo (i = 0)
    li $v0, 0               # $v0 = Accumulatore per la somma (inizializzato a 0)

ciclo_somma:
    beq $t1, $a1, fine_somma # Se abbiamo sommato 'n' elementi, termina

    lw $t2, 0($t0)          # Leggo la word dalla memoria
    add $v0, $v0, $t2       # somma = somma + elemento corrente

    addi $t0, $t0, 4        # ATTENZIONE: incremento di 4 (perche' array di word)
    addi $t1, $t1, 1        # Incremento il contatore (i++)
    j ciclo_somma           # Ripeti

fine_somma:
    # Procedura foglia (non chiama nient'altro), non serve salvare $ra
    jr $ra                  # Ritorno a mean
```

Esercizio 06: Calcolo del Fattoriale (Procedura Ricorsiva)

Testo: Scrivere un programma in assembly MIPS che calcoli il fattoriale di un numero n (es. $n = 3$) utilizzando una procedura **ricorsiva**.

1. Definire il valore di n e lo spazio per il risultato nella sezione `.data`.
2. Chiamare una procedura `fact` passando n come argomento (nel registro `$a0`).
3. All'interno di `fact`, gestire il salvataggio dei registri sullo **Stack**. Poiché la procedura chiama sé stessa, è obbligatorio salvare ad ogni iterazione l'indirizzo di ritorno (`$ra`) e il valore corrente di n (`$a0`).
4. Implementare la logica matematica: se $n \leq 1$, il caso base ritorna 1. Altrimenti, la procedura richiama sé stessa calcolando $n \times \text{fact}(n-1)$.
5. Caricare il programma su QtSpim ed eseguirlo passo passo tenendo d'occhio il pannello dei dati (sezione Stack) per vedere come la memoria si espande e si contrae.

Spiegazione (File: calcolo_fattoriale.asm): Il cuore di questo esercizio è la gestione dello **Stack**. Ogni volta che `fact` chiama sé stessa, il registro `$ra` viene sovrascritto dalla nuova `jal`, e anche il registro `$a0` viene modificato per passare $n-1$. Per non perdere i valori della "chiamata precedente", dobbiamo impilarli in memoria prima della chiamata ricorsiva, e "spilarli" (leggerli e rimuoverli) al ritorno.

```
.data
n:      .word 3         # Valore di partenza per cui calcolare n! (es. 3! = 6)
res:    .word 0         # Spazio in memoria per salvare il risultato finale

.text
.globl main

main:
    # Preparazione dell'argomento e chiamata
    lw   $a0, n         # Carica il valore di n in $a0 (argomento per fact)
    jal  fact           # Chiama la procedura ricorsiva fact(n)

    # Salvataggio del risultato e chiusura
```

```
    sw   $v0, res     # Il risultato restituito in $v0 viene salvato in 'res'
    li   $v0, 10      # Syscall 10: Termina il programma
    syscall

# ---------------------------------------------------------
# Procedura Ricorsiva: fact(n)
# Parametri: $a0 = n
# Ritorno:   $v0 = n!
# Logica:    se n <= 1 -> ritorna 1
#            altrimenti -> ritorna n * fact(n - 1)
# ---------------------------------------------------------
fact:
    # 1. Creazione dello Stack Frame (Prologo)
    # Dobbiamo salvare 2 registri ($ra e $a0), quindi ci servono 8 byte.
    addi $sp, $sp, -8        # Sposta lo Stack Pointer verso il basso
    sw   $ra, 4($sp)         # Salva l'indirizzo di ritorno al chiamante
    sw   $a0, 0($sp)         # Salva il valore attuale di n

    # 2. Controllo del Caso Base (n <= 1)
    li   $t0, 1              # Carica 1 in un registro temporaneo
    ble  $a0, $t0, base_case # Se n <= 1, salta all'etichetta base_case

    # 3. Passo Ricorsivo (n > 1)
    addi $a0, $a0, -1        # Calcola n - 1 (prepara il nuovo argomento in $a0)
    jal  fact                # CHIAMATA RICORSIVA: fact(n - 1)

    # 4. Calcolo post-ritorno: n * fact(n - 1)
    # Quando torniamo qui, in $v0 c'e' il risultato di fact(n - 1).
    # Il nostro 'n' originale era stato modificato, ma lo avevamo salvato nello stack!
    lw   $a0, 0($sp)         # Ripristina l'n originale di questo specifico frame
    mul  $v0, $a0, $v0       # Calcola: n * risultato_precedente. Salva in $v0

    # 5. Distruzione dello Stack Frame (Epilogo)
    lw   $ra, 4($sp)         # Ripristina l'indirizzo di ritorno originale
    addi $sp, $sp, 8         # Libera gli 8 byte allocati sullo stack
    jr   $ra                 # Ritorna alla funzione chiamante (main o un altro fact)

base_case:
    li   $v0, 1              # Caso base: il fattoriale di 1 (o 0) e' 1
    # Anche nel caso base dobbiamo ripulire lo stack che avevamo allocato al punto 1!
    lw   $ra, 4($sp)         # Ripristina $ra
    addi $sp, $sp, 8         # Libera lo spazio
    jr   $ra                 # Ritorna immediatamente
```

Esercizio 07: Inserimento in un Array (Codice da completare)

Testo: In questo esercizio vogliamo inserire un nuovo valore all'interno di un array preesistente, in una posizione specifica. Per fare spazio al nuovo elemento senza sovrascrivere i dati esistenti, è necessario "shiftare" (traslare) tutti gli elementi successivi di una posizione verso destra (ovvero di 4 byte in avanti).
Cosa fare: Scaricate il file `inserimento_array.asm` dalla pagina Moodle del corso. Il codice fornito è parziale: mancano le istruzioni cruciali all'interno del ciclo di shift e per l'inserimento finale. Sostituite i commenti contrassegnati da `# -- COMPLETA QUI --` con le istruzioni MIPS corrette. Infine, caricate lo script su QtSpim ed eseguitelo passo passo per verificare che l'array finale diventi `10, 20, 99, 30, 40`.

Soluzione (File completo: inserimento_array.asm): L'operazione di traslazione verso destra deve **obbligatoriamente** partire dall'ultimo elemento e procedere a ritroso, altrimenti si sovrascriverebbero i dati prima di averli copiati. Ecco il codice con le parti mancanti integrate:

```
.data
```

```
# Array con spazio extra alla fine (lo zero finale fa da "cuscinetto")
array:  .word 10, 20, 30, 40, 0
dim:    .word 4                 # Dimensione logica attuale
val:    .word 99                # Valore da inserire
pos:    .word 2                 # Indice in cui inserire (0-based, quindi al posto del 30)

.text
.globl main

main:
    la $t0, array               # Indirizzo base dell'array
    lw $t1, dim
    lw $t2, pos
    lw $t3, val

    # Calcolo l'indirizzo dell'ultimo elemento corrente (indice dim - 1)
    addi $t4, $t1, -1           # t4 = dim - 1
    sll $t4, $t4, 2             # Moltiplico per 4 (shift logico a sinistra di 2)
    add $t5, $t0, $t4           # t5 = Indirizzo dell'ultimo elemento

    # Calcolo l'indirizzo della posizione in cui inserire
    sll $t6, $t2, 2             # Moltiplico l'indice 'pos' per 4
    add $t6, $t0, $t6           # t6 = Indirizzo in cui dovra' andare il 99

loop_shift_right:
    blt $t5, $t6, fine_shift    # Se il puntatore scende sotto la pos di inserimento, stop

    # --- SOLUZIONE BLOCCHI 1 e 2 ---
    lw $t7, 0($t5)              # Leggo l'elemento attuale
    sw $t7, 4($t5)              # Lo salvo nella posizione successiva ($t5 + 4)
    # -------------------------------

    addi $t5, $t5, -4           # Arretro il puntatore all'elemento precedente
    j loop_shift_right

fine_shift:
    # --- SOLUZIONE BLOCCO 3 ---
    sw $t3, 0($t6)              # Inserisco il nuovo valore ($t3) all'indirizzo di dest. ($t6)
    # --------------------------

    # Aggiorno la dimensione
    addi $t1, $t1, 1
    sw $t1, dim

    li $v0, 10
    syscall
```

Esercizio 08: Conto alla Rovescia e Rimozione per Shift

Testo: Scrivere un programma in assembly MIPS che implementi un conto alla rovescia interattivo seguito dalla rimozione di un elemento da un array.

Il programma deve soddisfare i seguenti requisiti:

- Inizializzare in memoria un array di 5 interi (ad esempio `8, 15, 23, 42, 108`) e una variabile contenente la sua dimensione.
- Richiedere all'utente un numero di partenza tramite console.

- Implementare un ciclo che stampi a video un conto alla rovescia dal numero inserito fino a 0.
- Al termine del conteggio, il programma deve rimuovere fisicamente l'elemento in **posizione 1** (ovvero il secondo elemento dell'array, nell'esempio il `15`).

Spiegazione (File: countdown_rimozione.asm): A differenza dell'inserimento, la rimozione richiede uno shift verso **sinistra**. Il ciclo di copia parte dall'indice dell'elemento da sovrascrivere e procede in avanti fino alla fine dell'array copiando l'elemento successivo $(i + 1)$ in quello corrente (i). Nel ciclo del conto alla rovescia utilizziamo l'istruzione `beq` per verificare il raggiungimento dello zero.

```
.data
array:      .word 8, 15, 23, 42, 108
dim:        .word 5
prompt:     .asciiz "Inserisci i secondi per il countdown: "
newline:    .asciiz "\n"
msg_boom:   .asciiz "BOOM! Elemento rimosso.\n"

.text
.globl main

main:
    # Richiesta input utente
    li $v0, 4
    la $a0, prompt
    syscall

    li $v0, 5
    syscall
    move $t0, $v0               # $t0 = contatore per il countdown

loop_conto:
    # Stampa il numero corrente
    li $v0, 1
    move $a0, $t0
    syscall

    # Stampa a capo
    li $v0, 4
    la $a0, newline
    syscall

    # Controllo di uscita: se il contatore e' arrivato a 0, termina il countdown
    beq $t0, $zero, esegui_rimozione

    addi $t0, $t0, -1           # Decrementa contatore
    j loop_conto

esegui_rimozione:
    # Stampa messaggio rimozione
    li $v0, 4
    la $a0, msg_boom
    syscall

    # Impostazione dei puntatori per lo shift a sinistra
    la $t1, array               # Indirizzo base
    addi $t2, $t1, 4            # Puntatore alla cella da SOVRASCRIVERE (indice 1)

    lw $t3, dim
    addi $t3, $t3, -1           # Limite per il ciclo di shift (dim - 1)
    sll $t3, $t3, 2             # Moltiplico per 4 per ottenere l'offset in byte
    add $t4, $t1, $t3           # t4 = Indirizzo dell'ultimo elemento valido
```

```
52
53 loop_shift_left:
54     beq $t2, $t4, fine_prog     # Se il puntatore raggiunge l'ultimo elemento, esci
55
56     lw $t5, 4($t2)              # Leggo l'elemento successivo (i + 1)
57     sw $t5, 0($t2)              # Lo copio nella posizione corrente (i)
58
59     addi $t2, $t2, 4            # Avanzo al prossimo elemento
60     j loop_shift_left
61
62 fine_prog:
63     # Aggiornamento della dimensione logica in memoria
64     lw $t3, dim
65     addi $t3, $t3, -1
66     sw $t3, dim
67
68     li $v0, 10
69     syscall
```