Laboratorio di Architettura degli Elaboratori II

Corso di laurea triennale in Informatica
Università degli Studi di Milano, A.A. 2018-2019

Nicola.Basilico*@unimi.it*

Appello del 26 Luglio 2019

- L'esame ha una durata di 3 ore.
- È possibile consultare il libro di testo, appunti e la documentazione di MARS o Spim.
- È proibito l'accesso ad Internet con qualsiasi mezzo.
- Verranno corretti solo gli esercizi che non generano errori in compilazione ed esecuzione.
- I sorgenti vanno uploadati su https://upload.di.unimi.it/

1 Prodotto tra vettori

nome del file sorgente: dotproduct.asm

Si implementi la funzione `dotproduct` invocabile da altri file sorgenti e definita come segue.

- Input: nell'ordine, vettore `A`, intero `n`, vettore `B`, intero `m`; `n` e `m` indicano il numero di elementi in `A` e `B`, rispettivamente.
- Output: intero `s`.

Nel caso in cui `n` e `m` siano corretti (entrambi non negativi), l'intero `s` è calcolato come segue:

$$s = \sum_{i=0}^{\min\{n,m\}-1} A[i]B[i]$$

In caso contrario si adotti un comportamento a piacere.

1.1 Esempio

Con questa configurazione di input:

```
A=[1 2 3 4 5 6], n=6
B=[0 1 0 1 0 0 0 0 0 0 0 1], m=12
```

La funzione restituisce:

```
s=6
```

2 Stringa ben parentesizzata

nome del file sorgente: par.asm

Si implementi la funzione `par` che riceve in input una stringa `S` e restitiusce `1` se questa è ben parentesizza o `0` se questa non lo è. Con *stringa ben parentesizzata* si intenda una stringa che soddisfi queste due condizioni:

- in ogni suo prefisso il numero di parentesi tonde aperte è maggiore o uguale al numero di parentesi tonde chiuse;
- il numero totale di parentesi tonde aperte è uguale al numero totale di parentesi tonde chiuse.

La funzione deve essere invocabile da file sorgenti diversi da quello in cui è definita.

2.1 Info utili

- Il codice ASCII del carattere di fine stringa è `0`.
- Il codice ASCII del carattere `"("` è `40`.
- Il codice ASCII del carattere `")"` è `41`.

2.2 Esempio

Data la seguente stringa in input

```
S="(5+3)(4/2)(xy(_)z)"
```

la funzione restituisce `1`

Data la seguente stringa in input

```
S="((((((abc)"
```

la funzione restituisce `0`

3 Alice e Bob

nome del file sorgente: foo.asm

Alice, una studentessa alle prime esperienze con Assembly, svolge un esercizio per il corso di Architettura 2 implementando una funzione chiamata `foo` nel file sorgente `foo.asm`. La funzione `foo` riceve in input, nell'ordine:

- il base address di un array;
- il numero di elementi presenti in esso.

Quando invocata, la funzione restituisce un numero intero e, nello svolgere il proprio compito, invoca a sua volta un'altra funzione di nome `bar` che Alice ha implementato nel file `bar.asm`.

Bob, nel tentativo di fare colpo su Alice, implementa (senza conoscere il testo dell'esercizio da lei svolto) una versione equivalente di `foo` che è di gran lunga più semplice: ha un segmento testo più breve ed è una procedura foglia.

Come è fatta la funzione di Bob?

3.1 Funzione foo (versione di Alice)

```
    .text
    .globl foo
foo:
    li $v0 0
```

```
    beqz $a1 end

    # salvo su stack s0 e ra
    sub $sp $sp 8
    sw $s0 0($sp)
    sw $ra 4($sp)

    li $s0 0
    lw $t0 0($a0)
    beqz $t0 call_bar
    li $s0 1
call_bar:
    add $a0 $a0 4
    sub $a1 $a1 1
    jal bar
    add $v0 $s0 $v0

    # ripristino s0 e ra da stack
    lw $s0 0($sp)
    lw $ra 4($sp)
    add $sp $sp 8
end:
    jr $ra
```

3.2 Funzione bar (versione di Alice)

```
    .text
    .globl bar
bar:
    li $v0 0
    beqz $a1 end

    # salvo su stack s0 e ra
    sub $sp $sp 8
    sw $s0 0($sp)
    sw $ra 4($sp)

    li $s0 0
    lw $t0 0($a0)
    beqz $t0 call_foo
    li $s0 -1
call_foo:
    add $a0 $a0 4
    sub $a1 $a1 1
    jal foo
```

```
    add $v0 $s0 $v0

    # ripristino s0 e ra da stack
    lw $s0 0($sp)
    lw $ra 4($sp)
    add $sp $sp 8
end:
    jr $ra
```

4 Combina Array

nomi dei file sorgenti: combineInt.asm, combineArr.asm

Si implementino la funzione `combineInt` e la procedura `combineArr`, entrambe invocabili da sorgenti diversi da quello in cui sono definite.

La funzione `combineInt` riceve in input tre valori interi `a`, `b` e `k` (da passare in tale ordine) e restituisce un intero `c`:

- se `k` è uguale a `0` allora `c` è il massimo tra `a` e `b`;
- se `k` è uguale a `1` allora `c` è il minimo tra `a` e `b`;
- se `k` è uguale a `2` allora `c` è la differenza tra il massimo e il minimo (entrambi tra `a` e `b`);
- in ogni altro caso `c` è pari a `0`.

La procedura `combineArr` riceve in input tre array `A`, `B` e `C` (passaggio per indirizzo) e un intero `N` che rappresenta il numero di elementi in ciascun array. I parametri vanno passati nell'ordine in cui sono enunciati (`A`, `B`, `C` e `N`). La procedura sovrascrive l'array `C` in memoria sostituendo ogni emento di `C` in posizione `i`-esima con `combineInt(A[i],B[i],C[N-1-i])` *Attenzione! L'elemento di `C` passato alla funzione `combineInt` è sempre quello passato dal chiamante di `combineArr` (il valore presente **prima** della sovrascrittura).*

L'implementazione di `combineArr` **deve** basarsi su chiamate multiple alla funzione `combineInt` implementata precedentemente.

4.1 Esempio di `combineInt`

Data la seguente configurazione di input:

```
(a,b,k)=(2, 1, 1)
```

la funzione restituisce `c=1`.

Data la seguente configurazione di input:

```
(a,b,k)=(2, 1, 3)
```

la funzione restituisce `c=0`.

4.2 Esempio di `combineArr`

Data questa configurazione di input:

```
A = [10 -4 6 91]
B = [21 2 -1 15]
C = [2 0 1 7]
N=4
```

dopo la chiamata a procedura si avrà:

```
C = [0 -1 2 11]
```