Iniziato	giovedì, 18 luglio 2024, 11:28
Stato	Completato
Terminato	giovedì, 18 luglio 2024, 12:00
Tempo impiegato	32 min. 15 secondi
Valutazione	**4,0** su un massimo di 10,0 (**40**%)

Domanda **1**

Parzialmente corretta

Punteggio ottenuto 2,0 su 5,0

Scrivi una procedura chiamata `caesar_encrypt(char *str)` che cifra una stringa usando il Cifrario di Cesare. Il Cifrario di Cesare sostituisce ogni lettera della stringa con la lettera che si trova a 3 posizioni a sinistra nell'alfabeto. Ad esempio, 'd' diventa 'a', 'e' diventa 'b', ..., 'z' diventa 'w', 'a' diventa 'x', 'b' diventa 'y', 'c' diventa 'z'. I caratteri non alfabetici rimangono invariati.

Assumere che la stringa contenga solo lettere minuscole. Ad esempio, se `str` contiene `".abcdefghijklmnopqrstuvwxyz!"`, la funzione dovrà modificare `str` in `".xyzabcdefghijklmnopqrstuvw!"`.
Il seguente codice in C implementa `caesar_encrypt` (convertilo in RISC-V):

```
void caesar_encrypt(char *str) {
    int i = 0;
    while (str[i] != '\0') {
        if (str[i] >= 'a' && str[i] <= 'z') {
            str[i] = 'a' + (str[i] - 'a' + 23) % 26;
        }
        i++;
    }
}
```

Si ricorda che il codice ASCII del carattere 'a' é 97, e che la pseudo-istruzione `li x5, 'a'` carica il valore 97 in `x5`.
Si ricorda inoltre che l'operatore '%' in C restituisce il resto della divisione del primo operando per il secondo operando.

Attenzione:

- Incollare solo la funzione `caesar_encrypt` (in RISC-V) nel campo sottostante
- Attenzione alle convenzioni di chiamata!
- Usare il seguente codice `main` per lo sviluppo e il debugging nel simulatore RARS

```
    .globl _start
    .data
        str: .string ".abcdefghijklmnopqrstuvwxyz!"

    .text
    _start:
        # chiama caesar_encrypt
        la   a0, str
        jal  ra, caesar_encrypt

        # exit
        li   a7, 10
        ecall

    #*****************************************************
    # completa la funzione caesar_encrypt nel campo sottostante
```

Answer:

Reset answer

```
1  caesar_encrypt:
2      li t0, 97               # "a" ASCII
3      li t1, 121              # "z" ASCII
4      li t3, 26               # divisione per 26  (% 26)
5  loop:
6      lbu t2, 0(a0)           #load string
7      beqz t2, end            # while (str[i] != '\0')
8
9      ble t2, t0 , else       #    (str[i] >= 'a')
10     bge t2, t1 , else       # && (str[i] <= 'z')
11 else:
12     sub t4, t2, t0          #(str[i] - 'a')
```

```
13      addi t4, t4, 23         #(str[i] - 'a' + 23)
14      add t4, t4, t0          #'a' + (str[i] - 'a' + 23)
15      rem t2, t4, t3          #str[i] = 'a' + (str[i] - 'a' + 23) % 26;
16
17      addi a0, a0, 1          #i++
18      j loop
19  end:
20      mv a0, t2
21      ret
```

	Test	Expected	Got	
✖	.section .data str: .string ".abcdefghijklmnopqrstuvwxyz!"	str: .xyzabcdefghijklmnopqrstuvw!	str: .abcdefghijklmnopqrstuvwxyz!	✖

Some hidden test cases failed, too.
Your code must pass all tests to earn any marks. Try again.

Show differences

Question author's solution (Asm):

```
18      li   t3, 'z'
19      bgt  t2, t3, caesar_encrypt_next
20
21      # apply caesar encryption
22      li   t3, 'a'
23      li   t4, 26
24      sub  t2, t2, t3
25      addi t2, t2, 23
26      rem  t2, t2, t4
27      add  t2, t2, t3
28
29      # store the character back
30  caesar_encrypt_next:
31      sb   t2, 0(t1)
32      addi t0, t0, 1
33      j    caesar_encrypt_loop
34
35  caesar_encrypt_end:
36      ret
37
38
39
```

Parzialmente corretta

Punteggio di questo invio: 0,0/5,0.

Commento:
- Riga 3, la codifica ASCII di 'z' è 122
- Test di righe 9 e 10 errati, in questo modo si applica la trasformazione a qualunque carattere
- Calcolo del carattere a righe 14 e 15 errato, va prima fatta la rem e poi la add (regole di precedenza degli operatori)
- Non viene salvato in memoria il carattere calcolato (sb)

Domanda **2**

Parzialmente corretta

Punteggio ottenuto 2,0 su 5,0

Scrivi una procedura chiamata `ones_arr(arr1, arr2, pos)` che ritorna `countones(arr1[pos]) * countones(arr2[pos])`. Le variabili `arr1` e `arr2` sono array di half word.

La funzione `ones_arr` deve utilizzare la funzione `countones` che conta il numero di bit impostati a 1 in un intero a 64 bit. La funzione `countones` è già implementata in RISC-V ed è disponibile nel codice sottostante. Le soluzioni che non utilizzano la funzione `countones` saranno considerate errate.

Il valore di ritorno deve essere lasciato nel registro `a0`.

Il seguente codice in C implementa `ones_arr` (convertilo in RISC-V):

```
// int (in C) è equivalente a word (in RISC-V)
// short (in C) è equivalente a half (in RISC-V)
int ones_arr(short arr1[], short arr2[], int pos) {
    return countones(arr1[pos]) * countones(arr2[pos]);
}
```

Attenzione:

- Incollare solo la funzione `ones_arr` (in RISC-V) nel campo sottostante
- Attenzione alle convenzioni di chiamata!
- Usare il seguente codice `main` per lo sviluppo e il debugging nel simulatore RARS

```
.globl _start
.data
    arr1: .half -129, 127, 128, 127, -128, 127, 128, 127
    arr2: .half 127, 128, 127, -128, 127, 128, 127, -129
    pos:  .word 7

.text
_start:
    # chiama ones_arr
    la   a0, arr1
    la   a1, arr2
    la   a2, pos
    lw   a2, 0(a2)
    jal  ra, ones_arr

    # exit
    li   a7, 10
    ecall

#*****************************************************
# Procedure countones(n)
# a0 -> N
# a0 -> ritorna il conteggio dei bit impostati a 1
#*****************************************************
countones:
        li a1, 0
    ca: beqz a0, cb
        andi t0, a0, 1
        add  a1, a1, t0
        srli a0, a0, 1
        j ca
    cb: mv a0, a1
        ret

#*****************************************************
# completa la funzione ones_arr nel campo sottostante
```

Answer:

Reset answer

```
    sd s1, 24(sp)

    add s1, a2, zero  # pos


    mv a2, s2
    ld a0, 8(sp)
    jal ra, countones  #call function "countones"
    mv a4, a0          #transformare la risulta in a2

    mv a2, s2
    ld a1, 16(sp)
    jal ra, countones #call function "countones"
    mv a3, a0         #transformare la risulta in a3

    mul a3, a3, a4   #countones(arr1[pos]) * countones(arr2[pos])

    mv a0, a3
    ld ra, 0(sp)
    ld s1, 24(sp)
    addi sp, sp, 32
    ret
```

	Test	Expected	Got	

✖	`.section .data`
	`arr1: .half -129, 127, 128, 127, -128, 127, 128, 127`
	`arr2: .half 127, 128, 127, -128, 127, 128, 127, -129`
	`pos: .word 7`
		a0: 441	a0: 602377752918088	✖

Some hidden test cases failed, too.

Your code must pass all tests to earn any marks. Try again.

Show differences

Question author's solution (Asm):

```
#################################################################################
# Procedure ones_arr(arr1, arr2, pos)
# a0 -> address of array 1
# a1 -> address of array 2
# a2 -> pos
# return: countones(arr1[pos]) * countones(arr2[pos])
#################################################################################
ones_arr:
    addi sp, sp, -24
    sd   ra, 0(sp)
    sd   s1, 8(sp)
    sd   s2, 16(sp)

    slli a2, a2, 1
    add  s1, a2, a1

    add  a0, a0, a2
    lh   a0, 0(a0)
    jal  ra, countones
    mv   s2, a0

```

Parzialmente corretta

Punteggio di questo invio: 0,0/5,0.

Commento:
- A riga 11 e 16 copia il registro s2 (di cui non si conosce il contenuto) in a2: a che scopo?
- Non calcola gli indirizzi degli elementi dell'array da leggere