Iniziato	mercoledì, 19 giugno 2024, 11:22
Stato	Completato
Terminato	mercoledì, 19 giugno 2024, 12:10
Tempo impiegato	47 min. 6 secondi
Valutazione	**3,5** su un massimo di 10,0 (**35**%)

Domanda **1**

Parzialmente corretta

Punteggio ottenuto 1,5 su 5,0

Scrivere una procedura chiamata `anydiv7(array, size)` che restituisce 1 se almeno uno degli elementi presenti nell'array di word `array` è divisibile per 7, 0 altrimenti.

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

Assumere che `size` sia maggiore di 0.

Il seguente codice in C implementa `anydiv7` (convertilo in RISC-V):

```
// int (in C) equivale a word (in RISC-V)
int anydiv7(int array[], int size) {
    for (int i = 0; i < size; i++) {
        if (array[i] % 7 == 0) { // se divisibile per 7
            return 1;
        }
    }
    return 0;
}
```

Attenzione:

- Incollare solo la funzione `anydiv7` (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
        array: .word 8,5,3,7,2,6,4,1
        size:  .word 8

    .text
    _start:
        # chiama anydiv7
        la   a0, array
        la   a1, size
        lw   a1, 0(a1)
        jal  ra, anydiv7

        #exit
        li   a7, 10
        ecall

    #*****************************************************
    # completare la funzione anydiv7 nel campo di sotto
```

Answer:

Reset answer

```
1  addi sp, sp, -40
2      sd ra, 0(sp)
3      sd s1, 8(sp)
4      sd s2, 16(sp)
5      sd s3, 24(sp)
6      sd s4, 32(sp)
7
8      li s1, 0 #i
9      ld s2, 0(a1) #size
10     li s3, 7  #per dividere a 7
11     li t0, 1
12     add s4, zero, zero #salvare
13 for loop:
```

```
    ble s2, s1, end_save   #i < size
    ld a0, 0(a0)          #load aray
    div s4, a0, s3        # dividere array per 7
    beq s4,t0 , return_1  # se la risposta uguale 1
    li a0, 0
    addi a0, a0, 4
    addi s2, s2, -1

    j for_loop
```

	Test	Expected	Got	
✖	.section .data array: .word 8,5,3,7,2,6,4,1 size: .word 8	a0: 1	***Run error*** /usr/lib/gcc-cross/riscv64-linux-gnu/9/../../../../riscv64-linux-gnu/bin/ld: /tmp/ccPNo3FF.o: in function `.L0 ': (.text+0x394): undefined reference to `anydiv7' /usr/lib/gcc-cross/riscv64-linux-gnu/9/../../../../riscv64-linux-gnu/bin/ld: /tmp/ccPNo3FF.o: in function `for_loop': (.text+0x512): undefined reference to `return_1' collect2: error: ld returned 1 exit status ** Compilation failed. Testing aborted **	✖

Testing was aborted due to error.
Your code must pass all tests to earn any marks. Try again.

Show differences

Question author's solution (Asm):

```
################################################################################
# Procedure anydiv7(array, size)
# a0 -> address of array
# a1 -> size
# return 1 if there is at least one element divisible by 5, 0 otherwise
################################################################################
anydiv7:
    li   a2, 0     # i

anydiv7_loop:
    bge  a2, a1, anydiv7_return0

    # load array[i] in t0
    slli t0, a2, 2
    add  t0, a0, t0
    lw   t0, 0(t0)

    # check if divisible by 7
    li   t1, 7
    rem  t1, t0, t1
    beq  t1, zero, anydiv7_return1
    addi a2, a2, 1
```

Parzialmente corretta

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

Commento:

- manca l'etichetta anydiv7
- Poteva utilizzare registri t* invece di s* risparmiandosi tutti la procedura di salvataggio su stack.
- L'istruzione di riga 9 deve essere una lw, NON una ld.
- riga 9 errata, a1 contiene già il valore di size.
- a riga 16 deve usare l'istruzione rem, non div.
- manca l'etichetta e il codice per return_1

- a riga 15 perde l'indirizzo dell'array

Domanda **2**

Parzialmente corretta

Punteggio ottenuto 2,0 su 5,0

Scrivere una procedura chiamata `sumf(array, size)` che implementa in RISC-V il seguente codice:

```
// int (C) equivale a word (RISC-V)
int sumf(int array[], int size) {
    return foo(array[0]) + foo(array[size-1]);
}
```

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

La funzione `sumf` deve utilizzare la funzione `foo()`, già implementata in RISC-V. Soluzioni che non utilizzano la funzione `foo` verranno considerate invalide.

Attenzione:

- Incollare solo la funzione `sumf` (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
        array: .word 8,5,3,7,2,6,4,1
        size:  .word 8

    .text
    _start:
        # chiama sumf
        la   a0, array
        la   a1, size
        lw   a1, 0(a1)
        jal  ra, sumf

        # uscita
        li   a7, 10
        ecall

    #******************************************************
    # funzione foo
    #******************************************************
    foo:
        li    t0, 5
        div   a0, a0, t0
        slli  a0, a0, 1
        ret

    #******************************************************
    # completare la funzione sumf nel campo di sotto
```

Answer:

Reset answer

```
1  sumf:
2      addi sp, sp , -32
3      sd ra, 0(sp)
4      sd s1, 8(sp)
5      sd s2, 16(sp)
6      sd s3, 24(sp)
7
8      lw s1, 0(a1)
9      addi s1, s1, -1 #size[-1]
10     add s2, a0, zero #array
```

```
    add s3, a1, zero #size
    addi s3, s3, -1 #size-1
    jal ra, foo
    add a0, s2, zero
    add s2, s3, zero
    add a1, s2, zero
    add a0, a1, zero
    sw a0, 0(a0)
    jr ra
```

	Test	Expected	Got	

✖	`.section .data`
	`  array: .word 8,5,3,7,2,6,4,1`
	`  size:  .word 8`
		a0: 2	z 0000000000000000 ra 0000000000010488 sp 0000003ffffffb30 gp 0000003ffffffb50
			tp 0000003ffffffb50 t0 00012891fffdaedc t1 0000d3c6fffe5872 t2 000129f37ffdac19
			s0 0000003ffffffb50 s1 00012891fffdaedc a0 0000000000011864 a1 0000000000000008
			a2 0000d3c6fffe5872 a3 000129f37ffdac19 a4 000111edfffddc24 a5 0001289cfffdaec6
			a6 00014967fffd6d30 a7 000129167ffdadd3 s2 0000d3c6fffe5872 s3 000129f37ffdac19
			s4 000111edfffddc24 s5 0001289cfffdaec6 s6 00014967fffd6d30 s7 000129167ffdadd3
			s8 0001733b7ffd1989 s9 0000d0207ffe5fbf sA 0000f49f7ffe16c1 sB 0000003ffffffb50
			t3 000111edfffddc24 t4 0001289cfffdaec6 t5 00014967fffd6d30 t6 000129167ffdadd3
			pc 0000000000010704 va/inst 0000000000000008 sr 8000000200006020
			User load segfault @ 0x0000000000000008
				✖

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):

```
################################################################################
# sumf(array, size)
# a0 -> address of array
# a1 -> size
# return the sum of foo(array[0]) and foo(array[size-1])
################################################################################
sumf:
    addi sp, sp, -32
    sd   ra, 0(sp)
    sd   s1, 8(sp)
    sd   s2, 16(sp)
    sd   s3, 24(sp)

    mv   s1, a0
    addi s2, a1, -1

    lw   a0, 0(s1)
    jal  ra, foo
    mv   s3, a0

    slli s2, s2, 2
```

Parzialmente corretta

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

Commento:
- A riga 8 accesso in memoria all'indirizzo size!!! a1 contiene già il valore di size!
- Fa un solo richiamo alla funzione foo.