Esercitazione: Pari o Dispari?

Davide Bertozzi

ESERCIZIO

SI TRADUCA IN ASSEMBLER IL SEGUENTE CODICE C (vedi slide successiva), CHE:

- LEGGE UN INTERO DA STD INPUT
- VALUTA SE L'INTERO E' PARI O DISPARI
- STAMPA IL RISULTATO DELLA VALUTAZIONE A VIDEO

CODICE C

```
#include <stdio.h>
int paridispari(int a)
       if (a%2==0) return 0;
                                     else
                                               return 1;
int main()
int a, b;
printf("inserisci un numero: ");
scanf("%d", &a);
b=paridispari(a);
if (b==0) printf("il numero e' pari");
else printf("il numero e' dispari");
return 0;
```

Modalità di Soluzione

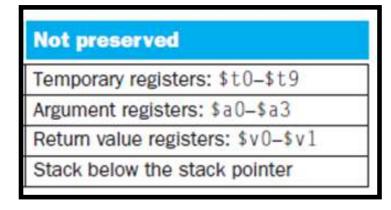
- Esistono due modalità di soluzione:
 - Traduzione letterale dal C
 - Divisione del numero intero per due
 - Osservazione del resto
 - Analisi del codice binario del numero intero
 - Se il bit meno significativo vale 0, il numero è positivo
 - Se il bit meno significativo vale 1, il numero è negativo

Analisi della Chiamata a Funzione

```
int main()
{
int a, b;
printf("inserisci un numero: ");
scanf("%d", &a);
b=paridispari(a);
if (b==0) printf("il numero e' pari");
else printf("il numero e' dispari");
return 0;
}
```

Domanda I -

Il Main scrive dei registri non-preservati prima della «jal» che legge dopo la «jal» stessa?



No: dopo la «jal» il Main fa solo postprocessing del valore di ritorno!

Analisi della Chiamata a Funzione

```
#include <stdio.h>
int paridispari(int a)
{
    if (a%2==0) return 0;
    else return 1;
}
```

Domanda II – C'è bisogno di gestire il frame pointer? Solo se:

- si accede a parametri passati attraverso lo stack
- Si accede a variabili locali alla procedura In questo caso, entrambe le condizioni sono false, dunque non c'è bisogno di gestire il frame pointer

Analisi della Chiamata a Funzione

```
#include <stdio.h>
int paridispari(int a)
{
    if (a%2==0) return 0;
    else return 1;
}
```

Domanda III -

C'è bisogno di allocare un frame sullo stack? Solo se vengono utilizzati i registri preservati:

Preserved
Saved registers: \$s0-\$s7
Stack pointer register: \$sp
Return address register: \$ra
Stack above the stack pointer

In questo caso:

- L'argomento «a» viene letto da \$a0
- Il valore di ritorno viene scritto direttamente su \$v0
- Non ci sono chiamate a subroutine
 Dunque NON SERVE ALLOCARE UN FRAME!

Divisione

Utilizzate l'istruzione:

div \$t0, \$t1 # divide \$t0 per \$t1

- La divisione produce automaticamente quoziente e resto nei registri speciali HI e LO.
- LO contiene il quoziente
- HI contiene il resto della divisione
- E' poi possibile usare quoziente e resto solo «travasandoli» sui registri standard con istruzioni apposite:
- mfhi \$t0 sposta il contenuto di HI in \$t0
- mflo \$t0 sposta il contenuto di LO in \$t0

Soluzione 1: Divisione per Due

```
.data
# a: .space 4 NON SERVE
# b: .space 4 NON SERVE
strinput: .asciiz "inserisci un numero: "
outputstringpari: .asciiz "il numero è pari."
outputstringdispari: .asciiz "il numero è dispari."
.text
Main:
              li $v0, 4
              la $a0, strinput
              syscall
              li $v0, 5
              syscall
              move $a0, $v0
              jal paridispari
              beq $v0, $zero, parimain
              la $a0, outputstringdispari
              li $v0, 4
              syscall
              j exitmain
```

```
parimain:
              la $a0, outputstringpari
              li $v0, 4
              syscall
exitmain:
              li $v0, 10
              syscall
paridispari:
              li $t0, 2
               div $a0, $t0
               mfhi $t0
               beg $t0, $zero, pari
              li $v0, 1
              j exit
pari:
              li $v0, 0
exit:
```

jr \$ra

Soluzione 2: Analisi del Bit Meno Significativo

Si riporta solo il codice della funzione

```
paridispari:

#li $t0, 2

#div $a0, $t0

#mfhi $t0

andi $t0, $a0, 0x0001

beq $t0, $zero, pari

li $v0, 1

j exit

pari:
```

li \$v0, 0

jr \$ra

exit:

0000...0001000

and-bit-per-bit

0000...0000001

0000...0000000

Numero «0» = numero pari!

Numero «8»

Soluzione 2: Analisi del Bit Meno Significativo

Si riporta solo il codice della funzione

```
.....
```

```
paridispari:
         #li $t0, 2
         #div $a0, $t0
         #mfhi $t0
         andi $t0, $a0, 0x0001
         beq $t0, $zero, pari
         li $v0, 1
         j exit
pari:
         li $v0, 0
exit:
         jr $ra
```

```
Numero «9»
```

 $0\,0\,0\,0\dots0\,0\,0\,1\,0\,0\,1$

and-bit-per-bit

 $0\,0\,0\,0\dots0\,0\,0\,0\,0\,0\,1$

 $0\,0\,0\,0\dots0\,0\,0\,0\,0\,0\,1$

Numero «1» = numero dispari!

Compito

 Ripetere lo stesso esercizio, ma verificando se l'intero è positivo oppure negativo