Laboratorio di Architettura degli Elaboratori II – a.a. 2023-24
NOTE ed esercizi: Ricorsione

ESERCIZIO 1: Implementare una funzione per il calcolo dell' n-esimo numero di Fibonacci dato il seguente codice C :

```
int fib (int n){
     if (n <= 1)
     return n;
else
     return fib (n – 1) + fib (n – 2);
}
```

Notare che il codice contiene **due** chiamate ricorsive. Prestare attenzione e assicurarsi di salvare il risultato della prima chiamata a fib prima di effettuare la seconda chiamata.

i) Assegnare I nomi dei riegistri alle variabili e determinare quali sono il caso base e la ricorsione (nella struttura di controllo di flusso da utilizzare, che sara' un *if*):

> . Un solo valore di input (int n) viene passato nell' *argument register* **$a0** . Il caso base sara' nella clausola *then* mentre il caso di ricorsione sara' nella clausola *else*.

ii) Convertire il codice per il caso base:

```
fib:
     bgt $a0, 1, recurse
     move $v0, $a0
     jr $ra
```

iii) Salvare I registri callee- e caller-saved sullo stack:

```
recurse:
     sub $sp, $sp, 12     # We need to store 3 registers to stack
     sw $ra, 0($sp)       # $ra is the first register
     sw $a0, 4($sp)       # $a0 is the second register, we cannot assume
                          # $a registers will not be overwritten by callee
```

iv) Chiamata ricorsive a fib :

```
     addi $a0, $a0, -1    # N-1
     jal fib
     sw $v0, 8($sp)       # store $v0, the third register to be stored on
                          # the stack so it doesn't get overwritten by callee
```

v) Chiamare ancora fib :

```
     lw $a0, 4($sp)       # retrieve original value of N
     addi $a0, $a0, -2    # N-2
     jal fib
```

E’ facile cadere nella tentazione di calcolare N-2 semplicemente sottraendo 1 dal valore presente in $a0 invece di ripristinare il valore originario di N e sottrarre ad esso 2. nonstante il fatto che questo sia tecnicamente corretto (posto che venga ripristinato il il valore originale di $a0 prima di ritornare dalla procedura) questo approccio e’ prono ad errori ed e’ una cattiva pratica di programmazione in MIPS assembly.

Le convenzioni MIPS impongono di **non fare assunzioni** riguardo a cio’ che viene restituito in ogni registro a parte **$s0-7, $sp e $ra** I cui valori verranno preservati attraverso la chiamata.

vi) Pulire lo stack e restituire il risultato calcolato :

```
     lw $t0, 8($sp)       # retrieve first function result
     add $v0, $v0, $t0
     lw $ra, 0($sp)       # retrieve return address
     addi $sp, $sp, 12
     jr $ra
```

ESERCIZIO 2: (Da svolgere tenendo presente che nel seguente codice MIPS assembly il valore in registro $a0 e’ un input ed il valore in registro $v0 e’ l’output)

i) Tradurre il seguente codice MIPS assembly in un equivalente codice scritto in un linguaggio ad alto livello (Go, C, C++, …):

```
func:
     addi $t0, $zero, 1       # i = 1
     addi $v0, $zero, 1       # v = 1
Loop:
     sle $t1, $t0, $a0        # set $t1 to 1 if (i <= arg)
     beq $t1, $zero, Exit     # exit loop if (i > arg)
     mul $v0, $v0, $t0        # v *= i
     addi $t0, $t0, 1         # i++
     j Loop                   # loop
Exit:
     jr $ra
```

ii) Quale funzione (matematica) e’ implementata dal codice sopra riportato?