Esercizio di Ricorsione

Implementare la funzione ricorsiva:

```
F(A,B)  = F(A-1,B)+A  A>1,B<=1
        = F(A,B-1)+B  A>1,B>1
        = 1           A<=1
```

Soluzione:

Una funzione ricorsiva deve essere implementata secondo questo schema (le ottimizzazioni si possono fare DOPO che si è ottenuta una procedura con questo schema FUNZIONANTE e CORRETTA):

1. Prologo (salvataggio dei registri da ripristinare dopo le chiamate ricorsive)
2. Check caso Base|caso passo
3. Caso base
4. Caso Passo
5. Epilogo (ripristino dei registri da preservare)

Inoltre:

- Un solo punto di ingresso alla procedura ricorsiva (la prima linea)
- un solo punto di uscita (l'ultima linea);
- appena entrati nella procedura si salvano tutti i registri necessari (prologo);
- un attimo prima di uscire si ripristinano i registri da preservare (epilogo).

Definiamo il prototipo della procedura (interfaccia di chiamata):

Input: A → $a0 , B → $a1
Output: $v0 ← F(A,B)

Scriviamo uno pseudo-codice C della procedura orientato alla traduzione in Assembly (uso costrutti IF ad una sola condizione intera, seguo le regole indicate sopra):

```
function F(AB)  {
      if (A>1)
            if(B>1)
                  R=F(A,B-1)+A
            else
                  R=F(A-1)+B
      else
            R=1
      return R
}
```

Il codice Assembly corrispondente è il seguente:

```
1     pF:
2                 #$a0 <- A
3                 #$a1 <- B
4                 #$v0 -> F(A,B)
5                 #F(A,B)    =    F(A-1,B)+A     (A>1,B<=1)
6                 #          =    F(A,B-1)+B     (A>1,B>1)
7                 #          =    1              (A<=1)
8
9                                      #prologo
10                addi $sp,$sp,-12
11                sw $ra,0($sp)
12                sw $a0,4($sp)
13                sw $a1,8($sp)
14
15                                     #check base/passo
16                bgt $a0,1,jABig
17                                     #base
18                li $v0,1
19                j jEnd
20    jABig:                           #check passo1/passo2
21                bgt $a1,1,jAeBBig
22                                     #passo 1
23                addi $a0,$a0,-1      #v0=F(A-1,B)
24                jal pF
25                lw $a0,4($sp)
26                add $v0,$v0,$a0      #v0=F(A-1,B)+A
27                j jEnd
28    jAeBBig:                   #passo 2
29                addi $a1,$a1,-1      #v0=F(A,B-1)
30                jal pF
31                lw $a1,8($sp)
32                add $v0,$v0,$a1      #v0=F(A,B-1)+B
33                #j jEnd
34    jEnd:
35                                     #epilogo
36                lw $ra,0($sp)
37                lw $a0,4($sp)
38                lw $a1,8($sp)
39                addi $sp,$sp,12
40
41                jr $ra
```

Note:
Nella funzione implementata sono presenti due casi passo. Nel primo caso occorre ricordare solo A dopo la chiamata ricorsiva in modo da realizzare la somma di A al risultato finale mentre nel secondo caso occorre ricordare solo B. E' possibile realizzare un codice più efficiente che sfrutti questa caratteristica per memorizzare un solo dato nello stack. Questa ottimizzazione però richiede un'implementazione con uno schema differente da quello presentato, di difficile comprensione immediata e difficile verifica di correttezza. In generale, considerando l'obbiettivo del corso è consigliabile (e richiesto) usare lo schema proposto per realizzare le funzioni ricorsive ed eventualmente proporre in un codice separato ogni ottimizzazione suggerita.

Una funzione ricorsiva richiama se stessa durante il caso passo. Nell'implementazione della funzione occorre implementare ogni chiamata secondo le convenzioni.: dopo la chiamata ricorsiva occorre considerare tutti i registri temporanei (a0-a4, v0-v1,t0-t9) come corrotti. Nel caso occorra utilizzare uno di questi registri dopo la chiamata (recupero di un parametro in ingresso memorizzato in a0-a4, conservazione di un dato temporaneo per un calcolo successivo), è necessario (richiesto per i fini del corso) che il valore originario venga ripristinato usando il dato salvato nello stack durante il prologo, oppure, per la memorizzazione temporanea di valori durante le chiamate ricorsive, utilizzare uno dei registri garantiti (s0-s9).