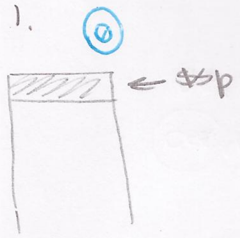
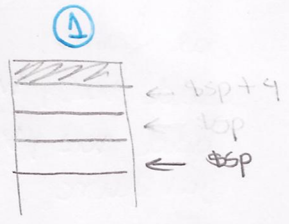
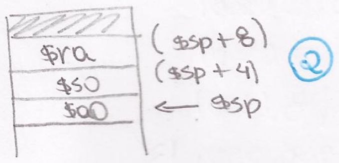
2.31, 2.34 ~~2.39, 2.40, 2.41, 2.42 / 2.43~~
↓ AEX5 → AEX6 → AEX8

AEX5

2.31)
```
int fib(int n) {        → $a0
    if (n==0) {
        return 0;
    } else if (n==1) {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
```







```
Fib:  addi  $sp, $sp, -12   ①
      sw    $ra, 8($sp)   ✪
      sw    $s0, 4($sp)
      sw    $a0, 0($sp)
      ---------------------   ②
      beq $a0, $0, L0
      addi $t0, $0, 1
      beq $a0, $t0, L1
      ---------------------
        addi $a0, $a0, -1
        jal fib  → chamada
        add $s0, $0, $v0  # fib(n-1) ✪
      ---------------------
        addi $a0, $a0, -1  ✪
        jal fib  → chamada
        add $v0, $s0, $v0  # fib(n-1) + fib(n-2)
        j DONE

L0:  add $v0, $0, $0
     j DONE
L1:  addi $v0, $0, 1
DONE: lw  $a0, 0($sp)
      lw  $s0, 4($sp)
      lw  $ra, 8($sp)
      addi $sp, $sp, 12
      jr $ra
```

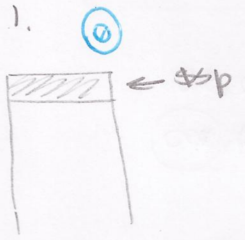
~~obs: como ocorre a chamada~~

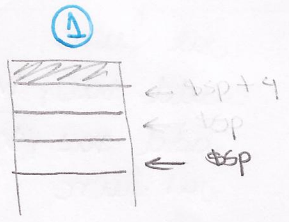
2.31, 2.34 → ~~2.39, 2.40, 2.41, 2.42, 2.43~~
↓ AEX5 ↳ AEX6 ↳ AEX8

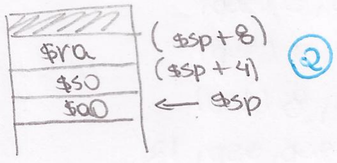
AEX5

```
                         $a0
2.31) int fib(int n) {
          if (n==0) {
              return 0;
          } else if (n==1) {
              return 1;
          } else {
              return fib(n-1) + fib(n-2);
          }

Fib: addi $sp, $sp, -12   ①
     sw  $ra, 8($sp)  ✪
     sw  $s0, 4($sp)
     sw  $a0, 0($sp)
     --------------------   ②
     beq $a0, $0, L0
     addi $t0, $0, 1
     beq $a0, $t0, L1
     --------------------
     addi $a0, $a0, -1
     jal fib → chamada
     add $s0, $0, $v0 # fib(n-1) ✪
     --------------------
     addi $a0, $a0-1 ✪
```





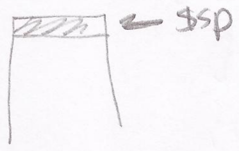


~~obs: como existe a chamada,~~

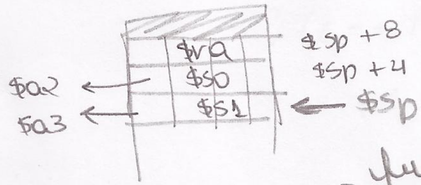
Caso n>1, ocorrerão duas chamadas da função fib,
ou seja, tanto o resultado de fib(n-1), o $ra e $a0
(✪) não são garantidos (os valores podem ser alterados
por outro programa ou até o tratador de exceções → ou outra rotina pode
precisar desses registradores), logo, é função da
chamadora guardar o fib(n-1) e o $a0 porque
são utilizados após uma chamada de função, enquanto
que o $ra deve ser guardado pela função que foi
chamada (ele será alterado quando outra rotina é
chamada).

2.34)
```
        $a0       $a1       $a2       $a3
int f(int a, int b, int c, int d) {
      return ( func(a,b), c+d );
}
```

func := int f(int a, int b),



```
F:  addi $sp, $sp, -12
    sw   $ra, 8($sp)
    sw   $s0, 4($sp)
    sw   $s1, 0($sp)
    add  $s0, $0, $a2
    add  $s1, $0, $a3
    ------------------
    jal func
    add $a0, $v0, $0
    add $a1, $s1, $s0
    jal func
    ------------------
    lw $s1, 0($sp)
    lw $s0, 4($sp)
    lw $ra, 8($sp)
    addi $sp, $sp, 12
    jr $ra.
```



função da chamadora

obs: Eu não sei o que a rotina func faz com c e d, logo preciso guardar eles. ↳ $s0 e $s1 ★ garantidos pela roti

Eu não vou usar a e b depois de chamar func(a,b) logo não preciso guardar.

Como tem chamada de rotinas preciso guardar o $ra.
↳ a

★ garantidos pela convenção de chamada (rotina chamada)
↳ a função func tem que me garantir isso mesmo não conhecendo ela.