CS2106 Tutorial 1

AY 25/26 Sem 1 — github/omgeta

Q1.
```
main: la $t0, a
      lw $a0, 0($t0)  ; read value of a
      la $t1, b
      lw $a1, 0($t1)  ; read value of b

      jal f           ; call f(a, b)

      la $t2, y
      sw $v0, 0($t2)  ; store result in y

      li $v0, 10
      syscall         ; exit to OS

f:    add  $t0, $a0, $a1  ; x + y
      sll  $v0, $t0, 1    ; 2(x + y)
      jr $ra
```

Q2.
```
main: la    $t0, a
      lw    $t0, 0($t0)  ; read value of a
      sw    $t0, 0($sp)  ; push onto stack
      addi  $sp, $sp, 4

      la    $t1, b
      lw    $t1, 0($t1)  ; read value of b
      sw    $t1, 0($sp)  ; push onto stack
      addi  $sp, $sp, 4

      jal   f            ; call f(a, b)

      addi  $sp, $sp, -4 ; pop from stack
      lw    $t0. 0($sp)
      la    $t2, y
      sw    $v0, 0($t2)  ; store result in y

      addi  $sp, $sp, -8 ; discard a, b

      li    $v0, 10
      syscall            ; exit to OS

f:    mov   $fp, $sp     ; frame pointer at top-of-stack

      lw    $t0, -8($fp)
      lw    $t1, -8($fp)

      add   $t2, $t0, $t1 ; x + y
      sll   $t2, $t2, 1   ; 2(x + y)

      sw    $t2, 0($sp)   ; push ret to top
      addi  $sp, $sp, 4

      jr $ra
```

Q3. No; in nested or recursive cases, we lose information of where we are in the stack, where we need to return, and we might also run out of registers.

Q4.
```
main: sw    $fp, 0($sp)   ; 1. push $fp, $sp to stack
      sw    $sp, 4($sp)
      mov   $fp, $sp      ; 2. copy $sp to $fp

      addi  $sp, $sp, 20  ; 3. reserve space for 2 params and $ra

      la    $t0, a        ; 4. read params into stack
      lw    $t0, 0($t0)
      sw    $t0, 8($fp)
      la    $t1, b
      lw    $t1, 0($t1)
      sw    $t1, 12($fp)

      jal   f             ; 5. call f(a, b)

      la    $t0, y        ; 1. get result
      lw    $t1, 8($fp)
      sw    $t1, 0($t0)

      lw    $sp, 4($fp)   ; 2. restore $sp, $fp
      lw    $fp, 0($fp)

      li $v0, 10
      syscall             ; exit to OS

f:    sw    $ra, 16($fp)  ; 1. save $ra to stack

      addi  $sp, $sp, 8   ; 2. reserve 2 bytes for registers
      sw    $t0, 20($fp)  ; 3. save registers before use
      sw    $t1, 24($fp)

      lw    $t0, 8($fp)   ; 4. load parameters
      lw    $t1, 12($fp)

      add   $t1, $t0, $t1 ; 5. compute result
      sll   $t1, $t1, 1

      sw    $t1, 8($fp)   ; 6. store result

      lw    $t0, 20($fp)  ; 7. restore registers
      lw    $t1, 24($fp)
      addi  $sp, $sp, -8

      lw    $ra, 16($fp)  ; 8. restore $ra

      jr    $ra           ; 9. return to caller
```

Q5. If not, after returning from a subcall, the values at expected registers could be different. However, main does not need to save registers as it is always a caller, and never a callee so it doesn't need to restore any state for a calling function.

Q6. We need to return to the caller of the current stack frame, instead of accidentally returning to the function calling any of the subcalls (e.g. if callee makes a subcall, it might return to itself if not for the saved $ra).

Q7.

Item	**Where it is stored/created**
a	Stack
*a	Heap
b	Stack
c	Data
x	Stack
y	Stack
z	Stack
fun1's return result	Stack
main's code	Text
Code for f	Text