6,6

Avaliação de LM

3) Ele usa o endereço de retorno colocado na pilha como parte do execução da instrução CALL.
É fundamental que o topo da pilha esteja apontando para o endereço de retorno na execução da instrução ret

1,0

5) a) LEA significa carregar efetivo
MOV significa carregar valor.
→ Lea carrega um ponteiro para o item que você está endereçando, enquanto o MOV carrega o valor real nesse endereço.

5) b) é um ponteiro de base, que aponta para a base do quadro de pilha atual, e % rsp é o ponteiro de pilha, que aponta para o topo do quadro de pilha atual.
% rbp sempre tem um valor mais alto do que % rsp por que a pilha começa em um endereço de memória alto, e cresce para baixo.

c) Enter → Cria uma stack frame.

```
; enter
push ebp
mov ebp, esp
```

Leave → destrói uma stack frame

```
; leave
mov esp, ebp
pop ebp
```

+0,2

Faltou explicar os parâmetros

5) d) O conceito de "variáveis locais", como estamos habituados em linguagens de programação de alto nível, não tem equivalência direta em assembly. As duas abordagens mais comuns são usar os registradores de processador ou a stack para armazenar dados. Os compiladores são muito eficientes em estabelecer a abordagem ideal para cada cenário.

→ Digamos que queremos reservar 8 bytes na pilha para as variáveis locais da função:

```
push rbp
mov rbp, rsp
sub rsp, 8; reserva 8 bytes
```

- tal sequência pode ser substituído por:

```
enter 8, 0
```

- para retornar ao IP do stack frame:

```
mov rsp, rbp
pop rbp
```

- também pode ser substituído por:

```
leave
```

5) e) Supondo que foram empilhados 4 valores e que cada valor ocupa 4 bytes:

Temos 2 modos de retirar esses argumentos da pilha:

Após o retorno desempilhar esses argumentos com quatro pops e o outro é utilizar RET 16, isso ocorria o SP a não mais considerar os argumentos como empilhados.

Questão 1

```
extern printf
section.data
        max equ 10
        arqNome db "texto10.txt",0
section .bss
        buff resb 64
section .text
        global _start
_start:
        ;lendo entrada
        mov rax, 0
        mov rdi, 0
        mov rsi, buff
        mov rdx, 64

        ;criando arquivo para escrita e append
        mov rax, 2
        mov rdi, 64 + 1024 + 1
        mov rsi, arqNome
        mov rdx,mov 0664o
        syscall

        push rax

        mov rcx, [max]
        mov rax, 0
        push rax
_loop:
        mov rdi, rax
        mov rax, 1
        mov rsi, buff
        mov rdx, 64
        syscall

        pop rax
        inc rax
        cmp rax, rcx
        push rax
        jl _loop

        pop rax

        ;fechando o arquivo
        mov rax, 3
        pop rdi
        syscall
```

funciona ao compilar?

C 1,0

Questão 4

```
%macro _printf 2
  mov rax,1             ; escrita
  mov rdi,1             ; fd=1, na tela
  mov rsi,%1             ;buffer a ser gravado
  mov rdx,%2             ;tamanho a ser gravado
  syscall
%endmacro

section .data
        nomeArq db "arquivo.txt",0
        insiraTexto "Insera um texto",0
section .bss
        buff resb 64
section .text
        global _start

_start:
        ;criando arquivo para escrita
        mov rax, 2
        mov rdi, 64 + 1
        mov rsi, nomeArq
        mov rdx,mov 0664o
        syscall


        _printf insiraTexto 16
        ;lendo texto
        mov rax, 0
        mov rdi, 0
        mov rsi, buff
        mov rdx, 64

        ;escrevendo no arquibo
        push rax
        mov rdi, rax
        mov rax, 1
        mov rsi, buff
        mov rdx, 64
        syscall

        ;fechando o arquivo
        mov rax, 3
        pop rdi
```

```
syscall


_printf insiraTexto 16
;lendo texto
mov rax, 0
mov rdi, 0
mov rsi, buff
mov rdx, 64

;criando arquivo para escrita append
mov rax, 2
mov rdi, 1024 + 1
mov rsi, [nomeArq]
mov rdx,mov 0664o
syscall

;escrevendo no arquibo
push rax
mov rdi, rax
mov rax, 1
mov rsi, buff
mov rdx, 64
syscall

;fechando o arquivo
mov rax, 3
pop rdi
syscall
```