

## Lista 2 – Felipe Melo – Thalles Nonato

DRE Felipe: 119093752

DRE Thalles: 119058809

**Questão 1) a) A linha 7 salva o valor de `%ecx` na pilha e a linha 14 restaura esse valor. Por que é necessário salvar esse registrador?**

O compilador realiza um `push` em `%ecx` antes de chamar a função `printf`. Isso ocorre pois `%ecx` é um registrador volátil e, por isso, deve assegurar que seu conteúdo seja restaurado após a chamada da função retornar.

**Questão 1) b) Desenhe o estado da pilha antes da linha 13 e justifique a existência da linha 8:**

L2: carrega o conteúdo de  $(\texttt{\%esp} + 4)$  em `%ecx`

L3: faz o topo da pilha se mover para o próximo endereço múltiplo de 16 após zerar os 4 bits menores do topo

L4: empilha o conteúdo de  $(\texttt{\%ecx} - 4)$

L5: empilha o conteúdo de `%ebp`

L6: copia o conteúdo de `%esp` para o registrador `%ebp`

L7: coloca `%ecx` no topo da pilha

**L8:** abre espaço de 1 byte na pilha. Isso é necessário porque, dessa forma, o endereço de memória é alinhado, fazendo com que seja um múltiplo de 16

L9: empilha a parte alta do valor

L10: empilha a parte baixa do valor

L11: põe a string que será impressa na pilha

L12: empilha o valor 1

Endereço	Conteúdo da Pilha
%ecx	MEM [%esp + 4]
⋮	
⋮	
%ebp + 8 (n x 16)	
%ebp + 4	Mem [%ecx - 4]
%ebp	Mem [antigo ebp] = RIP SO
%esp + 20	Mem [%ecx]
%esp + 16	
%esp + 12	0x400147AE
%esp + 8	0x40000000
%esp + 4	string
%esp	1

**Questão 1) c) Você acha que a linha 15 poderia ser suprimida ou ela é essencial? Justifique:**

Ela libera espaço na pilha antes do fim da execução do programa e, portanto, é necessária.



## Questão 1) d) O que faz a linha 16? Qual a justificativa para ela?

Zera o valor armazenado em `%eax`, uma vez que XOR de uma variável com ela mesma resulta em zero. A razão para isso é que `%eax` é o registrador padrão de retorno e, dessa forma, essa linha equivale ao `return 0` no código em C



## Questão 1) e) Comente as linhas 17, 18 e 19, mostrando o que cada uma delas causa e, em especial, a existência da linha 18 e sua finalidade explícita:

Leave prepara a pilha para o retorno, copiando o registrador `%ebp` para `%esp` e em seguida restaurando o conteúdo anterior de `%ebp` na pilha. A linha 18 é encarregada de mover para o topo da pilha o endereço de retorno da função. A última linha transfere o controle para o endereço de retorno da função `main` localizado na pilha.



**Questão 2) a) Desenhe a pilha após serem executadas as linhas 1 a 8. Marque os endereços alinhados e a base de main, seguindo o modelo apresentado em ex12-editado.pdf, Semana 4 no classroom:**

Endereço	Conteúdo da Pilha	Comentário
OFP	qualquer	
:		
%esp + 32 (n x 16)		
%esp + 28	Mem [%ecx - 4] = RIP SO	endereço para retorno ao SO
%ebp	Mem [antigo ebp]	contém o %ebp anterior
%esp + 20	Mem [%ecx]	armazena o argc
%esp + 16		
%esp + 12		
%esp + 8		
%esp + 4		
%esp		

**Questão 2) b) Comente cada uma das linhas de 9 a 13, indicando o que elas causam. Após executar a linha 13, indique o conteúdo de ST(0) e ST(1) na pilha x87:**

L9: carrega o conteúdo de .LC0 (5.26 em precisão simples) para o topo da pilha FPU

L10: grava o conteúdo do topo da pilha no endereço de (%ebp - 24) e retira da pilha FPU

L11: carrega o conteúdo de .LC1 (3.1) para o topo da pilha FPU

L12: grava a parte baixa conteúdo do topo no endereço de (%ebp - 16) e a parte baixa em (%ebp - 12) e retira da pilha


L13: carrega 5.26 (conteúdo do endereço (%ebp - 24)) para o topo da pilha

O conteúdo de ST(0) é 0x40A851EC

Não há conteúdo de ST(1)

**Questão 2) c) Comente cada uma das linhas de 14 a 16, indicando o que elas causam. Após executar a linha 16, indique o conteúdo de ST(0) e ST(1) na pilha x87:**

L14: subtrai 3.1 (conteúdo do endereço (%ebp - 16)) do conteúdo do topo da pilha FPU

L15: grava  $(5.26 - 3.1)$ , que é o conteúdo do topo da pilha, no endereço (%ebp - 20) e retira da pilha FPU 

L16: carrega  $(5.26 - 3.1)$ , que é o conteúdo do (%ebp - 20) para a pilha FPU

O conteúdo de ST(0) é o resultado de  $(2.16_{10} = 0x400A3D72)$  

Também não há conteúdo de ST(1)

**Questão 2) d) Indique a finalidade e a necessidade das linhas 17 e 18:**

L17: subtrai 4 do endereço do stackpointer (%esp)

L18: carrega o conteúdo de (%esp - 8) para o endereço de %esp

A necessidade é abrir mais espaço na pilha para alocar. 

**Questão 2) e) Mostre o conteúdo completo da pilha antes da execução da linha 21. Indique os endereços alinhados bem como os conteúdos respectivos. Use o campo comentários para relacionar a alteração da pilha com a linha do código:**

Endereço	Conteúdo da Pilha	Comentário
OFP	qualquer	
:		
%esp + 32 (n x 16)		
%esp + 28	Mem [%ecx - 4] = RIP SO	endereço para retorno ao SO
%ebp	Mem [antigo ebp]	contém o %ebp anterior
%esp + 20	Mem [%ecx]	armazena o argc
%esp + 16 (n x 16)		
%esp + 12	0x4008CCCC	parte alta
%esp + 8	0xCCCCCCCCD	parte baixa
%esp + 4	0x400A3D72	resultado da operação
%esp = %ebp - 24 (n x 16)	0x40A851EC	



**Questão 2) f) Justifique as diretivas nas linhas 28 e 30:**

Essas diretivas fazem o alinhamento das próximas instruções ou dados imediatamente após elas. Algumas instruções são executadas mais rapidamente se alinhadas em um limite determinado de bytes, ou seja, essas diretivas servem para otimizar o código.

### Questão 3)

- [illegible]

### Questão 4)

- ```

1.  endbr32
2.  movl  4(%esp), %ecx           // apanha o parâmetro x
3.  movl  $-1840700269, %edx      //transformando 0x92492493 para signed temos 0x6DB6DB6D
    // 0x6DB6DB6D = 011011011011011011011011011011012. Rearranjando esse valor:
    // [011][011][011][011][011]2 ... = (3÷8) + (3÷82) + (3÷83) + ... ou seja, uma progressão
    // geométrica de razão (1÷8). Utilizando a fórmula  $S_n = (a_1) \div (1 - q)$ , obtemos 3÷7.
    // Dessa forma, o número oculto P é igual a 7. Então  $n = (3 \div 7) * 2^{32}$ 
4.  movl  %ecx, %eax             // copia x de %ecx para %eax
5.  imull  %edx                  // multiplica %eax com %edx, ou seja,  $((3 \div 7) * 2^{32} * x)$ 
6.  leal   (%edx, %ecx), %eax     // computa  $x * (1 - 3 \div 7)$ , obtendo  $(4 \div 7) * x$ 
7.  sarl   $31, %ecx             // divide o conteúdo de %eax por 4, resultando em  $(1 \div 7) * x$ 
8.  sarl   $2, %eax              // compara se  $x < 0$ . Caso sim, chega a -1. Caso contrário,
    // obtém 0
9.  subl   %ecx, %eax            // compara se  $x < 0$ . Se sim, soma 1 para a divisão negativa
10. ret

```

