AV2016Q01:
Quantos e quais são os registradores visíveis de propósito geral das máquinas com arquitetura de 64 bits? Quais são as possíveis maneiras (com relação ao tamanho das palavras) de acessar esses registradores Gerais?

AV2016Q02:
Considere um suposto arquivo de texto chamado `copaKartComputacao.asm` que contém um código escrito em assembly com sintaxe Intel. Considere também que este código foi escrito para uma máquina de arquitetura de 64 bits e o montador nasm está instalado corretamente. Quais são os comandos necessários, em terminal linux, para `montar`, `linkar` (ligar) e `executar` a aplicação escrita no arquivo `copaKartComputacao.asm`?

AV2016Q03:
Considere a chamada de sistema `rename()`, que renomeia o arquivo `oldpath` para `newpath`, definida como:

```
int rename(const char *oldpath, const char *newpath);
```

Sendo o código Syscall `0x52 (82d)`.
A função do kernel retorna "0" se a renomeação ocorreu com sucesso, caso contrário "-1".

Considere também o trecho de código do Quadro av2016q03.

```
 1. section .data
 2.       arq1 : db '/home/av01.pdf'
 3.       arq2 : db '/home/lm_av01.pdf'
 4.       str1 : db 'sucesso',0xa, 0
 5.       str2 : db 'falhou ',0xa, 0
 6.
 7. section .text
 8.       global _start
 9. _start:
10.       ; rename()
11.       ; verificação
12. fim:
13.       ; finalização
```

Quadro av2016q03: trecho de código que executa rename.

Implemente/Complete o trecho de código apresentado no Quadro av2016q03 de modo a efetuar a chamada da função `rename()` passando como parâmetro_1 `arq1` e como parâmetro_2 `arq2` (da esquerda para direita). O código também deve verificar se a renomeação ocorreu com sucesso ou falha e imprimir no terminal as respectivas strings da `section .data`.

Não esqueça de finalizar o programa.

AV2016Q04:
Considere o trecho de código funcional do Quadro av2016q04. Informe os valores contidos nos registradores indicados na Tabela av2016q04, após a execução inicial com o debugger gdb configurado com um breakpoint no label `fim`.

```
 1. section .data
 2.    v1 : db 'Kart_'
 3.    v2 : dw 'é_'
 4.    v3 : db '  _  _  _  '
 5.
 6. section .text
 7.    global _start
 8.
 9. _start:
10.    mov eax, [v1]
11.    mov bx , [v1]
12.    mov cl , [v1]
13.    mov r8 , [v1]
14. fim: …
```

Quadro av2016q04: trecho de código da época que existia a Copa Kart de Computação (CoKaCo) – nostalgia!.

Tabela av2016q04: conteúdo dos registradores ao executar até o fim.

Registrador:	RAX	RBX	RCX	R8
Conteúdo:				

Importante: MSb é posicionado à esquerda.
Não é necessário usar tabela ASCII, use string delimitada por 'aspas simples'.

Considere o trecho de código do arquivo theCode`.asm` apresentado no Quadro av2017q0102030405:

```
1. section .data
2.       var1 : db 2
3.       var2 : dw 0
4.       var3 : dd 1
5.       var4 : dq 7
```

Quadro av2017q0102030405: esse é theCode, utilizado nas questões av2017q01 até av2017q05.

AV2017Q01:
A Seção do Quadro av2017q0102030405 representa uma área/região do programa em memória.
Qual é essa região?

a) Instruções
b) Dados Inicializados
c) Dados Não-Inicializados
d) Heap/Memória Dinâmica
e) Pilha
f) Nenhuma das Anteriores

AV2017Q02:
Faça o mapeamento das variáveis da memória do código apresentado na Quadro av2017q0102030405. Considere que a posição de memória inicial é 0x100. Para facilitar a produção da resposta, crie uma tabela com 2 linhas e n colunas contendo a posição da memória e seu respectivo conteúdo.
Exemplo:

Endereço:	0x100	0x101	0x102	...
Conteúdo:	var1-byte 0	??	??	...

AV2017Q03:
Considere que o código do Quadro av2017q0102030405 está completo, sem erros e compatível com arquitetura de 64 bits. Quais são os comandos, em terminal linux, para `montar`, `linkar` (ligar) e `executar` esse código? Também escreva o comando para verificar o valor de retorno de uma execução deste programa.

AV2017Q04:
Qual é a faixa de representação, em complemento de dois, para as variáveis var1, var2 e var3 do código do Quadro av2017q0102030405?

AV2017Q05:
Crie uma seção `.bss` com os elementos dim1, dim2, dim3 e dim4. O tamanho desses elementos respeitam os tamanhos dos elementos var1, var2, var3 e var4 do Quadro av2017q0102030405, respectivamente.

Considere a chamada de sistema linux fictícia `nomeNaoCriativo` do Quadro v2017q0607.

```
int nomeNaoCriativo(char *nome, int *dados, int tamanho);
```

Quadro av2017q0607: assinatura da chamada de sistema, utilizado nas questões av2017q06 e av2017q07.

Considere também o trecho de código do Quadro v2017q060708.

```
 1. section .data
 2.       nome : db “ItsVectorTime”, 10, 0
 3.       dados: dd 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF
 4.       cont : db 0
 5.
 6. section .bss
 7.       tamanho : resd 1
```

Quadro av2017q060708: trecho de código utilizado nas questões av2017q06, av2017q07 e av2017q08.

AV2017Q06:
Qual é a linha de código assembly, sintaxe intel, que atribuí o valor imediato 16_{10} ao elemento de memória `tamanho` do Quadro v2017q060708?

AV2017Q07:
Escreva o trecho de código que execute a chamada de sistema `nomeNaoCriativo`, do Quadro v2017q0607, passando os parâmetros do trecho de código do Quadro v2017q060708.
Atente-se à ordem/nomes dos parâmetros.

Considere também o trecho de código do Quadro v2017q08.

```
... ...
09. section .text
10.       global _start
11.
12. _start:
13.       cmp byte [cont], 9
14.       jge etiqueta1
15.
16.       xor r8,r8
17.       mov r8b,[cont]
18.       mov rax, 1
19.       mov rdi, 1
20.       mov rsi, nome
21.       add rsi, r8
22.       mov rdx, 1
23.       syscall
24.
```

```
25.       add byte[cont], 1
26.       jmp _start
27.
28. etiqueta1:
29.       mov rax, 1
30.       mov rdi, 1
31.       mov rsi, nome
32.       add rsi, 13
33.       mov rdx, 1
34.       syscall
35.
36. fim:
37.       mov rax, 60
38.       mov rdi, 0
39.       syscall
```

Quadro av2017q08: último trecho de código da avaliação de LM de 2017.

AV2017Q08:
Considere que o trecho de código do Quadro v2017q060708, em conjunto com o trecho de código do Quadro v2017q08, formam um programa completo e funcional (montável, ligável e executável) em linguagem de assembly, sintaxe intel.
Interprete esse código completo e defina/escreva o que ele executa.

~~AV2016Q01~~AV2017Q08:
Quais são os registradores visíveis de propósito geral existentes em uma máquina x86_64 (arquitetura 64bits Intel/AMD)? Quais são as formas de acessar esses registradores?

Look at next page!
(⊙.⊙)

AV2020QUNICA:
Construir um código iterativo que calcule o n-éssimo número fibonacci.

```
fib(n)
```

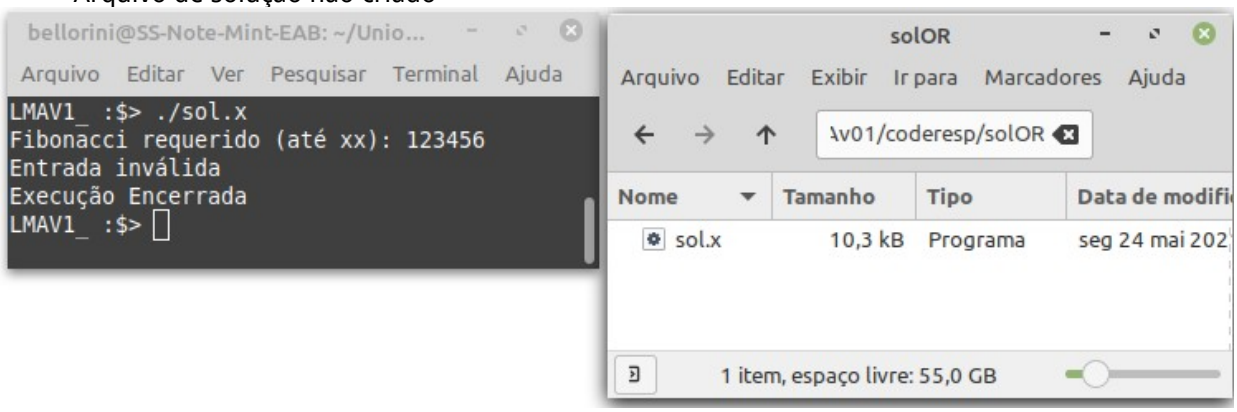
Requisitos:
- Código montável, ligável e executável
- O cálculo do n-éssimo número fibonacci deve ser iterativo
  - não usar recursividade, pois não estudamos funções até o momento
- Sintaxe Intel x64_86

Funcionamento:
- Deve ser solicitado ao usuário a entrada do n-éssimo número fibonacci buscado
  - A entrada é pelo teclado
    - uma string de caracteres ASCII que representa o número
    - máximo com 2 dígitos
      - usuário entrará sempre com 0, 1, 2 ou +3 dígitos e enter para finalizar
      - nunca caracteres alfabéticos ou especiais.
- Verificação de entrada
  - 1 ou 2 dígitos
    - verificação de limites de representação
      - Quanto é fib(99)?
      - Qual é o tamanho do registrador x64_86?
  - n=0 ou +3 dígitos
    - mensagem de falha genérica, limpeza de buffer e encerramento
- Conversão dos dígitos ASCII para número equivalente
    - dicas:
      - ASCII para todos os números são:

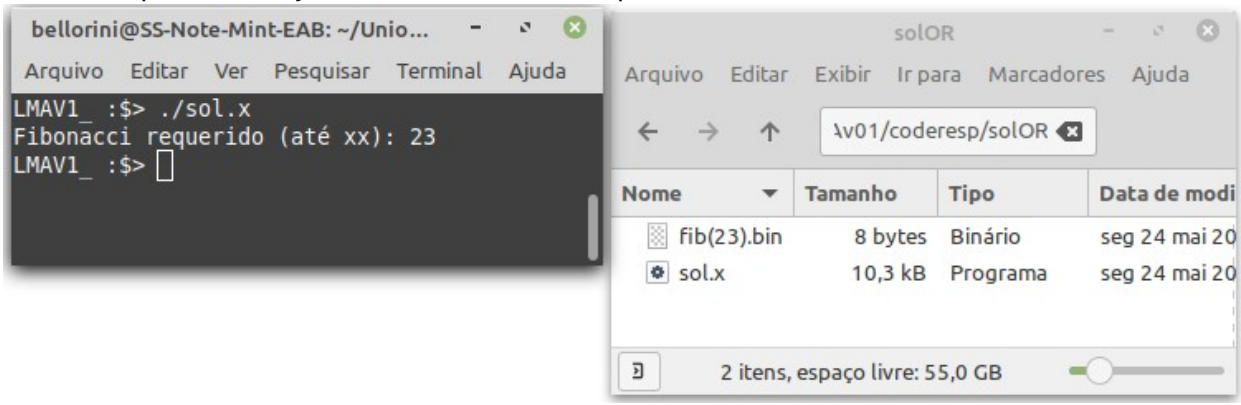
ASCII:	0011 0000	0011 0001	0011 0010	0011 0011	0011 0100	0011 0101	0011 0110	0011 0111	0011 0100	0011 0101
HEX:	0x30	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38	0x39
Dígito:	“0”	“1”	“2”	“3”	“4”	“5”	“6”	“7”	“8”	“9”
Num:	0	1	2	3	4	5	6	7	8	9

      - Após, usar a notação posicional
        - Exemplo1: 5_d é escrito como (0) + 5
        - Exemplo2: 34_d é escrito como (10*3) + 4
        - Exemplo3: 57_d é escrito como (10*5) + 7
      - Cuidado: verifique no gdb a ordem dos dígitos na memória!
- Caso a conversão tenha sucesso e o limite não tenha sido excedido
  - Calcular, de forma iterativa, fibonacci
    fib(0) = 0
    fib(1) = 1
    fib(2) = fib(1) + fib(0)
    fib(i) = fib(i-1) + fib(i-2)
    - Utilize os links para verificar sua solução.
      - WolframAlpha ou Ke!sanOC
- Para finalizar, grave um arquivo binário
  - Nome do arquivo: fib(n).bin, onde n é a entrada do usuário
  - Conteúdo: resultado em “formato” binário
    - isto é, não é necessário converter inteiro para caracteres ASCII.

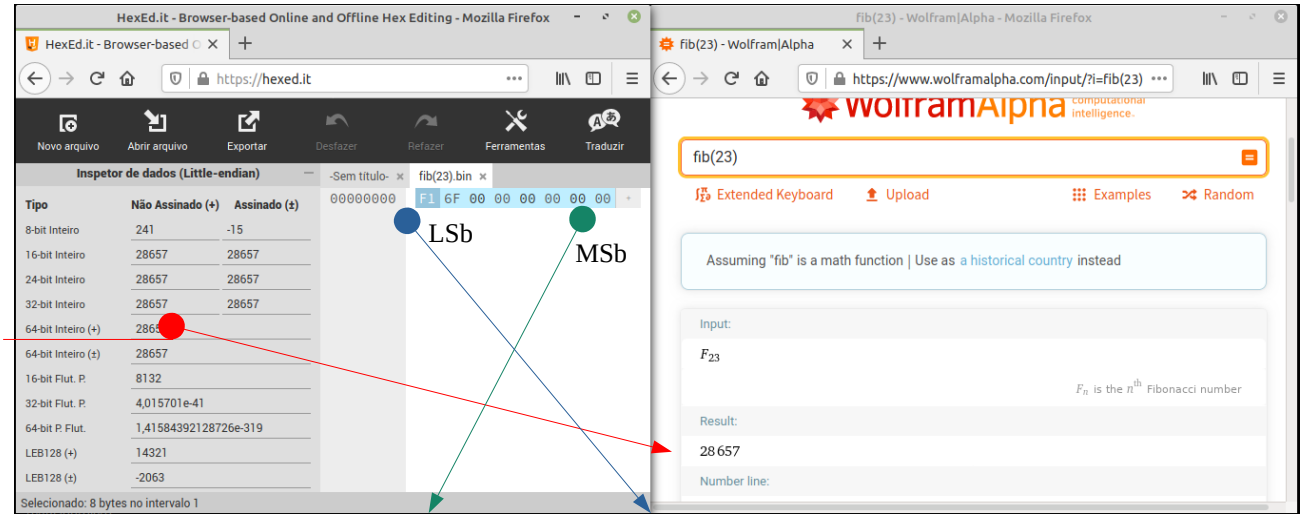
- Exemplo de execução com falha:
  - Mensagem de erro genérica
  - Limpeza de buffer
  - Arquivo de solução não criado



- Exemplo de execução com sucesso:
  - Sem mensagem
  - Arquivo de solução criado com o nome no padrão fib(n).bin



  - Arquivo de resultado:
    - utilize algum editor hexadecimal
      - sugestão: HexEdit



0x0000000000006FF1

(╯°□°)╯ ┻━┻