Lista de Exercício E02
2022

AV2016
AV2017
AV2021

2016AV02Q01:
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 os registradores classificados como Gerais[1]? É possível criar mais registradores de propósito geral, além dos citados na resposta, durante a escrita de um código em assembly?

nota(s) da questão:
[1] Durante uma aula da etapa 02 de LM, aluno R.B. pediu se poderia criar outro registrador, porque não gostava dos nomes atuais.

2016AV02Q02:
Considere que você implementou um novo editor de planilhas para o libre office no arquivo excellibre.asm[1] 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 nesta. O código faz uso somente de chamadas para as funções printf() e scanf() do C. Quais são os comandos necessários, em terminal linux, para montar e linkar (ligar) esta aplicação?

nota(s) da questão:
[1] Durante uma aula de LM, aluno J.V. pediu se tabela de um exercício poderiam ser criada no Execel do libreoffice.

~~2016AV02Q03:~~ (não aplicável - ignorar)
~~Quantos e quais são os registradores da FPU (ponto-flutuante)? Quais são as instruções que transferem os números reais da memória para estes registradores e para memória destes registradores?~~

2016AV02Q04: (questão adaptada para SSE e AVX, originalmente solicitava MMX)
Quantos e quais são os registradores SSE2? Cite 2 exemplos de instruções aritméticas e 1 exemplo de instrução lógica que opere nestes registradores.

2016AV02Q05:
Considere que uma grande empresa de produção de aeromodelos contratou você, aluno de CC da Unioeste – Cascavel, para transcrever um código em C, utilizado no mais recente modelo de Yellowcóptero[1], lançado especialmente para os cheaters[2] de pokemon[2], e apresentado no Quadro 2016AV02Q05, para Assembly (Sintaxe Intel).


```
 1. #include<stdio.h>
 2. #define flap 20
 3.
 4. unsigned int dflap(unsigned int n, unsigned  int exp){
 5.    return n*exp;
 6. }
 7.
 8. int main(){
 9.    printf("rotação em: %d\n", dflap(flap,2));
10.    return 0;
11. }
```

Quadro 2016AV02Q05. Código para cálculo do rotor traseiro do Yellowcóptero.

Observações:
- O código pode ser transcrito para arquitetura x64 ~~ou x86~~ e obrigatoriamente obedecer o protocolo de chamada de funções Unix-like da arquitetura escolhida.
- Caso algum dos passos do protocolo de chamada de subprogramas não seja necessário para completude da transcrição, é importante que este seja descrito e comentado para avaliação desta questão.
- Os valores de n e exp são sempre positivos e (n*exp) < +4.294.967.295

nota(s) da questão:
[1] Durante uma aula de LM, o aluno G.M. solicitou ao professor o que "era um pontinho amarelo no céu?"
[2] Durante outra aula de LM, o aluno J.V. solicitou se poderia deixar para fazer a prática depois porque "tinha um pokemon que ele queria pegar hoje usando VPN".

~~2016AV02Q06:~~ (não aplicável - ignorar)
~~Descreva as principais diferenças entre os protocolos de chamada de sub-programas das arquiteturas x64 e x86.~~

Gotta Catch 'Em All, depois da prova!

2017AV02Q01: (questão adaptada para SSE e AVX, originalmente solicitava FPU)
Quantos e quais são os registradores visíveis de propósito específico para ponto-flutuante SSE/AVX? ~~Qual destes registradores é considerado o topo da pilha de registradores de ponto-flututante? Responda com os nomes dos registradores usados para acesso aos dados.~~

2017AV02Q02: (questão adaptada para SSE e AVX, originalmente solicitava FPU)
Quais são as instruções que movem dados escalares entre a memória e um registrador de ponto-flutuante SSE/AVX? Exemplifique o uso destas instruções.

2016AV02Q03:
Considere que você implementou um programa para arquitetura x86_64 (64bits), que calcula a nota final dos alunos de Linguagem de Montagem no arquivo lmFinais.asm. Este programa implementa chamadas às funções printf() e scanf(). Qual é a sequência de comandos, em terminal Linux, para montar, ligar e executar o programa lmFinais.asm?

~~2016AV02Q04:~~ (não aplicável - ignorar)
~~Considere o código do Quadro 2016AV02Q04 abaixo:~~

```
 1. section .data
 2.     var1: dw 1, 2, 3, 4, 5, 6, 7, 8, 9, 0
 3.     var2: dw 1, 2, 3, 4, 5, 6, 7, 8, 0, 9
 4. section .bss
 5.     var3: resw 10
 6.     section .text
 7. global _start
 8. _start:
 9.     movq   mm0, [var1]
10.     movq   [var3], mm0
11.     movq   mm1, [var2]
12.     paddw  mm0, mm1
13.     movq   [var3], mm0
14. fim:
15.     mov rax, 1
16.     mov rbx, 0
17.     int 0x80
```

Quadro 2016AV02Q04. código completo e funcional para questão Q04.

Qual é o valor contido em var3 ao executarmos este código até o label fim?

(**2016AV02Q05** na próxima página)

2016AV02Q05:
Considere o código escrito em linguagem C, completo e funcional, contido no Quadro 2016AV02Q05. Traduza[1] os códigos das funções powerOf3 e isMulPossible para Linguagem de Montagem utilizando sintaxe Intel e tendo como arquitetura alvo a x86_64 (64 bits[2])

[1]Traduzir: não otimize o código
[2]64 bits : respeite o protocolo e comente passos, inclusive os não utilizados!

```
 1. #include<stdio.h>
 2. /* def: Função que Calcula n ao cubo
 3. *  param: n -> ponteiro para o número n
 4. *  return: n^3
 5. */
 6. void powerOf3(int *n){
 7.     int p = *n;
 8.     p = p*p*p;
 9.     *n = p;
10.     return *n;
11. }
12.
13. /* def: Função que Calcula o número de multiplicações usadas
14. *  em uma função de multiplicação de matrizes tradicional
15. *  param: c1 -> número de colunas da Matriz M1
16. *         l2 -> número de linhas  da Matriz M2
17. *  return: Se possível, retorna o número de multiplicações usadas
18. *          Caso contrário, retorna 0
19. */
20. int isMulPossible(int c1, int l2){
21.     int multiplicacoes = 0;
22.     if (c1 == l2){
23.         multiplicacoes = c1;
24.         powerOf3(&multiplicacoes);
25.     }
26.     return multiplicacoes;
27. }
28.
29. /* def: Função principal
30. *  executa a chamada à função isMulPossible e imprime resultado
31. */
32. int main(){
33.     int Mc1 = 5;
34.     int Ml2 = 5;
35.     printf("Número de Multiplicações: %d\n", isMulPossible(Mc1,Ml2));
36.     return 0;
37. }
```

Quadro 2016AV02Q05. Código completo e funcional em C para Questão 05.

(**AV2021Q01** na próxima página)

AV2021Q01 – Quantos são e quais são os registradores de propósito específico para ponto-flutuante em máquinas com arquitetura IA32-64 (x86-64, 64bits) com extensão SSE2/AVX?

AV2021Q02 – Considere a seguinte seção de código de dados inicializados e não inicializados abaixo:

```
01.  section .data
02.          num1 : dd 13.0
03.          num2 : dd 29.0
04.
05.  section .bss
06.          resp : resq 1
```

Quadro AV2021Q02. Dados inicializados e não inicializados para questão Q2.

Qual é a sequencia de instruções IA32-64 para efetuar o seguinte comando:

```
01.          resp = num1 + num2     ; resposta em resp = 42
```

AV2021Q03 – Considere o código `av2021q03.asm` abaixo:

```
01.  extern printf
02.  extern scanf
03.
04.  section .data
05.      numv : dd  4, 8, 15, 16, 23, 42, 55, 89, 91, 99
06.      str1 : db "Entre com seu RA: ", 0
07.      str1c: db "%d", 0
08.      str2c: db "resultado = %d", 10, 0
09.  section .bss
10.      ra : resd 1
11.
12.  section .text
13.      global main
14.
15.  main:
16.      push rbp
17.      mov rbp, rsp
18.
19.      xor rax, rax
20.      lea rdi, [str1]
21.      call printf
22.
23.      xor rax, rax
24.      lea rdi, [str1c]
25.      lea rsi, [ra]
26.      call scanf
27.      |
27.      |
28.      lea rdi, [ra]
29.      mov esi, 10
30.      call exec
31.
32.      lea rdi, [str2c]
32.      mov esi, [numv + eax*4]
33.      xor rax, rax
34.      call printf
35.
36.  fim:
37.      mov rsp, rbp
38.      pop rbp
39.
40.      mov rax, 60
41.      mov rdi, 0
42.      syscall
43.
44.  exec:
45.      push rbp
46.      mov rbp, rsp
47.
48.      xor rdx, rdx
49.      mov eax, [rdi]
50.      idiv esi
51.      mov eax, edx
52.
53.      mov rsp, rbp
54.      pop rbp
55.      ret
```

Quadro AV2021Q03. Código av2021q03.asm completo, montável, ligável e possível de ser executado em máquinas IA32-64.

a) Qual(is) é(são) a(s) linha(s) de comando(s) para montar, ligar e executar o código?
b) Qual é a assinatura do subprograma `exec`?
c) Qual é a saída em terminal apresentada ao executar o código? Use seu R.A. como entrada.

AV2021Q04 – E mais uma vez, considere o código parcial av2021q04 abaixo:

```
01.  extern printf
02.
03.  section .data
04.      strControle : db "char   = %c", 10,
                        "hInt   = %hu", 10,
                        "fInt   = %u", 10,
                        "lInt   = %lld", 10,
                        "float  = %f", 10,
                        "double = %lf", 10,
                        "string = %s", 10, 0

05.      halfInt  : dw 42000
06.      fullInt  : dd 420000000
07.      longInt  : dq 4200000000000000000
08.      fpSingle : dd 3.1415
09.      fpDouble : dq 3.1415
10.      stringS : db "quarenta e dois", 10, 0
11.      charC   : db "a"
12.
13.  section .text
14.      global main
15.
```

```
15.
16.  main:
17.      push rbp
18.      mov rbp, rsp
19.
20.      ; printf utilizando os parâmetros
21.      ???
22.      ???
23.      ???
24.      ???
25.      ???
26.      ???
27.      ???
28.      ???
29.      ???
30.      call printf
31.
32.  fim:
32.      mov rsp, rbp
33.      pop rbp
34.
35.      mov rax, 60
36.      mov rdi, 0
37.      syscall
```

Quadro AV2021Q04. Código av2021q04.asm parcial para questão Q04.

Escreva somente o código para as linhas 20 até 28 que organize os parâmetros apresentados na string de controle `strControle` da linha 05 para a chamada do `printf` da linha 29.

Pegadinhas? Espero que não, então, vamos lá:

→ formatos printf:

```
%c    → caracter de 1 byte
%hu   → inteiro de meia palavra (2 bytes) não sinalizado
%u    → inteiro "tradicional" não sinalizado
%lld  → inteiro de palavra dupla (8 bytes) sinalizado
%f    → ponto-flutuante de precisão simples (float)
%lf   → ponto-flutuante de precisão dupla (double)
%s    → ponteiro para string null-terminated
```

→ observações printf

```
passagens por valor para caracteres, inteiros e ponto-flutuantes
passagens por referência para strings
```

→ saída esperada ao executar o código av2021q04.x

```
./av2021q04.x
char   = a
hInt   = 42000
fInt   = 420000000
lInt   = 4200000000000000000
float  = 3.141500
double = 3.141500
string = quarenta e dois
```