Tamanho ocupado pelo array:

n \* sizeof(primeiro elemento)

.section .bss #section bss (uninitialized variables)

.comm array\_c, 48 #space for 6 long (6\*8 bytes), variable name: array\_c

.section .data #section data (initialized variables)

array\_e:

.int 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60

.section .bss

.com str\_a, 10

.section .data

Str\_b:

.asciz “computer architecture” #string initialization

Computação de endereços: leaq

Leaq origem, destino

Instrução leaq (load effective address) permite obter o endereço da zona de memoria de origem, armazenando-o no registo definido em destino

IMPORTANTE: ATENDENDO A QUE UM ENDEREÇO EM X86-64 TEM 8 BYTES, A INSTRUÇAO LEAQ NÃO TEM OUTRA EQUIVALENTE PARA OUTROS TAMANHOS

C:

Int vec[] = {1,2,3,4,5}

Int\* get\_vec\_address(){

return vec;

}

ASSEMBLY:

.section .data

.global vec

.section .text

.global get\_vec\_address:

leaq vec(%rip), %rax

ret

ENDEREÇAMENTO DE MEMÓRIA

Enderecimento indireto

movX (registo), destino ->>>> os parentesis são como o asterisco do C e fazem apontador

ENDEREÇAMENTO AO BYTE

PARA MOVER O APONTADOR É PRECISO SABER O TAMANHO DO ELEMENTO

USAR A VARIEDADE CORRETA DA INSTRUÇÃO MOV

NÃO SE PODE FAZER movX (%rbx), (%rax) PORQUE O MOV NÃO PODE UTILIZAR 2 ENDEREÇOS DE MEMORIA

ACEDER A VALORES

.section .data

array\_a:

.int 10,15,20,25,30,35,40,35,50,55,60

.section .text

.global tenth\_element

tenth\_element:

Leaq array a(%rio), %rdi #copiar o endereço de array\_a para rdi

addq $36, %rdi #avançar até ao 10º elemento ((10-1)\*4=36)

movl (%rdi), %eax #copiar o valor apontado por rdi para eax

ret

.section .data

Str:

.asciz “computer architecture”

.section .text

.global fifth\_char

Fifth\_char:

Leaq str(%rip), %rdi #copiar o endereço de str para rdi

Addq $4, %rdi #avançar até ap 5º elemento ((5-1)\*1=4)

movb (%rdi), %al

ret

Endereçamento indireto com offset

movX offset(registo), destino

movl 8(%rdi), %eax #rdi é um apontador para um array de inteiros

coloca em eax o valor representado com 4 bytes que está armazenado na memória, 8 bytes após o endereço apontado pelo registo rdi

equivalente:

addq $8, %rdi

movl (%rdi), %eax

subq $8, %rdi #restaurar o valor inicial de rdi

base\_address + offset + index \* size

.section .data

Array\_a: #declarar array de inteiros array\_a

.int 10,15,20,25,30,35

.section .text

Exemple:

Leaq array\_a(%rip),%rdi #endereço do array\_a em rdi

Movq $2, %rcx #indice pretendido: 3º elemento

Movl (%rdi,%rcx,4), %eax #copiar array\_a[2] para eax. 4 por ser um array de ints

Ret

INCREMENTAR EM UMA UNIDADE TODOS OS ELEMENTOS DE UM ARRAY DE INT

.section .data

.global vec

Vec:

.int 1,2,3,4,5,6,7,8,9,10

.equ SIZE, 10

.section .text

.global inc\_by\_one

Inc\_by\_one:

Leaq vec(%rip),%rdi #store the vec address in rdi

Movq $SIZE, %rcx

Cmpq $0, %rcx

Jle end

Inc\_loop:

Incl (%rdi)

Addq $4, %rdi

Loop inc\_loop

End:

Ret

CONTAR O NR DE CHARS NUMA STRING

.section .data

Str:

.asciz “computer srchitecture 101”

.section .text

.global str\_count

Str\_count:

Leaq str(%rip), %rdi #store the str address in rdi

Movl $0, %eax #string length = 0 #string length = 0

Str\_loop:

Movb (%rdi), %cl #cl=char pointed by rdi

Cmpb $0, %cl #validate size of vec

Je end\_str\_loop

Incl %eax #increase counter

Incq %rdi #point to next char

Jmp str\_loop

End\_str\_loop:

Ret