Systemy komputerowe

Lista zadań nr 3

Karolina Szlęk 300411

Zadanie 1

Adres	Wartość	Rejestr	Wartość
0x100	0xFF	%rax	0x100
0x108	0xAB	%rcx	1
0x110	0x13	%rdx	3
0x118	0x11		

Oblicz wartość poniższych operandów. Wskaż, które z nich mogą dotyczyć adresu w pamięci.

ODPOWIEDŹ

1. %rax -----> **0x100**

2. $0x110 -----> **0x110** - bo symbol $ oznacz, że bierzesz licze

3. 0x108 -----> **0xAB**

4. (%rax) -----> [0x100] = **0xFF**

5. 8(%rax) -----> [0x100 + 8=0x108] = **0xAB**

6. 23(%rax,%rcx) -----> 23 = 0x17

   [0x17+ 0x100 + 1 = 0x118] = **0x11**

7. 0x104(,%rdx,4) -----> [4*%rdx + 0x104 = 4*3+0x104 = [0x110]= **0x13**

8. (%rax,%rdx,8) -----> [0x100 + 3*8] = [0x100 + 0x18] = **0x11**

9. 265(%rcx,%rdx,2) -----> [265 + 1+ 3*2] = [0x109 + 0x7] = [0x110] = **0x13**

Adresu w pamięci dotyczą podpunkty: 3,4,5,6,7,8,9

Zadanie 2

	Treść	Adres	Wartość
1	addq %rcx,(%rax)	0x100	0xFF + 1 = **0x100**

2.	subq 16(%rax),%rdx	%rdx	3 - 0x13 = -16 = -0x10
3.	shrq $4, %rax	%rax	0x100>>4 = **0x10**
4.	incq 16(%rax)	0x100	0x100 + 0x10 = 0x110; [0x110] ++ = **0x14**
5.	decq %rcx	%rcx	1 -- = **0**
6.	imulq 8(%rax)	%rdx, %rax	0x100 * 0xAB = **0xAB00**
7.	leaq 7(%rcx, %rcx,8),%rdx	%rdx	7 + 1 + 1*8 = **0x10**
8.	leaq OxA(,%rdx,4),%rdx	%rdx	0xA + 3*4 = 22 = **0x16**

Notatka

3. 256/2/2/2/2

Zadanie 3.

8-byte register	Bytes 0-3	Bytes 0-1	Byte 0
%rax	%eax	%ax	%al
%rcx	%ecx	%cx	%cl
%rdx	%edx	%dx	%dl
%rbx	%ebx	%bx	%bl
%rsi	%esi	%si	%sil
%rdi	%edi	%di	%dil
%rsp	%esp	%sp	%spl
%rbp	%ebp	%bp	%bpl
%r8	%r8d	%r8w	%r8b
%r9	%r9d	%r9w	%r9b
%r10	%r10d	%r10w	%r10b
%r11	%r11d	%r11w	%r11b
%r12	%r12d	%r12w	%r12b
%r13	%r13d	%r13w	%r13b
%r14	%r14d	%r14w	%r14b
%r15	%r15d	%r15w	%r15b

In the following tables,

- "byte" refers to a one-byte integer (suffix **b**),
- "word" refers to a two-byte integer (suffix **w**),
- "doubleword" refers to a four-byte integer (suffix **l**), and
- "quadword" refers to an eight-byte value (suffix **q**).

Robię z jednego na drugie i zawsze patrzę na to co ma mniejszą ilość bajtów i to biorę.

1.	mov %eax,(%rsp)	mov**l**	%eax jest rejestrem 32-bitowym
2.	mov (%rax),%dx	mov**w**	%dx jest rejestrem 16-bitowym
3.	mov $0xFF, %bl	mov**b**	%bl jest rejestrem 8-bitowym
4.	mov (%rsp,%rdx,4),%dl	mov**b**	%dl jest rejestrem 8-bitowym
5.	mov (%rdx),%rax	mov**q**	%rax jest rejestrem 64-bitowym, a (%rdx) wskazuje na miejsce w pamięci, również 64-bitowe
6.	mov %dx, (%rax)	mov**w**	%dx jest rejestrem 16-bitowym

Zadanie 4.

1.	movb $0xF,(%ebx)	Jest dobrze. Wartość w %ebx zostanie skonwertowana do 64-bitów.
2.	movl %rax,(%rsp)	**Błąd.** Błędny suffix - tyczy się operandów 4 bajtowych, a w poleceniu mamy operandy 8 bajtowe.
3.	movw (%rax),4(%rsp)	**Błąd,** nie można robić transferu memory-memory. Dwa nawiasy nie są dozwolone. Src i Dest nie mogą jednocześnie odwoływać się do pamięci. Nie możemy bezpośrednio przenosić coś z jednego miejsca pamięci w drugie.
4.	movb %al,%sl	**Błąd**. Nie ma takiego rejestru jak %sl.
5.	movq %rax, $0x123	**Błąd,** przez $0x123 – nie może być wartością. Jest stałą, a nie możemy przenieść czegoś do stałej.

6.	movl %eax,%rdx	**Błąd,** Błędny suffix, tyczy się operandów 4 bajtowych, a %rdx jest operandem 8 Bajtowym.
7.	movb %si,8(%rbp)	**Błąd.** Błędny suffix - tyczy się operandów 1 bajtowych a %si jest operandem 2 Bajtowym.

Zadanie 5.

NOTATKA

leaq bierze source i go wylicza oraz następnie zapisuje do destination.

	Treść zadania	Zawartość rejestru %rdx
1.	leaq 6(%rax),%rdx	**x+6**
2.	leaq (%rax,%rcx),%rdx	**x+y**
3.	leaq (%rax,%rcx,4),%rdx	**x+4y**
4.	leaq 7(%rax,%rax,8),%rdx	**9x+7**
5.	leaq OxA(,%rcx,4),%rdx	**4y+10**
6.	leaq 9(%rax,%rcx,2),%rdx	**x+2y+9**

Zadanie 6.

Skorzystamy z tego, że:

-1 = =~x + x |+1
0 == ~x + x + 1 |-x
-x = ~x + 1

Dlatego, zamiast **y – x** skorzystamy z **y + ~x +1**
Zatem nasze rozwiązanie wygląda następująco:

notq %rsi
addq %rsi, %rdi
incq %rdi

NOTATKA
~~Mniej inteligentne (a bardziej oczywiste) ZŁE~~
~~rozwiązanie byłoby takie:~~

~~notq %rsi~~

~~incq %rsi~~

~~addq %rsi, %rdi~~

~~Dlaczego ono jest złe?~~
~~Po negacji %rsi i inkrementacji moglibyśmy osiągnąć overflow i ostateczne dodawanie dało by nam zły wynik. np . dla %rsi = INT_MIN.~~

Zadanie 7.

Funkcję o sygnaturze «uint64_t compute(int64_t x, int64_t y)» na kod asemblerowy.
compute:

```
leaq (%rdi,%rsi), %rax
movq %rax, %rdx
sarq $31, dx
xorq %rdx,%rax
subq %rdx, %rax
ret
```

Argumenty «x» i «y» zostały przekazane funkcji «compute» odpowiednio przez rejestry %rdi i %rsi, a wynik został zwrócony przez instrukcję ret w rejestrze %rax.
Jaką wartość oblicza ta funkcja?
Wskazówka To, że sarq jest instrukcją przesunięcia arytmetycznego a nie logicznego jest w tym zadaniu istotne.

Zapiszmy funkcje trochę inaczej

%rax = x + y
%rdx = %rax
%rdx = %rdx>>31
%rdx = %rax ^ %rdx
%rdx = %rax - %rdx

Niech z= x+ y

Funkcja obliczania wartości wyrażenia wygląda wtedy tak:
((z >> 31) ^ z) - (z >> 31)

Inaczej:
|x + y|

Ponieważ robimy przesunięcie o 31 bitów oraz x, y są typu int64_t,

x i y muszą się mieścić w zakresie od –2^31 do 2^31 - 1

Zadanie 8.

Rozwiąż poprzednie zadanie dla funkcji «int16_t compute2(int8_t m, int8_t s)». Jak poprzednio, pierwszy argument został przekazany w rejestrze %rdi, drugi w %rsi a wartość zwracana jest w %rax. Funkcja operuje na 8-, 16-, 32 i 64-bitowych rejestrach, a zwraca wynik w rejestrze 64-bitowym. Wyjaśnij, jak poszczególne wiersze kodu zmieniają starsze bajty rejestrów, których młodszymi bajtami są ich operandy.

compute2:

movsbw %dil, %di ---> **ta komenda przenosi z rejestru b(8 bitowego) do w(16 bitowego)**

movl %edi, %edx ---> **edx = edi, bity starsze, niż 31 zostaną wyzerowane**

sall $4, %edx ---> **edx zostaje przesunięty o 4 bity, więc teraz edx = edi*2^4 = edi*16**

subl %edi, %edx ---> **edx = edx - edi = 15*edi**

leal 0 (,%rdx,4), %eax ---> **eax = rdx*4, czyli teraz eax = 4*15*edi**

movsbw %sil, %si ---> **sil jest rzutowane na 16 bitów (rozszerzenie liczny ze znakiem 8-bitowej do 16-bitowej)**

addl %esi, %eax ---> **eax = eax + esi = 60*edi + esi**

ret ---> **zwraca nam wartość rejestru eax, czyli 60*edi + esi = 60*m + s**