Systemy komputerowe

Lista zadań nr 3

Na zajęcia zdalne 23 – 26 marca 2020

Zmiana w stosunku do wersji roboczej: dodano zadanie 9.

Zadanie 1. Poniżej podano wartości typu `int64_t` leżące pod wskazanymi adresami i w rejestrach:

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.

1. `%rax`
2. `$0x110`
3. `0x108`
4. `(%rax)`
5. `8(%rax)`
6. `23(%rax,%rcx)`
7. `0x104(,%rdx,4)`
8. `(%rax,%rdx,8)`
9. `265(%rcx,%rdx,2)`

Zadanie 2. Każdą z poniższych instrukcji wykonujemy w stanie maszyny opisanym tabelką z zadania 1. Wskaż miejsce, w którym zostanie umieszczony wynik działania instrukcji, oraz obliczoną wartość.

1. `addq %rcx,(%rax)`
2. `subq 16(%rax),%rdx`
3. `shrq $4,%rax`
4. `incq 16(%rax)`
5. `decq %rcx`
6. `imulq 8(%rax)`
7. `leaq 7(%rcx,%rcx,8),%rdx`
8. `leaq 0xA(,%rdx,4),%rdx`

Zadanie 3. Dla każdej z poniższych instrukcji wyznacz odpowiedni sufiks (tj. `b`, `w`, `l` lub `q`) na podstawie rozmiarów operandów:

1. `mov %eax,(%rsp)`
2. `mov (%rax),%dx`
3. `mov $0xFF, %bl`
4. `mov (%rsp,%rdx,4),%dl`
5. `mov (%rdx),%rax`
6. `mov %dx,(%rax)`

Zadanie 4. Które z poniższych linii generuje komunikat błędu asemblera i dlaczego?

1. `movb $0xF,(%ebx)`
2. `movl %rax,(%rsp)`
3. `movw (%rax),4(%rsp)`
4. `movb %al,%sl`
5. `movq %rax,$0x123`
6. `movl %eax,%rdx`
7. `movb %si,8(%rbp)`

Zadanie 5. Rejestry `%rax` i `%rcx` przechowują odpowiednio wartości `x` i `y`. Podaj wyrażenie, które będzie opisywać zawartość rejestru `%rdx` po wykonaniu każdej z poniższych instrukcji:

1. `leaq 6(%rax),%rdx`
2. `leaq (%rax,%rcx),%rdx`
3. `leaq (%rax,%rcx,4),%rdx`
4. `leaq 7(%rax,%rax,8),%rdx`
5. `leaq 0xA(,%rcx,4),%rdx`
6. `leaq 9(%rax,%rcx,2),%rdx`

Zadanie 6. Zastąp instrukcję `subq %rsi, %rdi` równoważnym ciągiem instrukcji bez jawnego użycia operacji odejmowania. Można używać dowolnych innych instrukcji i rejestrów.

Zadanie 7. Kompilator przetłumaczył funkcję o sygnaturze «`uint64_t compute(int64_t x, int64_t y)`» na następujący kod asemblerowy.

```
compute:
        leaq    (%rdi,%rsi), %rax
        movq    %rax, %rdx
        sarq    $31, %rdx
        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.

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
        movl    %edi, %edx
        sall    $4, %edx
        subl    %edi, %edx
        leal    0(,%rdx,4), %eax
        movsbw  %sil, %si
        addl    %esi, %eax
        ret
```

Zadanie 9. Zinterpretuj poniższe stałe szesnastkowe jako liczby pojedyńczej precyzji (32-bitowe) w formacie IEEE 754, następnie wykonaj dodawania i zapisz wynik w takim samym formacie.

1. `0xC0D20004 + 0x72407020`
2. `0xC0D20004 + 0x40DC0004`
3. `(0x5FBE4000 + 0x3FF80000) + 0xDFDE4000`