LISTA 3

Zad. 1

wartości typu int64_t

ADRES	WART.	REJESTR	WART.
0x100	0xFF	%rax	0x100
0x108	0xAB	%rcx	1
0x110	0x13	%rdx	3
0x118	0x11		

NIE dotyczy adresu pamięci
~~Może dotyczyć adresu~~
Nie dotyczy adresu pamięci
~~Może dotyczyć adresu~~
Pozostałe dotyczą

	WYRAŻENIE	WARTOŚĆ	
1.	%rax	0x100	
2.	$0x100	0x100	
3.	0x108	0xAB	
4.	(%rax) →(0x100)	0xFF	} (%rax) = M[R[E_t]] = M[0x100] = 0xFF
5.	8(%rax)	0xAB	} 8(%rax) = M[8 + 0x100] = M[0x108] = 0xAB
6.	23(%rax,%rcx)	0x11	} = M[23 + 0x100 + 1] = M[24 + 0x100] = M[0x118] = 0x11
7.	0x104(,%rdx,4)	0x13	} M[0x104 + (3 * 4)] = M[0x104 + 12] = M[0x110] =
8.	(%rax,%rdx,8)	0x11	} M[0x100 + 3 * 8] = M[0x100 + 24] = M[0x118] = 0x11
9.	265(%rcx,%rdx,2)	0x13	} M[265 + 1 + 3 * 2] = M[272] = M[0x111]

TO ZAWSZE MUSI BYĆ POTĘGA 2!

Zad. 2

	OPERACJA	WARTOŚĆ	MIEJSCE
1.	addq %rcx,(%rax)	0xFF + 1 = 0x100	0x100
	addq Src, Dest Dest = Dest + Src	256 + 16 = 272	= 0x110
2.	subq 16(%rax),%rdx → 3	3 − (16 + 0x100 = ; 0x13)	%rdx
	subq Src, Dest Dest = Dest − Src		bo 2, czyli miejsce
3.	shrq $4, %rax	0x100 >> 4 = 0x10	%rax
	shrq k, Dest Dest = Dest >> k (logical)		
4.	incq 16(%rax)	ADRES 0x110 → 0x13 + 1 = 0x14	? jak 0x13 to wiem
	incq Dest Dest = Dest + 1		0x100
5.	decq %rcx	1 − 1 = 0	%rcx
	decq Dest Dest = Dest − 1	→ Można przez pętle lub (100 S * rax > 2^{64} => >>	
6.	imulq 8(%rax) → mnożymy bez znaku (→ rejestr 64 bitowy)	0xAB * 0x100 = 0xAB00	%rdx:%rax
	imulq Src, Dest Dest = Dest * Src	→ bo 64*64 = 128 → więc przechowuje na 2 bitach	

imulq S signed full multiply of %rax by S
Result stored in %rdx:%rax

Scanned with CamScanner

%rdx

7. leaq 7(%rcx,%rcx,8),%rdx

leaq Src, Dest Dest = address of Src

M[7 + 1 + 1*8] = M[16] = [0x10]

%rdx

8. leaq 0xA(,%rdx,4),%rdx

M[0xA + 3*4] = M[10+3*4] = M[22] = [0x16]

NOTKA DO ZAD 1/2

Type	From	Operand Value	Name
Immediate	$Imm	Imm	Immediate
Register	Ea	R[Ea]	Register
Memory	Imm	M[Imm]	Absolute
Memory	(Ea)	M[R[Ea]]	Absolute
Memory	Imm(Eb,Ei,S)	M[Imm+R[Eb]+(R[Ei]xS)]	Scaled indexed

Zad. 3

b byte (8 bitów) 1 bajt

w word (2 bytes) (16 bitów) 2 bajty

l long (4 bytes) (32 bitów) 4 bajty

q quad (8 bytes) (64 bitów) 8 bajtów

1 movl %eax(32), (%rsp(64)) bo %eax ma 32 bity

2 movw (%rax(64)), %dx(16) %dx ma 16 bitów

3 movb $0xFF, %bl(8) 8 bitów

4 movb (%rsp(64), %rdx(64), 4), %dl(8) %dl więc 8 bitów

5 movq (%rdx(64)), %rax(64) ma 64 bity

6 movw %dx(16), (%rax(64)) ma 16 bity

Zad. 4

movb $0xF(2), (%ebx(32)) ✓

movl %rcx(64), (%rsp(64)) q

movw (%rax(64)), 4(%rsp(64)) q

movb %al(8), %sil ma ma takiego adresu %sil

movq %rcx(64), $0x123 → to jest wartość, a nie adres

może być tylko jedno wyjątkowe ecx(32) w pojedynczej instrukcji

32 64 ważne rozmiary rejestrów

movl %eax, %rdx powinno być (%rdx) albo %edx

movb %si, 8(%rbp) si ma 2 bajty, a b
16 64 przenosi 1

ZAD. 5

%rdx - z %rax x %rcx y

leaq Src, Dest Dest = adres of Src

leaq przenosi adresy, a nie wartości

1) $z = x + 6$ leaq 6(%rax), %rdx
2) $z = x + y$ leaq (%rax, %rcx), %rdx
3) $z = x + 4 \cdot y$ leaq (%rax, %rcx, 4), %rdx
4) $z = 7 + x + 8 * x = 9x + 7$ leaq 7(%rax, %rax, 8), %rdx
5) $z = 10 + 4 * y$ leaq 0xA(, %rcx, 4), %rdx
6) $z = 9 + x + 2 * y$ leaq 9(%rax, %rcx, 2), %rdx

ZAD. 6

subq %rsi, %rdi

subq Src, Dest Dest = Dest - Src

KORZYSTAMY: A - B == A + (-B)

imulq $-1, %rsi, %rax ? 2 argumenty? na rax zapisze się $-1 * %rsi

addq %rax, %rdi

USTAZ ZAD 3

ZAD. 7

%rdi x
%rsi y

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

? zwraca instrukcje (polecenie)

1) %rax = x + y
2) %rdx = %rax
3) %rdx = %rdx >> 31 (przesunięcie arytmetyczne)
4) %rax = %rax ^ %rdx
5) %rax = %rax - %rdx

Oznaczmy z = x + y

|x + y|

((z >> 31) ^ z) - (z >> 31)

(INT32_MIN; INT32_MAX)

, ale x i y muszą należeć do zakresu $(-2^{32}; 2^{32})$

ponieważ wykonujemy przesunięcie bitowo o 31, a x i y są typu uint64_t

ZAD. 8

movsbw %dil, %di → %rdi } z → wynik
bit word 8 → 16

movl %edi, %edx → %rdx } z = m } 16 bitów
long

sall $4, %edx z << 4 → przesunięcie arytmetyczne (kopiowanie najstarszego po prawej 0)

dopisze 4 zera

ZAD. 8

movsbw %dil, %di // przenosi z rejestru 8(8 bitowego) albo w (16 bitowego)

movl %edi, %edx // edx = edi, bity starsze niż 31 zostaną wyzerowane

sall $4, %edx // edx zostanie przesunięty o 4 // bity więc teraz edx = edi*16

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

leal 0(,%rdx,4), %ecx // ecx = rdx*4 czyli ecx = 4*15*edi

movsbw %sil, %si // sil jest rzutowane na 16 bitów

addl %esi, %ecx // ecx = ecx + esi = 60*edi + esi

ret // zwraca wartość rejestru ecx, czyli 60*edi + esi = 60*m + 5

ZAD 9.

Liczba zmiennoprzecinkowa w formacie IEEE 754 zapisywana jest za pomocą 32 bitów. Z czego 1 przeznaczamy na znak, 8 bitów na wykładnik, a następnie 23 bity na mantysę. BIAS jest równy 127.

1) 0xC0D2C004 + 0x7240702C

↓ |

0x 1100 0000 1101 0010 0000 0000 0000 0100

bit znaku -> 1

wykładnik = 0x1000 0001 - 127 = 129 - 127 = 2

Mantysa = 0x1.1010 0100 0000 0000 0000 100

↓

0x 0111 0010 0100 0000 0111 0000 0010 0000

bit znaku -> 0

wykładnik = 0x1110 0100 - bias = 228 - 127 = 101

Mantysa = 0x1.1000 0000 1110 0000 0100 000