X86_64 instruction set

1 Introduction

The goal of this assignment is to get familiar with some of the `X86_64` instructions. GNU assembler follows *AT&T* syntax. GNU assembler instructions generally have the form mnemonic, source, destination. E.g., `mov $10, %rax`; will move `10` to `%rax` register. In AT&T syntax:

- $ represents a constant value. E.g., $10 means constant number 10.
- An integer value without $ represents an address. E.g., 10 means an address 10.
- Most of the instructions (except few string instructions) have at most one memory operand.
- Instructions are suffixed with the letters "q", "b", "w", "l" to determine the size of the operands. Sometimes, the size can be determined using the size of the register operand. In case of conflicts (mostly due to memory operands), we need to provide suffix.

2 Addressing mode in `X86`

A memory operand is presented in the syntax: segment:disp[base, index, scale]. Here, disp is a 32-bit signed integer, base and index are registers, and scale can be one of the values between 1, 2, 4, and 8. An address is computed using: base of segment + disp + base + (index * scale). Base, index registers are optional (i.e., a memory instruction can only have base or index or none of them). The default segment register is %ds (if no segment register is given). Let's ignore segment registers for this homework and assume that the segment value is always zero. You can refer to Table 1 for some examples.

3 Turn in

Table 2 listed some of the `X86_64` instructions. Some of them are invalid. One way to check if they are valid is to disassemble them using GNU assembler and check for error messages. To disassemble them, create a file temp.s, write the

Operand	Computed address
0x100(%rax, %rdx, 4)	0x100 + %rax + (%rdx * 4)
0x100	0x100
(%rax)	%rax
(%eax)	%eax
0x100(%rax)	0x100 + %rax
(%rax, %rdx, 1)	%rax + (%rdx * 1)
(, %rdx, 1)	(%rdx * 1)
0x100(, %edx, 1)	0x100 + (%edx * 1)
0x100(%eax, %edx, 4)	0x100 + %eax + (%edx * 4)
0x100(, %rdx, 4)	0x100 + (%rdx * 4)

Table 1: Address computation on X86_64 architecture.

instruction as it is, and run **"as temp.s"**. You can specify, multiple instructions in this file separated by a newline. For every instruction in Table 2, write whether it is valid or not. If it is not valid, please give a reason about what it was trying to do, which is not permitted in X86_64. For a valid instruction, you need to write what it is doing.

You may refer to *"Intel manual - 2"* for details about all the X86 instructions. Please note that the Intel syntax, followed in the manual, differs from the AT&T syntax used in this homework.

4 How to submit

Submit your handwritten answers in the class before the lecture.

1	mov $100, 100
2	movb $100, 100
3	movl $100, 100
4	movl $100, 100(%eax, %edx, 8)
5	movq $100, 100(%rax, %rdx, 8)
6	add $100, 100(%eax, %edx, 8)
7	addw $100, 100(%eax, %edx, 8)
8	addq $100, 100(%rax, %edx, 8)
9	addq $100, 100(%rax, %rdx, 8)
10	add $100, %eax
11	add %eax, %ecx
12	add %eax, %rcx
13	lea %eax, %eax
14	lea (%eax), %eax
15	lea 100(%eax), %eax
16	lea 100(%rax, %rbx, 4), %rax
17	lea 100(%rax, %rbx, 4), %eax
18	lea 100(%rax, %ebx, 4), %eax
19	lea 100(%eax, %ebx, 4), %eax
20	lea %eax, 100(%eax)
21	ret
22	jmp 0x100
23	jmpw 0x100
24	jmp *0x100
25	jmpb *0x100
26	jmpw *0x100
27	jmp *rax
28	cmp %rax, (%rax)
29	cmp %eax, (%eax)
30	cmp $100, (%eax)
31	cmpb $100, (%eax)
32	je 0x100
33	je *0x100
34	jne 0x100
35	ja 0x100
36	jb 0x100
37	jae 0x100
38	call 0x100
39	call *0x100
40	callb *0x100
41	call *rax
42	and %eax, (%eax)
43	and %eax, (%rax)
44	and %rax, (%eax)
45	and %eax, %ecx
46	pushb %al
47	pushw %ax
48	push %eax
49	push %rax
50	shl $12, %eax
51	shr $12, %eax
52	or $0x100, %eax
53	xor $100, %eax
54	xchg %eax, %ecx
55	xchg %eax, (%ecx)
56	xchg %rax, (%ecx)
57	xadd %eax, (%ecx)
58	xadd %rax, (%rcx)
59	pushfl
60	popfl
61	lahf
62	sahf
63	rdtsc

Table 2: X86_64 instructions.