Homework 1: X86 instruction set

January 2, 2019

1 Introduction

The goal of this assignment is to get familiar with some of the `X86` instructions. GNU assembler follows *AT&T* syntax. GNU assembler instructions generally have the form mnemonic, source, destination. E.g., `mov $10, %eax`; will move `10` to `%eax` 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 "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` instructions. Some of them are invalid. One way to check if they are valid is to disassemble them using GNU assembler and

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

Table 1: Address computation on X86 architecture.

check for error messages. To disassemble them, create a file temp.s, write the instruction as it is, and run **"as –32 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. For a valid instruction, you need to write what it is doing.

You may refer to *"Intel software developers manual vol-2"* for details about all the X86 instructions.

4 How to submit

Please handle your hand-written answer sheets to the instructor before the lecture begins.

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

Table 2: X86[3]instructions.