Worksheet 2 Solutions

1. `mov` vs `lea` - describe the difference between the following:

```
movl (%rdx), %rax
leal (%rdx), %rax
```

```
movl rdx rax
```

movl takes the **contents** of what's stored in register %rdx and moves it to %rax. leal computes the load effective **address** and stores it in %rax. leal analogous to returning a pointer, whereas movl is analogous to returning a dereferenced pointer.

2. Invalid `mov` Instructions

Explain why these instructions would not be found in an assembly program.

a) `movl %eax, %rdx` — destination operand has the incorrect size

b) `movb %di, 8(%rdx)` — instruction suffix does not match register size

c) `movq (%rsi), 8(%rbp)` — source and destination cannot both be memory references

d) `movw 0xFF, (%eax)` — %eax cannot be used as an address register (not 64 bits)

3. What would be the corresponding instruction to move 64 bits of data from register %rax to register %rcx?

```
movq (%rax), %rcx
```

(important part is that you know the suffix of the MOV instruction!)

4. Operand Form Practice (see page 181 in textbook)

Assume the following values are stored in the indicated registers/memory addresses.

Address	Value	Register	Value
0x104	0x34	%rax	0x104
0x108	0xCC	%rcx	0x5
0x10C	0x19	%rdx	0x3
0x110	0x42	%rbx	0x4

Fill in the table for the indicated operands:

Operand	Value	Operand	Value
$0x110	0x110 (immediate value)	3(%rax, %rcx)	0x19 (value in %rax is 0x104, value in %rcx is 0x5, 3 + 0x104 + 0x5 = 0x10C, value in 0x10C is 0x19)
%rax	0x104 (value stored in %rax)	256(, %rbx, 2)	0xCC (value in %rbx is 0x4, 256 in hex is 0x100, 0x100+(0x4 * 2) = 0x108, value in memory address 0x108 is 0xCC)
0x110	0x42 (value stored in memory address 0x110)	(%rax, %rbx, 2)	0x19 (value in %rax is 0x104, value in %rbx is 0x4, 0x104+(0x4*2) = 0x10C, value in memory address 0x10C is 0x19)

(%rax)	0x34 (%rax holds 0x104, memory address 0x104 holds 0x34)
8(%rax)	0x19 (%rax holds 0x104, 8 + 0x104 = 0x10C, value in memory address 0x10C is 0x19)
(%rax, %rbx)	0xCC (value in %rax is 0x104, value in %rbx is 0x4, 0x104 + 0x4 = 0x108, value in memory address 0x108 is 0xCC)

- $ denotes immediates
- Note: any numbers starting with "0x" are hexadecimal numbers!!
- All of the operands can be evaluated using the specific formulas on page 181 in the textbook
- More generally, whenever you see an address of the form $D(r_b,r_i,s)$, where D is an number, r_b and r_i are registers, and s is either 1,2,4, or 8, you can use the following formula:

  $D + R[r_b] + R[r_i]*s$

  If D is missing, assume D == 0
  If r_b is missing, assume r_b == 0
  If r_s is missing, assume r_s == 0
  If s is missing, assume s == 1
- For more practice, try practice problem 3.1 on page 182 of the textbook

5. Condition Codes and Jumps

Assume the addresses and registers are in the same state as in Problem 2. Does the following code result in a jump to .L2?

```
leaq (%rax, %rbx), %rdi
cmpq $0x100, %rdi
jg .L2
```

Yes.

1. First line will put 0x104+0x04 = 0x108 into %rdi.
2. Second line sets codes according to 0x108 - 0x100, which sets no codes.
3. Since jg is evaluated as ~(SF^OF)&~ZF which in this case is ~(0^0)&~0 = 1&1 = 1.
4. So we will jump.

Students can make mistake of doing a memory read in leaq which would put $0xCC into %rdi. This would mean ZF = 1 after cmpq, which would mean jg logic evaluates to false (no jump).

Problem 6

```
int cool1(int a, int b) {
      if ( b < a )
            return b;
      else
            return a;
}

int cool2(int a, int b) {
      if ( a < b )
            return a;
      else
            return b;
}

int cool3(int a, int b) {
      unsigned ub = (unsigned) b;
      if ( ub < a )
            return a;
      else
            return ub;
}
```

Which of the functions would compile into this assembly code:

```
       movl %esi, %eax
       cmpl %eax, %edi //a-b
       jge .L4 //SF = OF
       movl %edi, %eax
.L4:   ret
```

cool2

- Arguments passed to a function is stored in the %edi, %esi, etc registers
  - %edi is a and %esi is b
- When comparing, we compare as *cmp Two One*
  - Thus the instruction jge is checking if %edi is greater than or equal to %eax
  - This is essentially checking if a >= b, which is the else condition
- We can observe that when we do jump, %eax is not updated
  - We return b in the else case

- If we don't jump, we update %eax to %edi
  - We return a in the if case
- Thus cool2
- This question was inspired by a previous midterm