Name: _____

Directions: **Work only on this sheet** (on both sides, if needed); do not turn in any supplementary sheets of paper. There is actually plenty of room for your answers, as long as you organize yourself BEFORE starting writing. In order to get full credit, **SHOW YOUR WORK**.

## An $\infty$ number of points will be deducted for illegibility.

**1.** Consider 6-bit 2s complement storage.

(a) (5) What is the representation of -3?

(b) (5) What ranges of numbers can be represented?

**2.** (25) What would be the output of the following code?

```
main()
{  int i; float x; char *p;
   p = &x;
   x = 56.12;  // see p.14 in "Inform. Rep. and Storage" PLN!
   printf("%x %c %d %u\n",*(p+3),*(p+3),*(p+3),*(p+3));
}
```

**3.** Consider the instruction

```
addl %eax, (%ebx)
```

Assume a 32-bit machine, and ignore any cache effects.

(a) (10) How many bytes of memory would be read due to this instruction? In other words, if this instruction were removed from the program, how many fewer bytes would be read?

(b) (10) Same question as (a), but for writes.

**4.** (20) As you know, the linker will fix up provisional addresses in instructions for which the assembler did not have enough information to fully assemble. Give a complete list of instructions in the example program on p.3 of the PLN unit on machine language which are changed by the linker, and show what the new versions of those instructions will be. Assume that the **.data** segment begins at 0x6200 and the **.text** segment begins as 0x8400.

**5.** (25) Displayed below will be a tragically buggy program. Your job is to fix it. **PLEASE NOTE THE RULES:** Your answers MUST be of the form "Replace line x by y," where x is a line number (or a contiguous range of line numbers), and y consists of 0 or more lines. Here are sample answer forms:

- Replace one line by two:

```
Replace line 3 by

   movl $168, %eax
   addl $4, %ecx
```

- Delete three lines:

```
Replace lines 16-18 by
```

- Insert a line. Say line 12 has `decl %eax` and you want to insert **jz q** before it. Write:

```
Replace line 12 by

   jz q
   decl %eax
```

You are not allowed to use constructs not in our course material, nor may you use instructions from the MOVS family.

The goal of the code is to count capital letters in a null-terminated string. Here is the code:

```
 1        # count capitals in s
 2        .data
 3        s:  .string "aBC 23 DEF =- ["
 4        .text
 5        .globl _start
 6        _start:
 7              movl $0, %ebx
 8              movl $s, %ecx
 9        top:  movb (%ecx), %al
10              cmp 'A', %al
11              js next
12              cmp '[', %al
13              js next
14              incl %ebx
15        next: cmpb $0, %al
16              jnz done
17              jmp top
18        done: movl $0, %eax
```

**Solutions:**

**1.a** 111101

**1.b** -32 to +31

**2.** From the PLN, 56.12 is represented by the bytes 0xe1 0x7a 0x60 0x42, in order of ascending address. The pointer **p** will point to the 0x42 byte. Printed out in hex, that is 42; interpreted as a character, it is B; and interpreted as signed and unsigned 8-bit numbers, it is 66 in both cases.

**3.a** First, there is the instruction fetch. According to the PLN, this is a 2-byte instruction, so there are 2 bytes read there. Then we must read the destination in order to use it as a summand in the addition, so that is 4 more bytes, for a total of 6.

**3.b** The sum is written to the destination, for a total of 4 bytes.

**4.** We know from the PLN that the instruction on line 27 must be fixed. The same is true for line 33. Line 32 is not a problem, since the jump target is specified as a distance from the current PC value.

Now, what does line 27 change to? The label **x** is offset 0 in the **.data** segment, so it has address 0x6200. Checking the PLN, we see that the instruction has the form B9IMM4, where IMM4 will be set to the address of **x** by the linker. So, the instruction will be B900620000 (keeping endianness in mind).

As to line 27, this is a register-to-direct MOV. We don't have the format listed in our PLN, but we can deduce it. Since the provisional address set by the assembler was **sum**'s offset, 0010, we can see that the IMM4 portion of the instruction is the last 8 hex digits, 10000000. Adding 0010 to 0x6200, we get 0x6210 as the final address, and the instruction is 891D10620000.

**5.**

```
# count capitals in s

.data
s:  .string "aBC 23 DEF =- ["

.text
.globl _start
_start:
      movl $0, %ebx  # EBX will store the sum
      movl $s, %ecx  # ECX will point to the current char
top:  movb (%ecx), %al
      cmpb $'A', %al
      js next
      cmpb $'[', %al
      jns next
      incl %ebx
next: cmpb $0, %al
      jz done
      incl %ecx
      jmp top
done: movl $0, %eax
```