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.

1. (10) Almost any kind of CPU has a register which states where the current (or next, depending on the time) instruction is in memory. On most machines, this is called the _____, and on Intel CPUs it is called the _____.

2. (10) Suppose our word size is 8 bits. Then the number -4 is represented as _____ in the 2s complement system, and as _____ in the signed-magnitude system.

3. (10) Suppose the beginning of the **.data** segment in an assembly language source file is

```
.data
a: .long 2
b: .long 30
...
```

Show the exact line in the output of **as -a** for the line corresponding to the source line labeled **b**. (Assume the **.data** line is line 1 of the source file.)

4. (10) Look at page 3 of the PLN unit on machine language. Suppose on line 30, the instruction had been a jump to **done**. Show the machine language if the jump type is JNZ. Then show the machine language if the jump type is JMP.

5. (10) The C library function **bcopy()** would be better written in assembly language, as a big advantage could come from using instructions like _____. (The function **bcopy()** copies a sequence of bytes from one place in memory to another.)

6. (10) For each of the following instructions, state how many reads and writes of memory will occur. The period under consideration is step C. Your answers will consist of two numbers for each instruction; write your answer right next to the instruction.

(a) **movl %eax, %ebx**

(b) **addl %eax, %ebx**

(c) **addl (%eax), %ebx**

(d) **cmpl %eax, (%ebx)**

(e) **subl $8, (%ebx)**

(f) **subl %eax, (%ebx)**

(g) **cmpl $8, (%ebx)**

(h) **jnz y**

7. (10) Consider the instruction

```
movl $0xfff0eee0, %eax
```

Suppose we forget the **$**. Which one is most likely?

(i) The assembler will complain.

(ii) A seg fault will occur.

(iii) A different kind of execution error than seg fault will occur.

(iv) No execution errors will be reported, but the program's results will be incorrect.

(v) No errors will be reported of any kind, and the program's results will be correct.

8. The following code counts lower-case letters in the array **x**, placing the results in the byte array **counts**. E.g. the count of the number of occurrences of 'a' and 'e' will be at **counts** and **counts+4**.

```
.data
x:  .string "c92jemc82ne<824j8vcm92jq3.,.u"
counts:
    .rept 26
    .byte 0
    .endr
.text
.globl _start
_start:
    movl $x, %eax
top:
   movl $0, %ebx
   -------------------------
   cmpb $0, %bl
   jz done
   cmpb $'a', ______________
   js nextchar
   cmpb $'z'+1, ____________
   jge nextchar
   subl $'a',%ebx
   addl $____________, %ebx
   incb _________________
nextchar:
   addl $_________, %eax
   jmp top
done: movl %edx, %edx
```

(a) (10) Fill in the blanks.

(b) (5) State the full GDB command we would use to check whether the program executed correctly when we get to **done**.

(c) (5) This program will work correctly as long as no letter has a count of more than _____.

(d) (10) Show how we could replace the lines with **addl** and **incb** just before **nextchar** by a single instruction.

Solutions:

1. PC, EIP

2. 11111100, 10000100

3.

```
2  004  1e000000
```

4. 7503, EB03

5. MOVS

6. 0 0; 0 0; 1 0; 1 0; 1 1; 1 1; 1 0; 0 0

7. (ii)

8.a.

```
   # EAX will always point to the current character to be tallied
```

```
   movl $x, %eax
top:
   # need to zero out all of EBX for later use (see subl)
   movl $0, %ebx
   # get the character to be tallied
   movb (%eax), %bl
   # check for end of string
   cmpb $0, %bl
   jz done
   # check to see if in range 'a'-'z'
   cmpb $'a', %bl
   js nextchar
   cmpb $'z'+1, %bl
   jge nextchar
   # find distance past counts where we will increment
   subl $'a',%ebx
   # add that distance to counts to get address of place to increment
   addl $counts, %ebx
   # now increment
   incb (%ebx)
   # OK, ready to go to the next character in the string
nextchar:
   addl $1, %eax
   jmp top
done: movl %edx, %edx
```

8.b.

```
x/26b &counts
```

8.c. $2^8 - 1 = 255$

8.d.

```
incb counts(%ebx)
```