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.** (5) What is the hex (i.e. base-16) form of the decimal number 52?

**2.** (5) Fill in the blank: If you are using subroutines, you may not use the ______________ register for data storage.

**3.** The following questions are about the *revised* version of the program in Sec. 6 of our unit on Linux assembly language. Assume throughout that **x** begins in location 0x500.

(a) (5) What will be the hex address of the word in the **.data** segment whose initial value is 22?

(b) (5) After the instruction labeled **init** is executed, what value will be in EAX?

(c) (5) Which instruction will be executed after the **ret** in **init()**? (Write out the exact assembly language instruction here.)

(d) (5) What value will be in EBX at the time the first call to **swap()** is made?

(e) (10) With the data we have here, the array is correctly rearranged from (1, 5, 2, 18, 25, 22, 4) to (1, 2, 4, 5, 18, 22, 25). But suppose we had accidentally forgotten to put the parentheses in the instruction labeled **top**. What would happen? Would the program likely generate a seg fault? If so, explain how. If not, what would the final ordering be in the array?

**4.** (5) In the example of the speaker in Sec. 13.4 in our unit on Linux assembly language, why not use $0xc instead of $0xfc? Explain fully.

**5.** (10) Consider the C statement

```
int g = 1985229328;
```

Note that 1985229328 = $7 \times 16^7 + 6 \times 16^6 + 5 \times 16^5 + 4 \times 16^4 + 3 \times 16^3 + 2 \times 16^2 + 1 \times 16^1 + 0 \times 16^0$. Suppose &g = 0x234. What is the content of byte 0x236, assuming this is run on an Intel machine?

**6.** (20) Suppose we have written an assembly language program **z.s** and we run **as** on it with the **–gstabs** option, so that the assembler saves the symbol table. This produces a file **z.o**, which we link to create an executable file **z**. For each of the following, answer either Yes or No, indicating whether the indicated information is saved in the symbol table or not.

(i) The labels (i.e. names like **x**, **sum** and **top**) will be saved.

(ii) For labels in the **.data** segment, the types of the data (integer, character, etc.) will be saved.

(iii) For labels in the **.data** segment, the sizes in bytes of the data items will be saved.

(iv) For labels in the **.data** segment, the addresses (at least the offsets) of the labels will be saved.

**7.** (25) The following code might appear in an English/Chinese text editor:

```
.data
line: .space 12
nb: .long 12
neb:  .long 0

.text
.globl _start
_start:
   movl $0, %ebx
   movl $line, %eax
   movl nb, %ecx
```

```
top:
   movb (%eax), %dl
   ______ ______, %dl  # fill in operation and
                       # source operand
   jz x
   addl ______, %eax
   decl %ecx
   jmp next
x:
   incl %ebx
   ___________  # fill in one instruction here
next:
   decl %ecx
   jnz top
   movl ______, neb
```

The string of bytes which comprise a line in the file to be edited starts at **line**. The label **nb** is the name of a word which contains the number of bytes in the line.

The code is supposed to count the number of English characters in the line, and place that count in **neb**. Fill in the blanks.

**Solutions:**

**1.** 0x34

**2.** ESP

**3.a.** 0x514

**3.b.** 0x500

**3.c.**

```
movl (%eax), %ecx
```

**3.d.** 5

**3.e.** 2, 4, 22, 0x504, 0x508, 0x510, 0x50c

**4.** We need to preserve the original values of the other bits.

**5.** The number in **g** is 0x76543210. Since Intel is little-endian, that implies that 0x10 is in byte 0x234, 0x32 is in byte 0x235, 0x54 is in byte 0x236 and 0x76 is in byte 0x237.

**6.** (i) Yes. (ii) No. Remember, there are no types at the assembly language level. (iii) No. Similarly, there are no sizes associated with labels. A label is only a name for the given byte (or, equivalently, the given word). (iv) Yes. This is how we can do things like `x/7w &x` in GDB or DDD.

**7.**

```
.data
line: .space 12  # a line of characters from the file being edited
nb: .long 12  # length of "line" in bytes
neb:  .long 0  # in end, will contain number of English characters

.text
.globl _start
_start:

   # EBX will hold a count of English characters
   # EAX will always point to the current character in "line"
   # ECX will serve as a loop counter
   movl $0, %ebx
   movl $line, %eax
   movl nb, %ecx
top:
   movb (%eax), %dl # get byte from "line"
   # test for high bit being a 1
   andb $0x80, %dl
   jz x
   # this is the Chinese case
   addl $2, %eax
   decl %ecx
   jmp next
x:
   # English case
   incl %ebx
   incl %eax
next:
   decl %ecx
   jnz top
   movl %ebx, neb
```