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. (15) Fill in the blanks: In a timesharing OS, suppose we are transitioning from process A's turn to B's. Then the values in the _____ at the time A's turn ended are saved in _____ and on _____. The saving is performed by _____.

2. Look at page 70 of the Paul Carter book, Fig.4.6.

(a) (5) Give the numbers of the lines which an ENTER instruction would serve in place of.

(b) (10) Suppose our subroutine's local variables consist of 2 integers and 12 pointers to characters. What would the value of LOCAL_BYTES be?

3. Consider the subroutine addone() on page 7 of the handout on subroutines. Note that even though this subroutine is called from C, the subroutine itself was written directly in assembly language; the listing you see on page 7 was not generated by applying **gcc -S** to a C version of the subroutine.

(a) (5) The absence of which instruction(s) (note: instructions, not directives) tells us that this was written directly in assembly language, not C?

(b) (5) The lack of use of which piece of the Intel hardware tells us that a C version of this subroutine would have **void** as the type of its return value?

(c) (15) Show what code to add so that addone() has two arguments instead of one. Both arguments are pointers to words to be incremented; for example, the code

```
int x = 7, y = 12;
addone(&x,&y);
```

would result in x = 8 and y = 13. (Just show the extra instructions to add, not the entire revised subroutine.)

4. Consider the following source code:

```
 1  main(int argc, char **argv)
 2
 3  {  int x,n,i;
 4
 5     x = atoi(argv[1]);
 6     scanf("%d",&n);
 7     for (i = 0; i < n; i++)  {
 8        x = x * 2;
 9     }
10     printf("%d\n",x);
11     exit(1);
12  }
```

Only lines 5-11 are executable statements. We compile on CSIF.

(a) (10) List the numbers of the lines which will definitely result in this program relinquishing control of the CPU.

(b) (5) List the numbers of the lines during whose execution an interrupt (i.e. a pulse of current arriving at the CPU) from the timer may occur.

(c) (5) List the numbers of the lines during which their execution an interrupt from the keyboard may occur.

5. (10) In the instruction **movl 532(%ecx),%esp** on page 9 of the handout on OS, list all the contents of MAR and MDR during execution of the instruction. Ignore instruction fetch and cache effects. Assume that just before the **movl** is executed, c(ECX) = 424, c(EBX) = 626, c(ESP) = 828, c(424) = 2000, c(626) = 10000, c(956) = 1600, c(1600) = 2828, c(828) = 121212, c(1360) = 200, c(1158) = 16, where c( ) means "contents of." (You may not use all of, or even most of, this information).

6. Consider this code:

```
1 main()
2
3 {  int x[10],y[10],i; char c[80];
4
5    for (i = 0; i < 20; i++)
6       y[i] = 0x41414141;
7    strcpy(c,y);
8    i = strlen(c);
9 }
```

(a) (10) State at which line(s) — if any — a seg fault might occur. Explain IN DETAIL why the seg fault(s) might occur (or if you don't think any will occur, state IN DETAIL why not).

(b) (5) State IN DETAIL what possible values i may have after line 8, assuming no seg fault occurs anywhere in the program.

In both cases, assume that when a program occupies just part of a page, the portion of that page which is not occupied contains random garbage.

Solutions:

1. Registers; TSS; the stack; the OS.

2. 2-4; 56.

3.a.

```
pushl %ebp
movl %esp, %ebp
```

3.b. EAX.

3.c.

```
movl 12(%especially), %ebx
incl (%ebx)
```

4.a. 6, 10, 11.

4.b. 5-11.

4.c. 5-11.

5. MAR 956; MDR 1600.

6.a.

There is no problem in line 6, since the accesses to "y" will be to x, thus still in a page or pages which are allocated to this program.

But line 7 could be trouble. The function strcpy() looks for a null character. It won't find one in y or x, and so will keep going. If the random garbage in the remainder of the page has a 0 byte in it, then fine, but otherwise a seg fault will occur.

6.b.

80 + distance to the null byte, up to 4096.