COS 217 Fall

Fall 2006

Please write your answers clearly in the space provided. For partial credit, show all work. State all assumptions. You have exactly 3 hours for this exam. This final is open book, open notes. Put your name on every page. Write out and sign the Honor Code pledge just before turning in the test. "*I pledge my honor that I have not violated the Honor Code during this examination.*"

Question	Score
1	/20
2	/20
3	/20
4	/40
5	/30
6	/30
7	/40
Total	/200

Name:

Honor Code:

1 Representations

At the end of the execution of the following code segment,

```
unsigned char x = 0121;
unsigned char y = 0xF0;
unsigned char z;

z = x + y;
```

1. What does the variable `x` represent when interpreted as:
   (a) An ASCII character.

   (b) An 8-bit unsigned integer value (expressed in decimal).

   (c) An 8-bit signed integer value (expressed in decimal).

2. What does the variable `y` represent when interpreted as:
   (a) An 8-bit unsigned integer value (expressed in decimal).

   (b) An 8-bit signed integer value (expressed in decimal).

3. What does the variable `z` represent when interpreted as:
   (a) An ASCII character.

   (b) An 8-bit unsigned integer value (expressed in decimal).

   (c) An 8-bit signed integer value (expressed in decimal).

4. The variable `y` cannot be interpreted as an ASCII character. Why not?

2 Recursion

Consider the following program compiled and executed on a machine with 32-bit integers:

```
int foo(int n, int m)
{
  if (m <= 0) return 0;
  else return n + foo(n, m - 1);
}
```

1. What is the value of `foo(5, 3)`?

2. While executing `foo(5, 3)`, show the state of the stack (at the C-level is fine) just after the call to `foo(5, 0)`. Show the value of n and m in each record, and label each activation record with function and argument values that initiated it.

3. What does the function `foo` do? Be sure to precisely describe all limitations.

3 Short Answer

Answer the following in only the space provided.

1. What is a race condition?

2. Registers and memory are both examples of what?

3. What is the difference between and exception and an interrupt? Give an example of each.

4. Name two C statements which together are sufficient to implement any arbitrary control flow construct.

5. In a machine without a flag register (such as EFLAGS), how would you detect overflow after an unsigned addition?

6. Is x86 more CISC than RISC? Why?

7. Are fixed length instructions more CISC than RISC? Why?

8. Is the cosine instruction more CISC than RISC? Why?

9. A program is 5% sequential and 95% parallel without limit. What is the max speedup on an machine with an unlimited number of processors?

4 DFA

You are the lead engineer on project to design an automatic door of the type you might find at a supermarket. As the lead engineer, safety of the product is your primary concern.

This systems has two overhead motion detectors, one on the front (entry side) and one on the rear (exit side, the door swings open to the exit side). These sensors create a string with the following symbols: F (for front area occupied), R (for rear area occupied), B (for both occupied), and N (for neither occupied). This symbol is sent every millisecond.

The door has two states: Opened and Closed. (Assume that the door opens and closes instantaneously.)

For safe operation: If the door is closed, it should only open when if the rear is clear and the front is occupied (input symbol F). When the door is open, it should only close if neither the front or rear areas are occupied (input symbol N).

Design a DFA for the door controller which takes sensor input and determines the appropriate next state for the door. The door is initially closed.

For extra credit (15 pts), design a door controller for a door which doesn't open instantaneously. In such a case, the door has the following states: Open, Closed, Opening, Closing, Stopped Midway. This door also has another pair of input symbols to indicate if the door has reached fully open (O) or fully closed (C). This is used to determine when to go from Opening/Closing to the Open/Closed states. The area sensor state is always repeated after an O or C symbol is sent. For example, the input string would contain "FOF" if a customer steps into the front area and stays there until after the door is fully open. For safety, the door should stop swinging if the rear area is occupied at any time while in motion. **Warning: This is hard and potentially very tedious. It may not be worth the points. You make the call.**

5 Bug Hunt!

Bug hunt is back, and it's badder than ever! The following assembly program receives a character string from standard input, counts each alphabet letter's frequency in a case-insensitive way, and prints out the result to standard output. It assembles correctly but has a number of critical bugs that make it impossible to run the program flawlessly. Your task is to help the poor programmer exterminate the bugs. When describing a bug, please indicate in which line(s) the bug exists. One of the bugs is found in multiple lines in a similar pattern; count this as a single bug.

1. Describe a bug which almost always leads to a segmentation fault at run time.
   (Hint: Look for one between line 20 and 36 inclusively.)

2. Assuming that the first bug is fixed, describe another bug which almost always leads to a segmentation fault at run time.
   (Hint: Look for one between line 37 and 59 inclusively.)

3. Assuming that the first and second bugs are fixed, describe another bug which almost always leads to a segmentation fault at run time
   (Hint: Look for one between line 60 and 89 inclusively.)

4. Describe a bug which may lead to a segmentation fault at run time for some inputs.

5. Describe another bug which may lead to a segmentation fault at run time for some inputs.

```
1           .equ    ARRAYSIZE, 26
2           .equ    UINTSIZE, 4
3         ## ============================
4                   .section ".rodata"
5         ## ============================
6     cPrompt:
7           .asciz  "Input a string: "
8     cPrintFormat:
9           .asciz  "%c: %d\n"
10    cErrorFormat:
11          .asciz  "String too long!\n"
12        ## ============================
13                  .section ".bss"
14        ## ============================
15    uiFreq:
16          .skip ARRAYSIZE * UINTSIZE
17        ## ============================
18                  .section ".text"
19        ## ============================
20          .equ    IBUF, -4
21          .equ    II, -260
22          .equ    ISLEN, -264
23          .equ    ITC, -268
24          .globl main
25          .type main, @function
26    main:
27          pushl   %ebp
28          movl    %esp, %ebp
29        ## char buf[256];
30          subl    $256, %esp
31        ## unsigned int i = 0;
32          pushl   $0
33        ## unsigned int slen
34          subl    $4, %esp;
35        ## int tc;
36          subl    $4, %esp;
37        ## printf("Input a string: ");
38          pushl   $cPrompt
39          call    printf
40          addl    $4, %esp
41        ## gets(str);
42          pushl   IBUF(%ebp)
43          call    gets
44          addl    $4, %esp
45        ## slen = strlen(str);
46          pushl   IBUF(%ebp)
47          call    strlen
48          addl    $4, %esp
49          movl    %eax, ISLEN(%ebp)
50        ## if (slen <= 255) goto endcheck;
51          cmpl    $255, ISLEN(%ebp)
52          jle     endcheck
53        ## printf("String too long!\n");
54          pushl   $cErrorFormat
55          call    printf
56          addl    $4, %esp
57        ## return 0;
58          jmp     endloop2
59    endcheck:
60        ## i = 0;
61          movl    $0, II(%ebp)
62    loop1:
63        ## if (i >= slen) goto endloop1;
64          movl    II(%ebp), %eax
65          cmpl    ISLEN(%ebp), %eax
66          jge     endloop1
67        ## tc = str[i] - 'a'
68          movl    II(%ebp), %eax
69          movl    IBUF(%ebp,%eax,1), %ecx
70          subl    $97, %ecx
71          movl    %ecx, ITC(%ebp)
72        ## if (tc >= 0) goto endif1;
73          cmpl    $0, ITC(%ebp)
74          jge     endif1
75        ## tc += 'a'-'A';
76          movl    $97, %eax
77          subl    $65, %eax
78          addl    %eax, ITC(%ebp)
79    endif1:
80        ## uiFreq[tc]++;
81          movl    ITC(%ebp), %eax
82          movl    $1, %ecx
83          addl    %ecx, uiFreq(,%eax,4)
84    endif2:
85        ## i++;
86          incl    II(%ebp)
87        ## goto loop1;
88          jmp     loop1
89    endloop1:
90        ## i = 0;
91          movl    $0, II(%ebp)
92    loop2:
93        ## if (i >= ARRAYSIZE) goto endloop2;
94          cmpl    $ARRAYSIZE, II(%ebp)
95          jge     endloop2
96        ## printf("%c: %d\n", i+'a', uiFreq[i]);
97          movl    II(%ebp), %eax
98          pushl   uiFreq(,%eax,4)
99          addl    $97, %eax
100         pushl   %eax
101         pushl   $cPrintFormat
102         call    printf
103         addl    $12, %esp
104       ## i++;
105         incl    II(%ebp)
106       ## goto loop2;
107         jmp     loop2
108   endloop2:
109       ## return 0;
110         movl    $0, %eax
111         movl    %ebp, %esp
112         popl    %ebp
113         ret
```

6 Portability

For each part below, write a piece of C code that works in one case, but not the other. Indicate the desired result for the code, the result expected in the case that works.

1. Works for big endian, but not little endian.

2. Works for 16-bit integers, but not 32-bit integers.

3. Works for Linux, but not Windows.

4. Works for ASCII, but not EBCDIC.

7 Signals and Alarms

The following (incomplete) program is intended to measure the computing speed of a processor by counting the number of times a simple loop iterates. To do so, this program should work as follows:

- Once the user hits CTRL-C, the counter `count` should be reset, and a SIGALRM should be set for delivery in 1 second.
- During this 1-second period, the program should ignore any CTRL-C keystrokes.
- After 1 second, the program should print the number of iterations executed by calling the `print_message()` function. In addition, after calling this function, the program should re-enable CTRL-C so that the user can run the experiment again (without terminating the program), i.e. hitting CTRL-C should restart the measurement.
- Assume that this program will be compiled on the Hats Linux cluster with the command line:
  `gcc -Wall speed.c -o speed`
  Remember that, by not specifying the `-ansi` option, signal handlers are not reset after executing (i.e. the current handler is not uninstalled).
- You can assume that neither SIGINT nor SIGALRM will be blocked at the start of the program.
- Your code should be well-styled, but need not include comments.

1. Complete the code below so that it works according to the specification above. You can add code to existing functions, as well as add new functions.

```
#include <stdio.h>
#include <signal.h>
#include <assert.h>
#include <unistd.h>

static int count;

static void print_message(void) {
  printf("\nIterated %d times.\n", count);
  printf("Press CTRL-C to restart timer...\n");
}

int main() {

  printf("Press CTRL-C to start timer...\n");

  while (1)
    count++;
  return 0;
}
```

2. Describe a way by which the user can terminate your program (without rebooting the OS/computer :-)).

3. Would it be more appropriate to use SIGPROF instead of SIGALRM to achieve this application's purpose? Justify your answer.