— 1E

UCLA Computer Science 33 (Fall 2015)
Midterm 1
99 points, 99 minutes, open book, open notes.
Questions are equally weighted (11 min. each).
Use a separate sheet of paper for each answer.
Put a big problem number at each sheet's top.
Turn in your sheets in increasing numeric order.

Name:_____ Student ID:_____

1	2	3	4	5	6	7	8	9	sum
11	6	10	11	6	8	4	9	6	71

1 (11 minutes). You want to create a repeated bit pattern in a 64-bit unsigned word. The pattern repeats every 8 bits. For example, repeating the bit-pattern 10011011 would yield the word 0x9b9b9b9b9b9b9b9b. Write a C function rbp(p) that returns such a word, given an 8-bit pattern p. Have your function execute as few instructions as possible.

2 (11 minutes). The PDP-11 architecture is "mixed-endian": within a 16-bit short word, the least significant byte comes first, whereas within a 32-bit long word, the *most* significant short word comes first. Diagram how the signed 32-bit number -25306982 (-0x1822766) is represented as a series of unsigned 8-bit bytes (a) on a PDP-11, (b) on an x86-64 machine, and (c) on a bigendian machine like the SPARC. Your diagram should list the offset of each byte.

3 (11 minutes). Consider these two functions:

```
#include <stdbool.h>
bool pushme (unsigned long v) {
  return 255 <= (v >> 3);
}
bool pullyou (long v) {
  return ! (0 <= (v >> 3) && (v >> 3) < 255);
}
```

and this assembly-language implementation:

```
pushme: cmpq    $2039, %rdi
        seta    %al
        ret
pullyou:cmpq    $2039, %rdi
        seta    %al
        ret
```

a. Explain why those "2039"s are correct, even though the source code does not mention 2039.

b. How can pushme and pullyou have identical machine code, even though the functions have different types and implementations? Explain.

4 (11 minutes). Would the following be a valid implementation of (3)'s pushme and pullyou functions? If not, explain why not. If so, give another implementation of pushme and pullyou that would be even shorter (i.e., would take fewer bytes of machine code).

```
pushme: cmpq    $2039, %rdi
        seta    %al
        ret
pullyou:jmp     pushme
```

5 (11 minutes). The following is a buggy implementation of (3)'s pushme function. Three of its instructions are incorrect. Fix the bugs with as few changes as you can and briefly explain why your fixes are needed.

```
pushme: pushq   %rbx
        movq    %rsp, %rbp
        movq    %rdi, -8(%rbp)
        movq    -8(%rbp), %rax
        shrq    $3, %rax
        cmpq    $255, %rax
        seta    %al
        popq    %rbx
        ret
```

6 (11 minutes). Explain what the following assembly-language function does, at a high level. Give C source code that corresponds to its behavior as closely as possible.

```
mystery: movzbl %dil,%eax
         movabs $0x101010101010101,%rdx
         imul   %rdx,%rax
         retq
```

7 (11 minutes). What does the following assembly-language code do? Briefly explain how to use it from C source code, how it executes, and what its behavior is from the C point of view.

```
callme: leaq    (%rdi,%rsi), %rax
        callq   .L1
.L1:    ret
```

8 (11 minutes). Consider the following C code:

```
 1  #include <stdio.h>
 2  #include <string.h>
 3
 4  long n = 3;
 5  extern int (*p) (void);
 6
 7  void
 8  output (int n)
 9  {
10    printf ("0x%x\n", n);
11  }
12
13  int
14  badfun (void)
15  {
16    int i;
17    memcpy (&i + 3, &p, sizeof p);
18    output (*(&i + n));
19    return i;
20  }
21
22  int
23  main (void)
24  {
25    return ! badfun ();
26  }
27
28  int (*p) (void) = main;
```

and the following machine code generated for two of its functions, in GDB disassembly format:

```
Dump of assembler code for function badfun:
   0x400550 <+0>:  sub    $0x18,%rsp
   0x400554 <+4>:  mov    0x200ae5(%rip),%rax
    # 0x601040 <p>
   0x40055b <+11>: mov    %rax,0x18(%rsp)
   0x400560 <+16>: mov    0x200ae1(%rip),%rax
    # 0x601048 <n>
   0x400567 <+23>: mov    0xc(%rsp,%rax,4),%edi
   0x40056b <+27>: callq  0x400540 <output>
   0x400570 <+32>: mov    0xc(%rsp),%eax
   0x400574 <+36>: add    $0x18,%rsp
   0x400578 <+40>: retq
Dump of assembler code for function main:
   0x400430 <+0>:  sub    $0x8,%rsp
   0x400434 <+4>:  callq  0x400550 <badfun>
   0x400439 <+9>:  test   %eax,%eax
   0x40043b <+11>: sete   %al
   0x40043e <+14>: add    $0x8,%rsp
   0x400442 <+18>: movzbl %al,%eax
   0x400445 <+21>: retq
```

For each instruction in the machine code, identify the corresponding source-code line number. If an instruction corresponds to two or more source-code line numbers, write them all down and explain.

9 (11 minutes). When (8)'s program is run it outputs about a million lines of text and then dumps core with a segmentation fault. Explain why this happens, in as much detail as you can. Your explanation should include what those text lines look like, and why.