Practice Exam 2

November 9, 2019

1. (a) If register %eax holds a pointer to an integer, then executing the cmd p/x $eax in gdb prints out the address of the integer.

   (b) The pushfl places the condition codes into the %ebx register.

   (c) xorl %eax,%eax sets %eax to 1 if %ebx is odd.

   (d) The %ebp register points to the current stack frame.

   (e) Assembly programmer's view of memory is equivalent to a linear array of bytes in C.

   (f) %ah register is the higher 8-bits of %eax register.

   (g) Are the following operations legal: mov 0x80(%eax,%ebx,%ecx),%edx.

2. Assume the following values are stored at the indicated memory addresses and registers.

Address	Value
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x11

Register	Value
%eax	0x100
%ecx	0x1
%edx	0x3

Fill in the following table showing the effects of the following instructions, in terms of both the register or memory location that will be updated and the resulting value. If the destination is a memory location, provide the effective address. If the destination is a register, provide the register name.

Instruction	Destination	Value
addl %ecx, (%eax)		
subl %edx, 4(%eax)		
imull $16, (%eax, %edx, 4)		
incl 8(%eax)		
decl %ecx		
subl %edx, %eax		

3. Write the final value of %ebx for each of the following code snippets.

   (a)

   ```
   movl $3, %ebx
   leal (%ebx,%ebx,2), %ebx
   ```

   (b)

   ```
   movl $0x123, %ebx
   shrl $8, %ebx
   leal 1(%ebx,%ebx,4), %ebx
   ```

   (c)

   ```
   xorl %ebx,%ebx
   leal 1(%ebx), %ebx
   sall $3, %ebx
   leal 2(%ebx,%ebx,8), %ebx
   ```

4. Consider the following C program

   ```
   #include <stdio.h>
   int p (int a, int b, int c) {
       int d, e, f;

       f = 0;
       for (d=a; d<b; d++)
               for (e=b; e<c; e++)
                       f += d + e;
       return f;
   }
   ```

   When compiled with a high level of optimization, it looks like this (with line numbers added for reference):

   ```
   01 p:
   02   pushl    %ebp
   ```

```
03   movl    %esp, %ebp
04   pushl   %edi
05   pushl   %esi
06   movl    12(%ebp), %edi
07   movl    8(%ebp), %ecx
08   pushl   %ebx
09   xorl    %ebx, %ebx
10   cmpl    %edi, %ecx
11   movl    16(%ebp), %esi
12   jge      .L28
13  .L26:
14   cmpl    %esi, %edi
15   movl    %edi, %edx
16   jge      .L30
17  .L25:
18   leal    (%edx,%ecx), %eax
19   incl    %edx
20   addl    %eax, %ebx
21   cmpl    %esi, %edx
22   jl       .L25
23  .L30:
24   incl    %ecx
25   cmpl    %edi, %ecx
26   jl       .L26
27  .L28:
28   movl    %ebx, %eax
29   popl    %ebx
30   popl    %esi
31   popl    %edi
32   movl    %ebp,%esp
33   popl    %ebp
34   ret
```

(a) Write the name of the register and the offset from %ebp corresponding to each of a, b, and c.

(b) Write the name of the register corresponding to each one of d,e, f, and the return value.

(c) Which line number corresponds to d = a;?

(d) Which line number is the last of the assembly statements that implements f += d + e;?

(e) If line 32 were removed, would the function still work properly? Why or why not?

5. As with the bomblab, you have to devise the inputs to this program. There are multiple inputs that solve this phase named foo. Identify all the inputs that would defuse this phase. The function explode bomb has the same behavior as in bomblab. The function sscanf has the following prototype:

```
int sscanf(const char *str, const char *format, ...);
```

sscanf reads its input from the character string pointed to by str. It returns the number of input items successfully matched to the format and assigned. An example usage is

```
sscanf(ptr, "%d␣%d␣%d", &a, &b, &c);
```

The function prototype of the phase is as follows:

```
void foo(char* input);
```

Further, the bomblab designer has ensured that this phase can indeed be diffused without requiring gdb. To help the students the bomblab designer has also annotated the assembly code.

```
.LC0:
    .string "%d %c\n"
        .text
.globl foo
    .type foo, @function
foo:
    pushl %ebp
    movl %esp, %ebp
    subl $40, %esp
    leal -13(%ebp), %eax
    movl %eax, 12(%esp)
    leal -12(%ebp), %eax
    movl %eax, 8(%esp)
    movl $.LC0, 4(%esp)
    movl 8(%ebp), %eax
    movl %eax, (%esp)
    call sscanf
    movl -12(%ebp), %eax
    testl %eax, %eax
    je .L3
    cmpl $1, %eax
    jne .L7
    jmp .L8
.L3:
    movl ptr1, %eax
    movzbl 3(%eax), %eax
    cmpb -13(%ebp), %al
    je .L6
    call explode_bomb
    jmp .L6
.L8:
    movl ptr2, %eax
    movzbl 2(%eax), %eax
    cmpb -13(%ebp), %al
    je .L6
    call explode_bomb
    jmp .L6
.L7:
    call explode_bomb
.L6:
    leave
    ret
.globl ptr1
    .section     .rodata.str1.1
.LC1:
    .string ”cs214”
    .data
    .align 4
    .type ptr1, @object
    .size ptr1, 4
ptr1:
    .long    .LC1
.globl ptr2
    .section     .rodata.str1.1
LC2:
    .string ”ee365”
    .data
    .align 4
    .type ptr2, @object
    .size ptr2, 4
ptr2:
    .long .LC2
```

(a) How many inputs does the phase take? What are their types?

(b) How many global pointers are present in this code? What are those? What do they point to?

(c) How many inputs diffuse this phase? How did you deduce it?

(d) Enumerate the inputs that diffuse the phase?