CS 261 – Fall 2021 – Lab6 during Week 7

Complete individually or in groups of 2.

In this week's lab, we will reverse engineer a switch statement from machine code. In the following procedure, the body of the switch statement has been removed:

```
long switch_lab(long x, long n)
{
        long result = x;
        switch(n) {
        /* The code for the cases goes here */
        }
        printf("%ld", result);
}
```

The compiler ensures efficient implementation of the switch statement using a data structure called the *jump table*. The jump table can be thought of as an array that stores an offset which indicates which instruction to move to next as one single address calculation.

For example, say we have specified cases 100, 103 and default case in a switch statement.

1. The compiler first subtracts 100 from the switch argument [variable n in the switch statement above] to bring the range of cases between 0 to 3 (from 100 to 103) [let's call this R].
2. Next, we check if R is greater than 3, in which case we jump to the address for the default case (instruction ja, short for jump if above).
3. Otherwise, the compiler uses the jump table with the value of R as an index into that array. It takes the address of the jump table and adds the appropriate offset from the jump table to obtain the start address of instructions of the applicable case.

Our jump table may look like this, for example:

0x2008: -108 -100 -100 -104

These 4 values are respectively the offsets corresponding to case values 100, 101, 102 and 103. Thus, instead of multiple comparisons, the compiler can use the value of R (0,1, 2, 3, or 4+) as index into the above array. Notice that we had not specified case 101 or 102, which implies execution should proceed to the default case. We can indeed see that the values at index 1 and 2 in the jump table are same, which is the offset for the default case.

The disassembled machine code for the function switch_lab is shown below. The jump table resides in a different area of memory. We can see from the lea instruction at stack address 0x0000000000001187 that the jump table begins at address 0x2008. Using gdb, we can examine the eight 4-byte words of memory comprising the jump table below, where the most common offset usually corresponds to the default case. The command is x/8wx:

```
(gdb) x/8wx 0x2008
0x2008: 0xffffff18c      0xffffff193      0xffffff199      0xffffff1b7
0x2018: 0xffffff1b7      0xffffff19e      0xffffff1b7      0xffffff1a4
```

```
Dump of assembler code for function switch_lab(long, long):
   0x0000000000001149 <+0>:     endbr64
   0x000000000000114d <+4>:     push   %rbp
   0x000000000000114e <+5>:     mov    %rsp,%rbp
   0x0000000000001151 <+8>:     sub    $0x20,%rsp
   0x0000000000001155 <+12>:    mov    %rdi,-0x18(%rbp)
   0x0000000000001159 <+16>:    mov    %rsi,-0x20(%rbp)
   0x000000000000115d <+20>:    mov    -0x18(%rbp),%rax
   0x0000000000001161 <+24>:    mov    %rax,-0x8(%rbp)
   0x0000000000001165 <+28>:    mov    -0x20(%rbp),%rax
   0x0000000000001169 <+32>:    sub    $0x46,%rax
   0x000000000000116d <+36>:    cmp    $0x7,%rax
   0x0000000000001171 <+40>:    ja     0x11bf <switch_lab(long, long)+118>
   0x0000000000001173 <+42>:    lea    0x0(,%rax,4),%rdx
   0x000000000000117b <+50>:    lea    0xe86(%rip),%rax        # 0x2008
   0x0000000000001182 <+57>:    mov    (%rdx,%rax,1),%eax
   0x0000000000001185 <+60>:    cltq
   0x0000000000001187 <+62>:    lea    0xe7a(%rip),%rdx        # 0x2008
   0x000000000000118e <+69>:    add    %rdx,%rax
   0x0000000000001191 <+72>:    notrack jmpq *%rax
   0x0000000000001194 <+75>:    addq   $0x2c,-0x8(%rbp)
   0x0000000000001199 <+80>:    jmp    0x11c7 <switch_lab(long, long)+126>
   0x000000000000119b <+82>:    shlq   -0x8(%rbp)
   0x000000000000119f <+86>:    jmp    0x11c7 <switch_lab(long, long)+126>
   0x00000000000011a1 <+88>:    addq   $0xc,-0x8(%rbp)
   0x00000000000011a6 <+93>:    shlq   -0x8(%rbp)
   0x00000000000011aa <+97>:    jmp    0x11c7 <switch_lab(long, long)+126>
   0x00000000000011ac <+99>:    mov    -0x8(%rbp),%rdx
   0x00000000000011b0 <+103>:   mov    %rdx,%rax
   0x00000000000011b3 <+106>:   add    %rax,%rax
   0x00000000000011b6 <+109>:   add    %rdx,%rax
   0x00000000000011b9 <+112>:   mov    %rax,-0x8(%rbp)
   0x00000000000011bd <+116>:   jmp    0x11c7 <switch_lab(long, long)+126>
   0x00000000000011bf <+118>:   movq   $0x5,-0x8(%rbp)
   0x00000000000011c7 <+126>:   mov    -0x8(%rbp),%rax
   0x00000000000011cb <+130>:   mov    %rax,%rsi
   0x00000000000011ce <+133>:   lea    0xe2f(%rip),%rdi        # 0x2004
   0x00000000000011d5 <+140>:   mov    $0x0,%eax
   0x00000000000011da <+145>:   callq  0x1050 <printf@plt>
   0x00000000000011df <+150>:   nop
   0x00000000000011e0 <+151>:   leaveq
   0x00000000000011e1 <+152>:   retq
```

Given the C code and information above answer the following questions about the switch statement: (15 points each question, **please submit your answers to Gradescope Lab 6 during week 7**).

1) Given the provided jump table offset, which offset corresponds to the default case?

2)How many cases are specified in the switch statement (not counting the default case)?

3)How much is subtracted from the switch argument?

4) What is the stack address of where the default case is?

5) Which offset from the jump table is used to calculate start address for the largest integral value?

6) What is the case value that falls through to the other case value?