21

g. CSE 6th Semester

05 May 2023

ISLAMIC UNIVERSITY OF TECHNOLOGY (IUT)

ORGANISATION OF ISLAMIC COOPERATION (OIC)

Department of Computer Science and Engineering (CSE)

SEMESTER FINAL EXAMINATION **SUMMER SEMESTER, 2021-2022**

DURATION: 3 HOURS **FULL MARKS: 150**

CSE 4649: Systems Programming

Programmable calculators are not allowed. Do not write anything on the question paper.

Answer **all 6 (six)** questions. Figures in the right margin indicate full marks of questions whereas corresponding CO and PO are written within parentheses.

[**For all the questions, assume 64-bit system unless otherwise mentioned**.]

1. a) Consider a C program given in Code Snippet 1. 7 (CO2) (PO2)

```
void fun_test(char *x, char *y){
        if (&x[0] < &y[4])
                printf("Trick!");
        else
                printf("Treat!");
}
int main(){
        char str[] = "Good Luck in your Exam :)";
        char *x = str;
        char *y = str;
        fun_test(x, y);
}
```

Code Snippet 1: Code Snippet for Question 1.a)

What will the above program print out and why?

b) In computer programming, a local variable that is assigned to some value but is not read by any subsequent instruction is referred to as a *dead store*. Optimizing compilers optimize code by removing dead stores in a program. The following C program in Code Snippet 2 contains one such dead store. When this program is compiled with an optimizing compiler, the elimination of dead store will introduce a security loophole. 10 (CO3) (PO4)

Find out the dead store in the program and explain how the security loophole can occur.

```
// function prototype for checking password
int check_pass(char *pass){}

int main(){
        char pwd[20];
        fgets(pwd, 20, stdin);
        if (check_pass(pwd))
                printf("You're logged in!");
        else
                printf("Incorrect password!");
        memset(pwd, 0, sizeof(pwd));
}
```

Code Snippet 2: Code Snippet for Question 1.b)

c) Explain the concept of Cycles Per Element (CPE) for expressing program performance with appropriate example. 8 (CO1) (PO1)

2. a) Consider the following C switch skeleton in Code Snippet 3 and the corresponding x86-64 assembly code with jump table.

```c
long fun_switch(long x, long y)
{
  long result = _____;
  switch(result) {
    case ____:
         _____;
    case ____:
         _____;
    case ____:
         _____;
      break;
    case ____:
         _____;
      break;
    case ____:
         _____;
    default:
         _____;
  }
  return result;
}
```

```
fun_switch:
# 'result' in %rax
   leaq    (%rdi,%rsi), %rax
   movq    %rax, (%rsp)
   leaq    -63(%rax), %rdx
   cmpq    $6, %rdx
   ja      .L2
   jmp     *.L4(,%rdx,8)
.L3:
   sarq    $2, %rax
   movq    %rax, (%rsp)
.L5:
   addq    (%rsp), %rsi
   addq    $7, %rsi
   movq    %rsi, (%rsp)
.L6:
   movq    (%rsp), %rax
   subq    %rdi, %rax
   movq    %rax, (%rsp)
   jmp     .L9
.L7:
   addq    $105, %rax
   movq    %rax, (%rsp)
   jmp     .L9
.L2:
   movq    %rsp, (%rsp)
.L9:
   movq    (%rsp), %rax
   ret
```

```
.L4:
    .quad   .L3
    .quad   .L5
    .quad   .L2
    .quad   .L6
    .quad   .L7
    .quad   .L2
    .quad   .L2
```

Code Snippet 3: Switch program for Question 2.a)

i. What are the values for the case labels in the switch statement? Complete the C source code. 3+1 (CO2 (PO4

ii. Explain how *switch* statements are more efficient than *if-else* blocks. Mention the requirements those need to be satisfied for efficient switch implementation in machine level. (CO (PO

b) State whether the following statements are correct or incorrect. 1 × (CO (PO

i. XOR instruction can be used to zero-out a register.

ii. If p is an integer pointer (int *) and p has the value 0 then after the expression `(p && p++)` is evaluated, p will have 1.

iii. In C, if one operand's type is unsigned short and another operand's type is signed short, then both of them will be converted to unsigned short before doing an arithmetic operation.

iv. Callee-saved registers can be safely used by caller function after execution returns to it.

v. Optimization by compilers is done at run-time.

3. a) What will be the output of the program in Code Snippet 4 assuming x, y and z have memory addresses 100, 200 and 300 respectively. (CC (PC

```c
int main() {
    int x = 1569;
    int *y = &x;
    int *z = (int *) x;
    printf("%d,%d,%d\n", (int) x, (int) &x, (int) (x+1));
    printf("%d,%d,%d\n", (int) y, (int) &y, (int) (y+1));
    printf("%d,%d,%d\n", (int) z, (int) &z, (int) (z+1));
}
```

Code Snippet 4: C program for Question 3.a)

Mention and explain the steps you would follow to design and implement an efficient and optimized system that can scale as well as perform well. 7 (CO3) (PO3)

c) Explain two types of optimization blockers with appropriate code example in C. 9 (CO1) (PO1)

4. a) The x86-64 assembly instructions of three functions are given in Code Snippet 5. Function signatures for the functions are `int fun_1(int x)`, `int fun_2(int n)` and `int main()`. 18 (CO2) (PO2)

```
4004e0 <fun_1>:
  4004e0: push  %rbp
  4004e4: mov   $0x7,%eax
  4004ef: mov   %eax,-0x8(%rbp)
  4004f2: mov   %edi,%eax
  4004f4: cltd
  4004f5: mov   -0x8(%rbp),%edi
  4004f8: idiv  %edi
  4004fa: pop   %rbp
=>4004fb: retq
```

```
400500 <fun_2>:
  400500: push  %rbp
  400508: mov   %edi,-0x4(%rbp)
  40050b: imul  $0x69,-0x4(%rbp),%edi
=>40050f: callq 4004e0 <fun_1>
  400518: pop   %rbp
=>400519: retq

400520 <main>:
  400520: push  %rbp
  400528: mov   $0x45,%edi
=>400534: callq 400500 <fun_2> ; START
  40053d: pop   %rbp
=>40053e: retq
```

Code Snippet 5: x86-64 assembly for Question 4.a)

Assuming a 64-bit system, write down the execution trace *before* and *after* each `call` and `ret` (marked with "=>") instruction according to the Table 1. First one before `call` in `main` is done for you. Note that, all values are in hexadecimal.

Table 1: Sample execution trace for Question 4.a)

	State Values			
%rip	%rdi	%rax	%rsp	*%rsp
0x400534	0x45	-	0x7fffffffe858	-

b) Suppose you're asked to design the function parameter passing convention of a newly proposed ISA i69. Would you use registers or stack or a combination of both? Provide justification in favor of your choice. 7 (CO3) (PO3)

5. a) Consider the C code given in Code Snippet 6.

```
void take_input(char *buf, unsigned int len){
        fgets(buf, len, stdin);
}
int main(){
        char buf[0x69];
        int n, max_len = 0x69;
        scanf("%d", &n);
        if (n < max_len)      take_input(buf, n);
        else                  printf("Not enough space! \n");
}
```

Code Snippet 6: Vulnerable Code Snippet for Question 5.a)

i. The above code is vulnerable to buffer overflow which lets anyone copy beyond the allocated space of the `buf` variable. Explain the reason of buffer overflow with a concrete input example. 8 (CO3) (PO4)

ii. Rewrite the relevant portion of code to mitigate the vulnerability. Explain how your fix is solving the problem. 4 (CO3) (PO3)

b) Consider the C code given in Code Snippet 7.

```
int main(){
        char str[69];
        fgets(str, 69, stdin);
}
```

Code Snippet 7: Code Snippet for Question 5.b)

Suppose the starting address of the `str` variable is stored in `%rsp` where `%rsp=0x7fffffffe869`. A user gives "**HelloWorld**" as input. Now, show how this string would be stored in memory in both *Little Endian* and *Big Endian* machines. Show memory address of every character for both types of machines.

c) How Sign Flag (SF) and Overflow Flag (OF) are used together to detect signed overflow. Explain with concrete examples. 8 (CO1) (PO1)

6. a) The x86-64 assembly instructions of a recursive function `int fun (int n)` is given in Code Snippet 8.

```
fun:
    cmpl    $1, %edi
    jle     .L3
    pushq   %rbp
    pushq   %rbx
    subq    $8, %rsp
    movl    %edi, %ebx
    leal    -1(%rdi), %edi
    call    fun
    movl    %eax, %ebp
    leal    -2(%rbx), %edi
    call    fun
    addl    %ebp, %eax
    jmp     .L2
```

```
.L3:
    movl    %edi, %eax
    ret
.L2:
    addq    $8, %rsp
    popq    %rbx
    popq    %rbp
    ret
```

Code Snippet 8: x86-64 assembly for Question 6.a)

i. Explain how the notion of callee-saved registers is being used for ensuring correct and expected execution of function `fun()`. 6 (CO1) (PO2)

ii. Write the equivalent C source code for the instructions in Code Snippet 8. 9 (CO2) (PO2)

b) Write down the sequence of x86-64 assembly instructions for the C code given in Code Snippet 9. 10 (CO2) (PO2)

```
void fun_arr(int arr[], int i, int *sum)
{
    *sum = 0;
    for (; i < 0; i--)
    {
        *sum += arr[i];
    }
}
```

Code Snippet 9: Code Snippet for Question 6.b)