

ICS Homework 7

November 12, 2019

1 Function Matching

Functions given below have same functionality (assume overflow will never happen). The functions' generated assembly codes are also given, but are misordered. Place pair the functions and the assembly codes and fill in the table. For function inputs, assume x in %edi, y in %esi, z in %edx. For local variable, assume i in -0x4(%rbp).

```
1  int A(int x, int y, int z) {
2      int i = 0;
3      return (x>0)?y + x * z : y;
4  }
5
6  int B(int x, int y, int z) {
7      int i = 0;
8      while (x > 0) {
9          x--;
10         y+= z;
11     }
12     return y;
13 }
14
15 int C(int x, int y, int z) {
16     int i = 0;
17     do {
18         x--;
19         if (x < 0) break;
20         y += z;
21     } while (1);
22     return y;
23 }
24
25 int D(int x, int y, int z) {
26     int i = 0;
27     for (; i < x; i++) {
28         y += z;
29     }
30     return y;
31 }
```

```

1  asm_1:
2      movl    $0x0,-0x4(%rbp)
3  .L1:
4      subl    $0x1,%edi
5      cmpl    $0x0,%edi
6      js      .L2
7      add     %edx,%esi
8      jmp     .L1
9  .L2:
10     mov     %esi,%eax
11     ret
12
13  asm_2:
14     movl    $0x0,-0x4(%rbp)
15     cmpl    $0x0,%edi
16     jle     .L3
17     imul    %edx,%edi
18     mov     %esi,%eax
19     add     %edi,%eax
20     jmp     .L4
21  .L3:
22     mov     %esi,%eax
23  .L4:
24     ret
25
26  asm_3:
27     movl    $0x0,-0x4(%rbp)
28     jmp     .L6
29  .L5:
30     add     %edx,%esi
31     addl    $0x1,-0x4(%rbp)
32  .L6:
33     mov     -0x4(%rbp),%eax
34     cmp     %edi,%eax
35     jl      .L5
36     mov     %esi,%eax
37     ret
38
39  asm_4:
40     movl    $0x0,-0x4(%rbp)
41     jmp     .L8
42  .L7:
43     subl    $0x1,%edi
44     add     %edx,%esi
45  .L8:

```

```

46      cmpl    $0x0,%edi
47      jg      .L7
48      mov     %esi,%eax
49      ret

```

Assembly Code	Function Name
asm_1	C
asm_2	A
asm_3	D
asm_4	B

2 Conditional Move

The generated assembly code of function A in Q1 use jump operations. Please use conditional move operations instead to achieve the same functionality.

```

1  A:
2      movl    $0x0,-0x4(%rbp)
3      mov     %edi,%ebx
4      imul    %edx,%ebx
5      add     %esi,%ebx
6      // fill in your assembly code here
7      mov     %esi,%eax
8      cmpl    $0x0,%edi
9      cmovg   %ebx,%eax
10     ret

```

3 Jump Table

Read the assembly code and jump table given below, Fill in the missing part of the C code.

For function inputs, assume x in %edi, y in %esi, z in %edx. For local variable, assume result in -0x4(%rbp).

```

1  int switcher(int x, int y, int z) {
2      int result = 0;
3      switch (x) {
4          (1) case 17:
5              result = x - y;
6              break;
7          (2) case 19: case 21:

```

```

8         result = (3)z;
9     (4) case 18:
10         result = (5)result + x;
11         break;
12     (6) case 20: case 23:
13         result = y;
14         break;
15     default:
16         result = (7)result + 20;
17         break;
18 }
19 return result;
20 }

```

```

1  switcher:
2      movl    $0x0,-0x4(%rbp)
3      mov     %edi,%eax
4      sub     $0x11,%eax
5      cmp     $0x6,%eax
6      ja      .L5
7      mov     %eax,%eax
8      jmp     *.L6(,%rax,8)
9  .L1:
10     mov     %edx,-0x4(%rbp)
11  .L2:
12     add     %edi,-0x4(%rbp)
13     jmp     .L7
14  .L3:
15     mov     %edi,%eax
16     sub     %esi,%eax
17     mov     %eax,-0x4(%rbp)
18     jmp     .L7
19  .L4:
20     mov     %esi,-0x4(%rbp)
21     jmp     .L7
22  .L5:
23     addl    0x14,-0x4(%rbp)
24  .L7:
25     mov     -0x4(%rbp),%eax
26     ret

```

```

1  .L6:
2      .quad   .L3
3      .quad   .L2

```

4	. quad	. L1
5	. quad	. L4
6	. quad	. L1
7	. quad	. L5
8	. quad	. L4