# **CS 410 Binary to C++ Activity**

This might be the autism, but explaining what the assembly code douse, using that to make c/c++ code. Then you tell me to explain what that c++code douse? What difference do you expect me to explain? What more can I explain? Do you want me to copy past the explanation? Do you realize you are asking for a one to one conversion from assumably to c/++ then doing it again in reverse and expecting a different answer for no reason? And why do you keep giving me cells to put things in? How do you fit all that information in a cell in a format that makes sense? Can you give me an example please?

## **File One**

0000000000000000 <main>:

0: 55 push %rbp

1: 48 89 e5 mov %rsp,%rbp

4: 48 83 ec 10 sub $0x10,%rsp

8: c7 45 f8 01 00 00 00 movl $0x1,-0x8(%rbp)

f: 83 7d f8 09 cmpl $0x9,-0x8(%rbp)

13: 0f 8f 8a 00 00 00 jg a3 <main+0xa3>

19: c7 45 f4 01 00 00 00 movl $0x1,-0xc(%rbp)

20: 83 7d f4 09 cmpl $0x9,-0xc(%rbp)

24: 7f 74 jg 9a <main+0x9a>

26: 8b 45 f8 mov -0x8(%rbp),%eax

29: 0f af 45 f4 imul -0xc(%rbp),%eax

2d: 89 45 fc mov %eax,-0x4(%rbp)

30: 8b 45 f8 mov -0x8(%rbp),%eax

33: 89 c6 mov %eax,%esi

35: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # 3c <main+0x3c>

3c: e8 00 00 00 00 call 41 <main+0x41>

41: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi # 48 <main+0x48>

48: 48 89 c7 mov %rax,%rdi

4b: e8 00 00 00 00 call 50 <main+0x50>

50: 48 89 c2 mov %rax,%rdx

53: 8b 45 f4 mov -0xc(%rbp),%eax

56: 89 c6 mov %eax,%esi

58: 48 89 d7 mov %rdx,%rdi

5b: e8 00 00 00 00 call 60 <main+0x60>

60: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi # 67 <main+0x67>

67: 48 89 c7 mov %rax,%rdi

6a: e8 00 00 00 00 call 6f <main+0x6f>

6f: 48 89 c2 mov %rax,%rdx

72: 8b 45 fc mov -0x4(%rbp),%eax

75: 89 c6 mov %eax,%esi

77: 48 89 d7 mov %rdx,%rdi

7a: e8 00 00 00 00 call 7f <main+0x7f>

7f: 48 89 c2 mov %rax,%rdx

82: 48 8b 05 00 00 00 00 mov 0x0(%rip),%rax # 89 <main+0x89>

89: 48 89 c6 mov %rax,%rsi

8c: 48 89 d7 mov %rdx,%rdi

8f: e8 00 00 00 00 call 94 <main+0x94>

94: 83 45 f4 01 addl $0x1,-0xc(%rbp)

98: eb 86 jmp 20 <main+0x20>

9a: 83 45 f8 01 addl $0x1,-0x8(%rbp)

9e: e9 6c ff ff ff jmp f <main+0xf>

a3: b8 00 00 00 00 mov $0x0,%eax

a8: c9 leave

a9: c3 ret

The function prologue instructions are normal and appear at the beginning (0x00): conserving the base pointer with push %rbp mov %rsp, %rbp (for stack frame setup) sub $0x10,%rsp (stack allocation of 16 bytes) Two local variables (probably integers) are initialized: Local variable at -0x8(%rbp) (probably i) is initialized to 1 by the command movl $0x1,-0x8(%rbp). movl $0x1,-0xc(%rbp) sets the value of another local variable (probably j) at -0xc(%rbp) to 1. For loop for i in the outer loop: Using cmpl $0x9,-0x8(%rbp), the value at -0x8(%rbp) (i.e., i) is compared to 9. It advances to the function's conclusion (a3) if i > 9. If not, it goes back into the loop. Inside Loop (for loop inside j): The code uses cmpl $0x9,-0xc(%rbp) to compare j's condition to 9. The next iteration of the outer loop is reached if j is greater than 9. The Process of Multiplication and Storage I and J are multiplied by imul -0xc(%rbp),%eax, and the result is stored in %eax. Next, the outcome is stored at -0x4(%rbp). Print Function Invocation: There are many call instructions, which are probably calls to a printf function or something similar. %rdi, %rsi, and %rdx load the format string and arguments, in accordance with the standard x86-64 calling convention for variadic functions such as printf. Increasing j and i by one: After adding 1 to j using addl $0x1,-0xc(%rbp), the code returns to the beginning of the inner loop. The outer loop repeats when j is greater than 9, addl $0x1,-0x8(%rbp) advances i by 1. Functional Epilogue: A clean function exit is indicated by the final lines of the function, which are mov $0x0,%eax, returning 0, leave, and ret.

#include <iostream>

int main() {

for (int i = 1; i <= 9; ++i) {

for (int j = 1; j <= 9; ++j) {

int result = i \* j;

std::cout << i << " \* " << j << " = " << result << std::endl;

}

}

return 0;

}

For step five ill say its double loop (see above)

## **File Two**

0000000000000000 <main>:

0: 55 push %rbp

1: 48 89 e5 mov %rsp,%rbp

4: 48 83 ec 30 sub $0x30,%rsp

8: 64 48 8b 04 25 28 00 mov %fs:0x28,%rax

f: 00 00

11: 48 89 45 f8 mov %rax,-0x8(%rbp)

15: 31 c0 xor %eax,%eax

17: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi # 1e <main+0x1e>

1e: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # 25 <main+0x25>

25: e8 00 00 00 00 call 2a <main+0x2a>

2a: 48 89 c2 mov %rax,%rdx

2d: 48 8b 05 00 00 00 00 mov 0x0(%rip),%rax # 34 <main+0x34>

34: 48 89 c6 mov %rax,%rsi

37: 48 89 d7 mov %rdx,%rdi

3a: e8 00 00 00 00 call 3f <main+0x3f>

3f: 48 8d 45 ec lea -0x14(%rbp),%rax

43: 48 89 c6 mov %rax,%rsi

46: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # 4d <main+0x4d>

4d: e8 00 00 00 00 call 52 <main+0x52>

52: 8b 55 ec mov -0x14(%rbp),%edx

55: 8b 45 ec mov -0x14(%rbp),%eax

58: 0f af d0 imul %eax,%edx

5b: 8b 45 ec mov -0x14(%rbp),%eax

5e: 0f af c2 imul %edx,%eax

61: 89 45 ec mov %eax,-0x14(%rbp)

64: 8b 45 ec mov -0x14(%rbp),%eax

67: f2 0f 2a c0 cvtsi2sd %eax,%xmm0

6b: f2 0f 10 0d 00 00 00 movsd 0x0(%rip),%xmm1 # 73 <main+0x73>

72: 00

73: f2 0f 59 c1 mulsd %xmm1,%xmm0

77: f2 0f 11 45 f0 movsd %xmm0,-0x10(%rbp)

7c: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi # 83 <main+0x83>

83: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # 8a <main+0x8a>

8a: e8 00 00 00 00 call 8f <main+0x8f>

8f: 48 89 c2 mov %rax,%rdx

92: 48 8b 45 f0 mov -0x10(%rbp),%rax

96: 48 89 45 d8 mov %rax,-0x28(%rbp)

9a: f2 0f 10 45 d8 movsd -0x28(%rbp),%xmm0

9f: 48 89 d7 mov %rdx,%rdi

a2: e8 00 00 00 00 call a7 <main+0xa7>

a7: b8 00 00 00 00 mov $0x0,%eax

ac: 48 8b 4d f8 mov -0x8(%rbp),%rcx

b0: 64 48 33 0c 25 28 00 xor %fs:0x28,%rcx

b7: 00 00

b9: 74 05 je c0 <main+0xc0>

bb: e8 00 00 00 00 call c0 <main+0xc0>

c0: c9 leave

c1: c3 ret

Introduction: Push %rbp; move %rsp; sub $0x30,%rbp; and allot 48 bytes to the stack and set up the stack frame. The segment %fs:0x28 is loaded into rax; this may be done to prevent buffer overflows using stack canary protection. Initial Call Configuration: Numerous lea instructions load addresses into registers before calling a function at 0x25, which is probably a function that accepts two inputs (rsi, rdi), presumably for the purpose of printing or logging. Multiplication of integers: Repetitive integer multiplication is performed on the value at -0x14(%rbp) by accessing it many times and multiplying it: imul %eax,%edx and imul %edx,%eax. Operations at Floating Points: cvtsi2sd %eax,%xmm0 is the result of converting the integer value to a double. The following is a floating-point number multiplication: mulsd %xmm1,%xmm0. The outcome is kept in -0x10(%rbp). An Additional Function Call There is another sequence of function calls (0x8a, 0xa2), which probably makes use of the outcomes of these computations (maybe for output). Conclusion: Verification of the stack canary (xor %fs:0x28,%rcx), a conditional jump (je), and the standard function cleanup with leave and ret come next.

#include <iostream>

int main() {

int x = 5; // Assume the initial value is set earlier or input

x = x \* x \* x; // Integer multiplication (x^3)

double result = static\_cast<double>(x) \* 3.14; // Floating-point multiplication

std::cout << "Result: " << result << std::endl;

return 0;

}

## **File Three**

0000000000000000 <main>:

0: 55 push %rbp

1: 48 89 e5 mov %rsp,%rbp

4: 48 83 ec 20 sub $0x20,%rsp

8: 64 48 8b 04 25 28 00 mov %fs:0x28,%rax

f: 00 00

11: 48 89 45 f8 mov %rax,-0x8(%rbp)

15: 31 c0 xor %eax,%eax

17: c7 45 f4 01 00 00 00 movl $0x1,-0xc(%rbp)

1e: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi # 25 <main+0x25>

25: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # 2c <main+0x2c>

2c: e8 00 00 00 00 call 31 <main+0x31>

31: 48 89 c2 mov %rax,%rdx

34: 48 8b 05 00 00 00 00 mov 0x0(%rip),%rax # 3b <main+0x3b>

3b: 48 89 c6 mov %rax,%rsi

3e: 48 89 d7 mov %rdx,%rdi

41: e8 00 00 00 00 call 46 <main+0x46>

46: 48 8d 45 e8 lea -0x18(%rbp),%rax

4a: 48 89 c6 mov %rax,%rsi

4d: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # 54 <main+0x54>

54: e8 00 00 00 00 call 59 <main+0x59>

59: 8b 45 e8 mov -0x18(%rbp),%eax

5c: 83 e8 01 sub $0x1,%eax

5f: 89 45 f4 mov %eax,-0xc(%rbp)

62: c7 45 f0 01 00 00 00 movl $0x1,-0x10(%rbp)

69: 8b 45 e8 mov -0x18(%rbp),%eax

6c: 39 45 f0 cmp %eax,-0x10(%rbp)

6f: 7f 72 jg e3 <main+0xe3>

71: c7 45 ec 01 00 00 00 movl $0x1,-0x14(%rbp)

78: 8b 45 ec mov -0x14(%rbp),%eax

7b: 3b 45 f4 cmp -0xc(%rbp),%eax

7e: 7f 19 jg 99 <main+0x99>

80: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi # 87 <main+0x87>

87: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # 8e <main+0x8e>

8e: e8 00 00 00 00 call 93 <main+0x93>

93: 83 45 ec 01 addl $0x1,-0x14(%rbp)

97: eb df jmp 78 <main+0x78>

99: 83 6d f4 01 subl $0x1,-0xc(%rbp)

9d: c7 45 ec 01 00 00 00 movl $0x1,-0x14(%rbp)

a4: 8b 45 f0 mov -0x10(%rbp),%eax

a7: 01 c0 add %eax,%eax

a9: 83 e8 01 sub $0x1,%eax

ac: 39 45 ec cmp %eax,-0x14(%rbp)

af: 7f 19 jg ca <main+0xca>

b1: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi # b8 <main+0xb8>

b8: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # bf <main+0xbf>

bf: e8 00 00 00 00 call c4 <main+0xc4>

c4: 83 45 ec 01 addl $0x1,-0x14(%rbp)

c8: eb da jmp a4 <main+0xa4>

ca: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi # d1 <main+0xd1>

d1: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # d8 <main+0xd8>

d8: e8 00 00 00 00 call dd <main+0xdd>

dd: 83 45 f0 01 addl $0x1,-0x10(%rbp)

e1: eb 86 jmp 69 <main+0x69>

e3: c7 45 f4 01 00 00 00 movl $0x1,-0xc(%rbp)

ea: c7 45 f0 01 00 00 00 movl $0x1,-0x10(%rbp)

f1: 8b 45 e8 mov -0x18(%rbp),%eax

f4: 83 e8 01 sub $0x1,%eax

f7: 39 45 f0 cmp %eax,-0x10(%rbp)

fa: 7f 75 jg 171 <main+0x171>

fc: c7 45 ec 01 00 00 00 movl $0x1,-0x14(%rbp)

103: 8b 45 ec mov -0x14(%rbp),%eax

106: 3b 45 f4 cmp -0xc(%rbp),%eax

109: 7f 19 jg 124 <main+0x124>

10b: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi # 112 <main+0x112>

112: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # 119 <main+0x119>

119: e8 00 00 00 00 call 11e <main+0x11e>

11e: 83 45 ec 01 addl $0x1,-0x14(%rbp)

122: eb df jmp 103 <main+0x103>

124: 83 45 f4 01 addl $0x1,-0xc(%rbp)

128: c7 45 ec 01 00 00 00 movl $0x1,-0x14(%rbp)

12f: 8b 45 e8 mov -0x18(%rbp),%eax

132: 2b 45 f0 sub -0x10(%rbp),%eax

135: 01 c0 add %eax,%eax

137: 83 e8 01 sub $0x1,%eax

13a: 39 45 ec cmp %eax,-0x14(%rbp)

13d: 7f 19 jg 158 <main+0x158>

13f: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi # 146 <main+0x146>

146: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # 14d <main+0x14d>

14d: e8 00 00 00 00 call 152 <main+0x152>

152: 83 45 ec 01 addl $0x1,-0x14(%rbp)

156: eb d7 jmp 12f <main+0x12f>

158: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi # 15f <main+0x15f>

15f: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # 166 <main+0x166>

166: e8 00 00 00 00 call 16b <main+0x16b>

16b: 83 45 f0 01 addl $0x1,-0x10(%rbp)

16f: eb 80 jmp f1 <main+0xf1>

171: b8 01 00 00 00 mov $0x1,%eax

176: 48 8b 4d f8 mov -0x8(%rbp),%rcx

17a: 64 48 33 0c 25 28 00 xor %fs:0x28,%rcx

181: 00 00

183: 74 05 je 18a <main+0x18a>

185: e8 00 00 00 00 call 18a <main+0x18a>

18a: c9 leave

18b: c3 ret

Introduction: Push %rbp, mov %rsp,%rbp, sub $0x20,%rsp (allocating 32 bytes on the stack) is the standard function prologue. Loaded (mov %fs:0x28,%rax), a reference to the thread-local storage or stack canary is placed at -0x8(%rbp). Starting Points: Local variable (i) is initialized to 1 by using movl $0x1,-0xc(%rbp). Subsequent initializations are made, assigning values of 1 at different times to -0x10(%rbp) and -0x14(%rbp). Primary Loop Architecture: The repeated jmp, cmp, and conditional jump instructions in the program show that it comprises several loops: A value (eax) is compared with another local variable (-0x10(%rbp)) using the cmp %eax, -0x10(%rbp) method. If larger, it advances to the location at 0xe3, which denotes a further phase of the loop. When two local variables are compared, cmp -0xc(%rbp),%eax jumps correspondingly based on the outcome. Repeated loops in this design keep the control flow going while modifying the values of the local variables. Operations with Arithmetic: Several sub and add instructions denote arithmetic operations on the local variables (sub $0x1,%eax, add %eax,%eax). Conditional jumps, which regulate the flow based on the outcomes of various operations, are inserted between them.

Calls to Functions: There are other function calls visible (such as calls 0x31 and 0x93), which probably correspond to print statements or other I/O activities. These function calls take place in between iterations of the loop, most likely to show the computation results. Statement of Return: The function ends with the following return value: 1: mov $0x1,%eax, and the function epilogue (leave, ret).

#include <iostream>

int main() {

int i = 1; // Initial value

int j, k;

while (i > 0) {

j = i - 1;

int result = 1;

while (j > 0) {

k = 1;

while (k <= j) {

// Perform some calculations with i, j, and k

result += (i \* j - k);

k++;

}

j--;

}

// Output the result of calculations (simulating the call instructions)

std::cout << "Result: " << result << std::endl;

i--; // Decrement i to eventually exit the loop

}

return 1;

}

## **File Four**

0000000000000000 <main>:

0: 55 push %rbp

1: 48 89 e5 mov %rsp,%rbp

4: 48 83 ec 30 sub $0x30,%rsp

8: 64 48 8b 04 25 28 00 mov %fs:0x28,%rax

f: 00 00

11: 48 89 45 f8 mov %rax,-0x8(%rbp)

15: 31 c0 xor %eax,%eax

17: 48 c7 45 e0 00 00 00 movq $0x0,-0x20(%rbp)

1e: 00

1f: 48 c7 45 e8 01 00 00 movq $0x1,-0x18(%rbp)

26: 00

27: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi # 2e <main+0x2e>

2e: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # 35 <main+0x35>

35: e8 00 00 00 00 call 3a <main+0x3a>

3a: 48 89 c2 mov %rax,%rdx

3d: 48 8b 05 00 00 00 00 mov 0x0(%rip),%rax # 44 <main+0x44>

44: 48 89 c6 mov %rax,%rsi

47: 48 89 d7 mov %rdx,%rdi

4a: e8 00 00 00 00 call 4f <main+0x4f>

4f: 48 8d 45 d8 lea -0x28(%rbp),%rax

53: 48 89 c6 mov %rax,%rsi

56: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # 5d <main+0x5d>

5d: e8 00 00 00 00 call 62 <main+0x62>

62: 48 8b 45 d8 mov -0x28(%rbp),%rax

66: 48 85 c0 test %rax,%rax

69: 0f 84 83 00 00 00 je f2 <main+0xf2>

6f: 48 8b 4d d8 mov -0x28(%rbp),%rcx

73: 48 ba 67 66 66 66 66 movabs $0x6666666666666667,%rdx

7a: 66 66 66

7d: 48 89 c8 mov %rcx,%rax

80: 48 f7 ea imul %rdx

83: 48 c1 fa 02 sar $0x2,%rdx

87: 48 89 c8 mov %rcx,%rax

8a: 48 c1 f8 3f sar $0x3f,%rax

8e: 48 29 c2 sub %rax,%rdx

91: 48 89 d0 mov %rdx,%rax

94: 48 89 45 f0 mov %rax,-0x10(%rbp)

98: 48 8b 55 f0 mov -0x10(%rbp),%rdx

9c: 48 89 d0 mov %rdx,%rax

9f: 48 c1 e0 02 shl $0x2,%rax

a3: 48 01 d0 add %rdx,%rax

a6: 48 01 c0 add %rax,%rax

a9: 48 29 c1 sub %rax,%rcx

ac: 48 89 c8 mov %rcx,%rax

af: 48 89 45 f0 mov %rax,-0x10(%rbp)

b3: 48 8b 45 f0 mov -0x10(%rbp),%rax

b7: 48 0f af 45 e8 imul -0x18(%rbp),%rax

bc: 48 01 45 e0 add %rax,-0x20(%rbp)

c0: 48 d1 65 e8 shlq -0x18(%rbp)

c4: 48 8b 4d d8 mov -0x28(%rbp),%rcx

c8: 48 ba 67 66 66 66 66 movabs $0x6666666666666667,%rdx

cf: 66 66 66

d2: 48 89 c8 mov %rcx,%rax

d5: 48 f7 ea imul %rdx

d8: 48 c1 fa 02 sar $0x2,%rdx

dc: 48 89 c8 mov %rcx,%rax

df: 48 c1 f8 3f sar $0x3f,%rax

e3: 48 29 c2 sub %rax,%rdx

e6: 48 89 d0 mov %rdx,%rax

e9: 48 89 45 d8 mov %rax,-0x28(%rbp)

ed: e9 70 ff ff ff jmp 62 <main+0x62>

f2: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi # f9 <main+0xf9>

f9: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # 100 <main+0x100>

100: e8 00 00 00 00 call 105 <main+0x105>

105: 48 89 c2 mov %rax,%rdx

108: 48 8b 45 e0 mov -0x20(%rbp),%rax

10c: 48 89 c6 mov %rax,%rsi

10f: 48 89 d7 mov %rdx,%rdi

112: e8 00 00 00 00 call 117 <main+0x117>

117: 48 89 c2 mov %rax,%rdx

11a: 48 8b 05 00 00 00 00 mov 0x0(%rip),%rax # 121 <main+0x121>

121: 48 89 c6 mov %rax,%rsi

124: 48 89 d7 mov %rdx,%rdi

127: e8 00 00 00 00 call 12c <main+0x12c>

12c: b8 00 00 00 00 mov $0x0,%eax

131: 48 8b 75 f8 mov -0x8(%rbp),%rsi

135: 64 48 33 34 25 28 00 xor %fs:0x28,%rsi

13c: 00 00

13e: 74 05 je 145 <main+0x145>

140: e8 00 00 00 00 call 145 <main+0x145>

145: c9 leave

146: c3 ret

Introduction: Push %rbp; move %rsp; sub $0x30,%rbp; and allot 48 bytes to the stack and set up the stack frame. Loaded into rax, a reference to the thread-local storage or stack canary is kept at -0x8(%rbp). Initialization of Variable: $0x0,-0x20(%rbp) movq sets to zero a local variable located at -0x20(%rbp). Another local variable at -0x18(%rbp) is initialized to 1 by the command movq $0x1,-0x18(%rbp). Start of Loop: The code then enters a loop, which starts at 0x62. Within the loop: (test %rax,%rax) A comparison is performed at 0x66 to see if a value stored at -0x28(%rbp) is zero. The loop ends at 0xf2 if zero. A sequence of arithmetic operations—mostly multiplications, shifts, and additions—occurs if the value is non-zero. Arithmetic Operations using Integers: A number of computations include subtracting values from rax and rcx, shifting (sar and shl), and multiplying a huge constant (0x666666666666666667)—these operations may be connected to division by a constant. The numbers at -0x18(%rbp) and -0x28(%rbp) are multiplied by imul at 0xb7, which then records the outcome. The loop jumps back to the loop's beginning (jmp 0x62) after iterating and modifying values. Close the loop and tidy up: Function calls are made at 0xf9 and 0x105 after the loop ends, potentially for output or more calculations. When the function eventually ends, the stack is cleared and 0 is returned.

#include <iostream>

int main() {

long result = 0; // Variable corresponding to -0x20(%rbp)

long value = 1; // Variable corresponding to -0x18(%rbp)

long x = 123456789; // A constant value similar to -0x28(%rbp)

while (x != 0) {

long temp = (x \* 0x6666666666666667) >> 2; // Division-like operation

long div = x - ((x >> 63) - temp); // Adjusted division result

long mul = div \* value; // Multiply div result with value

result += mul; // Accumulate result

value <<= 1; // Bit shift value (shl)

// The loop continues until x reaches 0

x = div;

}

// Some final output operations

std::cout << "Result: " << result << std::endl;

return 0;

}