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

## **File One**

**Step 2:** Explain the functionality of the blocks of assembly code.

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| push %rbp  mov %rsp,%rbp  sub $0x10,%rsp | Push register rbp to the stack  Move register rbp to the address value of rsp  Subtract 10 from register rsp |
| movl $0x1,-0x8(%rbp)  cmpl $0x9,-0x8(%rbp)  jg 0x9ad <main+163> | Variables a, b, and c are declared as type int. Set variable a o 8 bits of register rbp and set to 1.  Compare register rbp at 8 bits to 9  If rbp is less than or equal to 9, store 1 as value of register rbp at 8 bits. If greater than 9, jum to 9ad |
| movl $0x1,-0xc(%rbp)  cmpl $0x9,-0xc(%rbp)  jg 0x9a4 <main+154> | The next location in rbp is c. this will set the vale of the next vriable b.  Compare the value of register rbp at c bits to the value of 9.  If greater than 9, jump to 9a4. If less than or equal to 9, store 1 as the value for register rbp at c bits. |
| mov -0x8(%rbp),%eax  imul -0xc(%rbp),%eax  mov %eax,-0x4(%rbp)  mov -0x8(%rbp),%eax  mov %eax,%esi  lea 0x2006da(%rip),%rdi # 0x201020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x7e0 <\_ZNSolsEi@plt> | The register eax will store the value at address rbp at 8 bits to it.  Multiply eax by the value at address rbp at c bits.  Store the multiplied value to rbp at 4 bits  Output the value of the multiplication |
| lea 0x153(%rip),%rsi # 0xaa5  mov %rax,%rdi  callq 0x7b0 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | Output the value from register rdi to the program |
| mov %rax,%rdx  mov -0xc(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 0x7e0 <\_ZNSolsEi@plt> | Move register rdx to rax  Move register eax and store the value of rbp at c bits to it  Move register rdi to register rdx  Output the values stored to rdx |
| mov %rax,%rdx  mov -0x4(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 0x7e0 <\_ZNSolsEi@plt> | Move register rdx to rax  Move register eax and store the value of rbp at c bits to it  Move register rdi to register rdx  Output the values stored to rdx |

**Step 4:** Convert the assembly code to C++ code.

**Step 5:** Explain how the C++ code performs the same tasks as the blocks of assembly code.

| **Blocks of Assembly Code** | **C++ Code** | **Explanation of Functionality** |
| --- | --- | --- |
| push %rbp  mov %rsp,%rbp  sub $0x10,%rsp | Int main () {  Int a, b, c | This sets the main funtion of the code. Within the main function, variables a, b, and c are declared of type integer. |
| movl $0x1,-0x8(%rbp)  cmpl $0x9,-0x8(%rbp)  jg 0x9ad <main+163> | For (a = 1; a <= 9; ++a) | This establishes the for loop to iterate the variable a. It states that a is set to 1. As long as a is less than or equal to 9, jump to the next address |
| addl $0x1,-0x8(%rbp)  jmp 0x92a <main+32> | For (a = 1; a <= 9; ++a) | This section focuses on the incrementing of variable a by 1 after the comparison to 9.  Once the parameters are not met, the program will exit. |
| movl $0x1,-0xc(%rbp)  cmpl $0x9,-0xc(%rbp)  jg 0x9a4 <main+154> | for (b = 1; b <= 9; ++b) { | This establishes the for loop for variable b. This loop is nested within the variable a for loop. This loop states that b is set to 1. It puts comparison of b being less than or equal to 9 in order to increment |
| addl $0x1,-0x8(%rbp)  jmpq 0x919 <main+15> | for (b = 1; b <= 9; ++b) { | This code block focuses on the incrementing of variable b, based on the comparison to the value of 9. It will increment b by a value of 1.  Once the parameters are not met, the program will exit. |
| mov -0x8(%rbp),%eax  imul -0xc(%rbp),%eax  mov %eax,-0x4(%rbp)  mov -0x8(%rbp),%eax  mov %eax,%esi  lea 0x2006da(%rip),%rdi # 0x201020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x7e0 <\_ZNSolsEi@plt> | C = a \* b ; | Variable c is set to the product of variable a and variable b |
| mov %rax,%rdx  mov -0x4(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 0x7e0 <\_ZNSolsEi@plt> | cout << a << "\*" << b << "=" << c << endl; | Output through cout call of variable a , “\*” string, variable b, “=” string, and variable c. output a newline. |

## **File Two**

**Step 2:** Explain the functionality of the blocks of assembly code.

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| mov %rsp,%rbp  sub $0x30,%rsp  mov %fs:0x28,%rax  mov %rax,-0x8(%rbp) | Push register rbp to rsp  Subtract 30 from register rsp  Move register rax to offset 0x28  Move register rbp at 8 bits to rax |
| lea    0x191(%rip),%rsi  # 0xba9  lea    0x201601(%rip),%rdi  # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq  0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  mov    %rax,%rdx  mov    0x2015a2(%rip),%rax # 0x201fd0  mov    %rax,%rsi  mov    %rdx,%rdi  callq  0x8a0 <\_ZNSolsEPFRSoS\_E@plt> | Load effective address of rsi in to rip at ox191 bits  Load effective address of rdi into rip at 0x1601 bits  Call upon instruction to output string  Move register rdx into register rax  Move register rsi into register rax  Move rdi into register rdx |
| lea -0x14(%rbp),%rax  mov %rax,%rsi  lea 0x2016f9(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  callq 0x870 <\_ZNSirsERi@plt> | Load effective address of rax into rbp at 14 bits  Move register rsi to rax  Call instruction for cin for user input to set to variable a |
| mov -0x14(%rbp),%edx  mov -0x14(%rbp),%eax | Move register edx to address of register rbp at 14 bits  Move register eax to address of register rbp at 14 bits. This set variable pi to 3.14. |
| mov -0x14(%rbp),%edx  mov -0x14(%rbp),%eax  imul %eax,%edx  mov -0x14(%rbp),%eax  imul %edx,%eax  mov %eax,-0x14(%rbp)  mov -0x14(%rbp),%eax  cvtsi2sd %eax,%xmm0  movsd 0x15b(%rip),%xmm1 # 0xbc8  mulsd %xmm1,%xmm0  movsd %xmm0,-0x10(%rbp) | Move register edx to address of rbp at 14 bits  Move register eax to rbp at 14 bits  multiply value of regiser edx by value of register eax  move register eax to address of register rbp at 14 bits. This stores this multiplication value  The multiplication is of variable a collected multiplied by itself, and this product multiplied by variable a again. This product is then multiplied by variable pi. |
| lea 0x13a(%rip),%rsi # 0xbb7  lea 0x20159c(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  mov %rax,%rdx  mov -0x10(%rbp),%rax  mov %rax,-0x28(%rbp) | Load effective address of rsi to rip at 13a bits  Call output for instruction to specified string  Move register rdx to rax  Move rax to rbp at 10 bits  Move rbp at 28 bits to rax  The moves and calls the outputs of the specified strings and then outputs them along with variables that were called as well |

**Step 4:** Convert the assembly code to C++ code.

**Step 5:** Explain how the C++ code performs the same tasks as the blocks of assembly code.

| **Blocks of Assembly Code** | **C++ Code** | **Explanation of Functionality** |
| --- | --- | --- |
| mov %rsp,%rbp  sub $0x30,%rsp  mov %fs:0x28,%rax  mov %rax,-0x8(%rbp) | int main() {  int a,b;  float pi;  double volume; | Declare the function main for the program. Declare variables a and b of type int. Declare variable pi of type float. Declare variable volume as type double. |
| lea 0x191(%rip),%rsi # 0xba9  lea 0x201601(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  mov %rax,%rdx  mov 0x2015a2(%rip),%rax # 0x201fd0  mov %rax,%rsi  mov %rdx,%rdi  callq 0x8a0 <\_ZNSolsEPFRSoS\_E@plt> | cout << "Enter the Radius" << endl ; | Output the string “ Enter the Radius” and create a newline at the end of the output statement |
| lea -0x14(%rbp),%rax  mov %rax,%rsi  lea 0x2016f9(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  callq 0x870 <\_ZNSirsERi@plt> | cin >> a; | Collect user input via the cin call. Store user input into variable a. |
| mov -0x14(%rbp),%edx  mov -0x14(%rbp),%eax | pi = 3.14; | Set variable pi to 3.14 |
| mov -0x14(%rbp),%edx  mov -0x14(%rbp),%eax  imul %eax,%edx  mov -0x14(%rbp),%eax  imul %edx,%eax  mov %eax,-0x14(%rbp)  mov -0x14(%rbp),%eax  cvtsi2sd %eax,%xmm0  movsd 0x15b(%rip),%xmm1 # 0xbc8  mulsd %xmm1,%xmm0  movsd %xmm0,-0x10(%rbp) | volume = pi \* (a\*a\*a); | Set variable volume to pi multiplied by the product of  a\*a\*a ( or a cubed) |
| lea 0x13a(%rip),%rsi # 0xbb7  lea 0x20159c(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  mov %rax,%rdx  mov -0x10(%rbp),%rax  mov %rax,-0x28(%rbp) | cout << "The volume is: " << volume << endl; | Output call of string “The volume is: “ then the variable volume, along with a newline at the end of the statement. |

## **File Three**

**Step 2:** Explain the functionality of the blocks of assembly code.

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax | Move address of offset 0x28 value to register rax  Move register rax to rbp at 8 bits  set register eax to 0 the xor instruction, clearing the register |
| lea 0x256(%rip),%rsi # 0xc35  lea 0x20163a(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  mov %rax,%rdx  mov 0x2015db(%rip),%rax # 0x201fd0  mov %rax,%rsi  mov %rdx,%rdi  callq 0x870 <\_ZNSolsEPFRSoS\_E@plt> | Load effective address of rip at 256, to rsi.  Load effective address rip at 20163a to register rdi  This will call the output of the string  Move rax to register rdx. Through the moves being made it is storing a value of int a from the input by the user into register rdi, after the call instruction. |
| movl $0x1,-0x10(%rbp)  mov -0x18(%rbp),%eax  cmp %eax,-0x10(%rbp)  jg 0xa9d <main+227> | Move value of 0x1 to rbp at 10 bits  Move address of rbp at 18 bits to eax  Compare operands of eax and rbp at 10 bits  Jump if greater to main 227 |
| movl $0x1,-0x14(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0xa53 <main+153> | Move value of 0x1 to rbp at 14 bits  Move rbp at 14 bits to register eax  Compare rbp at 0xc bits to register eax  Jump if greater in the conditional to main 153 |
| movl -20(%rbp), %eax    sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0xa84 <main+202>  addl $0x1,-0x14(%rbp)  jmp 0xa32 <main+120> | Move rbp at 20 bits to register eax  subtract 1 from register eax  compare register eax to rbp at 14 bits  jump if greater to main 202  add 1 to value of rbp register at 14 bits  change program counter after executing unconditional jump to main 120 |
| lea 0x209(%rip),%rsi # 0xc4a  lea 0x2015d8(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | Load effective address of value of rip at 0x209 bits to register rsi  Call for the output function of program |
| subl $0x1,-0xc(%rbp)  movl $0x1,-0x14(%rbp)  mov -0x10(%rbp),%eax  add %eax,%eax  sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0xa84 <main+202> | Subtract 1 from register rbp at 0xc bits  Move 1 value of register rbp at 14 bits  Move 10 bits of rbp address to eax  Add register eax to to register eax  Subtract 1 from register eax  Compare eax to register rbp at 14 bits  Jump if greater to main 202 |
| lea 0x1da(%rip),%rsi # 0xc4c  lea 0x2015a7(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | Load effective address of rip at 0x1da to register rsi  Load effective address of rip at 0x15a7 to register rdi  Call to output string from function |
| addl $0x1,-0x14(%rbp)  jmp 0xa5e <main+164> | Add 1 to register rbp at 14 bits  Unconditional jump at main 164 of program counter |
| addl $0x1,-0x10(%rbp)  jmp 0xa23 <main+105>  movl $0x1,-0xc(%rbp)  movl $0x1,-0x10(%rbp)  mov -0x18(%rbp),%eax  sub $0x1,%eax  cmp %eax,-0x10(%rbp)  jg 0xb2b <main+369>  addl $0x1,-0x10(%rbp)  jmp 0xaab <main+241> | At 1 to register rbp at 10 bits  Unconditional jump at main 105 of program counter  Move 0x1 to rbp at 0xc bits  Move 0x1 to register rbp at 10 bits  Move rbp at 18 bits to register eax  Subtract 1 from register eax  Compare operands of eax and rbp at 10 bits  Jump if greater to main 369  At 1 to register rbp at 10 bits  Unconditional jump of program counter to main 241 |
| movl $0x1,-0x14(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0xade <main+292> | Move 0x1 to register rbp at 14 bits  Move register rbp at 14 bits to register eax  Compare operands of rbp at 0xc bits and register eax  Jump if greater to main 292 |
| addl $0x1,-0x14(%rbp)  jmp 0xabd <main+259> | Add 1 to register rbp at 14 bits  Unconditional jump of program counter to main 259 |
| movl $0x1,-0x14(%rbp) | Move 1 to register rbp at 14 bits |
| mov -0x18(%rbp),%eax  sub -0x10(%rbp),%eax  add %eax,%eax  sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0xb12 <main+344> | Move register rbp address at 18 bits to register eax  Subtract rbp at 10 bits from register eax  Add register eax to register eax  Subtract 1 from register eax  Compare register eax to register rbp at 14 bits  Jump if greater to main 244 |

**Step 4:** Convert the assembly code to C++ code.

**Step 5:** Explain how the C++ code performs the same tasks as the blocks of assembly code.

| **Blocks of Assembly Code** | **C++ Code** | **Explanation of Functionality** |
| --- | --- | --- |
| mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax | Int main () {  int n, i, j; | Declare main function of the program. Declare integers n, i, j |
| lea 0x256(%rip),%rsi # 0xc35  lea 0x20163a(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  mov %rax,%rdx  mov 0x2015db(%rip),%rax # 0x201fd0  mov %rax,%rsi  mov %rdx,%rdi  callq 0x870 <\_ZNSolsEPFRSoS\_E@plt> | cout<<"Enter number of rows"<<endl; | Use cout output to output string “Enter number of rows”. The next cout output is a newline |
| lea -0x18(%rbp),%rax  mov %rax,%rsi  lea 0x201732(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  callq 0x840 <\_ZNSirsERi@plt> | cin >> n; | Use cin input to gather user input and set to variable n |
| movl $0x1,-0x10(%rbp)  mov -0x18(%rbp),%eax  cmp %eax,-0x10(%rbp)  jg 0xa9d <main+227> | for (j = 1; j <= n; j++) | Declare a for loop of variable j set to 1, while j is less than or equal to n, increment j |
| movl -20(%rbp), %eax    sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0xa84 <main+202>  addl $0x1,-0x14(%rbp)  jmp 0xa32 <main+120> | for (i = 1; i <= n - j; i++) | This is a nested for loop within the other for loop. This conditional loop sets I to 1. While I is less than or equal to n-j, increment 1. This for loop will have other nested loops within. This will create the top half of the diamond pattern |
| lea 0x209(%rip),%rsi # 0xc4a  lea 0x2015d8(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | cout<<" " | Use cout output call to output string “ “. This creates a space |
| subl $0x1,-0xc(%rbp)  movl $0x1,-0x14(%rbp)  mov -0x10(%rbp),%eax  add %eax,%eax  sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0xa84 <main+202> | for (i = 1; i <= 2 \* j - 1; i++) | The next nested for loop happens as I is set to 1. While I is less than or equal to 2 \* j-1, increment 1 |
| lea 0x1da(%rip),%rsi # 0xc4c  lea 0x2015a7(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | cout<<"\*"; | Use cout output to output the string “\*” in order to create this symbol as an output. |
| addl $0x1,-0x14(%rbp)  jmp 0xabd <main+259> | cout << endl;; | The next cout output creates a newline |
| addl $0x1,-0x10(%rbp)  jmp 0xa23 <main+105>  movl $0x1,-0xc(%rbp)  movl $0x1,-0x10(%rbp)  mov -0x18(%rbp),%eax  sub $0x1,%eax  cmp %eax,-0x10(%rbp)  jg 0xb2b <main+369>  addl $0x1,-0x10(%rbp)  jmp 0xaab <main+241> | for (j = 1; j <= n; j++) | This is the next for loop that is started. This conditional sets j to 1. While j is less than or equal to n, increment j. This for loop will have other nested loops within it. This will create the bottom half of the diamond output. |
| movl $1, -16(%rbp)  mov -0x18(%rbp),%eax  cmp %eax,-0x10(%rbp)  jg 0xa9d <main+227> | for (i = 1; i <= j; i++) | This is the next for loop nested within. I is set to 1. While I is less than or equal to j increment i |
| lea 0x209(%rip),%rsi # 0xc4a  lea 0x2015d8(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | cout<<" "; | Use cout output to output the string “ “. This create a space between symbols |
| mov -0x18(%rbp),%eax  sub -0x10(%rbp),%eax  add %eax,%eax  sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0xb12 <main+344> | for (i = 1; i <= 2 \* (n - j) - 1; i++) | The next nested for loop sets I to 1. While I is less than or equal to 2\*(n-j) – 1, increment i |
| lea 0x1da(%rip),%rsi # 0xc4c  lea 0x2015a7(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x860 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt> | cout<<"\*"; | The cout output will print the string “\*” |
| addl $0x1,-0x14(%rbp)  jmp 0xabd <main+259> | Cout<< endl; | Create a newline |

## **File Four**

**Step 2:** Explain the functionality of the blocks of assembly code.

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax | Main  Move value from 0x28 to rax  Move value from rax to register rbp at 8 bits  set register eax to 0 the xor instruction, clearing the register |
| movq $0x0,-0x20(%rbp)  movq $0x1,-0x18(%rbp) | Var  Set value of register rbp at 20 bits to 0  Set value of register rbp at 10 bits to 1 |
| lea 0x201(%rip),%rsi # 0xc29  lea 0x2015f1(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  mov %rax,%rdx  mov 0x201592(%rip),%rax # 0x201fd0  mov %rax,%rsi  mov %rdx,%rdi  callq 0x8a0 <\_ZNSolsEPFRSoS\_E@plt> | Cout  Load effective address of rip at 201 to register rsi  Load effective address of rip at 2015f1 to register rdi  Call for output string at designated instruction  Move value of rax to rdx  Move value of 201592 of rip to register rax  Move value of rax to register rsi  Move value of rdx to register rdi  Call output instruction |
| lea -0x28(%rbp),%rax  mov %rax,%rsi  lea 0x2016e9(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  callq 0x870 <\_ZNSirsERl@plt> | Cin  Load effective address of rbp at 20 bits to register rax  Move value of rax to register rsi  Load effective address of rip to rdi. This loads value to variable binary  Call instruction to load cin |
| mov -0x28(%rbp),%rax  test %rax,%rax  je 0xaec <main+242>  jmpq 0xa5c <main+98> | While  Move rbp at 28 bits to register rax  Test for equivalency of rax and register rax  Jump if equal to load main 242  Unconditional jump to increase program counter to main 98 |
| mov -0x28(%rbp),%rcx  movabs $0x6666666666666667,%rdx  mov %rcx,%rax  imul %rdx  sar $0x2,%rdx  mov %rcx,%rax  sar $0x3f,%rax  sub %rax,%rdx  mov %rdx,%rax  mov %rax,-0x10(%rbp)  mov -0x10(%rbp),%rdx  mov %rdx,%rax  shl $0x2,%rax  add %rdx,%rax  add %rax,%rax  sub %rax,%rcx  mov %rcx,%rax  mov %rax,-0x10(%rbp) | Rem  Move rbp at 28 bits value to register rcx  load arbitrary 64-bit constant into register rdx and to load/store integer register from/to arbitrary constant 64-bit address is available  move register rcx value into register rax  multiply register rdx  shift right 2 of register rdx  move value of rcx to register rax  shift right 3f (hex) of register rax  subtract value of rax from register rdx  move value of rdx to register rax  move value of rax to register rbp at 10 bits  move value of rbp at 10 bits to register rdx  move value of rdx to register rax  shift left register rax value  add value of rdx to register rax  add value of rax to register rax  subtract value of rax from register rcx  move value of rcx to register rax  move value of rax to rbp at 10 bits |
| mov -0x10(%rbp),%rax  imul -0x18(%rbp),%rax  add %rax,-0x20(%rbp) | Hex rem  Move value of rbp at 10 bits to register rax  Multiply value of rbp at 18 bits by register rax  Add value of rax to rbp at 20 bits |
| shl -0x18(%rbp) | Shift left rbp at 18 bits |
| mov -0x28(%rbp),%rcx  movabs $0x6666666666666667,%rdx  mov %rcx,%rax  imul %rdx  sar $0x2,%rdx  mov %rcx,%rax  sar $0x3f,%rax  sub %rax,%rdx  mov %rdx,%rax  mov %rax,-0x28(%rbp) | Binary  Move value of rbp at 28 bits to register rcx  load arbitrary 64-bit constant into register rdx and to load/store integer register from/to arbitrary constant 64-bit address is available  move rcx to register rax  multiply register rdx  shift right 2 bits register rdx  move rcx to register rax  shift right 03f bits register rax  subtract value of rax from register rdx  move value of rdx to register rax  move value of rax to rbp at 28 bits |
| lea 0x155(%rip),%rsi # 0xc48  lea 0x201526(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  mov %rax,%rdx  mov -0x20(%rbp),%rax  mov %rax,%rsi  mov %rdx,%rdi  callq 0x8d0 <\_ZNSolsEl@plt> | Cout  Load effective address of rip at 155 bits to register rsi  Load effective address of rip to register rdi  Call instruction for cout output  Move rax to register rdx  Move value of rbp at 20 bits to register rax  Move rx to register rsi  Move rdx to register rdi  Call instruction for output of cout |

**Step 4:** Convert the assembly code to C++ code.

**Step 5:** Explain how the C++ code performs the same tasks as the blocks of assembly code.

| **Blocks of Assembly Code** | **C++ Code** | **Explanation of Functionality** |
| --- | --- | --- |
| mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax | Int main() {  Long binary, hex, remainder, a | Declare function main. Declare variables binary, hex, remainder, and a of type long |
| movq $0x0,-0x20(%rbp)  movq $0x1,-0x18(%rbp) | Hex = 0;  A = 1; | Set variable hex to 0.  Set variable a to 1 |
| lea 0x201(%rip),%rsi # 0xc29  lea 0x2015f1(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  mov %rax,%rdx  mov 0x201592(%rip),%rax # 0x201fd0  mov %rax,%rsi  mov %rdx,%rdi  callq 0x8a0 <\_ZNSolsEPFRSoS\_E@plt> | cout << "Enter Binary Number" << endl; | Utilize cout for output of string “Enter Binary Number. The utilize cout command to create a newline |
| lea -0x28(%rbp),%rax  mov %rax,%rsi  lea 0x2016e9(%rip),%rdi # 0x202140 <\_ZSt3cin@@GLIBCXX\_3.4>  callq 0x870 <\_ZNSirsERl@plt> | cin >> binary; | Utilize cin command to gather user input. Store user input into variable binary |
| mov -0x28(%rbp),%rax  test %rax,%rax  je 0xaec <main+242>  jmpq 0xa5c <main+98> | while (binary != 0) { | Create a while conditional statement to set parameter of varible binary not equalling 0 |
| mov -0x28(%rbp),%rcx  movabs $0x6666666666666667,%rdx  mov %rcx,%rax  imul %rdx  sar $0x2,%rdx  mov %rcx,%rax  sar $0x3f,%rax  sub %rax,%rdx  mov %rdx,%rax  mov %rax,-0x10(%rbp)  mov -0x10(%rbp),%rdx  mov %rdx,%rax  shl $0x2,%rax  add %rdx,%rax  add %rax,%rax  sub %rax,%rcx  mov %rcx,%rax  mov %rax,-0x10(%rbp) | remainder = binary % 10; | Set variable remainder to binary modulo 10. This will et the remainder to binarys data and divided by 10. Whatever the remainder after this division is will be set to the variable remainder. Example : binary = 19. Remainder will then be set to 9 after the division |
| mov -0x10(%rbp),%rax  imul -0x18(%rbp),%rax  add %rax,-0x20(%rbp) | hex = hex + (remainder \* i); | Set variable hex to hex value added to variable remainder multiplied by variable i. |
| shl -0x18(%rbp) | A = a \* 2; | Set variable a to variable a multiplied by 2 |
| mov -0x28(%rbp),%rcx  movabs $0x6666666666666667,%rdx  mov %rcx,%rax  imul %rdx  sar $0x2,%rdx  mov %rcx,%rax  sar $0x3f,%rax  sub %rax,%rdx  mov %rdx,%rax  mov %rax,-0x28(%rbp) | binary = binary / 10; | Set variable binary to the value of binary divided by 10. |
| lea 0x155(%rip),%rsi # 0xc48  lea 0x201526(%rip),%rdi # 0x202020 <\_ZSt4cout@@GLIBCXX\_3.4>  callq 0x890 <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>  mov %rax,%rdx  mov -0x20(%rbp),%rax  mov %rax,%rsi  mov %rdx,%rdi  callq 0x8d0 <\_ZNSolsEl@plt> | cout << "Equivalent hexadecimal value: " << hex; | Utilize cout command to output string “Equivalent hexadecimal value: “ and then the value of variable hex. |
| mov %rax,%rdx | Return 0; | Return 0 from the main function. Since main is of type int, it must return an int |