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

Ismael DeRocco

Southern New Hampshire University

CS-410

7/23/2023

## **File One**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| push %rbp | Push %rbp to register |
| mov %rsp,%rbp | Move value at %rsp to %rbp |
| sub $0x10,%rsp | Subtract 16 from %rsp |
| movl $0x1,-0x8(%rbp) | Move 1 to 8 bytes above %rbp |
| cmpl $0x9,-0x8(%rbp) | Check if value is value 8 bytes above %rbp is less than 9 |
| jg 0xa3 <main+163> | If greater jump to 163 |
| movl $0x1,-0xc(%rbp) | Move 1 to 12 bytes above %rbp register |
| cmpl $0x9,-0xc(%rbp) | Compare 9 and 12 bytes above %rbp |
| jg 0x9a <main+154> | Jump if greater to main 154 |
| mov -0x8(%rbp),%eax | Move 8 bytes above %rbp to %eax |
| imul -0xc(%rbp),%eax | Multiply 12 bytes above %rbp and %eax register |
| mov %eax,-0x4(%rbp) | Move value %eax to 4 bytes above %rbp |
| mov -0x8(%rbp),%eax | Move 8 bytes above %rbp to %eax |
| mov %eax,%esi | Move %eax to %esi |
| lea 0x0(%rip),%rdi # 0x3c <main+60> | Load effective address 0(%rip) to %rdi |
| callq 0x41 <main+65> | Call 65 |
| lea 0x0(%rip),%rsi # 0x48 <main+72> | Load effective address 0(%rip) to %rsi |
| mov %rax,%rdi | Move %rax to %rdi |
| callq 0x50 <main+80> | Call 80 |
| mov %rax,%rdx | Move %rax to %rdx |
| mov -0xc(%rbp),%eax | Move 12 bytes above %rbp to %eax |
| mov %eax,%esi | Move %eax to % esi |
| mov %rdx,%rdi | Move %rdx to %rdi |
| callq 0x60 <main+96> | Call 96 |
| lea 0x0(%rip),%rsi # 0x67 <main+103> | Load effective address 0(%rip) to %rsi |
| mov %rax,%rdi | Move %rax to %rdi |
| callq 0x6f <main+111> | Call 111 |
| mov %rax,%rdx | Move %rax to %rdx r |
| mov -0x4(%rbp),%eax | Move 4 bytes above %rbp to %eax r |
| mov %eax,%esi | Move %eax to %esi |
| mov %rdx,%rdi | Move %rdx to %rdi |
| callq 0x7f <main+127> | Call 127 |
| mov %rax,%rdx | Move %rax to %rdx |
| mov 0x0(%rip),%rax # 0x89 <main+137> | Move 0(%rip) to %rax |
| mov %rax,%rsi | Move %rax to %rsi |
| mov %rdx,%rdi | Move %rdx to %rdi |
| callq 0x94 <main+148> | Call 148 |
| addl $0x1,-0xc(%rbp) | Add 1 to 12 bytes above %rbp |
| jmp 0x20 <main+32> | Jump to 32 |
| addl $0x1,-0x8(%rbp) | Add 1 to 8 bytes above %rbp |
| jmpq 0xf <main+15> | Call 15 |
| mov $0x0,%eax | Move 0 to $eax |
| leaveq | Leave |
| retq | return |

**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** |
| --- | --- | --- |
| movl $0x1,-0x8(%rbp)  cmpl $0x9,-0x8(%rbp)  jmpq 0xf <main+15>  addl $0x1,-0x8(%rbp) | For ( a = 1;a <= 9; a++) | This is a loop that changes the value of “a” during each iteration |
| movl $0x1,-0xc(%rbp)  cmpl $0x9,-0xc(%rbp)  jmp 0x20 <main+32>  addl $0x1,-0xc(%rbp) | For(I = 1; I <=9; i++) | This is a loop that changes the value of “I” during each iteration |
| mov -0x8(%rbp),%eax  imul -0xc(%rbp),%eax  mov %eax,-0x4(%rbp)  mov -0x8(%rbp),%eax  mov %eax,%esi | X = a \* i | Calcualtes the product of a and I and stores it in x. |
| lea 0x0(%rip),%rdi # 0x3c <main+60>  callq 0x41 <main+65>  lea 0x0(%rip),%rsi # 0x48 <main+72>  mov %rax,%rdi  callq 0x50 <main+80> | Cout << a << “ \* “ << I << “ = ” << x << endl; | This prints out the value of a then a \* symbol, then the value of I followed by a = symbol. Then ends it with the value of X. |
| mov $0x0,%eax  leaveq  retq | Return 0; | Returns the value of zero and ends the function |

## **File Two**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| push %rbp | Push %rbp register |
| mov %rsp,%rbp | Move value at %rsp to %rbp |
| sub $0x30,%rsp | Subtract 48 from %rsp |
| mov %fs:0x28,%rax | Move %fs:40 to%rax |
| mov %rax,-0x8(%rbp) | Move %rax to 8 bytes above %rbp |
| xor %eax,%eax | Clear %eax r |
| lea 0x0(%rip),%rsi # 0x1e <main+30> | Load effective address 0(%rip) to %rsi |
| lea 0x0(%rip),%rdi # 0x25 <main+37> | Load effective address 0(%rip) to %rdi |
| callq 0x2a <main+42> | Call 42 |
| mov %rax,%rdx | Move %rax to %rdx r |
| mov 0x0(%rip),%rax # 0x34 <main+52> | Move 0(%rip) to %rax |
| mov %rax,%rsi | Move %rax to %rsi |
| mov %rdx,%rdi | Move %rdx to %rdi |
| callq 0x3f <main+63> | Call 63 |
| lea -0x14(%rbp),%rax | Load effective address 20 bytes above %rbp to %rax |
| mov %rax,%rsi | Move %rax to %rsi |
| lea 0x0(%rip),%rdi # 0x4d <main+77> | Load effective address 0(%rip) to %rdi |
| callq 0x52 <main+82> | Call 82 |
| mov -0x14(%rbp),%edx | Move 20 bytes above %rbp to %edx |
| mov -0x14(%rbp),%eax | Move 20 bytes above %rbp to %eax |
| imul %eax,%edx | Multiply %eax and %edx |
| mov -0x14(%rbp),%eax | Move 20 bytes above %rbp to %eax |
| imul %edx,%eax | Multiply %eax and %edx |
| mov %eax,-0x14(%rbp) | Move %eax to 20 bytes above %rbp |
| mov -0x14(%rbp),%eax | Move 20 bytes above %rbp to %eax |
| cvtsi2sd %eax,%xmm0 | Multiply constant value by the cube of the passed radius |
| movsd 0x0(%rip),%xmm1 # 0x73 <main+115> |
| mulsd %xmm1,%xmm0 |
| movsd %xmm0,-0x10(%rbp) |
| lea 0x0(%rip),%rsi # 0x83 <main+131> | Load effective address 0(%rip) to %rsi |
| lea 0x0(%rip),%rdi # 0x8a <main+138> | Load effective address 0(%rip) to %rdi |
| callq 0x8f <main+143> | Call 143 |
| mov %rax,%rdx | Move %rax to $rdx |
| mov -0x10(%rbp),%rax | Move 16 bytes above %rbp to %rax |
| mov %rax,-0x28(%rbp) | Move %rax to 40 bytes above %rbp |
| movsd -0x28(%rbp),%xmm0 | Move 40 bytes above %rbp to %xmm0 |
| mov %rdx,%rdi | Move %rdx to %rdi |
| callq 0xa7 <main+167> | Call 167 |
| mov $0x0,%eax | Move 0 to %eax |
| mov -0x8(%rbp),%rcx | Move 8 bytes above %rbp to %rcx |
| xor %fs:0x28,%rcx | Compares %fs:40 to %rcx |
| je 0xc0 <main+192> | Jump if equal to 192 |
| callq 0xc0 <main+192> | Calls 192 |
| leaveq | Leave |
| retq | return |

**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** |
| --- | --- | --- |
| lea 0x0(%rip),%rsi # 0x1e <main+30>  lea 0x0(%rip),%rdi # 0x25 <main+37>  callq 0x2a <main+42> | Cout << “Enter Radius:” << endl; | Prints to the console “Enter Radius:” |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 0x34 <main+52>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x3f <main+63>  lea -0x14(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x4d <main+77>  callq 0x52 <main+82> | Cin >> r; | Takes in the value input by the user and stores it in r |
| 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 0x0(%rip),%xmm1 # 0x73 <main+115>  mulsd %xmm1,%xmm0  movsd %xmm0,-0x10(%rbp) | V = 3.14 \* r \*r \* r | Calculate volume with variable r |
| lea 0x0(%rip),%rsi # 0x83 <main+131>  lea 0x0(%rip),%rdi # 0x8a <main+138>  callq 0x8f <main+143>  mov %rax,%rdx  mov-0x10(%rbp),%rax  mov %rax,-0x28(%rbp)  movsd-0x28(%rbp),%xmm0  mov %rdx,%rdi  callq 0xa7 <main+167>  mov $0x0,%eax  mov-0x8(%rbp),%rcx  xor %fs:0x28,%rcx | Cout << “Volume = ” << v << endl; | Prints out the word volume followed by the content of variable v. |
| mov $0x0,%eax  leaveq  retq | Return 0; | Returns zero, exit main |

## **File Three**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| push %rbp | Push %rbp register |
| mov %rsp,%rbp | Move %rsp to %rbp register |
| sub $0x20,%rsp | Sub 32 from %rsp |
| mov %fs:0x28,%rax | Move %fs:40 to %rax |
| mov %rax,-0x8(%rbp) | Move %rax to 8 bytes above %rbp |
| xor %eax,%eax | Clear %eax register |
| movl $0x1,-0xc(%rbp) | Move 1 to 12 bytes above %rbp |
| lea 0x0(%rip),%rsi # 0x25 <main+37> | Load effective address 0(%rip) to %rsi register |
| lea 0x0(%rip),%rdi # 0x2c <main+44> | Load effective address 0(%rip) to %rdi register |
| callq 0x31 <main+49> | Call 49 |
| mov %rax,%rdx | Move %rax to %rsi |
| mov 0x0(%rip),%rax # 0x3b <main+59> | Move 0x0(%rip) to %rax |
| mov %rax,%rsi | Move %rax to %rsi |
| mov %rdx,%rdi | Move %rdx to %rdi |
| callq 0x46 <main+70> | Call 70 |
| lea -0x18(%rbp),%rax | Load effective address 0(%rip) to %rdp register |
| mov %rax,%rsi | Move %rax to %rsi |
| lea 0x0(%rip),%rdi # 0x54 <main+84> | Load effective address 0(%rip) to %rdi register |
| callq 0x59 <main+89> | Call 89 |
| mov -0x18(%rbp),%eax | Move -0x18(%rbp) to %eax |
| sub $0x1,%eax | Subtract 1 from %eax register |
| mov %eax,-0xc(%rbp) | Move 1 to 16 bytes above %rbp |
| movl $0x1,-0x10(%rbp) | Move 24 bytes above %rbp to %eax |
| mov -0x18(%rbp),%eax |
| cmp %eax,-0x10(%rbp) | Compare %eax and 16 bytes above %rbp |
| jg 0xe3 <main+227> | Jump if greater than to 227 |
| movl $0x1,-0x14(%rbp) | Move 1 to 20 bytes above %rbp |
| mov -0x14(%rbp),%eax | Move 20 bytes above %rbp to %eax |
| cmp -0xc(%rbp),%eax | Compare 12 bytes above %rbp to %eax |
| jg 0x99 <main+153> | Jump if greater to 153 |
| lea 0x0(%rip),%rsi # 0x87 <main+135> | Load effective address 0(%rip) to %rsi |
| lea 0x0(%rip),%rdi # 0x8e <main+142> | Load effective address 0(%rip) to %rdi |
| callq 0x93 <main+147> | Call 147 |
| addl $0x1,-0x14(%rbp) | Add 1 to 20 bytes above %rbp |
| jmp 0x78 <main+120> | Jump to 120 |
| subl $0x1,-0xc(%rbp) | Subtract 1 from 12 bytes above %rbp |
| movl $0x1,-0x14(%rbp) | Move 1 to 20 bytes above %rbp |
| mov -0x10(%rbp),%eax | Move 16 bytes above %rbp to %eax |
| add %eax,%eax | Add %eax and %eax |
| sub $0x1,%eax | Subtract 1 from %eax register |
| cmp %eax,-0x14(%rbp) | Compare %eax and 20 bytes above %rbp |
| jg 0xca <main+202> | Jump if greater t0 202 |
| lea 0x0(%rip),%rsi # 0xb8 <main+184> | Load effective address 0(%rip) to %rsi |
| lea 0x0(%rip),%rdi # 0xbf <main+191> | Load effective address 0(%rip) to %rdi |
| callq 0xc4 <main+196> | Call 196 |
| addl $0x1,-0x14(%rbp) | Add 1 to 20 bytes above %rbp |
| jmp 0xa4 <main+164> | Jump to 164 |
| lea 0x0(%rip),%rsi # 0xd1 <main+209> | Load effective address 0(%rip) to %rsi |
| lea 0x0(%rip),%rdi # 0xd8 <main+216> | Load effective address 0(%rip) to %rdi |
| callq 0xdd <main+221> | Call 221 |
| addl $0x1,-0x10(%rbp) | Add 1 to 16 bytes above %rbp |
| jmp 0x69 <main+105> | Jump to 105 |
| movl $0x1,-0xc(%rbp) | Move 1 to 12 bytes above %rbp |
| movl $0x1,-0x10(%rbp) | Move 1 to 16 bytes above %rbp |
| mov -0x18(%rbp),%eax | Move 24 bytes above %rbp to %eax |
| sub $0x1,%eax | Subtract 1 from %eax |
| cmp %eax,-0x10(%rbp) | Compare %eax and 16 bytes above %rbp |
| jg 0x171 <main+369> | Jump if greater to 369 |
| movl $0x1,-0x14(%rbp) | Move 1 to 20 bytes above %rbp |
| mov -0x14(%rbp),%eax | Move 20 bytes above %rbp to %eax |
| cmp -0xc(%rbp),%eax | Move 20 bytes above %rbp to %eax |
| jg 0x124 <main+292> | Jump if greater to 292 |
| lea 0x0(%rip),%rsi # | Load effective address 0(%rip) to %rsi |
| lea 0x0(%rip),%rdi | Load effective address 0(%rip) to %rdi |
| callq 0x11e <main+286> | Call 286 |
| addl $0x1,-0x14(%rbp) | Add 1 to 20 bytes above %rbp |
| jmp 0x103 <main+259> | Jump to 259 |
| addl $0x1,-0xc(%rbp) | Add 1 to 20 bytes above %rbp |
| movl $0x1,-0x14(%rbp) | Move 1 to 20 bytes above %rbp |
| mov -0x18(%rbp),%eax | Move 24 bytes above %rbp to %eax |
| sub -0x10(%rbp),%eax | Subtract 16 bytes above %rbp from %eax |
| add %eax,%eax | Add %eax and %eax |
| sub $0x1,%eax | Subtract 1 from %eax |
| cmp %eax,-0x14(%rbp) | Compare %eax and 20 bytes above %rbp |
| jg 0x158 <main+344> | Jump if greater to 344 |
| lea 0x0(%rip),%rsi # 0x146 <main+326> | Load effective address 0(%rip) to %rsi |
| lea 0x0(%rip),%rdi # 0x14d <main+333> | Load effective address 0(%rip) to %rdi |
| callq 0x152 <main+338> | Call 338 |
| addl $0x1,-0x14(%rbp) | Add 1 to 20 bytes above %rbp |
| jmp 0x12f <main+303> | Jump to 303 |
| lea 0x0(%rip),%rsi # 0x15f <main+351> | Load effective address 0(%rip) to %rsi |
| lea 0x0(%rip),%rdi # 0x166 <main+358> | Load effective address 0(%rip) to %rdi |
| callq 0x16b <main+363> | Call 363 |
| addl $0x1,-0x10(%rbp) | Add 1 to 16 bytes above %rbp |
| jmp 0xf1 <main+241> | Jump to 241 |
| mov $0x1,%eax | Move 1 to %eax |
| mov -0x8(%rbp),%rcx | Move 8 bytes above %rbp to %rcx |
| xor %fs:0x28,%rcx | Compare %fs to %rcx |
| je 0x18a <main+394> | Jump to 394 |
| callq 0x18a <main+394> | Call 394 |
| leaveq | Leave |
| retq | return |

**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** |
| --- | --- | --- |
| lea 0x0(%rip),%rsi  lea 0x0(%rip),%rdi # 0x2c  callq 0x31  mov %rax,%rdx  mov 0x0(%rip),%rax #  mov %rax,%rsi  mov %rdx,%rdi  callq 0x46 <main+70>  lea 0x0(%rip),%rsi # 0x25 lea 0x0(%rip),%rdi # 0x2c callq 0x31 mov %rax,%rdx mov 0x0(%rip),%rax # 0x3b <main+59> mov %rax,%rsi mov %rdx,%rdi callq 0x46 <main+70> | Cout << “Enter a number of rows” << endl;  Cin >> rows; | Print message to console and take in user input for rows |
| movl $0x1,-0xc(%rbp)  sub $0x1,%eax  mov %eax,-0xc(%rbp)  movl $0x1,-0x10(%rbp)  mov-0x18(%rbp),%eax | Star = 1;  Spaces = rows – 1; | Give stars value of 1 and spaces value of one less then rows |
| cmp %eax,-0x10(%rbp)  jg 0xe3 <main+227>  movl $0x1,-0x14(%rbp)  mov-0x14(%rbp),%eax  addl $0x1,-0x14(%rbp)  jmp 0x78 <main+120> | for(i=1; i<rows\*2;i++) | For loop to iterate through the rows |
| cmp-0xc(%rbp),%eax jg 0x99 <main+153> | for(j=1; j <= spaces;j++) | For loop to iterate through spaces |
| lea 0x0(%rip),%rsi # 0x87  lea 0x0(%rip),%rdi # 0x8e  callq 0x93 <main+147 | Cout << “ “; | Print space |
| Cmp -0xc(%rbp),%eax  jg 0x124 <main+292> | for(j=1;j<stars\*2;j++) | For loop to iterate through stars |
| lea 0x0(%rip),%rsi # 0x112  lea 0x0(%rip),%rdi # 0x119  callq 0x11e <main+286> | Cout << “\*”; | Print \* |
| <+175>: jg 0xca <main+202> | If (I <rows) | Check if I is less then rows |
| subl $0x1,-0xc(%rbp) movl $0x1,-0x14(%rbp)  mov-0x10(%rbp),%eax  add %eax,%eax | Space--;  Stars++; | Subtract 1 from spaces while adding 1 to stars. |
| jg 0xca <main+202> | else | Else statement |
| addl $0x1,-0xc(%rbp)  movl $0x1,-0x14(%rbp)  mov -0x18(%rbp),%eax  sub-0x10(%rbp),%eax  add %eax,%eax  sub $0x1,%eax | Spaces++;  Stars--; | Add one to spaces while subtracting one from stars |
| mov $0x1,%eax  leaveq  retq | Return 1 | Returns one and program exits |

## **File Four**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| push %rbp | Push %rbp register |
| mov %rsp,%rbp | Move %rsp to %rbp |
| sub $0x30,%rsp | Subtract 48 and %rsp |
| mov %fs:0x28,%rax | Move %fs:40,%rax |
| mov %rax,-0x8(%rbp) | Move %rax to 8 bytes above %rbp |
| xor %eax,%eax | Clear %eax |
| movq $0x0,-0x20(%rbp) | Move 0 to 20 bytes above %rbp |
| movq $0x1,-0x18(%rbp) | Move 1 to 24 bytes above %rbp |
| lea 0x0(%rip),%rsi # 0x2e <main+46> | Load effective address 0(%rip) to %rsi |
| lea 0x0(%rip),%rdi # 0x35 <main+53> | Load effective address 0(%rip) to %rdi |
| callq 0x3a <main+58> | Call 58 |
| mov %rax,%rdx | Move %rax to %rdx |
| mov 0x0(%rip),%rax # 0x44 <main+68> | Move 0(%rip) to %rax |
| mov %rax,%rsi | Move %rax to %rsi |
| mov %rdx,%rdi | Move %rdx to %rdi |
| callq 0x4f <main+79> | Call 79 |
| lea -0x28(%rbp),%rax | Load effective address 40 bytes above %rbp to $rax |
| mov %rax,%rsi | Move %rax to %rsi |
| lea 0x0(%rip),%rdi # 0x5d <main+93> | Move %rax to %rsi |
| callq 0x62 <main+98> | Call 98 |
| mov -0x28(%rbp),%rax | Move 40 bytes above %rbp to %rax |
| test %rax,%rax | Compare %rax and %rax |
| je 0xf2 <main+242> | Jump if equal to 242 |
| mov -0x28(%rbp),%rcx | Move 40 bytes above %rbp to %rax |
| movabs $0x6666666666666667,%rdx | Move 0x6666666666666667 to %rdx |
| mov %rcx,%rax | Move %rcx to %rax |
| imul %rdx | Multiply %rdx |
| sar $0x2,%rdx | Shift %rdx two bytes |
| mov %rcx,%rax | Move %rcx to %rax |
| sar $0x3f,%rax | Shift %rax 63 bytes |
| sub %rax,%rdx | Subtract %rax and %rdx |
| mov %rdx,%rax | Move %rdx to %rax |
| mov %rax,-0x10(%rbp) | Move %rax to 16 bytes above %rbp |
| mov -0x10(%rbp),%rdx | Move 16 bytes above %rbp register to %rdx |
| mov %rdx,%rax | Move %rdx register to %rax |
| shl $0x2,%rax | Shift %rax register by 2 bytes |
| add %rdx,%rax | Add %rdx register and %rax |
| add %rax,%rax | Add %rax register and %rax |
| sub %rax,%rcx | Subtract %rax register from %rcx |
| mov %rcx,%rax | Move %rcx to %rax |
| mov %rax,-0x10(%rbp) | Move %rax to 16 bytes above %rbp |
| mov -0x10(%rbp),%rax | Move 16 bytes above %rbp to %rax |
| imul -0x18(%rbp),%rax | Multipy 24 bytes above %rbp and %rax |
| add %rax,-0x20(%rbp) | Add %rax and 32 bytes above %rbp |
| shlq -0x18(%rbp) | Shift 24 bytes above %rbp |
| mov -0x28(%rbp),%rcx | Move 40 bytes above %rbp to %rcx |
| movabs $0x6666666666666667,%rdx | Move 0x6666666666666667 to %rdx |
| mov %rcx,%rax | Move %rcx to %rax |
| imul %rdx | Multiply %rdx |
| sar $0x2,%rdx | Shift %rdx 2 bytes |
| mov %rcx,%rax | Move %rcx to %rax |
| sar $0x3f,%rax | Shift %rax 63 bytes |
| sub %rax,%rdx | Subtract %rax from %rdx |
| mov %rdx,%rax | Move %rdx to %rax |
| mov %rax,-0x28(%rbp) | Move %rax to 40 bytes above %rbp |
| jmpq 0x62 <main+98> | Jump to 98 |
| lea 0x0(%rip),%rsi # 0xf9 <main+249> | Load effective address 0(%rip) to %rsi |
| lea 0x0(%rip),%rdi # 0x100 <main+256> | Load effective address 0(%rip) to %rdi |
| callq 0x105 <main+261> | Call 261 |
| mov %rax,%rdx | Move %rax to %rdx |
| mov -0x20(%rbp),%rax | Move 32 bytes above %rbp to %rax |
| mov %rax,%rsi | Move %rax to %rsi register |
| mov %rdx,%rdi | Move %rdx to %rdi register |
| callq 0x117 <main+279> | Call 279 |
| mov %rax,%rdx | Move %rax to %rdx |
| mov 0x0(%rip),%rax # 0x121 <main+289> | Move 0(%rip) to %rax r |
| mov %rax,%rsi | Move %rax to %rsi |
| mov %rdx,%rdi | Move %rdx to %rdi |
| callq 0x12c <main+300> | Call 300 |
| mov $0x0,%eax | Move 0 to %eax |
| mov -0x8(%rbp),%rsi | Move 8 bytes above %rbp to %rsi r |
| xor %fs:0x28,%rsi | Compare %fs:40 to %rsi r |
| je 0x145 <main+325> | Jump if equal to 325 |
| callq 0x145 <main+325> | Call 325 |
| leaveq | Leave |
| retq | Return |

**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** |
| --- | --- | --- |
| movq $0x0,-0x20(%rbp) movq $0x1,-0x18(%rbp) | Long int bin.  Hex = 02.  I = 1.  Remainder; | Declares integers and sets the value for hex and i |
| lea 0x0(%rip),%rsi # 0x2e lea 0x0(%rip),%rdi # 0x35 callq 0x3a <main+58> | Cout << “Enter the binary number:” << endl; | Putputs to console “Enter the binary number:” |
| mov %rax,%rdx  mov 0x0(%rip),%rax  mov %rax,%rsi  mov %rdx,%rdi  callq 0x4f  lea -0x28(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi  callq 0x62  mov -0x28(%rbp),%rax | Cin >> bin; | Gest user input |
| test %rax,%rax je 0xf2  jmpq 0x62 | While(bin!=0) | while bin does not equal zero, stay in loop |
| 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 = bin % 10; | Stores value of bun mod 10 into the remainder variable |
| mov-0x10(%rbp),%rax  imul-0x18(%rbp),%rax  add %rax,-0x20(%rbp) | Hex = hex + remainder \* I; | Stores value of hex + remainder multiplied by I onto hex |
| shlq -0x18(%rbp) | I = i\*2; | Set I to I times 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) | Bin = bin/10 | Set bin to bin divided by 10 |
| lea 0x0(%rip),%rsi # 0xf9 <main+249>  lea 0x0(%rip),%rdi # 0x100  callq 0x105  mov %rax,%rdx  mov-0x20(%rbp),%rax  mov %rax,%rsi  mov %rdx,%rdi  callq 0x117  mov %rax,%rdx  mov 0x0(%rip),%rax # 0x121  mov %rax,%rsi  mov %rdx,%rdi  callq 0x12c | Cout << “equivalent hexadecimal value:” << hex << endl; | Output "Equivalent hexadecimal value: " followed by the value at hex onto the console |
| mov $0x0,%eax  leaveq  retq | Return 0; | Returns 0 and exits main function |