Binary to C++ Activity

CS-410-R4890 Software Reserve Engineering

Raphael Coloma

March 29, 2024

# **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 | Stack Management for main function |
| sub $0x10,%rsp | Clear 10 bytes from %rsp to reserve variables |
| movl $0x1,-0x8(%rbp)  cmpl $0x9,-0x8(%rbp)  jg 0xa3 <main+163> | Loop condition  Move value 1 to 8 bytes above register %rbp. Compare value 9 to value at 8 bytes above register %rbp. Jump to main+163 if greater |
| movl $0x1,-0xc(%rbp)  cmpl $0x9,-0xc(%rbp)  jg 0x9a <main+154> | Loop condition  Move value 1 to 8 bytes above register %rbp. Compare value 9 to value at 8 bytes above register %rbp. Jump to main+154 if greater |
| mov -0x8(%rbp),%eax  imul -0xc(%rbp),%eax  mov %eax,-0x4(%rbp)  mov -0x8(%rbp),%eax  mov %eax,%esi | Move value at 8 bytes above register %rbp to register %eax. Multiple value at 12 bytes above register %rbp by the value in register %eax and store the result in register %eax. Move the result to 4 bytes above register %rbp. Move value at 8 bytes above register %rbp to register %eax. Move value from register %eax to register %esi. |
| lea 0x0(%rip),%rdi # 0x3c <main+60>  callq 0x41 <main+65> | Put memory address of 0(%rip) into register %rdi. Print string to screen |
| lea 0x0(%rip),%rsi # 0x48 <main+72>  mov %rax,%rdi  callq 0x50 <main+80> | Put memory address of 0(%rip) into register %rsi. Move value from register %rax to register %rdi. Print string to screen |
| mov %rax,%rdx  mov -0xc(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 0x60 <main+96> | Move value from register %rax to register %rdx. Move value from 12 bytes above register %rbp to register %eax. Move value from register %eax to register %esi. Move value from register %rdx to register %rdi. Print value to screen |
| lea 0x0(%rip),%rsi # 0x67 <main+103>  mov %rax,%rdi  callq 0x6f <main+111> | Put memory address of 0(%rip) into register %rsi. Move value from register %rax to register %rdi. Print string to screen |
| mov %rax,%rdx  mov -0x4(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 0x7f <main+127> | Move value from register %rax to register %rdx. Move value from 4 bytes above register %rbp to register %eax. Move value from register %eax to register %esi. Move value from register %rdx to register %rdi. Print value to screen |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 0x89 <main+137>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x94 <main+148> | Move value from register %rax to register %rdx. Move value from register %rip to register %rax. Move value from register %rax to register %rsi. Move value from register %rdx to register %rdi. Print value to screen |
| addl $0x1,-0xc(%rbp)  jmp 0x20 <main+32> | Add the value 1 to the value at 12 bytes above register %rbp. Store result at 12 bytes above register %rbp. Jump to main+32 |
| addl $0x1,-0x8(%rbp)  jmpq 0xf <main+15> | Add the value 1 to the value at 8 bytes above register %rbp. Store result at 8 bytes above register %rbp. Jump to main+15 |
| mov $0x0,%eax  leaveq  retq | Move value 0 into register %eax  Return value and terminate function. |

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

int main()

{

int number, i, a, x;

for (a = 1; a <= 9; a++)

{

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

{

x = a \* i;

cout << a << " \* " << i << " = " << x << endl;

}

}

return 0;

}

**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 | int main()  { | Initialize main function |
| sub $0x10,%rsp | int number, i, a, x; | Declare integer variables number, i, a, and x. |
| movl $0x1,-0x8(%rbp)  cmpl $0x9,-0x8(%rbp)  jg 0xa3 <main+163>  …  addl $0x1,-0xc(%rbp)  jmp 0x20 <main+32> | for (a = 1; a <= 9; a++)  {  …  } | For Loop Condition  Initialize variable a to value 1. Compare the value stored in a to value 9. Continue loop if the value stored in variable a is less than value 9. Increment the value stored in a at the end of loop iteration. |
| movl $0x1,-0xc(%rbp)  cmpl $0x9,-0xc(%rbp)  jg 0x9a <main+154>  …  addl $0x1,-0x8(%rbp)  jmpq 0xf <main+15> | for (i = 1; i <= 9; i++)  {  …  } | (Inside first For Loop)  For Loop Condition  Initialize variable i to value 1. Compare the value stored in i to value 9. Continue loop if the value stored in variable i is less than value 9. Increment the value stored in i at the end of loop iteration. |
| mov -0x8(%rbp),%eax  imul -0xc(%rbp),%eax  mov %eax,-0x4(%rbp) | x = a \* i; | (Inside second For Loop)  Multiply the value stored in variable a by the value stored in variable i. Store the result in variable x |
| lea 0x0(%rip),%rdi  callq 0x41 <main+65>  lea 0x0(%rip),%rsi  mov %rax,%rdi  callq 0x50 <main+80>  mov %rax,%rdx  mov -0xc(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 0x60 <main+96>  lea 0x0(%rip),%rsi  mov %rax,%rdi  callq 0x6f <main+111>  mov %rax,%rdx  mov -0x4(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 0x7f <main+127>  mov %rax,%rdx  mov 0x0(%rip),%rax  mov %rax,%rsi  mov %rdx,%rdi  callq 0x94 <main+148> | cout << a << " \* " << i << " = " << x << endl; | (Inside second For Loop)  Print to screen the value stored in variable a. Then the string “ \* “. Then print to screen the value stored in variable i. Followed by the string “ = “. Next print to screen the value stored in variable x. Lastly print to screen a newline character. |
| mov $0x0,%eax  leaveq  retq | return 0;  } | Return 0 to signify a successful execution and terminate the main function. |

## **File Two**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| push %rbp  mov %rsp,%rbp  sub $0x30,%rsp | Stack Management for main function |
| mov %fs:0x28,%rax  mov %rax,-0x8(%rbp) | Store sentinel stack-guard value for double data type variables. |
| xor %eax,%eax | Exclusive or register %eax by itself to set it to 0. |
| lea 0x0(%rip),%rsi # 0x1e <main+30>  lea 0x0(%rip),%rdi # 0x25 <main+37>  callq 0x2a <main+42> | Put memory address of 0(%rip) into register %rsi. Put memory address of 0(%rip) into register %rdi Print string to screen |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 0x34 <main+52>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x3f <main+63> | Move value from register %rax to register %rdx. Move value from register 0(%rip) to register %rax. Move value from register %rax to register %rsi. Move value from register %rdx to register %rdi. Print string to screen. |
| lea -0x14(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x4d <main+77>  callq 0x52 <main+82> | Put memory address of 14 bytes above register %rbp into register %rax. Move value from register %rax to register %rsi. Put memory address of 0(%rip) into register %rdi. Call cin function for user data |
| mov -0x14(%rbp),%edx  mov -0x14(%rbp),%eax  imul %eax,%edx  mov -0x14(%rbp),%eax  imul %edx,%eax | Move the value 14 bytes above register %rbp to register %edx. Move the value 14 bytes above register %rbp to register %eax. Multiply the value in register %eax by the value in register %edx and store the result in register %edx. Move the value 14 bytes above register %rbp to register %eax. Multiply the value in register %eax by the value in register %edx and store the result in register %eax. |
| mov %eax,-0x14(%rbp)  mov -0x14(%rbp),%eax  cvtsi2sd %eax,%xmm0 | Move the value in register %eax to 14 bytes above register %rbp. Move the value at 14 bytes above register %rbp to register %eax. Convert the value in register %eax to a floating-point value and store the double value in register xmm0. |
| movsd 0x0(%rip),%xmm1 # 0x73 <main+115>  mulsd %xmm1,%xmm0  movsd %xmm0,-0x10(%rbp) | Move value from register %rip to register %xmm1. Multiply value from register %xmm1 by value in register %xmm0 and store the result in register %xmm0. Move the value from register %xmm0 to 10 bytes above register %rbp |
| lea 0x0(%rip),%rsi # 0x83 <main+131>  lea 0x0(%rip),%rdi # 0x8a <main+138>  callq 0x8f <main+143> | Put memory address of 0(%rip) into register %rsi. Put memory address of 0(%rip) into register %rdi Print string to screen |
| mov %rax,%rdx  mov -0x10(%rbp),%rax  mov %rax,-0x28(%rbp)  movsd -0x28(%rbp),%xmm0  mov %rdx,%rdi  callq 0xa7 <main+167> | Move value from register %rax to register %rdx. Move value from 10 bytes above register %rbp to register %rax. Move value from register %rax to 28 bytes above register %rbp. Move value at 28 bytes above %rdx to register %xmm0 as a double. Move value from register %rdx to register %rdi. Print value to screen. |
| mov $0x0,%eax  mov -0x8(%rbp),%rcx | Move value 0 to register %eax. Move value at 8 bytes above register %rbp to register %rcx. |
| xor %fs:0x28,%rcx  je 0xc0 <main+192> | Error check to see if the stored value on the stack was changed. If they are equal, jump to main+192 |
| callq 0xc0 <main+192> | Display stack check fail |
| leaveq  retq | Return 0 to signify a successful. Terminate the main function. |

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

int main()

{

double Pi = 3.14;

int radius, volume;

cout << "Enter Radius: " << endl;

cin >> radius;

volume = (radius \* radius \* radius) \* Pi;

cout << “The volume is: “ << volume;

return 0;

}

**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 $0x30,%rsp | int main()  { | Initialize main function |
| mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax | double Pi = 3.14;  int radius, volume; | Declare double variable Pi and initialize it to 3.14. Declare integer variables radius and volume. |
| lea 0x0(%rip),%rsi  lea 0x0(%rip),%rdi  callq 0x2a <main+42>  mov %rax,%rdx  mov 0x0(%rip),%rax  mov %rax,%rsi  mov %rdx,%rdi  callq 0x3f <main+63> | cout << "Enter Radius: " << endl; | Print to screen string “Enter Radius: ” followed by a newline character. |
| lea -0x14(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi  callq 0x52 <main+82> | cin >> radius; | Accept user input and store the value in variable radius. |
| 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  mulsd %xmm1,%xmm0  movsd %xmm0,-0x10(%rbp) | volume = (radius \* radius \* radius) \* Pi; | Cube the value in variable radius and multiply it by the value in variable Pi. Store the result in variable volume. |
| lea 0x0(%rip),%rsi  lea 0x0(%rip),%rdi  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> | cout << “The volume is: “ << volume; | Print to screen the string “The volume is: “ followed by the value in variable volume. No newline character is called. |
| mov $0x0,%eax  mov -0x8(%rbp),%rcx  xor %fs:0x28,%rcx  je 0xc0 <main+192>  callq 0xc0 <main+192>  leaveq  retq | return 0;  } | Return 0 to signify a successful execution and terminate the main function. |

## **File Three**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| push %rbp  mov %rsp,%rbp  sub $0x20,%rsp | Stack Management for main function |
| mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax | Store sentinel stack-guard value for double data type variables. Exclusive or register %eax by itself to set it to 0. |
| movl $0x1,-0xc(%rbp) | Move long value 1 to 12 bytes above register %rbp. |
| lea 0x0(%rip),%rsi # 0x25 <main+37>  lea 0x0(%rip),%rdi # 0x2c <main+44>  callq 0x31 <main+49> | Put memory address of 0(%rip) into register %rsi. Put memory address of 0(%rip) into register %rdi Print string to screen |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 0x3b <main+59>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x46 <main+70> | Move value from register %rax to register %rdx. Move value from register 0(%rip) to register %rax. Move value from register %rax to register %rsi. Move value from register %rdx to register %rdi. Print string to screen. |
| lea -0x18(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x54 <main+84>  callq 0x59 <main+89> | Put memory address of 18 bytes above register %rbp into register %rax. Move value from register %rax to register %rsi. Put memory address of 0(%rip) into register %rdi. Call cin function for user data |
| mov -0x18(%rbp),%eax  sub $0x1,%eax  mov %eax,-0xc(%rbp) | Move value at 18 bytes above register %rbp to register %eax. Subtract value 1 from value stored in register %eax. Store result in register %eax. Move value in register %eax to 12 bytes above register %rbp. |
| movl $0x1,-0x10(%rbp)  mov -0x18(%rbp),%eax  cmp %eax,-0x10(%rbp)  jg 0xe3 <main+227> | Move long value 1 at 10 bytes above register %rbp. Move value at 18 bytes above register %rbp to register %eax. Compare values stored at register %eax and at 10 bytes above register %rbp. Jump to main+227 if the value in register %eax is greater than the value at 10 bytes above register %rbp. |
| movl $0x1,-0x14(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x99 <main+153> | Move long value 1 at 14 bytes above register %rbp. Move value at 14 bytes above register %rbp to register %eax. Compare values stored at register %eax and at 12 bytes above register %rbp. Jump to main+153 if the value at 12 bytes above register %rbp is greater than the value in register %eax. |
| lea 0x0(%rip),%rsi # 0x87 <main+135>  lea 0x0(%rip),%rdi # 0x8e <main+142>  callq 0x93 <main+147> | Put memory address of 0(%rip) into register %rsi. Put memory address of 0(%rip) into register %rdi Print string to screen |
| addl $0x1,-0x14(%rbp)  jmp 0x78 <main+120> | Add value 1 to the value at 14 bytes above register %rbp. Jump to main+120 |
| subl $0x1,-0xc(%rbp)  movl $0x1,-0x14(%rbp)  mov -0x10(%rbp),%eax  add %eax,%eax | Subtract value 1 from value at 12 bytes above register %rbp. Move lobe value 1 to 14 bytes above register %rbp. Move value at 10 bytes above register %rbp to register %eax. Add value store in register %eax to itself and store result in register %eax. |
| sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0xca <main+202> | Loop conditon  Subtract value 1 from value stored in register %eax. Compare value stored in register %eax to the value at 14 bytes above register %rbp. Jump to main+202 if the value in register %eax is greater than the value at 14 bytes above register %rbp. |
| lea 0x0(%rip),%rsi # 0xb8 <main+184>  lea 0x0(%rip),%rdi # 0xbf <main+191>  callq 0xc4 <main+196> | Put memory address of 0(%rip) into register %rsi. Put memory address of 0(%rip) into register %rdi Print string to screen |
| addl $0x1,-0x14(%rbp)  jmp 0xa4 <main+164> | Add value 1 to the value at 14 bytes above register %rbp. Jump to main+164 |
| lea 0x0(%rip),%rsi # 0xd1 <main+209>  lea 0x0(%rip),%rdi # 0xd8 <main+216>  callq 0xdd <main+221> | Put memory address of 0(%rip) into register %rsi. Put memory address of 0(%rip) into register %rdi Print string to screen |
| addl $0x1,-0x10(%rbp)  jmp 0x69 <main+105> | Add value 1 to the value at 10 bytes above register %rbp. Jump to main+105 |
| movl $0x1,-0xc(%rbp)  movl $0x1,-0x10(%rbp)  mov -0x18(%rbp),%eax | Move long value 1 to 12 bytes above register %rbp. Move long value 1 to 10 bytes above register %rbp. Move value at 18 bytes above register %rbp to register %eax |
| sub $0x1,%eax  cmp %eax,-0x10(%rbp)  jg 0x171 <main+369> | Loop condition  Subtract value 1 from value stored in register %eax. Compare value stored in register %eax to the value at 10 bytes above register %rbp. Jump to main+369 if the value in register %eax is greater than the value at 14 bytes above register %rbp. |
| movl $0x1,-0x14(%rbp)  mov -0x14(%rbp),%eax | Move long value 1 to 14 bytes above register %rbp. Move value at 14 bytes above register %rbp to register %eax |
| cmp -0xc(%rbp),%eax  jg 0x124 <main+292> | Compare the value at 12 bytes above register %rbp to the value stored in register %eax. Jump to main+292 if the value at 14 bytes above register %rbp is greater than the value in register %eax. |
| lea 0x0(%rip),%rsi # 0x112 <main+274>  lea 0x0(%rip),%rdi # 0x119 <main+281>  callq 0x11e <main+286> | Put memory address of 0(%rip) into register %rsi. Put memory address of 0(%rip) into register %rdi Print string to screen |
| addl $0x1,-0x14(%rbp)  jmp 0x103 <main+259> | Add value 1 to the value at 14 bytes above register %rbp. Jump to main+259 |
| addl $0x1,-0xc(%rbp)  movl $0x1,-0x14(%rbp)  mov -0x18(%rbp),%eax  sub -0x10(%rbp),%eax  add %eax,%eax | Add value 1 to value at 12 bytes above register %rbp. Move long value 1 to 14 bytes above register %rbp. Move value at 18 bytes above register %rbp to register %eax. Subtract value at 10 bytes above register %rbp from value in register %eax and store result in register %eax. Add value store in register %eax to itself and store result in register %eax. |
| sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0x158 <main+344> | Loop condition  Subtract value 1 from value stored in register %eax. Compare value stored in register %eax to the value at 14 bytes above register %rbp. Jump to main+369 if the value in register %eax is greater than the value at 14 bytes above register %rbp. |
| lea 0x0(%rip),%rsi # 0x146 <main+326>  lea 0x0(%rip),%rdi # 0x14d <main+333>  callq 0x152 <main+338> | Put memory address of 0(%rip) into register %rsi. Put memory address of 0(%rip) into register %rdi Print string to screen |
| addl $0x1,-0x14(%rbp)  jmp 0x12f <main+303> | Add value 1 to the value at 14 bytes above register %rbp. Jump to main+303 |
| lea 0x0(%rip),%rsi # 0x15f <main+351>  lea 0x0(%rip),%rdi # 0x166 <main+358>  callq 0x16b <main+363> | Put memory address of 0(%rip) into register %rsi. Put memory address of 0(%rip) into register %rdi Print string to screen |
| addl $0x1,-0x10(%rbp)  jmp 0xf1 <main+241> | Add value 1 to the value at 10 bytes above register %rbp. Jump to main+241 |
| mov $0x1,%eax  mov -0x8(%rbp),%rcx | Move value 1 to register %eax. Move value at 8 bytes above register %rbp to register %rcx. |
| xor %fs:0x28,%rcx  je 0x18a <main+394> | Error check to see if the stored value on the stack was changed. If they are equal, jump to main+394 |
| callq 0x18a <main+394> | Display stack check fail |
| leaveq  retq | Terminate the main function. |

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

int main()

{

int i, j, rows;

i = 1;

cout << "Enter number of rows" << endl;

cin >> rows;

for (i; i <= rows; i++)

{

for (j = 1; j <= rows - i; j++)

cout << " ";

for (j = 1; j <= 2 \* i - 1; j++)

cout << "\*";

cout << endl;

}

for (i = rows - 1; i >= 1; i--)

{

for (j = 1; j <= rows - i; j++)

cout << " ";

for (j = 1; j <= 2 \* i - 1; j++)

cout << "\*";

cout << endl;

}

return 0;

}

**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 $0x20,%rsp | int main()  { | Initialize main function |
| mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax | int i, j, rows; | Declare integer variables i, j, and rows. |
| movl $0x1,-0xc(%rbp) | i = 1; | Initialize variable i to 1 |
| lea 0x0(%rip),%rsi # 0x25 <main+37>  lea 0x0(%rip),%rdi # 0x2c <main+44>  callq 0x31 <main+49>  mov %rax,%rdx  mov 0x0(%rip),%rax # 0x3b <main+59>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x46 <main+70> | cout << "Enter number of rows" << endl; | Print to screen string “Enter number of rows” followed by a newline character. |
| lea -0x18(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x54 <main+84>  callq 0x59 <main+89> | cin >> rows; | Accept user input and store the value in variable rows. |
| mov -0x18(%rbp),%eax  sub $0x1,%eax  mov %eax,-0xc(%rbp)  movl $0x1,-0x14(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x99 <main+153>  …  addl $0x1,-0x14(%rbp)  jmp 0x78 <main+120> | for (i = rows - 1; i >= 1; i--)  {  …  } | For Loop Conditions  The value in variable i is equal to the value in variable rows minus 1. The loop will continue while the value stored in variable i is greater than or equal to 1. At the end of each loop iteration subtract 1 from the value stored in variable i. |
| lea 0x0(%rip),%rsi # 0x87 <main+135>  lea 0x0(%rip),%rdi # 0x8e <main+142>  callq 0x93 <main+147> | cout << " "; | Print to screen string “ “. |
| subl $0x1,-0xc(%rbp)  movl $0x1,-0x14(%rbp)  mov -0x10(%rbp),%eax  add %eax,%eax  sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0xca <main+202>  …  addl $0x1,-0x14(%rbp)  jmp 0xa4 <main+164> | for (j = 1; j <= 2 \* i - 1; j++)  {  …  } | For Loop Conditions  The value in variable j is equal to the value 1. The loop will continue while the value stored in variable j is less than or equal to 2 times the value in variable i minus 1. At the end of each loop iteration add 1 to the value stored in variable j. |
| lea 0x0(%rip),%rsi # 0xb8 <main+184>  lea 0x0(%rip),%rdi # 0xbf <main+191>  callq 0xc4 <main+196> | cout << "\*"; | Print to screen string “x“. |
| lea 0x0(%rip),%rsi # 0xd1 <main+209>  lea 0x0(%rip),%rdi # 0xd8 <main+216>  callq 0xdd <main+221> | cout << endl; | Print to screen a newline character. |
| movl $0x1,-0xc(%rbp)  movl $0x1,-0x10(%rbp)  mov -0x18(%rbp),%eax  sub $0x1,%eax  cmp %eax,-0x10(%rbp)  jg 0x171 <main+369>  …  addl $0x1,-0x10(%rbp)  jmp 0xf1 <main+241> | for (j = 1; j <= rows - i; j++)  {  …  } | For Loop Conditions  The value in variable j is equal to the value 1. The loop will continue while the value stored in variable j is less than or equal to the value in the variable rows minus 1. At the end of each loop iteration add 1 to the value stored in variable j. |
| movl $0x1,-0x14(%rbp)  mov -0x14(%rbp),%eax  cmp -0xc(%rbp),%eax  jg 0x124 <main+292>  …  addl $0x1,-0x14(%rbp)  jmp 0x103 <main+259> | for (i ; i <= rows; i++)  {  …  } | For Loop Conditions  The loop will continue while the value stored in variable i is less than or equal to the value in variable rows. At the end of each loop iteration add 1 to the value stored in variable i. |
| lea 0x0(%rip),%rsi # 0x112 <main+274>  lea 0x0(%rip),%rdi # 0x119 <main+281>  callq 0x11e <main+286> | cout << " "; | Print to screen string “ “. |
| addl $0x1,-0xc(%rbp)  movl $0x1,-0x14(%rbp)  mov -0x18(%rbp),%eax  sub -0x10(%rbp),%eax  add %eax,%eax  …  addl $0x1,-0x14(%rbp)  jmp 0x12f <main+303> | for (j = 1; j <= 2 \* i - 1; j++)  {  …  } | For Loop Conditions  The value in variable j is equal to the value 1. The loop will continue while the value stored in variable j is less than or equal to 2 times the value in variable i minus 1. At the end of each loop iteration add 1 to the value stored in variable j. |
| sub $0x1,%eax  cmp %eax,-0x14(%rbp)  jg 0x158 <main+344>  …..  addl $0x1,-0x10(%rbp)  jmp 0x69 <main+105> | for (j = 1; j <= rows - i; j++)  {  …  } | For Loop Conditions  The value in variable j is equal to the value 1. The loop will continue while the value stored in variable j is less than or equal to the value in variable rows minus 1. At the end of each loop iteration add 1 to the value stored in variable j. |
| lea 0x0(%rip),%rsi # 0x146 <main+326>  lea 0x0(%rip),%rdi # 0x14d <main+333>  callq 0x152 <main+338> | cout << "\*"; | Print to screen string “x“. |
| lea 0x0(%rip),%rsi # 0x15f <main+351>  lea 0x0(%rip),%rdi # 0x166 <main+358>  callq 0x16b <main+363> | cout << endl; | Print to screen a newline character. |
| mov $0x1,%eax  mov -0x8(%rbp),%rcx  xor %fs:0x28,%rcx  je 0x18a <main+394>  callq 0x18a <main+394>  leaveq  retq | return 0;  } | Return 0 to signify a successful execution and terminate the main function. |

## **File Four**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| push %rbp  mov %rsp,%rbp  sub $0x30,%rsp | Stack Management for main function |
| mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax | Store sentinel stack-guard value for double data type variables. Exclusive or register %eax by itself to set it to 0. |
| movq $0x0,-0x20(%rbp)  movq $0x1,-0x18(%rbp) | Move quadword value 0 to 32 bytes above register %rbp. Move quadword value 1 to 24 bytes above register %rbp. |
| lea 0x0(%rip),%rsi # 0x2e <main+46>  lea 0x0(%rip),%rdi # 0x35 <main+53>  callq 0x3a <main+58> | Put memory address of 0(%rip) into register %rsi. Put memory address of 0(%rip) into register %rdi Print string to screen |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 0x44 <main+68>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x4f <main+79> | Move value from register %rax to register %rdx. Move value from register 0(%rip) to register %rax. Move value from register %rax to register %rsi. Move value from register %rdx to register %rdi. Print string to screen. |
| lea -0x28(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x5d <main+93>  callq 0x62 <main+98> | Put memory address of 18 bytes above register %rbp into register %rax. Move value from register %rax to register %rsi. Put memory address of 0(%rip) into register %rdi. Call cin function for user data |
| mov -0x28(%rbp),%rax  test %rax,%rax  je 0xf2 <main+242> | While Loop condition  Move value at 28 bytes above register %rbp to register %rax. AND value in register %rax with itself and set according flag on result. Jump to main+242 while values are equal. |
| mov -0x28(%rbp),%rcx  movabs $0x6666666666666667,%rdx  mov %rcx,%rax | Move value at 28 bytes above register %rbp to register %rcx. Move 64 bit constant to register %rdx. Move value in register %rcx to register %rax. |
| imul %rdx  sar $0x2,%rdx  mov %rcx,%rax  sar $0x3f,%rax  sub %rax,%rdx  mov %rdx,%rax | Muliply the value in register %rdx by itself by make a result take up 64 bits. Shift value in register %rdx right by value 2. Move value in register %rcx to register %rax. Shift value in register %rax by value 63. Subtract the value in register %rax from the value in register %rdx and store the result in register %rdx. Move the value in register %rdx to register %rax. |
| mov %rax,-0x10(%rbp)  mov -0x10(%rbp),%rdx  mov %rdx,%rax | Move the value in register %rax to 16 bytes above the register %rbp. Move the value 16 bytes above the register %rbp to register %rdx. Move the value in register %rdx to register %rax. |
| shl $0x2,%rax  add %rdx,%rax  add %rax,%rax  sub %rax,%rcx  mov %rcx,%rax | Shift value in register %rax right by value 4. Add the value from register %rdx to the value in register %rax and store the result in register %rax. Add the value from register %rax to itself and store the result in register %rax. Subtract the value in register %rax from the value in register %rcx and store the result in register %rcx. Move the value in register %rcx to register %rax. |
| mov %rax,-0x10(%rbp)  mov -0x10(%rbp),%rax  imul -0x18(%rbp),%rax | Move the value in register %rax to 16 bytes above the register %rbp. Move the value 16 bytes above the register %rbp to register %rax. Multiply the value 24 bytes above register %rbp by the value in register %rax and store the result in register %rax. |
| add %rax,-0x20(%rbp)  shlq -0x18(%rbp) | Add the value in register %rax to 32 bytes above register %rbp and store the value 32 bytes above register %rbp. Shift the value at 24 bytes above register %rbp to the left by 2. |
| mov -0x28(%rbp),%rcx  movabs $0x6666666666666667,%rdx  mov %rcx,%rax | Move value at 28 bytes above register %rbp to register %rcx. Move 64 bit constant to register %rdx. Move value in register %rcx to register %rax. |
| imul %rdx  sar $0x2,%rdx  mov %rcx,%rax  sar $0x3f,%rax  sub %rax,%rdx  mov %rdx,%rax | Muliply the value in register %rdx by itself by make a result take up 64 bits. Shift value in register %rdx right by value 2. Move value in register %rcx to register %rax. Shift value in register %rax by value 63. Subtract the value in register %rax from the value in register %rdx and store the result in register %rdx. Move the value in register %rdx to register %rax. |
| mov %rax,-0x28(%rbp)  jmpq 0x62 <main+98> | Move the value from register %rax to 40 bytes above register %rbp. Jump to main+98 |
| lea 0x0(%rip),%rsi # 0xf9 <main+249>  lea 0x0(%rip),%rdi # 0x100 <main+256>  callq 0x105 <main+261> | Put memory address of 0(%rip) into register %rsi. Put memory address of 0(%rip) into register %rdi Print string to screen |
| mov %rax,%rdx  mov -0x20(%rbp),%rax  mov %rax,%rsi  mov %rdx,%rdi  callq 0x117 <main+279> | Move value from register %rax to register %rdx. Move value from 32 bytes above register %rip to register %rax. Move value from register %rax to register %rsi. Move value from register %rdx to register %rdi. Print value to screen. |
| mov %rax,%rdx  mov 0x0(%rip),%rax # 0x121 <main+289>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x12c <main+300> | Move value from register %rax to register %rdx. Move value from register 0(%rip) to register %rax. Move value from register %rax to register %rsi. Move value from register %rdx to register %rdi. Print string to screen. |
| mov $0x0,%eax  mov -0x8(%rbp),%rsi | Move value 1 to register %eax. Move value at 8 bytes above register %rbp to register %rcx. |
| xor %fs:0x28,%rsi  je 0x145 <main+325> | Error check to see if the stored value on the stack was changed. If they are equal, jump to main+394 |
| callq 0x145 <main+325> | Display stack check fail |
| leaveq  retq | Terminate the main function. |

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

int main()

{

int binary, mod;

int hexDec = 0;

int base = 1;

cout << "Enter the binary number: " << endl;

cin >> binary;

while (binary)

{

mod = binary % 10;

binary = binary / 10;

hexDec += mod \* base;

base = base \* 2;

}

cout << "Equivalent hexadecimal value: " << hexDec << endl;

return 0;

}

**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 $0x30,%rsp | int main()  { | Initialize main function |
| mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax | int binary, mod; | Declare integer variables binary and mod. |
| movq $0x0,-0x20(%rbp)  movq $0x1,-0x18(%rbp) | int hexDec = 0;  int base = 1; | Initialize variables hexDec to 0 and base to 1. |
| lea 0x0(%rip),%rsi # 0x2e <main+46>  lea 0x0(%rip),%rdi # 0x35 <main+53>  callq 0x3a <main+58>  mov %rax,%rdx  mov 0x0(%rip),%rax # 0x44 <main+68>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x4f <main+79> | cout << "Enter the binary number: " << endl; | Print to screen string “Enter the binary number:” followed by a newline character. |
| lea -0x28(%rbp),%rax  mov %rax,%rsi  lea 0x0(%rip),%rdi # 0x5d <main+93>  callq 0x62 <main+98> | cin >> binary; | Accept user input and store the value in variable binary. |
| mov -0x28(%rbp),%rax  test %rax,%rax  je 0xf2 <main+242>  …  mov %rax,-0x28(%rbp)  jmpq 0x62 <main+98> | while (binary)  {  …  } | While loop condition  The value in the variable binary is not 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 | mod = binary % 10; | Modulo the value in the variable binary by the value 10 and store the result in variable mod. |
| 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 | binary = binary / 10; | Divide the value of variable binary by the value 10 and store the result in variable binary. |
| mov %rax,-0x10(%rbp)  mov -0x10(%rbp),%rax  imul -0x18(%rbp),%rax  add %rax,-0x20(%rbp)  shlq -0x18(%rbp) | hexDec += mod \* base; | Multiply the value in variable mod by the value in variable base and add that to the value in variable hexDec. Store the result in variable hexDec. |
| 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 | base = base \* 2; | Multiply the value in variable base by the value 2 and store the result in variable base. |
| lea 0x0(%rip),%rsi # 0xf9 <main+249>  lea 0x0(%rip),%rdi # 0x100 <main+256>  callq 0x105 <main+261>  mov %rax,%rdx  mov -0x20(%rbp),%rax  mov %rax,%rsi  mov %rdx,%rdi  callq 0x117 <main+279>  mov %rax,%rdx  mov 0x0(%rip),%rax # 0x121 <main+289>  mov %rax,%rsi  mov %rdx,%rdi  callq 0x12c <main+300> | cout << "Equivalent hexadecimal value: " << hexDec << endl; | Print to screen the string “Equivalent hexadecimal value: ”. Followed by the value in the variable hexDec. Then end with a newline character. |
| mov $0x0,%eax  mov -0x8(%rbp),%rsi  xor %fs:0x28,%rsi  je 0x145 <main+325>  callq 0x145 <main+325>  leaveq  retq | return 0;  } | Return 0 to signify a successful execution and terminate the main function. |