# **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** |
| --- | --- |
| 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 callq 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 callq 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 callq 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 callq 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 callq 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 callq 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 jmpq f <main+0xf>  a3: b8 00 00 00 00 mov $0x0,%eax  a8: c9 leaveq  a9: c3 retq | This part of the assembly code(apart from the dump numbers) this declares all of the variables |
| aa: 55 push %rbp  ab: 48 89 e5 mov %rsp,%rbp  ae: 48 83 ec 10 sub $0x10,%rsp  b2: 89 7d fc mov %edi,-0x4(%rbp)  b5: 89 75 f8 mov %esi,-0x8(%rbp)  b8: 83 7d fc 01 cmpl $0x1,-0x4(%rbp)  bc: 75 32 jne f0 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x46>  be: 81 7d f8 ff ff 00 00 cmpl $0xffff,-0x8(%rbp)  c5: 75 29 jne f0 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x46>  c7: 48 8d 3d 00 00 00 00 lea 0x0(%rip),%rdi # ce <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x24>  ce: e8 00 00 00 00 callq d3 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x29>  d3: 48 8d 15 00 00 00 00 lea 0x0(%rip),%rdx # da <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x30>  da: 48 8d 35 00 00 00 00 lea 0x0(%rip),%rsi # e1 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x37>  e1: 48 8b 05 00 00 00 00 mov 0x0(%rip),%rax # e8 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x3e>  e8: 48 89 c7 mov %rax,%rdi  eb: e8 00 00 00 00 callq f0 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x46>  f0: 90 nop  f1: c9 leaveq  f2: c3 retq | This assembly code I believe is setting some loops to iterate through the variables. |
| f3: 55 push %rbp  f4: 48 89 e5 mov %rsp,%rbp  f7: be ff ff 00 00 mov $0xffff,%esi  fc: bf 01 00 00 00 mov $0x1,%edi  101: e8 a4 ff ff ff callq aa <\_Z41\_\_static\_initialization\_and\_destruction\_0ii>  106: 5d pop %rbp  107: c3 retq | I believe this code is still just iterating throught the loops |
|  |  |
|  |  |

**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  mov %edi,-0x4(%rbp)  mov %esi,-0x8(%rbp)  cmpl $0x1,-0x4(%rbp)  jne f0 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x46>  cmpl $0xffff,-0x8(%rbp)  jne f0 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x46>  lea 0x0(%rip),%rdi # ce <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x24>  callq d3 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x29>  lea 0x0(%rip),%rdx # da <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x30>  lea 0x0(%rip),%rsi # e1 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x37>  mov 0x0(%rip),%rax # e8 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x3e>  mov %rax,%rdi  callq f0 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x46>  nop  leaveq  retq | for (a = 1; a <= 9; a++){  for (i = 1; i <= 9; i++){ | Both of these lines of code are looping through the variables a and i and running the code under it |
| lea 0x0(%rip),%rdi # 3c <main+0x3c>  callq 41 <main+0x41>  lea 0x0(%rip),%rsi # 48 <main+0x48>  mov %rax,%rdi  callq 50 <main+0x50>  mov %rax,%rdx  mov -0xc(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 60 <main+0x60>  lea 0x0(%rip),%rsi # 67 <main+0x67>  mov %rax,%rdi  callq 6f <main+0x6f>  mov %rax,%rdx  mov -0x4(%rbp),%eax  mov %eax,%esi  mov %rdx,%rdi  callq 7f <main+0x7f>  mov %rax,%rdx  mov 0x0(%rip),%rax # 89 <main+0x89>  mov %rax,%rsi  mov %rdx,%rdi  callq 94 <main+0x94>  addl $0x1,-0xc(%rbp)  jmp 20 <main+0x20>  addl $0x1,-0x8(%rbp)  jmpq f <main+0xf>  mov $0x0,%eax  leaveq  retq | x = a \* i; | This code replaces x with a multiplied by i |
| push %rbp  mov %rsp,%rbp  mov $0xffff,%esi  mov $0x1,%edi  callq aa <\_Z41\_\_static\_initialization\_and\_destruction\_0ii>  pop %rbp  retq | cout << a << " \* " << i << " = " << x << endl; | This line of code prints to the terminal the output sentence $a \* $i = $x |
|  |  |  |
|  |  |  |

## **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  mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax  lea 0x0(%rip),%rsi # 1e <main+0x1e>  lea 0x0(%rip),%rdi # 25 <main+0x25>  callq 2a <main+0x2a>  mov %rax,%rdx  mov 0x0(%rip),%rax # 34 <main+0x34>  mov %rax,%rsi  mov %rdx,%rdi  callq 3f <main+0x3f>  lea -0x14(%rbp),%rax  mov %rax,%rsi | This assembly code is setting the variables for the file |
| lea 0x0(%rip),%rdi # 4d <main+0x4d>  callq 52 <main+0x52>  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 # 73 <main+0x73> | This assembly code is the declaration for the main function and moving more data to different sections in RAM |
| mulsd %xmm1,%xmm0  movsd %xmm0,-0x10(%rbp)  lea 0x0(%rip),%rsi # 83 <main+0x83  lea 0x0(%rip),%rdi # 8a <main+0x8a  callq 8f <main+0x8f>  mov %rax,%rdx  mov -0x10(%rbp),%rax  mov %rax,-0x28(%rbp)  movsd -0x28(%rbp),%xmm0  mov %rdx,%rdi  callq a7 <main+0xa7>  mov $0x0,%eax  mov -0x8(%rbp),%rcx  xor %fs:0x28,%rcx  je c0 <main+0xc0>  callq c0 <main+0xc0>  leaveq  retq | I believe this section is printing some print lines to the terminal |
| push %rbp  mov %rsp,%rbp  sub $0x10,%rsp  mov %edi,-0x4(%rbp)  mov %esi,-0x8(%rbp)  cmpl $0x1,-0x4(%rbp)  jne 108 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x46>  cmpl $0xffff,-0x8(%rbp)  jne 108 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x46>  lea 0x0(%rip),%rdi # e6 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x24>  callq eb <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x29>  lea 0x0(%rip),%rdx # f2 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x30>  lea 0x0(%rip),%rsi # f9 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x37>  mov 0x0(%rip),%rax # 100 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x3e>  mov %rax,%rdi  callq 108 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x46>  nop  leaveq  retq | This line Is destroying some temp variables in the stack |
| push %rbp  mov %rsp,%rbp  mov $0xffff,%esi  mov $0x1,%edi  callq c2 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii>  pop %rbp  retq | Finally this block of code I believe is printing the final line of code to the terminal |

**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 $0x30,%rsp  mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax  lea 0x0(%rip),%rsi # 1e <main+0x1e>  lea 0x0(%rip),%rdi # 25 <main+0x25>  callq 2a <main+0x2a>  mov %rax,%rdx  mov 0x0(%rip),%rax # 34 <main+0x34>  mov %rax,%rsi  mov %rdx,%rdi  callq 3f <main+0x3f>  lea -0x14(%rbp),%rax  mov %rax,%rsi | int rad1;  float volsp; | Declaring varaibles |
| lea 0x0(%rip),%rdi # 4d <main+0x4d>  callq 52 <main+0x52>  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 # 73 <main+0x73> | Cout << “Radius: “ << endl; | Terminal printline |
| mulsd %xmm1,%xmm0  movsd %xmm0,-0x10(%rbp)  lea 0x0(%rip),%rsi # 83 <main+0x83  lea 0x0(%rip),%rdi # 8a <main+0x8a  callq 8f <main+0x8f>  mov %rax,%rdx  mov -0x10(%rbp),%rax  mov %rax,-0x28(%rbp)  movsd -0x28(%rbp),%xmm0  mov %rdx,%rdi  callq a7 <main+0xa7>  mov $0x0,%eax  mov -0x8(%rbp),%rcx  xor %fs:0x28,%rcx  je c0 <main+0xc0>  callq c0 <main+0xc0>  leaveq  retq | Cin >> rad1; | Receiving the input of Radius from the shell line |
| push %rbp  mov %rsp,%rbp  sub $0x10,%rsp  mov %edi,-0x4(%rbp)  mov %esi,-0x8(%rbp)  cmpl $0x1,-0x4(%rbp)  jne 108 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x46>  cmpl $0xffff,-0x8(%rbp)  jne 108 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x46>  lea 0x0(%rip),%rdi # e6 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x24>  callq eb <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x29>  lea 0x0(%rip),%rdx # f2 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x30>  lea 0x0(%rip),%rsi # f9 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x37>  mov 0x0(%rip),%rax # 100 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x3e>  mov %rax,%rdi  callq 108 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+0x46>  nop  leaveq  retq | Coutn << (4\*3.14\*rad1\*rad1\*rad1)/3; | Logic for the volume that is returned |
| push %rbp  mov %rsp,%rbp  mov $0xffff,%esi  mov $0x1,%edi  callq c2 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii>  pop %rbp  retq | return 0;  } | Ending of the program successfully |

## **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 $0x30,%rsp  mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax  lea 0x0(%rip),%rsi # 0x1e <main+30>  lea 0x0(%rip),%rdi # 0x25 <main+37>  callq 0x2a <main+42>  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>  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)  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 | Main function that declares variables has a while loop and and print statements |
| push %rbp  mov %rsp,%rbp  sub $0x10,%rsp  mov %edi,-0x4(%rbp)  mov %esi,-0x8(%rbp)  cmpl $0x1,-0x4(%rbp)  jne 0x108 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+70>  cmpl $0xffff,-0x8(%rbp)  jne 0x108 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+70>  lea 0x0(%rip),%rdi # 0xe6 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+36>  callq 0xeb <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+41>  lea 0x0(%rip),%rdx # 0xf2 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+48>  lea 0x0(%rip),%rsi # 0xf9 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+55>  mov 0x0(%rip),%rax # 0x100 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+62>  mov %rax,%rdi  callq 0x108 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii+70>  nop  leaveq  retq | Iteration through the loop and changing the data in between stacks |
| push %rbp  mov %rsp,%rbp  mov $0xffff,%esi  mov $0x1,%edi  callq 0xc2 <\_Z41\_\_static\_initialization\_and\_destruction\_0ii>  pop %rbp | Import statements and ending of program |
|  |  |
|  |  |

**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** |
| --- | --- | --- |
| call puts  leaq -20(%rbp), %rax  movq %rax, %rsi  movl $.LC1, %edi  movl $0, %eax  call \_\_isoc99\_scanf  movl $1, -8(%rbp)  movl -20(%rbp), %eax  subl $1, %eax  movl %eax, -4(%rbp) | printf("Enter number of rows \n");  scanf("%d", &rows);  stars = 1;  spaces = rows - 1; |  |
| .L9:  movl $1, -12(%rbp)  jmp .L3  .L4:  movl $32, %edi  call putchar  addl $1, -12(%rbp)  .L3:  movl -12(%rbp), %eax  cmpl -4(%rbp), %eax | for(i=1; i<rows\*2; i++)  for(j=1; j<=spaces; j++){  printf(" ");  }  j<=spaces  j=1;  j++; |  |
| movl $10, %edi  call putchar  movl -20(%rbp), %eax  cmpl %eax, -16(%rbp)  .L7:  addl $1, -4(%rbp)  subl $1, -8(%rbp)  .L8:  addl $1, -16(%rbp)  .L2:  movl -20(%rbp), %eax  addl %eax, %eax  cmpl -16(%rbp), %eax  jg .L9  movl $0, %eax  leave  ret | if(i<rows)  if(i<rows){  spaces--;  stars++;}  spaces++;  stars--;  i++;  return(0); |  |
|  |  |  |
|  |  |  |

## **File Four**

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| .section .rodata  .LC0:  .string "Enter the binary number: "  .LC1:  .string "%ld"  .align 8  .LC2:  .string "Equivalent hexadecimal value: %lX \n"  .text  .globl main  .type main, @function  main:  pushq %rbp  movq %rsp, %rbp  subq $32, %rsp  movq $0, -24(%rbp)  movq $1, -16(%rbp)  movl $.LC0, %edi  movl $0, %eax  call printf  leaq -32(%rbp), %rax  movq %rax, %rsi  movl $.LC1, %edi  movl $0, %eax  call \_\_isoc99\_scanf  jmp .L2 | Sets all the variables and has print statements and scan statements |
| .L3:  movq -32(%rbp), %rcx  movabsq $7378697629483820647, %rdx  movq %rcx, %rax  imulq %rdx  sarq $2, %rdx  movq %rcx, %rax  sarq $63, %rax  subq %rax, %rdx  movq %rdx, %rax  movq %rax, -8(%rbp)  movq -8(%rbp), %rdx  movq %rdx, %rax  salq $2, %rax  addq %rdx, %rax  addq %rax, %rax  subq %rax, %rcx  movq %rcx, %rax  movq %rax, -8(%rbp) | While loop and I believe setting more variables |
| movq -8(%rbp), %rax  imulq -16(%rbp), %rax  addq %rax, -24(%rbp)  salq -16(%rbp)  movq -32(%rbp), %rcx  movabsq $7378697629483820647, %rdx  movq %rcx, %rax  imulq %rdx  sarq $2, %rdx  movq %rcx, %rax  sarq $63, %rax  subq %rax, %rdx  movq %rdx, %rax  movq %rax, -32(%rbp) | Moving around bits and adding bits together |
| .L2:  movq -32(%rbp), %rax  testq %rax, %rax  jne .L3  movq -24(%rbp), %rax  movq %rax, %rsi  movl $.LC2, %edi  movl $0, %eax  call printf  movl $0, %eax  leave  ret | Moving more bits and more print statements |
|  |  |

**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** |
| --- | --- | --- |
| .section .rodata  .LC0:  .string "Enter the binary number: "  .LC1:  .string "%ld"  .align 8  .LC2:  .string "Equivalent hexadecimal value: %lX \n"  .text  .globl main  .type main, @function  main:  pushq %rbp  movq %rsp, %rbp  subq $32, %rsp  movq $0, -24(%rbp)  movq $1, -16(%rbp)  movl $.LC0, %edi  movl $0, %eax  call printf  leaq -32(%rbp), %rax  movq %rax, %rsi  movl $.LC1, %edi  movl $0, %eax  call \_\_isoc99\_scanf  jmp .L2 | long int hexnumber = 0, count = 1  printf("Enter the binary number: ");  scanf("%ld", &bnumber);  while (bnumber != 0) | Setting hexnumber to 0  Set count to 1  Print to the terminal “enter the binary number:”  Scan 1d &bnumber  While loop to keep going while the variable is not equal to 0 |
| .L3:  movq -32(%rbp), %rcx  movabsq $7378697629483820647, %rdx  movq %rcx, %rax  imulq %rdx  sarq $2, %rdx  movq %rcx, %rax  sarq $63, %rax  subq %rax, %rdx  movq %rdx, %rax  movq %rax, -8(%rbp)  movq -8(%rbp), %rdx  movq %rdx, %rax  salq $2, %rax  addq %rdx, %rax  addq %rax, %rax  subq %rax, %rcx  movq %rcx, %rax  movq %rax, -8(%rbp) | remainder = bnumber % 10;  hexnumber = hexnumber + remainder \* count;  count = count \* 2; | Sets remainder equal to bnumber % 10  Hexnumber equals itself remainder times count  Count equal count times 2 |
| movq -8(%rbp), %rax  imulq -16(%rbp), %rax  addq %rax, -24(%rbp)  salq -16(%rbp)  movq -32(%rbp), %rcx  movabsq $7378697629483820647, %rdx  movq %rcx, %rax  imulq %rdx  sarq $2, %rdx  movq %rcx, %rax  sarq $63, %rax  subq %rax, %rdx  movq %rdx, %rax  movq %rax, -32(%rbp) | bnumber != 0  printf("Equivalent hexadecimal value: %lX \n", hexnumber); | Sets bnumber to zero and prints out “equivalent hexadecimal value: %1X\n” |
| .L2:  movq -32(%rbp), %rax  testq %rax, %rax  jne .L3  movq -24(%rbp), %rax  movq %rax, %rsi  movl $.LC2, %edi  movl $0, %eax  call printf  movl $0, %eax  leave  ret | Return(0) | Exits program successfully |
|  |  |  |