# **CS 410 Binary to C++ With Security Vulnerabilities Activity Template**

**Step 1:** Convert the binary file to assembly code.

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

| **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- |
| 1. 0x0000000000000079 <+0>:   push %rbp   1. 0x000000000000007a <+1>:   mov %rsp,%rbp   1. 0x000000000000007d <+4>:   sub $0x20,%rsp   1. 0x0000000000000081 <+8>:   mov %fs:0x28,%rax   1. 0x000000000000008a <+17>:   mov %rax,-0x8(%rbp)   1. 0x000000000000008e <+21>:   xor %eax,%eax | 1. This instruction saves the base pointer of the previous stack frame by pushing it onto the stack. 2. This sets the base pointer for the current stack frame, it moves the stack pointer into the base pointer register. 3. This subtracts the value 0x20, which in base 10 is 32, from the stack pointer register, allocating 32 bytes of space on the stack for local variables. 4. This instruction moves the value stored at an 0x28 offset in the fs segment register into the %rax register. 5. This moves the value previously stored in the %rax register into the location at -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register. 6. This instruction effectively clears the %eax register by performing an XOR operation on itself, which results in a zero value. This zero value is stored back into the %eax register. |
| 1. 0x0000000000000090 <+23>: movl $0x0,-0x14(%rbp) 2. 0x0000000000000097 <+30>: mov -0x14(%rbp),%eax 3. 0x000000000000009a <+33>: cmp $0x5,%eax 4. 0x000000000000009d <+36>:   je 0x308 <main+655> | 1. This instruction moves the value 0 and stores it into the location -0x14(%rbp), 0x14 in base 10 is 20, which is an -20 offset or 20 bytes above the base pointer register. This is the value of the variable choice, which is being initialized to zero, and is the control variable for the program’s while loop. 2. This instruction moves the value previously stored at -0x14(%rbp) into the %eax register to prepare it for use in the following comparison. 3. This compares the value 5 with the value stored in the %eax register. 4. If from the previously made comparison, the value stored in the %eax register is equal to 5, this instruction jumps the program’s execution to address 0x308, which is line 655 of the main function, exiting the program’s while loop. |
| 1. 0x00000000000000a3 <+42>:   lea 0x0(%rip),%rsi # 0xaa <main+49>   1. 0x00000000000000aa <+49>:   lea 0x0(%rip),%rdi # 0xb1 <main+56>   1. 0x00000000000000b1 <+56>: callq 0xb6 <main+61> | 1. This instruction loads the effective address 0x0(%rip), which is the next instruction, and moves its value into the %rsi register, the second argument register to be used in the next function call. The LEA instruction acts as a pointer. Here, it is placing the address of the instruction locates at 0x0(%rip), into the %rsi register 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register to be used in the following function call. 3. This instruction calls the output function to display the string “----------------“, followed by a new line. |
| 1. 0x00000000000000b6 <+61>:   lea 0x0(%rip),%rsi # 0xbd <main+68>   1. 0x00000000000000bd <+68>:   lea 0x0(%rip),%rdi # 0xc4 <main+75>   1. 0x00000000000000c4 <+75>: callq 0xc9 <main+80> | 1. This instruction loads the effective address 0x0(%rip), which is the next instruction, and moves its value into the %rsi register, the second argument register to be used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register to be used in the following function call. 3. This instruction calls the output function to display the string “- 1) Add -“, followed by a new line. |
| 1. 0x00000000000000c9 <+80>:   lea 0x0(%rip),%rsi # 0xd0 <main+87>   1. 0x00000000000000d0 <+87>:   lea 0x0(%rip),%rdi # 0xd7 <main+94>   1. 0x00000000000000d7 <+94>: callq 0xdc <main+99> | 1. This instruction loads the effective address 0x0(%rip), which is the next instruction, and moves its value into the %rsi register, the second argument register to be used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register to be used in the following function call. 3. This instruction calls the output function to display the string “- 2) Subtract -“, followed by a new line. |
| 1. 0x00000000000000dc <+99>:   lea 0x0(%rip),%rsi # 0xe3 <main+106>   1. 0x00000000000000e3 <+106>: lea 0x0(%rip),%rdi # 0xea <main+113> 2. 0x00000000000000ea <+113>: callq 0xef <main+118> | 1. This instruction loads the effective address 0x0(%rip), which is the next instruction, and moves its value into the %rsi register, the second argument register to be used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register to be used in the following function call. 3. This instruction calls the output function to display the string “- 3) Multiply -“, followed by a new line. |
| 1. 0x00000000000000ef <+118>: lea 0x0(%rip),%rsi # 0xf6 <main+125> 2. 0x00000000000000f6 <+125>: lea 0x0(%rip),%rdi # 0xfd <main+132> 3. 0x00000000000000fd <+132>: callq 0x102 <main+137> | 1. This instruction loads the effective address 0x0(%rip), which is the next instruction, and moves its value into the %rsi register, the second argument register to be used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register to be used in the following function call. 3. This instruction calls the output function to display the string “- 4) Exit -“, followed by a new line. |
| 1. 0x0000000000000102 <+137>: lea 0x0(%rip),%rsi # 0x109 <main+144> 2. 0x0000000000000109 <+144>: lea 0x0(%rip),%rdi # 0x110 <main+151> 3. 0x0000000000000110 <+151>: callq 0x115 <main+156> | 1. This instruction loads the effective address 0x0(%rip), which is the next instruction, and moves its value into the %rsi register, the second argument register to be used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register to be used in the following function call. 3. This instruction calls the output function to display the string “----------------“, followed by a new line. |
| 1. 0x0000000000000115 <+156>: lea -0x14(%rbp),%rax 2. 0x0000000000000119 <+160>: mov %rax,%rsi 3. 0x000000000000011c <+163>: lea 0x0(%rip),%rdi # 0x123 <main+170> 4. 0x0000000000000123 <+170>: callq 0x128 <main+175> | 1. This instruction loads the effective address -0x14(%rbp) into the %rax register to prepare for use in the next input function call. This is the value of the variable choice. 2. This moves the value previously stored in the %rax register into the %rsi register, the second argument register for use in the next function call. 3. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register for use in the following function call. 4. This instruction calls the input function to receive a value and store it in the variable choice. |
| 1. 0x0000000000000128 <+175>: mov -0x14(%rbp),%eax 2. 0x000000000000012b <+178>: cmp $0x1,%eax 3. 0x000000000000012e <+181>: jne 0x1c9 <main+336> | 1. This moves the value stored at -0x14(%rbp), the value of the variable choice, into the %eax register to prepare for use in the following comparison. 2. This instruction compares the value 1 with the value previously stored in the %eax register. 3. If based on the previous comparison the value stored in the %eax register does not equal 1, this instruction jumps the program’s execution to address 0x1c9, which is line 336, where it will prepare for the next comparison to be made. |
| 1. 0x0000000000000134 <+187>: lea -0x10(%rbp),%rax 2. 0x0000000000000138 <+191>: mov %rax,%rsi 3. 0x000000000000013b <+194>: lea 0x0(%rip),%rdi # 0x142 <main+201> 4. 0x0000000000000142 <+201>: callq 0x147 <main+206> | 1. This instruction loads the effective address -0x10(%rbp), 0x10 in base 10 is 16, which is an -16 offset or 16 bytes above the base pointer register, into the %rax register. This is the variable num1. 2. This moves the value previously stored in the %rax register into the %rsi register, the second argument register, for use in the next function call. 3. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register for the following function call. 4. This instruction calls the address 0x147. This block is the declaration of num1. |
| 1. 0x0000000000000147 <+206>: mov %rax,%rdx 2. 0x000000000000014a <+209>: lea -0xc(%rbp),%rax 3. 0x000000000000014e <+213>: mov %rax,%rsi 4. 0x0000000000000151 <+216>: mov %rdx,%rdi 5. 0x0000000000000154 <+219>: callq 0x159 <main+224> | 1. This moves the value previously stored in the %rax register into the %rdx register. 2. This loads the effective address -0xc(%rbp), 0xc in base 10 is 12, which is a -12 offset or 12 bytes above the base pointer register, into the %rax register. This is the variable num2. 3. This moves the previously stored value in the %rax register into the %rsi register, the second argument register, in preparation for the next function call. 4. This moves the value stored in the %rdx register into the %rdi register, the first argument register, for use in the following function call. 5. This calls the function at address 0x159. This block is the declaration of num2. |
| 1. 0x0000000000000159 <+224>: mov -0x10(%rbp),%eax 2. 0x000000000000015c <+227>: mov %eax,%esi 3. 0x000000000000015e <+229>: lea 0x0(%rip),%rdi # 0x165 <main+236> 4. 0x0000000000000165 <+236>: callq 0x16a <main+241> | 1. This instruction moves the value stored at -0x10(%rbp) into the %eax register in preparation for the next output function call. This is the value of the variable num1. 2. This moves the previously stored value in the %eax register into the %esi register, a second argument register to be used in the following function call. 3. This loads the effective address 0x0(%rip), which is the next instruction, into the %rdi register, the first argument register used in the following output function call. 4. This calls the output function to display the value of the variable num1. |
| 1. 0x000000000000016a <+241>: lea 0x0(%rip),%rsi # 0x171 <main+248> 2. 0x0000000000000171 <+248>: mov %rax,%rdi 3. 0x0000000000000174 <+251>: callq 0x179 <main+256> | 1. This loads the effective address 0x0(%rip), which is the next instruction, and points to the string “ – “, into the %rsi register, the second argument register, in preparation for the next output function call. 2. This moves the value previously stored in the %rax register into the %rdi register, the first argument register to be used in the following output function call. 3. This calls the output function to output the string “ – “. |
| 1. 0x0000000000000179 <+256>: mov %rax,%rdx 2. 0x000000000000017c <+259>: mov -0xc(%rbp),%eax 3. 0x000000000000017f <+262>: mov %eax,%esi 4. 0x0000000000000181 <+264>: mov %rdx,%rdi 5. 0x0000000000000184 <+267>: callq 0x189 <main+272> | 1. This moves the value previously stored in the %rax register into the %rdx register. 2. This moves the value stored at -0xc(%rbp), the value of the variable num2, into the %eax register, to prepare it for use in the next output function call. 3. This moves the value previously stored in the %eax register into the %esi register, the second argument register used in the next output function call. 4. This moves the value stored in the %rdx register into the %rdi register, the first argument register to be used in the following output function call. 5. This calls the output function to display the value of the variable num2. |
| 1. 0x0000000000000189 <+272>: lea 0x0(%rip),%rsi # 0x190 <main+279> 2. 0x0000000000000190 <+279>: mov %rax,%rdi 3. 0x0000000000000193 <+282>: callq 0x198 <main+287> | 1. This instruction loads the effective address 0x0(%rip), which is the next instruction and points to the location of the string “ = “, into the %rsi register, the second argument register to be used in the next output function call. 2. This moves the value previously stored in the %rax register into the %rdi register, the first argument register used in the following output function call. 3. This calls the output function to display the string “ = “. |
| 1. 0x0000000000000198 <+287>: mov %rax,%rcx 2. 0x000000000000019b <+290>: mov -0x10(%rbp),%edx 3. 0x000000000000019e <+293>: mov -0xc(%rbp),%eax 4. 0x00000000000001a1 <+296>: sub %eax,%edx 5. 0x00000000000001a3 <+298>: mov %edx,%eax 6. 0x00000000000001a5 <+300>: mov %eax,%esi 7. 0x00000000000001a7 <+302>: mov %rcx,%rdi 8. 0x00000000000001aa <+305>: callq 0x1af <main+310> | 1. This moves the value stored in the %rax register into the %rcx register. 2. This moves the value stored at -0x10(%rbp), which is the value of the variable num1, into the %edx register to prepare for the subtraction calculation. 3. This instruction moves the value stored at -0xc(%rbp), which is the value of the variable num2, into the %eax register to prepare it for use in the following calculation. 4. This subtracts the value stored in the %eax register from the value stored in the %edx register and stores the calculated value back into the %edx register. 5. This moves the previously stored value in the %edx register into the %eax register. 6. This moves the value stored in the %eax register into the %esi register, the second argument register to be used in the next function call. 7. This moves the value stored in the %rcx register into the %rdi register, the first argument register to be used in the following function call. 8. This calls the function at address 0x1af to prepare for the next output function call. |
| 1. 0x00000000000001af <+310>: mov %rax,%rdx 2. 0x00000000000001b2 <+313>: mov 0x0(%rip),%rax # 0x1b9 <main+320> 3. 0x00000000000001b9 <+320>: mov %rax,%rsi 4. 0x00000000000001bc <+323>: mov %rdx,%rdi 5. 0x00000000000001bf <+326>: callq 0x1c4 <main+331> | 1. This instruction moves the value stored in the %rax register into the %rdx register. 2. This moves the value stored at 0x0(%rip), the next instruction, into the %rax register to prepare for the next output function call. 3. This moves the value stored in the %rax register into the %rsi register, the second argument register to be used in the next output function call. 4. This moves the value stored in the %rdx register into the %rdi register, the first argument register to be used in the following output function call. 5. This calls the function to output the value of num1 – num2. |
| 1. 0x00000000000001c4 <+331>: jmpq 0x97 <main+30> | 1. This instruction jumps the program’s execution to address 0x97, which is line 30 of the main function. This is the end of the first if statement and is taking us back to the beginning of the while loop. |
| 1. 0x00000000000001c9 <+336>: mov -0x14(%rbp),%eax 2. 0x00000000000001cc <+339>: cmp $0x2,%eax 3. 0x00000000000001cf <+342>: jne 0x268 <main+495> | 1. This instruction moves the value stored at -0x14(%rbp), which is the value of the variable choice, into the %eax register to prepare it for use in the following comparison. 2. This compares the value 2 with the value previously stored in the %eax register. 3. If from the previous comparison, the value stored in the %eax register does not equal the value 2, this instruction jumps the program’s execution to address 0x268, which is line 495 of the main function. |
| 1. 0x00000000000001d5 <+348>: lea -0x10(%rbp),%rax 2. 0x00000000000001d9 <+352>: mov %rax,%rsi 3. 0x00000000000001dc <+355>: lea 0x0(%rip),%rdi # 0x1e3 <main+362> 4. 0x00000000000001e3 <+362>: callq 0x1e8 <main+367> | 1. This instruction loads the effective address -0x10(%rbp), 0x10 in base 10 is 16, which is an -16 offset or 16 bytes above the base pointer register, into the %rax register. This is the variable num1. 2. This moves the value previously stored in the %rax register into the %rsi register, the second argument register, for use in the next function call. 3. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register for the following function call. 4. This instruction calls the address 0x1e8. This block is the declaration of num1. |
| 1. 0x00000000000001e8 <+367>: mov %rax,%rdx 2. 0x00000000000001eb <+370>: lea -0xc(%rbp),%rax 3. 0x00000000000001ef <+374>: mov %rax,%rsi 4. 0x00000000000001f2 <+377>: mov %rdx,%rdi 5. 0x00000000000001f5 <+380>: callq 0x1fa <main+385> | 1. This moves the value previously stored in the %rax register into the %rdx register. 2. This loads the effective address -0xc(%rbp), 0xc in base 10 is 12, which is a -12 offset or 12 bytes above the base pointer register, into the %rax register. This is the variable num2. 3. This moves the previously stored value in the %rax register into the %rsi register, the second argument register, in preparation for the next function call. 4. This moves the value stored in the %rdx register into the %rdi register, the first argument register, for use in the following function call. 5. This calls the function at address 0x1fa. This block is the declaration of num2. |
| 1. 0x00000000000001fa <+385>: mov -0x10(%rbp),%eax 2. 0x00000000000001fd <+388>: mov %eax,%esi 3. 0x00000000000001ff <+390>: lea 0x0(%rip),%rdi # 0x206 <main+397> 4. 0x0000000000000206 <+397>: callq 0x20b <main+402> | 1. This instruction moves the value stored at -0x10(%rbp) into the %eax register in preparation for the next output function call. This is the value of the variable num1. 2. This moves the previously stored value in the %eax register into the %esi register, a second argument register to be used in the following function call. 3. This loads the effective address 0x0(%rip), which is the next instruction, into the %rdi register, the first argument register used in the following output function call. 4. This calls the output function to display the value of the variable num1. |
| 1. 0x000000000000020b <+402>: lea 0x0(%rip),%rsi # 0x212 <main+409> 2. 0x0000000000000212 <+409>: mov %rax,%rdi 3. 0x0000000000000215 <+412>: callq 0x21a <main+417> | 1. This loads the effective address 0x0(%rip), which is the next instruction, and points to the string “ + “, into the %rsi register, the second argument register, in preparation for the next output function call. 2. This moves the value previously stored in the %rax register into the %rdi register, the first argument register to be used in the following output function call. 3. This calls the output function to output the string “ + “. |
| 1. 0x000000000000021a <+417>: mov %rax,%rdx 2. 0x000000000000021d <+420>: mov -0xc(%rbp),%eax 3. 0x0000000000000220 <+423>: mov %eax,%esi 4. 0x0000000000000222 <+425>: mov %rdx,%rdi 5. 0x0000000000000225 <+428>: callq 0x22a <main+433> | 1. This moves the value previously stored in the %rax register into the %rdx register. 2. This moves the value stored at -0xc(%rbp), the value of the variable num2, into the %eax register, to prepare it for use in the next output function call. 3. This moves the value previously stored in the %eax register into the %esi register, the second argument register used in the next output function call. 4. This moves the value stored in the %rdx register into the %rdi register, the first argument register to be used in the following output function call. 5. This calls the output function to display the value of the variable num2. |
| 1. 0x000000000000022a <+433>: lea 0x0(%rip),%rsi # 0x231 <main+440> 2. 0x0000000000000231 <+440>: mov %rax,%rdi 3. 0x0000000000000234 <+443>: callq 0x239 <main+448> | 1. This instruction loads the effective address 0x0(%rip), which is the next instruction and points to the location of the string “ = “, into the %rsi register, the second argument register to be used in the next output function call. 2. This moves the value previously stored in the %rax register into the %rdi register, the first argument register used in the following output function call. 3. This calls the output function to display the string “ = “. |
| 1. 0x0000000000000239 <+448>: mov %rax,%rcx 2. 0x000000000000023c <+451>: mov -0x10(%rbp),%edx 3. 0x000000000000023f <+454>: mov -0xc(%rbp),%eax 4. 0x0000000000000242 <+457>: add %edx,%eax 5. 0x0000000000000244 <+459>: mov %eax,%esi 6. 0x0000000000000246 <+461>: mov %rcx,%rdi 7. 0x0000000000000249 <+464>: callq 0x24e <main+469> | 1. This moves the value stored in the %rax register into the %rcx register. 2. This moves the value stored at -0x10(%rbp), which is the value of the variable num1, into the %edx register to prepare for the addition calculation. 3. This instruction moves the value stored at -0xc(%rbp), which is the value of the variable num2, into the %eax register to prepare it for use in the following calculation. 4. This adds the value stored in the %edx register with the value stored in the %eax register and stores the calculated value back into the %eax register. 5. This moves the value stored in the %eax register into the %esi register, the second argument register to be used in the next function call. 6. This moves the value stored in the %rcx register into the %rdi register, the first argument register to be used in the following function call. 7. This calls the function at address 0x24e to prepare for the next output function call. |
| 1. 0x000000000000024e <+469>: mov %rax,%rdx 2. 0x0000000000000251 <+472>: mov 0x0(%rip),%rax # 0x258 <main+479> 3. 0x0000000000000258 <+479>: mov %rax,%rsi 4. 0x000000000000025b <+482>: mov %rdx,%rdi 5. 0x000000000000025e <+485>: callq 0x263 <main+490> | 1. This instruction moves the value stored in the %rax register into the %rdx register. 2. This moves the value stored at 0x0(%rip), the next instruction, into the %rax register to prepare for the next output function call. 3. This moves the value stored in the %rax register into the %rsi register, the second argument register to be used in the next output function call. 4. This moves the value stored in the %rdx register into the %rdi register, the first argument register to be used in the following output function call. 5. This calls the function to output the value of num1 + num2. |
| 1. 0x0000000000000263 <+490>: jmpq 0x97 <main+30> | 1. This instruction jumps the program’s execution to address 0x97, which is line 30 of the main function. This is the end of the second if statement and is taking us back to the beginning of the while loop. |
| 1. 0x0000000000000268 <+495>: mov -0x14(%rbp),%eax 2. 0x000000000000026b <+498>: cmp $0x3,%eax 3. 0x000000000000026e <+501>: jne 0x97 <main+30> | 1. This instruction moves the value stored at -0x14(%rbp), which is the value of the variable choice, into the %eax register to prepare it for use in the following comparison. 2. This compares the value 3 with the value previously stored in the %eax register. 3. If from the previous comparison, the value stored in the %eax register does not equal the value 3, this instruction jumps the program’s execution to address 0x97, which is line 30 of the main function, returning it back to the beginning of the while loop. |
| 1. 0x0000000000000274 <+507>: lea -0x10(%rbp),%rax 2. 0x0000000000000278 <+511>: mov %rax,%rsi 3. 0x000000000000027b <+514>: lea 0x0(%rip),%rdi # 0x282 <main+521> 4. 0x0000000000000282 <+521>: callq 0x287 <main+526> | 1. This instruction loads the effective address -0x10(%rbp), 0x10 in base 10 is 16, which is an -16 offset or 16 bytes above the base pointer register, into the %rax register. This is the variable num1. 2. This moves the value previously stored in the %rax register into the %rsi register, the second argument register, for use in the next function call. 3. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register for the following function call. 4. This instruction calls the address 0x287. This block is the declaration of num1. |
| 1. 0x0000000000000287 <+526>: mov %rax,%rdx 2. 0x000000000000028a <+529>: lea -0xc(%rbp),%rax 3. 0x000000000000028e <+533>: mov %rax,%rsi 4. 0x0000000000000291 <+536>: mov %rdx,%rdi 5. 0x0000000000000294 <+539>: callq 0x299 <main+544> | 1. This moves the value previously stored in the %rax register into the %rdx register. 2. This loads the effective address -0xc(%rbp), 0xc in base 10 is 12, which is a -12 offset or 12 bytes above the base pointer register, into the %rax register. This is the variable num2. 3. This moves the previously stored value in the %rax register into the %rsi register, the second argument register, in preparation for the next function call. 4. This moves the value stored in the %rdx register into the %rdi register, the first argument register, for use in the following function call. 5. This calls the function at address 0x299. This block is the declaration of num2. |
| 1. 0x0000000000000299 <+544>: mov -0x10(%rbp),%eax 2. 0x000000000000029c <+547>: mov %eax,%esi 3. 0x000000000000029e <+549>: lea 0x0(%rip),%rdi # 0x2a5 <main+556> 4. 0x00000000000002a5 <+556>: callq 0x2aa <main+561> | 1. This instruction moves the value stored at -0x10(%rbp) into the %eax register in preparation for the next output function call. This is the value of the variable num1. 2. This moves the previously stored value in the %eax register into the %esi register, a second argument register to be used in the following function call. 3. This loads the effective address 0x0(%rip), which is the next instruction, into the %rdi register, the first argument register used in the following output function call. 4. This calls the output function to display the value of the variable num1. |
| 1. 0x00000000000002aa <+561>: lea 0x0(%rip),%rsi # 0x2b1 <main+568> 2. 0x00000000000002b1 <+568>: mov %rax,%rdi 3. 0x00000000000002b4 <+571>: callq 0x2b9 <main+576> | 1. This loads the effective address 0x0(%rip), which is the next instruction, and points to the string “ / “, into the %rsi register, the second argument register, in preparation for the next output function call. 2. This moves the value previously stored in the %rax register into the %rdi register, the first argument register to be used in the following output function call. 3. This calls the output function to output the string “ / “. |
| 1. 0x00000000000002b9 <+576>: mov %rax,%rdx 2. 0x00000000000002bc <+579>: mov -0xc(%rbp),%eax 3. 0x00000000000002bf <+582>: mov %eax,%esi 4. 0x00000000000002c1 <+584>: mov %rdx,%rdi 5. 0x00000000000002c4 <+587>: callq 0x2c9 <main+592> | 1. This moves the value previously stored in the %rax register into the %rdx register. 2. This moves the value stored at -0xc(%rbp), the value of the variable num2, into the %eax register, to prepare it for use in the next output function call. 3. This moves the value previously stored in the %eax register into the %esi register, the second argument register used in the next output function call. 4. This moves the value stored in the %rdx register into the %rdi register, the first argument register to be used in the following output function call. 5. This calls the output function to display the value of the variable num2. |
| 1. 0x00000000000002c9 <+592>: lea 0x0(%rip),%rsi # 0x2d0 <main+599> 2. 0x00000000000002d0 <+599>: mov %rax,%rdi 3. 0x00000000000002d3 <+602>: callq 0x2d8 <main+607> | 1. This instruction loads the effective address 0x0(%rip), which is the next instruction and points to the location of the string “ = “, into the %rsi register, the second argument register to be used in the next output function call. 2. This moves the value previously stored in the %rax register into the %rdi register, the first argument register used in the following output function call. 3. This calls the output function to display the string “ = “. |
| 1. 0x00000000000002d8 <+607>: mov %rax,%rcx 2. 0x00000000000002db <+610>: mov -0x10(%rbp),%eax 3. 0x00000000000002de <+613>: mov -0xc(%rbp),%esi 4. 0x00000000000002e1 <+616>: cltd 5. 0x00000000000002e2 <+617>: idiv %esi 6. 0x00000000000002e4 <+619>: mov %eax,%esi 7. 0x00000000000002e6 <+621>: mov %rcx,%rdi 8. 0x00000000000002e9 <+624>: callq 0x2ee <main+629> | 1. This instruction moves the value stored in the %rax register into the %rcx register. 2. This moves the value stored at -0x10(%rbp), which is the value of the variable num1, into the %eax register to prepare it for use in the division operation. 3. This instruction moves the value stored at -0xc(%rbp), the value of variable num2, into the %esi register to prepare it for use in the division operation. 4. This instruction sign-extends the value in the %eax register to the %edx:%eax register pair, in preparation for the following division operation. 5. This instruction divides the value stored in the previous %edx:%eax register pair by the value stored in the %esi register, then stores this value back into the %eax register. 6. This instruction moves the value previously calculated and stored in the %eax register into the %esi register, the second argument register in preparation for the following function call. 7. This instruction moves the value stored in the %rcx register into the %rdi register, the second argument register used in the following function call. 8. Calls the function at address 0x2ee, in preparation for the next output function call. |
| 1. 0x00000000000002ee <+629>: mov %rax,%rdx 2. 0x00000000000002f1 <+632>: mov 0x0(%rip),%rax # 0x2f8 <main+639> 3. 0x00000000000002f8 <+639>: mov %rax,%rsi 4. 0x00000000000002fb <+642>: mov %rdx,%rdi 5. 0x00000000000002fe <+645>: callq 0x303 <main+650> | 1. This instruction moves the value stored in the %rax register into the %rdx register. 2. This moves the value stored at 0x0(%rip), the next instruction, into the %rax register in preparation for the next output function call. 3. This moves the value stored in the %rax register into the %rsi register, the second argument register used in the next output function call. 4. This instruction moves the value stored in the %rdx register into the %rdi register, the first argument register to be used in the following function call. 5. This instruction calls the output function to display the value of num1 / num2. |
| 1. 0x0000000000000303 <+650>: jmpq 0x97 <main+30> | 1. This instruction jumps the program’s execution to address 0x97, which is line 30 of the main function, and the beginning of the while loop. This is the end of the third if statement. |
| 1. 0x0000000000000308 <+655>: mov $0x0,%eax 2. 0x000000000000030d <+660>: mov -0x8(%rbp),%rcx 3. 0x0000000000000311 <+664>: xor %fs:0x28,%rcx 4. 0x000000000000031a <+673>: je 0x321 <main+680> 5. 0x000000000000031c <+675>: callq 0x321 <main+680> 6. 0x0000000000000321 <+680>: leaveq | 1. This instruction moves the value 0 into the %eax register. 2. This moves the value stored at location -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register, into the %rcx register. 3. This instruction performs an XOR comparison between the value at the %fs segment register with an offset 0x28, and the value stored in the %rcx register. 4. If from the previous comparison, a Zero Flag is set, meaning the two values equaled each other, this instruction jumps the program’s execution to address 0x321, which is line 680 of the main function. 5. This calls the function at address 0x321, which is line 680 of the main function. 6. This instruction cleans up the stack frame before returning from the function. It moves the base pointer to the stack pointer and pops the old base pointer off the stack. |
| DisplayMenu function | The following cells are the assembly instructions for the function DisplayMenu, which in this program is never called. We see at the beginning of the while loop, it makes its own output function calls to display the same information each iteration of the while loop. This is fixed in the C++ code that I wrote. |
| 1. 0x0000000000000000 <+0>: push %rbp 2. 0x0000000000000001 <+1>: mov %rsp,%rbp | 1. This instruction saves the base pointer of the previous stack frame by pushing it onto the stack. 2. This sets the base pointer for the current stack frame, it moves the stack pointer into the base pointer register. |
| 1. 0x0000000000000004 <+4>:   lea 0x0(%rip),%rsi # 0xb <\_Z11DisplayMenuv+11>   1. 0x000000000000000b <+11>:   lea 0x0(%rip),%rdi # 0x12 <\_Z11DisplayMenuv+18>   1. 0x0000000000000012 <+18>: callq 0x17 <\_Z11DisplayMenuv+23> | 1. This instruction loads the effective address 0x0(%rip), which is the next instruction, and moves its value into the %rsi register, the second argument register to be used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register to be used in the following function call. 3. This instruction calls the output function to display the string “----------------“, followed by a new line. |
| 1. 0x0000000000000017 <+23>:   lea 0x0(%rip),%rsi # 0x1e <\_Z11DisplayMenuv+30>   1. 0x000000000000001e <+30>:   lea 0x0(%rip),%rdi # 0x25 <\_Z11DisplayMenuv+37>   1. 0x0000000000000025 <+37>: callq 0x2a <\_Z11DisplayMenuv+42> | 1. This instruction loads the effective address 0x0(%rip), which is the next instruction, and moves its value into the %rsi register, the second argument register to be used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register to be used in the following function call. 3. This instruction calls the output function to display the string “- 1) Add -“, followed by a new line. |
| 1. 0x000000000000002a <+42>: lea 0x0(%rip),%rsi # 0x31 <\_Z11DisplayMenuv+49> 2. 0x0000000000000031 <+49>: lea 0x0(%rip),%rdi # 0x38 <\_Z11DisplayMenuv+56> 3. 0x0000000000000038 <+56>: callq 0x3d <\_Z11DisplayMenuv+61> | 1. This instruction loads the effective address 0x0(%rip), which is the next instruction, and moves its value into the %rsi register, the second argument register to be used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register to be used in the following function call. 3. This instruction calls the output function to display the string “- 2) Subtract -“, followed by a new line. |
| 1. 0x000000000000003d <+61>: lea 0x0(%rip),%rsi # 0x44 <\_Z11DisplayMenuv+68> 2. 0x0000000000000044 <+68>: lea 0x0(%rip),%rdi # 0x4b <\_Z11DisplayMenuv+75> 3. 0x000000000000004b <+75>: callq 0x50 <\_Z11DisplayMenuv+80> | 1. This instruction loads the effective address 0x0(%rip), which is the next instruction, and moves its value into the %rsi register, the second argument register to be used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register to be used in the following function call. 3. This instruction calls the output function to display the string “- 3) Multiply -“, followed by a new line. |
| 1. 0x0000000000000050 <+80>: lea 0x0(%rip),%rsi # 0x57 <\_Z11DisplayMenuv+87> 2. 0x0000000000000057 <+87>: lea 0x0(%rip),%rdi # 0x5e <\_Z11DisplayMenuv+94> 3. 0x000000000000005e <+94>: callq 0x63 <\_Z11DisplayMenuv+99> | 1. This instruction loads the effective address 0x0(%rip), which is the next instruction, and moves its value into the %rsi register, the second argument register to be used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register to be used in the following function call. 3. This instruction calls the output function to display the string “- 4) Exit -“, followed by a new line. |
| 1. 0x0000000000000063 <+99>: lea 0x0(%rip),%rsi # 0x6a <\_Z11DisplayMenuv+106> 2. 0x000000000000006a <+106>: lea 0x0(%rip),%rdi # 0x71 <\_Z11DisplayMenuv+113> 3. 0x0000000000000071 <+113>: callq 0x76 <\_Z11DisplayMenuv+118> | 1. This instruction loads the effective address 0x0(%rip), which is the next instruction, and moves its value into the %rsi register, the second argument register to be used in the next function call. 2. This loads the effective address 0x0(%rip), the next instruction, into the %rdi register, the first argument register to be used in the following function call. 3. This instruction calls the output function to display the string “----------------“, followed by a new line. |
| 1. 0x0000000000000076 <+118>: nop 2. 0x0000000000000077 <+119>: pop %rbp 3. 0x0000000000000078 <+120>: retq | 1. This instruction is a No Operation instruction, which essentially does nothing, but instructs the execution to move to the next instruction without making any changes to register values, memory locations or flags. 2. This instruction restored the value of the base pointer register. 3. This instruction returns from the DisplayMenu function. |

**Step 3:** Convert the assembly code to binary.

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

| **Blocks of Assembly Code** | **C++ Code** |
| --- | --- |
| 1. 0x0000000000000079 <+0>:   push %rbp   1. 0x000000000000007a <+1>:   mov %rsp,%rbp   1. 0x000000000000007d <+4>:   sub $0x20,%rsp   1. 0x0000000000000081 <+8>:   mov %fs:0x28,%rax   1. 0x000000000000008a <+17>:   mov %rax,-0x8(%rbp)   1. 0x000000000000008e <+21>:   xor %eax,%eax | int main() { |
| 1. 0x0000000000000090 <+23>:   movl $0x0,-0x14(%rbp) | int choice = 0; |
| 1. 0x0000000000000097 <+30>:   mov -0x14(%rbp),%eax   1. 0x000000000000009a <+33>:   cmp $0x5,%eax   1. 0x000000000000009d <+36>:   je 0x308 <main+655> | while (choice != 5) { |
| 1. 0x00000000000000a3 <+42>:   lea 0x0(%rip),%rsi # 0xaa <main+49>   1. 0x00000000000000aa <+49>:   lea 0x0(%rip),%rdi # 0xb1 <main+56>   1. 0x00000000000000b1 <+56>:   callq 0xb6 <main+61> | cout << “----------------” << endl; |
| 1. 0x00000000000000b6 <+61>:   lea 0x0(%rip),%rsi # 0xbd <main+68>   1. 0x00000000000000bd <+68>:   lea 0x0(%rip),%rdi # 0xc4 <main+75>   1. 0x00000000000000c4 <+75>:   callq 0xc9 <main+80> | cout << “- 1) Add –“ << endl; |
| 1. 0x00000000000000c9 <+80>:   lea 0x0(%rip),%rsi # 0xd0 <main+87>   1. 0x00000000000000d0 <+87>:   lea 0x0(%rip),%rdi # 0xd7 <main+94>   1. 0x00000000000000d7 <+94>:   callq 0xdc <main+99> | cout << “- 2) Subtract –“ << endl; |
| 1. 0x00000000000000dc <+99>:   lea 0x0(%rip),%rsi # 0xe3 <main+106>   1. 0x00000000000000e3 <+106>:   lea 0x0(%rip),%rdi # 0xea <main+113>   1. 0x00000000000000ea <+113>:   callq 0xef <main+118> | cout << “- 3) Multiply –“ << endl; |
| 1. 0x00000000000000ef <+118>:   lea 0x0(%rip),%rsi # 0xf6 <main+125>   1. 0x00000000000000f6 <+125>:   lea 0x0(%rip),%rdi # 0xfd <main+132>   1. 0x00000000000000fd <+132>:   callq 0x102 <main+137> | cout << “- 4) Exit –“ << endl; |
| 1. 0x0000000000000102 <+137>:   lea 0x0(%rip),%rsi # 0x109 <main+144>   1. 0x0000000000000109 <+144>:   lea 0x0(%rip),%rdi # 0x110 <main+151>   1. 0x0000000000000110 <+151>:   callq 0x115 <main+156> | cout << “----------------” << endl; |
| 1. 0x0000000000000115 <+156>:   lea -0x14(%rbp),%rax   1. 0x0000000000000119 <+160>:   mov %rax,%rsi   1. 0x000000000000011c <+163>:   lea 0x0(%rip),%rdi # 0x123 <main+170>   1. 0x0000000000000123 <+170>:   callq 0x128 <main+175> | cin >> choice; |
| 1. 0x0000000000000128 <+175>:   mov -0x14(%rbp),%eax   1. 0x000000000000012b <+178>:   cmp $0x1,%eax   1. 0x000000000000012e <+181>:   jne 0x1c9 <main+336> | if (choice == 1) { |
| 1. 0x0000000000000134 <+187>:   lea -0x10(%rbp),%rax   1. 0x0000000000000138 <+191>:   mov %rax,%rsi   1. 0x000000000000013b <+194>:   lea 0x0(%rip),%rdi # 0x142 <main+201>   1. 0x0000000000000142 <+201>:   callq 0x147 <main+206>   1. 0x0000000000000147 <+206>:   mov %rax,%rdx   1. 0x000000000000014a <+209>:   lea -0xc(%rbp),%rax   1. 0x000000000000014e <+213>:   mov %rax,%rsi   1. 0x0000000000000151 <+216>:   mov %rdx,%rdi   1. 0x0000000000000154 <+219>:   callq 0x159 <main+224> | int num1, num2; |
| 1. 0x0000000000000159 <+224>:   mov -0x10(%rbp),%eax   1. 0x000000000000015c <+227>:   mov %eax,%esi   1. 0x000000000000015e <+229>:   lea 0x0(%rip),%rdi # 0x165 <main+236>   1. 0x0000000000000165 <+236>:   callq 0x16a <main+241> | cout << num1 |
| 1. 0x000000000000016a <+241>:   lea 0x0(%rip),%rsi # 0x171 <main+248>   1. 0x0000000000000171 <+248>:   mov %rax,%rdi   1. 0x0000000000000174 <+251>:   callq 0x179 <main+256> | << “ – “ |
| 1. 0x0000000000000179 <+256>:   mov %rax,%rdx   1. 0x000000000000017c <+259>:   mov -0xc(%rbp),%eax   1. 0x000000000000017f <+262>:   mov %eax,%esi   1. 0x0000000000000181 <+264>:   mov %rdx,%rdi   1. 0x0000000000000184 <+267>:   callq 0x189 <main+272> | << num2 |
| 1. 0x0000000000000189 <+272>:   lea 0x0(%rip),%rsi # 0x190 <main+279>   1. 0x0000000000000190 <+279>:   mov %rax,%rdi   1. 0x0000000000000193 <+282>:   callq 0x198 <main+287> | << “ = “ |
| 1. 0x0000000000000198 <+287>:   mov %rax,%rcx   1. 0x000000000000019b <+290>:   mov -0x10(%rbp),%edx   1. 0x000000000000019e <+293>:   mov -0xc(%rbp),%eax   1. 0x00000000000001a1 <+296>:   sub %eax,%edx   1. 0x00000000000001a3 <+298>:   mov %edx,%eax   1. 0x00000000000001a5 <+300>:   mov %eax,%esi   1. 0x00000000000001a7 <+302>:   mov %rcx,%rdi   1. 0x00000000000001aa <+305>:   callq 0x1af <main+310>   1. 0x00000000000001af <+310>:   mov %rax,%rdx   1. 0x00000000000001b2 <+313>:   mov 0x0(%rip),%rax # 0x1b9 <main+320>   1. 0x00000000000001b9 <+320>:   mov %rax,%rsi   1. 0x00000000000001bc <+323>:   mov %rdx,%rdi   1. 0x00000000000001bf <+326>:   callq 0x1c4 <main+331> | << num1 – num2 << endl; |
| 1. 0x00000000000001c4 <+331>:   jmpq 0x97 <main+30> | } // end of the first if statement |
| 1. 0x00000000000001c9 <+336>:   mov -0x14(%rbp),%eax   1. 0x00000000000001cc <+339>:   cmp $0x2,%eax   1. 0x00000000000001cf <+342>:   jne 0x268 <main+495> | if (choice == 2) { |
| 1. 0x00000000000001d5 <+348>:   lea -0x10(%rbp),%rax   1. 0x00000000000001d9 <+352>:   mov %rax,%rsi   1. 0x00000000000001dc <+355>:   lea 0x0(%rip),%rdi # 0x1e3 <main+362>   1. 0x00000000000001e3 <+362>:   callq 0x1e8 <main+367>   1. 0x00000000000001e8 <+367>:   mov %rax,%rdx   1. 0x00000000000001eb <+370>:   lea -0xc(%rbp),%rax   1. 0x00000000000001ef <+374>:   mov %rax,%rsi   1. 0x00000000000001f2 <+377>:   mov %rdx,%rdi   1. 0x00000000000001f5 <+380>:   callq 0x1fa <main+385> | int num1, num2; |
| 1. 0x00000000000001fa <+385>:   mov -0x10(%rbp),%eax   1. 0x00000000000001fd <+388>:   mov %eax,%esi   1. 0x00000000000001ff <+390>:   lea 0x0(%rip),%rdi # 0x206 <main+397>   1. 0x0000000000000206 <+397>:   callq 0x20b <main+402> | cout << num1 |
| 1. 0x000000000000020b <+402>: lea 0x0(%rip),%rsi # 0x212 <main+409> 2. 0x0000000000000212 <+409>: mov %rax,%rdi 3. 0x0000000000000215 <+412>: callq 0x21a <main+417> | << “ + “ |
| 1. 0x000000000000021a <+417>: mov %rax,%rdx 2. 0x000000000000021d <+420>: mov -0xc(%rbp),%eax 3. 0x0000000000000220 <+423>: mov %eax,%esi 4. 0x0000000000000222 <+425>: mov %rdx,%rdi 5. 0x0000000000000225 <+428>: callq 0x22a <main+433> | << num2 |
| 1. 0x000000000000022a <+433>:   lea 0x0(%rip),%rsi # 0x231 <main+440>   1. 0x0000000000000231 <+440>:   mov %rax,%rdi   1. 0x0000000000000234 <+443>:   callq 0x239 <main+448> | << “ = “ |
| 1. 0x0000000000000239 <+448>:   mov %rax,%rcx   1. 0x000000000000023c <+451>:   mov -0x10(%rbp),%edx   1. 0x000000000000023f <+454>:   mov -0xc(%rbp),%eax   1. 0x0000000000000242 <+457>:   add %edx,%eax   1. 0x0000000000000244 <+459>:   mov %eax,%esi   1. 0x0000000000000246 <+461>:   mov %rcx,%rdi   1. 0x0000000000000249 <+464>:   callq 0x24e <main+469>   1. 0x000000000000024e <+469>:   mov %rax,%rdx   1. 0x0000000000000251 <+472>:   mov 0x0(%rip),%rax # 0x258 <main+479>   1. 0x0000000000000258 <+479>:   mov %rax,%rsi   1. 0x000000000000025b <+482>:   mov %rdx,%rdi   1. 0x000000000000025e <+485>:   callq 0x263 <main+490> | << num1 + num2 << endl; |
| 1. 0x0000000000000263 <+490>:   jmpq 0x97 <main+30> | } |
| 1. 0x0000000000000268 <+495>:   mov -0x14(%rbp),%eax   1. 0x000000000000026b <+498>:   cmp $0x3,%eax   1. 0x000000000000026e <+501>:   jne 0x97 <main+30> | if (choice == 3) { |
| 1. 0x0000000000000274 <+507>:   lea -0x10(%rbp),%rax   1. 0x0000000000000278 <+511>:   mov %rax,%rsi   1. 0x000000000000027b <+514>:   lea 0x0(%rip),%rdi # 0x282 <main+521>   1. 0x0000000000000282 <+521>:   callq 0x287 <main+526>   1. 0x0000000000000287 <+526>:   mov %rax,%rdx   1. 0x000000000000028a <+529>:   lea -0xc(%rbp),%rax   1. 0x000000000000028e <+533>:   mov %rax,%rsi   1. 0x0000000000000291 <+536>:   mov %rdx,%rdi   1. 0x0000000000000294 <+539>:   callq 0x299 <main+544> | double num1, num2; |
| 1. 0x0000000000000299 <+544>:   mov -0x10(%rbp),%eax   1. 0x000000000000029c <+547>:   mov %eax,%esi   1. 0x000000000000029e <+549>:   lea 0x0(%rip),%rdi # 0x2a5 <main+556>   1. 0x00000000000002a5 <+556>:   callq 0x2aa <main+561> | cout << num1 |
| 1. 0x00000000000002aa <+561>:   lea 0x0(%rip),%rsi # 0x2b1 <main+568>   1. 0x00000000000002b1 <+568>:   mov %rax,%rdi   1. 0x00000000000002b4 <+571>:   callq 0x2b9 <main+576> | << “ / “ |
| 1. 0x00000000000002b9 <+576>:   mov %rax,%rdx   1. 0x00000000000002bc <+579>:   mov -0xc(%rbp),%eax   1. 0x00000000000002bf <+582>:   mov %eax,%esi   1. 0x00000000000002c1 <+584>:   mov %rdx,%rdi   1. 0x00000000000002c4 <+587>:   callq 0x2c9 <main+592> | << num2 |
| 1. 0x00000000000002c9 <+592>:   lea 0x0(%rip),%rsi # 0x2d0 <main+599>   1. 0x00000000000002d0 <+599>:   mov %rax,%rdi   1. 0x00000000000002d3 <+602>:   callq 0x2d8 <main+607> | << “ = “ |
| 1. 0x00000000000002d8 <+607>:   mov %rax,%rcx   1. 0x00000000000002db <+610>:   mov -0x10(%rbp),%eax   1. 0x00000000000002de <+613>:   mov -0xc(%rbp),%esi   1. 0x00000000000002e1 <+616>:   cltd   1. 0x00000000000002e2 <+617>:   idiv %esi   1. 0x00000000000002e4 <+619>:   mov %eax,%esi   1. 0x00000000000002e6 <+621>:   mov %rcx,%rdi   1. 0x00000000000002e9 <+624>:   callq 0x2ee <main+629>   1. 0x00000000000002ee <+629>:   mov %rax,%rdx   1. 0x00000000000002f1 <+632>:   mov 0x0(%rip),%rax # 0x2f8 <main+639>   1. 0x00000000000002f8 <+639>:   mov %rax,%rsi   1. 0x00000000000002fb <+642>:   mov %rdx,%rdi   1. 0x00000000000002fe <+645>:   callq 0x303 <main+650> | << num1 / num2 << endl; |
| 1. 0x0000000000000303 <+650>:   jmpq 0x97 <main+30> | } // end of the third if statement |
| 1. 0x0000000000000308 <+655>:   mov $0x0,%eax   1. 0x000000000000030d <+660>:   mov -0x8(%rbp),%rcx   1. 0x0000000000000311 <+664>:   xor %fs:0x28,%rcx   1. 0x000000000000031a <+673>:   je 0x321 <main+680>   1. 0x000000000000031c <+675>:   callq 0x321 <main+680>   1. 0x0000000000000321 <+680>:   leaveq | } // end of while loop  } // end of main function |
| DisplayMenu function | The following cells are the C++ code for the DisplayMenu function |
| 1. 0x0000000000000000 <+0>:   push %rbp   1. 0x0000000000000001 <+1>:   mov %rsp,%rbp | void DisplayMenu() { |
| 1. 0x0000000000000004 <+4>:   lea 0x0(%rip),%rsi # 0xb <\_Z11DisplayMenuv+11>   1. 0x000000000000000b <+11>:   lea 0x0(%rip),%rdi # 0x12 <\_Z11DisplayMenuv+18>   1. 0x0000000000000012 <+18>:   callq 0x17 <\_Z11DisplayMenuv+23> | cout << “----------------” << endl; |
| 1. 0x0000000000000017 <+23>:   lea 0x0(%rip),%rsi # 0x1e <\_Z11DisplayMenuv+30>   1. 0x000000000000001e <+30>:   lea 0x0(%rip),%rdi # 0x25 <\_Z11DisplayMenuv+37>   1. 0x0000000000000025 <+37>:   callq 0x2a <\_Z11DisplayMenuv+42> | cout << “- 1) Add –“ << endl; |
| 1. 0x000000000000002a <+42>:   lea 0x0(%rip),%rsi # 0x31 <\_Z11DisplayMenuv+49>   1. 0x0000000000000031 <+49>:   lea 0x0(%rip),%rdi # 0x38 <\_Z11DisplayMenuv+56>   1. 0x0000000000000038 <+56>:   callq 0x3d <\_Z11DisplayMenuv+61> | cout << “- 2) Subtract –“ << endl; |
| 1. 0x000000000000003d <+61>:   lea 0x0(%rip),%rsi # 0x44 <\_Z11DisplayMenuv+68>   1. 0x0000000000000044 <+68>:   lea 0x0(%rip),%rdi # 0x4b <\_Z11DisplayMenuv+75>   1. 0x000000000000004b <+75>:   callq 0x50 <\_Z11DisplayMenuv+80> | cout << “- 3) Multiply –“ << endl; |
| 1. 0x0000000000000050 <+80>:   lea 0x0(%rip),%rsi # 0x57 <\_Z11DisplayMenuv+87>   1. 0x0000000000000057 <+87>:   lea 0x0(%rip),%rdi # 0x5e <\_Z11DisplayMenuv+94>   1. 0x000000000000005e <+94>:   callq 0x63 <\_Z11DisplayMenuv+99> | cout << “- 4) Exit –“ << endl; |
| 1. 0x0000000000000063 <+99>:   lea 0x0(%rip),%rsi # 0x6a <\_Z11DisplayMenuv+106>   1. 0x000000000000006a <+106>:   lea 0x0(%rip),%rdi # 0x71 <\_Z11DisplayMenuv+113>   1. 0x0000000000000071 <+113>:   callq 0x76 <\_Z11DisplayMenuv+118> | cout << “----------------” << endl; |
| 1. 0x0000000000000076 <+118>:   nop   1. 0x0000000000000077 <+119>:   pop %rbp   1. 0x0000000000000078 <+120>:   retq | } |