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

Step 1: Convert the assembly code into C++ code.

Step 2: Explain the function of the converted C++ code.

| **Assembly Code** | **C++ Code** | | **Explanation of Functionality** |
| --- | --- | --- | --- |
| 1. movl −8(%rbp), %eax 2. sall $3, %eax 3. subl $3, %eax 4. movl %eax, −4(%rbp) | 1. int i, j; 2. i = i \*8; 3. i = i – 3; 4. j = i; | | 1. The integer i is the value stored at   -8(%rbp) an -8 offset or 8 bytes above the base pointer register, and integer j is the value that will be stored in  -4(%rbp), an -4 offset or 4 bytes above the base pointer register. the value of i is moved into the %eax register to be temporarily stored.   1. sall performs a left arithmetic shift on the %eax register by 3 bits. This is the same thing as multiplying the value in the %eax register by 2^3 = 8 or i = i \* 8 2. This line subtracts the value 3 from the value stored in the %$eax register (variable i) 3. The last line stores the value in the %eax register (variable i) into the variable j located at the memory location   -4(%rbp), an -4 offset or 4 bytes above the base pointer register. |
| 1. movl −8(%rbp), %eax 2. sall $2, %eax 3. subl $1, %eax   // The following three lines are abstracted away   1. leal 7(%rax), %edx 2. testl %eax, %eax 3. cmovs %edx, %eax 4. sarl $3, %eax 5. movl %eax, −4(%rbp) | 1. int i,j; 2. i = i \* 4; 3. i = i – 1; 4. i = i / 8; 5. j = I;   // Essentially the //following:  int i, j;  j = (((i \* 4) – 1)/8); | 1. The integer i is the value stored at   -8(%rbp), an -8 offset or 8 bytes above the base pointer register, and the value of i is moved into to %eax register to be temporarily stored. The integer j is the value to be stored at -4(%rbp), an -4 offset or 4 bytes above the base pointer register.   1. The value stored in register %eax is multiplied by 4 because it is shifted 2 bits to the left which gives us 2^2 = 4. 2. 1 is subtracted from the value stored in the %eax register. 3. Loads the effective address of what is stored 7 bits away from what is stored in the %rax register and store that value into the %edx register (another general purpose register that can be used for data manipulation). 4. Test the %eax register to see if it’s above zero. This is a bitwise AND operation between the contents of the %eax register and itself. A conditional flag, such as a sign flag (SF), is set based on the result. 5. Next, if the sign flag (SF) is set from the previous conditional check, then the contents of the %edx register are moved to the %eax register. 6. Take the value stored in the %eax register and divide it by 8 because it is shifted 3 bits to the right and therefore 2^3 = 8. Because we shifted right is why we divide. Shifting left would mean we multiply. 7. Last, the contents of the %eax register is moved into the memory location -4(%rbp), an -4 offset or 4 bits above the base pointer register. | |
| 1. movl −8(%rbp), %eax 2. leal 7(%rax), %edx 3. testl %eax, %eax 4. cmovs %edx, %eax 5. sarl $3, %eax 6. movl −8(%rbp), %edx 7. sall $2, %edx 8. addl %edx, %eax 9. movl %eax, −4(%rbp) | 1. int i, j;   // The following three lines //(lines 2-4) we would not //actually see in the C++ code //because they are abstracted //away and are happening //“behind the scene.”  //However, I am showing them //here to demonstrate that a //comparison is being made on //the value for i, which is stored //in the %eax register, and if //the value is zero, then we see //that the value stored in the //%edx register is then moved //into the %eax register, giving //it a new value.   1. int temp1; 2. if ( i == 0) 3. i = temp1; 4. i = i / 8; 5. int temp2 = i; 6. temp2 = temp2 \* 4; 7. i = i + temp2; 8. j = i;   Abstracted, the code is essentially the following:  int i, j;  j = (((i / 8) \* 4) + i) | | 1. The integer i is the value stored at -8(%rbp) and the value of i is moved into to %eax register to be temporarily stored. The integer j is the value to be stored at -4(%rbp) 2. Loads the effective address of what is stored 7 bits away from what is stored in the %rax register and store that value into the %edx register (another general purpose register that can be used for data manipulation). 3. Test the %eax register to see if it’s above zero. This is a bitwise AND operation between the contents of the %eax register and itself. A conditional flag, such as a sign flag (SF), is set based on the result. 4. Next, if the sign flag (SF) is set from the previous conditional check, then the contents of the %edx register are moved to the %eax register. 5. Take the value stored in the %eax register and divide it by 8 because it is shifted 3 bits to the right and therefore 2^3 = 8. Because we shifted right is why we divide. Shifting left would mean we multiply. 6. The value stored at -8(%rbp), an -8 offset or 8 bytes above the base pointer register is stored into the %edx register for future data manipulation.   (continued on next page…) |
|  |  | | 1. The value stored in register %edx is multiplied by 4 because it is shifted 2 bits to the left which gives us 2^2 = 4. 2. The value stored in the %edx register is added to the value stored in the %eax register and stored in the %eax register. 3. The resulting value in the %eax register is stored -4(%rbp) an -4 offset or 4 bytes above the base pointer register, which is the variable j. |