# **CS 410 Binary to Assembly Activity Template**

**Step 1:** List the binary file name.

**Step 2:** Identify the functions in the binary file.

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

**Step 4:** Align the blocks of assembly code with their corresponding function in the binary file.

**Step 5:** Explain the functionality of the blocks of assembly code.

## File One: [assignment3\_1.o]

| **Functions** | **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- | --- |
| main | 1. 0x000000000040057d <+0>: push %rbp 2. 0x000000000040057e <+1>: mov %rsp,%rbp 3. 0x0000000000400581 <+4>: mov $0x400634,%edi 4. 0x0000000000400586 <+9>: callq 0x400450 <puts@plt> | 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 moves the address $0x400634 into the %edi register, which is used by puts. The address $0x400634 points to the location of the string: “Ship to: John Smith” Loading this address into the %edi register prepares the string to be used in the following puts call. 4. Calls the function puts to print the string at the address stored in the %edi register. |
|  | 1. 0x000000000040058b <+14>: mov $0x400648,%edi 2. 0x0000000000400590 <+19>: callq 0x400450 <puts@plt> | 1. This moves the address $0x400648 into the %edi register, which is used by puts. The address $0x400648 points to the location of the string: “123 Los Angeles rd.” Loading this address into the %edi register prepares the string to be used in the following puts call. 2. Calls the function puts to print the string at the address stored in the %edi register. |
|  | 1. 0x0000000000400595 <+24>: mov $0x40065c,%edi 2. 0x000000000040059a <+29>: callq 0x400450 <puts@plt> | 1. This moves the address $0x40065c into the %edi register, which is used by puts. The address $0x40065c points to the location of the string: “Los Angeles, CA 90025” Loading this address into the %edi register prepares the string to be used in the following puts call. 2. Calls the function puts to print the string at the address stored in the %edi register. |
|  | 1. 0x000000000040059f <+34>: mov $0x0,%edi 2. 0x00000000004005a4 <+39>: callq 0x400480 <exit@plt> | 1. This instruction moves the value 0 into the %edi register to prepare for the next function call (exit). The value 0 is used to indicate that the program exits successfully. 2. This instruction calls the exit function to   terminate the program's execution. |

## File Two: [assignment3\_2.o]

| **Functions** | **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- | --- |
| main | 1. 0x000000000040062d <+0>: push %rbp 2. 0x000000000040062e <+1>: mov %rsp,%rbp 3. 0x0000000000400631 <+4>: sub $0x20,%rsp 4. 0x0000000000400635 <+8>: mov %fs:0x28,%rax 5. 0x000000000040063e <+17>: mov %rax,-0x8(%rbp) 6. 0x0000000000400642 <+21>: xor %eax,%eax 7. 0x0000000000400644 <+23>: mov $0x400714,%edi 8. 0x0000000000400649 <+28>: callq 0x4004e0 <puts@plt> | 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 setting up the stack frame for the main function. 3. This line subtracts the hexadecimal value 0x20 (32 in base 10 or decimal form) from the stack pointer register, allocating 32 bytes of space on the stack to be used for local variables or saving the states of registers when functions are called. 4. Here, the value stored at the fragment segment register address %fs:0x28 is moved to the %rax register 5. The value previously stored in the %rax register is stored at the location -0x8(%rbp), which is an -8 offset or 8 bytes above the base pointer register. 6. This instruction performs an XOR (exclusive OR) operation between the %eax register and itself. In an XOR operation, any number and itself results in a zero value. With XOR False and False = False, and True and True = False. Therefore, this instruction is effectively setting the value of the %eax register to zero. 7. This line moves the address $0x400714 into the %edi register, which is used by puts. The address $0x400714 points to the location of the string: “Please enter your name” Loading this address into the %edi register prepares it to be used by the following puts call. 8. This calls the puts function which is located at 0x4004e0. This prints the value stored in the %edi register (the previously stored string). |
|  | 1. 0x000000000040064e <+33>: lea -0x20(%rbp),%rax 2. 0x0000000000400652 <+37>: mov %rax,%rsi 3. 0x0000000000400655 <+40>: mov $0x40072b,%edi 4. 0x000000000040065a <+45>: mov $0x0,%eax 5. 0x000000000040065f <+50>: callq 0x400520 <\_\_isoc99\_scanf@plt> | 1. This loads the effective address at the location -0x20(%rbp), an -20 offset or 20 bytes above the base pointer register, into the %rax register. 2. This moves the value that was previously stored in the %rax register to the %rsi register, which is used for the second argument in the scanf function. 3. This moves the address $0x40072b (“%s” – formatted string) to the %edi register for use with the scanf function (reading input from the user) 4. This clears the %eax register by moving the value 0 ($0x0) into the %eax register. 5. This calls the scanf function to read input from the user. |
|  | 1. 0x0000000000400664 <+55>: lea -0x20(%rbp),%rax 2. 0x0000000000400668 <+59>: mov %rax,%rsi 3. 0x000000000040066b <+62>: mov $0x40072e,%edi 4. 0x0000000000400670 <+67>: mov $0x0,%eax 5. 0x0000000000400675 <+72>: callq 0x4004f0 <printf@plt> | 1. This instruction loads the effective address of -0x20(%rbp), an -20 offset or 20 bytes above the base pointer register, into the %rax register. 2. This moves the previously stored value in the %rax register to the %rsi register, which will be used as the second argument for the printf function. 3. The moves the value stored at the address $0x40072e, the string: “Welcome Mr. %s\n” to the %edi register in preparation for use by the puts function. This string includes the formatted string previously taken from input. 4. This clears the %eax register by moving the value 0 ($0x0) into the %eax register. 5. This calls the printf function to print the output, which is the value stored in the %edi register. |
|  | 1. 0x000000000040067a <+77>: mov $0x0,%edi 2. 0x000000000040067f <+82>: callq 0x400530 <exit@plt> | 1. This instruction moves the value 0 into the %edi register to prepare for the next function call (exit). The value 0 is used to indicate that the program exits successfully. 2. This instruction calls the exit function to terminate the program's execution. |

## File Three: [assignment3\_3.o]

| **Functions** | **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- | --- |
| main | 1. 0x0000000000400641 <+0>: push %rbp 2. 0x0000000000400642 <+1>: mov %rsp,%rbp 3. 0x0000000000400645 <+4>: sub $0x10,%rsp 4. 0x0000000000400649 <+8>: mov $0x400734,%edi 5. 0x000000000040064e <+13>: callq 0x4004e0 <puts@plt> | 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 instruction subtracts the value $0x10 (16 in decimal form) from the stack pointer register, allocating 16 bytes on the stack for local variables. 4. This moves the address $0x400734 into the %edi register, which is used by puts. The address $0x400734 points to the location of the string: “Enter two numbers:” Loading this address into the %edi register prepares the string to be used in the following puts call. 5. Calls the function puts to print the string at the address stored in the %edi register. |
|  | 1. 0x0000000000400653 <+18>: lea -0x8(%rbp),%rdx 2. 0x0000000000400657 <+22>: lea -0xc(%rbp),%rax 3. 0x000000000040065b <+26>: mov %rax,%rsi 4. 0x000000000040065e <+29>: mov $0x400747,%edi 5. 0x0000000000400663 <+34>: mov $0x0,%eax 6. 0x0000000000400668 <+39>: callq 0x400520 <\_\_isoc99\_scanf@plt> | 1. This loads the effective address of the value stored at -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register, into the %rdx register. This is the first variable. 2. This loads the effective address of the value stored at -0xc(%rbp), (c converted to decimal is 12) therefore this is an offset of -12 or 12 bytes above the base pointer register, and loads this into the %rax register. This is the second variable that will be used. 3. This moves the address of the value previously stored in the %rax register into the %rsi register. 4. This instruction moves the address $0x400747 into the %edi register in preparation for use in the scanf function call. The address $0x400747 points to the formatted string “%d %d” which is the expected format of the input from the scanf function call. 5. This clears the %eax register by moving the value 0 ($0x0) into the %eax register. 6. This calls the scanf function to read input from the user |
|  | 1. 0x000000000040066d <+44>: mov -0x8(%rbp),%edx 2. 0x0000000000400670 <+47>: mov -0xc(%rbp),%eax 3. 0x0000000000400673 <+50>: mov %edx,%esi 4. 0x0000000000400675 <+52>: mov %eax,%edi 5. 0x0000000000400677 <+54>: callq 0x40062d <AddNumbers> | 1. This moves the address of the value stored at -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register, into the %edx register. This is the first variable to be passed to the AddNumbers function. 2. This moves the address of the value stored at -0xc(%rbp), (c converted to decimal is 12) therefore this is an offset of -12 or 12 bytes above the base pointer register, into the %eax register. This is the second variable that will be passed to the AddNumbers function. 3. This moves the value previously stored in the %edx register into the %esi register in preparation for the AddNumbers function call. 4. This moves the value previously stored in the %eax register into the %edi register in preparation for the AddNumbers function call. 5. This calls the function AddNumbers. |
|  | 1. 0x000000000040067c <+59>: mov %eax,-0x4(%rbp) 2. 0x000000000040067f <+62>: mov -0x8(%rbp),%edx 3. 0x0000000000400682 <+65>: mov -0xc(%rbp),%eax 4. 0x0000000000400685 <+68>: mov -0x4(%rbp),%ecx 5. 0x0000000000400688 <+71>: mov %eax,%esi 6. 0x000000000040068a <+73>: mov $0x40074d,%edi 7. 0x000000000040068f <+78>: mov $0x0,%eax 8. 0x0000000000400694 <+83>: callq 0x4004f0 <printf@plt> | 1. This stores the result of the call to AddNumbers in a local variable by moving the value in the %eax register and storing it at the location -0x4(%rbp), an -4 offset or 4 bytes above the base pointer register. 2. This moves the value stored at -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register, into the %edx register to prepare it for use in the call to printf. 3. This moves the value stored at   -0xc(%rbp) (c converted to decimal is 12) which is an -12 offset or 12 bytes above the base pointer register, into the %eax register to prepare it for use in the call to printf.   1. This moves the value stored at -0x4(%rbp), an -4 offset or 4 bytes above the base pointer register, into the %ecx register to prepare it for use in the call to printf 2. This moves the value stored in the %eax register into the %esi register in preparation for the call to printf. This is also because the %eax register is about to be set to 0 for the printf function call. Printf expects the %eax register to be cleared. 3. This moves the address at $0x40074d into the %edi register which is used by the printf. The address $0x40074d points to the formatted string: “%d + %d = %d\n”. Loading this address into the %edi register prepares it for use by printf. 4. This clears the %eax register by moving the value 0 ($0x0) into the %eax register. 5. This calls the function printf and prints the string with its associated values. |
|  | 1. 0x0000000000400699 <+88>: mov $0x0,%edi 2. 0x000000000040069e <+93>: callq 0x400530 <exit@plt> | 1. This instruction moves the value 0 into the %edi register to prepare for the next function call (exit). The value 0 is used to indicate that the program exits successfully. 2. This instruction calls the exit function to terminate the program's execution. |
| AddNumbers | 1. 0x000000000040062d <+0>: push %rbp 2. 0x000000000040062e <+1>: mov %rsp,%rbp 3. 0x0000000000400631 <+4>: mov %edi,-0x4(%rbp) 4. 0x0000000000400634 <+7>: mov %esi,-0x8(%rbp) 5. 0x0000000000400637 <+10>: mov -0x8(%rbp),%eax 6. 0x000000000040063a <+13>: mov -0x4(%rbp),%edx 7. 0x000000000040063d <+16>: add %edx,%eax 8. 0x000000000040063f <+18>: pop %rbp 9. 0x0000000000400640 <+19>: retq | 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 instruction moves the value previously stored in the %edi register into the location -0x4(%rbp) an -4 offset or four bytes above the base pointer register. This is the value for the first argument of the AddNumbers function. 4. This instruction moves the value previously stored in the %esi register to the location -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register. This is the value for the second argument of the AddNumbers function. 5. This moves the value of the second argument of the function, which is stored at -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register, into the %eax register for future use (Adding two variables together). 6. This moves the value of the first argument of the function, which is stored at -0x4(%rbp) an -4 offset or 4 bytes above the base pointer register, into the %edx register for future use (adding two variables together). 7. This instruction adds the values stored in the %edx register with the value stored in the %eax register and stores the result back in the %eax register. 8. This restores the previous stack frame by popping the current stack frame. This is effectively cleaning up the stack frame. 9. This returns from the AddNumbers function based on the return address stored on the stack. |

## File Four: [assignment3\_4.o]

| **Functions** | **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- | --- |
| main | 1. 0x0000000000400717 <+0>: push %rbp 2. 0x0000000000400718 <+1>: mov %rsp,%rbp 3. 0x000000000040071b <+4>: sub $0x10,%rsp 4. 0x000000000040071f <+8>: movl $0x0,-0x8(%rbp) 5. 0x0000000000400726 <+15>: jmp 0x4007a0 <main+137> 6. 0x0000000000400728 <+17>: mov $0x0,%eax 7. 0x000000000040072d <+22>: callq 0x4006df <DisplayMenu> | 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 instruction subtracts the value $0x10 (16 in decimal form) from the stack pointer register, allocating 16 bytes on the stack for local variables. 4. This initializes the local variable stored at -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register to 0 by moving the value 0 into the location at -0x8(%rbp). 5. This instruction jumps to the instruction stored at 0x4007a0, which is line 137 of the main function. When this happens, the variable stored at -0x8(%rbp) is then moved to the %eax register where a comparison is made to determine if the program should continue to loop or exit and terminate the program’s execution. This is a menu-driven program and uses a while loop based on the variable stored at -0x8(%rbp). 6. This instruction clears the %eax register to prepare for the function call to DisplayMenu by moving the value 0 into the %eax register. This is reached after the previous jmp is completed where if the program does not terminate execution, the program is jumped back to this instruction. 7. This calls the function DisplayMenu, which display’s the program’s menu to output. |
|  | 1. 0x0000000000400732 <+27>: mov $0x400886,%edi 2. 0x0000000000400737 <+32>: callq 0x4004e0 <puts@plt> | 1. After the DisplayMenu function is called and performs its operations, the function returns to this instruction where it left off. This instruction moves the value stored at $0x400886 into the %edi register. The address $0x400886 points to the location of the string: “Enter your number:” Moving it to the %edi register prepares it for use by the puts function. 2. Calls the puts function to output the previously stored string in the %edi register. |
|  | 1. 0x000000000040073c <+37>: lea -0x8(%rbp),%rax 2. 0x0000000000400740 <+41>: mov %rax,%rsi 3. 0x0000000000400743 <+44>: mov $0x400899,%edi 4. 0x0000000000400748 <+49>: mov $0x0,%eax 5. 0x000000000040074d <+54>: callq 0x400520 <\_\_isoc99\_scanf@plt> | 1. This loads the effective address of the variable stored at -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register, into the %rax register. This is the variable that is used to determine what code is going to be executed. 2. This moves the value previously stored in the %rax register into the %rsi register to prepare it for use with the scanf function. 3. This moves the address $0x400899 into the %edi register. The address $0x400899 points to the location of the format for the string that will be used in the scanf function call when getting input. The string format is: “%d” 4. This instruction clears the %eax register in preparation for the scanf function call by moving the value of 0 into the %eax register. 5. Calls the function scanf to read input. |
|  | 1. 0x0000000000400752 <+59>: mov -0x8(%rbp),%eax 2. 0x0000000000400755 <+62>: cmp $0x3,%eax 3. 0x0000000000400758 <+65>: je 0x40077a <main+99> 4. 0x000000000040075a <+67>: mov $0x40089c,%edi 5. 0x000000000040075f <+72>: callq 0x4004e0 <puts@plt> | 1. This instruction moves the value located at -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register, into the %eax register, which is going to be used to make a comparison to determine whether or not the user input the value 3, which would end the while loop that is driving the program. 2. Here we make the comparison to see if the value stored in the %eax register is the value 3. If the value is not three the following line will not be executed. During the comparison, if the values equal each other a Zero Flag is set which instructs the next line to be executed. 3. je stands for jump if equal. If the Zero Flag is set from the previous comparison, the execution jumps to line 99 in the main function. Otherwise, it continues to the next instruction. 4. This moves the address $0x40089c into the %edi register which is used by puts. The address $0x40089c points to the string: “Enter a number:” Loading it into the %edi register prepares it for use with the following puts call. 5. Calls the function puts to print the string at the address stored in the %edi register. |
|  | 1. 0x0000000000400764 <+77>: lea -0x4(%rbp),%rax 2. 0x0000000000400768 <+81>: mov %rax,%rsi 3. 0x000000000040076b <+84>: mov $0x400899,%edi 4. 0x0000000000400770 <+89>: mov $0x0,%eax 5. 0x0000000000400775 <+94>: callq 0x400520 <\_\_isoc99\_scanf@plt> | 1. This instruction loads the effective address of -0x4(%rbp), an -4 offset or 4 bytes above the base pointer register, into the %rax register. This is the variable that will be used to pass to the PrintSum or PrintFact functions. 2. The value previously stored in the %rax register is moved into the %rsi register to prepare it for use by the scanf function call. 3. The address located at $0x400899 is moved into the %edi register in preparation for use with the following scanf function call. The address $0x400899 points to the formatted string: “%d” which is the expected format of the input that will be taken from the scanf function call. 4. This instruction clears the %eax register in preparing for the scanf function call by moving the value 0 into the %eax register. 5. This calls the scanf function to read input from the user. |
|  | 1. 0x000000000040077a <+99>: mov -0x8(%rbp),%eax 2. 0x000000000040077d <+102>: cmp $0x1,%eax 3. 0x0000000000400780 <+105>: jne 0x40078e <main+119> 4. 0x0000000000400782 <+107>: mov -0x4(%rbp),%eax 5. 0x0000000000400785 <+110>: mov %eax,%edi 6. 0x0000000000400787 <+112>: callq 0x40062d <PrintFact> | 1. This instruction moves the value stored at -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register, into the %eax register to be used for the following comparison. 2. This compares the value in the %eax register to the value of 1. If the value is not 1, the following line will be executed. During the comparison, if the values equal each other, a Zero Flag is set which instructs the next line to NOT be executed. 3. jne stands for jump if not equal. Therefore, if the Zero Flag is not set from the previous comparison, the execution of the code jumps to the address 0x40078e, which is line 119 of the main function. Otherwise, it continues to the next instruction. 4. This instruction moves the value stored at -0x4(%rbp), an -4 offset or 4 bytes above the base pointer register, into the %eax register. This is the variable to be passed in the function call to PrintFact. 5. The value stored in the %eax register is moved into the %edi register to prepare for the function call to PrintFact. 6. Calls the function PrintFact, passing along the value stored in the %edi register as an argument. |
|  | 1. 0x000000000040078c <+117>: jmp 0x4007a0 <main+137> 2. 0x000000000040078e <+119>: mov -0x8(%rbp),%eax 3. 0x0000000000400791 <+122>: cmp $0x2,%eax 4. 0x0000000000400794 <+125>: jne 0x4007a0 <main+137> 5. 0x0000000000400796 <+127>: mov -0x4(%rbp),%eax 6. 0x0000000000400799 <+130>: mov %eax,%edi 7. 0x000000000040079b <+132>: callq 0x400688 <PrintSum> | 1. This instructs the execution of the code to jump to the address $0x4007a0, which points to line 137 of the main function. 2. This line is reached based on the comparison made in the previous block of assembly code. This instruction moves the value stored at -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register, into the %eax register to prepare to be used for the following comparison. 3. This compares the value in the %eax register to the value of 2. If the value is not 2, the following line will be executed. During the comparison, if the values equal each other, a Zero Flag is set which instructs the next line to NOT be executed. 4. jne stands for jump if not equal. Therefore, if the Zero Flag is not set from the previous comparison, the execution of the code jumps to the address 0x4007a0, which is line 137 of the main function. Otherwise, it continues to the next instruction. 5. This instruction moves the value stored at -0x4(%rbp), an -4 offset or 4 bytes above the base pointer register, into the %eax register. This is the variable to be passed in the function call to PrintSum. 6. The value stored in the %eax register is moved to the %edi register to prepare for the function call to PrintSum. 7. Calls the function PrintSum, passing along the value stored in the %edi register as an argument. |
|  | 1. 0x00000000004007a0 <+137>: mov -0x8(%rbp),%eax 2. 0x00000000004007a3 <+140>: cmp $0x3,%eax 3. 0x00000000004007a6 <+143>: jne 0x400728 <main+17> 4. 0x00000000004007a8 <+145>: mov $0x0,%edi 5. 0x00000000004007ad <+150>: callq 0x400530 <exit@plt> | 1. This moves the value stored at -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register, into the %eax register for use with the following comparison. 2. This compares the value in the %eax register to the value of 3. If the value is not 3, the following line will be executed. During the comparison, if the values equal each other, a Zero Flag is set which instructs the next line to NOT be executed. 3. Jne stands for jump if not equal. Therefore, if the Zero Flag is not set from the previous comparison, the execution of the code jumps to the address 0x400728, which is line 17 of the main function. Otherwise, it continues to the next instruction. 4. This instruction moves the value 0 into the %edi register to prepare for the next function call (exit). The value 0 is used to indicate that the program exits successfully. 5. This instruction calls the exit function to terminate the program's execution. |
| DisplayMenu | 1. 0x00000000004006df <+0>: push %rbp 2. 0x00000000004006e0 <+1>: mov %rsp,%rbp 3. 0x00000000004006e3 <+4>: mov $0x400851,%edi 4. 0x00000000004006e8 <+9>: callq 0x4004e0 <puts@plt> | 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 instruction moves the address $0x400851 into the %edi register in preparation for the function call to puts. The address $0x400851 points to a string containing the ‘\*’ character repeated 18 times. 4. Calls the function puts to print the string at the address stored in the %edi register. |
|  | 1. 0x00000000004006ed <+14>: mov $0x400864,%edi 2. 0x00000000004006f2 <+19>: callq 0x4004e0 <puts@plt> | 1. This instruction moves the address $0x400864 into the %edi register in preparation for the function call to puts. The address $0x400864 points to the string “1. Factorial”. 2. Calls the function puts to print the string at the address stored in the %edi register. |
|  | 1. 0x00000000004006f7 <+24>: mov $0x400871,%edi 2. 0x00000000004006fc <+29>: callq 0x4004e0 <puts@plt> | 1. This instruction moves the address $0x400871 into the %edi register in preparation for the function call to puts. The address $0x400871 points to the string: “2: Summation”. 2. Calls the function puts to print the string at the address stored in the %edi register. |
|  | 1. 0x0000000000400701 <+34>: mov $0x40087e,%edi 2. 0x0000000000400706 <+39>: callq 0x4004e0 <puts@plt> | 1. This instruction moves the address $0x40087e into the %edi register in preparation for the function call to puts. The address $0x40087e points to the string: “3. Quit”. 2. Calls the function puts to print the string at the address stored in the %edi register. |
|  | 1. 0x000000000040070b <+44>: mov $0x400851,%edi 2. 0x0000000000400710 <+49>: callq 0x4004e0 <puts@plt> | 1. This instruction moves the address $0x400851 into the %edi register in preparation for the function call to puts. The address $0x400851 points to a string containing the ‘\*’ character repeated 18 times. 2. Calls the function puts to print the string at the address stored in the %edi register. |
|  | 1. 0x0000000000400715 <+54>: pop %rbp 2. 0x0000000000400716 <+55>: retq | 1. This restores the previous stack frame by popping the current stack frame. This is effectively cleaning up the stack frame. 2. This returns from the DisplayMenu function based on the return address stored on the stack. |
| PrintSum | 1. 0x0000000000400688 <+0>: push %rbp 2. 0x0000000000400689 <+1>: mov %rsp,%rbp 3. 0x000000000040068c <+4>: sub $0x20,%rsp 4. 0x0000000000400690 <+8>: mov %edi,-0x14(%rbp) 5. 0x0000000000400693 <+11>: movl $0x0,-0x4(%rbp) 6. 0x000000000040069a <+18>: mov -0x14(%rbp),%eax 7. 0x000000000040069d <+21>: mov %eax,-0x8(%rbp) 8. 0x00000000004006a0 <+24>: jmp 0x4006c0 <PrintSum+56> 9. 0x00000000004006a2 <+26>: mov -0x8(%rbp),%eax 10. 0x00000000004006a5 <+29>: add %eax,-0x4(%rbp) 11. 0x00000000004006a8 <+32>: mov -0x8(%rbp),%eax 12. 0x00000000004006ab <+35>: mov %eax,%esi 13. 0x00000000004006ad <+37>: mov $0x400844,%edi 14. 0x00000000004006b2 <+42>: mov $0x0,%eax 15. 0x00000000004006b7 <+47>: callq 0x4004f0 <printf@plt>   Starting from instruction 9) 🡪  This loop is used to calculate the summation of the value passed in as an argument to the function PrintSum. The result is calculated by taking the current result which starts at zero, then adds the initial value that was passed in as the function argument. Each iteration decrements the passed-in value by 1, and the next value of result is calculated result = result + the loop control variable. | 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 instruction subtracts the value $0x20 (32 in decimal form) from the stack pointer register, allocating 32 bytes on the stack for local variables. 4. This moves the value stored in the %edi register, which is the argument that was passed in when the function call to PrintSum was made, storing its value at -0x14(%rbp), which is an -14 offset or 14 bytes above the base pointer register. 5. This instruction moves the value 0 into the location -0x4(%rbp), an -4 offset or 4 bytes above the base pointer register. 6. This moves the value stored at -0x14(%rbp), a -14 offset or 14 bytes above the base pointer register, into the %eax register. 7. The value stored in the %eax register is then moved and stored at the location -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register (essentially storing the value that was passed in into a second variable). 8. This instruction jumps to the instruction located at the address 0x4006c0, which is line 56 of the PrintSum function. 9. This line begins the logic inside a while loop. This instruction moves the value stored at the location -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register, into the %eax register to prepare it for use. 10. This instruction adds the value stored in the %eax register to the value at the location -0x4(%rbp), an -4 offset or 4 bytes above the base pointer register, and stores the calculation back into the location at -0x4(%rbp). 11. This instruction moves the value stored at -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register, into the %eax register. 12. This moves the value in the %eax register into the %esi register, preparing it for use with the printf function call.   ( continued on next page…) |
|  |  | 1. This moves the address $0x400844 into the %edi register in preparation for use in the printf function call. The address at $0x400844 points to the formatted string: “%d ”. 2. This instruction clears the %eax register in preparing for the printf function call by moving the value 0 into the %eax register. 3. Calls the printf function to output the value previously stored in the %edi register. |
|  | 1. 0x00000000004006bc <+52>: subl $0x1,-0x8(%rbp) 2. 0x00000000004006c0 <+56>: cmpl $0x0,-0x8(%rbp) 3. 0x00000000004006c4 <+60>: jg 0x4006a2 <PrintSum+26> 4. 0x00000000004006c6 <+62>: mov -0x4(%rbp),%eax 5. 0x00000000004006c9 <+65>: mov %eax,%esi 6. 0x00000000004006cb <+67>: mov $0x400848,%edi 7. 0x00000000004006d0 <+72>: mov $0x0,%eax 8. 0x00000000004006d5 <+77>: callq 0x4004f0 <printf@plt> | 1. This instruction subtracts the value 1 from the value stored at the location -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register. 2. This compares the value 0 with the value stored at the location -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register. 3. If from the previous comparison, the value stored at -0x8(%rbp) is greater than 0, this line instructs the execution to jump to line 26 of the PrintSum function. (This is a loop that ends once the value stored in -0x8(%rbp) is zero. 4. This line is executed once the previous loop ends. This instruction moves the value stored at location -0x4(%rbp), an -4 offset or 4 bytes above the base pointer register, into the %eax register. 5. The value stored in the %eax register is then moved to the %esi register for preparation for use in the printf function call. 6. The address $0x400848 is moved into the %edi register, which is used by the printf function call. The address $0x400848 points to the formatted string: “ [%d] \n”. 7. This instruction clears the %eax register in preparing for the printf function call by moving the value 0 into the %eax register. 8. Calls the printf function to output the value previously stored in the %edi register. |
|  | 1. 0x00000000004006da <+82>: mov -0x4(%rbp),%eax 2. 0x00000000004006dd <+85>: leaveq 3. 0x00000000004006de <+86>: retq | 1. This instruction moves the value stored at the location -0x4(%rbp), an -4 offset or 4 bytes above the base pointer register, into the %eax register. This is the return value. 2. This restores the previous stack frame (cleans it up). 3. This instruction returns back to the main function based on the return address stored on the stack frame. |
| PrintFact | 1. 0x000000000040062d <+0>: push %rbp 2. 0x000000000040062e <+1>: mov %rsp,%rbp 3. 0x0000000000400631 <+4>: sub $0x20,%rsp 4. 0x0000000000400635 <+8>: mov %edi,-0x14(%rbp) 5. 0x0000000000400638 <+11>: movl $0x1,-0x4(%rbp) 6. 0x000000000040063f <+18>: mov -0x14(%rbp),%eax 7. 0x0000000000400642 <+21>: mov %eax,-0x8(%rbp) 8. 0x0000000000400645 <+24>: jmp 0x400669 <PrintFact+60> 9. 0x0000000000400647 <+26>: mov -0x4(%rbp),%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 instruction subtracts the value $0x20 (32 in decimal form) from the stack pointer register, allocating 32 bytes on the stack for local variables. 4. This moves the value stored in the %edi register, which is the argument that was passed in when the function call to PrintFact was made, storing its value at -0x14(%rbp), which is an -14 offset or 14 bytes above the base pointer register. 5. This instruction moves the value 1 into the location -0x4(%rbp), an -4 offset or 4 bytes above the base pointer register. 6. This moves the value stored at -0x14(%rbp), a -14 offset or 14 bytes above the base pointer register, into the %eax register. 7. The value stored in the %eax register is then moved and stored at the location -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register (essentially storing the value that was passed in into a second variable). 8. This instruction jumps to the instruction located at the address 0x400669, which is line 60 of the PrintFact function. 9. This line begins the logic inside a while loop. This instruction moves the value stored at the location -0x4(%rbp), an -4 offset or 4 bytes above the base pointer register, into the %eax register to prepare it for use. |
|  | 1. 0x000000000040064a <+29>: imul -0x8(%rbp),%eax 2. 0x000000000040064e <+33>: mov %eax,-0x4(%rbp) 3. 0x0000000000400651 <+36>: mov -0x8(%rbp),%eax 4. 0x0000000000400654 <+39>: mov %eax,%esi 5. 0x0000000000400656 <+41>: mov $0x400844,%edi 6. 0x000000000040065b <+46>: mov $0x0,%eax 7. 0x0000000000400660 <+51>: callq 0x4004f0 <printf@plt>   This loop is used to calculate the factorial of the value passed in as an argument to the function PrintFact. The result is calculated by multiplying the starting value that is passed into the PrintFact function as an argument and multiplying it by the starting value of 1 stored in the -0x4(%rbp) location. In each iteration, the argument’s value is decremented by one and multiplied by the previous iteration’s result. The final iteration results in the factorial of the passed-in value (argument). | 1. This instruction multiples the value stored in the location -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register, by the value stored in the %eax register, then stores it back into the %eax register. 2. This moves the value stored into the %eax register into the location -0x4(%rbp), an -4 offset or 4 bytes above the base pointer register. 3. This moves the value stored in the location -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register, into the %eax register. 4. The value stored in the %eax register is then moved to the %esi register in preparation for use by the printf function call. 5. This moves the address $0x400844 into the %edi register in preparation for use in the printf function call. The address at $0x400844 points to the formatted string: “%d ”. 6. This instruction clears the %eax register in preparing for the printf function call by moving the value 0 into the %eax register. 7. Calls the printf function to output the value previously stored in the %edi register. |
|  | 1. 0x0000000000400665 <+56>: subl $0x1,-0x8(%rbp) 2. 0x0000000000400669 <+60>: cmpl $0x0,-0x8(%rbp) 3. 0x000000000040066d <+64>: jg 0x400647 <PrintFact+26> 4. 0x000000000040066f <+66>: mov -0x4(%rbp),%eax 5. 0x0000000000400672 <+69>: mov %eax,%esi 6. 0x0000000000400674 <+71>: mov $0x400848,%edi 7. 0x0000000000400679 <+76>: mov $0x0,%eax 8. 0x000000000040067e <+81>: callq 0x4004f0 <printf@plt> | 1. This instruction subtracts the value 1 from the value stored at location -0x8(%rbp), an -8 offset or 8 bytes above the base pointer register, and keeps it stored at this location. This variable is used as the control variable for the loop. 2. This line compares the loop control variable to zero and if the value is greater than 0, the next instruction is executed to either start the loop or continue the next iteration of the loop. 3. If from the previous comparison, the value stored at -0x8(%rbp) is greater than 0, this line instructs the execution to jump to line 26 of the PrintFact function. (This is a loop that ends once the value stored in -0x8(%rbp) is zero. 4. This line is executed once the previous loop ends. This instruction moves the value stored at location -0x4(%rbp), an -4 offset or 4 bytes above the base pointer register, into the %eax register. 5. The value stored in the %eax register is then moved to the %esi register for preparation for use in the printf function call. 6. The address $0x400848 is moved into the %edi register, which is used by the printf function call. The address $0x400848 points to the formatted string: “ [%d] \n”. 7. This instruction clears the %eax register in preparing for the printf function call by moving the value 0 into the %eax register. 8. Calls the printf function to output the value previously stored in the %edi register. |
|  | 1. 0x0000000000400683 <+86>: mov -0x4(%rbp),%eax 2. 0x0000000000400686 <+89>: leaveq 3. 0x0000000000400687 <+90>: retq | 1. This instruction moves the value stored at the location -0x4(%rbp), an -4 offset or 4 bytes above the base pointer register, into the %eax register. This is the return value. 2. This restores the previous stack frame (cleans it up). 3. This instruction returns back to the main function based on the return address stored on the stack frame. |