# **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** |
| --- | --- | --- |
| int main() | push %rbp  mov %rsp,%rbp | Reserve the Stack pointer in rbp so we don’t mess the original SP value. |
| mov $0x400634,%edi | Move the string value reference to rdi |
| call 0x400450 <puts@plt> | Print Ship to: John Smith. |
| mov $0x400648,%edi | Move the string value reference to rdi |
| call 0x400450 <puts@plt> | Print 123 Los Angeles Rd. |
| mov $0x40065c,%edi | Move the string value reference to rdi |
| call 0x400450 <puts@plt> | Print Los Angeles, CA 90025 |
| mov $0x0,%edi | Set the return value to 0 |
| call 0x400480 <exit@plt> |  |

## File Two: assignment3\_2.o

| **Functions** | **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- | --- |
| int main() | push %rbp  mov %rsp,%rbp  sub $0x20,%rsp | Reserve the Stack Pointer value.  Reserve 32 bytes in the stack |
|  | mov %fs:0x28,%rax  mov %rax,-0x8(%rbp) | Copy the value in memory address to RAX |
|  | xor %eax,%eax | Set EAX zero |
|  | mov $0x400714,%edi | Get the reference of the text value “ |
|  | call 0x4004e0 <puts@plt> | Printout Please enter your name |
|  | lea -0x20(%rbp),%rax |  |
|  | mov %rax,%rsi |  |
|  | mov $0x40072b,%edi |  |
|  | mov $0x0,%eax | Set EAX 0 |
|  | call 0x400520 <\_\_isoc99\_scanf@plt> | Read from the string value in the buffer using scanf and %d which is declared in the const segment. |
|  | lea -0x20(%rbp),%rax |  |
|  | mov %rax,%rsi |  |
|  | mov $0x40072b,%edi |  |
|  | mov $0x0,%eax |  |
|  | call 0x4004f0 <printf@plt> | Print out Welcome Mr. “entered value” |
|  | mov $0x0,%edi | Store the return value  edi |
|  | call 0x400530 <exit@plt> | Exit the procedure |

## File Three: assignment3\_3.o

| **Functions** | **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- | --- |
| int main() | push%rbp  mov %rsp,%rbp  sub $0x10,%rsp | Reserve the Stack Pointer value.  Reserve 16 bytes in the stack for two variables |
|  | mov $0x400734,%edi  call0x4004e0 <puts@plt> | Store the reference for “Enter two numbers:”  and print out the string |
|  | lea -0x8(%rbp),%rdx  lea -0xc(%rbp),%rax  mov %rax,%rsi  mov $0x400747,%edi  mov $0x0,%eax  call0x400520 <\_\_isoc99\_scanf@plt> | Get the address of both variables and the address of “%d %d”, prepare EAX register to receive the results.  Call scanf to get the values from the user |
|  | mov -0x8(%rbp),%edx  mov -0xc(%rbp),%eax  mov %edx,%esi  mov %eax,%edi  call0x40062d <AddNumbers> | Store the fetched values from the users  Call AddNumbers function |
|  | mov %eax,-0x4(%rbp)  mov -0x8(%rbp),%edx  mov -0xc(%rbp),%eax  mov -0x4(%rbp),%ecx  mov %eax,%esi  mov $0x40074d,%edi  mov $0x0,%eax  call0x4004f0 <printf@plt> | Call printf using “%d + %d = %d” string and printout the returned value from AddNumbers |
|  | mov $0x0,%edi  call0x400530 <exit@plt> | Exit the procedure |
| int AddNumbers(int a, int b) | push%rbp  mov %rsp,%rbp | Reserve the Stack Pointer value. |
|  | mov %edi,-0x4(%rbp)  mov %esi,-0x8(%rbp)  mov -0x8(%rbp),%eax  mov -0x4(%rbp),%edx  add %edx,%eax | Get the values from the parameters and Sum them and return the value |
|  | pop %rbp  ret | Reset the Stack pointer to the previous value before executing the proc. |

## File Four: assignment3\_4.o

| **Functions** | **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- | --- |
| int main() | push %rbp  mov %rsp,%rbp  sub $0x10,%rsp | Reserve the Stack Pointer value.  Reserve 16 bytes in the stack for two variables |
|  | movl $0x0,-0x8(%rbp)  jmp 0x4007a0 <main+137>  mov $0x0,%eax  call 0x4006df <DisplayMenu> | Set var1 = 0  Create a loop and call DisplayMenu function to show the options, if we check 0x4007a0+137 it will result to the closing bracket for the loop |
|  | mov $0x400886,%edi  call 0x4004e0 <puts@plt>  lea -0x8(%rbp),%rax  mov %rax,%rsi  mov $0x400899,%edi  mov $0x0,%eax  call 0x400520 <\_\_isoc99\_scanf@plt>  mov -0x8(%rbp),%eax | Print out “Enter a number” using cout, use scanf to fetch the value from the user into var1 and store the value from memory to EAX |
|  | cmp $0x3,%eax  je 0x40077a <main+99>  mov $0x40089c,%edi  call 0x4004e0 <puts@plt>  lea -0x4(%rbp),%rax  mov %rax,%rsi  mov $0x400899,%edi  mov $0x0,%eax  call 0x400520 <\_\_isoc99\_scanf@plt>  mov -0x8(%rbp),%eax | Compare and Check EAX value.  If it’s 3 Jump to exit  Otherwise  Print out “Enter a number” and now the number will be stored in var2 via using scanf |
|  | cmp $0x1,%eax  jne 0x40078e <main+119>  mov -0x4(%rbp),%eax  mov %eax,%edi  call 0x40062d <PrintFact>  jmp 0x4007a0 <main+137>  mov -0x8(%rbp),%eax | Check if var1 is 1 then call PrintFact, otherwise jump to end of the loop |
|  | cmp $0x2,%eax  jne 0x4007a0 <main+137>  mov -0x4(%rbp),%eax  mov %eax,%edi  call 0x400688 <PrintSum>  mov -0x8(%rbp),%eax | Check if var1 is 2 then call PrintSum function, otherwise repeat the loop |
|  | cmp $0x3,%eax  jne 0x400728 <main+17>  mov $0x0,%edi  call 0x400530 <exit@plt> |  |
|  |  |  |
| DisplayMenu() | push %rbp  mov %rsp,%rbp | Reserve Stack Pointer before executing the proc |
| mov $0x400851,%edi  call 0x4004e0 <puts@plt> | Print out “1. Factorial” |
| mov $0x400864,%edi  call 0x4004e0 <puts@plt> | Print out “2. Summation” |
| mov $0x400871,%edi  call 0x4004e0 <puts@plt> | Print out “3. Quit” |
| mov $0x40087e,%edi  call 0x4004e0 <puts@plt>  mov $0x400851,%edi  call 0x4004e0 <puts@plt>  pop %rbp  ret | Print out “Enter your number:” |
|  |  | Print the factorial of n |
| int PrintFact(int n) | push %rbp  mov %rsp,%rbp  sub $0x20,%rsp | Reserve Stack Pointer, Reserve 32 bytes (4 variables) |
| mov %edi,-0x14(%rbp)  movl $0x1,-0x4(%rbp)  mov -0x14(%rbp),%eax  mov %eax,-0x8(%rbp) | Set variable a = 1  Set variable b = n |
| jmp 0x400669 <PrintFact+60>  mov -0x4(%rbp),%eax  imul -0x8(%rbp),%eax  mov %eax,-0x4(%rbp)  mov -0x8(%rbp),%eax  mov %eax,%esi  mov $0x400844,%edi  mov $0x0,%eax  call 0x4004f0 <printf@plt>  subl $0x1,-0x8(%rbp)  cmpl $0x0,-0x8(%rbp)  jg 0x400647 <PrintFact+26> | Start a loop and multiply  a = a \* b  Call print out using printf and string literals %d and print out the b value  Then decrement b value by 1, check if b greater than zero  Continue the loop until b value is zero |
| mov -0x4(%rbp),%eax  mov %eax,%esi  mov $0x400848,%edi  mov $0x0,%eax  call 0x4004f0 <printf@plt>  mov -0x4(%rbp),%eax | Print out the factorial value in a variable and return the value |
| Leave  ret | End the proc |
| int PrintSum(int n) | push %rbp  mov %rsp,%rbp  sub $0x20,%rsp | Reserve Stack Pointer, Reserve 32 bytes (4 variables/spaces) |
|  | mov %edi,-0x14(%rbp)  movl $0x0,-0x4(%rbp)  mov -0x14(%rbp),%eax  mov %eax,-0x8(%rbp) | Set variable1 = Given parameter X  Set variable2 = 0 |
|  | jmp 0x4006c0 <PrintSum+56>  mov -0x8(%rbp),%eax  add %eax,-0x4(%rbp)  mov -0x8(%rbp),%eax  mov %eax,%esi  mov $0x400844,%edi  mov $0x0,%eax  call 0x4004f0 <printf@plt>  subl $0x1,-0x8(%rbp)  cmpl $0x0,-0x8(%rbp)  jg 0x4006a2 <PrintSum+26> | While variable1 is greater than zero  Add variable1 to variable2 and set the value in variable2  Call printf to print out the value of variable1 , decrement variable 1, keep going until variable1 is zero |
|  | mov -0x4(%rbp),%eax  mov %eax,%esi  mov $0x400848,%edi  mov $0x0,%eax  call 0x4004f0 <printf@plt> | Move the result to EAX  Print out the sum value (variabe2) |
|  | mov -0x4(%rbp),%eax  leave  ret | Return value in EAX and exit the proc |