# **CS 410 Binary to Assembly Activity**

**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 > assignment3\_1.asm

| **Functions** | **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- | --- |
| main | 40057d: push %rbp  40057e: mov %rsp,%rbp  400581: mov $0x400634,%edi  400586: call 400450 <puts@plt>  40058b: mov $0x400648,%edi  400590: call 400450 <puts@plt>  400595: mov $0x40065c,%edi  40059a: call 400450 <puts@plt>  40059f: mov $0x0,%edi  4005a4: call 400480 <exit@plt> | This function displays three messages on the screen using the puts function. Each mov loads a memory address (a string) into a register and then call sends it to puts. After printing all three messages, it calls the exit function to end the program. |
| \_start | 400490: xor %ebp,%ebp  400492: mov %rdx,%r9  400495: pop %rsi  400496: mov %rsp,%rdx  400499: and $0xfffffffffffffff0,%rsp  40049d: push %rax  40049e: push %rsp  40049f: mov $0x400620,%r8  4004a6: mov $0x4005b0,%rcx  4004ad: mov $0x40057d,%rdi  4004b4: call 400460 <\_\_libc\_start\_main@plt> | This is the first function the program runs. It sets up the system to call main later. It adjusts the stack (where temporary data is stored), saves some values, and then calls \_\_libc\_start\_main, which handles calling the main function properly. |
| puts@plt | 400450: jmpq \*0x200bc2(%rip)  400456: pushq $0x0  40045b: jmpq 400440 <.plt> | Function placeholder for dynamic linking |
| exit@plt | 400480: jmpq \*0x200baa(%rip)  400486: pushq $0x3  40048b: jmpq 400440 <.plt> | Handles exit |
| \_init | 400418: sub $0x8,%rsp  40041c: mov 0x200bd5(%rip),%rax  400423: test %rax,%rax  400426: je 40042d <\_init+0x15>  400428: call 400470 <\_\_gmon\_start\_\_@plt>  40042d: add $0x8,%rsp  400431: retq | This function checks if a special profiling feature is available. If so, it calls it. This is mostly used by tools for measuring performance. Otherwise, it just returns and doesn’t do anything noticeable. |
| \_\_libc\_start\_main@plt | 400460: jmpq \*0x200bba(%rip)  400466: pushq $0x1  40046b: jmpq 400440 <.plt> | Start point of execution after \_start |

## File Two: assignment3\_2.o > assignment3\_2.asm

| **Functions** | **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- | --- |
| main | 40062d: push %rbp  40062e: mov %rsp,%rbp  400631: sub $0x20,%rsp  400635: mov %fs:0x28,%rax  40063e: mov %rax,-0x8(%rbp)  400642: xor %eax,%eax  400644: mov $0x400714,%edi  400649: callq 4004e0 <puts@plt>  40064e: lea -0x20(%rbp),%rax  400652: mov %rax,%rsi  400655: mov $0x40072b,%edi  40065a: mov $0x0,%eax  40065f: callq 400520 <\_\_isoc99\_scanf@plt>  400664: lea -0x20(%rbp),%rax  400668: mov %rax,%rsi  40066b: mov $0x40072e,%edi  400670: mov $0x0,%eax  400675: callq 4004f0 <printf@plt>  40067a: mov $0x0,%edi  40067f: callq 400530 <exit@plt> | This is the main function executed when the program runs. It first prints a message using puts(), then uses scanf() to get user input and stores it in a local variable, and finally prints a message using printf() to display what was entered. After that, it calls exit() to end the program. |
| \_start | 400540: xor %ebp,%ebp  400542: mov %rdx,%r9  400545: pop %rsi  400546: mov %rsp,%rdx  400549: and $0xfffffffffffffff0,%rsp  40054d: push %rax  40054e: push %rsp  40054f: mov $0x400700,%r8  400556: mov $0x400690,%rcx  40055d: mov $0x40062d,%rdi  400564: callq 400500 <\_\_libc\_start\_main@plt> | The actual entry point of the program before main() is called. It sets up the necessary register values and then calls \_\_libc\_start\_main() with main() as one of the arguments. |

## File Three: assignment3\_3.o > assignment3\_3.asm

| **Functions** | **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- | --- |
| \_start | 400540: xor %ebp,%ebp  400542: mov %rdx,%r9  400545: pop %rsi  400546: mov %rsp,%rdx  400549: and $0xfffffffffffffff0,%rsp  40054d: push %rax  40054e: push %rsp  40054f: mov $0x400720,%r8  400556: mov $0x4006b0,%rcx  40055d: mov $0x400641,%rdi  400564: callq 400500 <\_\_libc\_start\_main@plt>  400569: hlt  40056a: nopw 0x0(%rax,%rax,1) | This is the very first thing that runs when the program begins. It sets up the environment and prepares everything the system needs before running the actual main function. It passes control to a system function called \_\_libc\_start\_main, which then calls main. Even though you don’t see \_start in most C programs, it is automatically included and is essential for getting the program started the right way. |
| AddNumbers | 40062d: push %rbp  40062e: mov %rsp,%rbp  400631: mov %edi,-0x4(%rbp)  400634: mov %esi,-0x8(%rbp)  400637: mov -0x8(%rbp),%eax  40063a: mov -0x4(%rbp),%edx  40063d: add %edx,%eax  40063f: pop %rbp  400640: retq | This is a custom helper function made to do one thing—add two numbers together. When the function is called, it takes two input values, stores them temporarily, adds them, and returns the result. This function makes the code cleaner by separating the math part from the rest of the logic in main. |
| main | 400641: push %rbp  400642: mov %rsp,%rbp  400645: sub $0x10,%rsp  400649: mov $0x400734,%edi  40064e: callq 4004e0 <puts@plt>  400653: lea -0x8(%rbp),%rdx  400657: lea -0xc(%rbp),%rax  40065b: mov %rax,%rsi  40065e: mov $0x400747,%edi  400663: mov $0x0,%eax  400668: callq 400520 <\_\_isoc99\_scanf@plt>  40066d: mov -0x8(%rbp),%edx  400670: mov -0xc(%rbp),%eax  400673: mov %edx,%esi  400675: mov %eax,%edi  400677: callq 40062d <AddNumbers>  40067c: mov %eax,-0x4(%rbp)  40067f: mov -0x8(%rbp),%edx  400682: mov -0xc(%rbp),%eax  400685: mov -0x4(%rbp),%ecx  400688: mov %eax,%esi  40068a: mov $0x40074d,%edi  40068f: mov $0x0,%eax  400694: callq 4004f0 <printf@plt>  400699: mov $0x0,%edi  40069e: callq 400530 <exit@plt>  4006a3: nopw %cs:0x0(%rax,%rax,1)  4006ad: nopl (%rax) | This is where the main logic of the program happens. It starts by printing a message to the screen to let the user know what to do. Then it uses scanf to collect two numbers that the user types in. These numbers are stored in memory. Next, the function calls another function named AddNumbers and passes the two numbers to it. After getting the result back, main prints out the numbers and their sum using printf. Finally, it ends the program by calling the exit function to close everything properly. |

## File Four: assignment3\_4.o > assignment3\_4.asm

| **Functions** | **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- | --- |
| \_start | 400540: xor %ebp,%ebp  400542: mov %rdx,%r9  400545: pop %rsi  400546: mov %rsp,%rdx  400549: and $0xfffffffffffffff0,%rsp  40054d: push %rax  40054e: push %rsp  40054f: mov $0x400830,%r8  400556: mov $0x4007c0,%rcx  40055d: mov $0x400717,%rdi  400564: callq 400500 <\_\_libc\_start\_main@plt>  400569: hlt  40056a: nopw 0x0(%rax,%rax,1) | The \_start function is automatically included and runs before main. It prepares the memory and stack, sets up function pointers, and then calls \_\_libc\_start\_main, passing it the main function. This is required for every program but usually hidden from the programmer. Its job is to connect the operating system to the program’s actual logic. |
| main | 400717: push %rbp  400718: mov %rsp,%rbp  40071b: sub $0x10,%rsp  40071f: movl $0x0,-0x8(%rbp)  400726: jmp 4007a0 <main+0x89>  400728: mov $0x0,%eax  40072d: callq 4006df <DisplayMenu>  400732: mov $0x400886,%edi  400737: callq 4004e0 <puts@plt>  40073c: lea -0x8(%rbp),%rax  400740: mov %rax,%rsi  400743: mov $0x400899,%edi  400748: mov $0x0,%eax  40074d:callq 400520 <\_\_isoc99\_scanf@plt>  400752: mov -0x8(%rbp),%eax  400755: cmp $0x3,%eax  400758: je 40077a <main+0x63>  40075a: mov $0x40089c,%edi  40075f: callq 4004e0 <puts@plt>  400764: lea -0x4(%rbp),%rax  400768: mov %rax,%rsi  40076b: mov $0x400899,%edi  400770: mov $0x0,%eax  400775:callq 400520 <\_\_isoc99\_scanf@plt>  40077a: mov -0x8(%rbp),%eax  40077d: cmp $0x1,%eax  400780: jne 40078e <main+0x77>  400782: mov -0x4(%rbp),%eax  400785: mov %eax,%edi  400787: callq 40062d <PrintFact>  40078c: jmp 4007a0 <main+0x89>  40078e: mov -0x8(%rbp),%eax  400791: cmp $0x2,%eax  400794: jne 4007a0 <main+0x89>  400796: mov -0x4(%rbp),%eax  400799: mov %eax,%edi  40079b: callq 400688 <PrintSum>  4007a0: mov -0x8(%rbp),%eax  4007a3: cmp $0x3,%eax  4007a6: jne 400728 <main+0x11>  4007a8: mov $0x0,%edi  4007ad: callq 400530 <exit@plt>  4007b2: nopw %cs:0x0(%rax,%rax,1)  4007bc: nopl 0x0(%rax) | This is the core loop of this program. It begins by setting up a variable to store the user's menu choice. It then enters a loop that continues until the user chooses option 3 to exit. At the start of each loop, the program calls DisplayMenu to show a list of options. Then it prompts the user to enter a number for their choice. If the choice is 1 or 2, the program also asks for a second number to process. If the user picks option 1, it calls PrintFact to compute and display the factorial of the number. If option 2 is selected, it calls PrintSum to add up all numbers from the input number down to 1 and display the total. The loop repeats until the user picks 3, which ends the program by calling exit. |
| PrintFact | 40062d: push %rbp  40062e: mov %rsp,%rbp  400631: sub $0x20,%rsp  400635: mov %edi,-0x14(%rbp)  400638: movl $0x1,-0x4(%rbp)  40063f: mov -0x14(%rbp),%eax  400642: mov %eax,-0x8(%rbp)  400645: jmp 400669 <PrintFact+0x3c>  400647: mov -0x4(%rbp),%eax  40064a: imul -0x8(%rbp),%eax  40064e: mov %eax,-0x4(%rbp)  400651: mov -0x8(%rbp),%eax  400654: mov %eax,%esi  400656: mov $0x400844,%edi  40065b: mov $0x0,%eax  400660: callq 4004f0 <printf@plt>  400665: subl $0x1,-0x8(%rbp)  400669: cmpl $0x0,-0x8(%rbp)  40066d: jg 400647 <PrintFact+0x1a>  40066f: mov -0x4(%rbp),%eax  400672: mov %eax,%esi  400674: mov $0x400848,%edi  400679: mov $0x0,%eax  40067e: callq 4004f0 <printf@plt>  400683: mov -0x4(%rbp),%eax  400686: leaveq  400687: retq | This function calculates the factorial of the number given to it. A factorial is the product of a number and every number below it (e.g., 5! = 5×4×3×2×1). It uses a loop that multiplies numbers one by one, showing each step with printf, then prints the final result. The function is helpful for students to see how the calculation works step-by-step. |
| PrintSum | 400688: push %rbp  400689: mov %rsp,%rbp  40068c: sub $0x20,%rsp  400690: mov %edi,-0x14(%rbp)  400693: movl $0x0,-0x4(%rbp)  40069a: mov -0x14(%rbp),%eax  40069d: mov %eax,-0x8(%rbp)  4006a0: jmp 4006c0 <PrintSum+0x38>  4006a2: mov -0x8(%rbp),%eax  4006a5: add %eax,-0x4(%rbp)  4006a8: mov -0x8(%rbp),%eax  4006ab: mov %eax,%esi  4006ad: mov $0x400844,%edi  4006b2: mov $0x0,%eax  4006b7: callq 4004f0 <printf@plt>  4006bc: subl $0x1,-0x8(%rbp)  4006c0: cmpl $0x0,-0x8(%rbp)  4006c4: jg 4006a2 <PrintSum+0x1a>  4006c6: mov -0x4(%rbp),%eax  4006c9: mov %eax,%esi  4006cb: mov $0x400848,%edi  4006d0: mov $0x0,%eax  4006d5: callq 4004f0 <printf@plt>  4006da: mov -0x4(%rbp),%eax  4006dd: leaveq  4006de: retq | This function adds up all the numbers from the input number down to 1. For example, if the user enters 4, it calculates 4 + 3 + 2 + 1. Like the factorial function, it prints out each number as it adds them. After the loop, it shows the total sum. |
| DisplayMenu | 4006df: push %rbp  4006e0: mov %rsp,%rbp  4006e3: mov $0x400851,%edi  4006e8: callq 4004e0 <puts@plt>  4006ed: mov $0x400864,%edi  4006f2: callq 4004e0 <puts@plt>  4006f7: mov $0x400871,%edi  4006fc: callq 4004e0 <puts@plt>  400701: mov $0x40087e,%edi  400706: callq 4004e0 <puts@plt>  40070b: mov $0x400851,%edi  400710: callq 4004e0 <puts@plt>  400715: pop %rbp  400716: retq | This function simply prints several lines of text that make up the menu. The options displayed likely include "1. Factorial", "2. Sum", and "3. Exit". It uses multiple puts function calls to display the menu cleanly to the user. |

puts@plt, printf@plt, \_\_isoc99\_scanf@plt, exit@plt: These are placeholders for standard C library functions. They’re dynamically linked, so the actual addresses are resolved at runtime via the Procedure Linkage Table (PLT).

\_init, \_\_libc\_start\_main@plt, \_\_gmon\_start\_\_@plt: These are related to initialization before the program's main logic runs.

deregister\_tm\_clones, register\_tm\_clones, \_\_do\_global\_dtors\_aux, frame\_dummy, \_\_libc\_csu\_init, \_\_libc\_csu\_fini, \_fini: These are compiler-generated helper functions. They manage initialization and destruction of global/static objects, especially in C++ or multi-threaded programs.

Since these are compiler generated or support/shortcut functions I will not include these.