**3-2 Activity: Binary to Assembly**

Dylan Jeffery

SNHU

CS-410

Brian West

1/24/2022

# **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** |
| --- | --- | --- |
| \_init | 0x0000000000400418 <+0>:  sub $0x8,%rsp  0x000000000040041c <+4>:  mov 0x200bd5(%rip),%rax  # 0x600ff8  0x0000000000400423 <+11>:  test %rax,%rax  0x0000000000400426 <+14>:  je 0x40042d <\_init+21>  0x0000000000400428 <+16>: callq 0x400470 <\_\_gmon\_start\_\_@plt>  0x000000000040042d <+21>:  add $0x8,%rsp  0x0000000000400431 <+25>: retq | This section holds executable instructions that contribute to the process initialization code. That is, when a program starts to run the system arranges to execute the code in this section before the main program entry point |
| <puts@plt> | jmpq \*0x200bc2(%rip) # 601018 <puts@GLIBC\_2.2.5>  pushq $0x0  jmpq 400440 <.plt> | Library dependencies(like all the plt sections above)? |
| <\_start> | xor %ebp,%ebp  mov %rdx,%r9  pop %rsi  mov %rsp,%rdx  and $0xfffffffffffffff0,%rsp  push %rax  push %rsp  mov $0x400620,%r8  mov $0x4005b0,%rcx  mov $0x40057d,%rdi  callq 400460 <\_\_libc\_start\_main@plt>  hlt  nopw 0x0(%rax,%rax,1) | Make sure all links are initialized and then call main |
| <deregister\_tm\_clones> | mov $0x60104f,%eax  push %rbp  sub $0x601048,%rax  cmp $0xe,%rax  mov %rsp,%rbp  ja 4004d7 <deregister\_tm\_clones+0x17>  pop %rbp  retq  mov $0x0,%eax  test %rax,%rax  je 4004d5 <deregister\_tm\_clones+0x15>  pop %rbp  mov $0x601048,%edi  jmpq \*%rax  nopl 0x0(%rax) | Silly pointer arithmetic |
| <register\_tm\_clones> | mov $0x601048,%eax  push %rbp  sub $0x601048,%rax  sar $0x3,%rax  mov %rsp,%rbp  mov %rax,%rdx  shr $0x3f,%rdx  add %rdx,%rax  sar %rax  jne 400514 <register\_tm\_clones+0x24>  pop %rbp  retq  mov $0x0,%edx  test %rdx,%rdx  je 400512 <register\_tm\_clones+0x22>  pop %rbp  mov %rax,%rsi  mov $0x601048,%edi  jmpq \*%rdx  nopl 0x0(%rax) | Silly pointer arithmetic |
| <\_\_do\_global\_dtors\_aux> | cmpb $0x0,0x200b11(%rip) # 601048 <\_\_TMC\_END\_\_>  jne 40054a <\_\_do\_global\_dtors\_aux+0x1a>  push %rbp  mov %rsp,%rbp  callq 4004c0 <deregister\_tm\_clones>  pop %rbp  movb $0x1,0x200afe(%rip) # 601048 <\_\_TMC\_END\_\_>  repz retq  nopl 0x0(%rax) | Silly pointer arithmetic |
| <frame\_dummy>: | cmpq $0x0,0x2008c8(%rip) # 600e20 <\_\_JCR\_END\_\_>  je 400578 <frame\_dummy+0x28>  mov $0x0,%eax  test %rax,%rax  je 400578 <frame\_dummy+0x28>  push %rbp  mov $0x600e20,%edi  mov %rsp,%rbp  callq \*%rax  pop %rbp  jmpq 4004f0 <register\_tm\_clones>  nopl (%rax)  jmpq 4004f0 <register\_tm\_clones> | initialize exception handling frame and transaction memory |
| main | push %rbp  mov %rsp,%rbp  mov $0x400634,%edi  callq 400450 <puts@plt>  mov $0x400648,%edi  callq 400450 <puts@plt>  mov $0x40065c,%edi  callq 400450 <puts@plt>  mov $0x0,%edi  callq 400480 <exit@plt> | Sends string:  Ship to: John Smith.123 Los Angeles Rd Los Angeles, CA 90025.  This string is split into sections and passed through registers and calls (puts@plt sections). |
| <\_fini>: | sub $0x8,%rsp  add $0x8,%rsp  retq | This section holds executable instructions that contribute to the process termination code. That is, when a program exits normally, the system arranges to execute the code in this section. |

## File Two: assignment3\_2.o

| **Functions** | **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- | --- |
| main | push %rbp  mov %rsp,%rbp  sub $0x20,%rsp  mov %fs:0x28,%rax  mov %rax,-0x8(%rbp)  xor %eax,%eax  mov $0x400714,%edi  callq 4004e0 <puts@plt>  lea -0x20(%rbp),%rax  mov %rax,%rsi  mov $0x40072b,%edi  mov $0x0,%eax  callq 400520 <\_\_isoc99\_scanf@plt>  lea -0x20(%rbp),%rax  mov %rax,%rsi  mov $0x40072e,%edi  mov $0x0,%eax  callq 4004f0 <printf@plt>  mov $0x0,%edi  callq 400530 <exit@plt>  nopw %cs:0x0(%rax,%rax,1)  xchg %ax,%a | Prompts user for input with string  Please enter your name  Take user input and then return  Welcome Mr. %s |

## File Three: assignment3\_3.o

| **Functions** | **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- | --- |
| <AddNumbers> | push %rbp  mov %rsp,%rbp  mov %edi,-0x4(%rbp)  mov %esi,-0x8(%rbp)  mov -0x8(%rbp),%eax  mov -0x4(%rbp),%edx  add %edx,%eax  pop %rbp | Function for performing addition |
| main | push %rbp  mov %rsp,%rbp  sub $0x10,%rsp  mov $0x400734,%edi  callq 4004e0 <puts@plt>  lea -0x8(%rbp),%rdx  lea -0xc(%rbp),%rax  mov %rax,%rsi  mov $0x400747,%edi  mov $0x0,%eax  callq 400520 <\_\_isoc99\_scanf@plt>  mov -0x8(%rbp),%edx  mov -0xc(%rbp),%eax  mov %edx,%esi  mov %eax,%edi  callq 40062d <AddNumbers>  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  callq 4004f0 <printf@plt>  mov $0x0,%edi  callq 400530 <exit@plt>  nopw %cs:0x0(%rax,%rax,1) | Prompt the user to-  Enter two numbers:  %d %d  %d + %d = %d  Call the AddNumbers function, move the result into memory and exit main |

## File Four: assignment3\_4.o

| **Functions** | **Blocks of Assembly Code** | **Explanation of Functionality** |
| --- | --- | --- |
| PrintFact | push %rbp  mov %rsp,%rbp  sub $0x20,%rsp  mov %edi,-0x14(%rbp)  movl $0x1,-0x4(%rbp)  mov -0x14(%rbp),%eax  mov %eax,-0x8(%rbp)  jmp 400669 <PrintFact+0x3c>  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  callq 4004f0 <printf@plt>  subl $0x1,-0x8(%rbp)  cmpl $0x0,-0x8(%rbp)  jg 400647 <PrintFact+0x1a>  mov -0x4(%rbp),%eax  mov %eax,%esi  mov $0x400848,%edi  mov $0x0,%eax  callq 4004f0 <printf@plt>  mov -0x4(%rbp),%eax | Function to handle calculating and printing factorial of user provided number. |
| PrintSum | push %rbp  mov %rsp,%rbp  sub $0x20,%rsp  mov %edi,-0x14(%rbp)  movl $0x0,-0x4(%rbp)  mov -0x14(%rbp),%eax  mov %eax,-0x8(%rbp)  jmp 4006c0 <PrintSum+0x38>  mov -0x8(%rbp),%eax  add %eax,-0x4(%rbp)  mov -0x8(%rbp),%eax  mov %eax,%esi  mov $0x400844,%edi  mov $0x0,%eax  callq 4004f0 <printf@plt>  subl $0x1,-0x8(%rbp)  cmpl $0x0,-0x8(%rbp)  jg 4006a2 <PrintSum+0x1a>  mov -0x4(%rbp),%eax  mov %eax,%esi  mov $0x400848,%edi  mov $0x0,%eax  callq 4004f0 <printf@plt>  mov -0x4(%rbp),%eax | Handles summation function and printing of user provided numbers |
| DisplayMenu | push %rbp  mov %rsp,%rbp  mov $0x400851,%edi  callq 4004e0 <puts@plt>  mov $0x400864,%edi  callq 4004e0 <puts@plt>  mov $0x400871,%edi  callq 4004e0 <puts@plt>  mov $0x40087e,%edi  callq 4004e0 <puts@plt>  mov $0x400851,%edi  callq 4004e0 <puts@plt>  pop %rbp | Displays the menu  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1. Factorial  2. Summation  3. Quit |
| main | push %rbp  mov %rsp,%rbp  sub $0x10,%rsp  movl $0x0,-0x8(%rbp)  jmp 4007a0 <main+0x89>  mov $0x0,%eax  callq 4006df <DisplayMenu>  mov $0x400886,%edi  callq 4004e0 <puts@plt>  lea -0x8(%rbp),%rax  mov %rax,%rsi  mov $0x400899,%edi  mov $0x0,%eax  callq 400520 <\_\_isoc99\_scanf@plt>  mov -0x8(%rbp),%eax  cmp $0x3,%eax  je 40077a <main+0x63>  mov $0x40089c,%edi  callq 4004e0 <puts@plt>  lea -0x4(%rbp),%rax  mov %rax,%rsi  mov $0x400899,%edi  mov $0x0,%eax  callq 400520 <\_\_isoc99\_scanf@plt>  mov -0x8(%rbp),%eax  cmp $0x1,%eax  jne 40078e <main+0x77>  mov -0x4(%rbp),%eax  mov %eax,%edi  callq 40062d <PrintFact>  jmp 4007a0 <main+0x89>  mov -0x8(%rbp),%eax  cmp $0x2,%eax  jne 4007a0 <main+0x89>  mov -0x4(%rbp),%eax  mov %eax,%edi  callq 400688 <PrintSum>  mov -0x8(%rbp),%eax  cmp $0x3,%eax  jne 400728 <main+0x11>  mov $0x0,%edi  callq 400530 <exit@plt>  nopw %cs:0x0(%rax,%rax,1)  nopl 0x0(%rax) | Upon entry into main function user is prompted with a menu.  User input is then taken and saved, and registers are cleared before more user input is taken.  Depending on menu choice users then enter a value(s) of which the summation or factorial is calculated and printed (this is done through calling the respective function for the operation).  Several compares and checks for inequality are performed to control program flow. |