Southern
New Hampshire
University

CS 410 Binary to C++ Activity

By Ivy Pokorny

File One

The binary file has been successfully converted to assembly code, which can be found in the file named *assignment4_1.s*. Additionally, the assembly code has been converted back to binary, and the resulting file is *assignment4_1new.o.*

Step 2: The functionality of the blocks of assembly code.

Function	Blocks of Assembly Code	Explanation of Functionality

main	push %rbp	Push %rbp onto stack
	mov %rsp,%rbp	Move %rsp into %rbp
	sub $0x10,%rsp	Subtract 16 from %rsp (allocate stack space
	movl $0x1,-0x8(%rbp)	Move the value 1 into var1 at -0x8(%rbp)
	cmpl $0x9,-0x8(%rbp)	Compare var1 with 9
	jg a3 <main+0xa3>	Jump to address a3 if var1 > 9
	movl $0x1,-0xc(%rbp)	Move the value 1 into var2 at -0xc(%rbp)
	cmpl $0x9,-0xc(%rbp)	Compare var2 with 9
	jg 9a <main+0x9a>	Jump to address 9a if var2 > 9
	mov -0x8(%rbp),%eax	Load var1 into %eax
	imul -0xc(%rbp),%eax	Multiply %eax by var2
	mov %eax,-0x4(%rbp)	Move the result into result variable at -0x4(%rbp)
	mov -0x8(%rbp),%eax	Load var1 into %eax
	mov %eax,%esi	Move %eax into %esi
	lea 0x0(%rip),%rdi	Load effective address into %rdi
	callq 41 <main+0x41>	Call function at address 41
	lea 0x0(%rip),%rsi	Load effective address into %rsi
	mov %rax,%rdi	Move %rax into %rdi
	callq 50 <main+0x50>	Call function at address 50
	mov %rax,%rdx	Move %rax into %rdx (store return value)
	mov -0xc(%rbp),%eax	Load var2 into %eax
	mov %eax,%esi	Move %eax into %esi
	mov %rdx,%rdi	Move %rdx into %rdi
	callq 60 <main+0x60>	Call function at address 60
	lea 0x0(%rip),%rsi	Load effective address into %rsi
	mov %rax,%rdi	Move %rax into %rdi
	callq 6f <main+0x6f>	Call function at address 6f
	mov %rax,%rdx	Move %rax into %rdx (store return value)
	mov -0x4(%rbp),%eax	Load result into %eax
	mov %eax,%esi	Move %eax into %esi
	mov %rdx,%rdi	Move %rdx into %rdi
	callq 7f <main+0x7f>	Call function at address 7f
	mov %rax,%rdx	Move %rax into %rdx (store return value)
	mov 0x0(%rip),%rax	Load address from memory into %rax
	mov %rax,%rsi	
	mov %rdx,%rdi	
	callq 94 <main+0x94>	
	addl $0x1,-0xc(%rbp)	

Step 4: The assembly code converted to C++ code.

```
include<iostream>
using namespace std;

int main()
{
 int number, i, a, x;

 for (a = 1; a <= 9; a++)
   {
      for (i = 1; i <= 9; i++){
        x = a * i;
        cout << a << " * " << i << " = " << x << endl;
      }
   }
 return 0;
}
```

Step 5: Explain how the C++ code performs the same tasks as the blocks of assembly code.

Blocks of Assembly Code	C++ Code	Explanation of Functionality
push %rbp mov %rsp,%rbp sub $0x10,%rsp	int main(){	Push %rbp onto stack Move %rsp into %rbp Subtract 16 from %rsp (allocate stack space
movl $0x1,-0x8(%rbp)	int number, i, a, x;	Move the value 1 into var1 at -0x8(%rbp)
cmpl $0x9,-0x8(%rbp) jg a3 <main+0xa3>	for (a = 1; a <= 9; a++){	Compares a with 9 to check if the loop should continue. Jumps to the end if a is greater than 9, exiting the loop.
movl $0x1,-0x4(%rbp) cmpl $0x9,-0xc(%rbp) jg 9a <main+0x9a>	for (i = 1; i <= 9; i++){	Initializes variable i to 1 (stored at -0xc(%rbp)) Compares i with 9 to check if the inner loop should continue. Jumps to the end if i is greater than 9, exiting the inner loop
mov -0x8(%rbp),%eax imul -0xc(%rbp),%eax mov %eax,-0x4(%rbp)	x = a * i;	Loads the value of a into the register %eax. Multiplies a in %eax by i, storing the result back in %eax Stores the multiplication result in x (at -0x4(%rbp)).
addl $0x1,-0xc(%rbp) jmp 20 <main+0x20> addl $0x1,-0x8(%rbp) jmpq f <main+0xf> mov $0x0,%eax	return 0;	Increments i for the next iteration of the inner loop. Jumps back to reevaluate the inner loop condition. Increments a for the next iteration of the outer loop Jumps back to reevaluate the outer loop condition. Prepares to return 0, indicating successful execution.

leaveq retq		Cleans up the stack frame. Returns from the main function.

Blocks of Assembly Code	C++ Code	Explanation of Functionality

File Two

The binary file has been successfully converted to assembly code, which can be found in the file named *assignment4_2.s*. Additionally, the assembly code has been converted back to binary, and the resulting file is *assignment4_2new.o.*

Step 2: Explain the functionality of the blocks of assembly code.

Function	Blocks of Assembly Code	Explanation of Functionality

Main	push %rbp	Push %rbp onto stack
	mov %rsp,%rbp	Move %rsp into %rbp
	sub $0x30,%rsp	Subtract 48 from %rsp
	mov %fs:0x28,%rax	Load the value at %fs:0x28 into %rax
	mov %rax,-0x8(%rbp)	Move %rax into -0x8(%rbp) (store TLS value)
	xor %eax,%eax	XOR %eax, %eax
	lea 0x0(%rip),%rsi	LEA the address of the next instruction into %rsi
	lea 0x0(%rip),%rdi	LEA the address of the next instruction into %rdi
	callq 2a <main+0x2a>	Call function at address 2a
	mov %rax,%rdx	Move %rax into %rdx
	mov 0x0(%rip),%rax	Move the value at the address into %rax
	mov %rax,%rsi	Move %rax into %rsi
	mov %rdx,%rdi	Move %rdx into %rdi
	callq 3f <main+0x3f>	Call function at address 3f
	lea -0x14(%rbp),%rax	LEA -0x14(%rbp) into %rax (address of local variable)
	mov %rax,%rsi	Move %rax into %rsi
	lea 0x0(%rip),%rdi	LEA another address into %rdi
	callq 52 <main+0x52>	Call function at address 52
	mov -0x14(%rbp),%edx	Load local variable into %edx
	mov -0x14(%rbp),%eax	Load local variable into %eax
	imul %eax,%edx	IMUL %eax, %edx
	mov -0x14(%rbp),%eax	Load local variable into %eax
	imul %edx,%eax	IMUL %edx, %eax
	mov %eax,-0x14(%rbp)	Move %eax into the local variable
	mov -0x14(%rbp),%eax	Load local variable into %eax
	cvtsi2sd %eax,%xmm0	CVTSI2SD %eax into %xmm0
	movsd 0x0(%rip),%xmm1	Load the value at address into %xmm1
	mulsd %xmm1,%xmm0	MULSD %xmm1, %xmm0
	movsd %xmm0,-0x10(%rbp)	Move %xmm0 into the local variable
	lea 0x0(%rip),%rsi	LEA another address into %rsi
	lea 0x0(%rip),%rdi	LEA another address into %rdi
	callq 8f <main+0x8f>	Call function at address 8f
	mov %rax,%rdx	Move %rax into %rdx
	mov -0x10(%rbp),%rax	Load local variable into %rax
	mov %rax,-0x28(%rbp)	Move %rax into another local variable
	movsd -0x28(%rbp),%xmm0	Load the double value into %xmm0
	mov %rdx,%rdi	Move %rdx into %rdi
	callq a7 <main+0xa7>	Call function at address a7
	mov $0x0,%eax	Move 0 into %eax (return value)

Step 4: Convert the assembly code to C++ code.

```
#include <iostream>
#include <cmath>

using namespace std;

void function1();
void function2();
void function3();

int main() {
    // Set up local variables
    double result;
    int value = 1; // Assuming initial value similar to a stack variable
    int temp;

    // Call function1
    function1();

    // Call function2
    function2();

    temp = value * value; // Value squared
    temp *= value; // Multiply by value again
    result = static_cast<double>(temp) * 1.0; // Convert to double and multiply by 1.0

       double finalResult = result;

    // Call function3 with the final result
    function3();

    // Output the result
    cout << "Final result: " << finalResult << endl;

    return 0; // Return success
}

// Function implementations (placeholders)
void function1() {
    // Function code unknown
}

void function2() {
    // Function code unknown
}

void function3() {
    // Function code unknown
}
```

Step 5: Explain how the C++ code performs the same tasks as the blocks of assembly code.

Blocks of Assembly Code	C++ Code	Explanation of Functionality

push %rbp
mov %rsp,%rbp
subq $48, %rsp
movl $0x1,-0x4(%rbp)
	int main() {
	double result;
	int value = 1;
	int temp;
		Push %rbp onto stack
		Move %rsp into %rbp
		Allocates space on the stack for local variables (like result and value

call _Z9function1v
call _Z9function2v
	function1();
	function2();
		Calls the first function. Both lines execute the same function.

movl -24(%rbp), %eax
imull -24(%rbp), %eax
movl %eax, -20(%rbp)
	temp = value * value;
	temp *= value;
		Loads value into temp for calculations.
		Multiplies temp by value. The result is stored back in temp.

cvtsi2sd -20(%rbp), %xmm0	result = static_cast<double>(temp) * 1.0;	Converts temp from an integer to a double and stores it in result.
movsd %xmm0, -16(%rbp)	double finalResult = result;	Stores the double value into a local variable for output.
call _Z9function3v	function3();	Calls the third function
leaq .LC0(%rip), %rsi call _ZStlsISt11char_traitsIcEERSt13basic_ostream...	cout << "Final result: " << finalResult << endl;	Prepares the string for output to the console. Outputs the final result using cout.
movl $0, %eax leave ret	return 0;	Cleans up the stack and returns 0 from main, indicating successful execution

File Three

The binary file has been successfully converted to assembly code, which can be found in the file named *assignment4_3.s.* Additionally, the assembly code has been converted back to binary, and the resulting file is *assignment4_3new.o.*

Step 2: Explain the functionality of the blocks of assembly code.

Function	Blocks of Assembly Code	Explanation of Functionality

Main	push %rbp	Push %rbp onto the stack
	mov %rsp,%rbp	Move %rsp into %rbp
	sub $0x20,%rsp	Subtract 32 from %rsp (allocate space on the stack)
	mov %fs:0x28,%rax	Move value from FS segment at offset 0x28 into %rax
	mov %rax,-0x8(%rbp)	Store %rax at -8(%rbp) (save for error handling)
	xor %eax,%eax	Set %eax to 0 (initialize)
	movl $0x1,-0xc(%rbp)	Move 1 into -12(%rbp) (initialize a variable)
	lea 0x0(%rip),%rsi	Load address of the next instruction into %rsi
	lea 0x0(%rip),%rdi	Load address of the next instruction into %rdi
	callq 31 <main+0x31>	Call function at address 31
	mov %rax,%rdx	Move return value from %rax into %rdx
	mov 0x0(%rip),%rax	Load value from memory into %rax
	mov %rax,%rsi	Move %rax into %rsi
	mov %rdx,%rdi	Move %rdx into %rdi
	callq 46 <main+0x46>	Call function at address 46
	lea -0x18(%rbp),%rax	Load address of local variable at -24(%rbp) into %rax
	mov %rax,%rsi	Move %rax into %rsi
	lea 0x0(%rip),%rdi	Load address of the next instruction into %rdi
	callq 59 <main+0x59>	Call function at address 59
	mov -0x18(%rbp),%eax	Load value from -24(%rbp) into %eax
	sub $0x1,%eax	Subtract 1 from %eax
	mov %eax,-0xc(%rbp)	Move %eax into -12(%rbp)
	movl $0x1,-0x10(%rbp)	Move 1 into -16(%rbp) (initialize another variable)
	mov -0x18(%rbp),%eax	Load value from -24(%rbp) into %eax
	cmp %eax,-0x10(%rbp)	Compare %eax with -16(%rbp)
	jg e3 <main+0xe3>	If greater, jump to address e3
	movl $0x1,-0x14(%rbp)	Move 1 into -20(%rbp)
	mov -0x14(%rbp),%eax	Load value from -20(%rbp) into %eax
	cmp -0xc(%rbp),%eax	Compare %eax with -12(%rbp)
	jg 99 <main+0x99>	If greater, jump to address 99
	lea 0x0(%rip),%rsi	Load address of the next instruction into %rsi
	lea 0x0(%rip),%rdi	Load address of the next instruction into %rdi
	callq 93 <main+0x93>	Call function at address 93
	addl $0x1,-0x14(%rbp)	Add 1 to -20(%rbp)
	jmp 78 <main+0x78>	Jump back to address 78
	subl $0x1,-0xc(%rbp)	Subtract 1 from -12(%rbp)
	movl $0x1,-0x14(%rbp)	Move 1 into -20(%rbp)
	mov -0x10(%rbp),%eax	Load value from -16(%rbp) into %eax
	add %eax,%eax	Add %eax to itself

Step 4: Convert the assembly code to C++ code.

```
#include <iostream>

using namespace std;

// Function prototypes
void function1();
void function2();
void function3();

int main() {
    // Set up local variables
    int result = 1; // Initialized to 1
    int temp1 = 1;  // Represents the counter
    int temp2 = 1;  // Another temp variable

    // Call function1 and function2
    function1();
    function2();

    // Main logic
    while (true) {
        // Simulating the loop with temp1
        if (--temp1 <= 0) {
            break; // Exit if temp1 becomes 0
        }

        temp2 = 1; // Reset temp2
        while (temp2 <= temp1) {
            function3(); // Placeholder for function3 logic
            temp2++; // Increment temp2
        }

        result++; // Increment result
    }

    result -= 1; // Adjust result

    // Output the final result
    cout << "Final result: " << result << endl;

    return 0; // Indicate successful completion
}

void function1() {
    // Function code unknown
}

void function2() {
    // Function code unknown
}

void function3() {
    // Function code unknown
}
```

Step 5: Explain how the C++ code performs the same tasks as the blocks of assembly code.

Blocks of Assembly Code	C++ Code	Explanation of Functionality

pushq %rbp
movq %rsp, %rbp
subq $16, %rsp
movl $1, -12(%rbp)movl $1, -8(%rbp)
movl $1, -4(%rbp)
	int main() {
	int outerCounter = 1;
	int innerCounter = 1;
	int limit = 1;
		Push %rbp onto stack
		Move %rsp into %rbp
		Allocates space on the stack for local variables
		Initializes outerCounter, innerCounter, and limit to 1,

call _Z9function1v
call _Z9function2v
	function1();
	function2();
		Calls the first & second function.

.L6:	while (outerCounter <= limit) {	Marks the beginning of a loop that continues while outerCounter is less than or equal to limit
subl $1, -8(%rbp)	limit--;	Decreases limit by 1 at the start of each iteration of the outer loop.

.L5:	innerCounter = 1;
	while (innerCounter <= limit) {
		Resets innerCounter to 1 and starts the inner loop that runs while innerCounter is less than or equal to limit.

call _Z9function3v	function3();	Calls the third function in each iteration of the inner loop.

Blocks of Assembly Code	C++ Code	Explanation of Functionality
addl $1, -12(%rbp)	outerCounter++;	After the inner loop completes, increments outerCounter.
leaq .LC0(%rip), %rsi call _ZStlsISt11char_traitsIcEERSt13basic_ostream...	cout << "Final result: " << outerCounter << endl;	Prepares and outputs the final result using cout, mirroring the output functionality in assembly.
movl $0, %eaxleave ret	return 0;	Cleans up the stack and returns 0 from main, indicating successful execution

File Four

The binary file has been successfully converted to assembly code, which can be found in the file named *assignment4_4.s.* Additionally, the assembly code has been converted back to binary, and the resulting file is *assignment4_4new.o.*

Step 2: Explain the functionality of the blocks of assembly code.

Blocks of Assembly Code	Explanation of Functionality

push %rbp	Push %rbp onto the stack
mov %rsp,%rbp	Move %rsp into %rbp
sub $0x30,%rsp	Subtract 48 from %rsp (allocate space on the stack)
mov %fs:0x28,%rax	Move value from FS segment at offset 0x28 into %rax
mov %rax,-0x8(%rbp)	Store %rax at -8(%rbp) (save for error handling)
xor %eax,%eax	Set %eax to 0 (initialize)
movq $0x0,-0x20(%rbp)	Move 0 into -32(%rbp) (initialize a variable)
movq $0x1,-0x18(%rbp)	Move 1 into -24(%rbp) (initialize another variable)
lea 0x0(%rip),%rsi	Load address of the next instruction into %rsi
lea 0x0(%rip),%rdi	Load address of the next instruction into %rdi
callq 3a <main+0x3a>	Call function at address 3a
mov %rax,%rdx	Move return value from %rax into %rdx
mov 0x0(%rip),%rax	Load value from memory into %rax
mov %rax,%rsi	Move %rax into %rsi
mov %rdx,%rdi	Move %rdx into %rdi
callq 4f <main+0x4f>	Call function at address 4f
lea -0x28(%rbp),%rax	Load address of local variable at -40(%rbp) into %rax
mov %rax,%rsi	Move %rax into %rsi
lea 0x0(%rip),%rdi	Load address of the next instruction into %rdi
callq 62 <main+0x62>	Call function at address 62
mov -0x28(%rbp),%rax	Load value from -40(%rbp) into %rax
test %rax,%rax	Test %rax with itself (check if zero)
je f2 <main+0xf2>	If zero, jump to address f2
mov -0x28(%rbp),%rcx	Load value from -40(%rbp) into %rcx
movabs $0x6666666666666667,%rdx	Move absolute value 0x6666666666666667 into %rdx
mov %rcx,%rax	Move %rcx into %rax
imul %rdx	Multiply %rdx by %rax
sar $0x2,%rdx	Arithmetic shift right %rdx by 2
mov %rcx,%rax	Move %rcx into %rax
sar $0x3f,%rax	Arithmetic shift right %rax by 63
sub %rax,%rdx	Subtract %rax from %rdx
mov %rdx,%rax	Move %rdx into %rax
mov %rax,-0x10(%rbp)	Store %rax at -16(%rbp)
mov -0x10(%rbp),%rdx	Load value from -16(%rbp) into %rdx
mov %rdx,%rax	Move %rdx into %rax
shl $0x2,%rax	Shift left %rax by 2
add %rdx,%rax	Add %rdx to %rax
add %rax,%rax	Double %rax
sub %rax,%rcx	Subtract %rax from %rcx

Step 4: Convert the assembly code to C++ code.

```
#include <iostream>

using namespace std;

// Function prototypes
void function1();
void function2();
void function3();

int main() {
    long long temp1 = 0; // Variable initialized to 0
    long long temp2 = 1; // Variable initialized to 1
    long long result;

    // Call function1
    function1();

    // Call function2
    function2();

    // Main calculations
    while (true) {
        // Simulate the assembly logic
        if (temp2 == 0) {
            break; // Exit the loop if temp2 is 0
        }

        long long value = temp2 * 0x6666666666666667; // Multiply with the constant
        value >>= 2; // Right shift by 2
        result = (temp2 >> 63) - value;

        temp1 += result;
        temp2++; // Increment temp2
    }

    // Final output
    cout << "Final result: " << temp1 << endl;

    return 0; // Indicates successful completion
}

void function1() {
    // Function code unknown
}

void function2() {
    // Function code unknown
}

void function3() {
    // Function code unknown
}
```

Step 5: Explain how the C++ code performs the same tasks as the blocks of assembly code.

Blocks of Assembly Code	C++ Code	Explanation of Functionality

pushq %rbp
movq %rsp, %rbp
subq $32, %rsp
	int main() {
	long long temp1 = 0;
	long long temp2 = 1;
	long long result;
		Push %rbp onto stack
		Move %rsp into %rbp
		Allocates space on the stack for local variables (temp1 & temp2)

call _Z9function1v
call _Z9function2v
	function1();
	function2();
		Calls the first & second function.

Blocks of Assembly Code	C++ Code	Explanation of Functionality

movq -24(%rbp), %rdx
movabsq $7378697629483820647, %rax
imulq %rdx, %rax
movq %rax, -16(%rbp)
sarq $2, -16(%rbp)
movq -24(%rbp), %rax
sarq $63, %rax
subq -16(%rbp), %rax
movq %rax, -8(%rbp)
	long long value = temp2 * 7378697629483820647;
	value >>= 2;
	result = (temp2 >> 63) - value;
		The assembly multiplies temp2 by the constant 7378697629483820647 and stores the result in -16(%rbp).
		It then performs a right shift of the result by 2 bits (sarq $2).
		The sign adjustment is calculated by shifting temp2 right by 63 bits.
		Finally, the assembly subtracts the shifted result from the sign adjustment and stores it in -8(%rbp)

movq -8(%rbp), %rax addq %rax, -32(%rbp)	temp1 += result;	Adds the value stored at -8(%rbp) (the result of the calculations) to temp1 (stored at -32(%rbp))
addq $1, -24(%rbp) jmp .L4	temp2++;	Increments temp2 by 1 and jumps back to the start of the loop (.L4) for the next iteration.

.L7:
nop
	if (temp2 == 0) {
	break;
	}
		The assembly uses a je instruction to exit the loop when temp2 equals 0

leaq .LC0(%rip), %rsi leaq _ZSt4cout(%rip), %rdi call _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@PLT movq %rax, %rdx movq -32(%rbp), %rax movq %rax, %rsi movq %rdx, %rdi call _ZNSolsEx@PLT movq %rax, %rdx movq *ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6*@GOTPCREL(%rip), %rax movq %rax, %rsi movq %rdx, %rdi call _ZNSolsEPFRSoS_E@PLT	cout << "Final result: " << temp1 << endl;	Preparing arguments for the operator<< calls. Outputting the string "Final result: ". Outputting the value of temp1. Appending a newline and flushing the stream with std::endl.
movl $0, %eax leave ret	return 0;	Cleans up the stack and returns 0 from main, indicating successful execution