**CSCE 312 – Lab 5 Report**

**Texas A&M University**

**March 22, 2024**

**Poyi Ou­­­**

**Problem 1**

**Assembly Code for case a:**

**A screenshot of a computer program

Description automatically generated**  
**\*Note: Register of i → %rdi, Register of j → %rsi**

**Results:**

**Case a → i = 1, j = 2; Output: i = 7, j = 3**

**A screenshot of a computer code

Description automatically generated**

**Case b → i = 8, j = 7; Output: i = 5, j = 11**

**A screenshot of a computer code

Description automatically generated**

**Problem 2**

**Assembly Code assuming j = 3, k = 5:**

**A screenshot of a computer program

Description automatically generated**

**\*Note: Register of j → %r8, Register of k → %r9, , Register of i → %r10**

**Results:**

**Case 1: → j = 3, k = 5; Output: j = 20, k = 16**

**A screenshot of a computer code

Description automatically generated**

**Problem 3  
Assembly Code for 3.1: Assembly Code for 3.2:**

A screen shot of a computer code

Description automatically generated A computer screen shot of a program

Description automatically generated

**Compare and Analysis**

By comparing the lengths of the two different assembly codes, we can observe that problem 3.2 involves more steps than problem 3.1, resulting in the use of more memory in problem 3.2 compared to problem 3.1.

Both files have a **.LC0** section that stores strings. The **main** section contains all the computations. The **.LFB0** section marks the beginning of the computation, accessing registers and memory to change values. Sections .**LFE0, 0, 1, 2, 3, 4** store basic information about our compiler.

In the assembly code of problem 3.1, it simply sets up the **main** function within the **.LFB0** section and prints the "Hello, world" string.

In the assembly code of problem 3.2, additional memory is used for storing the value of "i", which is stored 4 bytes below the address of %rbp. The increment operation i++ is also implemented at the same memory address. Subsequently, the value is moved to the return register %eax.

**Problem 4**

**Assembly Code:**

A screen shot of a computer program

Description automatically generated**A screen shot of a computer code

Description automatically generated**

While this code segment actually achieves the same task as problem 3.1, it introduces a new function called print\_hello() to accomplish its goal.

Upon examining the assembly code, a new section called **print\_hello** is created, given that it is the function name specified in C++. Within this **print\_hello** section, **%rbp** is pushed onto the stack when the function is called. After printing "Hello, world" using the **puts** function, **%rbp** is then popped off the stack, and the function returns to the calling point, which is the **main** segment.

In both the assembly code for problem 3.1 and problem 4, the address of "Hello, world" is saved into respective registers, where it is stored in **%rax**. Overall, both assembly codes perform the same task by loading the address of the string into a register. In problem 3.1, it simply returns the value, while in problem 4, it returns to the main function, where it was originally located within the **print\_hello** block.

**Problem 5**

**Assembly Code for 5\_main: Assembly Code for 5\_print:**

A screen shot of a computer program

Description automatically generated A screen shot of a computer code

Description automatically generated

While there isn't much difference between both method of calling the print function, the only obvious distinction is that when the program calls a function that resides in another file, it adds **@PLT** to the call, where **@PLT** stands for "Procedure Linkage Table". In problem 4, the syntax is **call** **print\_hello**, while in problem 5, it's **call print\_hello@PLT.**

**Problem 6**

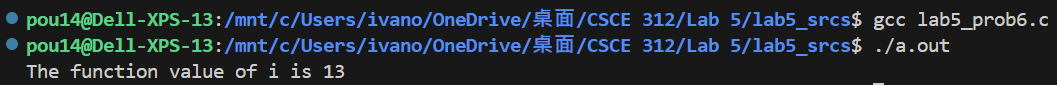
**Assembly code:**

**A computer screen shot of a program code

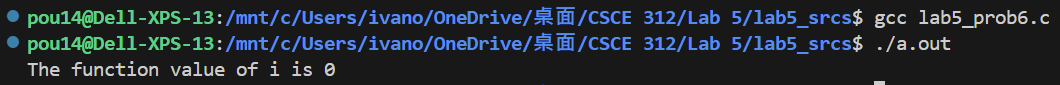
Description automatically generated**

\*Note: Using a longward (32-bit) to implement this function because variable type integer is in 32-bit.

**Output: i = 12**

****

**Output: i = 21**

****