Lab 5 Report

Julian Tiu (624005611)

April 9, 2017

Lab 5 Problem 1:

```
irmovl $2, %eax
rmmovl %ecx, 4(%esp)
irmovl $1, %ecx
rmmovl %ecx, 4(%esp)
irmovl $5, %edx
irmovl $1, %ebx
subl %eax, %ecx
jl greater
halt
greater:
    addl %edx, %eax
    mrmovl 4(%esp), %ecx
    halt
else:
    mrmovl 4(%esp), %ecx
    irmovl $0, %eax
    addl %ebx, %ecx
```

- The initial value for i is 2, and the final value is 7.
- The initial value for j is 1, and the final value is 1.

Lab 5 Problem 2:

```
irmovl $1, %eax #declare value for j
rmmovl %eax, 4(%esp) #store value of j
irmovl $1, %eax #declare value for k
rmmovl %eax, 8(%esp) #store value of k
irmovl $0, %eax #declare value for i
rmmovl %eax, 12(%esp) #store value of i
irmovl $1, %edx #declare value of incrementer
irmovl $4, %ebx #declare value of the limit for i
jmp Loop
Loop:
```

```
    mrmovl 12(%esp), %eax #retrieve the value of i
    addl %eax, %eax #i*2
    rmmovl %eax, 4(%esp) #stores this to j
    addl %edx, %eax #adds value of j and 1
    rmmovl %eax, 8(%esp) #store the above result to k
    mrmovl 12(%esp), %eax #retrieve value of i to be modified
    mrmovl 12(%esp), %edi #retrieve value of i
    addl %edx, %eax #new value of i
    rmmovl %eax, 12(%esp) #store this for new value of i
    subl %ebx, %edi #subtracts current value of i # and the limit
    jl Loop
mrmovl 4(%esp), %eax
mrmovl 8(%esp), %ecx
```

- The initial value for j is 1, and the final value is 8.
- The initial value for k is 1, and the final value is 9.
- The initial value of i is 0.

Lab 5 Problem 3:

Printout for lab5_prob3_1.c:

Hello, world

Explanation of the segments in the generated assembly code:

- The generated assembly code varies with the different optimization flags during the compilation command. The assembly code shown here is the generated assembly code without any optimization flags. Since this code is not optimized, it will contain unneccesary assembly statements.
- In "_main:", the register %rbp is pointing to the frame.
- In "Ltmp1:", the registers %rsp and %rbp point to the same location in the stack.
- In "Ltmp2:", the statements prior to "callq _printf" allocates the arguments for main in the stack, which are not used in the program. The program then calls _printf, which is not defined in the assembly file, and prints out "Hello, world." Since the program ends with a return 0 statement, this is done with the "xorl %ecx, %ecx" that sets %ecx to 0 and "retq" for returning.
- L_.str is where the string to be printed is contained.

Printout for lab5_prob3_2.c:

The value of i is 2

Explanation of the segments in the generated assembly code:

- The generated assembly code varies with the different optimization flags during the compilation command. The assembly code shown here is the generated assembly code without any optimization flags. Since this code is not optimized, it will contain unneccesary assembly statements.
- In "_main:", the register %rbp is pointing to the frame.
- In "Ltmp1:", the registers %rsp and %rbp point to the same location in the stack.
- In "Ltmp2:", the statements prior to "movl $1, -20(%rbp)" allocates the arguments for main in the stack, which are not used in the program. The program then sets i to 1 with the statement "movl $1, -20(%rbp), which will be pointed by the register %edi. i then gets incremented by 1 with the statement "addl $1, %edi" This incremented value is the one that will be printed when "_printf" is called; _ptintf is also not defined in here.
- L_.str is where the string to be printed is contained.

Lab 5 Problem 4:

Both programs were compiled with gcc, which generated the *.s files. Both codes materialized the same way within the computer — both examples only had one file to be compiled, which only produced one .s file each. With the separate .s files, they are each compiled to separate .o files, and after the linking process, two separate files are produced, namely lab5_prob3_1.exe and lab5_prob4_main.exe. The only difference between them are the actual segments within the assembly files. In lab5_prob4_main.s, there is an extra segment that defines the print_hello() function, namely _print_hello. lab5_prob3_1.s does not have this definition because there is only one function within the code which calls _printf directly rather than calling a separate function.

Lab 5 Problem 5:

Both programs were compiled with gcc, which generated the *.s files. The two examples materialized differently in the computer, because in problem 4 we see that it only has one file, for which contains two functions, main and print_hello(); however, in the example given in problem 5, there are two files given, lab5_prob5_main.c and lab5_prob5_print.c — one containing the main driver, and the other containing a print_hello() function that gets utilized in main that does the printing. This is even evident in the assembly code for lab5_prob5_main.s, because there is no function segment or definition for the print_hello() function when it is being called in the assembly code in lab5_prob5_main.s, which means that the definition is within a different file which needs to be linked together. When the code in problem 4 gets compiled, only one .o file gets produced and it only gets linked with the necessary libraries;

while with the codes in problem 4, the linker has to link the two .o files together along with the necessary libraries, and the compilation of two files produces one executable file.

Lab 5 Problem 6:

The answer to problem 6 is within the file, lab5_prob6.c