Problem 1.

i is in %rax and j is in %rbx

i = 10, j = 6

```
Stopped in 10 steps at PC = 0x4b.  Status 'HLT', CC Z=0 S=0 O=0
Changes to registers:
%rax:   0x0000000000000000      0x0000000000000001
%rcx:   0x0000000000000000      0x0000000000000002
%rdx:   0x0000000000000000      0x0000000000000001
%rbx:   0x0000000000000000      0x0000000000000007
```

i = 5, j = 6

```
Stopped in 10 steps at PC = 0x55.  Status 'HLT', CC Z=0 S=0 O=0
Changes to registers:
%rax:   0x0000000000000000      0x0000000000000007
%rcx:   0x0000000000000000      0x0000000000000002
%rdx:   0x0000000000000000      0x0000000000000001
%rbx:   0x0000000000000000      0x0000000000000006
```

i = 6, j = 6

```
Changes to registers:
%rax:   0x0000000000000000      0x0000000000000008
%rcx:   0x0000000000000000      0x0000000000000002
%rdx:   0x0000000000000000      0x0000000000000001
%rbx:   0x0000000000000000      0x0000000000000006
```

Problem 2.

j is in %rax and k is in %rbx

j = 5, k = 2

```
Stopped in 51 steps at PC = 0x63.  Status 'HLT', CC Z=1 S=0 O=
Changes to registers:
%rax:   0x0000000000000000      0x0000000000000008
%rdx:   0x0000000000000000      0x0000000000000005
%rbx:   0x0000000000000000      0x0000000000000009
%rsi:   0x0000000000000000      0x0000000000000001
%rdi:   0x0000000000000000      0x0000000000000002
```

j = 3, k = 2

```
Stopped in 51 steps at PC = 0x63.  Status 'HLT', CC Z=1 S=0 O=0
Changes to registers:
%rax:   0x0000000000000000      0x0000000000000008
%rdx:   0x0000000000000000      0x0000000000000005
%rbx:   0x0000000000000000      0x0000000000000009
%rsi:   0x0000000000000000      0x0000000000000001
%rdi:   0x0000000000000000      0x0000000000000002
```

j = 10, k = 20

```
Stopped in 51 steps at PC = 0x63.  Status 'HLT', CC Z=1 S=0 O=0
Changes to registers:
%rax:   0x0000000000000000      0x0000000000000008
%rdx:   0x0000000000000000      0x0000000000000005
%rbx:   0x0000000000000000      0x0000000000000009
%rsi:   0x0000000000000000      0x0000000000000001
%rdi:   0x0000000000000000      0x0000000000000002
```

Problem 3.

lab5_prob3_1.s

```
    .file   "lab5_prob3_1.c"
    .section    .rodata
.LC0:
    .string "Hello, world!"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $16, %rsp
    movl    %edi, -4(%rbp)
    movq    %rsi, -16(%rbp)
    movl    $.LC0, %edi
    call    puts
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (GNU) 4.8.5 20150623 (Red Hat 4.8.5-44)"
    .section    .note.GNU-stack,"",@progbits
```

prob3_2.s

```
    .file   "lab5_prob3_2.c"
    .section    .rodata
.LC0:
    .string "The value of i is %d\n"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $32, %rsp
    movl    %edi, -20(%rbp)
    movq    %rsi, -32(%rbp)
    movl    $1, -4(%rbp)
    addl    $1, -4(%rbp)
    movl    -4(%rbp), %eax
    movl    %eax, %esi
    movl    $.LC0, %edi
    movl    $0, %eax
    call    printf
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (GNU) 4.8.5 20150623 (Red Hat 4.8.5-44)"
    .section    .note.GNU-stack,"",@progbits
```

In each code segment, there are similarities and differences. One similarity is that in both codes, there is the .LC0 section which holds the strings that are output to the screen even if they are different. Also in both outputs, pushq, movq, and subq set up the stack frame for the function which allows each code to run in the stack. Then, movl moves the arguments passed in by the main function to their respective places on the stack frame which can be accessed later. In the second code, one is added to the i register and this is why there is a call to addl which adds one to the register with a specific memory address. The next instruction they both share is the call instruction which calls the puts function in the first program and printf function in the second program. The leave call is to ultimately end the program and return out. The difference between the two codes is that in one, a function is called to put “hello world” to the screen while in the other, a variable is incremented by one and printed to the screen. To do this they use a few different lines, but the main idea of both is still the same.

Problem 4.

prob4.s

```
    .file   "prob4.c"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $16, %rsp
    movl    %edi, -4(%rbp)
    movq    %rsi, -16(%rbp)
    movl    $0, %eax
    call    print_hello
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .section    .rodata
.LC0:
    .string "Hello, world!"
    .text
    .globl  print_hello
    .type   print_hello, @function
print_hello:
.LFB1:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    movl    $.LC0, %edi
    call    puts
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE1:
    .size   print_hello, .-print_hello
    .ident  "GCC: (GNU) 4.8.5 20150623 (Red Hat 4.8.5-44)"
    .section    .note.GNU-stack,"",@progbits

```

Looking at the lab5_prob3_1 and this file, we can see a few main differences. The first being that that instead of the main function calling puts, it called the function called print_hello(). The rest of the .LFB0 portion of the code is really similar and not much has changed other than the call to the different function. The new code then calls the function and then the following lines are executed. The first thing the function does is set up the stack pointer using movq %rsp, %rbp. Next, the function loads the address of the string "Hello, world!" into the %edi register using movl $.LC0, %edi. Then the puts function is called and this function puts the string to the screen. Then to finish the function, the stack frame is then removed by using popq %rbp and then it will return out with the ret line. Furthermore, the .cfi_def_cfa_register 6 command sets the current frame pointer register which is used to find the location of the stack pointer at any given point. In this specific example, register 6 is set to the location that points to the stack and if the stack is needed, the register will hold the stack pointer.

Problem 5.

prob5_main.s

```
    .file   "lab5_prob5_main.c"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    subq    $16, %rsp
    movl    %edi, -4(%rbp)
    movq    %rsi, -16(%rbp)
    movl    $0, %eax
    call    print_hello
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (GNU) 4.8.5 20150623 (Red Hat 4.8.5-44)"
    .section    .note.GNU-stack,"",@progbits

```

prob5_print.s

```
    .file   "lab5_prob5_print.c"
    .section    .rodata
.LC0:
    .string "Hello, world"
    .text
    .globl  print_hello
    .type   print_hello, @function
print_hello:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    movl    $.LC0, %edi
    call    puts
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   print_hello, .-print_hello
    .ident  "GCC: (GNU) 4.8.5 20150623 (Red Hat 4.8.5-44)"
    .section    .note.GNU-stack,"",@progbits
```

The print_hello() function signature is the same in both problems 4 and 5. In the prob5_print.s, the same exact calls are made to the stack which puts the string “Hello, world” to the screen. The main difference is that there are two separate files for the main and the function. In the prob5_main.s file, there is a call to print_hello but the function isn’t defined so it will not execute anything. Furthermore, the same calls are made in the main function as in problem 4 except they are just split up file by file. If the file was to be copied and pasted together, problem 5 and problem 4 would have the same output and look the same.

Problem 6.

lab5_prob6_asm.c > very_fast_function(int)

```c
include <stdio.h>
nt very_fast_function(int i){
    int result;
    asm("movl %1, %%eax\n\t" //moves variable i to eax
        "addl $1, %%eax\n\t" //adds one to i
        "imull $64, %%eax\n\t" //multiplies i by 64
        "cmpl $1024, %%eax\n\t" //compares the number to 1024
        "jg larger\n\t" //if greater, will jump
        "movl $0, %0\n\t" //if the number is less than it will make i = 0 and jump to the end
        "jmp end\n" //jump to the end if the number is less than
        "larger:\n\t" //larger condition
        "movl %1, %0\n" //will move i into the ebx register
        "end:" //end of the code
        : "=r" (result) //result is a memory operand and it holds the return value
        : "r" (i) //i is a register operand
        : "%eax", "cc" //this represents the condition codes that are chnaged in the assembly code
    );
    return result;



nt main(int argc, char *argv[]) {
    int i;
    i = 40;
    printf("The function value of i is %d\n", very_fast_function(i) );
    return 0;

```

```
ated]
(base) matthewpeterka@connect-172-31-16-159 lab5 % ./a.out
The function value of i is 40
(base) matthewpeterka@connect-172-31-16-159 lab5 %
```