For loop

```
        .file   "sumWithFor.c"
        .text
.globl sumArguments
        .type   sumArguments, @function
sumArguments:
.LFB31:
        .cfi_startproc
        movl    %edi, %edx
        movl    $0, %eax
        cmpl    %esi, %edi
        jg      .L3
.L6:
        addl    %edx, %eax
        addl    $1, %edx
        cmpl    %edx, %esi
        jge     .L6
.L3:
        rep; ret
        .cfi_endproc
.LFE31:
        .size   sumArguments, .-sumArguments
        .section        .rodata.str1.8,"aMS",@progbits,1
        .align 8
.LC0:
        .string  "The sum of all the numbers between %d and %d, inclusive, is %d\n"
        .text
.globl main
        .type   main, @function
main:
.LFB30:
        .cfi_startproc
        subq    $8, %rsp
        .cfi_def_cfa_offset 16
        movl    $10, %esi
        movl    $5, %edi
        call    sumArguments
        movl    %eax, %ecx
        movl    $10, %edx
        movl    $5, %esi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        movl    $0, %eax
```

```
        addq    $8, %rsp
        .cfi_def_cfa_offset 8
        ret
        .cfi_endproc
.LFE30:
        .size   main, .-main
        .ident  "GCC: (GNU) 4.4.7 20120313 (Red Hat 4.4.7-23)"
        .section        .note.GNU-stack,"",@progbits
```

While loop

```
        .file   "sumWithWhile.c"
        .text
.globl sumArguments
        .type   sumArguments, @function
sumArguments:
.LFB31:
        .cfi_startproc
        movl    $0, %eax
        cmpl    %esi, %edi
        jg      .L3
.L6:
        addl    %edi, %eax
        addl    $1, %edi
        cmpl    %edi, %esi
        jge     .L6
.L3:
        rep; ret
        .cfi_endproc
.LFE31:
        .size   sumArguments, .-sumArguments
        .section        .rodata.str1.8,"aMS",@progbits,1
        .align 8
.LC0:
        .string  "The sum of all the numbers between %d and %d, inclusive, is %d\n"
        .text
.globl main
        .type   main, @function
main:
.LFB30:
        .cfi_startproc
        subq    $8, %rsp
        .cfi_def_cfa_offset 16
        movl    $10, %esi
        movl    $5, %edi
```

```
        call    sumArguments
        movl    %eax, %ecx
        movl    $10, %edx
        movl    $5, %esi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        movl    $0, %eax
        addq    $8, %rsp
        .cfi_def_cfa_offset 8
        ret
        .cfi_endproc
.LFE30:
        .size   main, .-main
        .ident  "GCC: (GNU) 4.4.7 20120313 (Red Hat 4.4.7-23)"
        .section        .note.GNU-stack,"",@progbits
```

Do … while loop

```
        .file   "sumWithDo.c"
        .text
.globl sumArguments
        .type   sumArguments, @function
sumArguments:
.LFB31:
        .cfi_startproc
        movl    $0, %eax
.L2:
        addl    %edi, %eax
        addl    $1, %edi
        cmpl    %esi, %edi
        jle     .L2
        rep; ret
        .cfi_endproc
.LFE31:
        .size   sumArguments, .-sumArguments
        .section        .rodata.str1.8,"aMS",@progbits,1
        .align 8
.LC0:
        .string  "The sum of all the numbers between %d and %d, inclusive, is %d\n"
        .text
.globl main
        .type   main, @function
main:
.LFB30:
```

```
        .cfi_startproc
        subq    $8, %rsp
        .cfi_def_cfa_offset 16
        movl    $10, %esi
        movl    $5, %edi
        call    sumArguments
        movl    %eax, %ecx
        movl    $10, %edx
        movl    $5, %esi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        movl    $0, %eax
        addq    $8, %rsp
        .cfi_def_cfa_offset 8
        ret
        .cfi_endproc
.LFE30:
        .size   main, .-main
        .ident  "GCC: (GNU) 4.4.7 20120313 (Red Hat 4.4.7-23)"
        .section        .note.GNU-stack,"",@progbits
```

GoTo loop

```
        .file   "sumWithGoTo.c"
        .text
.globl sumArguments
        .type   sumArguments, @function
sumArguments:
.LFB31:
        .cfi_startproc
        movl    $0, %eax
        cmpl    %esi, %edi
        jg      .L5
.L3:
        addl    %edi, %eax
        addl    $1, %edi
.L4:
        cmpl    %edi, %esi
        jge     .L3
.L5:
        rep; ret
        .cfi_endproc
.LFE31:
```

```
        .size   sumArguments, .-sumArguments
        .section        .rodata.str1.8,"aMS",@progbits,1
        .align 8
.LC0:
        .string  "The sum of all the numbers between %d and %d, inclusive, is %d\n"
        .text
.globl main
        .type   main, @function
main:
.LFB30:
        .cfi_startproc
        subq    $8, %rsp
        .cfi_def_cfa_offset 16
        movl    $10, %esi
        movl    $5, %edi
        call    sumArguments
        movl    %eax, %ecx
        movl    $10, %edx
        movl    $5, %esi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        movl    $0, %eax
        addq    $8, %rsp
        .cfi_def_cfa_offset 8
        ret
        .cfi_endproc
.LFE30:
        .size   main, .-main
        .ident  "GCC: (GNU) 4.4.7 20120313 (Red Hat 4.4.7-23)"
        .section        .note.GNU-stack,"",@progbits
```

Comparison of the four assembly codes

The main section of the code is the same for all four scripts as the main section in the C code is identical in all four. The only changes occur in the sumArguments function. There are a couple of differences in the sumArguments section.

First, the for loop version has an additional line at the start of the section: movl %edi, %edx. This line is added because the for loop initializes an additional variable, i, which is used in the loop. This could have been avoided by using different expressions in the for loop:

for(int i = num1; i <= num2; i++) {}

Could have been replaced with:

for(num1 <= num2; num1++){}

Which would have achieved the same results without the inclusion of the mov instruction in the assembly code.

Additionally, because of the inclusion of int i in the for loop, there is another register used, %edx, which is used in the addl %edx, %eax and addl $1, %edx; however, the three other loops use %edi. This difference could be avoided using the same for loop change described above. Beyond these two differences, there are no other differences between the for and the while loops.

The do loop is different from the for loop in a couple ways. There is no comparison and jump at the start of the section. This is because the do loop always executes at least once, so it will evaluate the jump conditions after the loop has been executed instead of before. There is also only one label whereas the others all have at least two. That is because there is only one jump condition, at the end of the loop whereas the for and the while loop both have two jump conditions, one for the initial entry into the loop and another to determine whether to iterate through the loop again.

Finally, the goto loop has the most unique looking assembly code. The main difference is the additional label, with this code having three instead of the two in the for and while loop versions.