First , I gonna show the graders how the original mystery .s works logically . The mystery stuff is using recursion way to find the Fibonacci number of the passing parameter .

Here is the mystery.s given by the professor . Look to the main function below :

```
.file  "mystery.c"
        .text
.globl add
        .type    add, @function
add:
        pushl    %ebp
        movl     %esp, %ebp
        movl     12(%ebp), %eax
        addl     8(%ebp), %eax
        popl %ebp
        ret
        .size add, .-add
.globl dothething
        .type    dothething, @function
dothething:
        pushl    %ebp
        movl     %esp, %ebp
        pushl    %ebx
        subl     $28, %esp
        movl     8(%ebp), %eax
        movl     num(,%eax,4), %eax
        cmpl     $-1, %eax
        je       .L4
        movl     8(%ebp), %eax
        movl     num(,%eax,4), %eax
        movl     %eax, -24(%ebp)
        jmp      .L6
.L4:
        movl     $-1, -8(%ebp)
        cmpl     $0, 8(%ebp)
        jne      .L7
        movl     $0, -8(%ebp)
        jmp      .L9
.L7:
        cmpl     $1, 8(%ebp)
        jne      .L10
        movl     $1, -8(%ebp)
        jmp  .L9
.L10:
        movl     8(%ebp), %eax
        subl     $2, %eax
```

The add function here are just do the mathematical add operation ,it will be called when the int passed to the dothething function is neither 0 or 1;

Here is the dothething function .first it will compare the passing int with 0 and 1 , corresponding at the label L4 and label L7. If the int is equal to 0 or 1 , it will just return 0 or 1 corresponding .If it is not ,then go to the L10

```
        movl    %eax, (%esp)
        call    dothething
        movl    %eax, %ebx
        movl    8(%ebp), %eax
        subl    $1, %eax
        movl    %eax, (%esp)
        call    dothething
        movl    %ebx, 4(%esp)
        movl    %eax, (%esp)
        call    add
        movl    %eax, -8(%ebp)
.L9:
        movl    8(%ebp), %eax
        movl    num(,%eax,4), %eax
        cmpl    $-1, %eax
        jne     .L12
        movl    8(%ebp), %edx
        movl    -8(%ebp), %eax
        movl    %eax, num(,%edx,4)
.L12:
        movl    8(%ebp), %eax
        movl    num(,%eax,4), %eax
        movl    %eax, -24(%ebp)
.L6:
        movl    -24(%ebp), %eax
        addl    $28, %esp
        popl    %ebx
        popl    %ebp
        ret
        .size dothething, .-dothething
        .section    .rodata
.LC0:
        .string "Value:     %d\n"
        .text
.globl main
        .type   main, @function
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ecx
        subl    $36, %esp
        movl    4(%ecx), %eax
        addl    $4, %eax
        movl    (%eax), %eax
        movl    %eax, (%esp)
        call    atoi
```

Here, the dothething function just recursively call the dothething . It just return the dothething(n-2)+dothething(n-1).

Here the main function to get the data from the command line and call atoi to turn the string to the int type.

```
        movl    %eax, -12(%ebp)
        movl    $0, -8(%ebp)
        jmp  .L16
.L17:
        movl    -8(%ebp), %eax
        movl    $-1, num(,%eax,4)
        addl    $1, -8(%ebp)
.L16:
        cmpl    $199, -8(%ebp)
        jle     .L17
        movl    -12(%ebp), %eax
        movl    %eax, (%esp)
        call    dothething
        movl    %eax, 4(%esp)
        movl    $.LC0, (%esp)
        call    printf
        movl    $0, %eax
        addl    $36, %esp
        popl    %ecx
        popl    %ebp
        leal    -4(%ecx), %esp
        ret
        .size main, .-main
        .comm   num,800,32
        .ident  "GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-51)"
        .section    .note.GNU-stack,"",@progbits
```

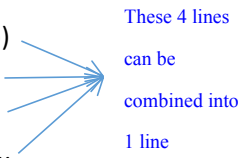
Here , compare the the int with 199 , If less or equal to 199 ,go to the dothething function . If it is not , printf 0.
Else , it will printf whatever return from the dothething function and end up the program

Then , since I figure out that the mystery is doing the Fibonacci job in the recursion way , it is very expensive whether for the memory or for the running time , so I decided to implement the same task with loop operation rather than recursion . What is shown below is the comparison between the “-O” compilation and without the “-O” compilation . My C code is in the mystery .c file. The left side below is the optimization version and the right side is the normal version .

```
.file  "mystery.c"
       .text
       .globl      mystery
       .type       mystery, @function
mystery:
.LFB54:
       .cfi_startproc
       pushl       %esi
       .cfi_def_cfa_offset 8
       .cfi_offset 6, -8
       pushl       %ebx
       .cfi_def_cfa_offset 12
       .cfi_offset 3, -12
       movl        12(%esp), %esi
       Cmpl        $2, %esi
       jle         .L4
       movl        $3, %edx
       movl        $1, %ecx
       movl        $1, %ebx
.L3:
       leal        (%ebx,%ecx), %eax
       addl        $1, %edx
       cmpl        %edx, %esi
       jl          .L2
       movl        %ecx, %ebx
       movl        %eax, %ecx
       jmp         .L3
.L4:
       movl        $1, %eax
.L2:
       popl        %ebx
       .cfi_restore 3
       .cfi_def_cfa_offset 8
       popl        %esi
       .cfi_restore 6
       .cfi_def_cfa_offset 4
       ret
       .cfi_endproc
.LFE54:
       .size mystery, .-mystery
       .section
       .rodata.str1.1,"aMS",@progbits,1
.LC0:
       .string     "value:\t %d\n"
       .text
       .globl      main
       .type       main, @function
```

Look at the mystery function . The optimization is pretty cool ..The compiler seems to figure out that it is just a loop .. So it "smartly" takes it as a general loop operation rather than create a new stack frame in the right side . The left side use three registers %ebx %ecx %edx to store the arithmetic data which is pretty straightforward and efficient . Compare the label 3 in both sides , the left one use the lea instruction just simply add the content in the %ebx and %ecx and put the result into the %eax which is very smart ..

```
.file  "mystery.c"
       .text
       .globl      mystery
       .type       mystery, @function
mystery:
.LFB2:
       .cfi_startproc
       pushl       %ebp
       .cfi_def_cfa_offset 8
       .cfi_offset 5, -8
       movl        %esp, %ebp
       .cfi_def_cfa_register 5
       subl        $16, %esp
       movl        $1, -4(%ebp)
       movl        $1, -8(%ebp)
       movl        $3, -12(%ebp)
       jmp         .L2
.L3:
       movl        -8(%ebp), %eax
       movl        -4(%ebp), %edx
       addl        %edx, %eax
       movl        %eax, -16(%ebp)
       movl        -8(%ebp), %eax
       movl        %eax, -4(%ebp)
       movl        -16(%ebp), %eax
       movl        %eax, -8(%ebp)
       addl        $1, -12(%ebp)
.L2:
       movl        -12(%ebp), %eax
       cmpl        8(%ebp), %eax
       jle         .L3
       movl        -8(%ebp), %eax
       leave
       .cfi_restore 5
       .cfi_def_cfa 4, 4
       ret
       .cfi_endproc
.LFE2:
       .size mystery, .-mystery
       .section    .rodata
.LC0:
       .string     "value:\t %d\n"
       .text
       .globl      main
       .type       main, @function
```



```
main:
.LFB55:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        andl    $-16, %esp
        subl    $16, %esp
        movl    $10, 8(%esp)
        movl    $0, 4(%esp)
        movl    12(%ebp), %eax
        movl    4(%eax), %eax
        movl    %eax, (%esp)
        call    strtol
        cmpl    $199, %eax
        jle     .L7
        movl    $0, 4(%esp)
        movl    $.LC0, (%esp)
        call    printf
        jmp     .L8
.L7:
        movl    %eax, (%esp)
        call    mystery
        movl    %eax, 4(%esp)
        movl    $.LC0, (%esp)
        call    printf
.L8:
        movl    $0, %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE55:
        .size main, .-main
        .ident  "GCC: (GNU) 4.8.5 20150623
(Red Hat 4.8.5-4)"
        .section .note.GNU-stack,"",@progbits
```

Here compare to the right, the compiler know that 16 space is enough for the stack frame. So it just sub 16 with the %esp rather than 32. And the left side just directly located the argument with mov instruction , it will be more efficient. Then , the lest side call strtol rather than the atoi, it is because the strtol can return whether the string can be convert to int or not , the atoi can not tell us the "0" and the failure operation ..as for the comparison with 199, the left side directly mov 0 to the esp and return .

As for the calling of mystery function . The left side is pretty straightforward.

```
main:
.LFB3:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        andl    $-16, %esp
        subl    $32, %esp
        movl    12(%ebp), %eax
        movl    4(%eax), %eax
        movl    %eax, 28(%esp)
        movl    28(%esp), %eax
        movl    %eax, (%esp)
        call    atoi
        movl    %eax, 24(%esp)
        movl    $0, 20(%esp)
        cmpl    $199, 24(%esp)
        jle     .L6
        movl    20(%esp), %eax
        movl    %eax, 4(%esp)
        movl    $.LC0, (%esp)
        call    printf
        movl    $0, %eax
        jmp     .L7
.L6:
        movl    24(%esp), %eax
        movl    %eax, (%esp)
        call    mystery
        movl    %eax, 20(%esp)
        movl    20(%esp), %eax
        movl    %eax, 4(%esp)
        movl    $.LC0, (%esp)
        call    printf
        movl    $0, %eax
.L7:
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE3:
        .size main, .-main
        .ident  "GCC: (GNU) 4.8.5 20150623
(Red Hat 4.8.5-4)"
        .section .note.GNU-stack,"",@progbits
```

This two lines is worthless

These two lines can be combined

These two lines can be combined into one

These three lines can be combined into one