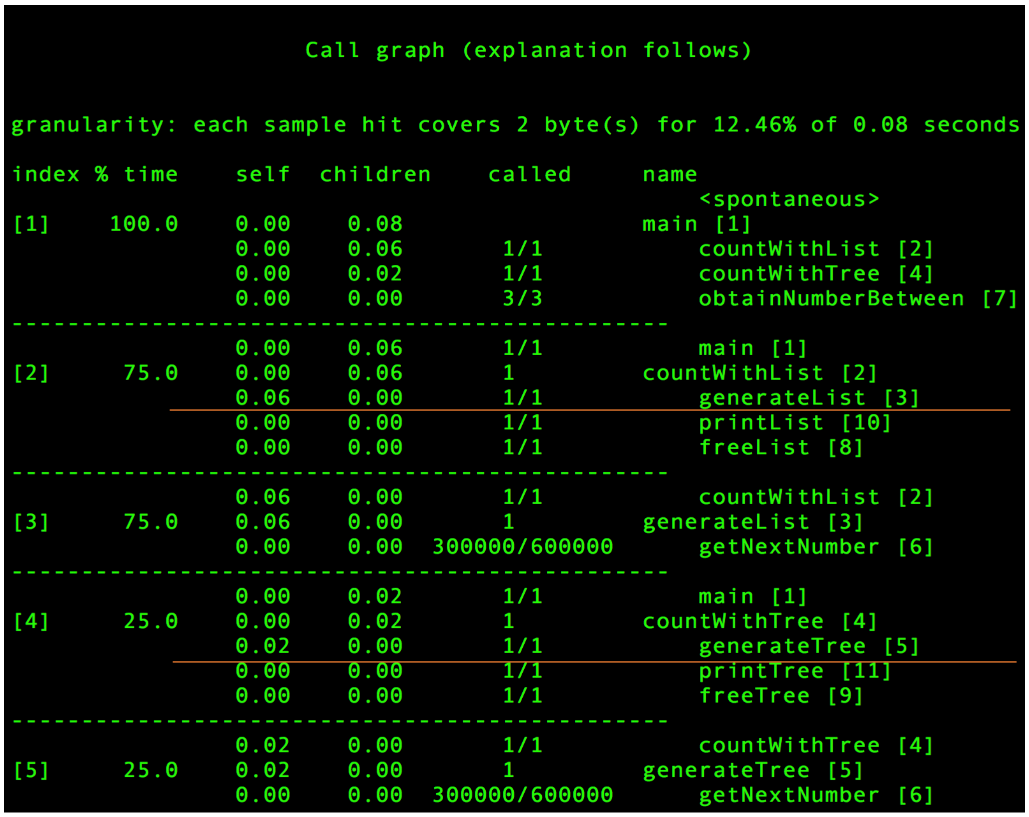
CSC407
Haonan(Harry Chen)
Homework1

1. C programming
   See harry.zip for detail.
2. Timing Part1(with no optimization)
   Here is the report of Function Call without optimization

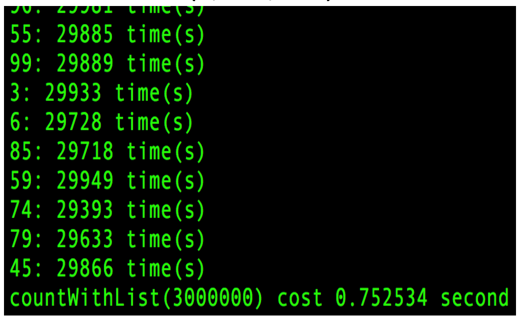


```
                         Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 12.46% of 0.08 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.08                 main [1]
                0.00    0.06       1/1           countWithList [2]
                0.00    0.02       1/1           countWithTree [4]
                0.00    0.00       3/3           obtainNumberBetween [7]
-----------------------------------------------
                0.00    0.06       1/1           main [1]
[2]     75.0    0.00    0.06       1         countWithList [2]
                0.06    0.00       1/1           generateList [3]
                0.00    0.00       1/1           printList [10]
                0.00    0.00       1/1           freeList [8]
-----------------------------------------------
                0.06    0.00       1/1           countWithList [2]
[3]     75.0    0.06    0.00       1         generateList [3]
                0.00    0.00  300000/600000      getNextNumber [6]
-----------------------------------------------
                0.00    0.02       1/1           main [1]
[4]     25.0    0.00    0.02       1         countWithTree [4]
                0.02    0.00       1/1           generateTree [5]
                0.00    0.00       1/1           printTree [11]
                0.00    0.00       1/1           freeTree [9]
-----------------------------------------------
                0.02    0.00       1/1           countWithTree [4]
[5]     25.0    0.02    0.00       1         generateTree [5]
                0.00    0.00  300000/600000      getNextNumber [6]
```

   GenerateList cost 0.06 self second and GenerateTree cost 0.02 self second without optimization.

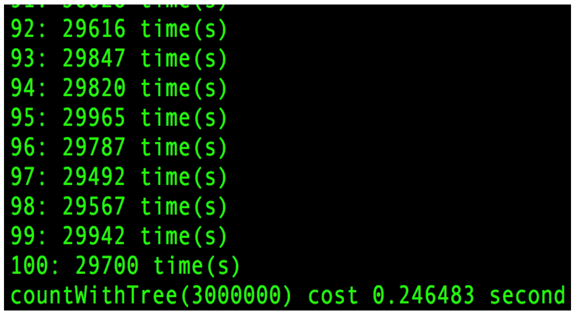
   I also write a “timer function” embed in the main function, here are the results

   countWithList(3,000,000)



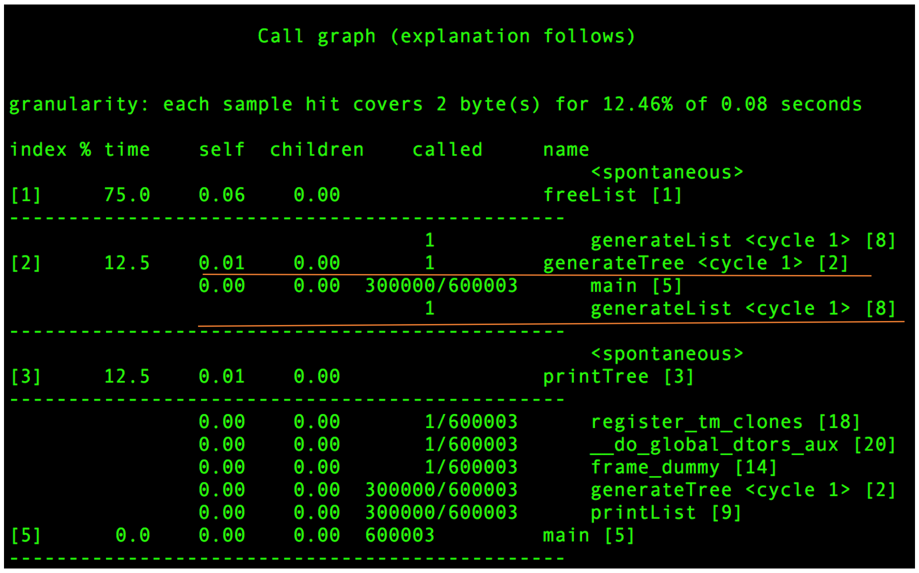
```
55: 29885 time(s)
99: 29889 time(s)
3: 29933 time(s)
6: 29728 time(s)
85: 29718 time(s)
59: 29949 time(s)
74: 29393 time(s)
79: 29633 time(s)
45: 29866 time(s)
countWithList(3000000) cost 0.752534 second
```

   countWithTree(3,000,000)



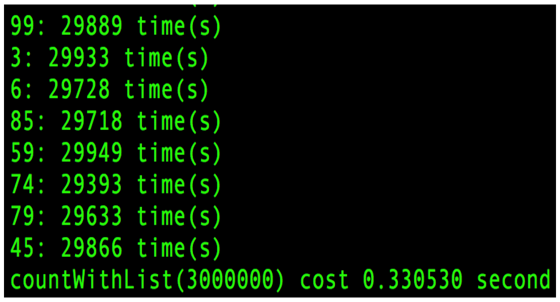
```
92: 29616 time(s)
93: 29847 time(s)
94: 29820 time(s)
95: 29965 time(s)
96: 29787 time(s)
97: 29492 time(s)
98: 29567 time(s)
99: 29942 time(s)
100: 29700 time(s)
countWithTree(3000000) cost 0.246483 second
```

3. Timing Part2(with optimization)
   Here is the report of Function Call with optimization



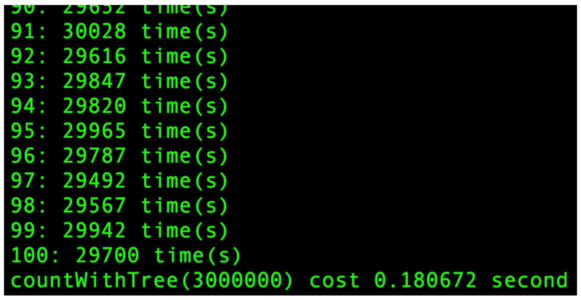
```
                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 12.46% of 0.08 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]     75.0    0.06    0.00                 freeList [1]
-----------------------------------------------
                                   1             generateList <cycle 1> [8]
[2]     12.5    0.01    0.00       1         generateTree <cycle 1> [2]
                0.00    0.00  300000/600003      main [5]
                                   1             generateList <cycle 1> [8]
-----------------------------------------------
                                                 <spontaneous>
[3]     12.5    0.01    0.00                 printTree [3]
-----------------------------------------------
                0.00    0.00       1/600003      register_tm_clones [18]
                0.00    0.00       1/600003      __do_global_dtors_aux [20]
                0.00    0.00       1/600003      frame_dummy [14]
                0.00    0.00  300000/600003      generateTree <cycle 1> [2]
                0.00    0.00  300000/600003      printList [9]
[5]      0.0    0.00    0.00  600003         main [5]
-----------------------------------------------
```

countWithList(3,000,000)



```
99: 29889 time(s)
3: 29933 time(s)
6: 29728 time(s)
85: 29718 time(s)
59: 29949 time(s)
74: 29393 time(s)
79: 29633 time(s)
45: 29866 time(s)
countWithList(3000000) cost 0.330530 second
```

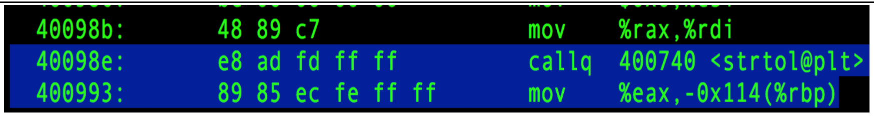
countWithTree(3,000,000)

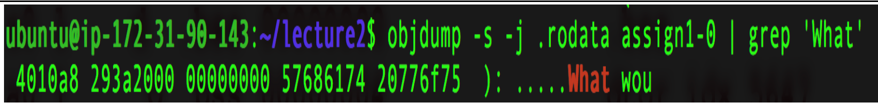


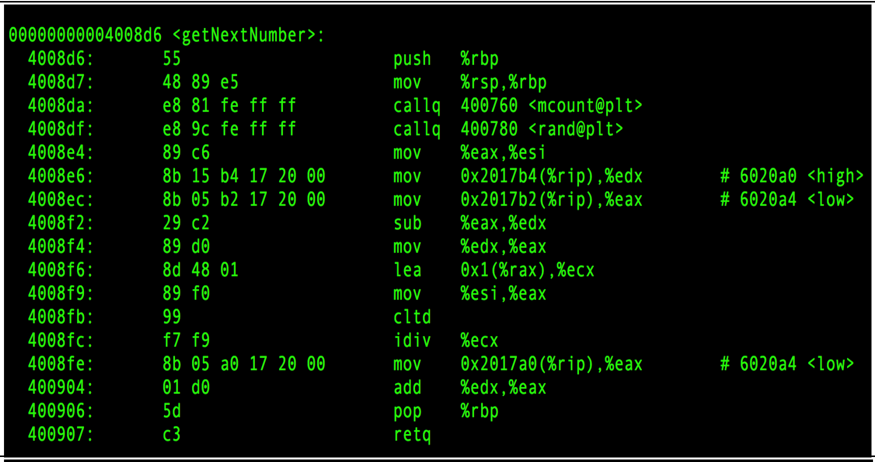
```
91: 30028 time(s)
92: 29616 time(s)
93: 29847 time(s)
94: 29820 time(s)
95: 29965 time(s)
96: 29787 time(s)
97: 29492 time(s)
98: 29567 time(s)
99: 29942 time(s)
100: 29700 time(s)
countWithTree(3000000) cost 0.180672 second
```

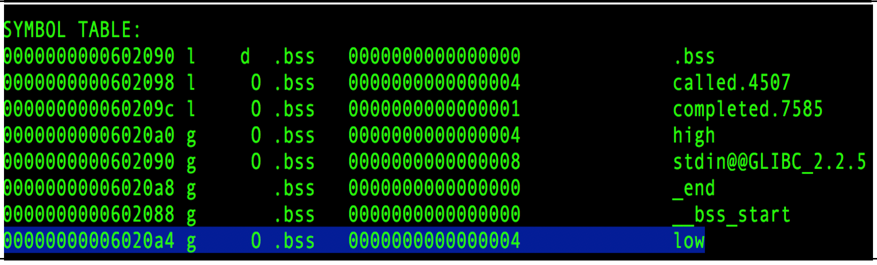
4. Parts of an executable

Question	Command	Result

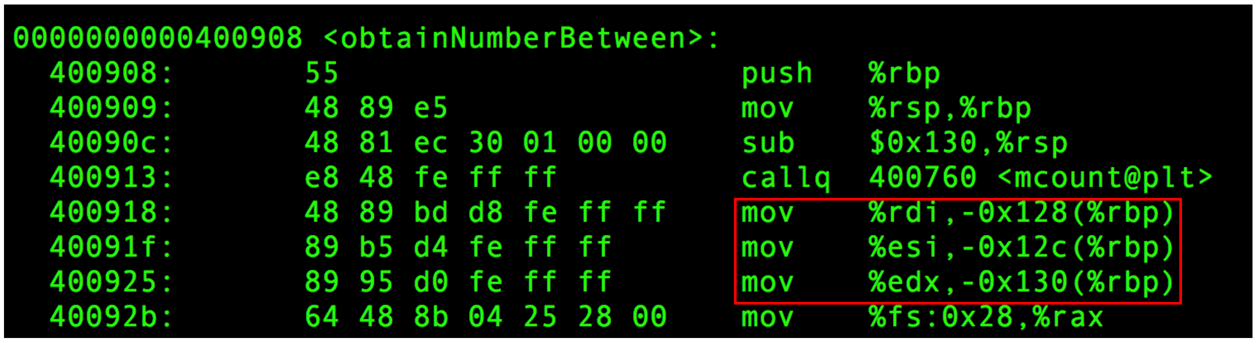
(A)	**CANNOT BE FOUND**	 `40098b:  48 89 c7  mov  %rax,%rdi`
		`40098e:  e8 ad fd ff ff  callq  400740 <strtol@plt>`
		`400993:  89 85 ec fe ff ff  mov  %eax,-0x114(%rbp)`
		Entry is a local variable in `` `obtainNumberBetween()` `` function, it will be save in %eax register in runtime.

(B)	**objdump -s -j .rodata assign1-0 | grep 'What'**	 `ubuntu@ip-172-31-90-143:~/lecture2$ objdump -s -j .rodata assign1-0 | grep 'What'` `4010a8 293a2000 00000000 57686174 20776f75  ): .....What wou`

(C)	**objdump -d -j .text assign1-0**	 `00000000004008d6 <getNextNumber>:`
		`4008d6:  55  push  %rbp`
		`4008d7:  48 89 e5  mov  %rsp,%rbp`
		`4008da:  e8 81 fe ff ff  callq  400760 <mcount@plt>`
		`4008df:  e8 9c fe ff ff  callq  400780 <rand@plt>`
		`4008e4:  89 c6  mov  %eax,%esi`
		`4008e6:  8b 15 b4 17 20 00  mov  0x2017b4(%rip),%edx  # 6020a0 <high>`
		`4008ec:  8b 05 b2 17 20 00  mov  0x2017b2(%rip),%eax  # 6020a4 <low>`
		`4008f2:  29 c2  sub  %eax,%edx`
		`4008f4:  89 d0  mov  %edx,%eax`
		`4008f6:  8d 48 01  lea  0x1(%rax),%ecx`
		`4008f9:  89 f0  mov  %esi,%eax`
		`4008fb:  99  cltd`
		`4008fc:  f7 f9  idiv  %ecx`
		`4008fe:  8b 05 a0 17 20 00  mov  0x2017a0(%rip),%eax  # 6020a4 <low>`
		`400904:  01 d0  add  %edx,%eax`
		`400906:  5d  pop  %rbp`
		`400907:  c3  retq`

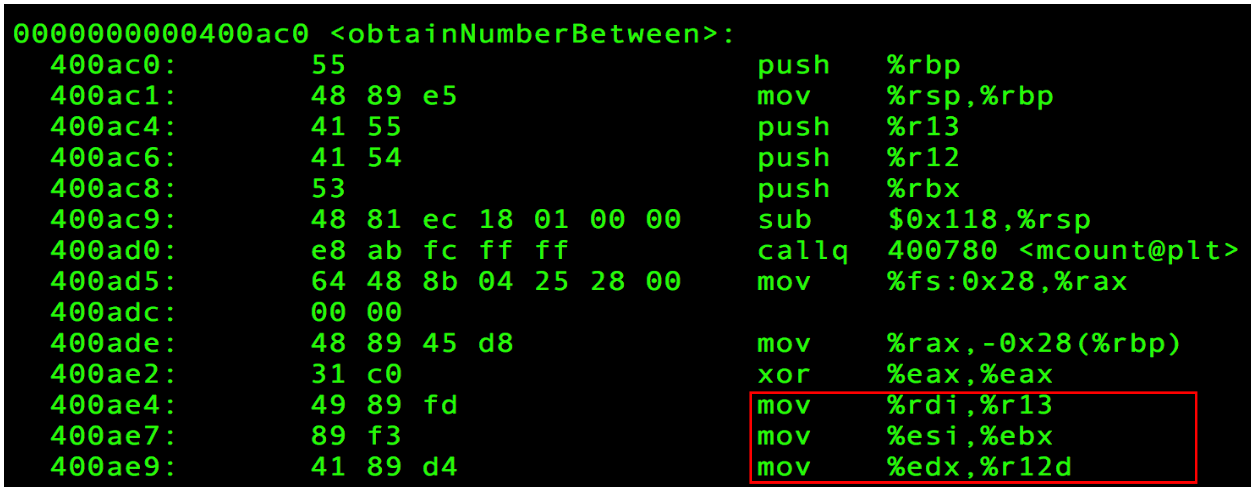
(D)	**objdump -t -j .bss assign1-0**	 `SYMBOL TABLE:`
		`0000000000602090 l  d .bss  0000000000000000  .bss`
		`0000000000602098 l  O .bss  0000000000000004  called.4507`
		`000000000060209c l  O .bss  0000000000000001  completed.7585`
		`00000000006020a0 g  O .bss  0000000000000004  high`
		`0000000000602090 g  O .bss  0000000000000008  stdin@@GLIBC_2.2.5`
		`00000000006020a8 g    .bss  0000000000000000  _end`
		`0000000000602088 g    .bss  0000000000000000  __bss_start`
		`00000000006020a4 g  O .bss  0000000000000004  low`

5. Compare optimizations.
   a. Save in register rather than in memory
   Non-optimization version use memory to save the value of 3 arguments includes descriptionCPtr, low and high in `obtainNumberBetween()` function.



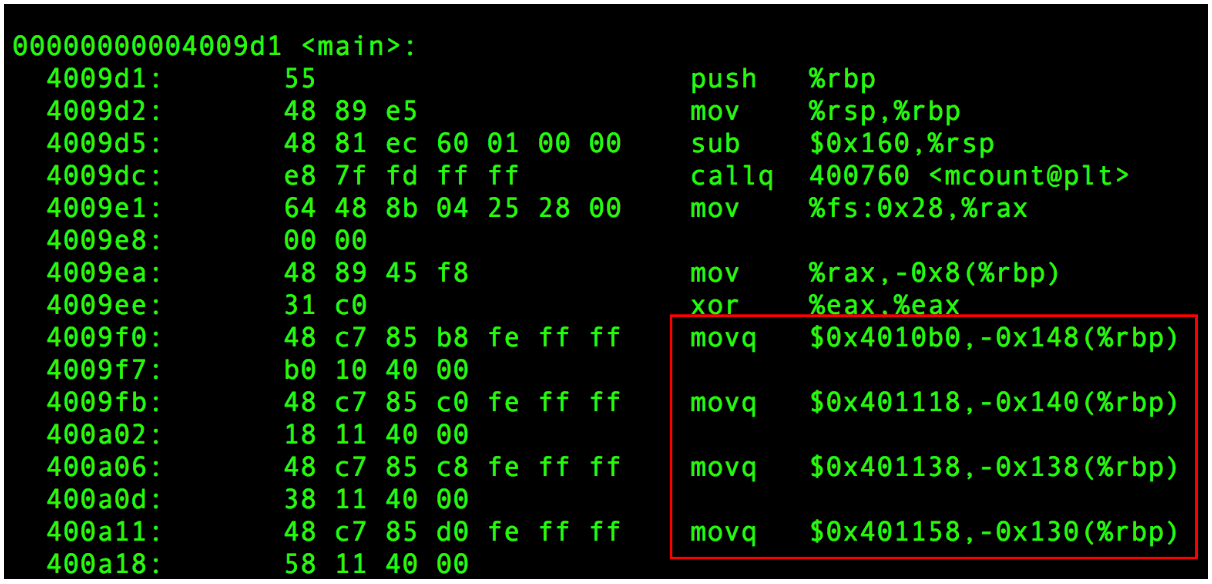
```
0000000000400908 <obtainNumberBetween>:
  400908:       55                      push   %rbp
  400909:       48 89 e5                mov    %rsp,%rbp
  40090c:       48 81 ec 30 01 00 00    sub    $0x130,%rsp
  400913:       e8 48 fe ff ff          callq  400760 <mcount@plt>
  400918:       48 89 bd d8 fe ff ff    mov    %rdi,-0x128(%rbp)
  40091f:       89 b5 d4 fe ff ff       mov    %esi,-0x12c(%rbp)
  400925:       89 95 d0 fe ff ff       mov    %edx,-0x130(%rbp)
  40092b:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
```

Optimization version use register to save the value.



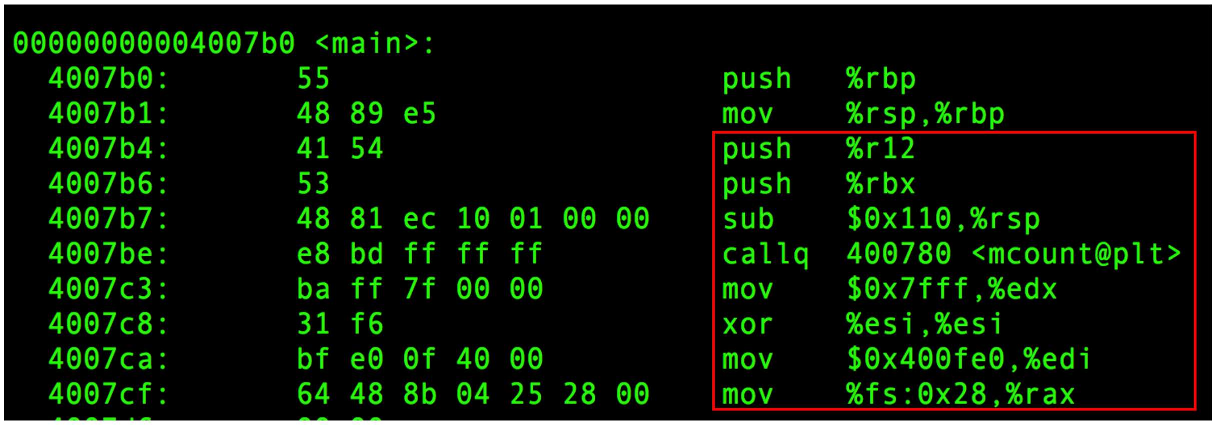
```
0000000000400ac0 <obtainNumberBetween>:
  400ac0:       55                      push   %rbp
  400ac1:       48 89 e5                mov    %rsp,%rbp
  400ac4:       41 55                   push   %r13
  400ac6:       41 54                   push   %r12
  400ac8:       53                      push   %rbx
  400ac9:       48 81 ec 18 01 00 00    sub    $0x118,%rsp
  400ad0:       e8 ab fc ff ff          callq  400780 <mcount@plt>
  400ad5:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  400adc:       00 00
  400ade:       48 89 45 d8             mov    %rax,-0x28(%rbp)
  400ae2:       31 c0                   xor    %eax,%eax
  400ae4:       49 89 fd                mov    %rdi,%r13
  400ae7:       89 f3                   mov    %esi,%ebx
  400ae9:       41 89 d4                mov    %edx,%r12d
```

b. There is another optimization example of using register rather than memory
Assign1-0 use memory



```
00000000004009d1 <main>:
  4009d1:       55                      push   %rbp
  4009d2:       48 89 e5                mov    %rsp,%rbp
  4009d5:       48 81 ec 60 01 00 00    sub    $0x160,%rsp
  4009dc:       e8 7f fd ff ff          callq  400760 <mcount@plt>
  4009e1:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  4009e8:       00 00
  4009ea:       48 89 45 f8             mov    %rax,-0x8(%rbp)
  4009ee:       31 c0                   xor    %eax,%eax
  4009f0:       48 c7 85 b8 fe ff ff    movq   $0x4010b0,-0x148(%rbp)
  4009f7:       b0 10 40 00
  4009fb:       48 c7 85 c0 fe ff ff    movq   $0x401118,-0x140(%rbp)
  400a02:       18 11 40 00
  400a06:       48 c7 85 c8 fe ff ff    movq   $0x401138,-0x138(%rbp)
  400a0d:       38 11 40 00
  400a11:       48 c7 85 d0 fe ff ff    movq   $0x401158,-0x130(%rbp)
  400a18:       58 11 40 00
```

Assign1-2 use register



```
00000000004007b0 <main>:
  4007b0:       55                      push   %rbp
  4007b1:       48 89 e5                mov    %rsp,%rbp
  4007b4:       41 54                   push   %r12
  4007b6:       53                      push   %rbx
  4007b7:       48 81 ec 10 01 00 00    sub    $0x110,%rsp
  4007be:       e8 bd ff ff ff          callq  400780 <mcount@plt>
  4007c3:       ba ff 7f 00 00          mov    $0x7fff,%edx
  4007c8:       31 f6                   xor    %esi,%esi
  4007ca:       bf e0 0f 40 00          mov    $0x400fe0,%edi
  4007cf:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
```