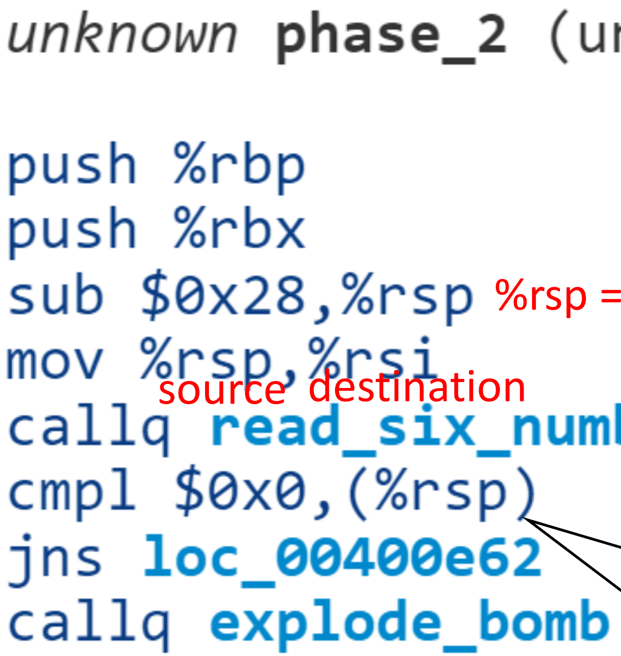
Assembly 101

- opcode
  - instruction for computer
  - e.g., push, sub, mov, callq, cmpl, jns
- registers
  - starting with ‘%’
  - e.g., %rbp, %rbx, %rsp, %rsi
- constant
  - Starting with ‘$’
  - e.g., $0x28 (numeric constant in hexadecimal)



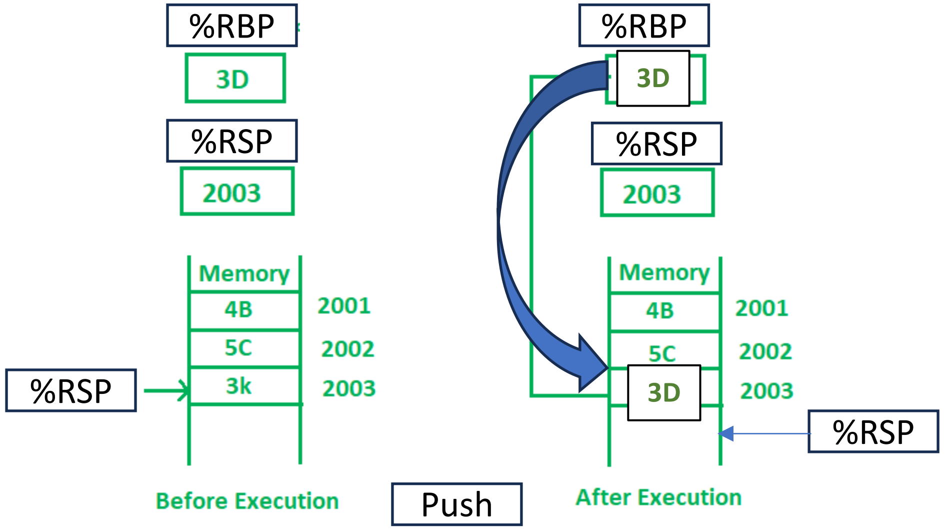
```
unknown phase_2 (unknown)

push %rbp
push %rbx
sub $0x28,%rsp
mov %rsp,%rsi
callq read_six_numbers
cmpl $0x0,(%rsp)
jns loc_00400e62
callq explode_bomb
```

%rsp = %rsp – 0x28

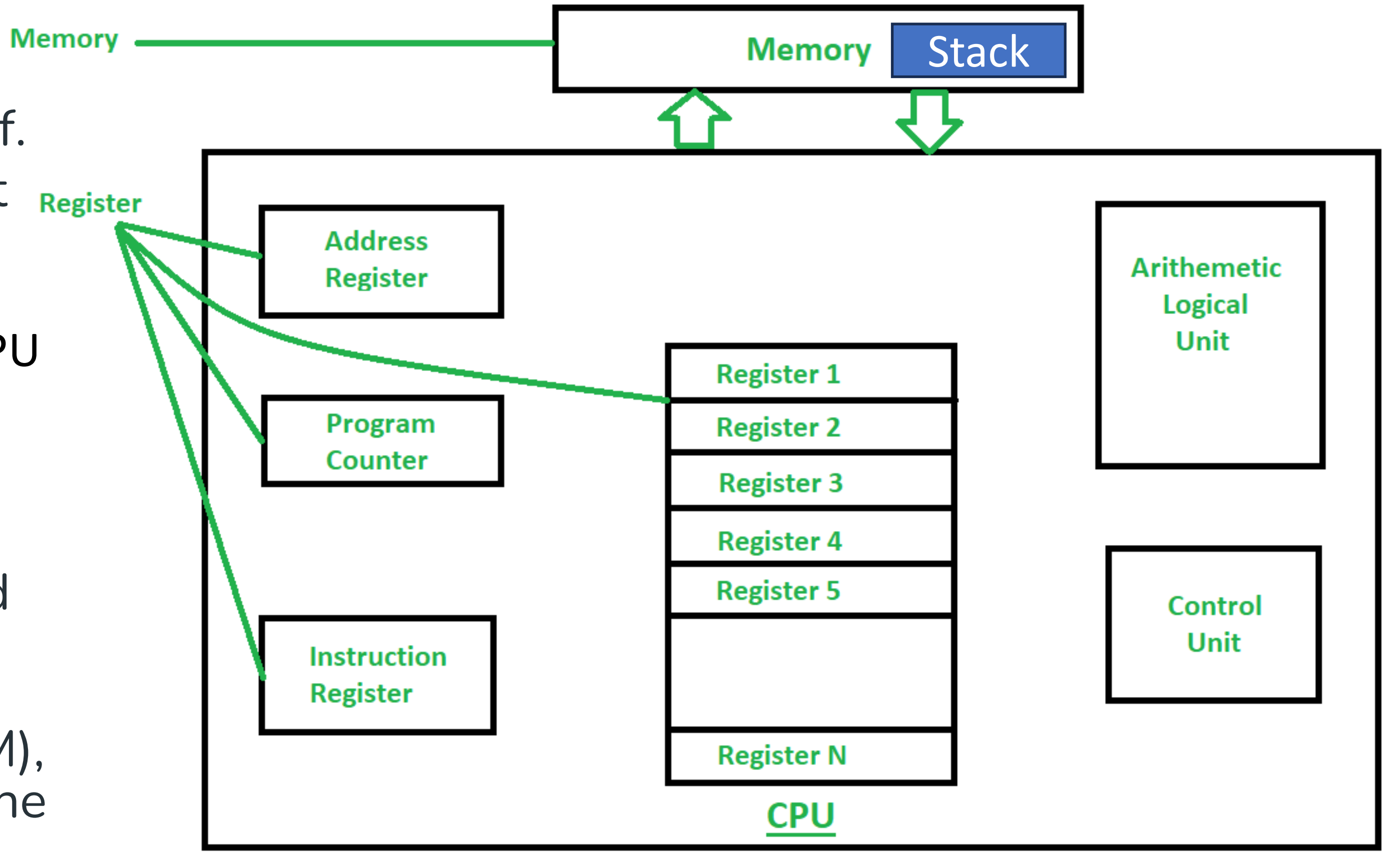
source destination

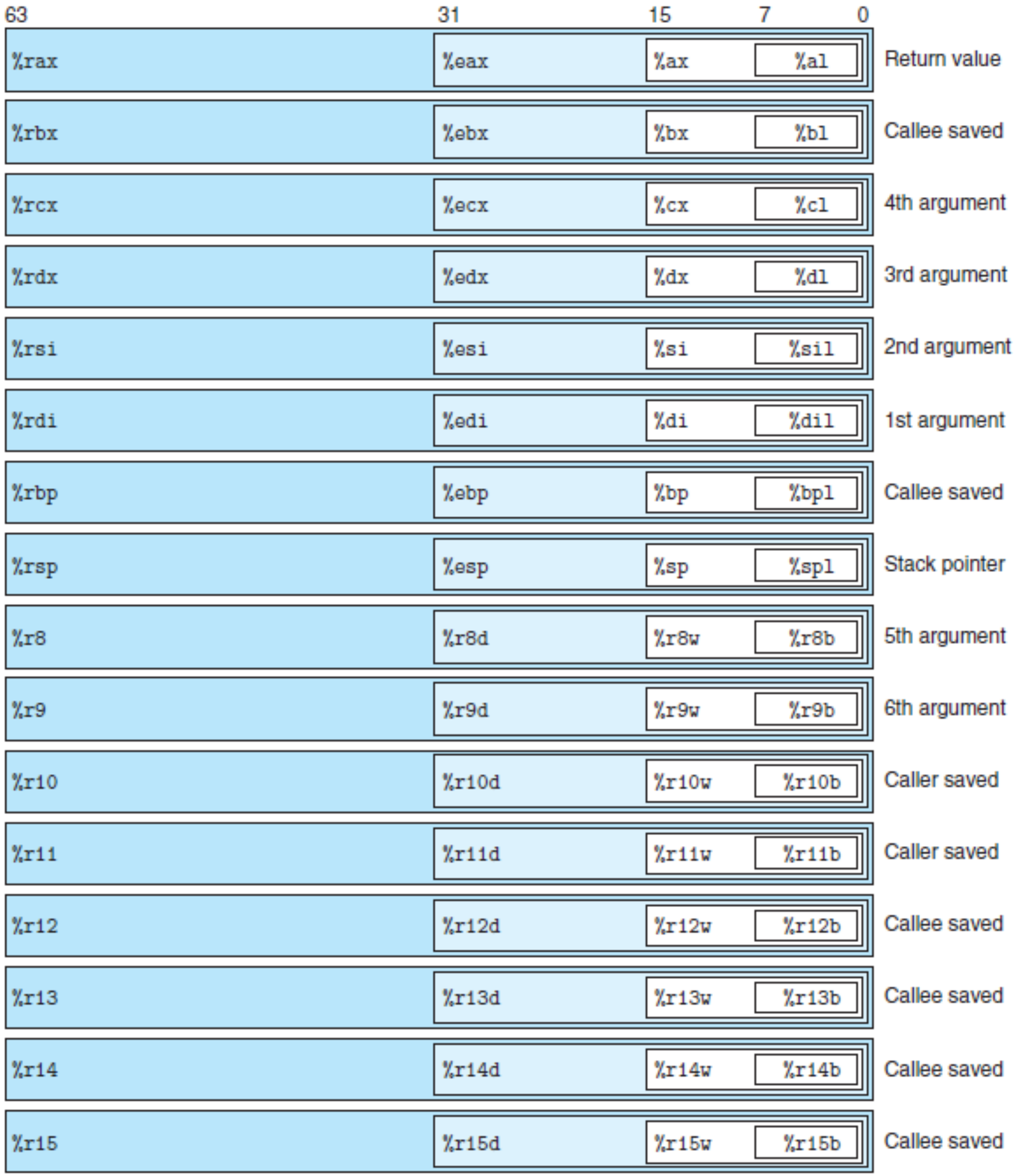
Memory Locations
(Equivalent to “*” in C)



Register and stack

- Register
  - The smallest data holding elements that are built into the processor itself.
  - These are the memory locations that are directly accessible by the processor.
  - Hold the operands or instruction that CPU is currently processing
- Memory
  - A hardware device used to store computer programs, instructions and data.
  - The memory that is internal to the processor is a primary memory (RAM), and the memory that is external to the processor is a SSD or HDD





63	31	15	7 0	
%rax	%eax	%ax	%al	Return value
%rbx	%ebx	%bx	%bl	Callee saved
%rcx	%ecx	%cx	%cl	4th argument
%rdx	%edx	%dx	%dl	3rd argument
%rsi	%esi	%si	%sil	2nd argument
%rdi	%edi	%di	%dil	1st argument
%rbp	%ebp	%bp	%bpl	Callee saved
%rsp	%esp	%sp	%spl	Stack pointer
%r8	%r8d	%r8w	%r8b	5th argument
%r9	%r9d	%r9w	%r9b	6th argument
%r10	%r10d	%r10w	%r10b	Caller saved
%r11	%r11d	%r11w	%r11b	Caller saved
%r12	%r12d	%r12w	%r12b	Callee saved
%r13	%r13d	%r13w	%r13b	Callee saved
%r14	%r14d	%r14w	%r14b	Callee saved
%r15	%r15d	%r15w	%r15b	Callee saved

Figure 3.2 **Integer registers.** The low-order portions of all 16 registers can be accessed as byte, word (16-bit), double word (32-bit), and quad word (64-bit) quantities.

CF: Carry flag. The most recent operation generated a carry out of the most significant bit. Used to detect overflow for unsigned operations.

ZF: Zero flag. The most recent operation yielded zero.

SF: Sign flag. The most recent operation yielded a negative value.

OF: Overflow flag. The most recent operation caused a two's-complement overflow—either negative or positive.

C declaration	Intel data type	Assembly-code suffix	Size (bytes)
char	Byte	b	1
short	Word	w	2
int	Double word	l	4
long	Quad word	q	8
char *	Quad word	q	8
float	Single precision	s	4
double	Double precision	l	8

Figure 3.1 **Sizes of C data types in x86-64.** With a 64-bit machine, pointers are 8 bytes long.

Addressing Modes

- General form
  - *displacement*(basereg, idxreg, scale) → Mem[basereg + scale*idxreg + displacement]
  - *displacement* is signed value up to 32 bits
  - scale = 1, 2, 4, or 8
- Examples

Expression	**Meaning**
%rax	Value of %rax
(%rax)	Content of Mem[%rax]
0x18(%rax)	Content of Mem[%rax + 0x18]
(%rax, %rbx)	Content of Mem[%rax + %rbx]
(%rax, %rbx, 4)	Content of Mem[%rax + 4*%rbx]
0x40(%rax, %rbx, 8)	Content of Mem[%rax + 8*%rbx + 0x40]

Arithmetic and Movement Operations

Instruction	Effect	Instruction	Effect
add (%rdx), $r8	r8 += memory[rdx]	shl $2, %r8	r8 <<= 2
mul $3, %r8	r8 *= 3	inc %r8	r8++
sub $1, $r8	r8--	neg %r8	r8 = -r8
mov %rbx, %rdx	rdx = rbx	imul %rbx, %rdx	rdx *= rbx
movslq %ebx, %rdx	rdx = ebx (sign-extend)	and $0x7f, %rdx	rdx &= 0x7f
lea (%rax, %rbx, 2), %rdx	rdx = rax + rbx*2	or $1, %r8	r8 |= 0x01
shr %cl, %r8 (logical)	r8 >>= cl (zero filled)	xor %rax, %rdx	rdx ^= rax
sar $3, %r8 (arithmetic)	r8 >>= 3 (sign copied)	not %rdx	rdx = ~rdx

- movslq: **MOV**e with **S**ign-extension, from **L**ong to **Q**uad
- lea: **L**oad **E**ffective **A**ddress (load effective address, mov: load value)

Control Codes

- Comparison & Test Instructions (textbook p.202)
  - Result determines next conditional jump instruction
  - **“cmp b, a” computes (a-b), “test b, a” computes (a&b)**
- Jump Instructions (textbook p.206)

```
test %eax, %eax
----------------------
If (eax == 0 )
        ZF =1;
else
        ZF = 0;
```

Instruction	Effect	Instruction	Effect
jmp	Always jump	ja	Jump if above (unsigned >)
je / jz	Jump if equal / zero (ZF)	jae	Jump if above / equal
jne / jnz	Jump if !equal / !zero (ZF)	jb	Jump if below (unsigned <)
jg	Jump if greater (signed >)	jbe	Jump if below / equal
jge	Jump if greater / equal	js	Jump if sign bit is 1 (negative)
jl	Jump if less (signed <)	jns	Jump if sign bit is 0 (non-negative)
jle	Jump if less / equal		

Comparison and Jump Instructions

- Example #1

```
cmp $0x15213, %r12
jge   0xdeadbeef
```

→ if %r12 >= 0x15213,
jump to the address of 0xdeadbeef

- Example #2

```
cmp %rax, %rdi
jae   0x15213b
```

→ if the unsigned value of %rdi is at or above the unsigned value of %rax,
jump to the address of 0x15213b

- Example #3

```
test %r8, %r8
jnz   0x15213
jmp *(%rsi)
```

→ if %r8 & %r8 is nonzero,
jump to the address of 0x15213,
otherwise, jump to the address stored in memory location of %rsi

Bomb Lab Overview

- Every bomb has a **unique** number and a **different set of problems**.

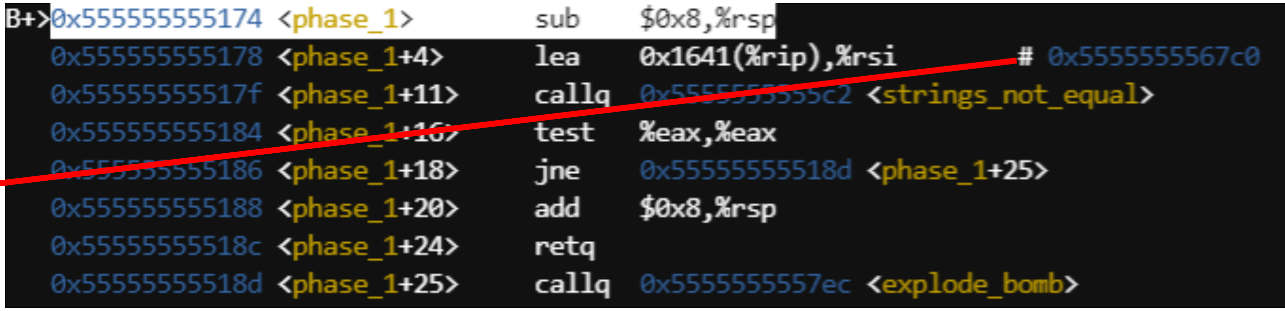
- Bomb lab has 6 phases (+1 secret phase for extra credits)
  - Inputting the right string moves you to the next phase
  - Make the solution (solution.txt) that includes your 6 answers in 6 lines
  - Make sure that you add an extra empty line after your solution strings in the solution.txt file
    - 6answers in 6lines + 1 empty line

```
This is answer.
1 2 3 4 5 6
1  111
3 111
12m123
4 6 1 2 5 3
<1 empty line>
```

Phase 1

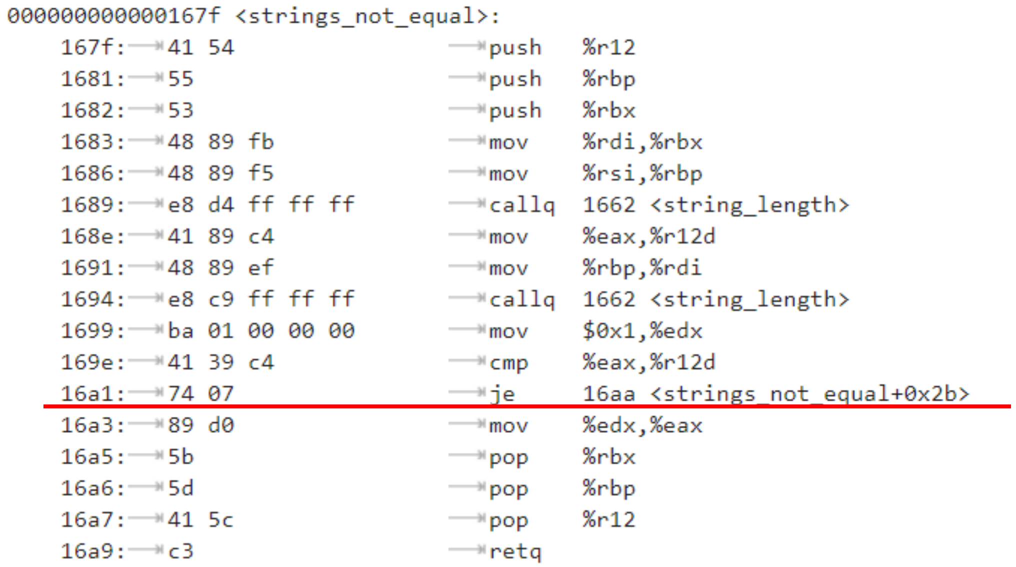
- **Find strings**

- Check registers and memory addresses
  - x/s $rsi : string
  - x/s x/s 0x5555555567c0



```
B+>0x555555555174 <phase_1>        sub    $0x8,%rsp
   0x555555555178 <phase_1+4>      lea    0x1641(%rip),%rsi        # 0x5555555567c0
   0x55555555517f <phase_1+11>     callq  0x5555555555c2 <strings_not_equal>
   0x555555555184 <phase_1+16>     test   %eax,%eax
   0x555555555186 <phase_1+18>     jne    0x55555555518d <phase_1+25>
   0x555555555188 <phase_1+20>     add    $0x8,%rsp
   0x55555555518c <phase_1+24>     retq
   0x55555555518d <phase_1+25>     callq  0x5555555557ec <explode_bomb>
```

- Go inside the <strings_not_equal> function
  - Check registers
    - x/s $rdi
    - x/s $rsi



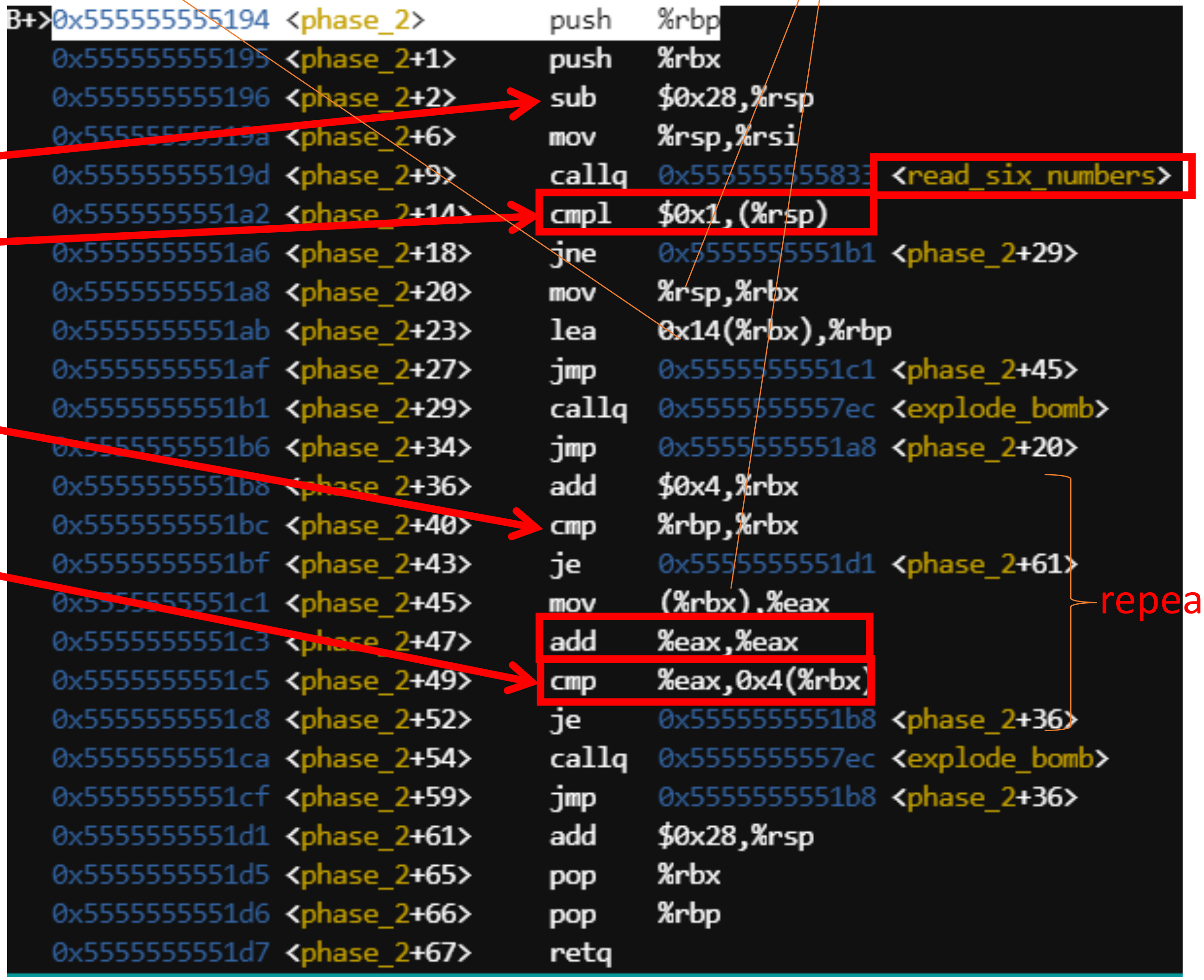
```
000000000000167f <strings_not_equal>:
    167f:   41 54                   push   %r12
    1681:   55                      push   %rbp
    1682:   53                      push   %rbx
    1683:   48 89 fb                mov    %rdi,%rbx
    1686:   48 89 f5                mov    %rsi,%rbp
    1689:   e8 d4 ff ff ff          callq  1662 <string_length>
    168e:   41 89 c4                mov    %eax,%r12d
    1691:   48 89 ef                mov    %rbp,%rdi
    1694:   e8 c9 ff ff ff          callq  1662 <string_length>
    1699:   ba 01 00 00 00          mov    $0x1,%edx
    169e:   41 39 c4                cmp    %eax,%r12d
    16a1:   74 07                   je     16aa <strings_not_equal+0x2b>
    16a3:   89 d0                   mov    %edx,%eax
    16a5:   5b                      pop    %rbx
    16a6:   5d                      pop    %rbp
    16a7:   41 5c                   pop    %r12
    16a9:   c3                      retq
```

Phase 2

- **Find 6 numbers**
  - **Read six numbers**
    - Make spaces in stack
    - First comparison
    - Compare the last number’s address with the current number’s address
    - Other comparisons
- Check registers
  - x/w $rsp : input number
  - x/6w $rsp : all input numbers
  - p $eax : after addition
  - x $rbx

0x14 = 20_{10} (4*5)
6th value of your input
($rsp) : 1st value

$rbx = $rsp: rbx has a pointer to 6 numbers
$eax = $rbx: eax has a pointer to 6 numbers

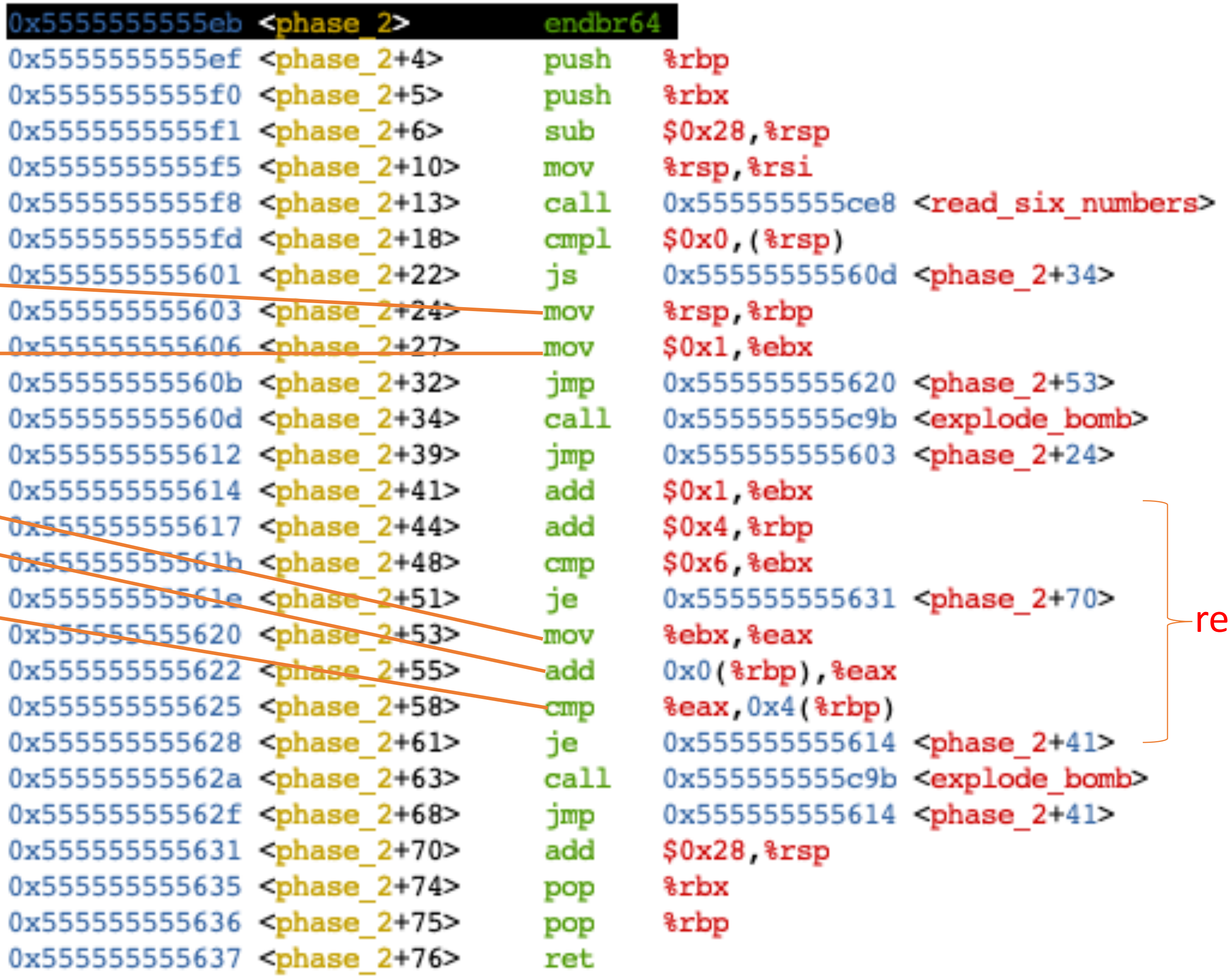


```
B+>0x555555555194 <phase_2>        push   %rbp
   0x555555555195 <phase_2+1>      push   %rbx
   0x555555555196 <phase_2+2>      sub    $0x28,%rsp
   0x55555555519a <phase_2+6>      mov    %rsp,%rsi
   0x55555555519d <phase_2+9>      callq  0x555555555833 <read_six_numbers>
   0x5555555551a2 <phase_2+14>     cmpl   $0x1,(%rsp)
   0x5555555551a6 <phase_2+18>     jne    0x5555555551b1 <phase_2+29>
   0x5555555551a8 <phase_2+20>     mov    %rsp,%rbx
   0x5555555551ab <phase_2+23>     lea    0x14(%rbx),%rbp
   0x5555555551af <phase_2+27>     jmp    0x5555555551c1 <phase_2+45>
   0x5555555551b1 <phase_2+29>     callq  0x5555555557ec <explode_bomb>
   0x5555555551b6 <phase_2+34>     jmp    0x5555555551a8 <phase_2+20>
   0x5555555551b8 <phase_2+36>     add    $0x4,%rbx
   0x5555555551bc <phase_2+40>     cmp    %rbp,%rbx
   0x5555555551bf <phase_2+43>     je     0x5555555551d1 <phase_2+61>
   0x5555555551c1 <phase_2+45>     mov    (%rbx),%eax
   0x5555555551c3 <phase_2+47>     add    %eax,%eax
   0x5555555551c5 <phase_2+49>     cmp    %eax,0x4(%rbx)
   0x5555555551c8 <phase_2+52>     je     0x5555555551b8 <phase_2+36>
   0x5555555551ca <phase_2+54>     callq  0x5555555557ec <explode_bomb>
   0x5555555551cf <phase_2+59>     jmp    0x5555555551b8 <phase_2+36>
   0x5555555551d1 <phase_2+61>     add    $0x28,%rsp
   0x5555555551d5 <phase_2+65>     pop    %rbx
   0x5555555551d6 <phase_2+66>     pop    %rbp
   0x5555555551d7 <phase_2+67>     retq
```

repeat

Phase 2

- %rsp: input numbers
- %rbp: input number
- %ebx: counter
- %eax 🡐%ebx
- %eax + %rbp(input number)
- Compare with input number
- Ex)
  - 2nd value = 1st value + counter



```
0x5555555555eb <phase_2>        endbr64
0x5555555555ef <phase_2+4>      push   %rbp
0x5555555555f0 <phase_2+5>      push   %rbx
0x5555555555f1 <phase_2+6>      sub    $0x28,%rsp
0x5555555555f5 <phase_2+10>     mov    %rsp,%rsi
0x5555555555f8 <phase_2+13>     call   0x555555555ce8 <read_six_numbers>
0x5555555555fd <phase_2+18>     cmpl   $0x0,(%rsp)
0x555555555601 <phase_2+22>     js     0x55555555560d <phase_2+34>
0x555555555603 <phase_2+24>     mov    %rsp,%rbp
0x555555555606 <phase_2+27>     mov    $0x1,%ebx
0x55555555560b <phase_2+32>     jmp    0x555555555620 <phase_2+53>
0x55555555560d <phase_2+34>     call   0x555555555c9b <explode_bomb>
0x555555555612 <phase_2+39>     jmp    0x555555555603 <phase_2+24>
0x555555555614 <phase_2+41>     add    $0x1,%ebx
0x555555555617 <phase_2+44>     add    $0x4,%rbp
0x55555555561b <phase_2+48>     cmp    $0x6,%ebx
0x55555555561e <phase_2+51>     je     0x555555555631 <phase_2+70>
0x555555555620 <phase_2+53>     mov    %ebx,%eax
0x555555555622 <phase_2+55>     add    0x0(%rbp),%eax
0x555555555625 <phase_2+58>     cmp    %eax,0x4(%rbp)
0x555555555628 <phase_2+61>     je     0x555555555614 <phase_2+41>
0x55555555562a <phase_2+63>     call   0x555555555c9b <explode_bomb>
0x55555555562f <phase_2+68>     jmp    0x555555555614 <phase_2+41>
0x555555555631 <phase_2+70>     add    $0x28,%rsp
0x555555555635 <phase_2+74>     pop    %rbx
0x555555555636 <phase_2+75>     pop    %rbp
0x555555555637 <phase_2+76>     ret
```

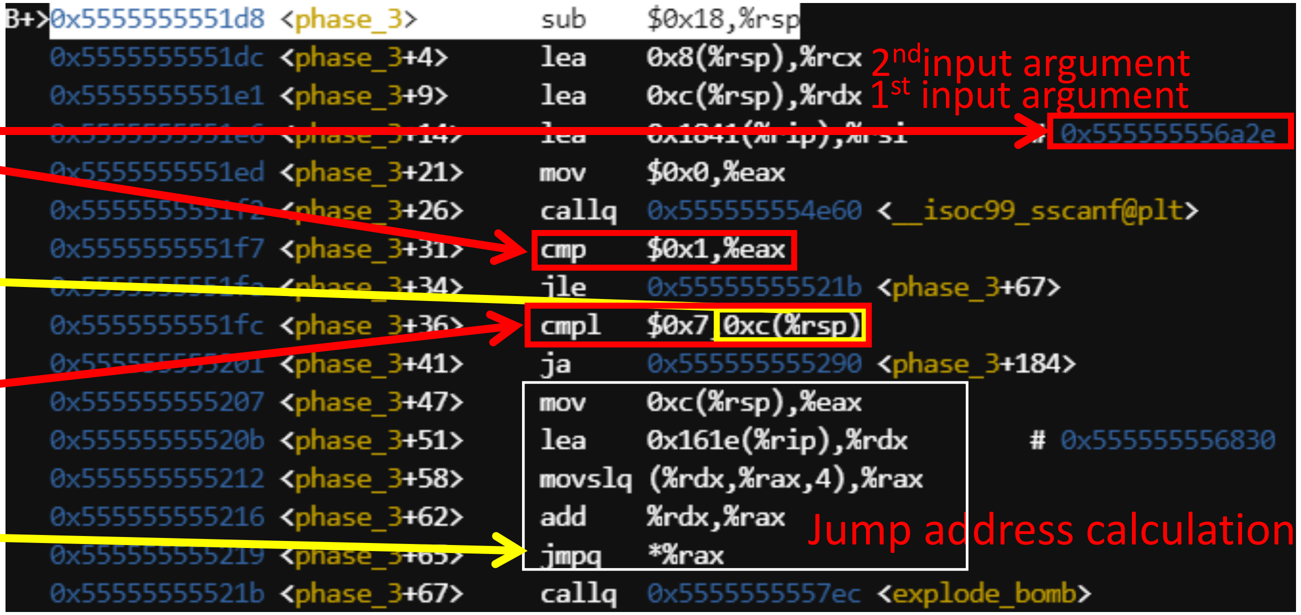
repeat

js: jump if sign bit is 1 (negative)

Phase 3: switch statement

- Input arguments: 2
  - x/s 0x555555556a2e -> “%d %d”
- x/s $rsp+0xc
  - Check if the number of 0xc($rsp) is the first input or the second input
- The input number must not above 7.
  - **Switch statement** has 8 cases
  - **Jump address: x/s $rax**
  - Check 0x7(%rsp), 0xc(%rsp), 0x8(%rsp)
    - cmpl $0x7, 0xc($rsp) : 1st input comparison
    - cmpl $0x257, 0x8($rsp) : 2nd input comparison
    - cmpl $0x106, 0x8($rsp)
    - ⋮
    - cmp %al, 0x7($rsp)

scanf function return value => number of inputs

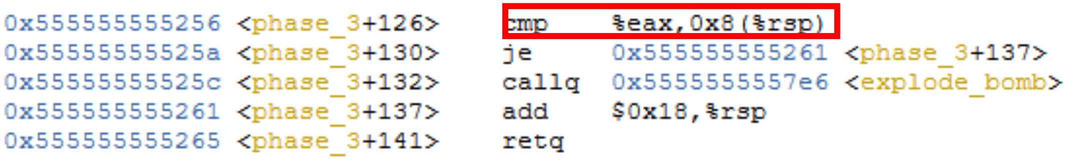


```
B+>0x5555555551d8 <phase_3>        sub    $0x18,%rsp
   0x5555555551dc <phase_3+4>      lea    0x8(%rsp),%rcx
   0x5555555551e1 <phase_3+9>      lea    0xc(%rsp),%rdx
   0x5555555551e6 <phase_3+14>     lea    0x1841(%rip),%rsi        # 0x555555556a2e
   0x5555555551ed <phase_3+21>     mov    $0x0,%eax
   0x5555555551f2 <phase_3+26>     callq  0x555555554e60 <__isoc99_sscanf@plt>
   0x5555555551f7 <phase_3+31>     cmp    $0x1,%eax
   0x5555555551fa <phase_3+34>     jle    0x55555555521b <phase_3+67>
   0x5555555551fc <phase_3+36>     cmpl   $0x7,0xc(%rsp)
   0x555555555201 <phase_3+41>     ja     0x555555555290 <phase_3+184>
   0x555555555207 <phase_3+47>     mov    0xc(%rsp),%eax
   0x55555555520b <phase_3+51>     lea    0x161e(%rip),%rdx        # 0x555555556830
   0x555555555212 <phase_3+58>     movslq (%rdx,%rax,4),%rax
   0x555555555216 <phase_3+62>     add    %rdx,%rax
   0x555555555219 <phase_3+65>     jmpq   *%rax
   0x55555555521b <phase_3+67>     callq  0x5555555557ec <explode_bomb>
```

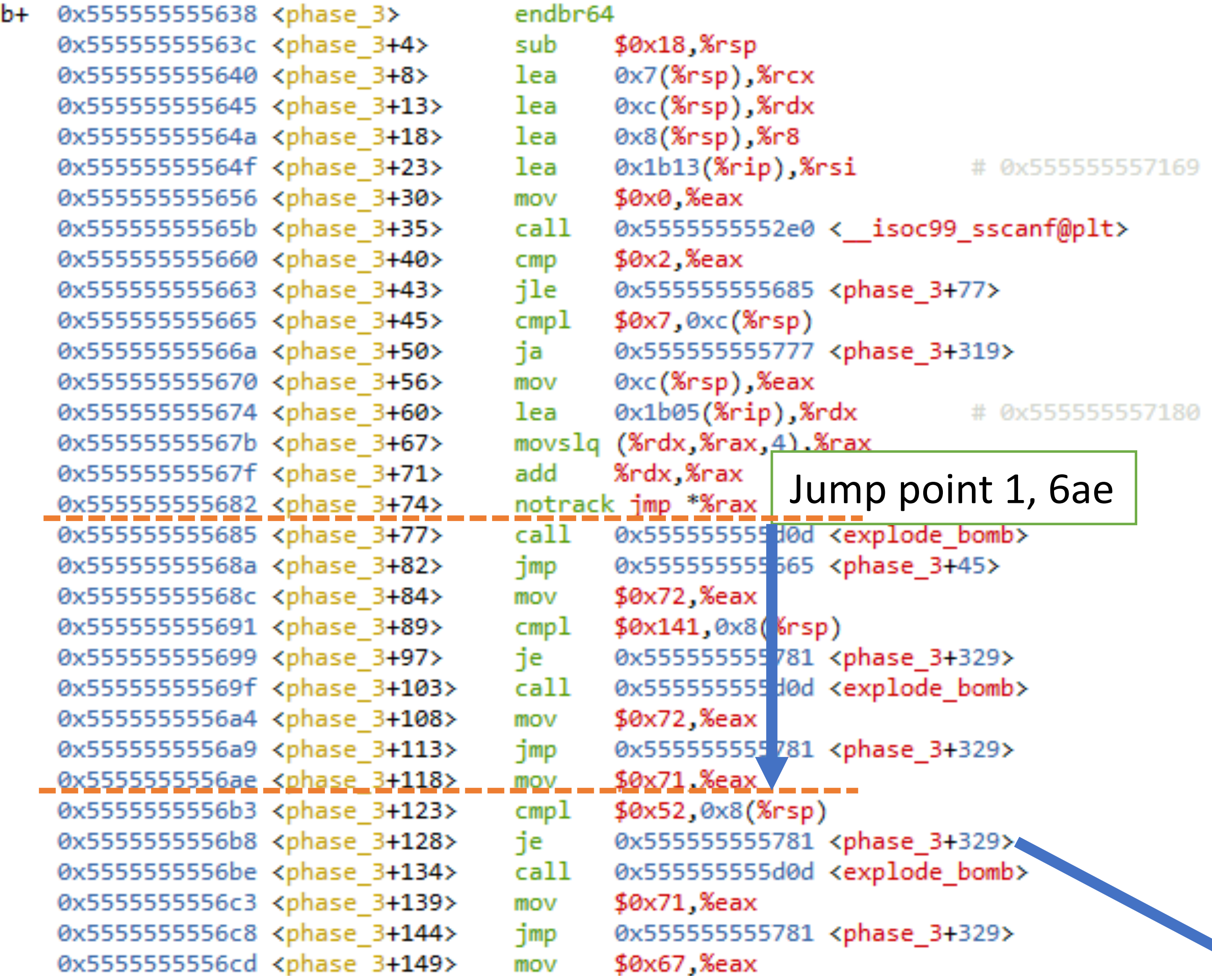
2nd input argument

1st input argument

Jump address calculation

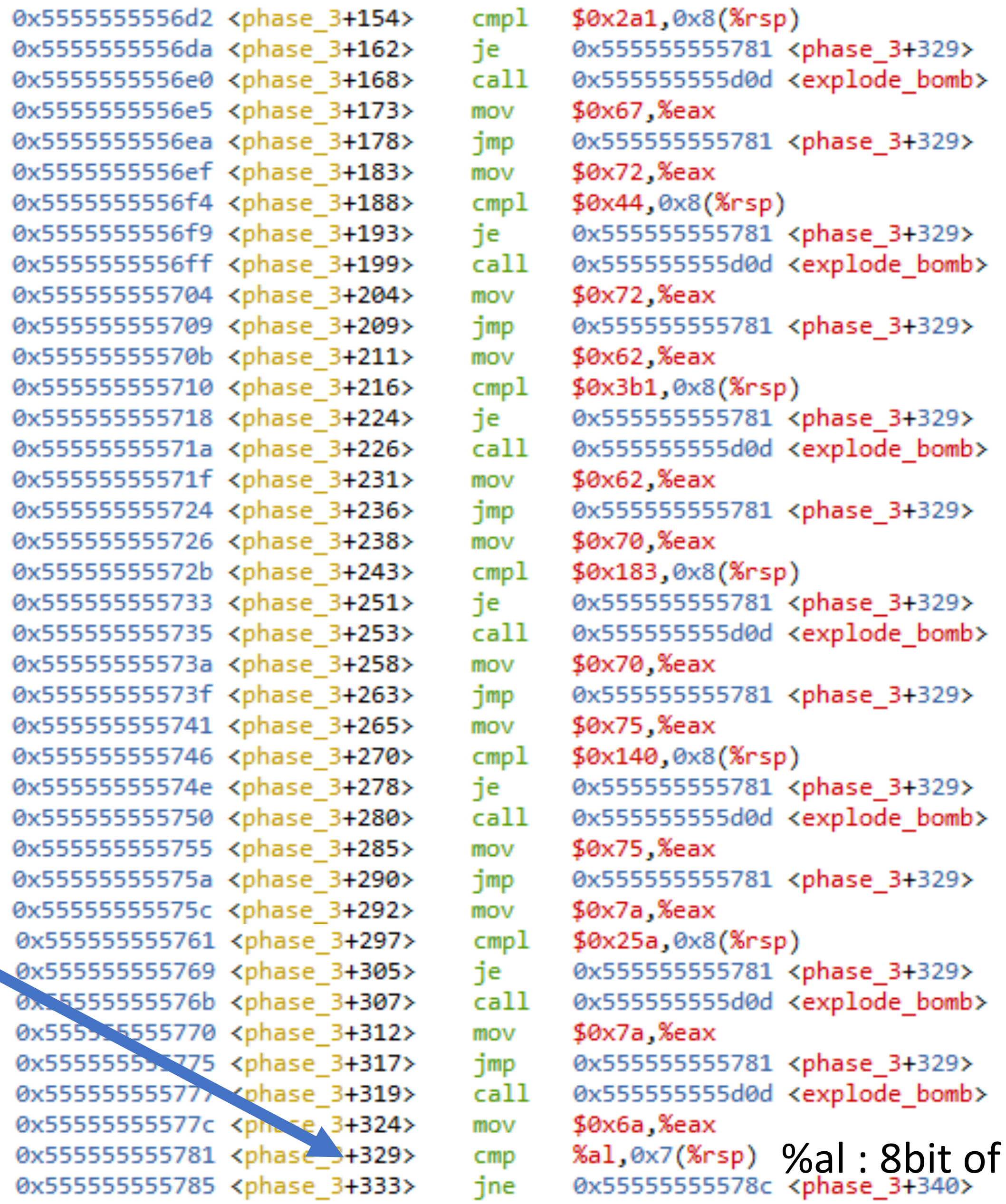


```
0x555555555256 <phase_3+126>     cmp    %eax,0x8(%rsp)
0x55555555525a <phase_3+130>     je     0x555555555261 <phase_3+137>
0x55555555525c <phase_3+132>     callq  0x5555555557e6 <explode_bomb>
0x555555555261 <phase_3+137>     add    $0x18,%rsp
0x555555555265 <phase_3+141>     retq
```



```
b+  0x555555555638 <phase_3>       endbr64
    0x55555555563c <phase_3+4>     sub    $0x18,%rsp
    0x555555555640 <phase_3+8>     lea    0x7(%rsp),%rcx
    0x555555555645 <phase_3+13>    lea    0xc(%rsp),%rdx
    0x55555555564a <phase_3+18>    lea    0x8(%rsp),%r8
    0x55555555564f <phase_3+23>    lea    0x1b13(%rip),%rsi        # 0x555555557169
    0x555555555656 <phase_3+30>    mov    $0x0,%eax
    0x55555555565b <phase_3+35>    call   0x5555555552e0 <__isoc99_sscanf@plt>
    0x555555555660 <phase_3+40>    cmp    $0x2,%eax
    0x555555555663 <phase_3+43>    jle    0x555555555685 <phase_3+77>
    0x555555555665 <phase_3+45>    cmpl   $0x7,0xc(%rsp)
    0x55555555566a <phase_3+50>    ja     0x555555555777 <phase_3+319>
    0x555555555670 <phase_3+56>    mov    0xc(%rsp),%eax
    0x555555555674 <phase_3+60>    lea    0x1b05(%rip),%rdx        # 0x555555557180
    0x55555555567b <phase_3+67>    movslq (%rdx,%rax,4),%rax
    0x55555555567f <phase_3+71>    add    %rdx,%rax
    0x555555555682 <phase_3+74>    notrack jmp *%rax
    0x555555555685 <phase_3+77>    call   0x555555555d0d <explode_bomb>
    0x55555555568a <phase_3+82>    jmp    0x555555555665 <phase_3+45>
    0x55555555568c <phase_3+84>    mov    $0x72,%eax
    0x555555555691 <phase_3+89>    cmpl   $0x141,0x8(%rsp)
    0x555555555699 <phase_3+97>    je     0x555555555781 <phase_3+329>
    0x55555555569f <phase_3+103>   call   0x555555555d0d <explode_bomb>
    0x5555555556a4 <phase_3+108>   mov    $0x72,%eax
    0x5555555556a9 <phase_3+113>   jmp    0x555555555781 <phase_3+329>
    0x5555555556ae <phase_3+118>   mov    $0x71,%eax
    0x5555555556b3 <phase_3+123>   cmpl   $0x52,0x8(%rsp)
    0x5555555556b8 <phase_3+128>   je     0x555555555781 <phase_3+329>
    0x5555555556be <phase_3+134>   call   0x555555555d0d <explode_bomb>
    0x5555555556c3 <phase_3+139>   mov    $0x71,%eax
    0x5555555556c8 <phase_3+144>   jmp    0x555555555781 <phase_3+329>
    0x5555555556cd <phase_3+149>   mov    $0x67,%eax
```

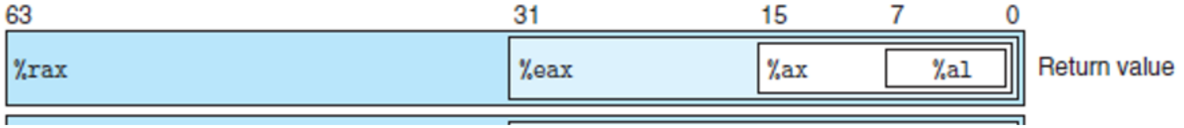
Jump point 1, 6ae



```
0x5555555556d2 <phase_3+154>   cmpl   $0x2a1,0x8(%rsp)
0x5555555556da <phase_3+162>   je     0x555555555781 <phase_3+329>
0x5555555556e0 <phase_3+168>   call   0x555555555d0d <explode_bomb>
0x5555555556e5 <phase_3+173>   mov    $0x67,%eax
0x5555555556ea <phase_3+178>   jmp    0x555555555781 <phase_3+329>
0x5555555556ef <phase_3+183>   mov    $0x72,%eax
0x5555555556f4 <phase_3+188>   cmpl   $0x44,0x8(%rsp)
0x5555555556f9 <phase_3+193>   je     0x555555555781 <phase_3+329>
0x5555555556ff <phase_3+199>   call   0x555555555d0d <explode_bomb>
0x555555555704 <phase_3+204>   mov    $0x72,%eax
0x555555555709 <phase_3+209>   jmp    0x555555555781 <phase_3+329>
0x55555555570b <phase_3+211>   mov    $0x62,%eax
0x555555555710 <phase_3+216>   cmpl   $0x3b1,0x8(%rsp)
0x555555555718 <phase_3+224>   je     0x555555555781 <phase_3+329>
0x55555555571a <phase_3+226>   call   0x555555555d0d <explode_bomb>
0x55555555571f <phase_3+231>   mov    $0x62,%eax
0x555555555724 <phase_3+236>   jmp    0x555555555781 <phase_3+329>
0x555555555726 <phase_3+238>   mov    $0x70,%eax
0x55555555572b <phase_3+243>   cmpl   $0x183,0x8(%rsp)
0x555555555733 <phase_3+251>   je     0x555555555781 <phase_3+329>
0x555555555735 <phase_3+253>   call   0x555555555d0d <explode_bomb>
0x55555555573a <phase_3+258>   mov    $0x70,%eax
0x55555555573f <phase_3+263>   jmp    0x555555555781 <phase_3+329>
0x555555555741 <phase_3+265>   mov    $0x75,%eax
0x555555555746 <phase_3+270>   cmpl   $0x140,0x8(%rsp)
0x55555555574e <phase_3+278>   je     0x555555555781 <phase_3+329>
0x555555555750 <phase_3+280>   call   0x555555555d0d <explode_bomb>
0x555555555755 <phase_3+285>   mov    $0x75,%eax
0x55555555575a <phase_3+290>   jmp    0x555555555781 <phase_3+329>
0x55555555575c <phase_3+292>   mov    $0x7a,%eax
0x555555555761 <phase_3+297>   cmpl   $0x25a,0x8(%rsp)
0x555555555769 <phase_3+305>   je     0x555555555781 <phase_3+329>
0x55555555576b <phase_3+307>   call   0x555555555d0d <explode_bomb>
0x555555555770 <phase_3+312>   mov    $0x7a,%eax
0x555555555775 <phase_3+317>   jmp    0x555555555781 <phase_3+329>
0x555555555777 <phase_3+319>   call   0x555555555d0d <explode_bomb>
0x55555555577c <phase_3+324>   mov    $0x6a,%eax
0x555555555781 <phase_3+329>   cmp    %al,0x7(%rsp)
0x555555555785 <phase_3+333>   jne    0x55555555578c <phase_3+340>
```

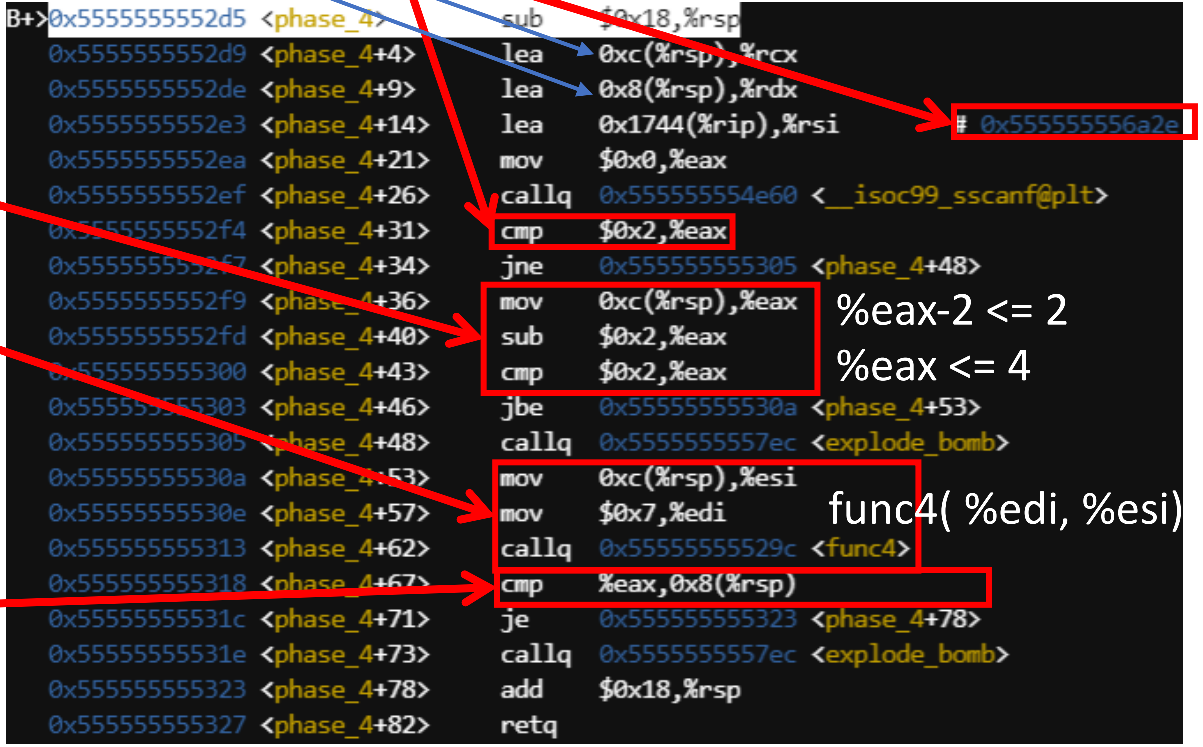
%al : 8bit of %rax

(gdb) x/s $rsp + 0x8 : “R” 🡨 82 decimal(0x52)
(gdb) x/s $rsp + 0xc: “\001” 🡨 1st input
(gdb) x/s $rsp + 0x7: “q” 🡨 2nd input r, 0x71



Phase 4: Recursive function

- **Binary search** or **Variant Fibonacci sequence**
- Input arguments: 2 (x/s 0x555555556a23 -> “%d %d”)
- 2nd input value ≤ 4
- Type stepi (si)
  - Go into func4
- Check input values
  - x/s $rsp + 0x8
  - x/s $rsp + 0xc
- Recursive output
  - p $eax



```
B+>0x5555555552d5 <phase_4>       sub    $0x18,%rsp
   0x5555555552d9 <phase_4+4>     lea    0xc(%rsp),%rcx
   0x5555555552de <phase_4+9>     lea    0x8(%rsp),%rdx
   0x5555555552e3 <phase_4+14>    lea    0x1744(%rip),%rsi        # 0x555555556a2e
   0x5555555552ea <phase_4+21>    mov    $0x0,%eax
   0x5555555552ef <phase_4+26>    callq  0x555555554e60 <__isoc99_sscanf@plt>
   0x5555555552f4 <phase_4+31>    cmp    $0x2,%eax
   0x5555555552f7 <phase_4+34>    jne    0x555555555305 <phase_4+48>
   0x5555555552f9 <phase_4+36>    mov    0xc(%rsp),%eax
   0x5555555552fd <phase_4+40>    sub    $0x2,%eax
   0x555555555300 <phase_4+43>    cmp    $0x2,%eax
   0x555555555303 <phase_4+46>    jbe    0x55555555530a <phase_4+53>
   0x555555555305 <phase_4+48>    callq  0x5555555557ec <explode_bomb>
   0x55555555530a <phase_4+53>    mov    0xc(%rsp),%esi
   0x55555555530e <phase_4+57>    mov    $0x7,%edi
   0x555555555313 <phase_4+62>    callq  0x55555555529c <func4>
   0x555555555318 <phase_4+67>    cmp    %eax,0x8(%rsp)
   0x55555555531c <phase_4+71>    je     0x555555555323 <phase_4+78>
   0x55555555531e <phase_4+73>    callq  0x5555555557ec <explode_bomb>
   0x555555555323 <phase_4+78>    add    $0x18,%rsp
   0x555555555327 <phase_4+82>    retq
```

%eax-2 <= 2

%eax <= 4

func4(%edi, %esi)

func4(your input, 7)

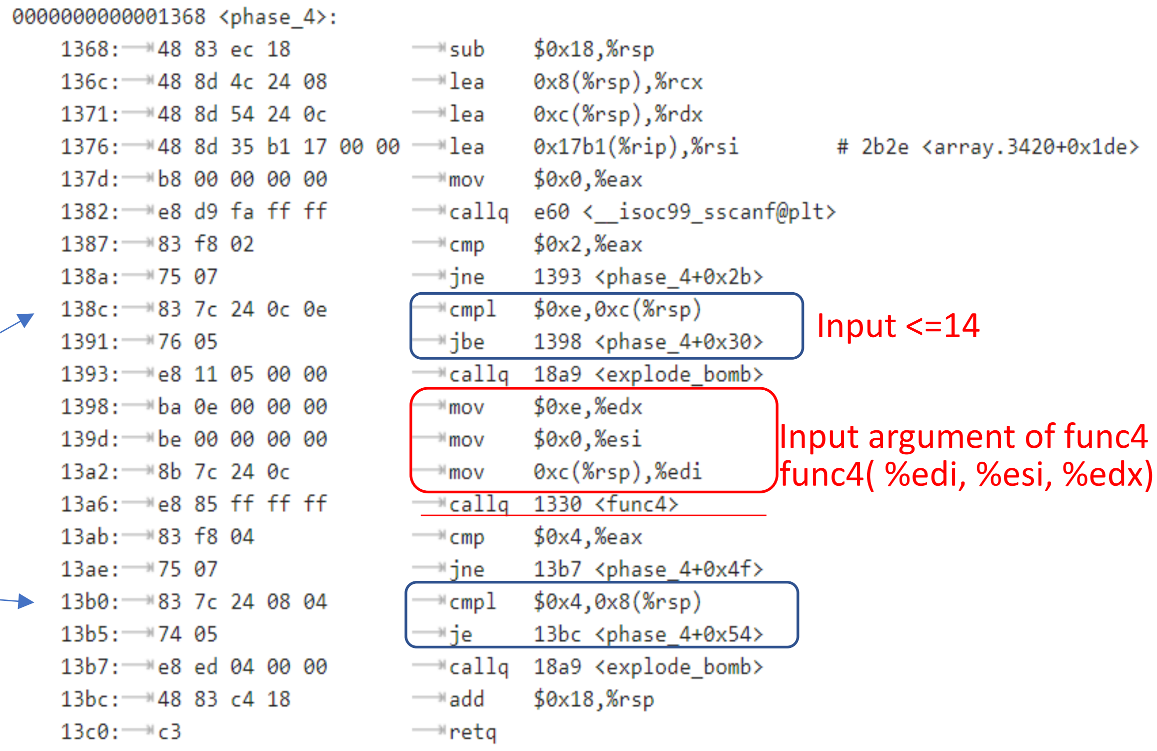
Phase 4: Recursive function

- Recursive function
  - Find appropriate type of input values
  - Check 0x8(%rsp), 0xc(%rsp)
    - ◦ cmpl $0xe, 0xc($rsp) // at 138c line
    - ◦ cmpl $0x4, 0x8($rsp) // at 13b0 line

- Binary search or Variant Fibonacci sequence
  - Figure out the function of <func4>
    - ◦ edi, esi, edx values
    - ◦ Recursive calling
    - ◦ Return line and value

Do not brute force approach

```
If($ecx  > $edi )
     call func4…
else if ($ecx  < $edi )
     call func4…
else
      return 0
```

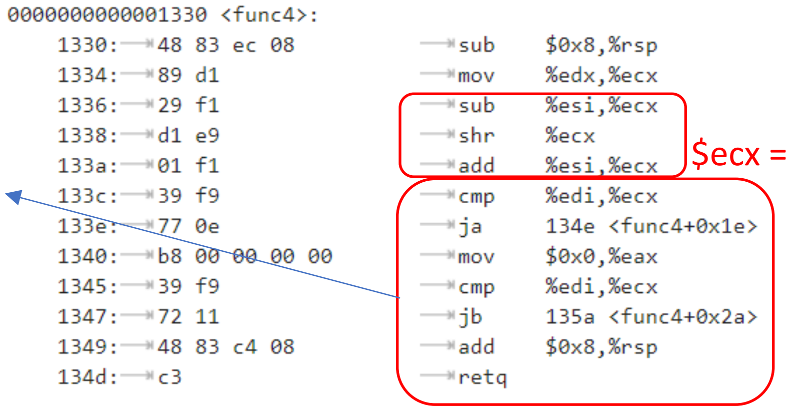


```
0000000000001368 <phase_4>:
    1368:	48 83 ec 18          	sub    $0x18,%rsp
    136c:	48 8d 4c 24 08       	lea    0x8(%rsp),%rcx
    1371:	48 8d 54 24 0c       	lea    0xc(%rsp),%rdx
    1376:	48 8d 35 b1 17 00 00 	lea    0x17b1(%rip),%rsi        # 2b2e <array.3420+0x1de>
    137d:	b8 00 00 00 00       	mov    $0x0,%eax
    1382:	e8 d9 fa ff ff       	callq  e60 <__isoc99_sscanf@plt>
    1387:	83 f8 02             	cmp    $0x2,%eax
    138a:	75 07                	jne    1393 <phase_4+0x2b>
    138c:	83 7c 24 0c 0e       	cmpl   $0xe,0xc(%rsp)
    1391:	76 05                	jbe    1398 <phase_4+0x30>
    1393:	e8 11 05 00 00       	callq  18a9 <explode_bomb>
    1398:	ba 0e 00 00 00       	mov    $0xe,%edx
    139d:	be 00 00 00 00       	mov    $0x0,%esi
    13a2:	8b 7c 24 0c          	mov    0xc(%rsp),%edi
    13a6:	e8 85 ff ff ff       	callq  1330 <func4>
    13ab:	83 f8 04             	cmp    $0x4,%eax
    13ae:	75 07                	jne    13b7 <phase_4+0x4f>
    13b0:	83 7c 24 08 04       	cmpl   $0x4,0x8(%rsp)
    13b5:	74 05                	je     13bc <phase_4+0x54>
    13b7:	e8 ed 04 00 00       	callq  18a9 <explode_bomb>
    13bc:	48 83 c4 18          	add    $0x18,%rsp
    13c0:	c3                   	retq
```

Input <=14

Input argument of func4
func4(%edi, %esi, %edx)

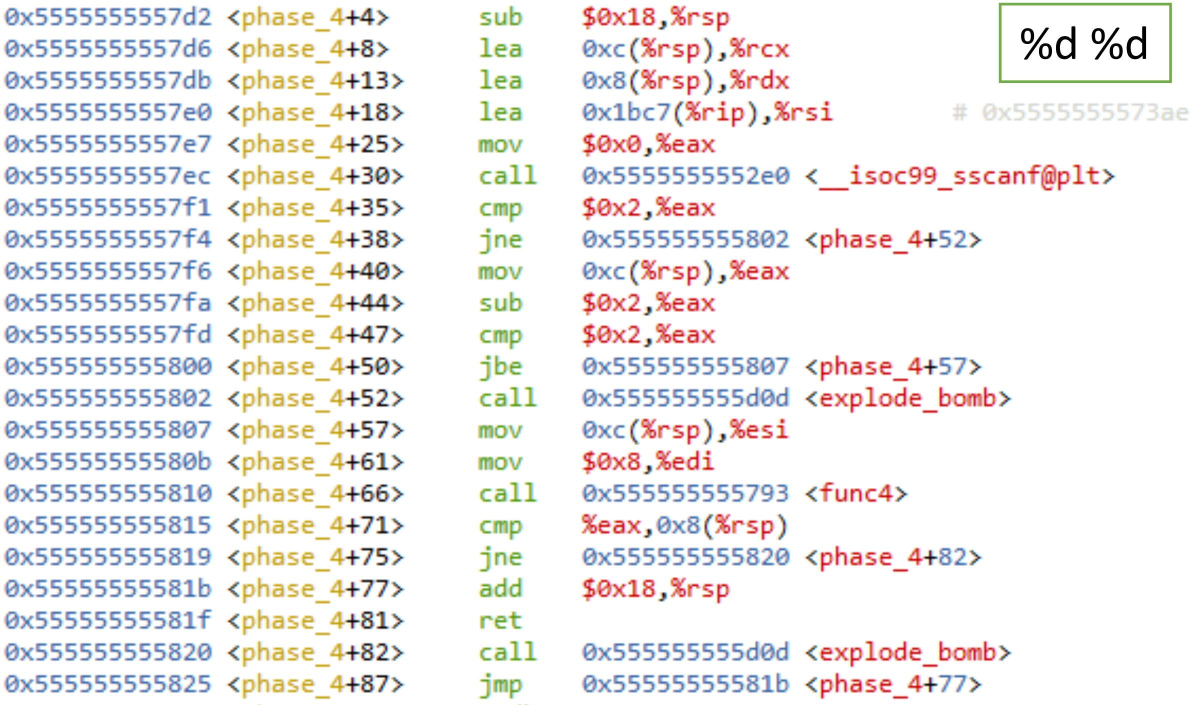
func4(1st input($edi), 0($esi), 14($edx))



```
0000000000001330 <func4>:
    1330:	48 83 ec 08          	sub    $0x8,%rsp
    1334:	89 d1                	mov    %edx,%ecx
    1336:	29 f1                	sub    %esi,%ecx
    1338:	d1 e9                	shr    %ecx
    133a:	01 f1                	add    %esi,%ecx
    133c:	39 f9                	cmp    %edi,%ecx
    133e:	77 0e                	ja     134e <func4+0x1e>
    1340:	b8 00 00 00 00       	mov    $0x0,%eax
    1345:	39 f9                	cmp    %edi,%ecx
    1347:	72 11                	jb     135a <func4+0x2a>
    1349:	48 83 c4 08          	add    $0x8,%rsp
    134d:	c3                   	retq
```

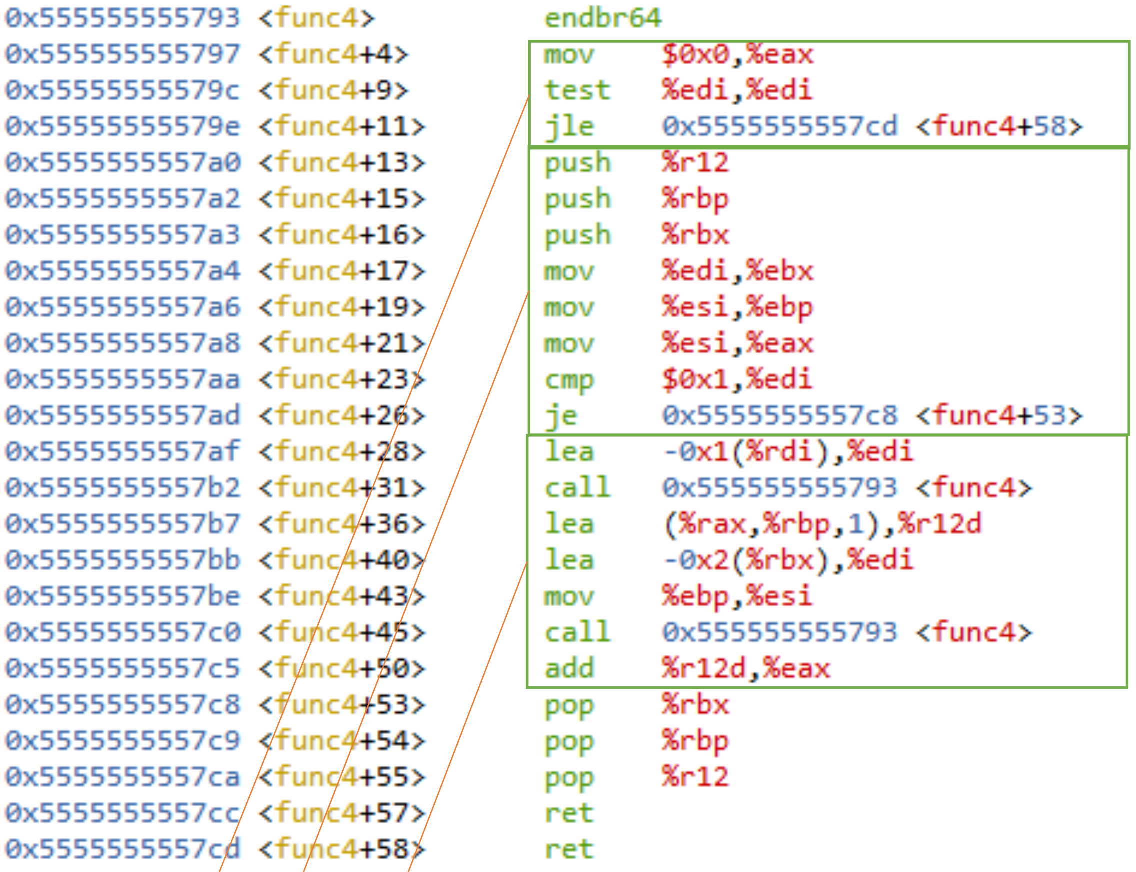
$ecx = $esi + ($ecx – $esi)/2

Phase 4: Recursive function



```
0x5555555557d2 <phase_4+4>     sub    $0x18,%rsp
0x5555555557d6 <phase_4+8>     lea    0xc(%rsp),%rcx
0x5555555557db <phase_4+13>    lea    0x8(%rsp),%rdx
0x5555555557e0 <phase_4+18>    lea    0x1bc7(%rip),%rsi        # 0x5555555573ae
0x5555555557e7 <phase_4+25>    mov    $0x0,%eax
0x5555555557ec <phase_4+30>    call   0x5555555552e0 <__isoc99_sscanf@plt>
0x5555555557f1 <phase_4+35>    cmp    $0x2,%eax
0x5555555557f4 <phase_4+38>    jne    0x555555555802 <phase_4+52>
0x5555555557f6 <phase_4+40>    mov    0xc(%rsp),%eax
0x5555555557fa <phase_4+44>    sub    $0x2,%eax
0x5555555557fd <phase_4+47>    cmp    $0x2,%eax
0x555555555800 <phase_4+50>    jbe    0x555555555807 <phase_4+57>
0x555555555802 <phase_4+52>    call   0x555555555d0d <explode_bomb>
0x555555555807 <phase_4+57>    mov    0xc(%rsp),%esi
0x55555555580b <phase_4+61>    mov    $0x8,%edi
0x555555555810 <phase_4+66>    call   0x555555555793 <func4>
0x555555555815 <phase_4+71>    cmp    %eax,0x8(%rsp)
0x555555555819 <phase_4+75>    jne    0x555555555820 <phase_4+82>
0x55555555581b <phase_4+77>    add    $0x18,%rsp
0x55555555581f <phase_4+81>    ret
0x555555555820 <phase_4+82>    call   0x555555555d0d <explode_bomb>
0x555555555825 <phase_4+87>    jmp    0x55555555581b <phase_4+77>
```

%d %d



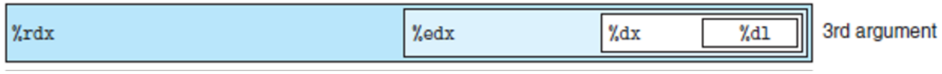
```
0x555555555793 <func4>        endbr64
0x555555555797 <func4+4>      mov    $0x0,%eax
0x55555555579c <func4+9>      test   %edi,%edi
0x55555555579e <func4+11>     jle    0x5555555557cd <func4+58>
0x5555555557a0 <func4+13>     push   %r12
0x5555555557a2 <func4+15>     push   %rbp
0x5555555557a3 <func4+16>     push   %rbx
0x5555555557a4 <func4+17>     mov    %edi,%ebx
0x5555555557a6 <func4+19>     mov    %esi,%ebp
0x5555555557a8 <func4+21>     mov    %esi,%eax
0x5555555557aa <func4+23>     cmp    $0x1,%edi
0x5555555557ad <func4+26>     je     0x5555555557c8 <func4+53>
0x5555555557af <func4+28>     lea    -0x1(%rdi),%edi
0x5555555557b2 <func4+31>     call   0x555555555793 <func4>
0x5555555557b7 <func4+36>     lea    (%rax,%rbp,1),%r12d
0x5555555557bb <func4+40>     lea    -0x2(%rbx),%edi
0x5555555557be <func4+43>     mov    %ebp,%esi
0x5555555557c0 <func4+45>     call   0x555555555793 <func4>
0x5555555557c5 <func4+50>     add    %r12d,%eax
0x5555555557c8 <func4+53>     pop    %rbx
0x5555555557c9 <func4+54>     pop    %rbp
0x5555555557ca <func4+55>     pop    %r12
0x5555555557cc <func4+57>     ret
0x5555555557cd <func4+58>     ret
```

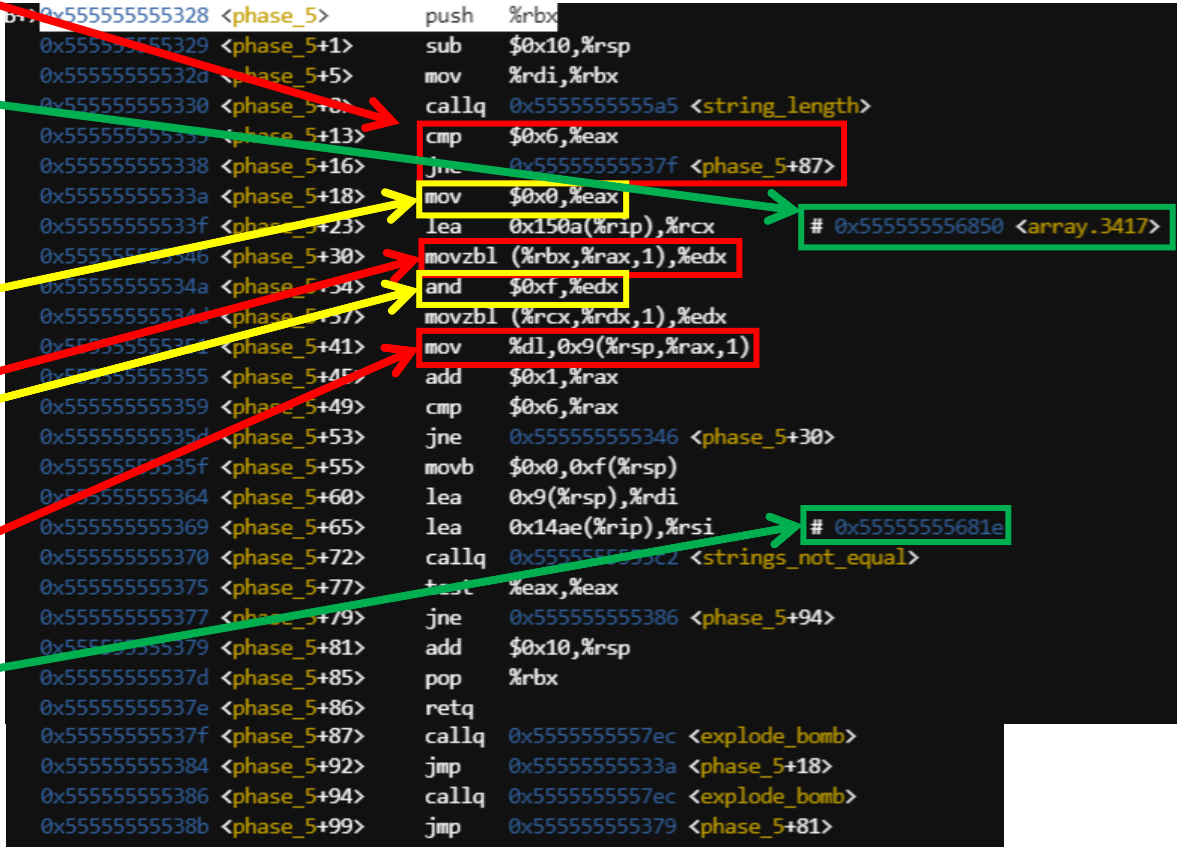
$rsp + 0x8 : 1st input
$rsp + 0xc : 2nd input

```
If(edi ==0){
   return 0;
}else if(edi == 1){
   return esi;
}else{
   esi + func4(rdi -1, esi) + func4(rdx -2, esi)
}
```

Phase 5: Array

- String length must be equal to 6.
- An array of 16 characters
  - matching from 0 to 15
  - $rcx takes the array
- $eax is the current index of array
- $edx takes the current character
- Extract the least 4 bits
  - 0xF & %edx
- Save the matched character
- Check the answer string



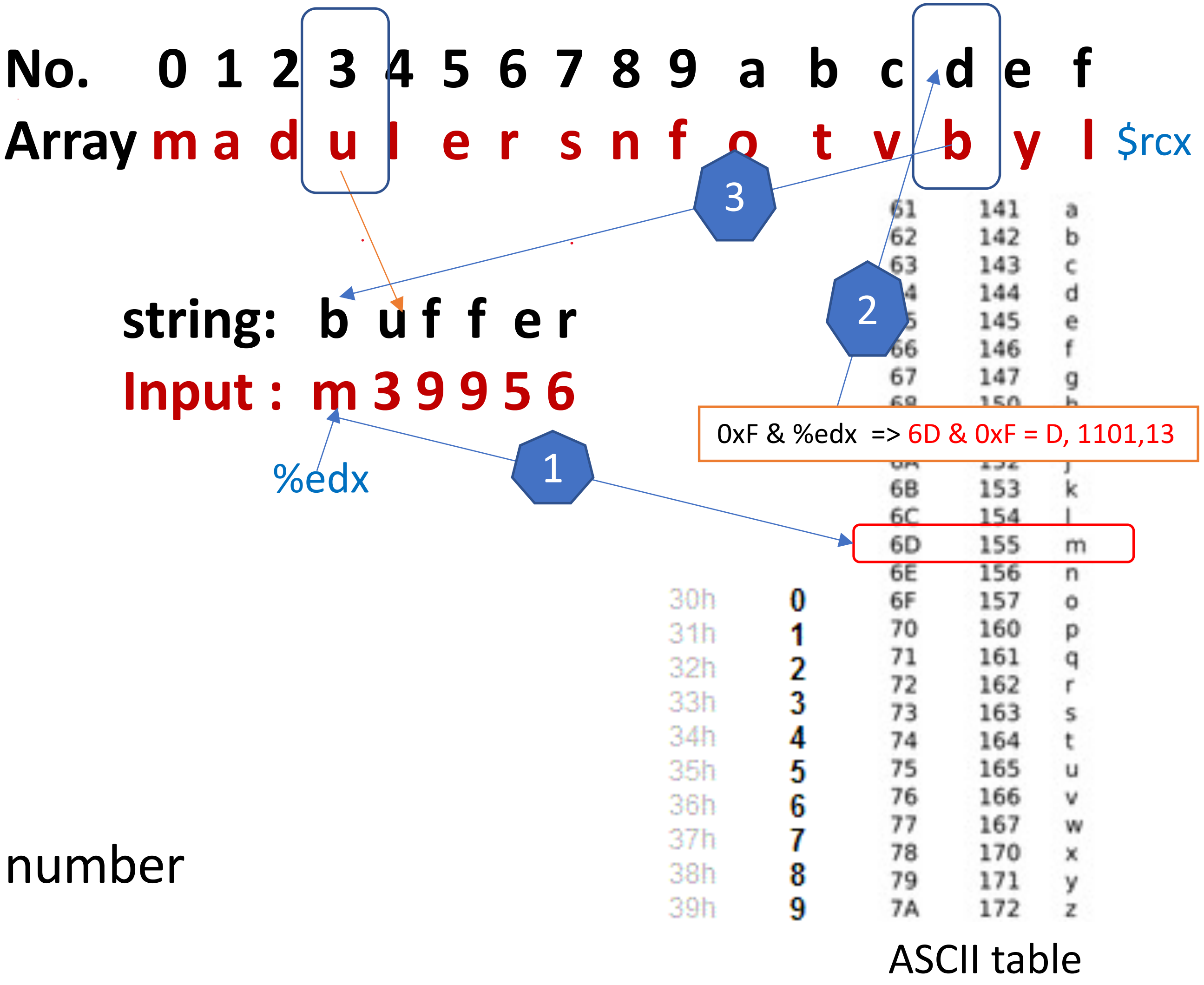


```
B+> 0x555555555328 <phase_5>         push   %rbx
    0x555555555329 <phase_5+1>       sub    $0x10,%rsp
    0x55555555532d <phase_5+5>       mov    %rdi,%rbx
    0x555555555330 <phase_5+8>       callq  0x5555555555a5 <string_length>
    0x555555555335 <phase_5+13>      cmp    $0x6,%eax
    0x555555555338 <phase_5+16>      jne    0x55555555537f <phase_5+87>
    0x55555555533a <phase_5+18>      mov    $0x0,%eax
    0x55555555533f <phase_5+23>      lea    0x150a(%rip),%rcx        # 0x555555556850 <array.3417>
    0x555555555346 <phase_5+30>      movzbl (%rbx,%rax,1),%edx
    0x55555555534a <phase_5+34>      and    $0xf,%edx
    0x55555555534d <phase_5+37>      movzbl (%rcx,%rdx,1),%edx
    0x555555555351 <phase_5+41>      mov    %dl,0x9(%rsp,%rax,1)
    0x555555555355 <phase_5+45>      add    $0x1,%rax
    0x555555555359 <phase_5+49>      cmp    $0x6,%rax
    0x55555555535d <phase_5+53>      jne    0x555555555346 <phase_5+30>
    0x55555555535f <phase_5+55>      movb   $0x0,0xf(%rsp)
    0x555555555364 <phase_5+60>      lea    0x9(%rsp),%rdi
    0x555555555369 <phase_5+65>      lea    0x14ae(%rip),%rsi        # 0x55555555681e
    0x555555555370 <phase_5+72>      callq  0x5555555553c2 <strings_not_equal>
    0x555555555375 <phase_5+77>      test   %eax,%eax
    0x555555555377 <phase_5+79>      jne    0x555555555386 <phase_5+94>
    0x555555555379 <phase_5+81>      add    $0x10,%rsp
    0x55555555537d <phase_5+85>      pop    %rbx
    0x55555555537e <phase_5+86>      retq
    0x55555555537f <phase_5+87>      callq  0x5555555557ec <explode_bomb>
    0x555555555384 <phase_5+92>      jmp    0x55555555533a <phase_5+18>
    0x555555555386 <phase_5+94>      callq  0x5555555557ec <explode_bomb>
    0x55555555538b <phase_5+99>      jmp    0x555555555379 <phase_5+81>
```

movzbl: move an 8bit value and extend it to a 32bit value, zero-fill

Phase 5: Array

- Example
- Get an array of 16 characters
  - matching from 0 to 15
- Get the input string
  - 6 characters %rbx
  - Extract the least 4 bits: 0xF & $edx
    - Array[(input[i] & 0x0f)];
- Find the answer
- Other cases
  - Find summation value: ex)53
  - Summation until find 15 or another number



No. 0 1 2 3 4 5 6 7 8 9 a b c d e f

Array m a d u l e r s n f o t v b y l $rcx

string: b u f f e r

Input : m 3 9 9 5 6

%edx

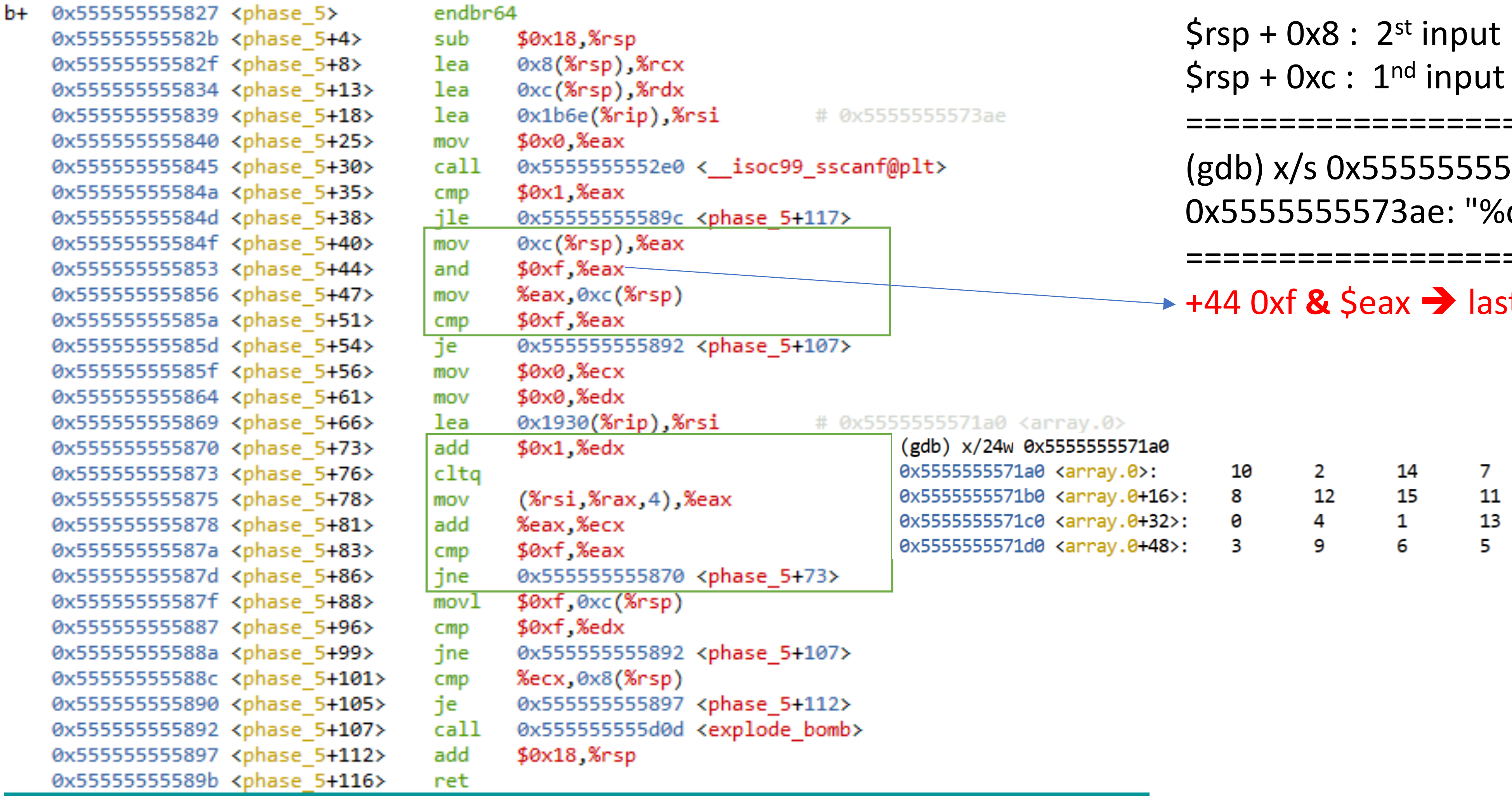
0xF & %edx => 6D & 0xF = D, 1101,13

30h	0
31h	1
32h	2
33h	3
34h	4
35h	5
36h	6
37h	7
38h	8
39h	9

61	141	a
62	142	b
63	143	c
64	144	d
65	145	e
66	146	f
67	147	g
68	150	h
6A	152	j
6B	153	k
6C	154	l
6D	155	m
6E	156	n
6F	157	o
70	160	p
71	161	q
72	162	r
73	163	s
74	164	t
75	165	u
76	166	v
77	167	w
78	170	x
79	171	y
7A	172	z

ASCII table

Phase 5: Array



```
b+  0x555555555827 <phase_5>         endbr64
    0x55555555582b <phase_5+4>       sub    $0x18,%rsp
    0x55555555582f <phase_5+8>       lea    0x8(%rsp),%rcx
    0x555555555834 <phase_5+13>      lea    0xc(%rsp),%rdx
    0x555555555839 <phase_5+18>      lea    0x1b6e(%rip),%rsi        # 0x5555555573ae
    0x555555555840 <phase_5+25>      mov    $0x0,%eax
    0x555555555845 <phase_5+30>      call   0x5555555552e0 <__isoc99_sscanf@plt>
    0x55555555584a <phase_5+35>      cmp    $0x1,%eax
    0x55555555584d <phase_5+38>      jle    0x55555555589c <phase_5+117>
    0x55555555584f <phase_5+40>      mov    0xc(%rsp),%eax
    0x555555555853 <phase_5+44>      and    $0xf,%eax
    0x555555555856 <phase_5+47>      mov    %eax,0xc(%rsp)
    0x55555555585a <phase_5+51>      cmp    $0xf,%eax
    0x55555555585d <phase_5+54>      je     0x555555555892 <phase_5+107>
    0x55555555585f <phase_5+56>      mov    $0x0,%ecx
    0x555555555864 <phase_5+61>      mov    $0x0,%edx
    0x555555555869 <phase_5+66>      lea    0x1930(%rip),%rsi        # 0x5555555571a0 <array.0>
    0x555555555870 <phase_5+73>      add    $0x1,%edx
    0x555555555873 <phase_5+76>      cltq
    0x555555555875 <phase_5+78>      mov    (%rsi,%rax,4),%eax
    0x555555555878 <phase_5+81>      add    %eax,%ecx
    0x55555555587a <phase_5+83>      cmp    $0xf,%eax
    0x55555555587d <phase_5+86>      jne    0x555555555870 <phase_5+73>
    0x55555555587f <phase_5+88>      movl   $0xf,0xc(%rsp)
    0x555555555887 <phase_5+96>      cmp    $0xf,%edx
    0x55555555588a <phase_5+99>      jne    0x555555555892 <phase_5+107>
    0x55555555588c <phase_5+101>     cmp    %ecx,0x8(%rsp)
    0x555555555890 <phase_5+105>     je     0x555555555897 <phase_5+112>
    0x555555555892 <phase_5+107>     call   0x555555555d0d <explode_bomb>
    0x555555555897 <phase_5+112>     add    $0x18,%rsp
    0x55555555589b <phase_5+116>     ret
```

```
(gdb) x/24w 0x5555555571a0
0x5555555571a0 <array.0>:       10      2       14      7
0x5555555571b0 <array.0+16>:    8       12      15      11
0x5555555571c0 <array.0+32>:    0       4       1       13
0x5555555571d0 <array.0+48>:    3       9       6       5
```

$rsp + 0x8 : 2nd input
$rsp + 0xc : 1nd input

==============================

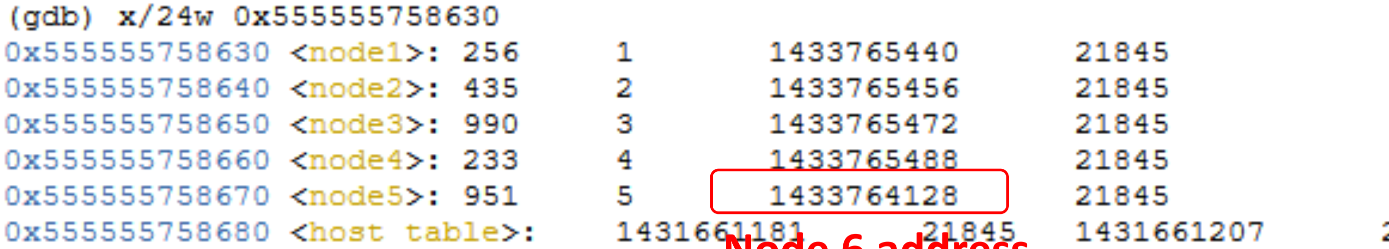
(gdb) x/s 0x5555555573ae
0x5555555573ae: "%d %d"

==============================

+44 0xf **&** $eax ➔ last 4bits

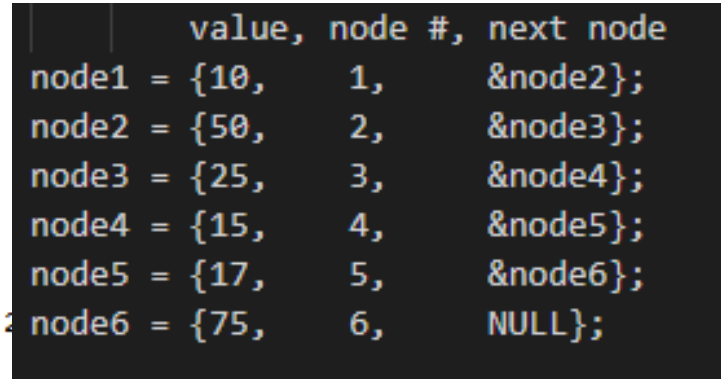
Phase 6: Linked list

- **Linked Lists Sorting in Descending/Ascending Order**
  - Figure out how to check the input values
    - <read_six_numbers>
    - Comparison and Jump instructions
      - each integer <= 6 and no integer should be the same as any others
  - Figure out how to rearrange the lists according to the input
    - Find the original lists
      - x/24w 0x555555758630 shows you the table of linked lists
      - Find node6 address



```
(gdb) x/24w 0x555555758630
0x555555758630 <node1>: 256      1        1433765440      21845
0x555555758640 <node2>: 435      2        1433765456      21845
0x555555758650 <node3>: 990      3        1433765472      21845
0x555555758660 <node4>: 233      4        1433765488      21845
0x555555758670 <node5>: 951      5        1433764128      21845
0x555555758680 <host table>:     1431661181      21845   1431661207
```

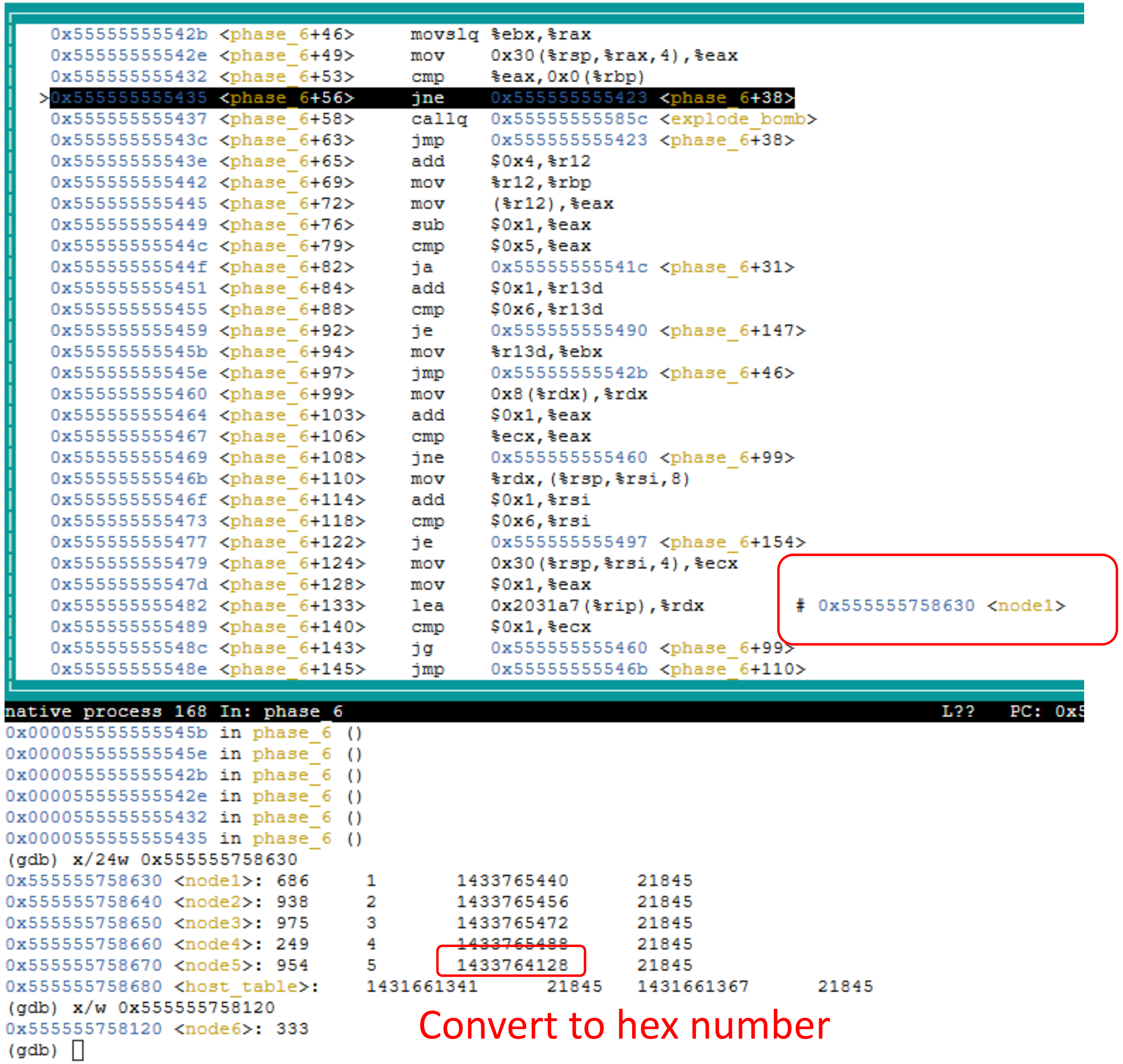
Node 6 address



```
                value, node #, next node
node1 = {10,     1,      &node2};
node2 = {50,     2,      &node3};
node3 = {25,     3,      &node4};
node4 = {15,     4,      &node5};
node5 = {17,     5,      &node6};
node6 = {75,     6,      NULL};
```

    - Try sorting the lists in ascending or descending order
    - Some bomb reverses the order : input index = 7 - input index

Phase 6: Linked list



```
   0x55555555542b <phase_6+46>     movslq %ebx,%rax
   0x55555555542e <phase_6+49>     mov    0x30(%rsp,%rax,4),%eax
   0x555555555432 <phase_6+53>     cmp    %eax,0x0(%rbp)
  >0x555555555435 <phase_6+56>     jne    0x555555555423 <phase_6+38>
   0x555555555437 <phase_6+58>     callq  0x55555555585c <explode_bomb>
   0x55555555543c <phase_6+63>     jmp    0x555555555423 <phase_6+38>
   0x55555555543e <phase_6+65>     add    $0x4,%r12
   0x555555555442 <phase_6+69>     mov    %r12,%rbp
   0x555555555445 <phase_6+72>     mov    (%r12),%eax
   0x555555555449 <phase_6+76>     sub    $0x1,%eax
   0x55555555544c <phase_6+79>     cmp    $0x5,%eax
   0x55555555544f <phase_6+82>     ja     0x55555555541c <phase_6+31>
   0x555555555451 <phase_6+84>     add    $0x1,%r13d
   0x555555555455 <phase_6+88>     cmp    $0x6,%r13d
   0x555555555459 <phase_6+92>     je     0x555555555490 <phase_6+147>
   0x55555555545b <phase_6+94>     mov    %r13d,%ebx
   0x55555555545e <phase_6+97>     jmp    0x55555555542b <phase_6+46>
   0x555555555460 <phase_6+99>     mov    0x8(%rdx),%rdx
   0x555555555464 <phase_6+103>    add    $0x1,%eax
   0x555555555467 <phase_6+106>    cmp    %ecx,%eax
   0x555555555469 <phase_6+108>    jne    0x555555555460 <phase_6+99>
   0x55555555546b <phase_6+110>    mov    %rdx,(%rsp,%rsi,8)
   0x55555555546f <phase_6+114>    add    $0x1,%rsi
   0x555555555473 <phase_6+118>    cmp    $0x6,%rsi
   0x555555555477 <phase_6+122>    je     0x555555555497 <phase_6+154>
   0x555555555479 <phase_6+124>    mov    0x30(%rsp,%rsi,4),%ecx
   0x55555555547d <phase_6+128>    mov    $0x1,%eax
   0x555555555482 <phase_6+133>    lea    0x2031a7(%rip),%rdx        # 0x555555758630 <node1>
   0x555555555489 <phase_6+140>    cmp    $0x1,%ecx
   0x55555555548c <phase_6+143>    jg     0x555555555460 <phase_6+99>
   0x55555555548e <phase_6+145>    jmp    0x55555555546b <phase_6+110>
native process 168 In: phase_6                                      L??   PC: 0x5
0x000055555555545b in phase_6 ()
0x000055555555545e in phase_6 ()
0x000055555555542b in phase_6 ()
0x000055555555542e in phase_6 ()
0x0000555555555432 in phase_6 ()
0x0000555555555435 in phase_6 ()
(gdb) x/24w 0x555555758630
0x555555758630 <node1>: 686     1       1433765440      21845
0x555555758640 <node2>: 938     2       1433765456      21845
0x555555758650 <node3>: 975     3       1433765472      21845
0x555555758660 <node4>: 249     4       1433765488      21845
0x555555758670 <node5>: 954     5       1433764128      21845
0x555555758680 <host_table>:    1431661341      21845   1431661367      21845
(gdb) x/w 0x555555758120
0x555555758120 <node6>: 333
(gdb) 
```

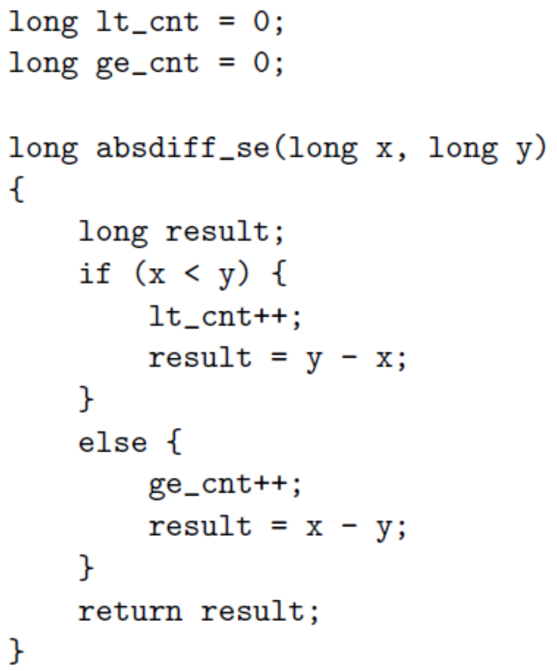
Convert to hex number

Subroutines

call 0x15213	→ Push address of instruction following the call on the stack, then jump to the address of 0x15213
… more code …	
[at address 0x15213:]	
push %r12	→ Push callee-saved registers on stack
… subroutine body …	→ Perform subroutine
pop %r12	→ Restore registers from stack
ret	→ Pop return address and jump there

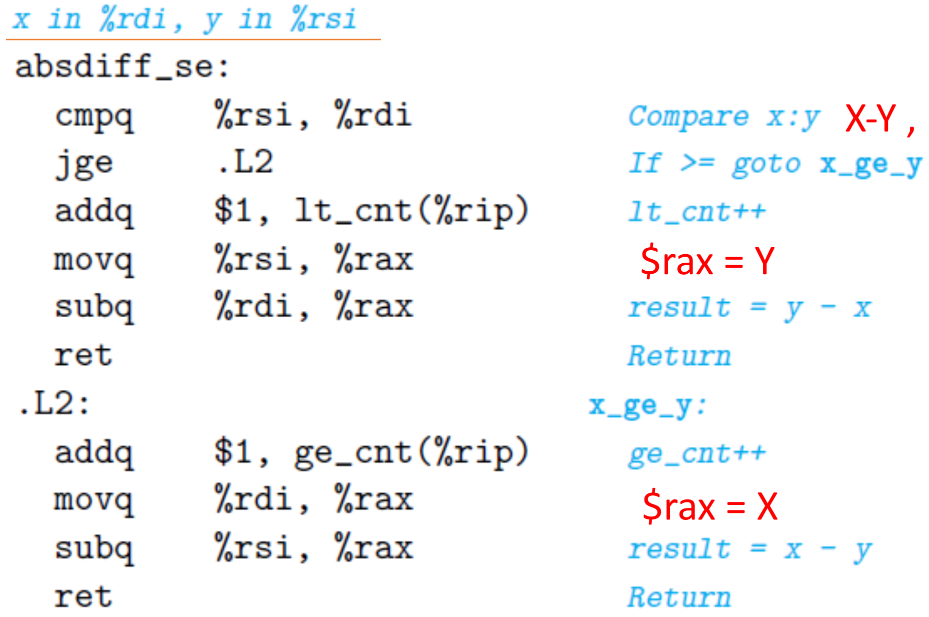
Assembly Codes Reading

- Example codes (textbook p.210)



```
long lt_cnt = 0;
long ge_cnt = 0;

long absdiff_se(long x, long y)
{
    long result;
    if (x < y) {
        lt_cnt++;
        result = y - x;
    }
    else {
        ge_cnt++;
        result = x - y;
    }
    return result;
}
```



```
x in %rdi, y in %rsi
absdiff_se:
  cmpq    %rsi, %rdi           Compare x:y   X-Y ,  $rdi - $rsi
  jge     .L2                  If >= goto x_ge_y   Greater or equal
  addq    $1, lt_cnt(%rip)     lt_cnt++
  movq    %rsi, %rax            $rax = Y
  subq    %rdi, %rax           result = y - x
  ret                          Return
.L2:                         x_ge_y:
  addq    $1, ge_cnt(%rip)     ge_cnt++
  movq    %rdi, %rax            $rax = X
  subq    %rsi, %rax           result = x - y
  ret                          Return
```

Examples with explanation: Understanding the stack with GCC

https://ulrichbuschbaum.wordpress.com/2015/10/30/understanding-the-stack-with-gcc/