COMPUTER SYSTEMS AND ORGANIZATION

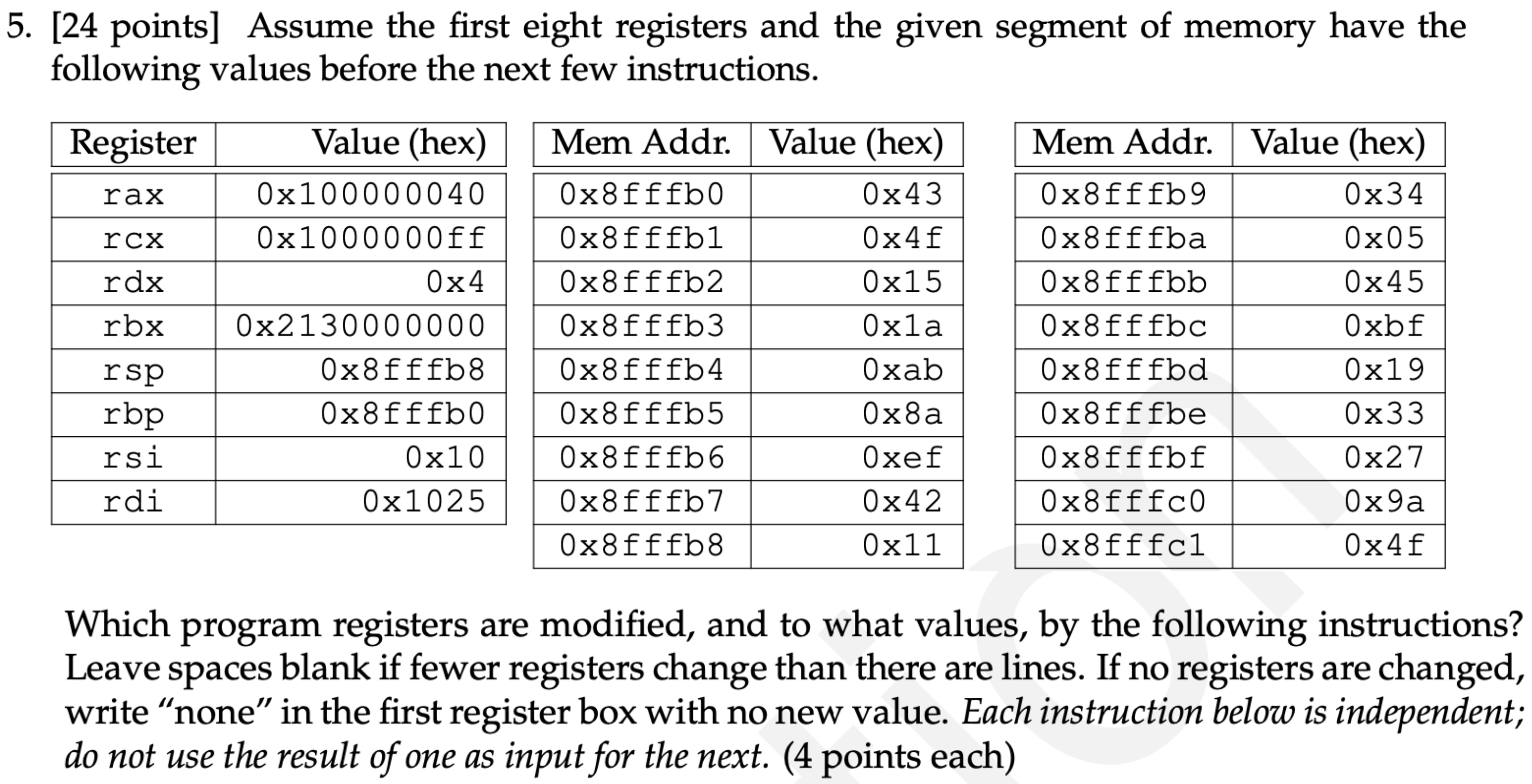
Part 1

Daniel G. Graham Ph.D

University of Virginia | ENGINEERING

Contents

1. Work on past exam questions
2. Discussion of Patents, Copyrights, and Open Source Technology. Think about the future as engineers.
3. Only 16 lectures left. We close out with C



5. [24 points] Assume the first eight registers and the given segment of memory have the following values before the next few instructions.

Register	Value (hex)
rax	0x100000040
rcx	0x1000000ff
rdx	0x4
rbx	0x2130000000
rsp	0x8fffb8
rbp	0x8fffb0
rsi	0x10
rdi	0x1025

Mem Addr.	Value (hex)
0x8fffb0	0x43
0x8fffb1	0x4f
0x8fffb2	0x15
0x8fffb3	0x1a
0x8fffb4	0xab
0x8fffb5	0x8a
0x8fffb6	0xef
0x8fffb7	0x42
0x8fffb8	0x11

Mem Addr.	Value (hex)
0x8fffb9	0x34
0x8fffba	0x05
0x8fffbb	0x45
0x8fffbc	0xbf
0x8fffbd	0x19
0x8fffbe	0x33
0x8fffbf	0x27
0x8fffc0	0x9a
0x8fffc1	0x4f

Which program registers are modified, and to what values, by the following instructions? Leave spaces blank if fewer registers change than there are lines. If no registers are changed, write "none" in the first register box with no new value. *Each instruction below is independent; do not use the result of one as input for the next.* (4 points each)

Register	Value (hex)
rax	0x100000040
rcx	0x1000000ff
rdx	0x4
rbx	0x2130000000
rsp	0x8fffb8
rbp	0x8fffb0
rsi	0x10
rdi	0x1025

Mem Addr.	Value (hex)
0x8fffb0	0x43
0x8fffb1	0x4f
0x8fffb2	0x15
0x8fffb3	0x1a
0x8fffb4	0xab
0x8fffb5	0x8a
0x8fffb6	0xef
0x8fffb7	0x42
0x8fffb8	0x11

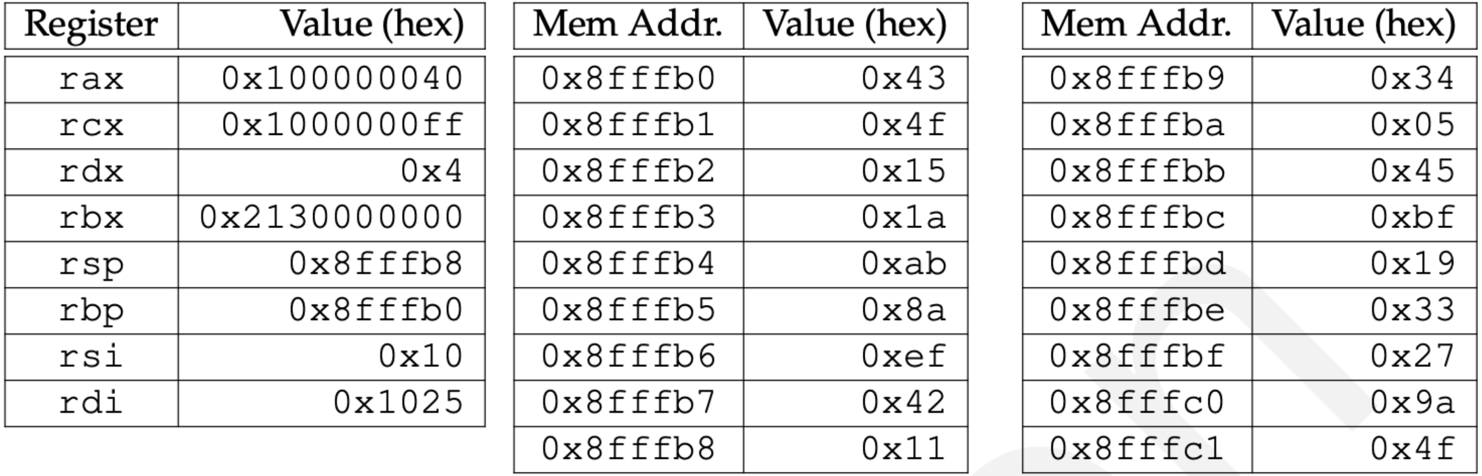
Mem Addr.	Value (hex)
0x8fffb9	0x34
0x8fffba	0x05
0x8fffbb	0x45
0x8fffbc	0xbf
0x8fffbd	0x19
0x8fffbe	0x33
0x8fffbf	0x27
0x8fffc0	0x9a
0x8fffc1	0x4f

```
movl 0x8(%rbp), %edx
```

Register	New Value

```
leaq 0x8(%rbp), %rdx
```

Register	New Value



Register	Value (hex)
rax	0x100000040
rcx	0x1000000ff
rdx	0x4
rbx	0x2130000000
rsp	0x8fffb8
rbp	0x8fffb0
rsi	0x10
rdi	0x1025

Mem Addr.	Value (hex)
0x8fffb0	0x43
0x8fffb1	0x4f
0x8fffb2	0x15
0x8fffb3	0x1a
0x8fffb4	0xab
0x8fffb5	0x8a
0x8fffb6	0xef
0x8fffb7	0x42
0x8fffb8	0x11

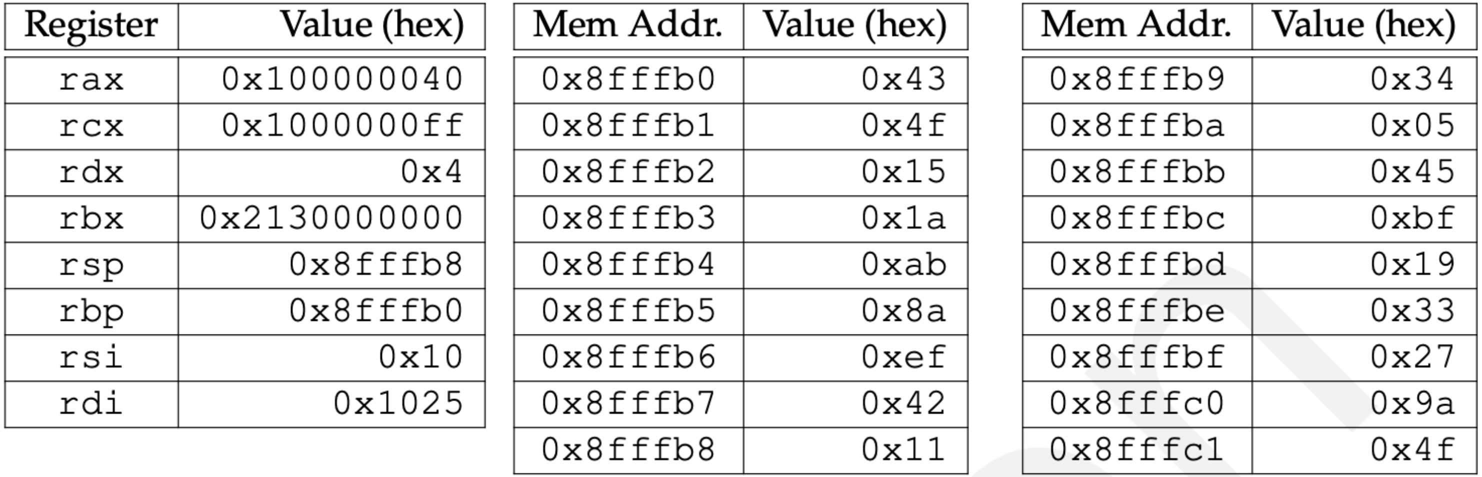
Mem Addr.	Value (hex)
0x8fffb9	0x34
0x8fffba	0x05
0x8fffbb	0x45
0x8fffbc	0xbf
0x8fffbd	0x19
0x8fffbe	0x33
0x8fffbf	0x27
0x8fffc0	0x9a
0x8fffc1	0x4f

```
testq %rdx, %rdi
```

Register	New Value

```
andl -0x10(%rsp,%rdx,2), %ecx
```

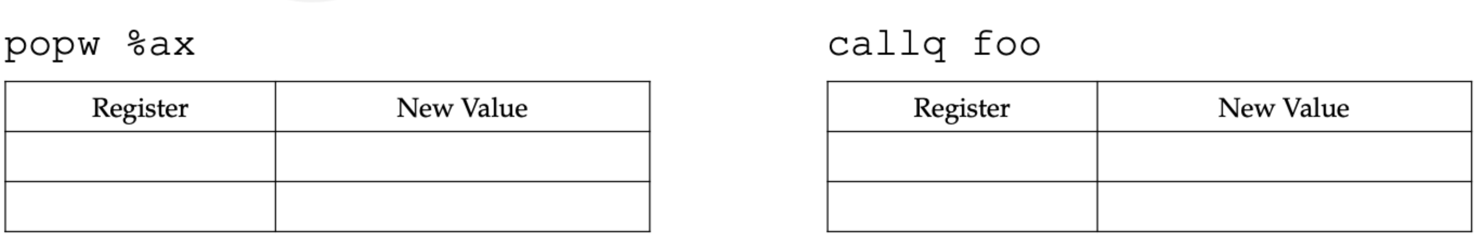
Register	New Value



Register	Value (hex)
rax	0x100000040
rcx	0x1000000ff
rdx	0x4
rbx	0x2130000000
rsp	0x8fffb8
rbp	0x8fffb0
rsi	0x10
rdi	0x1025

Mem Addr.	Value (hex)
0x8fffb0	0x43
0x8fffb1	0x4f
0x8fffb2	0x15
0x8fffb3	0x1a
0x8fffb4	0xab
0x8fffb5	0x8a
0x8fffb6	0xef
0x8fffb7	0x42
0x8fffb8	0x11

Mem Addr.	Value (hex)
0x8fffb9	0x34
0x8fffba	0x05
0x8fffbb	0x45
0x8fffbc	0xbf
0x8fffbd	0x19
0x8fffbe	0x33
0x8fffbf	0x27
0x8fffc0	0x9a
0x8fffc1	0x4f

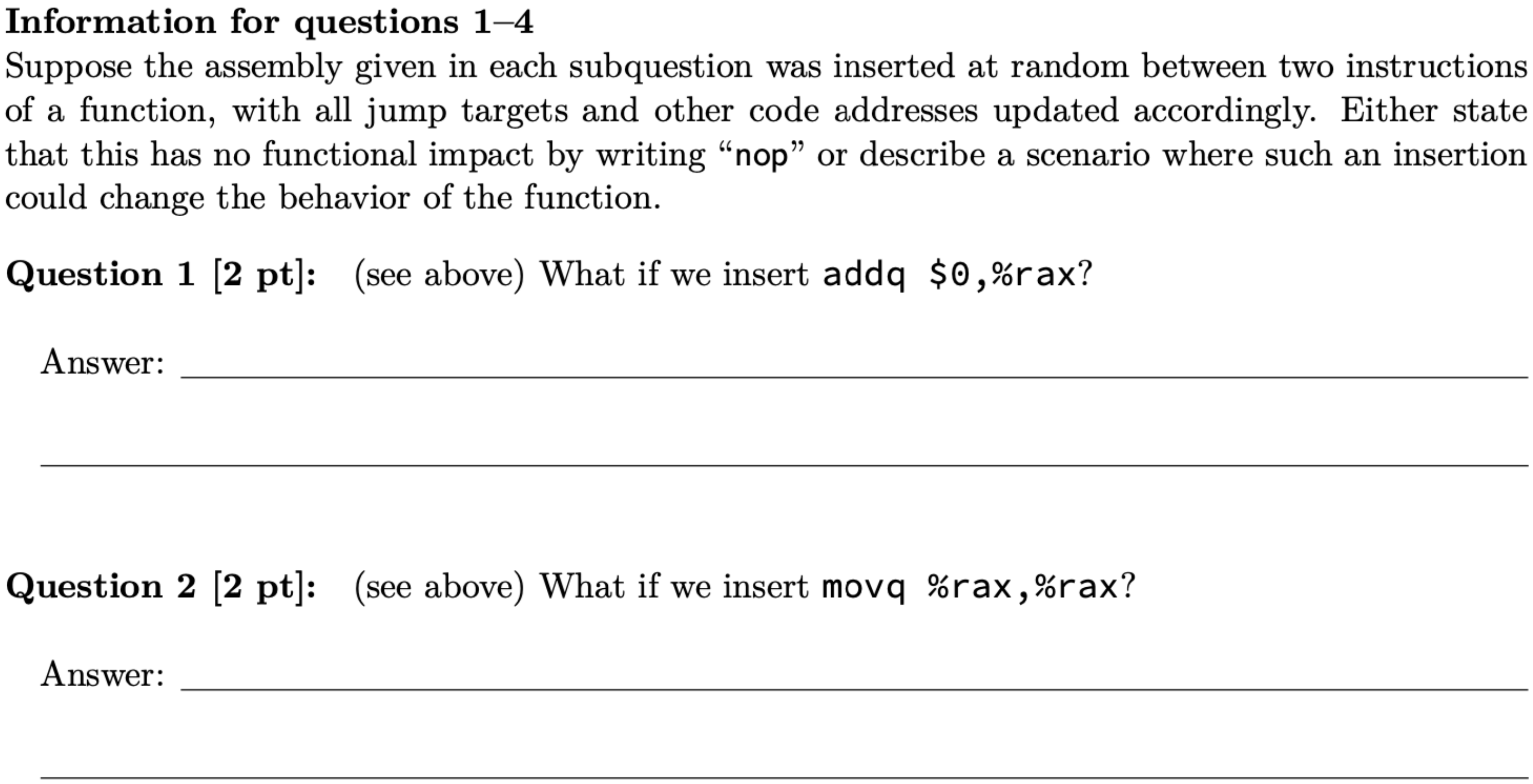


```
popw %ax
```

Register	New Value

```
callq foo
```

Register	New Value



Information for questions 1–4

Suppose the assembly given in each subquestion was inserted at random between two instructions of a function, with all jump targets and other code addresses updated accordingly. Either state that this has no functional impact by writing "`nop`" or describe a scenario where such an insertion could change the behavior of the function.

Question 1 [2 pt]: (see above) What if we insert `addq $0,%rax`?

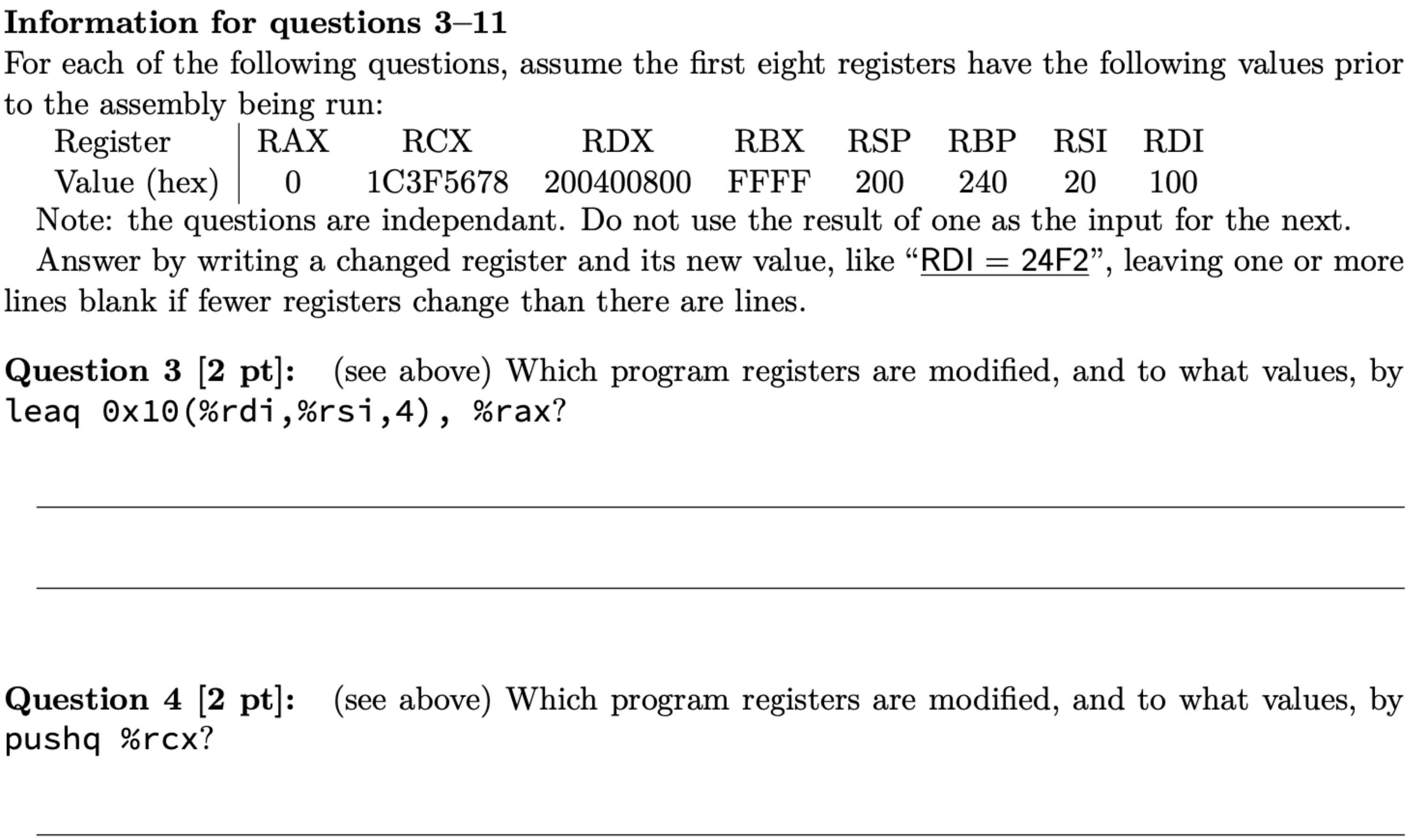
Answer: ____

____

Question 2 [2 pt]: (see above) What if we insert `movq %rax,%rax`?

Answer: ____

____



Information for questions 3–11

For each of the following questions, assume the first eight registers have the following values prior to the assembly being run:

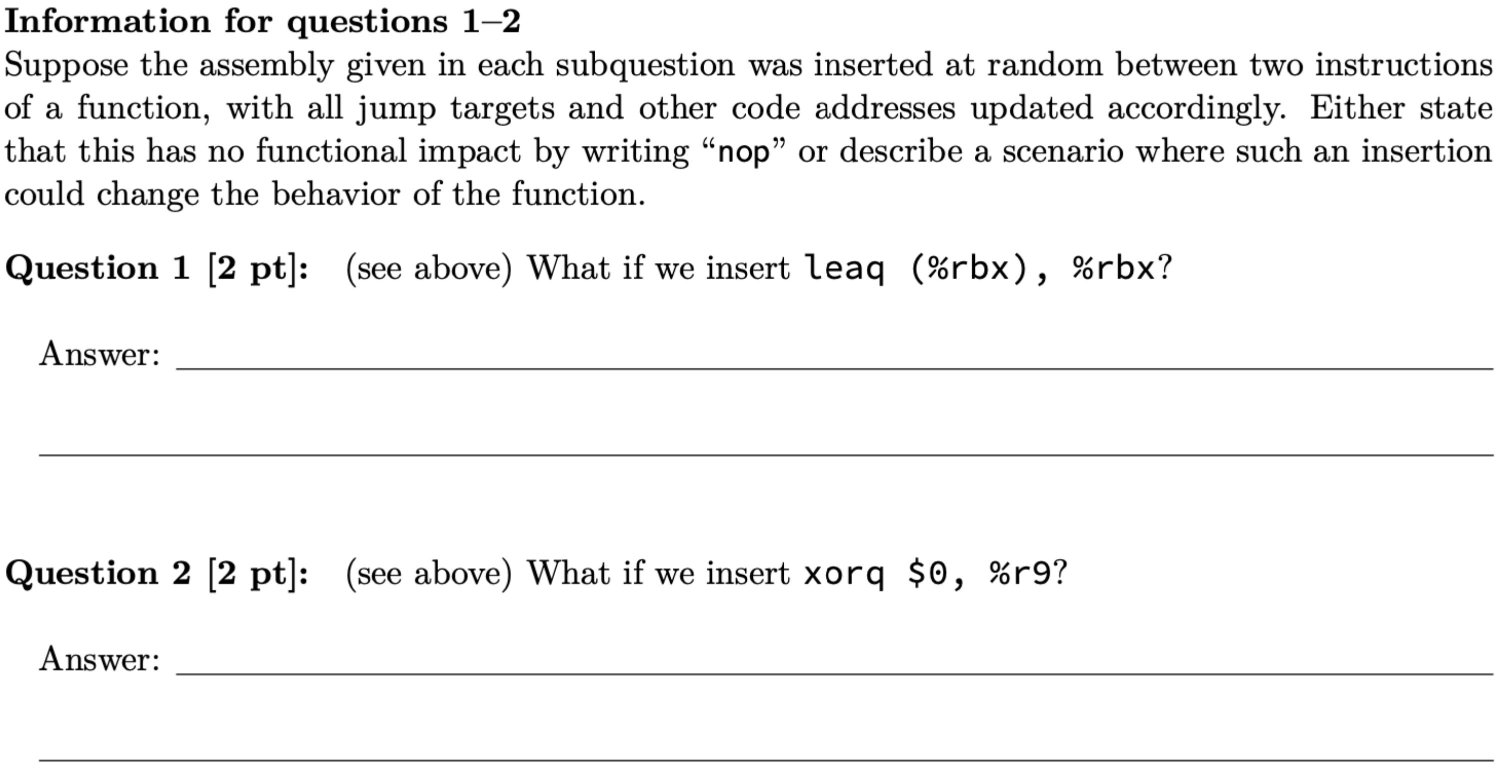
Register	RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI
Value (hex)	0	1C3F5678	200400800	FFFF	200	240	20	100

Note: the questions are independant. Do not use the result of one as the input for the next.

Answer by writing a changed register and its new value, like “RDI = 24F2”, leaving one or more lines blank if fewer registers change than there are lines.

Question 3 [2 pt]: (see above) Which program registers are modified, and to what values, by `leaq 0x10(%rdi,%rsi,4), %rax`?

Question 4 [2 pt]: (see above) Which program registers are modified, and to what values, by `pushq %rcx`?



Information for questions 1–2

Suppose the assembly given in each subquestion was inserted at random between two instructions of a function, with all jump targets and other code addresses updated accordingly. Either state that this has no functional impact by writing "`nop`" or describe a scenario where such an insertion could change the behavior of the function.

Question 1 [2 pt]: (see above) What if we insert `leaq (%rbx), %rbx`?

Answer: ____

____

Question 2 [2 pt]: (see above) What if we insert `xorq $0, %r9`?

Answer: ____

____

```
je target          jump if ZF is 1
```

Let `%edi` store 0x10. Will we jump in the following cases? `%edi` | `0x10`

1.
   ```
   cmp $0x10,%edi
   je  40056f
   add  $0x1,%edi
   ```

2.
   ```
   test $0x10,%edi
   je   40056f
   add  $0x1,%edi
   ```

```
je target            jump if ZF is 1
```

Let `%edi` store 0x10. Will we jump in the following cases? `%edi` 0x10

1.
```
cmp $0x10,%edi
je   40056f
add  $0x1,%edi
```

S2 - S1 == 0, so jump

2.
```
test $0x10,%edi
je    40056f
add  $0x1,%edi
```

S2 & S1 != 0, so don't jump

```
int if_then(int param1) {
  if ( __________ ) {
     _________;
  }

  return __________;
}
```

```
00000000004004d6 <if_then>:
  4004d6:    cmp  $0x6,%edi
  4004d9:    jne  4004de
  4004db:    add  $0x1,%edi
  4004de:    lea  (%rdi,%rdi,1),%eax
  4004e1:    retq
```

🤔

```c
int if_then(int param1) {
  if (param1 == 6 ) {
       param1++ ;
  }

  return  param1 * 2;
}
```

```
0000000000004004d6 <if_then>:
  4004d6:    cmp  $0x6,%edi
  4004d9:    jne  4004de
  4004db:    add  $0x1,%edi
  4004de:    lea  (%rdi,%rdi,1),%eax
  4004e1:    retq
```

🤔

```
if ( __________ ) {
    _________;
} else {
    ________;
}
______;
```

```
400552 <+0>:   cmp  $0x3,%edi
400555 <+3>:   jle  0x40055e <if_else+12>
400557 <+5>:   mov  $0xa,%eax
40055c <+10>:  jmp  0x400563 <if_else+17>
40055e <+12>:  mov  $0x0,%eax
400563 <+17>:  add  $0x1,%eax
```

```
if ( arg > 3   ) {
    ret = 10;
} else {
    ret = 0;
}
ret++;
```

```
400552 <+0>:  cmp  $0x3,%edi
400555 <+3>:  jle  0x40055e <if_else+12>
400557 <+5>:  mov  $0xa,%eax
40055c <+10>: jmp  0x400563 <if_else+17>
40055e <+12>: mov  $0x0,%eax
400563 <+17>: add  $0x1,%eax
```

ESCAPE ROOM FUN

```
escapeRoom:
  leal (%rdi,%rdi), %eax
  cmpl $5, %eax
  jg .L3
  cmpl $1, %edi
  jne .L4
  movl $1, %eax
  ret
.L3:
  movl $1, %eax
  ret
.L4:
  movl $0, %eax
  ret
```

What must be passed to the Escape Room so that it returns true. Assume that we can supply an integer as input.

ESCAPE ROOM FUN

```
escapeRoom:
  leal (%rdi,%rdi), %eax
  cmpl $5, %eax
  jg .L3
  cmpl $1, %edi
  jne .L4
  movl $1, %eax
  ret
.L3:
  movl $1, %eax
  ret
.L4:
  movl $0, %eax
  ret
```

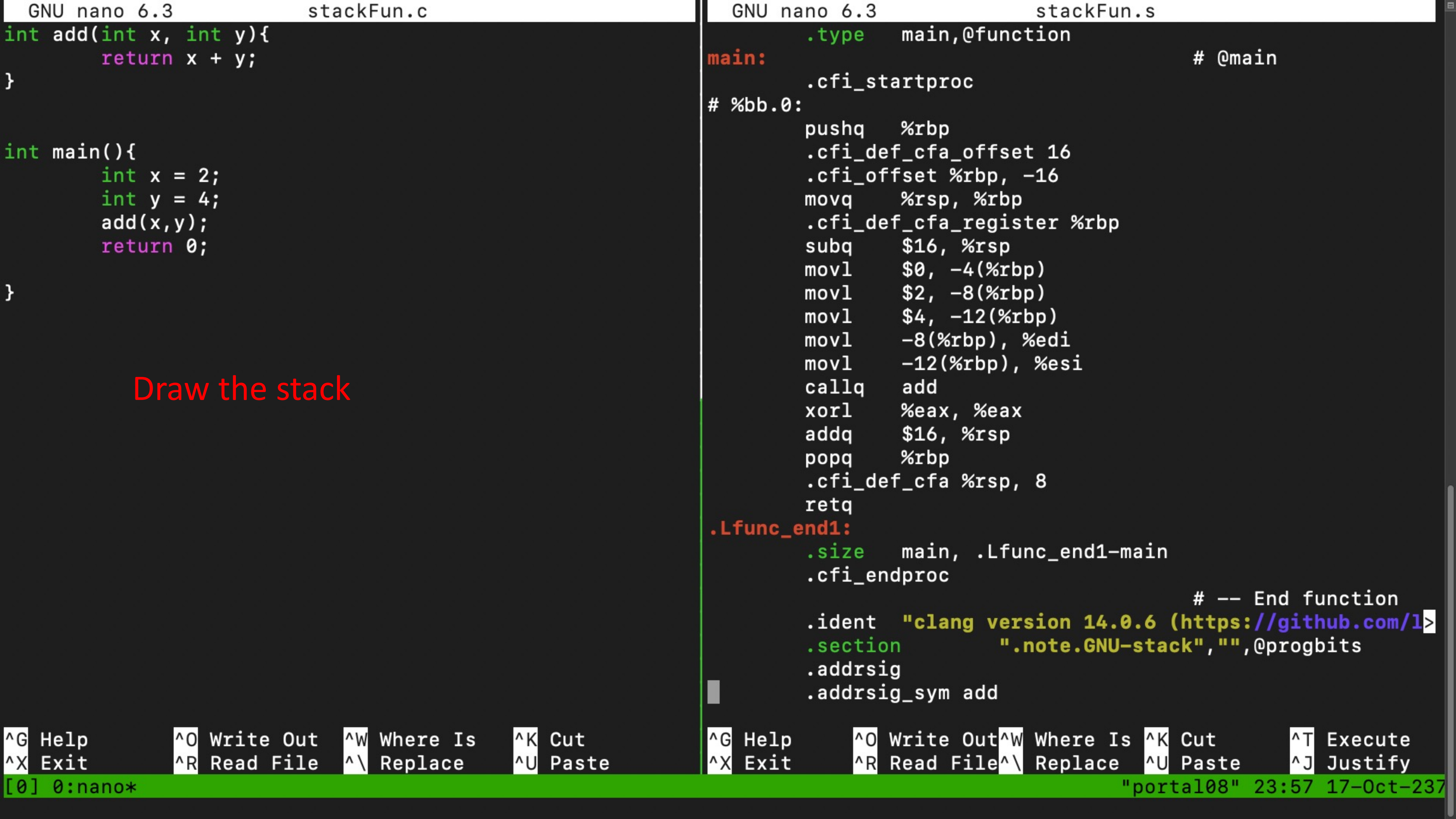
What must be passed to the Escape Room so that it returns true

First param > 2 or == 1

FUNCTION PARAMETERS AND THE STACK

Local variables are stored on the stack.

Let’s look at example.



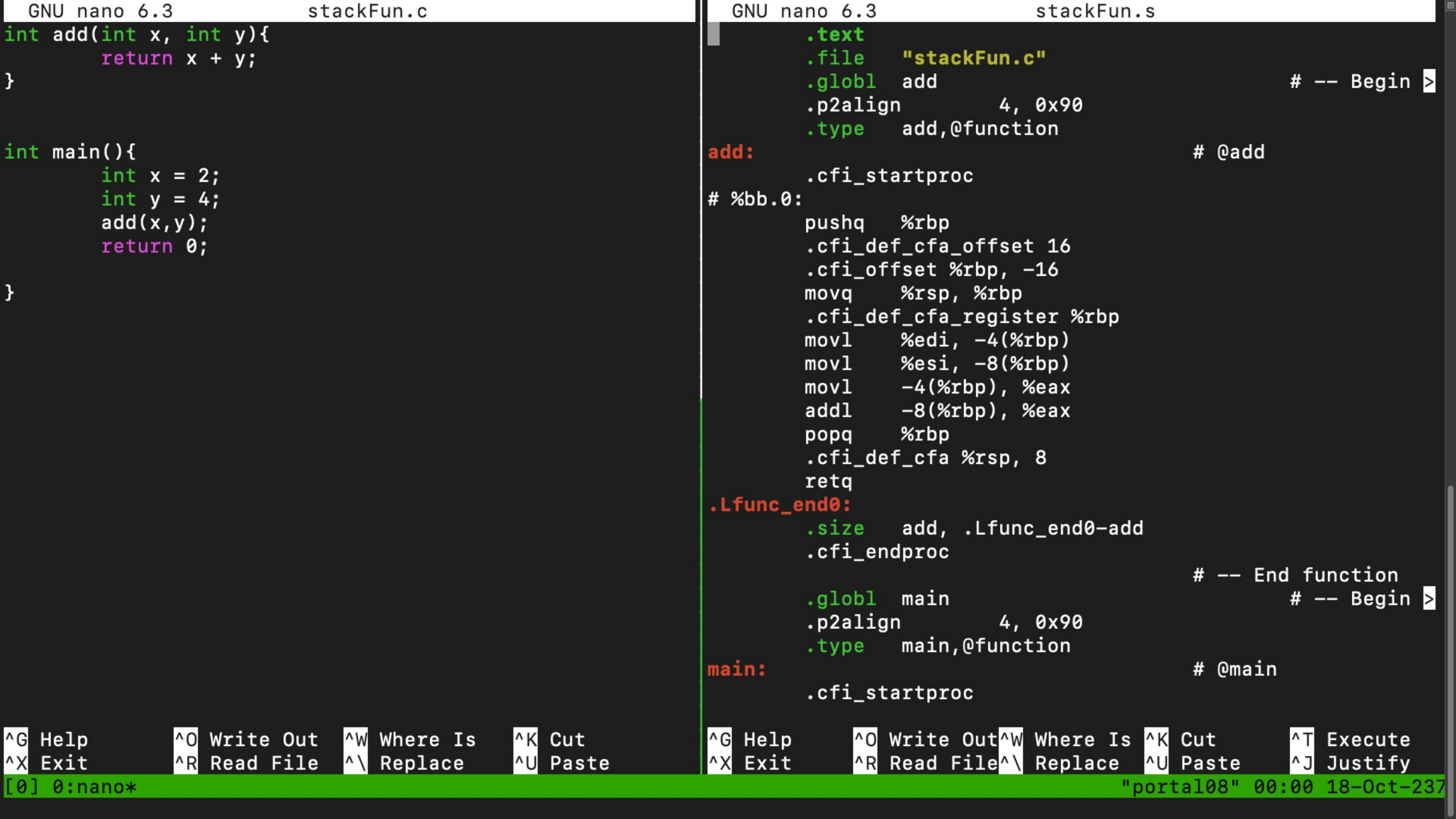
```c
int add(int x, int y){
        return x + y;
}


int main(){
        int x = 2;
        int y = 4;
        add(x,y);
        return 0;

}
```

Draw the stack

```
        .type   main,@function
main:                                   # @main
        .cfi_startproc
# %bb.0:
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register %rbp
        subq    $16, %rsp
        movl    $0, -4(%rbp)
        movl    $2, -8(%rbp)
        movl    $4, -12(%rbp)
        movl    -8(%rbp), %edi
        movl    -12(%rbp), %esi
        callq   add
        xorl    %eax, %eax
        addq    $16, %rsp
        popq    %rbp
        .cfi_def_cfa %rsp, 8
        retq
.Lfunc_end1:
        .size   main, .Lfunc_end1-main
        .cfi_endproc
                                        # -- End function
        .ident  "clang version 14.0.6 (https://github.com/l>
        .section        ".note.GNU-stack","",@progbits
        .addrsig
        .addrsig_sym add
```

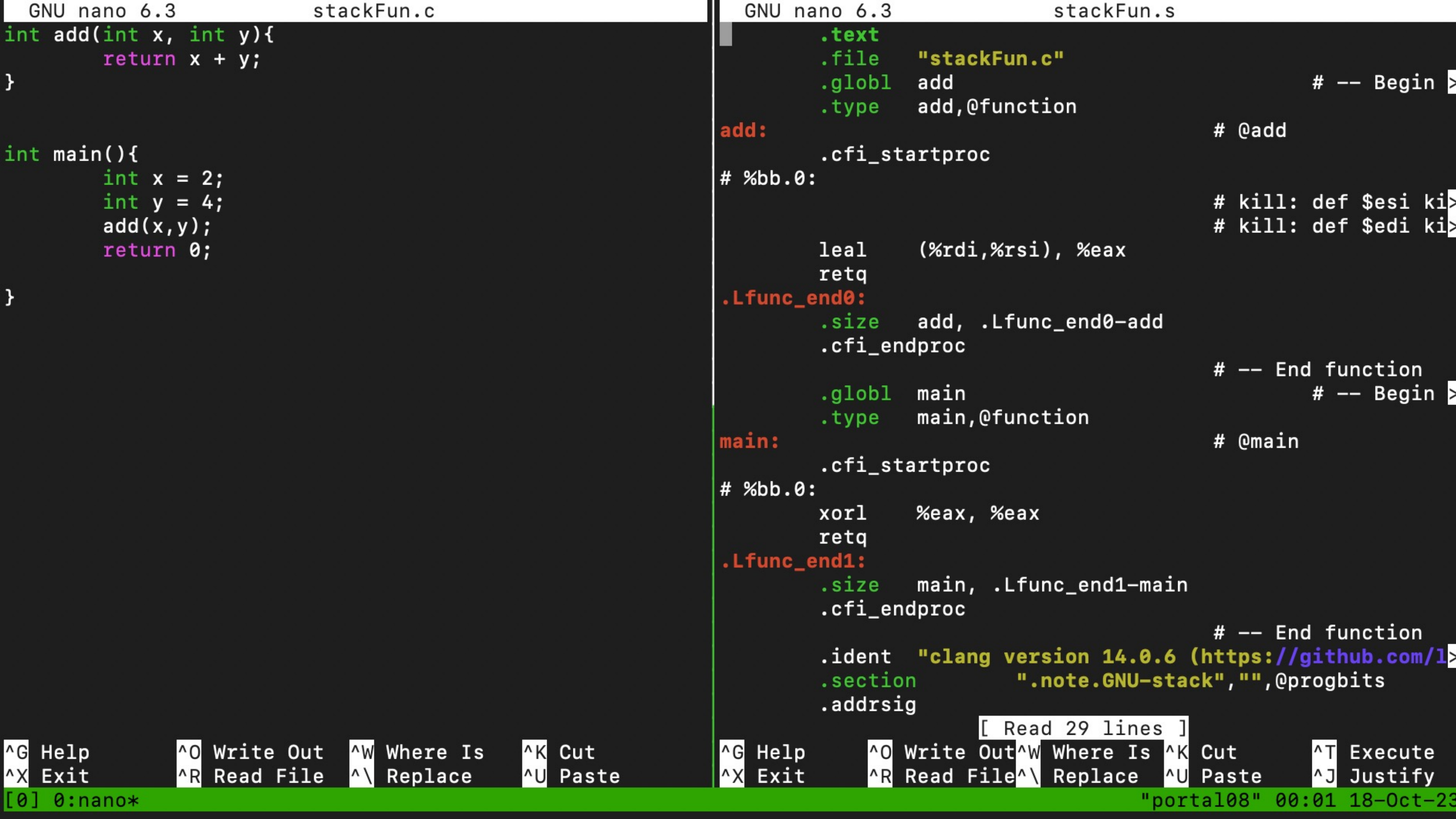


```
int add(int x, int y){
        return x + y;
}


int main(){
        int x = 2;
        int y = 4;
        add(x,y);
        return 0;

}
```

```
        .text
        .file   "stackFun.c"
        .globl  add                             # -- Begin >
        .p2align        4, 0x90
        .type   add,@function
add:                                    # @add
        .cfi_startproc
# %bb.0:
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register %rbp
        movl    %edi, -4(%rbp)
        movl    %esi, -8(%rbp)
        movl    -4(%rbp), %eax
        addl    -8(%rbp), %eax
        popq    %rbp
        .cfi_def_cfa %rsp, 8
        retq
.Lfunc_end0:
        .size   add, .Lfunc_end0-add
        .cfi_endproc
                                        # -- End function
        .globl  main                            # -- Begin >
        .p2align        4, 0x90
        .type   main,@function
main:                                   # @main
        .cfi_startproc
```



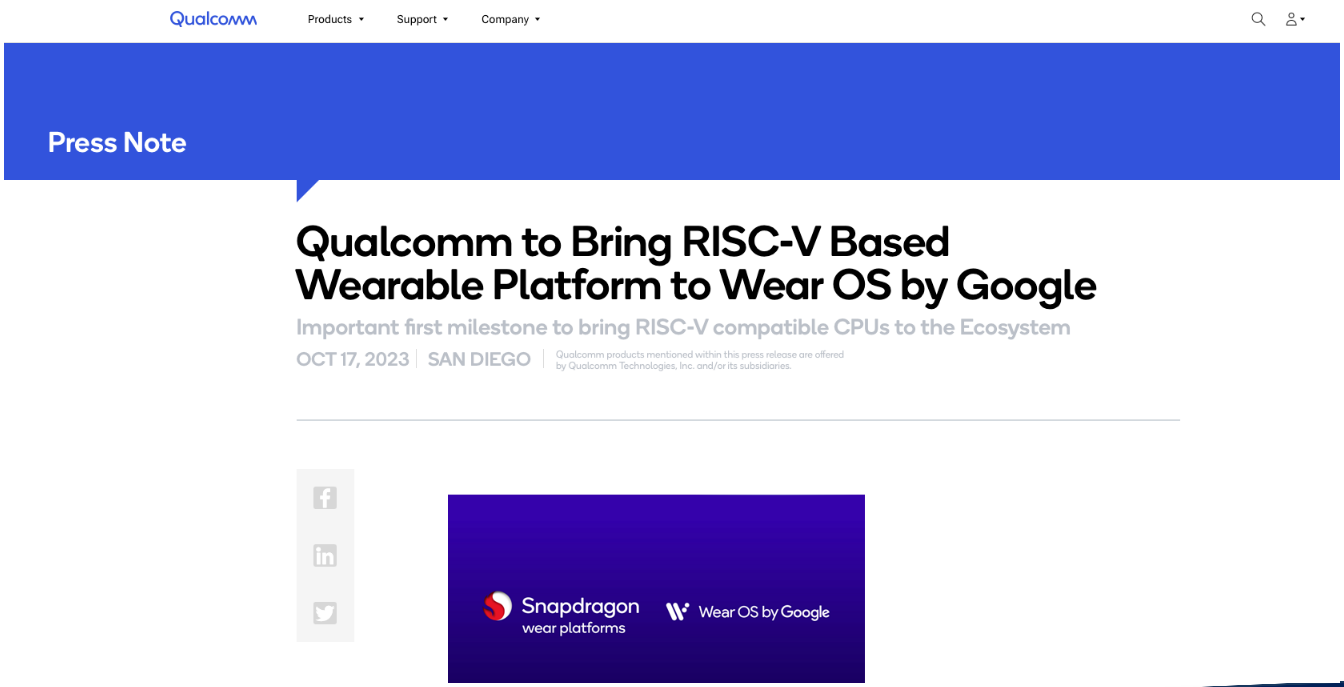
```c
int add(int x, int y){
        return x + y;
}


int main(){
        int x = 2;
        int y = 4;
        add(x,y);
        return 0;

}
```

```
        .text
        .file   "stackFun.c"
        .globl  add                             # -- Begin >
        .type   add,@function
add:                                    # @add
        .cfi_startproc
# %bb.0:
                                        # kill: def $esi ki>
                                        # kill: def $edi ki>
        leal    (%rdi,%rsi), %eax
        retq
.Lfunc_end0:
        .size   add, .Lfunc_end0-add
        .cfi_endproc
                                        # -- End function
        .globl  main                            # -- Begin >
        .type   main,@function
main:                                   # @main
        .cfi_startproc
# %bb.0:
        xorl    %eax, %eax
        retq
.Lfunc_end1:
        .size   main, .Lfunc_end1-main
        .cfi_endproc
                                        # -- End function
        .ident  "clang version 14.0.6 (https://github.com/l>
        .section        ".note.GNU-stack","",@progbits
        .addrsig
```

TECH GIANTS MOVING TO OPEN SOURCE



Press Note

Qualcomm to Bring RISC-V Based Wearable Platform to Wear OS by Google

Important first milestone to bring RISC-V compatible CPUs to the Ecosystem

OCT 17, 2023 | SAN DIEGO | Qualcomm products mentioned within this press release are offered by Qualcomm Technologies, Inc. and/or its subsidiaries.

FUN DISCUSSION ON THE FUTURE OF OPEN SOURCE, COPYRIGHTS AND PATENTS

Show companies be able to patent ISA, our architecture

1. Apple M1 (protected architecture)
2. Intel x86 (protected architecture)
3. Arm (Who company based on licensing an architecture to other companies like Qualcomm)
4. Risc-v (Research group at Berkley Open source architecture)

A CASE FOR THE VALUE OF UNIVERSITIES AND THERE CONTRIBUTION TO THE TECH STACK

What will you add to the tech stack?

Operating Systems	**OpenBSD**	Apple's macOS and iOS, which derived from them Open BSD which was forked from NetBSD Developed at Berkeley
Compiler	Llvm/clang	University of Illinois at Urbana–Champaign
Processor	Risc-v	University of California, Berkeley