COMPUTER SYSTEMS AND ORGANIZATION

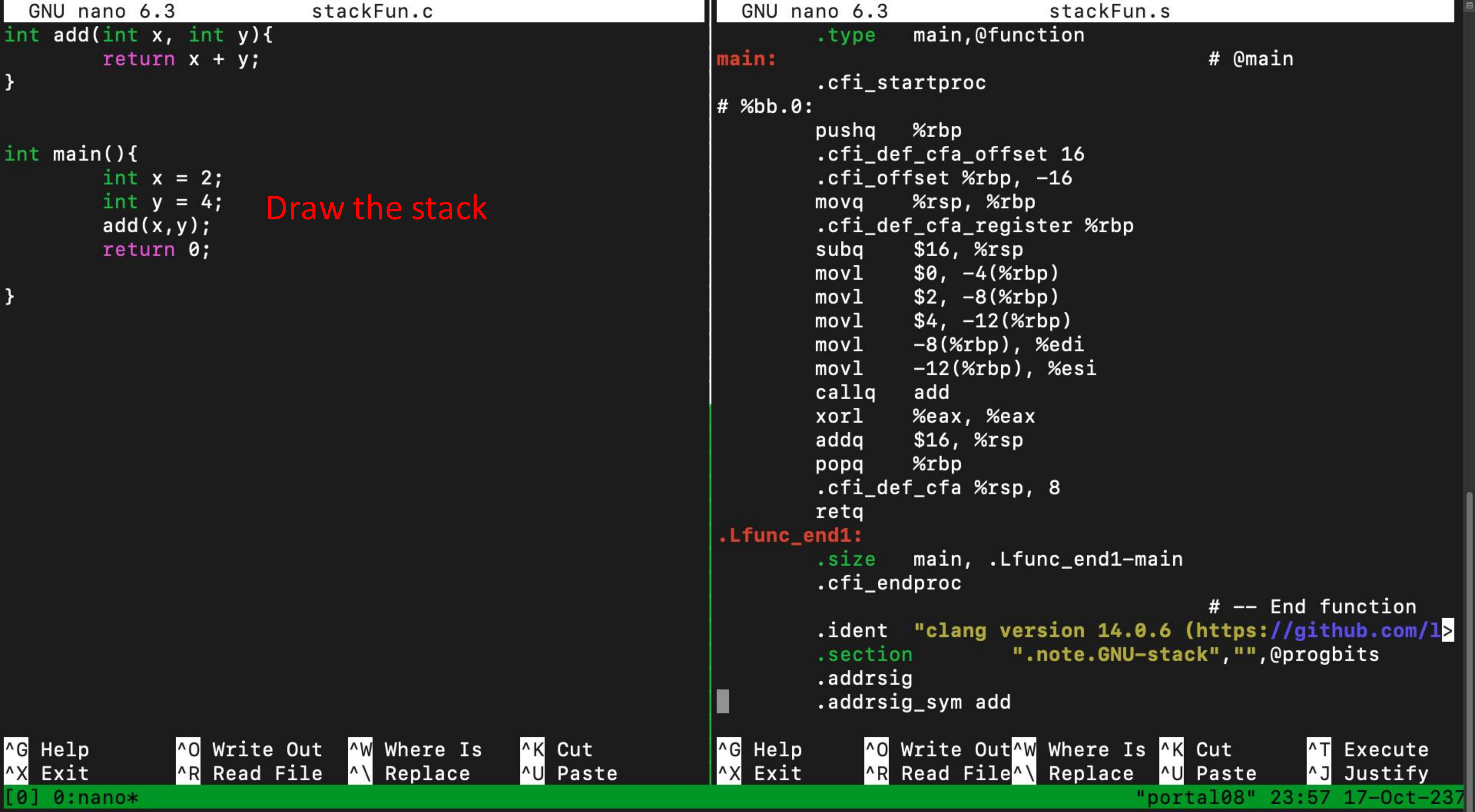
Part 1

Daniel G. Graham Ph.D

UNIVERSITY of VIRGINIA | ENGINEERING

Contents

1. Some more assembly examples
2. Work on past exam questions
3. Jmp Tables.
4. Only 16 lectures left. We'll close out with C

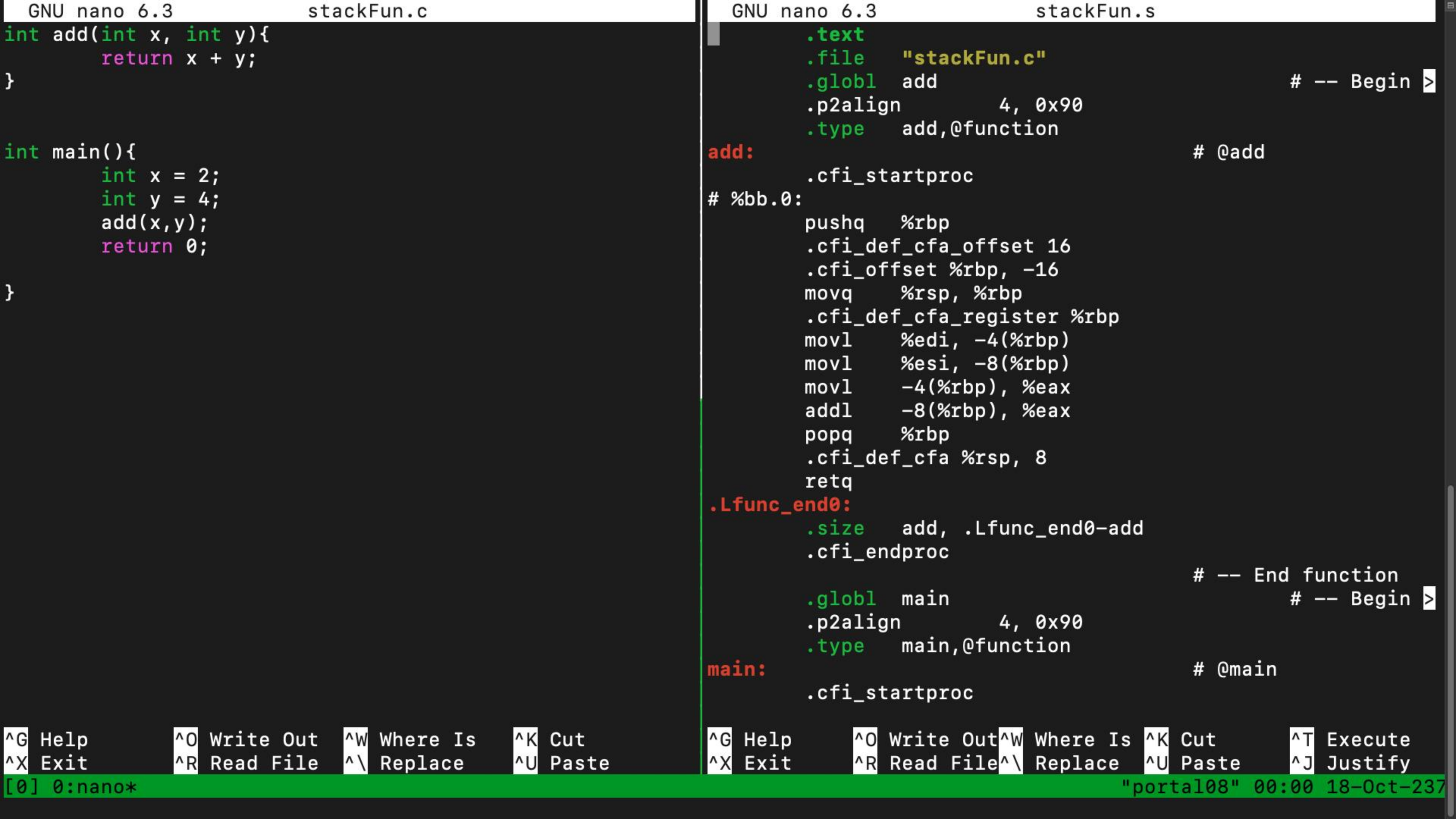


```c
int add(int x, int y){
        return x + y;
}


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

}
```

Draw the stack

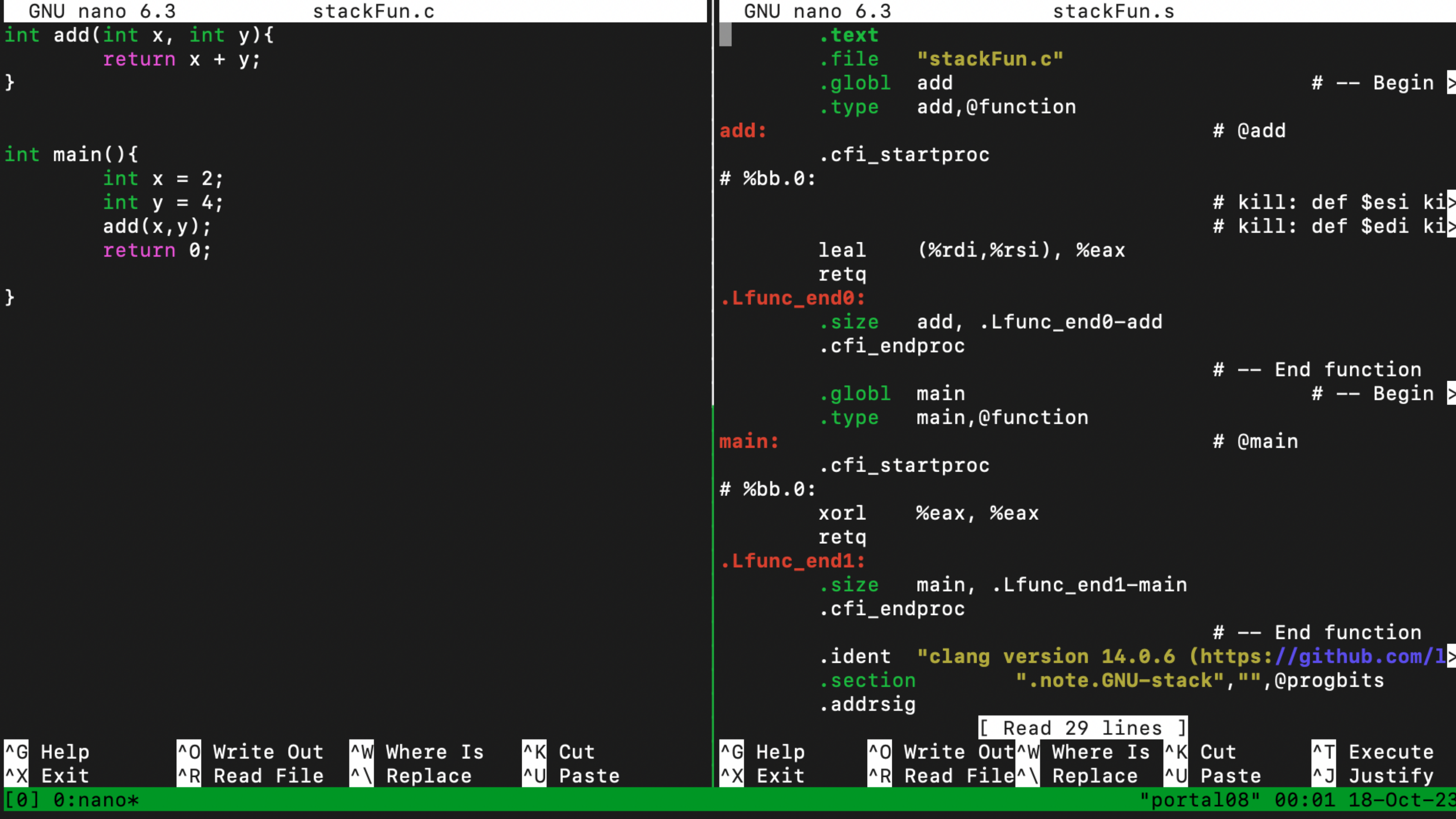
```asm
        .type   main,@function
main:                                   # @main
        .cfi_startproc
# %bb.0:
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register %rbp
        subq    $16, %rsp
        movl    $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
```

^G Help ^O Write Out ^W Where Is ^K Cut
^X Exit ^R Read File ^\ Replace ^U Paste
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify
[0] 0:nano* "portal08" 23:57 17-Oct-237



```
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
```



GNU nano 6.3 stackFun.c

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


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

}
```

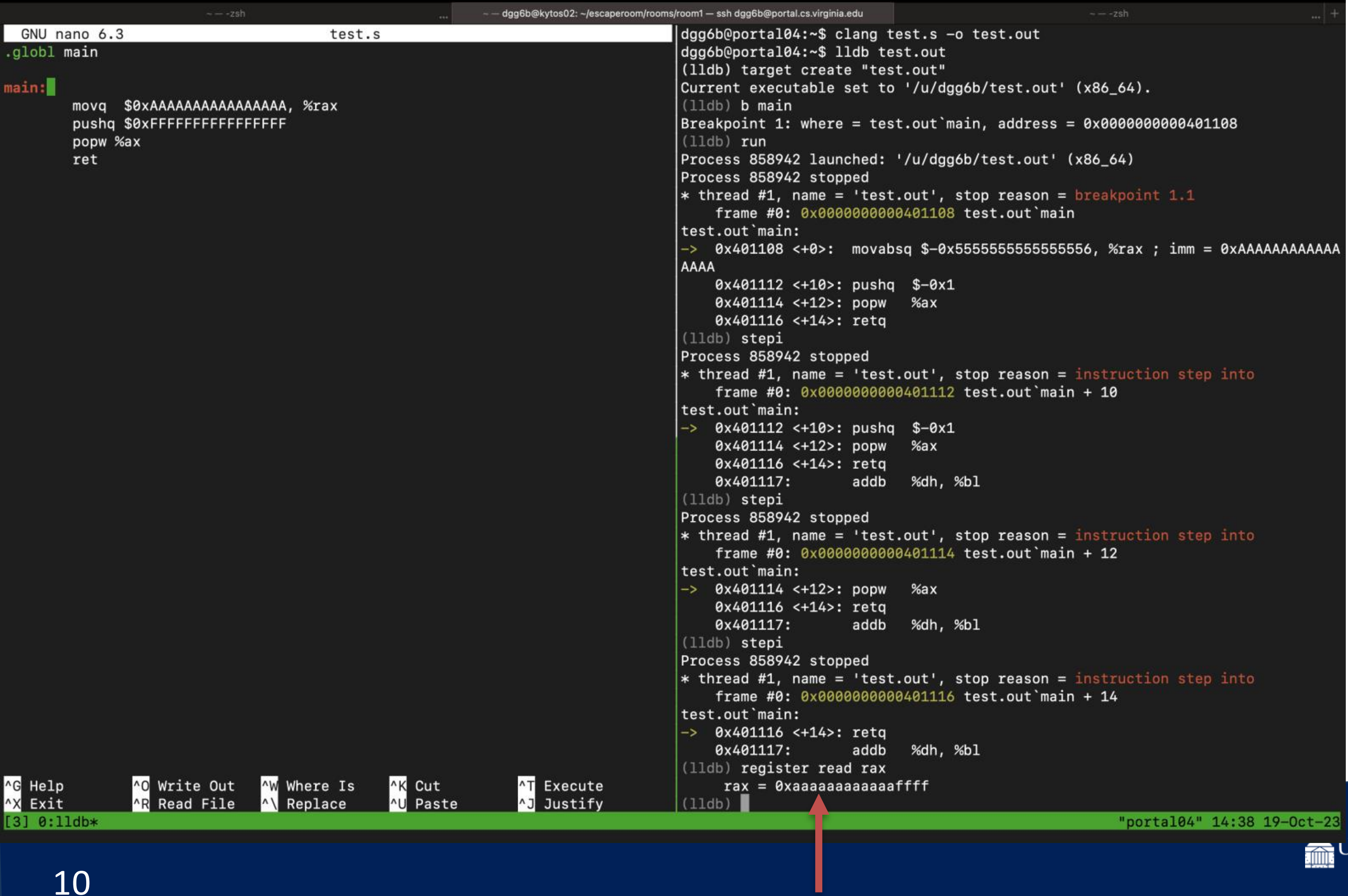
GNU nano 6.3 stackFun.s

```asm
        .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
```

[Read 29 lines]

[0] 0:nano* "portal08" 00:01 18-Oct-23

32 BITS OR BELOW



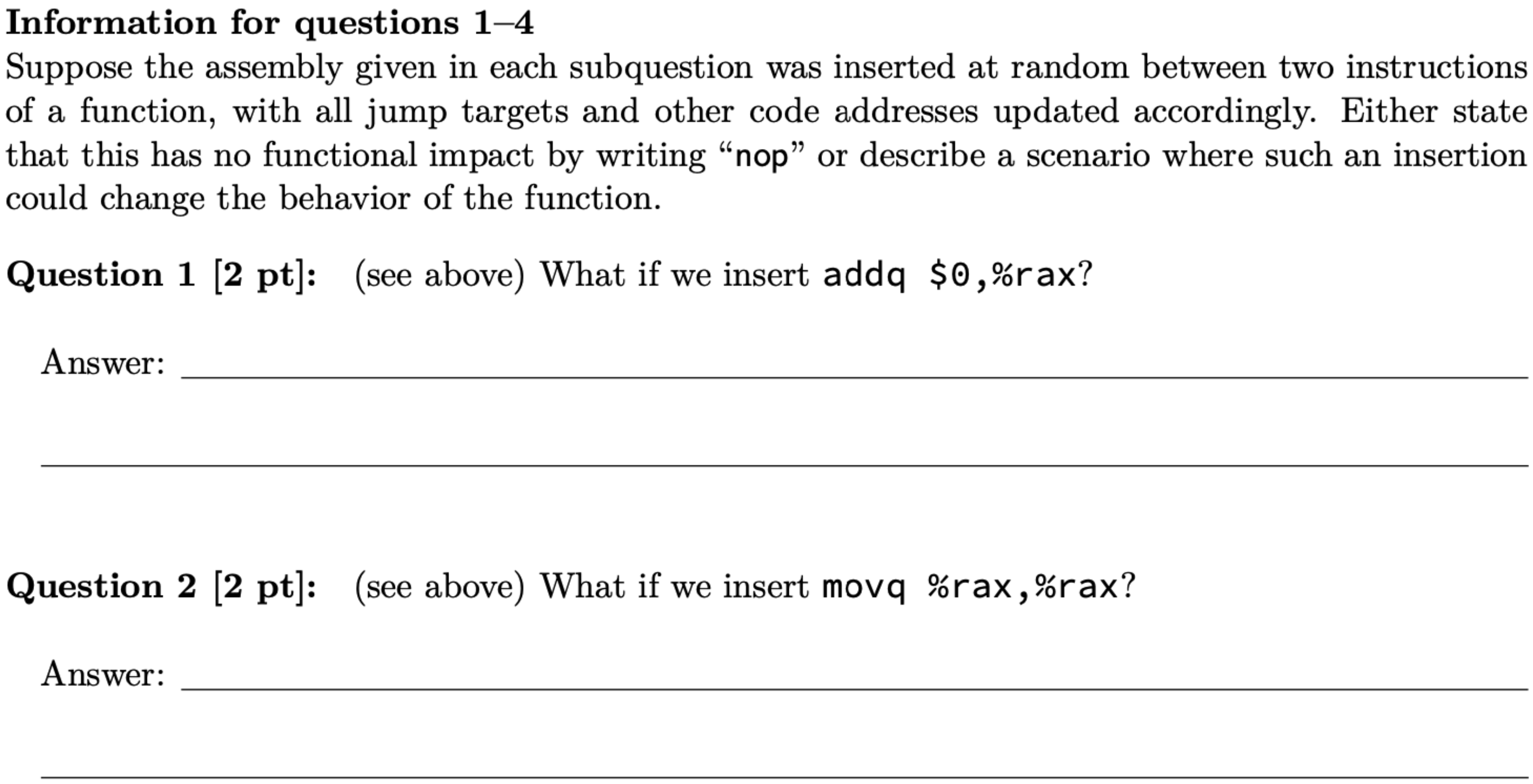
~ — -zsh
~ — dgg6b@kytos02: ~/escaperoom/rooms/room1 — ssh dgg6b@portal.cs.virginia.edu
~ — -zsh

```
  GNU nano 6.3                      test.s
.globl main

main:
        movq  $0xAAAAAAAAAAAAAAAA, %rax
        pushq $0xFFFFFFFFFFFFFFFF
        popw %ax
        ret

^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify
```

```
dgg6b@portal04:~$ clang test.s -o test.out
dgg6b@portal04:~$ lldb test.out
(lldb) target create "test.out"
Current executable set to '/u/dgg6b/test.out' (x86_64).
(lldb) b main
Breakpoint 1: where = test.out`main, address = 0x0000000000401108
(lldb) run
Process 858942 launched: '/u/dgg6b/test.out' (x86_64)
Process 858942 stopped
* thread #1, name = 'test.out', stop reason = breakpoint 1.1
    frame #0: 0x0000000000401108 test.out`main
test.out`main:
->  0x401108 <+0>:  movabsq $-0x5555555555555556, %rax ; imm = 0xAAAAAAAAAAAA
AAAA
    0x401112 <+10>: pushq  $-0x1
    0x401114 <+12>: popw   %ax
    0x401116 <+14>: retq
(lldb) stepi
Process 858942 stopped
* thread #1, name = 'test.out', stop reason = instruction step into
    frame #0: 0x0000000000401112 test.out`main + 10
test.out`main:
->  0x401112 <+10>: pushq  $-0x1
    0x401114 <+12>: popw   %ax
    0x401116 <+14>: retq
    0x401117:       addb   %dh, %bl
(lldb) stepi
Process 858942 stopped
* thread #1, name = 'test.out', stop reason = instruction step into
    frame #0: 0x0000000000401114 test.out`main + 12
test.out`main:
->  0x401114 <+12>: popw   %ax
    0x401116 <+14>: retq
    0x401117:       addb   %dh, %bl
(lldb) stepi
Process 858942 stopped
* thread #1, name = 'test.out', stop reason = instruction step into
    frame #0: 0x0000000000401116 test.out`main + 14
test.out`main:
->  0x401116 <+14>: retq
    0x401117:       addb   %dh, %bl
(lldb) register read rax
     rax = 0xaaaaaaaaaaaaffff
(lldb)
```

```
[3] 0:lldb*                                                    "portal04" 14:38 19-Oct-23
```

Notice that the top section of the register is preserved.



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`?

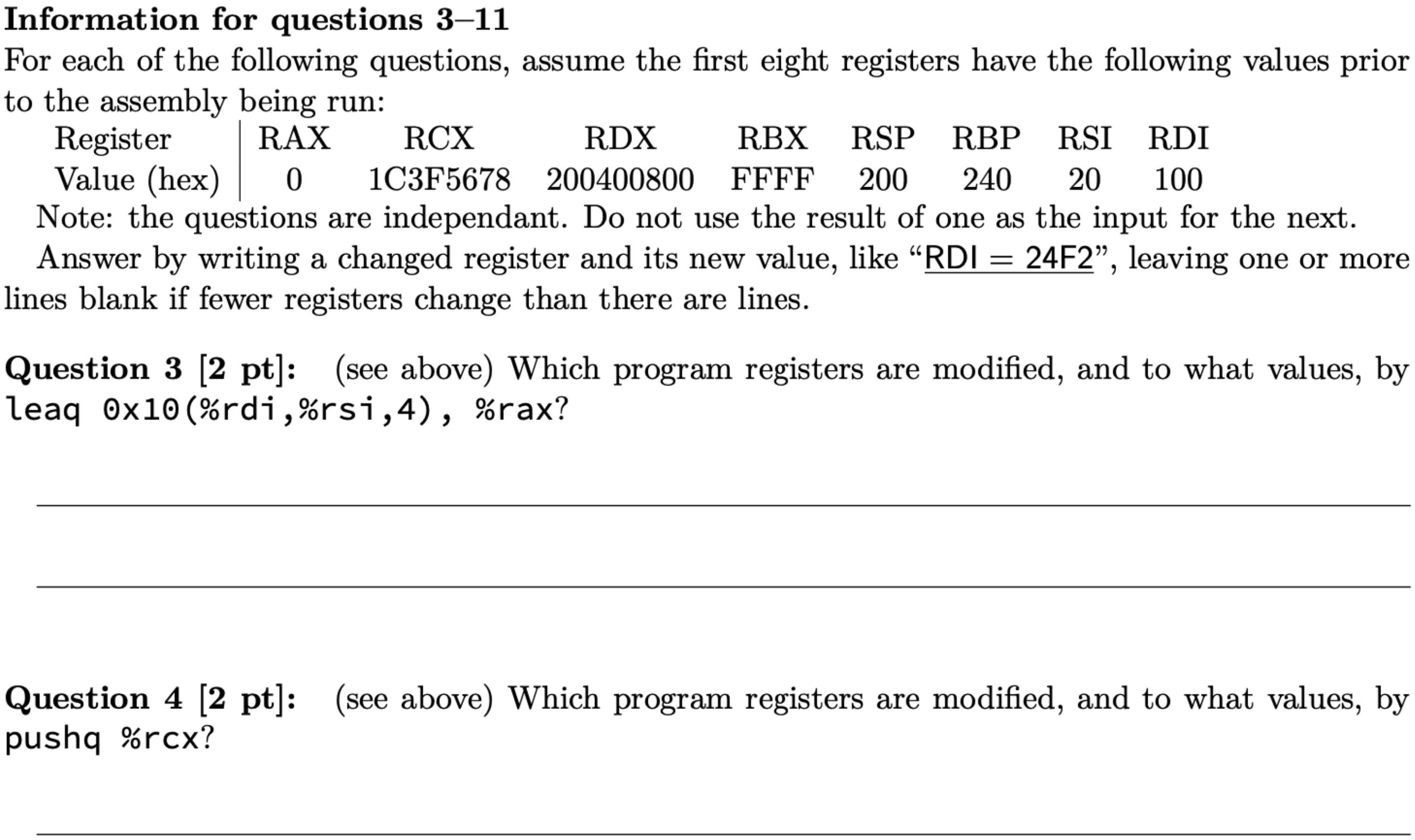
Answer: ____

____

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:

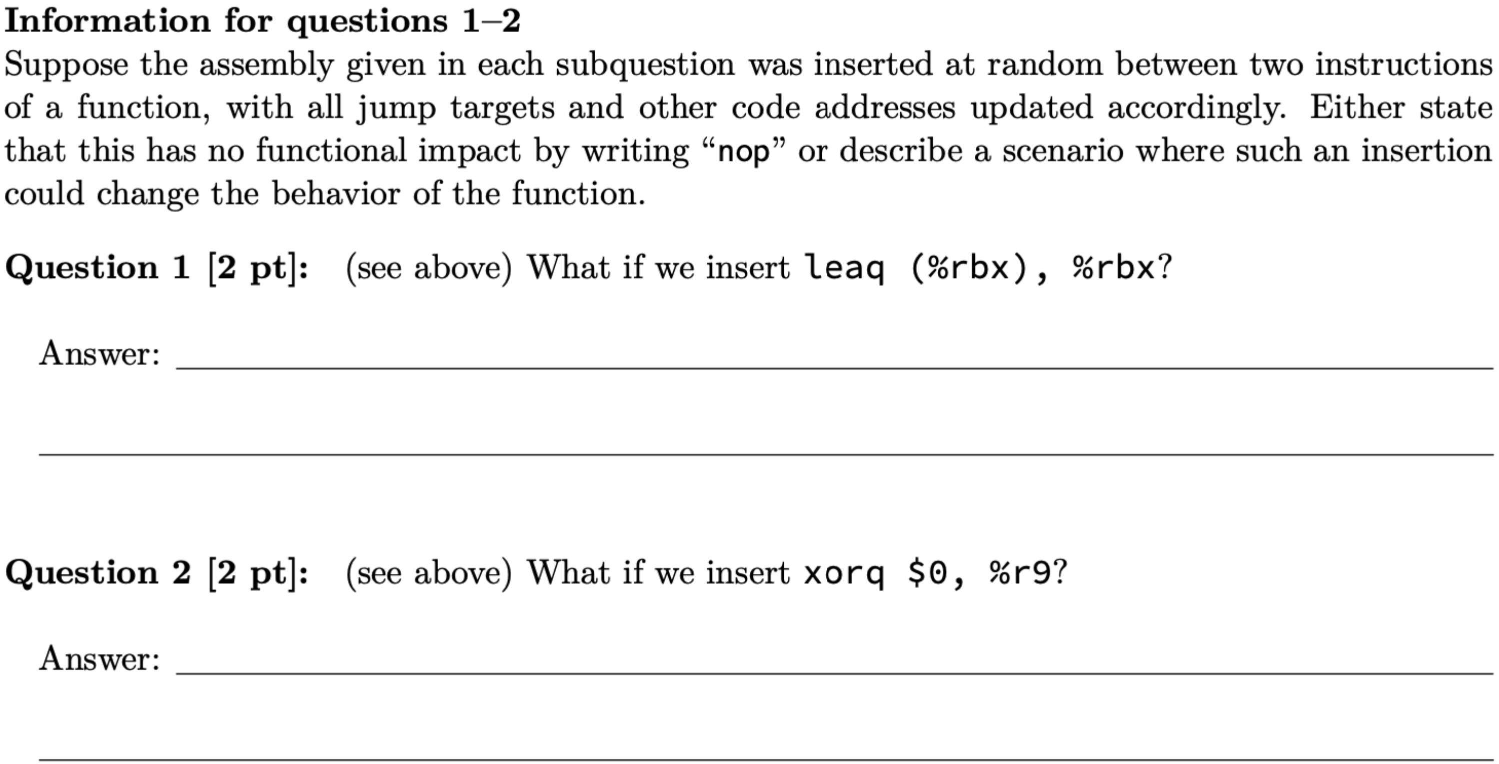
Register	RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI
Value (hex)	0	1C3F5678	200400800	FFFF	200	240	20	100

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 __________;
}
```

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

🤔

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

  return  param1 * 2;
}
```

```
00000000004004d6 <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

SWITCH STATEMENT AND JUMP TABLES

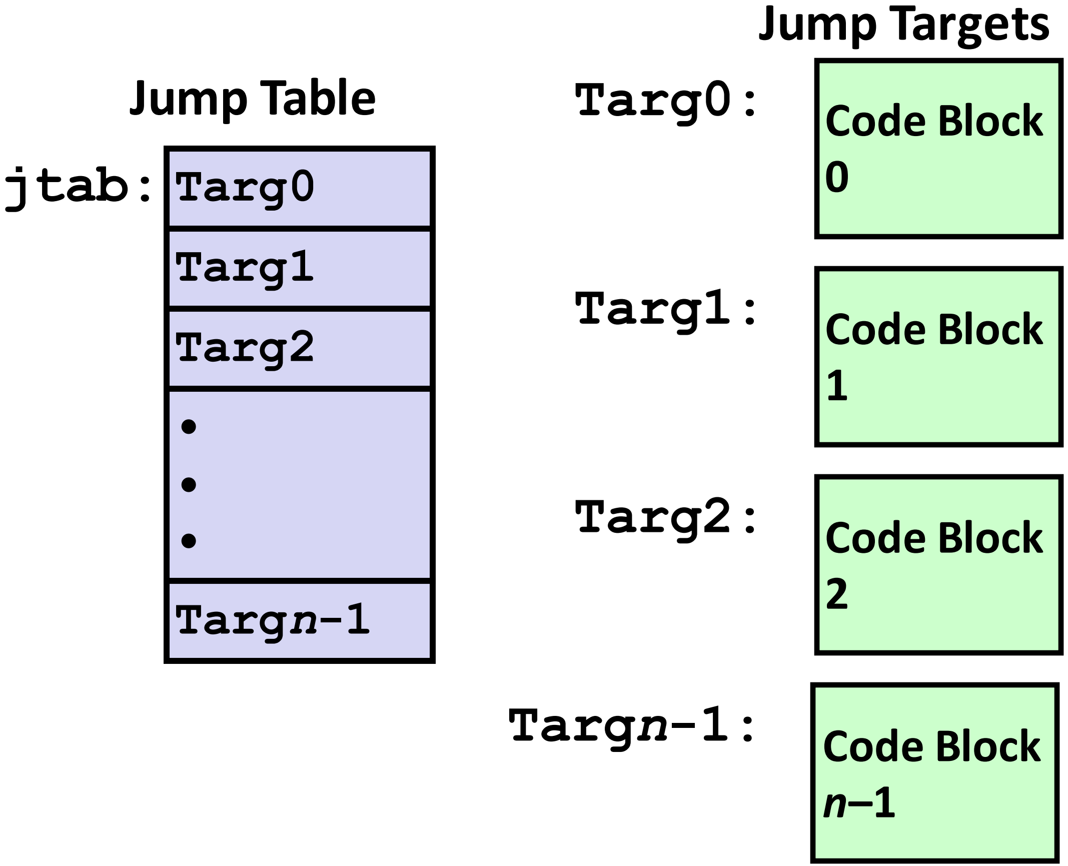
```
long switch_eg(long x, long y, long z){
    long w = 1;
    switch(x) {
    case 1:
        w = y*z;
        break;
    case 2:
        w = y/z;
        /* Fall Through */
    case 3:
        w += z;
        break;
    case 5:
    case 6:
        w -= z;
        break;
    default:
        w = 2;
    }
    return w;
}
```

SWITCH STATEMENT

- Fall through cases
  - Here: 2
- Multiple case labels
  - Here: 5 & 6
- Missing cases
  - Here: 4

Switch Form

```
switch(x) {
  case val_0:
     Block 0
  case val_1:
     Block 1
     • • •
  case val_n-1:
     Block n–1
}
```



Jump Table

jtab:

- Targ0
- Targ1
- Targ2
- •
•
•
- Targ*n*-1

Jump Targets

Targ0: Code Block 0

Targ1: Code Block 1

Targ2: Code Block 2

Targ*n*-1: Code Block *n*–1

```
long switch_eg(long x, long y, long z)
{
    long w = 1;
    switch(x) {
      . . .
    }
    return w;
}
```

Register	Use(s)
`%rdi`	Argument **`x`**
`%rsi`	Argument **`y`**
`%rdx`	Argument **`z`**
`%rax`	Return value

Setup:

What range of values takes default?

```
switch_eg:
    --SNIP--
     cmpq     $6, %rdi    # x:6
     ja       .L8
     jmp      *.L4(,%rdi,8)
```

Note that **`w`** not initialized here

```
long switch_eg(long x, long y, long z)
{
    long w = 1;
    switch(x) {
      . . .
    }
    return w;
}
```

Jump table

```
.section    .rodata
  .align 8
.L4:
  .quad     .L8   # x = 0
  .quad     .L3   # x = 1
  .quad     .L5   # x = 2
  .quad     .L9   # x = 3
  .quad     .L8   # x = 4
  .quad     .L7   # x = 5
  .quad     .L7   # x = 6
```

Setup:

```
switch_eg:
    movq    %rdx, %rcx
    cmpq    $6, %rdi          # x:6
    ja      .L8               # Use default
    jmp     *.L4(,%rdi,8)     # goto *JTab[x]
```

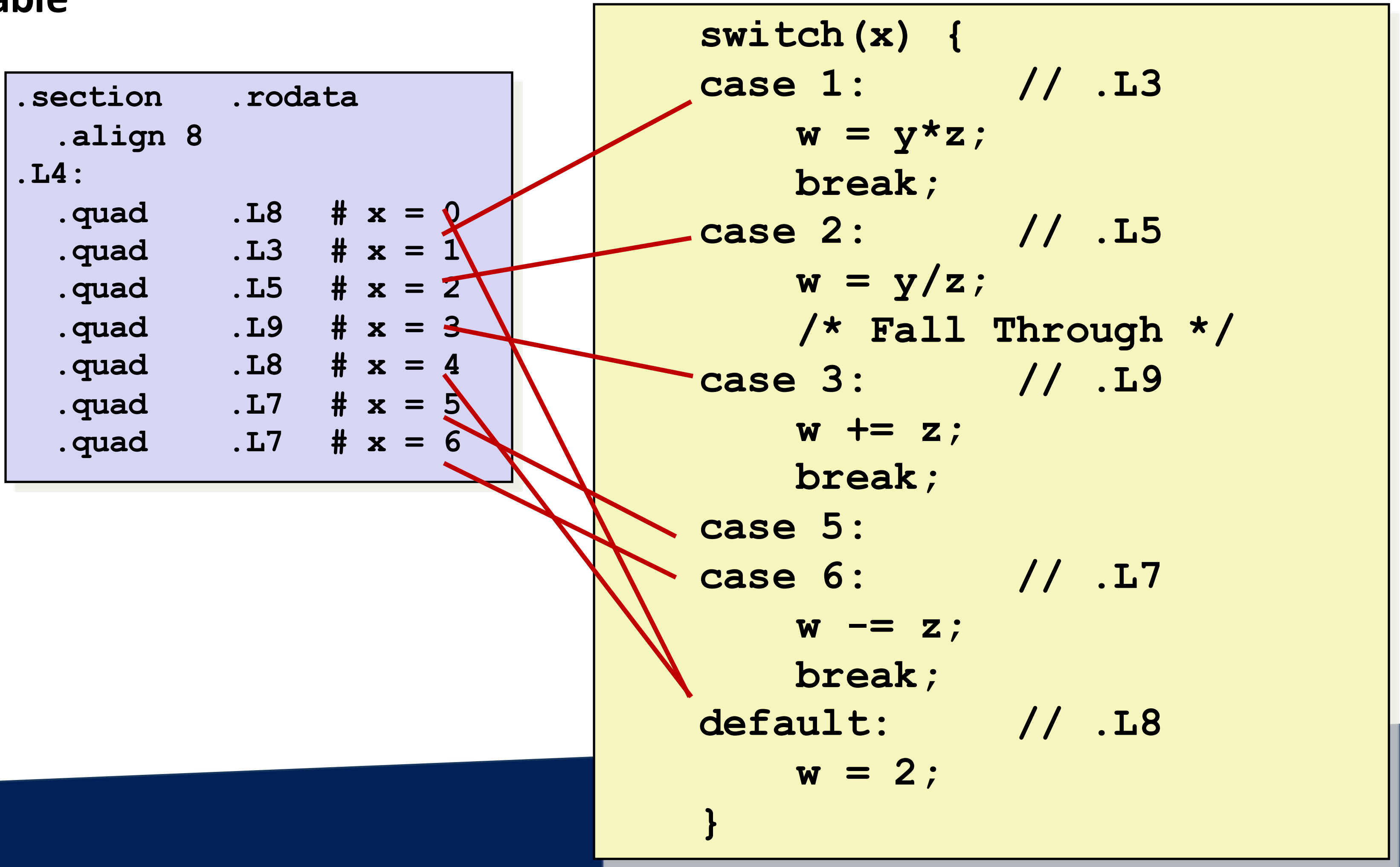
Indirect jump →

- Table Structure
  - Each target requires 8 bytes
  - Base address at **.L4**
- Jumping
  - **Direct: jmp .L8**
  - Jump target is denoted by label **.L8**
  - **Indirect: jmp *.L4(,%rdi,8)**
  - Start of jump table: **.L4**

Jump table

```
.section    .rodata
  .align 8
.L4:
  .quad     .L8   # x = 0
  .quad     .L3   # x = 1
  .quad     .L5   # x = 2
  .quad     .L9   # x = 3
  .quad     .L8   # x = 4
  .quad     .L7   # x = 5
  .quad     .L7   # x = 6
```

Jump table



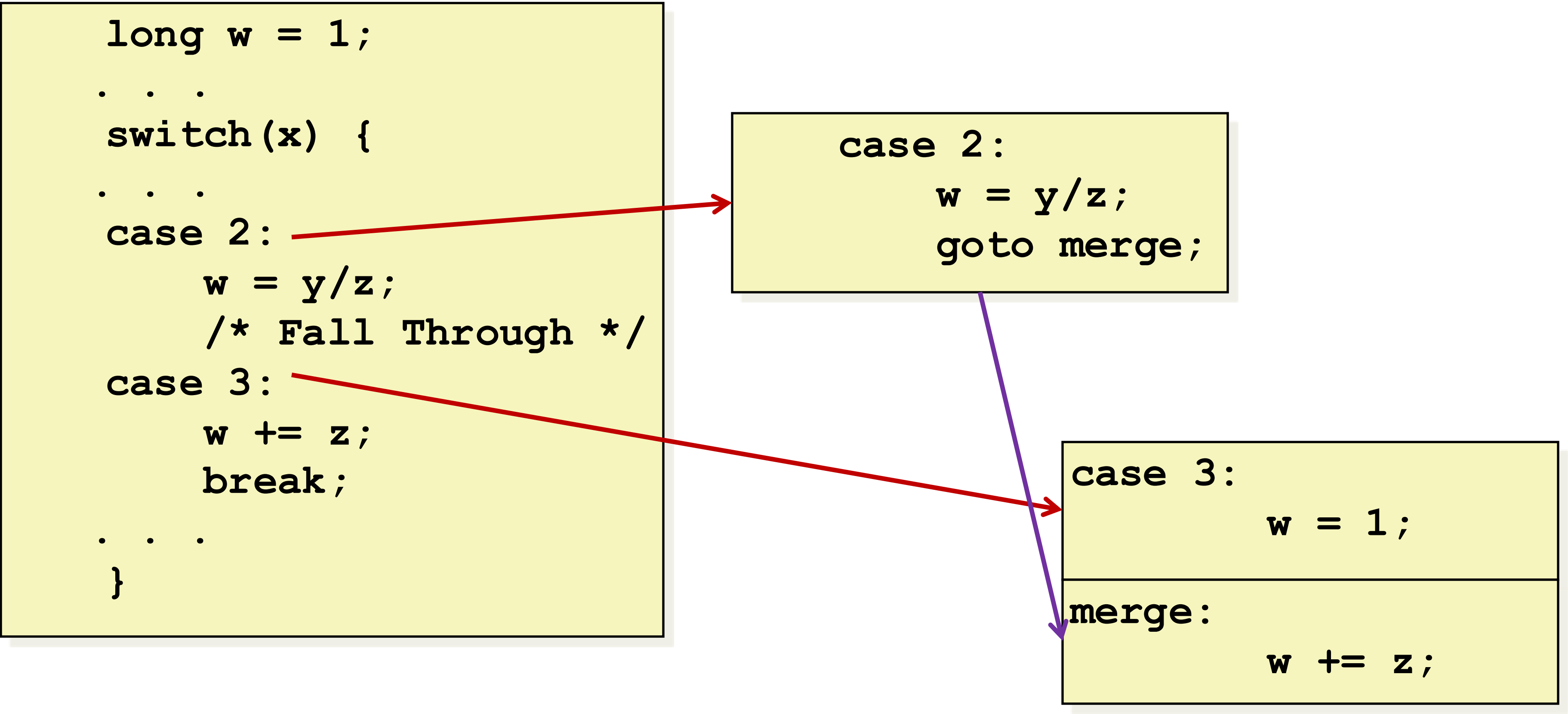
```
.section    .rodata
  .align 8
.L4:
  .quad     .L8   # x = 0
  .quad     .L3   # x = 1
  .quad     .L5   # x = 2
  .quad     .L9   # x = 3
  .quad     .L8   # x = 4
  .quad     .L7   # x = 5
  .quad     .L7   # x = 6
```

```
switch(x) {
case 1:       // .L3
    w = y*z;
    break;
case 2:       // .L5
    w = y/z;
    /* Fall Through */
case 3:       // .L9
    w += z;
    break;
case 5:
case 6:       // .L7
    w -= z;
    break;
default:      // .L8
    w = 2;
}
```

```
   switch(x) {
   case 1:    // .L3
         w = y*z;
         break;
    . . .
}
```

```
.L3:
   movq    %rsi, %rax   # y
   imulq   %rdx, %rax   # y*z
   ret
```

Register	Use(s)
%rdi	Argument **x**
%rsi	Argument **y**
%rdx	Argument **z**
%rax	Return value



```
long w = 1;
. . .
switch(x) {
. . .
case 2:
     w = y/z;
     /* Fall Through */
case 3:
     w += z;
     break;
. . .
 }
```

```
    case 2:
         w = y/z;
         goto merge;
```

```
case 3:
        w = 1;
merge:
        w += z;
```

```
    long w = 1;
   . . .
    switch(x) {
   . . .
    case 2:
        w = y/z;
        /* Fall Through */
    case 3:
        w += z;
        break;
   . . .
    }
```

```
.L5:                      # Case 2
   movq    %rsi, %rax
   cqto
   idivq   %rcx           #  y/z
   jmp     .L6            #  goto merge
.L9:                      # Case 3
   movl    $1, %eax       #  w = 1
.L6:                      # merge:
   addq    %rcx, %rax #  w += z
   ret
```

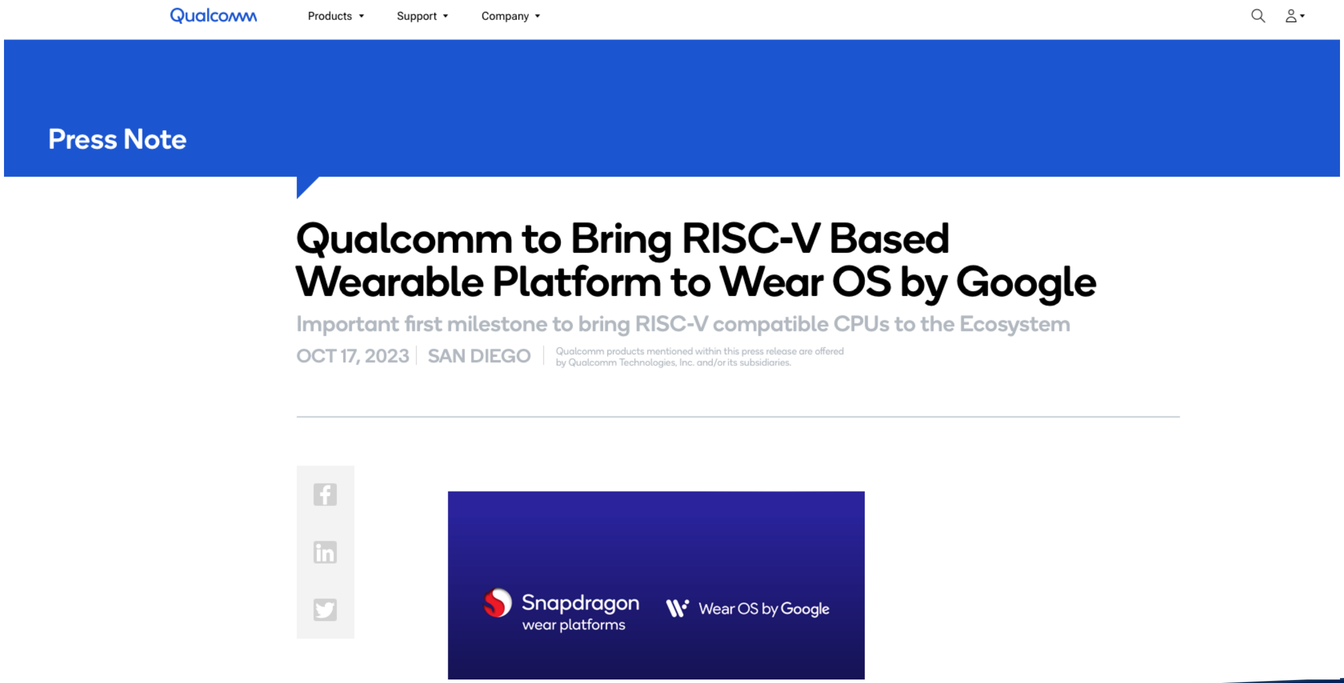
Register	Use(s)
`%rdi`	Argument **`x`**
`%rsi`	Argument **`y`**
`%rdx`	Argument **`z`**
`%rax`	Return value

```
  switch(x) {
    . . .
    case 5:  // .L7
    case 6:  // .L7
        w -= z;
        break;
    default: // .L8
        w = 2;
}
```

```
.L7:                    # Case 5,6
  movl  $1, %eax        #  w = 1
  subq  %rdx, %rax      #  w -= z
  ret
.L8:                    # Default:
  movl  $2, %eax        #  2
  ret
```

Register	Use(s)
`%rdi`	Argument **`x`**
`%rsi`	Argument **`y`**
`%rdx`	Argument **`z`**
`%rax`	Return value

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

Should companies be able to patent ISA, our architecture

1. Apple M1 (protected architecture)
2. Intel x86 (protected architecture)
3. Arm (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 THEIR CONTRIBUTION TO THE TECH STACK

What will you add to the tech stack?

Operating Systems	**OpenBSD**	Apple's macOS and iOS, which derived from the 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