Spring 2025
Exam II
2025-4-8

Instructions

This exam contains 11 pages (including this cover page) and 16 questions. It is out of 50 points.

You have **75 minutes** to complete the examination.

For this exam, you have been given a separate answer sheet to fill in your responses.

DO shade in the bubbles on your answer sheet completely without going outside the lines

DO keep written answers inside the box. *Any answers that stray outside the box will not be graded*

DO feel free to write on this exam packet

DO NOT write anything on your answer sheet except for your name, computing ID, signature, and answers in the designated areas

DO NOT use a calculator, consult notes, or collaborate with classmates

The next page contains reference material which you are welcome to tear off and refer to during the test.

Datatype Sizes and x86 Calling Convention

You may assume the following datatype sizes:

x86-64 Suffix	C Types	size in bits
`b`	`char`	8
`w`	`short`	16
`l`	`int` and `float`	32
`q`	`long`, `double`, and all pointer types	64

Function arguments are in (in order) `%rdi`, `%rsi`, `%rdx`, `%rcx`, `%r8`, `%r9`; return values are in `%rax`.

The following are **callee-save** registers: `%rbx`, `%rsp`, `%rbp`, and `%r12` through `%r15`

The following are **caller-save** registers: `%rax`, `%rcx`, `%rdx`, `%rsi`, `%rdi`, and `%r8` through `%r11`

1 Assembly Questions

Questions 1-3 use the following disassembly of a binary. Assume this is running on a typical 64-bit system. Assume values are stored on the stack in little endian.

```
0000000000401106 <main>:
  401106: 49 c7 c4 19 00 00 00  mov    $0x19,%r12
  40110d: 48 c7 c7 05 00 00 00  mov    $0x5,%rdi
  401114: 48 c7 c6 07 00 00 00  mov    $0x7,%rsi
  40111b: 48 c7 c2 01 00 00 00  mov    $0x1,%rdx
  401122: 48 c7 c1 02 00 00 00  mov    $0x2,%rcx
  401129: e8 0c 00 00 00        call   40113a <addtwice>
  40112e: 49 29 c4              sub    %rax,%r12
  401131: 4d 6b e4 02           imul   $0x2,%r12,%r12
  401135: 49 83 c4 0a           add    $0xa,%r12
  401139: c3                    ret

000000000040113a <addtwice>:
  40113a: 41 54                 push   %r12
  40113c: 41 50                 push   %r8
  40113e: 49 89 fc              mov    %rdi,%r12
  401141: 49 01 f4              add    %rsi,%r12
  401144: 49 89 d0              mov    %rdx,%r8
  401147: 49 01 c8              add    %rcx,%r8
  40114a: 4d 01 c4              add    %r8,%r12
  40114d: 4c 89 e0              mov    %r12,%rax
  401150: 41 58                 pop    %r8
  401152: c3                    ret
```

1. (5 points) Immediately after executing the `call 40113a <addtwice>` instruction, but before executing the first instruction of the `addtwice` function, what is the **32-bit** value that is pointed to by `%rsp`?

   (a) `40112e`
   (b) `401129`
   (c) `40113a`
   (d) `401152`

   Solution:

   a 0x2e114000 or big endian acceptable as well.

2. (3 points) Currently, the program will crash with a segmentation fault. Which of the following changes could allow it to run without a segmentation fault (the program doesn't necessarily have to function normally).

   (a) Insert a `push %rdi` before the `call` instruction and a corresponding `pop %rdi` afterwards.

(b) Add a `pop %r12` after the `pop %r8` in the `addtwice` subroutine.

(c) Remove the `pop %r8` instruction in the `addtwice` subroutine.

(d) None of the above.

Solution: (b)

3. (4 points) Which condition codes are set after the execution of the following instructions. **Select all that apply.**

   Hint: It may be helpful to write out the arithmetic in binary two's complement.

```
movq $-10, %rax
add $4, %rax
jl somefunc
```

   (a) `SF`

   (b) `ZF`

   (c) `OF`

   (d) `CF`

Solution: (a)

4. (5 points) Consider the following snippet of x86 Assembly:

```
pushw $1
pushw $2
pushw $3
pushw $4
popq %rax
```

   What will be the value of `%rax` after the code executes?

   (a) `0x0000000000000004`

   (b) `0x0004000300020001`

   (c) `0x4000000000000000`

   (d) `0x0001000200030004`

Solution: (d) `0x0001000200030004`

Consider the following assembly for questions 6-8, "..." represents additional code.

```
.JumpTable:
    .quad case1
    .quad case2
    .quad case3
    .quad case4
    .quad case4
    .quad case1
somefunc:
    cmpq $5, %rdi
    ja case1       # jump if above zero in an unsigned comparison
    jmp *.JumpTable(,%rdi,8)
case2:
    movq %rdi, %rax
    imulq $2, %rax
    retq
case3:
    movq %rdi, %rax
    addq $5, %rax
case4:
    movl $1, %eax
    subq %rdi, %rax
    retq
case1:
    movl $2, %eax
    retq
```

5. (4 points) Which code block will execute if the value of `%rdi` is 3 at the start of the execution of `somefunc`?

   (a) `case1`

   (b) `case2`

   (c) `case3`

   (d) `case4`

Solution: (d) `case4`

6. (5 points) What will the decimal value of `%rax` be at the termination of somefunc if `%rdi` is 2 at the start of the execution of `somefunc`?

   (a) 5

   (b) 7

   (c) 6

   (d) -1

(d) 0

Solution: -1

Assume the first 8 registers and the given segment of (little-endian) memory have the following initial values:

Register	Value
rax	0x2130
rcx	0x5
rdx	0x64
rbx	0x101
rsp	0x700000
rbp	0x8
rsi	0x700009
rdi	0x12345678ABCD

Memory Address	Value
0x700000	0xFE
0x700001	0x07
0x700002	0x09
0x700003	0xE7
0x700004	0xA1
0x700005	0x13
0x700006	0x77
0x700007	0x88

Memory Address	Value
0x700008	0xAB
0x700009	0xCD
0x70000A	0x86
0x70000B	0x64
0x70000C	0x01
0x70000D	0x28
0x70000E	0x72
0x70000F	0x45

Each instruction below is independent; do not use the result of one as the input for the next.

Note: `%sil` is the lowest byte of `%rsi`.

For example, if `rdi` becomes `0x64`, you would bubble in the combination `d` and `e`.

If there is no change, you would just bubble in `a`.

If `rsi` becomes `0x64`, you would just bubble in `e`, since both options correspond to `e`.

Register

(a) `No Change`
(b) `rdx`
(c) `rbx`
(d) `rdi`
(e) `rsi`

Value

(a) `0x2130`
(b) `0x70000E`
(c) `0x700009`
(d) `0x457228016486CDAB`
(e) `0x64`

7. (2 points) `movb 2(%rsp), %sil`

Solution:

(A) No change ✓

(B) Register: %_____ (B1)

New value: _____ (B2)

8. (2 points) `cmpq %rdi, %rsi`

Solution:

(A) No change ✓

(B) Register: %_____ (B1)

New value: _____ (B2)

9. (2 points) `leaq (%rsi, %rcx), %rbx`

Solution:

(A) No change _

(B) Register: %__**rbx**__ (B1)
New value: **0x70000E** (B2)

10. (2 points) `movq (%rsp, %rbp), %rdx`

Solution:

(A) No change _

(B) Register: %__**rdx**__ (B1)
New value: **0x457228016486CDAB** (B2)

11. (2 points) `testq %rax, %rax`

Solution:

(A) No change ✓

(B) Register: %_____ (B1)
New value: _____ (B2)

12. (2 points) `movl %eax, %edi`

Solution:

(A) No change _

(B) Register: %__**rdi**__ (B1)
New value: **0x2130** (B2)

2 C Questions

Consider the following snippet of C code:

```
unsigned long a = 0x20;     // originally stored in memory at 0x18
unsigned long x = 0x28;     // originally stored in memory at 0x20
unsigned long *b = &a;      // originally stored in memory at 0x28
unsigned long *y = &x;      // originally stored in memory at 0x30
unsigned long **c = &b;     // originally stored in memory at 0x38
unsigned long **z = &y;     // originally stored in memory at 0x40
z = (unsigned long **)x;
**c += 2;
*y -= 1;
y--;
*b = *y - **z;
```

What are the values of `a`, `x`, `b`, and `&c` after the code executes?

13. (3 points) What is the value of a?

(a) `0x20`
(b) `0x18`
(c) `0x00`
(d) `0x22`

14. (3 points) What is the value of x?

(c) `0x00`
(b) `0x27`
(c) `0x18`
(d) `0x21`

15. (3 points) What is the value `b`?

(a) `0x08`
(b) `0x00`
(c) `0x28`
(d) `0x18`

16. (3 points) What is the value `&c`?

(a) `0x22`
(b) `0x18`
(c) `0x28`
(d) `0x38`

Solution:

```
a = 0x00
x = 0x27
b = 0x00
&c = 0x38
```

Solution:

```
b = {1, -1, 6, 1, 5, 1}
```