CSO-Recitation 08

CSCI-UA 0201-007

R08: Assessment 07 & Assembly

Today’s Topics

- Assessment 07
- Some exercises
  - give some senses about lab3

Q1 Set_five

Given the following C function from Lab 1,

```c
void set_five(int *p)
{
  *p = 5;
}

void test()
{
  int p = 0;
  set_five(&p);
}
```

The assembly for `set_five` function is:

```
0x00000000000005fa <+0>: ???
0x0000000000000600 <+6>: retq
```

The assembly for `test` function is:

```
0x0000000000000601 <+0>:     sub    $0x10,%rsp
0x0000000000000605 <+4>:     movl   $0x0,0xc(%rsp)
0x000000000000060d <+12>:    lea    0xc(%rsp),%rdi
0x0000000000000612 <+17>:    callq  0x5fa <set_five>
0x0000000000000617 <+22>:    add    $0x10,%rsp
0x000000000000061b <+26>:    retq
```

Q1.1 %rsp

Under normal program execution, suppose the value of %rsp is `0x7fff856001d8` **just prior to** executing the first instruction of `test`. What is the value of %rsp just prior to executing the first instruction of `set_five`?

A: 0x7fff856001d8

The assembly for `set_five` function is:

```
0x00000000000005fa <+0>: ???
0x0000000000000600 <+6>: retq
```

The assembly for `test` function is:

```
0x0000000000000601 <+0>:     sub    $0x10,%rsp
0x0000000000000605 <+4>:     movl   $0x0,0xc(%rsp)
0x000000000000060d <+12>:    lea    0xc(%rsp),%rdi
0x0000000000000612 <+17>:    callq  0x5fa <set_five>
0x0000000000000617 <+22>:    add    $0x10,%rsp
0x000000000000061b <+26>:    retq
```

Q1.1 %rsp

Under normal program execution, suppose the value of %rsp is `0x7fff856001d8` **just prior to** executing the first instruction of `test`. What is the value of %rsp just prior to executing the first instruction of `set_five`?

A: 0x7fff856001d8

The assembly for `set_five` function is:

```
0x00000000000005fa <+0>: ???
0x0000000000000600 <+6>: retq
```

The assembly for `test` function is:

```
0x0000000000000601 <+0>:     sub    $0x10,%rsp
0x0000000000000605 <+4>:     movl   $0x0,0xc(%rsp)
0x000000000000060d <+12>:    lea    0xc(%rsp),%rdi
0x0000000000000612 <+17>:    callq  0x5fa <set_five>
0x0000000000000617 <+22>:    add    $0x10,%rsp
0x000000000000061b <+26>:    retq
```

Q1.1 %rsp

Under normal program execution, suppose the value of %rsp is `0x7fff856001d8` **just prior to** executing the first instruction of `test`. What is the value of %rsp just prior to executing the first instruction of `set_five`?

A: 0x7fff856001d8

The assembly for `set_five` function is:

```
0x00000000000005fa <+0>: ???
0x0000000000000600 <+6>: retq
```

The assembly for `test` function is:

```
0x7fff856001c8 + 0xc
= 0x7fff856001d4
```

```
0x0000000000000601 <+0>:    sub    $0x10,%rsp
0x0000000000000605 <+4>:    movl   $0x0,0xc(%rsp)
0x000000000000060d <+12>:   lea    0xc(%rsp),%rdi
0x0000000000000612 <+17>:   callq  0x5fa <set_five>
0x0000000000000617 <+22>:   add    $0x10,%rsp
0x000000000000061b <+26>:   retq
```

Q1.1 %rsp

Under normal program execution, suppose the value of %rsp is `0x7fff856001d8` **just prior to** executing the first instruction of `test`. What is the value of %rsp just prior to executing the first instruction of `set_five`?

A: 0x7fff856001d8

The assembly for `set_five` function is:

```
0x00000000000005fa <+0>: ???
0x0000000000000600 <+6>: retq
```

The assembly for `test` function is:

```
0x0000000000000601 <+0>:    sub    $0x10,%rsp
0x0000000000000605 <+4>:    movl   $0x0,0xc(%rsp)
0x000000000000060d <+12>:   lea    0xc(%rsp),%rdi
0x0000000000000612 <+17>:   callq  0x5fa <set_five>
0x0000000000000617 <+22>:   add    $0x10,%rsp
0x000000000000061b <+26>:   retq
```

```
0x7fff856001c8 + 0xc
= 0x7fff856001d4
= %rdi
```

Q1.1 %rsp

Under normal program execution, suppose the value of %rsp is `0x7fff856001d8` **just prior to** executing the first instruction of `test`. What is the value of %rsp just prior to executing the first instruction of `set_five`?

A: 0x7fff856001d8

The assembly for `set_five` function is:

```
0x00000000000005fa <+0>: ???
0x0000000000000600 <+6>: retq
```

The assembly for `test` function is:

0x7fff856
= 0x7fff8

```
0x0000000000000601 <+0>:    sub    $0x10,%rsp
0x0000000000000605 <+4>:    movl   $0x0,0xc(%rsp)
0x000000000000060d <+12>:   lea    0xc(%rsp),%rdi
0x0000000000000612 <+17>:   callq  0x5fa <set_five>
0x0000000000000617 <+22>:   add    $0x10,%rsp
0x000000000000061b <+26>:   retq
```

1. Decrease %rsp by 8

Q1.1 %rsp

Under normal program execution, suppose the value of %rsp is `0x7fff856001d8` **just prior to** executing the first instruction of `test`. What is the value of %rsp just prior to executing the first instruction of `set_five`?

A: 0x7fff856001d8

The assembly for `set_five` function is:

```
0x00000000000005fa <+0>: ???
0x0000000000000600 <+6>: retq
```

The assembly for `test` function is:

```
0x7fff856
= 0x7fff8

0x                    $0x10,%rsp
0x                    $0x0,0xc(%rsp)
0x                    0xc(%rsp),%rdi
0x                    0x5fa <set_five>
0x0000000000000617 <+22>:    add    $0x10,%rsp
0x000000000000061b <+26>:    retq
```

Return to the next instruction after calling

1. Decrease %rsp by 8
2. Store the return address at %rsp

Q1.1 %rsp

Under normal program execution, suppose the value of %rsp is `0x7fff856001d8` **just prior to** executing the first instruction of `test`. What is the value of %rsp just prior to executing the first instruction of `set_five`?

A. 0x7fff856001d8
B. 0x7fff856001c8
C. 0x7fff856001e8
D. 0x7fff856001c0
E. 0x7fff856001d0
F. 0x7fff856001c4
G. 0x7fff856001cc
H. None of the above

Q1.2

Under normal program execution, what is the 8-byte value stored under the address specified by %rsp **just prior to** executing the first instruction of `set_five`?

A. 0x7fff856001d8
B. 0x7fff856001c0
C. 0x000000000000060d
D. 0x0000000000000612
E. 0x0000000000000617
F. It could be any arbitrary 8-byte value

Q1.3

After executing instruction 0x0000000000000600 <+6>: retq in set_five, what's new %rip value?

A. 0x000000000000060d

The assembly for `set_five` function is:

```
0x00000000000005fa <+0>: ???
0x0000000000000600 <+6>: retq
```

The assembly for `test` function is:

```
0x                                  $0x10,%rsp
0x                                  $0x0,0xc(%rsp)
0x                                  0xc(%rsp),%rdi
0x                                  0x5fa <set_five>
0x0000000000000617 <+22>:    add    $0x10,%rsp
0x000000000000061b <+26>:    retq
```

Return to the next instruction after calling

Q1.3

After executing instruction 0x0000000000000600 <+6>: retq in set_five, what's new %rip value?

A. 0x000000000000060d
B. 0x0000000000000612
C. 0x0000000000000617
D. 0x0000000000000604
E. 0x0000000000000608

Q1.4 p's location (WI

Where is the local variable p stor

A. some register
B. memory (data segment)
C. memory (stack)
D. memory (heap)

- char, int, long, … (primitive data types) =>
  - use registers whenever possible
  - stack otherwise
- local array/struct variables => stack

Given the following C function from Lab 1,

```
void set_five(int *p)
{
  *p = 5;
}

void test()
{
  int p = 0;
  set_five(&p);
}
```

The assembly for `set_five` function is:

```
0x00000000000005fa <+0>: ???
0x0000000000000600 <+6>: retq
```

The assembly for `test` function is:

```
0x0000000000000601 <+0>:    sub    $0x10,%rsp
0x0000000000000605 <+4>:    movl   $0x0,0xc(%rsp)
0x000000000000060d <+12>:   lea    0xc(%rsp),%rdi
0x0000000000000612 <+17>:   callq  0x5fa <set_five>
0x0000000000000617 <+22>:   add    $0x10,%rsp
0x000000000000061b <+26>:   retq
```

Register or stack? Look at the code.
'&p' => p is on memory => stack

Q1.5 p's location

If your answer of 1.4 is memory, where in memory (aka what address) is p stored (assuming the value of %rsp is 0x7fff856001d8 just prior to executing the first instruction of test)?

The assembly for `set_five` function is:

```
0x00000000000005fa <+0>: ???
0x0000000000000600 <+6>: retq
```

The assembly for `test` function is:

```
0x0000000000000601 <+0>:    sub    $0x10,%rsp
0x0000000000000605 <+4>:    movl   $0x0,0xc(%rsp)
0x000000000000060d <+12>:   lea    0xc(%rsp),%rdi
0x0000000000000612 <+17>:   callq  0x5fa <set_five>
0x0000000000000617 <+22>:   add    $0x10,%rsp
0x000000000000061b <+26>:   retq
```

```
void set_five(int *p)
{
  *p = 5;
}

void test()
{
  int p = 0;
  set_five(&p);
}
```

Q1.5 p's location

If your answer of 1.4 is memory, where in memory (aka what address) is p stored (assuming the value of %rsp is 0x7fff856001d8 just prior to executing the first instruction of test)?

A. 0x7fff856001d8
B. 0x7fff856001c8
C. 0x7fff856001d4
D. 0x7fff856001d0
E. 0x7fff856001cc

Q1.5 p's location

If your answer of 1.4 is memory, where in memory (aka what address) is p stored (assuming the value of %rsp is 0x7fff856001d8 just prior to executing the first instruction of test)?

The assembly for `set_five` function is:

```
0x00000000000005fa <+0>: ???
0x0000000000000600 <+6>: retq
```

The assembly for `test` function is:

```
0x7fff856001c8 + 0xc
= 0x7fff856001d4
= %rdi
```

```
0x0000000000000601 <+0>:    sub    $0x10,%rsp
0x0000000000000605 <+4>:    movl   $0x0,0xc(%rsp)
0x000000000000060d <+12>:   lea    0xc(%rsp),%rdi
0x0000000000000612 <+17>:   callq  0x5fa <set_five>
0x0000000000000617 <+22>:   add    $0x10,%rsp
0x000000000000061b <+26>:   retq
```

%rdi stores the first argument to the call

```
void set_five(int *p)
{
  *p = 5;
}

void test()
{
  int p = 0;
  set_five(&p);
}
```

Q1.5 p's location

If your answer of 1.4 is memory, where in memory (aka what address) is p stored (assuming the value of %rsp is 0x7fff856001d8 just prior to executing the first instruction of test)?

A. 0x7fff856001d8
B. 0x7fff856001c8
C. 0x7fff856001d4
D. 0x7fff856001d0
E. 0x7fff856001cc

Q1.6 set_five

What's the missing first instruction of set_five (aka the instruction corresponding to ???)

A. `movl $0x5,(%rdi)`

```
void set_five(int *p)
{
  *p = 5;
}

void test()
{
  int p = 0;
  set_five(&p);
}
```

%rdi stores the first argument to the call
%rdi=p

The assembly for `set_five` function is:

```
0x00000000000005fa <+0>: ???
0x0000000000000600 <+6>: retq
```

The assembly for `test` function is:

```
0x0000000000000601 <+0>:    sub    $0x10,%rsp
0x0000000000000605 <+4>:    movl   $0x0,0xc(%rsp)
0x000000000000060d <+12>:   lea    0xc(%rsp),%rdi
0x0000000000000612 <+17>:   callq  0x5fa <set_five>
0x0000000000000617 <+22>:   add    $0x10,%rsp
0x000000000000061b <+26>:   retq
```

Q1.6 set_five

```
void set_five(int *p)
{
  *p = 5;
}

void test()
{
  int p = 0;
  set_five(&p);
}
```

%rdi stores the first argument to the call
%rdi=p

What's the missing first instruction of set_five (aka the instruction corresponding to ???)

A. `movl $0x5,(%rdi)`
B. `movq $0x5,(%rdi)`
C. `movl $0x5,(%rsi)`
D. `movq $0x5,(%rdi)`
E. `movl $0x5, %edi`
F. `movq $0x5, %rdi`
G. `movl $0x5, %esi`
H. `movq $0x5, %rsi`

p: int * => movl
Pointer is 64-bit => %rdi

Question

- After executing instruction 0x0000000000000600 <+6>: retq in set_five, what's new %rsp value?

The assembly for `set_five` function is:

```
0x00000000000005fa <+0>: ???
0x0000000000000600 <+6>: retq
```

The assembly for `test` function is:

```
0x0000000000000601 <+0>:     sub    $0x10,%rsp
0x0000000000000605 <+4>:     movl   $0x0,0xc(%rsp)
0x000000000000060d <+12>:    lea    0xc(%rsp),%rdi
0x0000000000000612 <+17>:    callq  0x5fa <set_five>
0x0000000000000617 <+22>:    add    $0x10,%rsp
0x000000000000061b <+26>:    retq
```

Q2 cmp and set

Suppose the contents of register %rdi and register %rsi are: `%rdi=0x0000000000000002` `%rsi=0x8000000000000001`.

Consider the following 2 instruction combo:

```
cmpq %rdi, %rsi
setX %al
```

Q2.1 RFLAGS

%rdi=0x0000000000000002
%rsi=0x8000000000000001

```
cmpq %rdi, %rsi
setX %al
```

In Q2, which of the following status flags are set after executing cmpq %rdi, %rsi (aka the rest are cleared)?

A. ZF

B. SF

C. CF

D. OF

- %rsi-%rdi = 0x7fff…ff
- SF=MSB=0

dst=src-dst

cmpq `src, dst`

– Set CF, ZF, SF and OF like **subq** `src, dst` except `dst` is unchanged

flag	status
ZF (Zero Flag)	set if the result is zero.
SF (Sign Flag)	set if the result is negative.
CF (Carry Flag)	Overflow for unsigned-integer arithmetic
OF (Overflow Flag)	Overflow for signed-integer arithmetic

CF and OF

- The CPU doesn’t know if operands are signed or unsigned
- So, it calculates both the signed overflow (OF) and the unsigned overflow (CF) for each instruction
  - OF is set assuming both operands are signed
  - CF is set assuming both operands are unsigned

Q2.1 RFLAGS

In Q2, which of the following status flags are set after executing cmpq %rdi, %rsi (aka the rest are cleared)?

A. ZF

B. SF

C. CF

D. OF

- %rsi-%rdi = 0x7fff…ff
- SF=MSB=0, not set
- CF: treat as unsigned
  - => no overflow, not set
- OF: treat as signed =>
  - => overflow, set

%rdi=0x0000000000000002
%rsi=0x8000000000000001

```
cmpq %rdi, %rsi
setX %al
```

Unsigned range: 0 ~ 2^64 - 1
%rdi = 2
%rsi = 2^63 + 1
%rsi - %rdi = 2^63 – 1: in the range

Signed range: -2^63 ~ 2^63 - 1
%rdi = 2
%rsi = -2^63 + 1
%rsi - %rdi = -2^63 – 1: out of the range

Tricks to quickly decide whether the signed/unsigned computation is overflow: see the previous recitation slides.

How to decide whether there is overflow?

- Machine:
  - Unsigned: there is a carry/borrow of the MSB
  - Signed:
    - if there is carry-in but no carry-out of MSB
    - or, there is no carry-in but there’s carry out of MSB

Q2.2 Set instruction

%rdi=0x0000000000000002
%rsi=0x8000000000000001

```
cmpq %rdi, %rsi
setX %al
```

In Q2, which of the following setX instruction would result in register %al being 1 after execution?

A. sete %al
B. setne %al
C. sets %al
D. setns %al
E. setg %al
F. setge %al
G. setl %al
H. setle %al
I. seta %al
J. setb %al

setX	Condition	Description
sete	ZF	Equal / Zero
setne	~ZF	Not Equal / Not Zero
sets	SF	Negative
setns	~SF	Nonnegative
setg	~(SF^OF) &~ZF	Greater (Signed)
setge	~(SF^OF)	Greater or Equal (Signed)
setl	(SF^OF)	Less (Signed)
setle	(SF^OF) | ZF	Less or Equal (Signed)
seta	~CF&~ZF	Above (unsigned)
setb	CF	Below (unsigned)

b >= a
rsi >= rdi, under signed interpretation

Exercise

%rdi=0x8000000000000002
%rsi=0x8000000000000001

```
cmpq %rdi, %rsi
setX %al
```

Which of the following status flags are set after executing cmpq %rdi, %rsi (aka the rest are cleared)?

A. ZF

B. SF

C. CF

D. OF

%rsi - %rdi = 0xffffffffffffffff
SF = MSB = 1
CF = 1

Q3 Test and set

Suppose the content of register %rsi is `%rsi=0x8000000000000001`.

Consider the following 2 instruction combo:

```
testq %rsi, %rsi
setX %al
```

%rsi=0x8000000000000001

```
testq %rsi, %rsi
setX %al
```

Q3.1 RFLAGS

In Q3, which of the following status flags are set after executing testq %rsi, %rsi (aka the rest are cleared)?

A. ZF

B. SF

C. CF

D. OF

- %rsi and %rdi =0x80…01
- SF=MSB=1
- CF, OF: test/and clears CF, OF

testq src, dst

– Set ZF, SF like **andq src, dst** except dst is unchanged

flag	status
ZF (Zero Flag)	set if the result is zero.
SF (Sign Flag)	set if the result is negative.
CF (Carry Flag)	Overflow for unsigned-integer arithmetic
OF (Overflow Flag)	Overflow for signed-integer arithmetic

Q3.2 Set instruction

ZF=0, SF=1, CF=0, OF=0

- In Q3, which of the following setX instruction would result in register %al being 1 after execution?

A. sete %al
B. setne %al
C. sets %al
D. setns %al
E. setg %al
F. setge %al
G. setl %al
H. setle %al
I. seta %al
J. setb %al

setX	Condition	Description
sete	ZF	Equal / Zero
setne	~ZF	Not Equal / Not Zero
sets	SF	Negative
setns	~SF	Nonnegative
setg	~ (SF^OF) &~ZF	Greater (Signed)
setge	~ (SF^OF)	Greater or Equal (Signed)
setl	(SF^OF)	Less (Signed)
setle	(SF^OF) | ZF	Less or Equal (Signed)
seta	~CF&~ZF	Above (unsigned)
setb	CF	Below (unsigned)

b >= a

Lab3 -- Uncover the mystery

To view assembly code: objdump -d -M suffix ./tester_sol > tester_sol.s
Search for function label <ex1>

Exercise

Hint:
??? func(??? x, ??? y)
{
???
}

- Guess what's the C code for function func
- git clone https://github.com/lazycal/test.git

```
0000000000400579 <func>:
  400579: 53                      pushq  %rbx
  40057a: 48 89 f3                movq   %rsi,%rbx
  40057d: 85 ff                   testl  %edi,%edi
  40057f: 74 11                   je     400592 <func+0x19>
  400581: 48 8d 76 ff             leaq   -0x1(%rsi),%rsi
  400585: 83 ef 01                subl   $0x1,%edi
  400588: e8 ec ff ff ff          callq  400579 <func>
  40058d: 48 01 d8                addq   %rbx,%rax
  400590: eb 03                   jmp    400595 <func+0x1c>
  400592: 48 89 f0                movq   %rsi,%rax
  400595: 5b                      popq   %rbx
  400596: c3                      retq
```

Exercise

- After running `cmpl %eax %ebx`, what are the status of CF and OF?

eax	ebx	CF	OF
0x7fffffff	0x00000000	1	0
0x7fffffff	0xffffffff	0	0
0x80000000	0x00000000	1	1
0x80000000	0x80000000	0	0

Exercise

- Guess what's the C code for function m
- wget https://raw.githubusercontent.com/DingDTest/Recitation-examples/main/r08/example_func2