

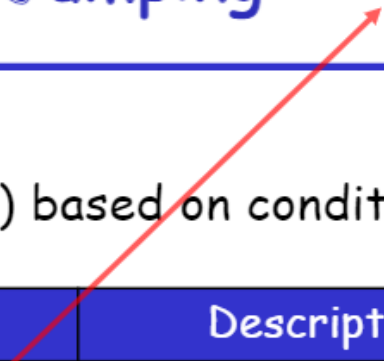
More x86-64, Assembly Lab

Condition Codes

- `cmpq op2, op1` # computes result = $op1 - op2$, discards result, sets condition codes
- `testq op2, op1` # computes result = $op1 \& op2$, discards result, sets condition codes
- Condition Codes - **ZF** (zero flag), **SF** (sign flag), **OF** (overflow flag, signed), and **CF** (carry flag, unsigned)

Using Condition Codes: **Jumping**

Don't worry too much about these!



- `j*` Instructions
 - Jumps to *target* (an address) based on condition codes

Instruction	Condition	Description
<code>jmp target</code>	1	Unconditional
<code>je target</code>	ZF	Equal / Zero
<code>jne target</code>	$\sim ZF$	Not Equal / Not Zero
<code>js target</code>	SF	Negative
<code>jns target</code>	$\sim SF$	Nonnegative
<code>jg target</code>	$\sim (SF \wedge OF) \ \& \ \sim ZF$	Greater (Signed)
<code>jge target</code>	$\sim (SF \wedge OF)$	Greater or Equal (Signed)
<code>jl target</code>	$(SF \wedge OF)$	Less (Signed)
<code>jle target</code>	$(SF \wedge OF) \mid ZF$	Less or Equal (Signed)
<code>ja target</code>	$\sim CF \ \& \ \sim ZF$	Above (unsigned ">")
<code>jb target</code>	CF	Below (unsigned "<")

11

- **set* Instructions**
 - Set low-order byte of `dst` to `0x00` or `0x01` based on condition codes
 - Does not alter remaining 7 bytes

Instruction	Condition	Description
sete <i>dst</i>	ZF	Equal / Zero
setne <i>dst</i>	\sim ZF	Not Equal / Not Zero
sets <i>dst</i>	SF	Negative
setns <i>dst</i>	\sim SF	Nonnegative
setg <i>dst</i>	$\sim (SF \wedge OF) \ \& \ \sim ZF$	Greater (Signed)
setge <i>dst</i>	$\sim (SF \wedge OF)$	Greater or Equal (Signed)
setl <i>dst</i>	$(SF \wedge OF)$	Less (Signed)
setle <i>dst</i>	$(SF \wedge OF) \ \ ZF$	Less or Equal (Signed)
seta <i>dst</i>	$\sim CF \ \& \ \sim ZF$	Above (unsigned ">")
setb <i>dst</i>	CF	Below (unsigned "<")

Exercise 1

```
int compound(int x, int y)
{
                        
    return 1;
else
    return 2;
}
```

compound:

```
    cmp1 $-48, %edi
```

```
    jl .L3
```

```
    movl $1, %eax
```

```
    ret
```

.L3:

```
    movl    $2, %eax
```

```
    ret
```

Exercise 1

```
int compound(int x, int y)
{
    if(x > -49)
        return 1;
    else
        return 2;
}
```

compound:

```
    cmp1 $-48, %edi
```

```
    jl .L3
```

```
    movl $1, %eax
```

```
    ret
```

.L3:

```
    movl    $2, %eax
```

```
    ret
```

Exercise 2

...

...

```
movl $0xABCD, %eax
```

```
movl $0x1BCD, %edi
```

```
cmpl %edi, %eax
```

```
setg %al
```

...

...

- What is the final value of %al?
 - 0000 0001
- What is the final value of %eax?
 - 0xAB**01**

Exercise 3

```
int compound(int x, int y, int *z)
{
    if( ) {
        if ( ) {
            x = ;
        }
        else if( ) {
            y = ;
        }
        else {
            y = ;
        }
    }
    return x + y;
}
```

compound:

```
        cmp1    %esi, %edi
        jle     .L2
        cmp1    $5, %esi
        jle     .L3
        addl    $3, %edi
.L2:
        leal    (%rdi,%rsi), %eax
        ret
.L3:
        cmp1    %esi, 4(%rdx)
        je      .L4
        addl    %edi, %esi
        jmp     .L2
.L4:
        addl    $5, %esi
        jmp     .L2
```


Exercise 3

```
int compound(int x, int y, int *z)
{
    if(x > y){
        if (y > 5){
            x = x + 3;
        }
        else if(y != *(z+1)){
            y = y + x;
        }
        else{
            y = y + 5;
        }
    }
    return x + y;
}
```

compound:

```
        cmpb    %edi, %esi
        jle     .L2
        cmpb    $5, %esi
        jle     .L3
        addb    $3, %edi
.L2:
        leal    (%rdi,%rsi), %eax
        ret
.L3:
        cmpb    %esi, 4(%rdx)
        je      .L4
        addb    %edi, %esi
        jmp     .L2
.L4:
        addb    $5, %esi
        jmp     .L2
```

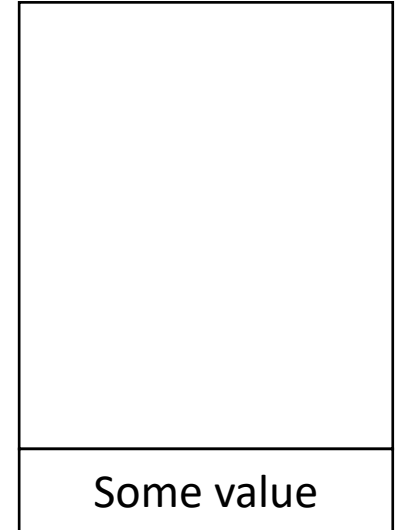
Push and Pop to Memory stack

```
pushq %rbp
```

```
...
```

```
...
```

%rsp → 0x0..108



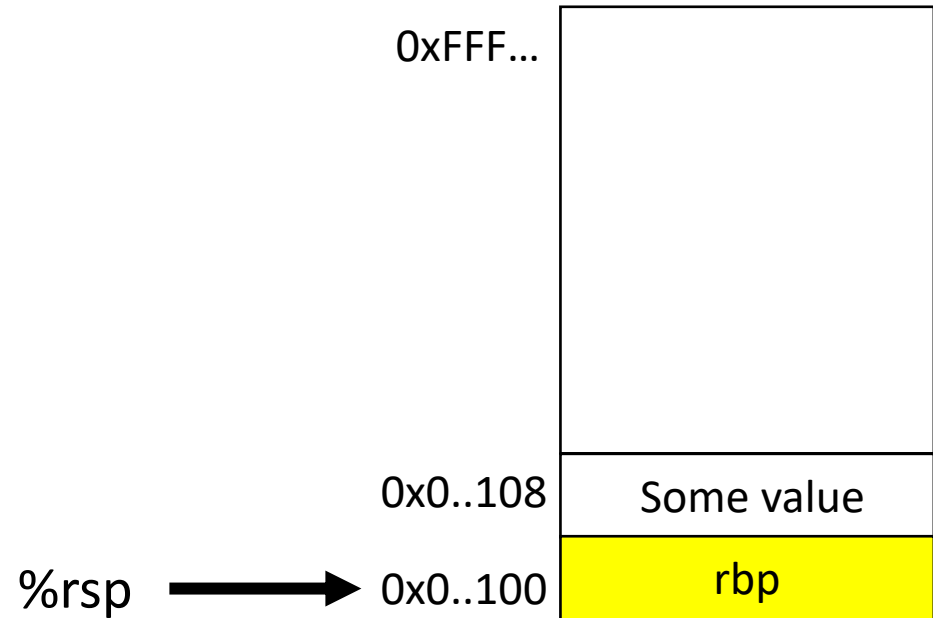
Push and Pop to Memory stack

```
pushq %rbp
```

...

...

```
pop %rbp
```



Push and Pop to Memory stack

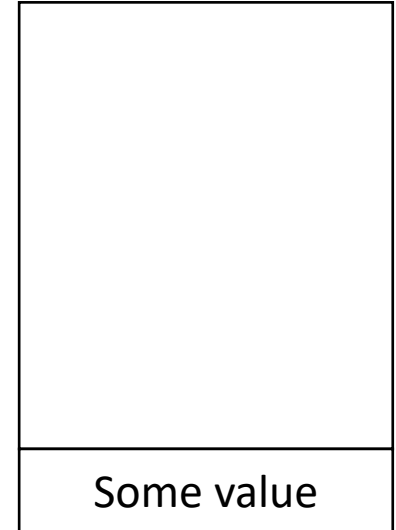
pushq %rbp

...

...

pop %rbp

%rsp → 0x0..108



echo:

24 bytes (decimal)

```
000000000040069c <echo>:
40069c: 48 83 ec 18      sub    $0x18,%rsp
4006a0: 48 89 e7         mov    %rsp,%rdi
4006a3: e8 a5 ff ff ff   callq 40064d <gets>
4006a8: 48 89 e7         mov    %rsp,%rdi
4006ab: e8 50 fe ff ff   callq 400500 <puts@plt>
4006b0: 48 83 c4 18      add    $0x18,%rsp
4006b4: c3              retq
```

After echo() finishes execution, the return address 4006c3 is popped from the stack

call_echo:

return address

```
4006b5: 48 83 ec 08      sub    $0x8,%rsp
4006b9: b8 00 00 00 00   mov    $0x0,%eax
4006be: e8 d9 ff ff ff   callq 40069c <echo>
4006c3: 48 83 c4 08      add    $0x8,%rsp
4006c7: c3              retq
```

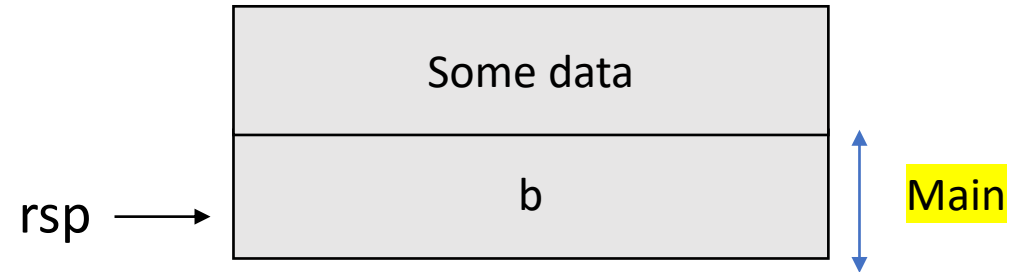
call to function echo() pushes the return address 4006c3 to the stack

Functions and Memory Stack

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

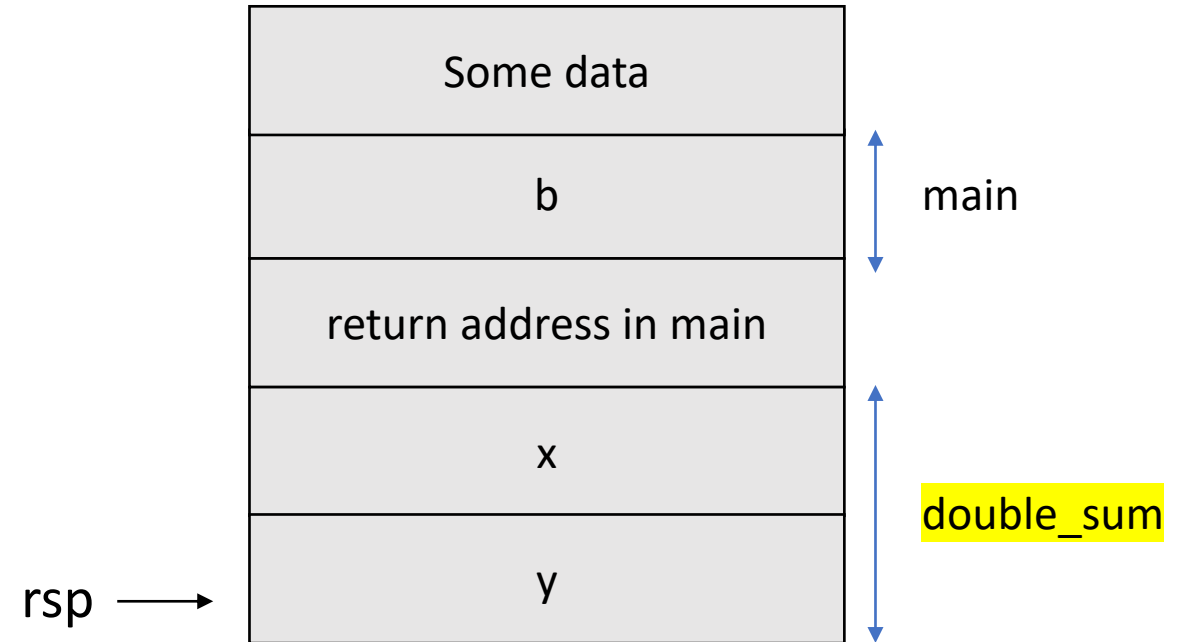

int double_sum(int x, int y){
return 2*add(x,y);
}

int main() {
int b = double_sum(10, 20);
return 0;
}
```



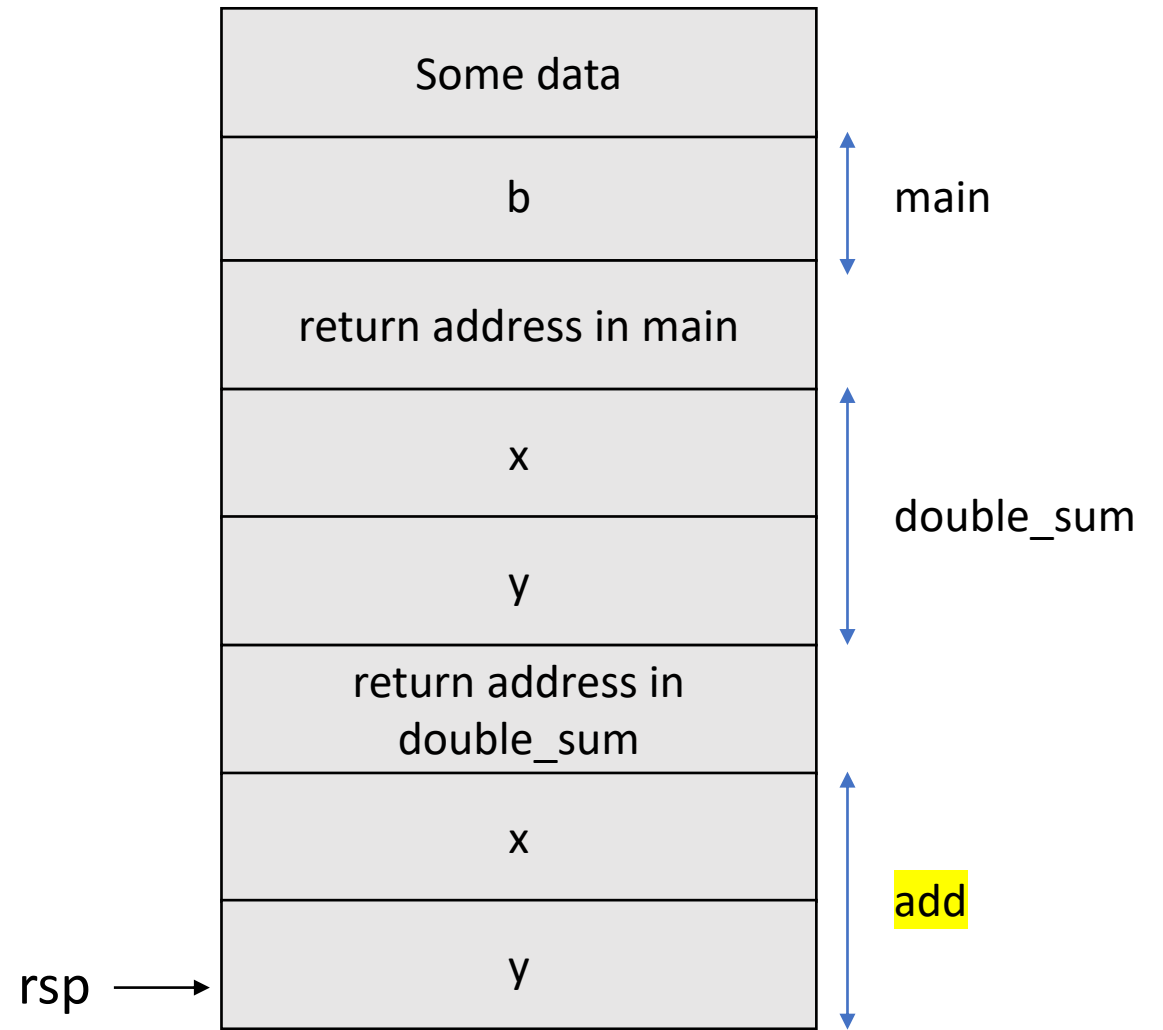

Functions and Memory Stack

```
int add(int x, int y)
{ return x+y;
}
int double_sum(int x, int y){
return 2*add(x,y);
}
int main() {
int b = double_sum(10, 20);
return 0;
}
```



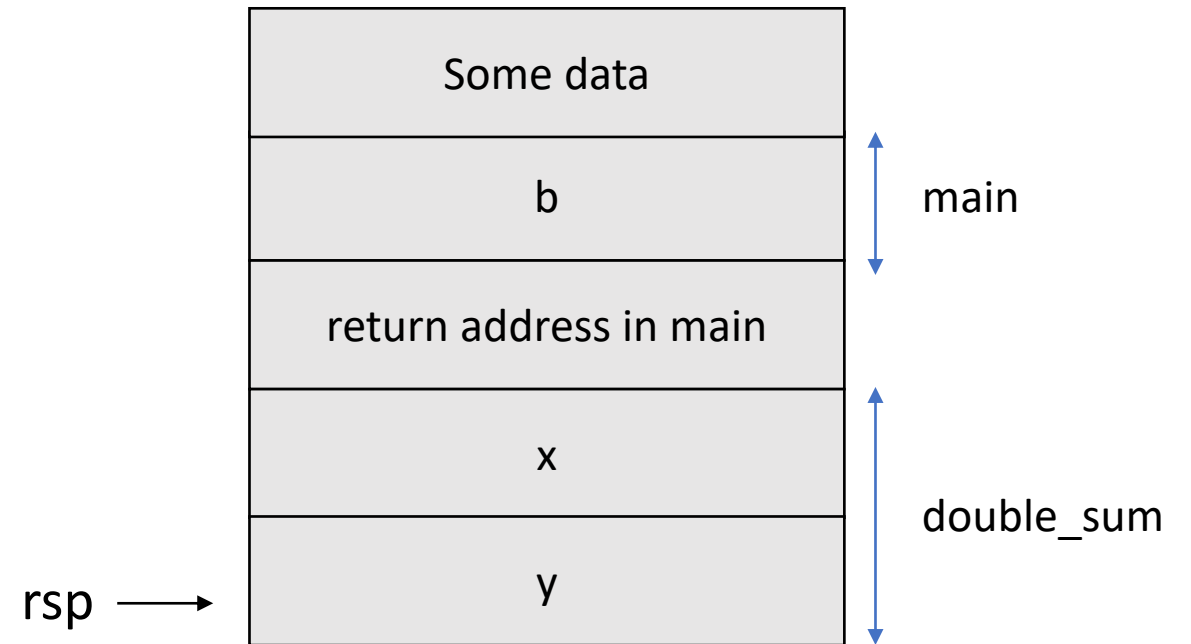

Functions and Memory Stack

```
int add(int x, int y)
{ return x+y;
}
int double_sum(int x, int y){
return 2*add(x,y);
}
int main() {
int b = double_sum(10, 20);
return 0;
}
```



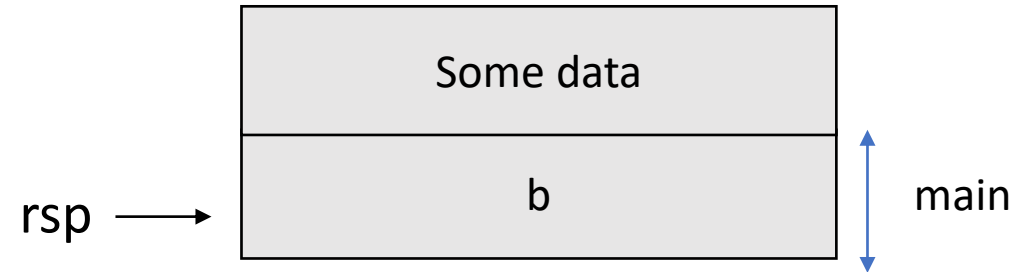
Functions and Memory Stack

```
int add(int x, int y)
{ return x+y;
}
int double_sum(int x, int y){
return 2*add(x,y);
}
int main() {
int b = double_sum(10, 20);
return 0;
}
```



Functions and Memory Stack

```
int add(int x, int y)
{ return x+y;
}
int double_sum(int x, int y){
return 2*add(x,y);
}
int main() {
int b = double_sum(10, 20);
return 0;
}
```



Functions and Memory Stack



```
int add(int x, int y)
{ return x+y;
}
int double_sum(int x, int y){
return 2*add(x,y);
}
int main() {
int b = double_sum(10, 20);
return 0;
}
```

Reverse Engineering a x86 code to C code - I

.LFB0

```
    pushq    %rbp
    movq     %rsp,%rbp
    movl     %edi,-4(%rbp)
    movl     %esi,-8(%rbp)
    movl     -4(%rbp),%eax
    cmpl     -8(%rbp),%eax
    jg       .L2
    movl     -8(%rbp),%eax
    jmp      .L3
.L2:
    movl     -4(%rbp),%eax
.L3:
    popq     %rbp
    ret
```

Step 1



```
int LFB0 (int edi, int esi)
{
    rbp = rsp
    [rbp - 4] = edi
    [rbp - 8] = esi
    eax = [rbp - 4]
    if (eax > [rbp - 8]) goto L2
    eax = [rbp - 8]
    goto L3
L2:
    eax = [rbp - 4]
L3:
    return eax
}
```

Reverse Engineering a x86 code to C code - I

```
int LFB0 (int edi, int esi)
{
    rbp = rsp
    [rbp - 4] = edi
    [rbp - 8] = esi
    eax = [rbp - 4]
    if (eax > [rbp - 8]) goto L2
    eax = [rbp - 8]
    goto L3
L2:
    eax = [rbp - 4]
L3:
    return eax
}
```

Step 2



```
int LFB0 (int edi, int esi)
{
    int eax;

    eax = edi;
    if (eax > esi) goto L2;
    eax = esi;
    goto L3;
L2:
    eax = edi;
L3:
    return eax;
}
```

Reverse Engineering a x86 code to C code - I

```
int LFB0 (int edi, int esi)
{
    int eax;

    eax = edi;
    if (eax > esi) goto L2;
    eax = esi;
    goto L3;
L2:
    eax = edi;
L3:
    return eax;
}
```

Step 3



```
int LFB0 (int edi, int esi)
{
    if (edi <= esi) {
        return esi;
    }
    else {
        return edi;
    }
}
```

Reverse Engineering a x86 code to C code - I

```
int LFB0 (int edi, int esi)
{
    if (edi <= esi) {
        return esi;
    }
    else {
        return edi;
    }
}
```

Step 4



```
int LFB0 (int x, int y)
{
    if (x <= y) {
        return y;
    }
    else {
        return x;
    }
}
```

```
int LFB0 (int x, int y)
{
    return (x > y) ? x : y;
}
```

Step 5



Reverse Engineer the following x86-64 code to C code

```
0000000000001129 <add>:
1129:    f3 0f 1e fa    endbr64
112d:    55              push    %rbp
112e:    48 89 e5        mov     %rsp,%rbp
1131:    89 7d ec        mov     %edi,-0x14(%rbp)
1134:    89 75 e8        mov     %esi,-0x18(%rbp)
1137:    c7 45 fc 00 00 00 00    movl    $0x0,-0x4(%rbp)
113e:    eb 0e          jmp     114e <add+0x25>
1140:    83 7d ec 64     cmpl    $0x64,-0x14(%rbp)
1144:    7e 04          jle     114a <add+0x21>
1146:    83 6d ec 0a     subl    $0xa,-0x14(%rbp)
114a:    83 45 fc 01     addl    $0x1,-0x4(%rbp)
114e:    83 7d fc 09     cmpl    $0x9,-0x4(%rbp)
1152:    7e ec          jle     1140 <add+0x17>
1154:    8b 45 ec        mov     -0x14(%rbp),%eax
1157:    5d              pop     %rbp
1158:    c3              retq
```

Pseudocode 1

```
add (int edi, int esi)
{
    rbp = rsp
    [rbp-0x14]= edi
    [rbp-0x18]= esi
    [rbp -0x4] = 0
    goto L1
L3: if [rbp-0x14]<=$0x64
        goto L2
    [rbp-0x14]-= 0xa
L2: [rbp -0x4] +=1
L1: if [rbp -0x4] <=0x9
        goto L3
    eax = [rbp-0x14]
}
```


Pseudocode 1

```
add (int edi, int esi)
{
    rbp = rsp
    [rbp-0x14]= edi
    [rbp-0x18]= esi
    [rbp -0x4] = 0
    goto L1
L3:   if [rbp-0x14]<=$0x64
        goto L2
    [rbp-0x14]-= 0xa
L2:   [rbp -0x4] +=1
L1:   if [rbp -0x4] <=0x9
        goto L3
    eax = [rbp-0x14]
}
```



Pseudocode 2

```
add(int edi, int esi)
{
    i = 0
    while i <= 0x9
    {
        if edi > 0x64
            edi -= 0xa

        i ++
    }
    return edi
}
```

Pseudocode 2

```
add(int edi, int esi)
{
    i = 0
    while i <= 0x9{
        if edi > 0x64
            edi -= 0xa

        i ++
    }
    return edi
}
```



C code

```
add (int x, int y)
{
    i = 0;
    while (i <= 9){
        if (x > 100)
            x -= 10;

        i ++;
    }
    return x;
}
```

Assembly Lab

- Refer to the Panopto video