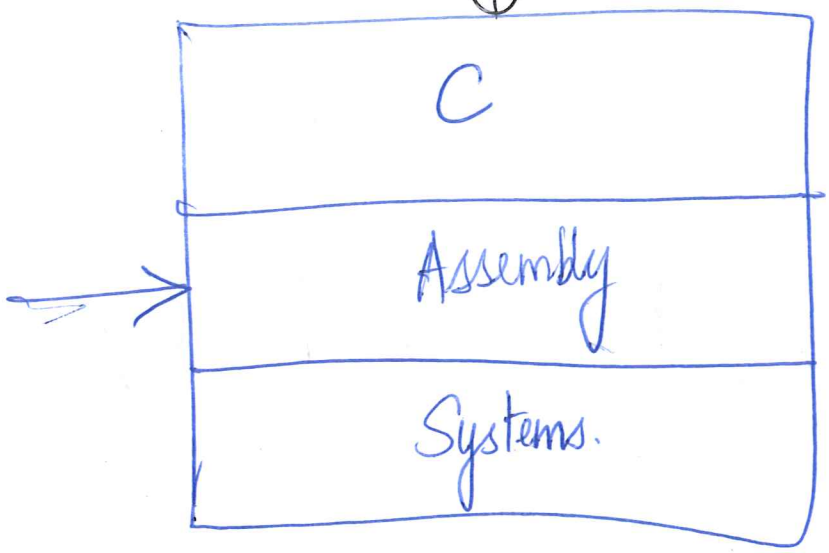
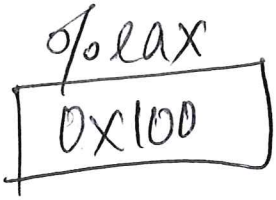
CS 354 - Lecture 8

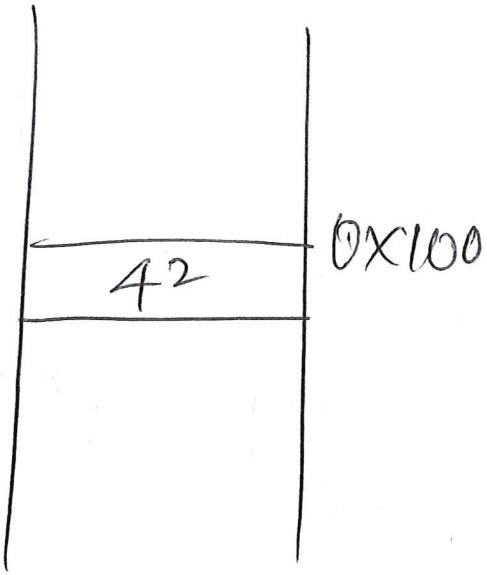
①



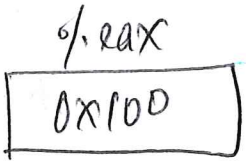
Registers

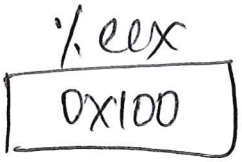
Memory



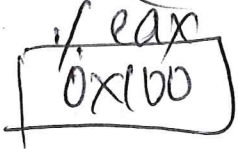


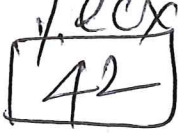
movl %eax, %ecx





movl (%eax), %ecx





Arithmetic, bitwise, etc.

(2) Control Flow

if / else

loops

cmpl a, b

b - a

testl a, b

b & a

%.eflags / CC.

CF, ZF, SF, OF

jmp label

jl, jle, jg, jge → signed

jb, jbe, ja, jae → unsigned.

%esp — stack pointer (Top of the stack)

leal → leal 4(%.eax), %.ecx

Today

Functions in Assembly

```
int sum ( int x, int y) {
    int total;
    total = x + y;
    return total;
}

int main ( ) {
    int s = sum (1, 2);
    return 0;
}
```

main() - caller

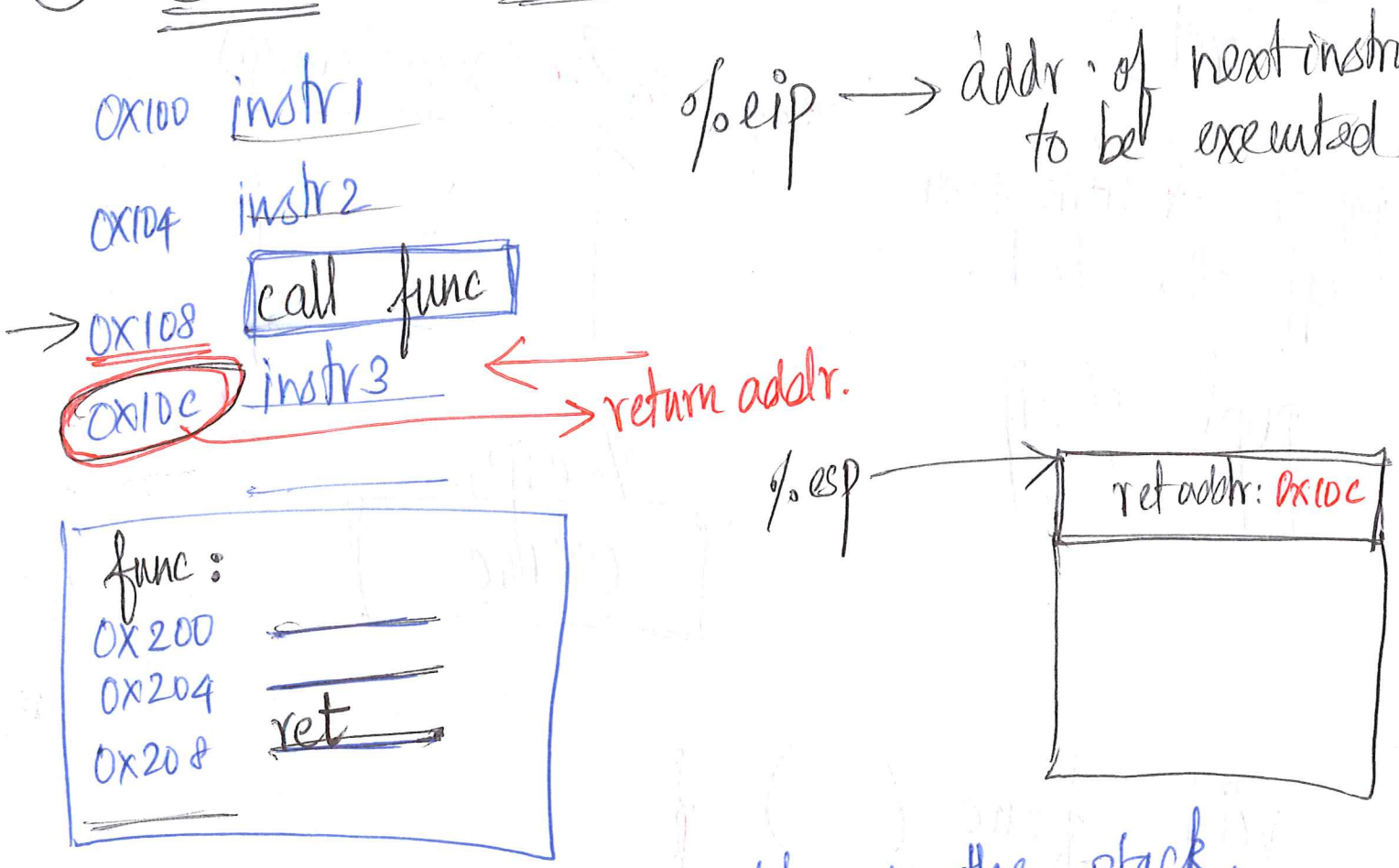
↓

sum() - callee

Functions – Calling Conventions (4) x86 (cdecl)

1. How to call a function?
2. How to return a control back to the caller func?
3. How to return value to the caller?
4. How to pass parameters?
5. How to allocate space for locals?
6. How to access parameters & locals?
7. How to deal with registers in a func?

① Call – new instruction



1. push the return addr on the stack.

```
pushl %eip
```

2. transfer control to the target.

```
jmp <target>
```

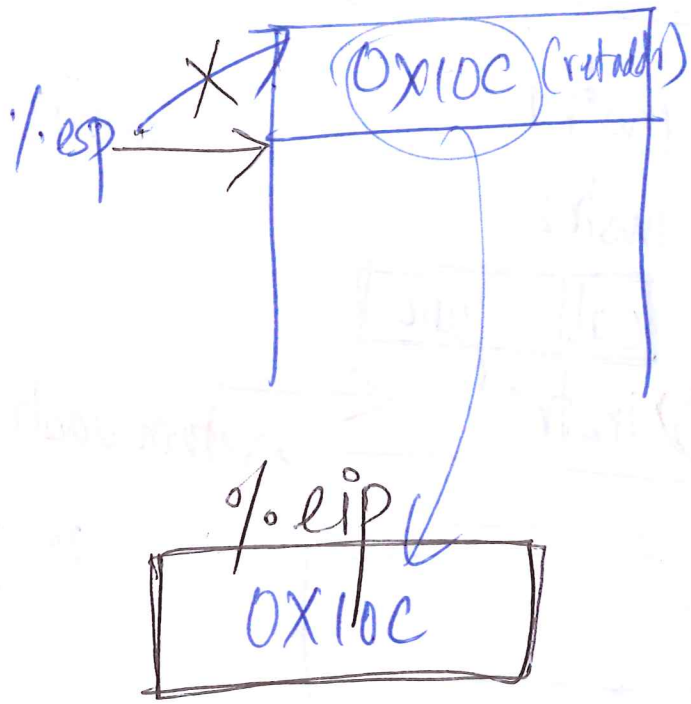
eg. `jmp func`

2) Return control

ret

1) pop the return addr off the stack.

```
popl %eip
```




```
void func ( ) {
    return;
}
```

```
int func() {
    return 42;
}
```

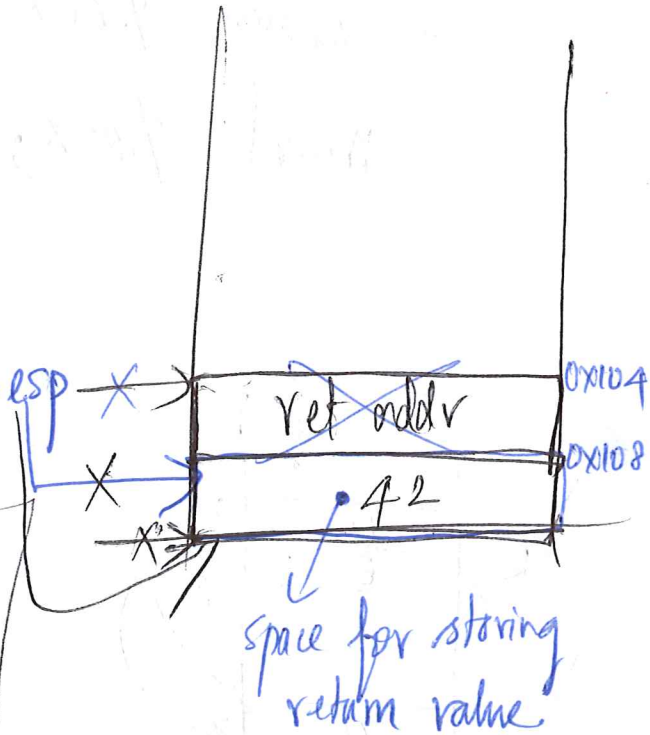
```
int main() {
    print func();
}
```

3. Return value?

Convention #1: store the return value onto the stack

```
func:
# put return value into stack location.
    movl $42, 4(%esp)
    ret

main:
# grow stack for return value.
    subl $4, %esp
    call func.
# move return value into eax.
    movl (%esp), %eax
# reduce stack
    addl $4, %esp
```



Convention #2: Use %eax to store the return value.

func:

```
movl $42, %eax
```

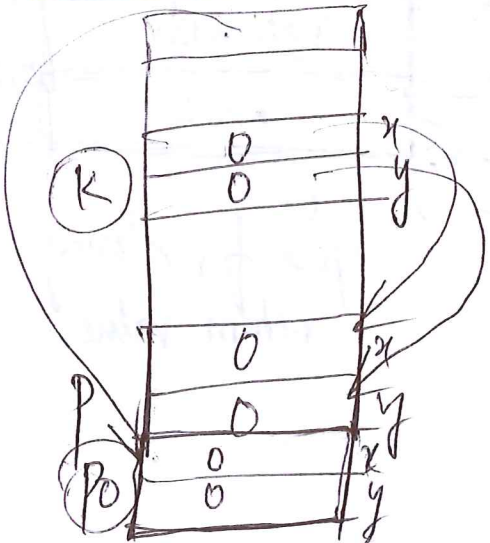
main:

```
call func.
```

→ assume that %eax has the return value.

```
# assume x is at 0x300.
# assume %ecx = 0x300.
movl %eax, (%ecx)
```

```
main() {
    int (x) = func();
}
```



Po: x: 0, y: 0

struct point (P) = initialize (Po);

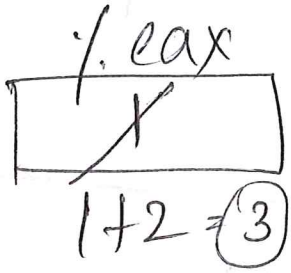
initialize (struct point (K)) {

4. How to pass parameters?

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

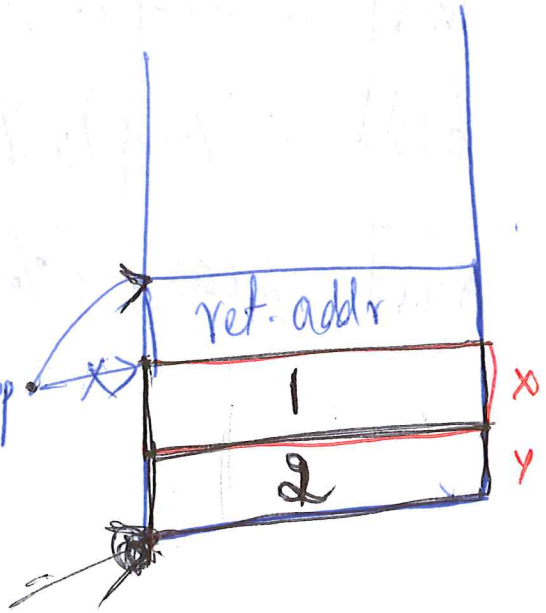
```
main() {
    func(1, 2);
}
```

```
func:
# add two args
    movl 4(%esp), %eax
    addl 8(%esp), %eax
# store return value in %eax
    ret
```




```
main:
# grow stack; put 2 args on top.
    subl $8, %esp
    movl $2, 4(%esp)
    mov $1, (%esp)
    call func
# reduce stack by 8 bytes
    addl $8, %esp
```

```
    pushl $2
    pushl $1
    call func
```



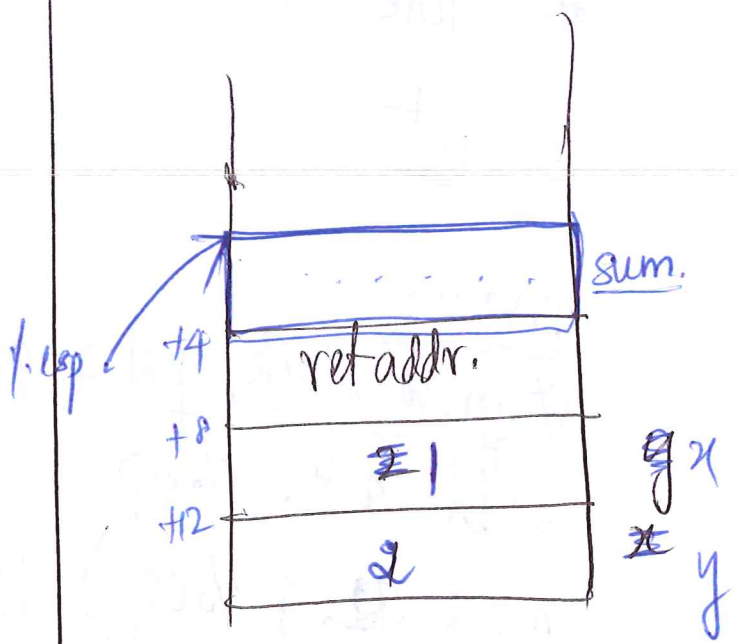
5. Local Variables

```
int func(int x, int y)
{
    int sum;
    sum = x + y;
    return sum;
}
```

```
main() {
    func(1, 2);
}
```

func:

```
#create space for local variable.
    subl $4, %esp
    movl 8(%esp), %eax
    addl 12(%esp), %eax
    movl %eax, (%esp)
```



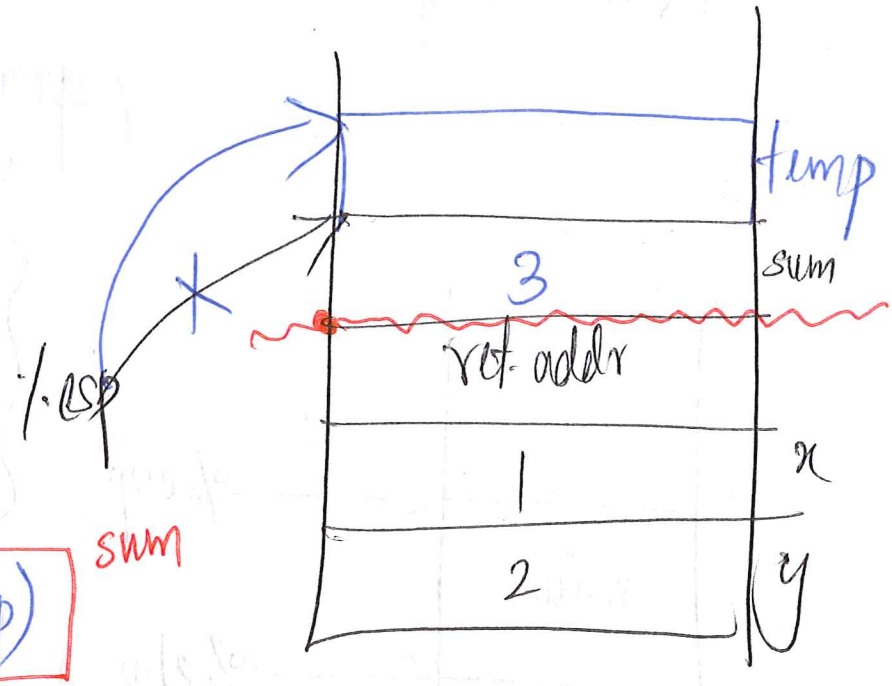
(11)

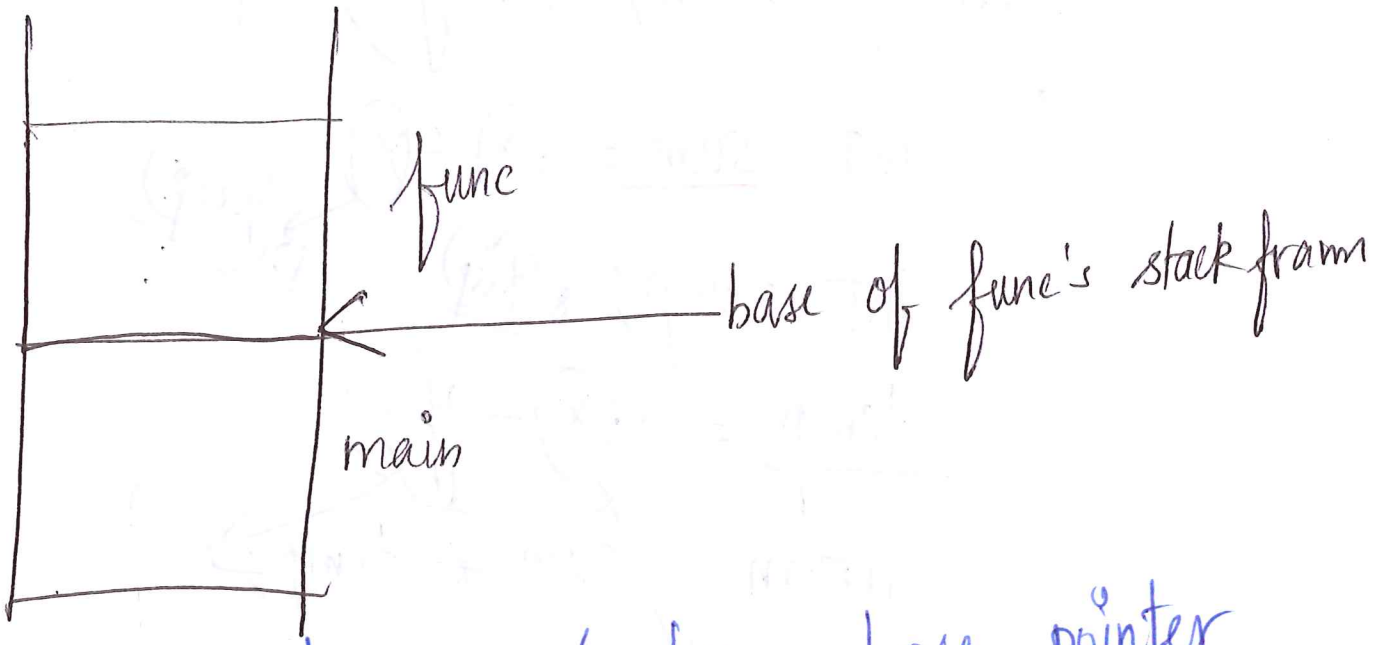
```
int func(int x, int y) {
    int sum = x + y;
    int temp;
    temp = x - y;
    return sum * temp;
}
```

x → 8(%esp), y → 12(%esp)

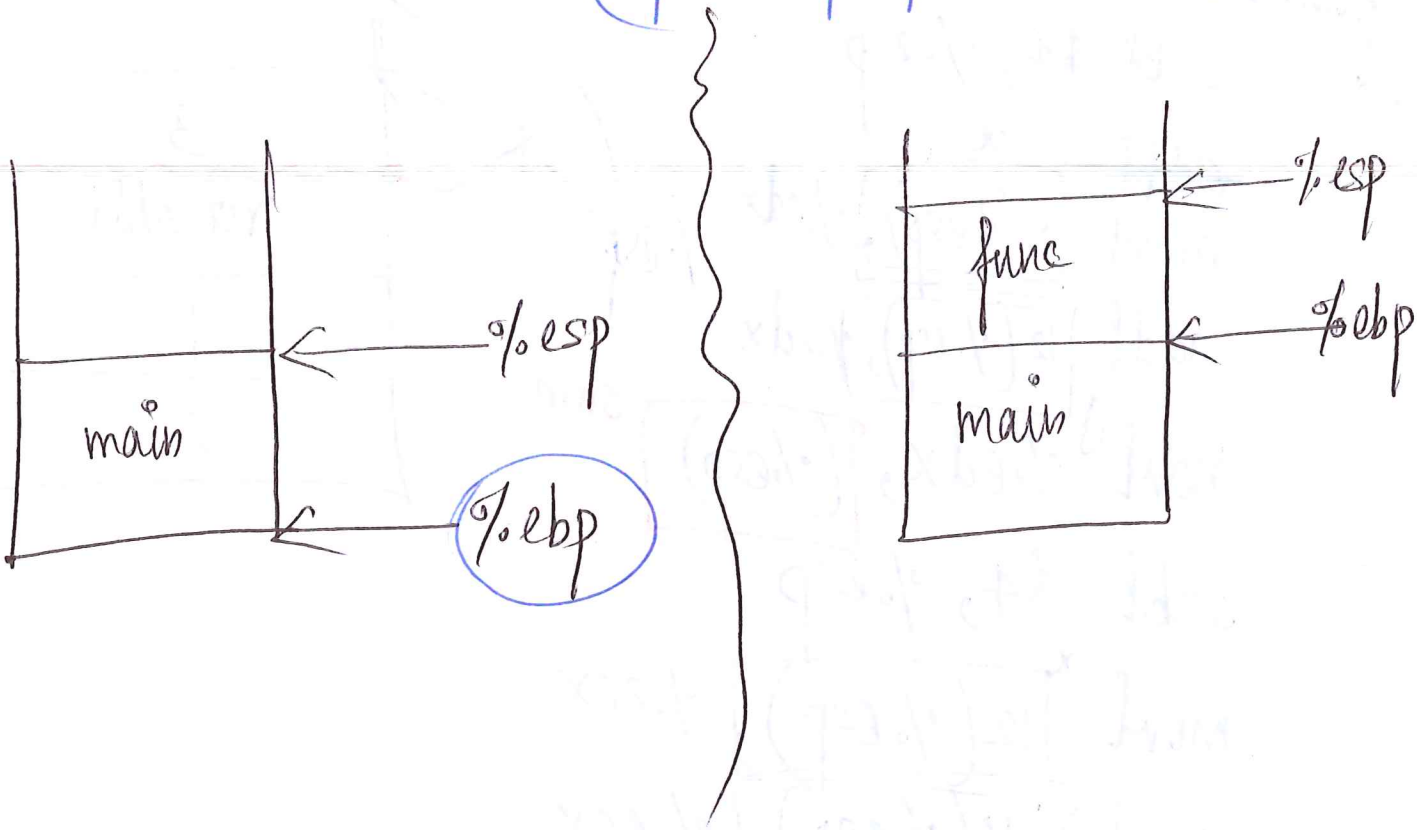
func:

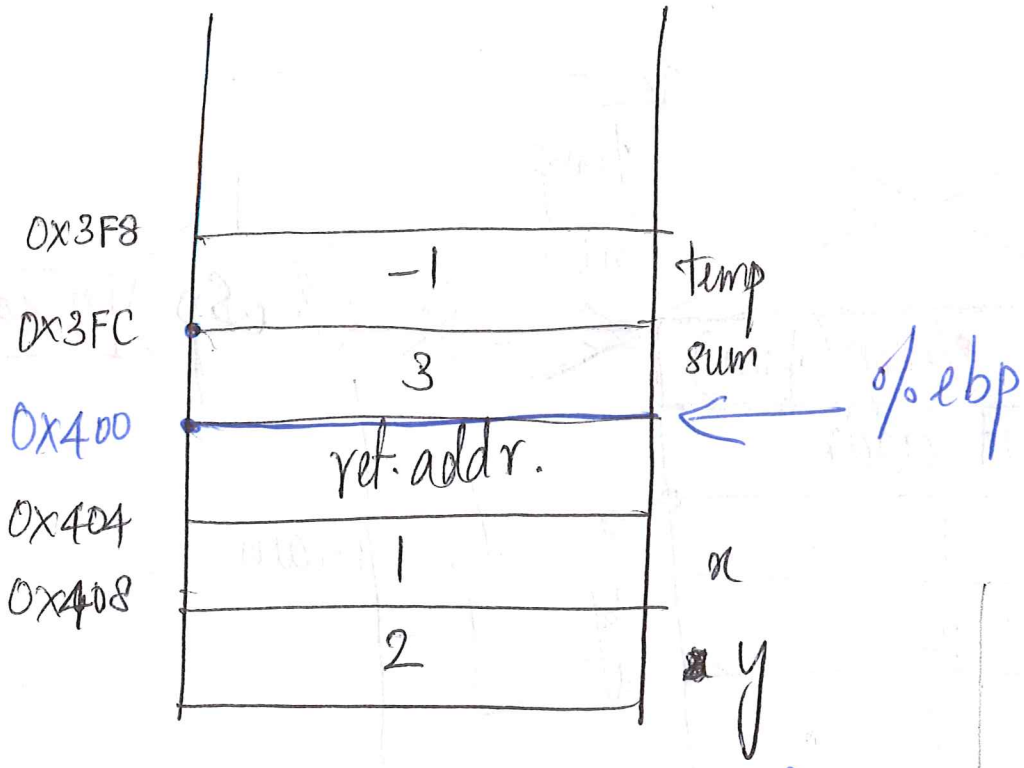
```
subl $4, %esp
movl 8(%esp), %edx
addl 12(%esp), %edx
movl %edx, (%esp)        # sum
subl $4, %esp
movl 12(%esp), %ecx
subl 16(%esp), %ecx
movl %ecx, (%esp)        # temp
```





New register - %ebp - base pointer
(special purpose register)





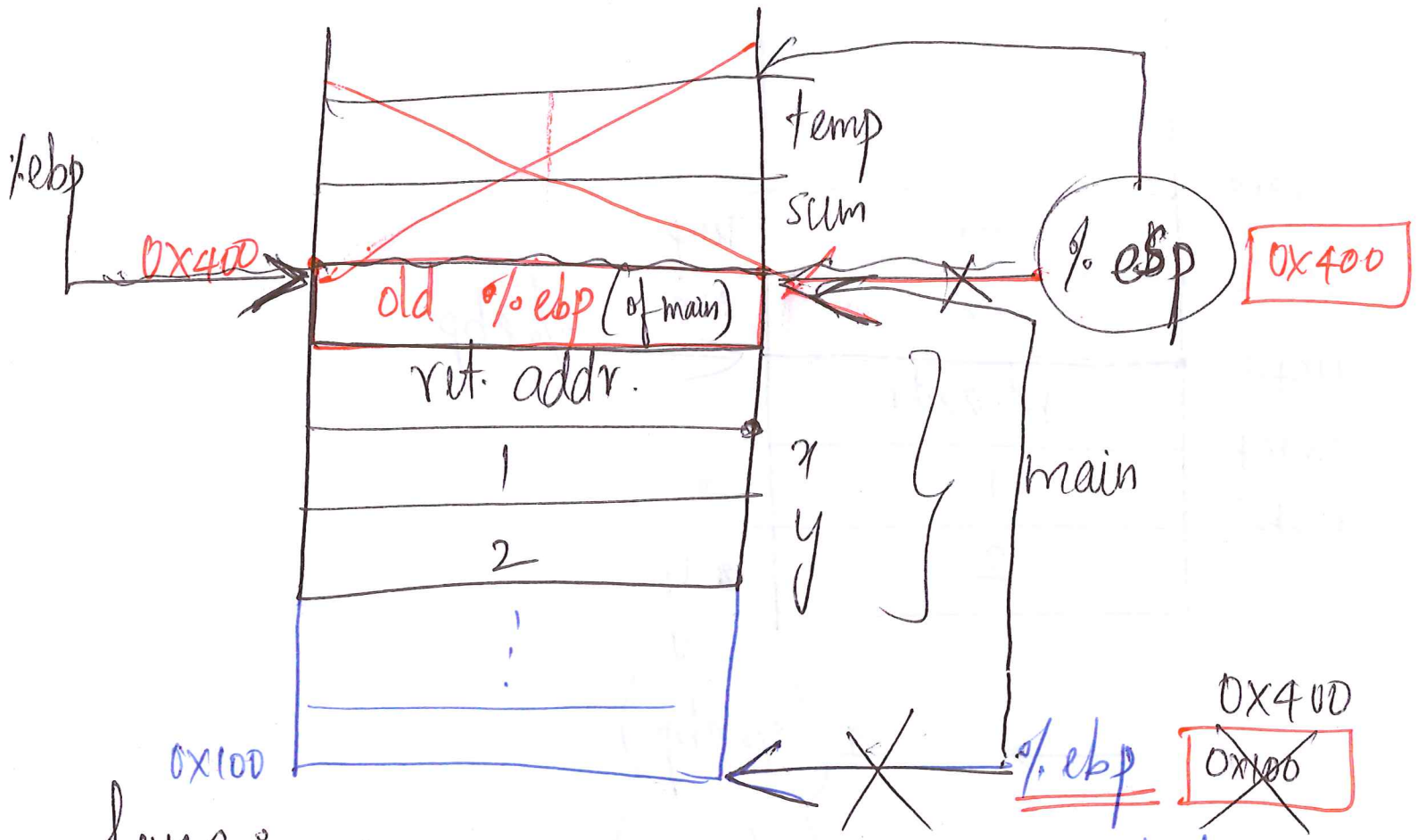
$x \longrightarrow 4(\%ebp)$

$y \longrightarrow 8(\%ebp)$

$(\%ebp) \rightarrow$? return addr.

$sum \longrightarrow -4(\%ebp)$

$temp \longrightarrow -8(\%ebp)$



func:

store caller's %ebp onto the stack.

setup
- pushl %ebp
- movl %esp, %ebp

1st arg (x) → 8(%ebp)

2nd arg (y) → 12(%ebp)

create local variables.

do stuff

tear down.
- movl %ebp, %esp # deallocate locals
- popl %ebp → leave

1st local → -4(%ebp)

2nd " → -8(%ebp)

7. Registers?

callee: func():
```
movl $42, %eax
ret
```

caller: main():
```
%eax | 10
pushl %eax
call func
#more return value to some location
popl %eax
```
value of %eax = ? 42

Convention

1. caller saved registers
   %eax, %ecx, %edx

2. callee saved registers
   %ebx, %esi, %edi

func:

pushl %ebx
use %ebx within func.
popl %ebx

main:

movl $10, %ebx

call func

→ # main can assume that %ebx is unchanged during the call.