CS 354 – Lecture 9

①

1. Functions
2. Pointers
3. Arrays
4. Structures

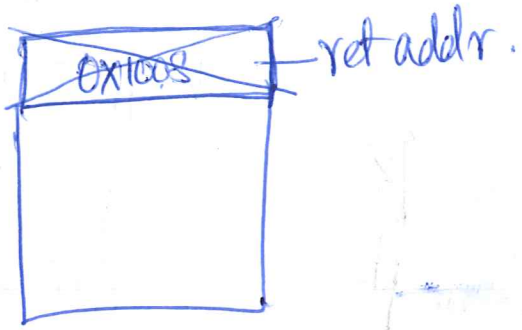
Review

(1, 2) main:

instr 1

→ call func

0x100-8 instr2 → (continue exec)



func:

→ instr 3

" 4

ret

③ return values → %eax int, ptrs, & data ≤ 4 bytes

→ stack → structures.

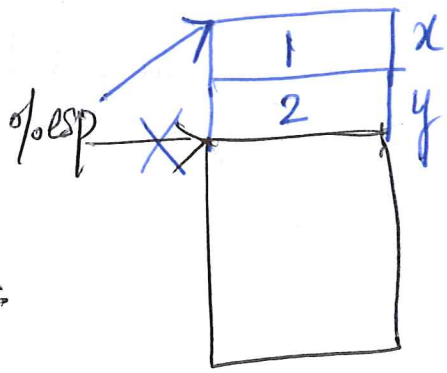
④ passing parameters

main() {

func(x, y);

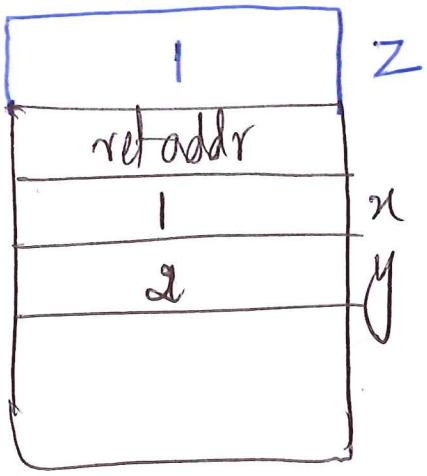
}

1 2

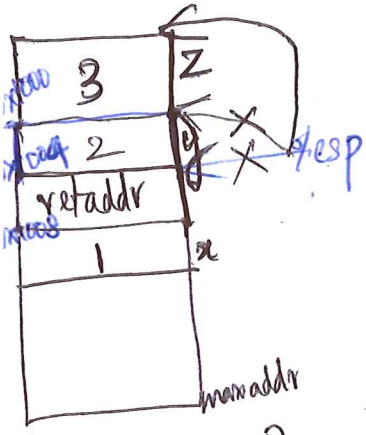


5 Local variables

```
int func(int x, int y){
    int z = 1;
    . . . . . .
}
```




```
int func (int x) {
    int y;
    y = x+1;
    int z;
    z = x+y;
    return z;
}
```



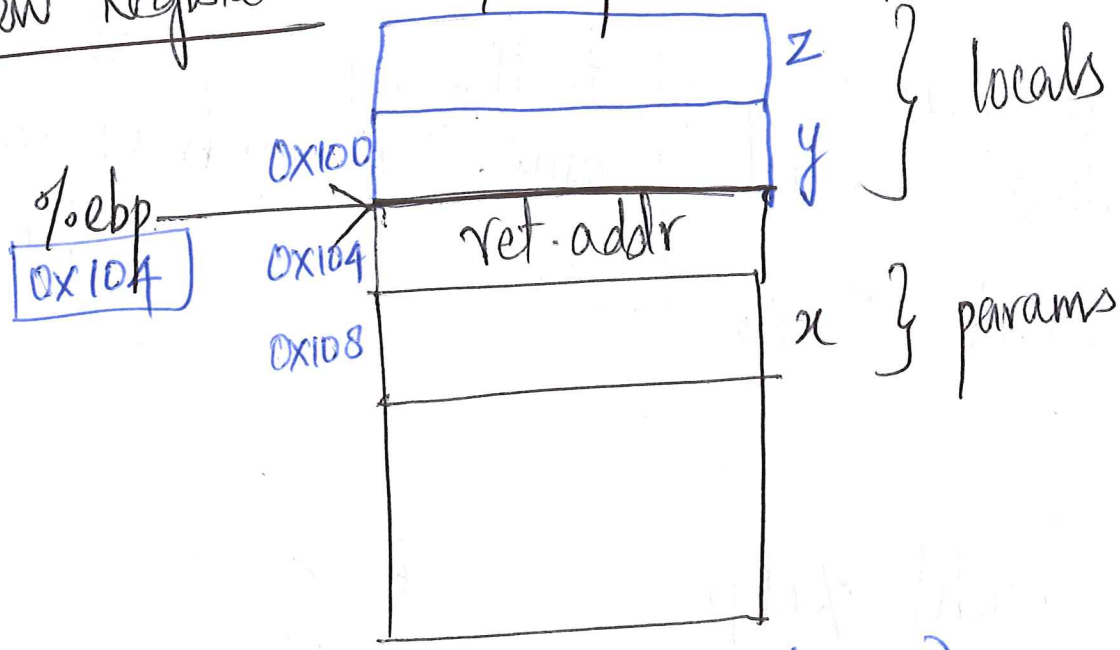

```
func:
    subl $4, %esp
    movl 8(%esp), %eax
    addl $1, %eax
    movl %eax, 0(%esp)
    subl $4, %esp
    movl 12(%esp), %eax
    addl 4(%esp), %eax
    movl %eax, 0(%esp)
    ret   addl $8, %esp
```

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

```
subl $4, %esp
movl $1, (%esp)
call func
```

Goal: To refer to params or/and locals in a consistent way even though %esp may be changing in a function.

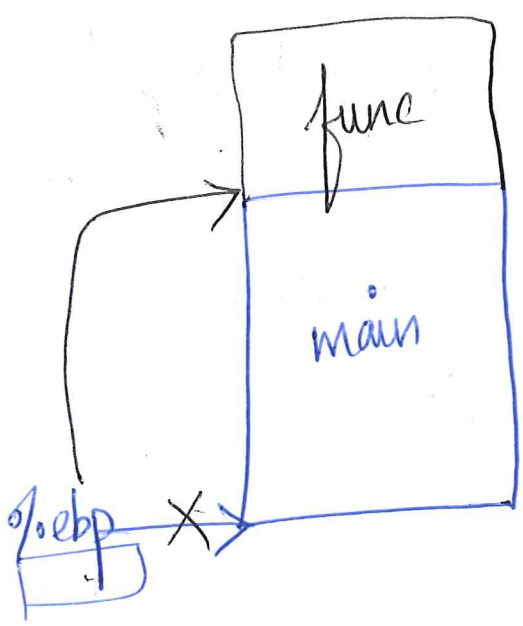
New Register – %ebp – Base pointer / Frame pointer

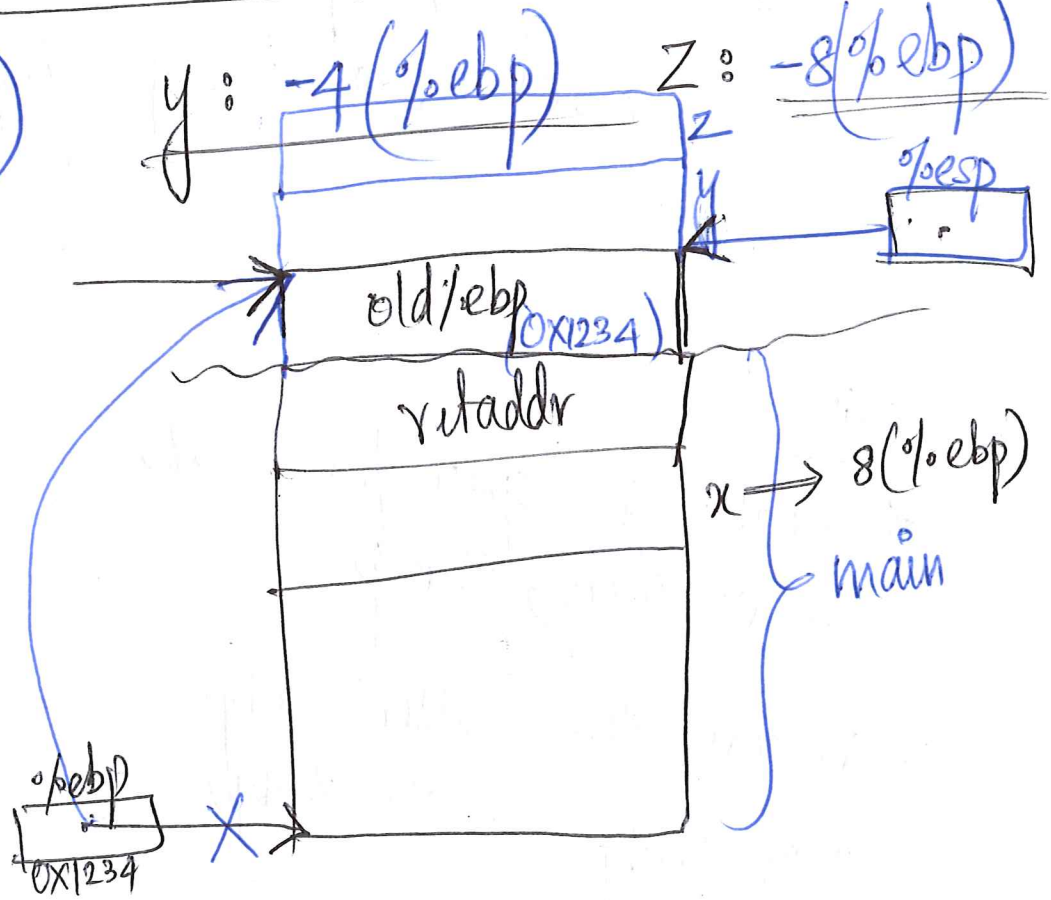


x : 4(%ebp)

y : -4(%ebp)

z : -8(%ebp)





Protocol

1. Inside a function.

   save old %ebp.

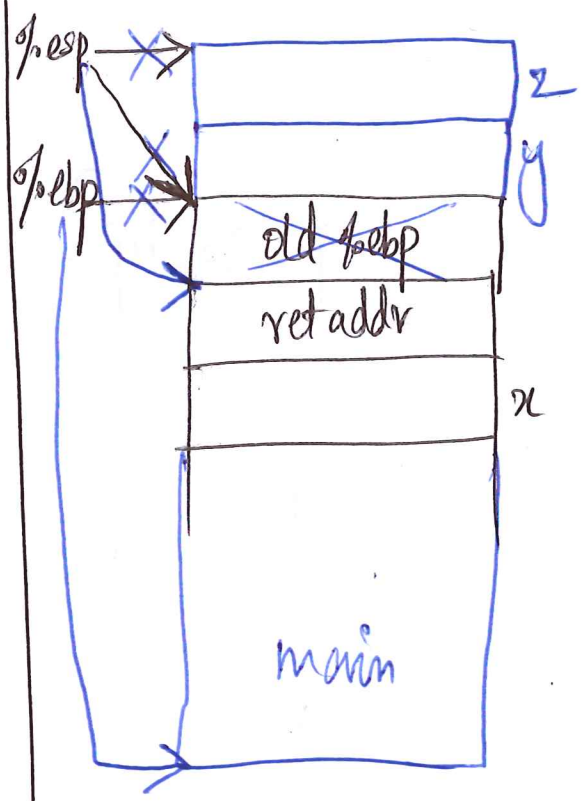
   ```
   pushl %ebp
   ```

2. Update %ebp to point to the base of the callee's stack frame. (OR) current top of stack.

   ```
   movl %esp, %ebp
   ```

entry:

```
pushl %ebp
movl %esp, %ebp
```



exit:

```
# deallocate space for locals
movl %ebp, %esp

# restore the caller's ebp.
  (old %ebp)
popl %ebp
```

5

```
int incr(int x) {
    return x+1;
}
```

```
incr:  pushl %ebp
       movl %esp, %ebp
       movl 8(%ebp), %eax
       addl $1, %eax
       popl %ebp
       ret
```

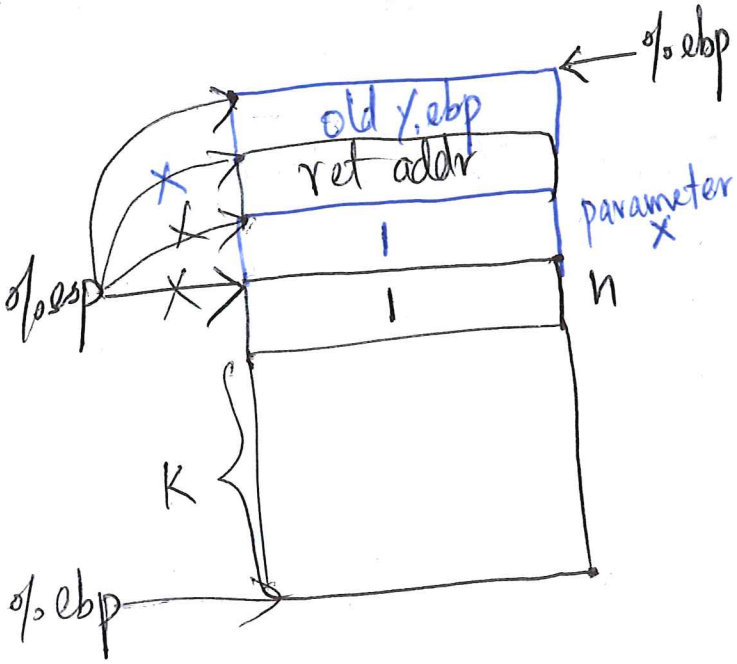
```
int main() {
    int n = 1;
    n = incr(n);
    return 0;
}
```

print n

print n

```
subl $4, %esp
movl $1, (%esp)
pushl (%esp)
call incr
addl $4, %esp
movl %eax, K(%ebp)
movl $0, %eax
```

```
movl %ebp, %esp
popl %ebp
```



7. How to deal with registers?

```
movl $10, %ecx
call func
```

func:

```
movl $20, %ecx
```

main ( ) caller

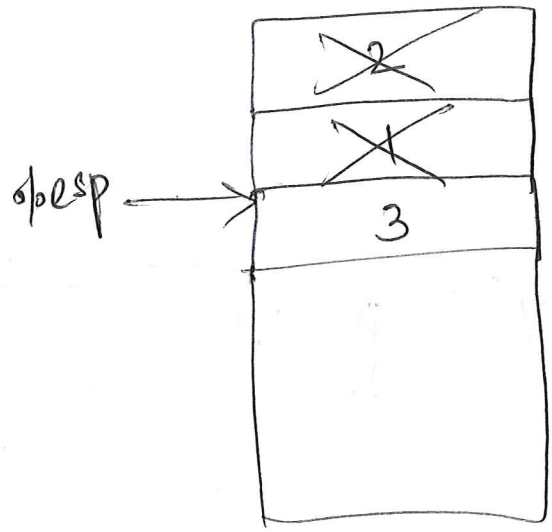
↓

func( ) callee

caller	callee
%eax, %ecx, %edx	%ebx, %esi, %edi

main:

```
# set %eax to some value
    movl $3, %eax
# save %eax (caller save)
    pushl %eax
# grow stack; put 2 args on top
    pushl $1      } 8 bytes
    pushl $2
# call func
    call func
# reduce stack
    addl $8, %esp
# save retval in %ebx
    movl %eax, %ebx
# restore %eax    popl %eax
```



---

func: entry

~~movl $42, %eax~~

~~exit~~

~~ret~~

8

%eip = 0x 8 0 4 8 4 5 d

\+ 0x F F F F F F B F

0x 0 8 0 4 8 4 1 C

```
void pincr(int *x) {
    *x = *x + 1;
}
```

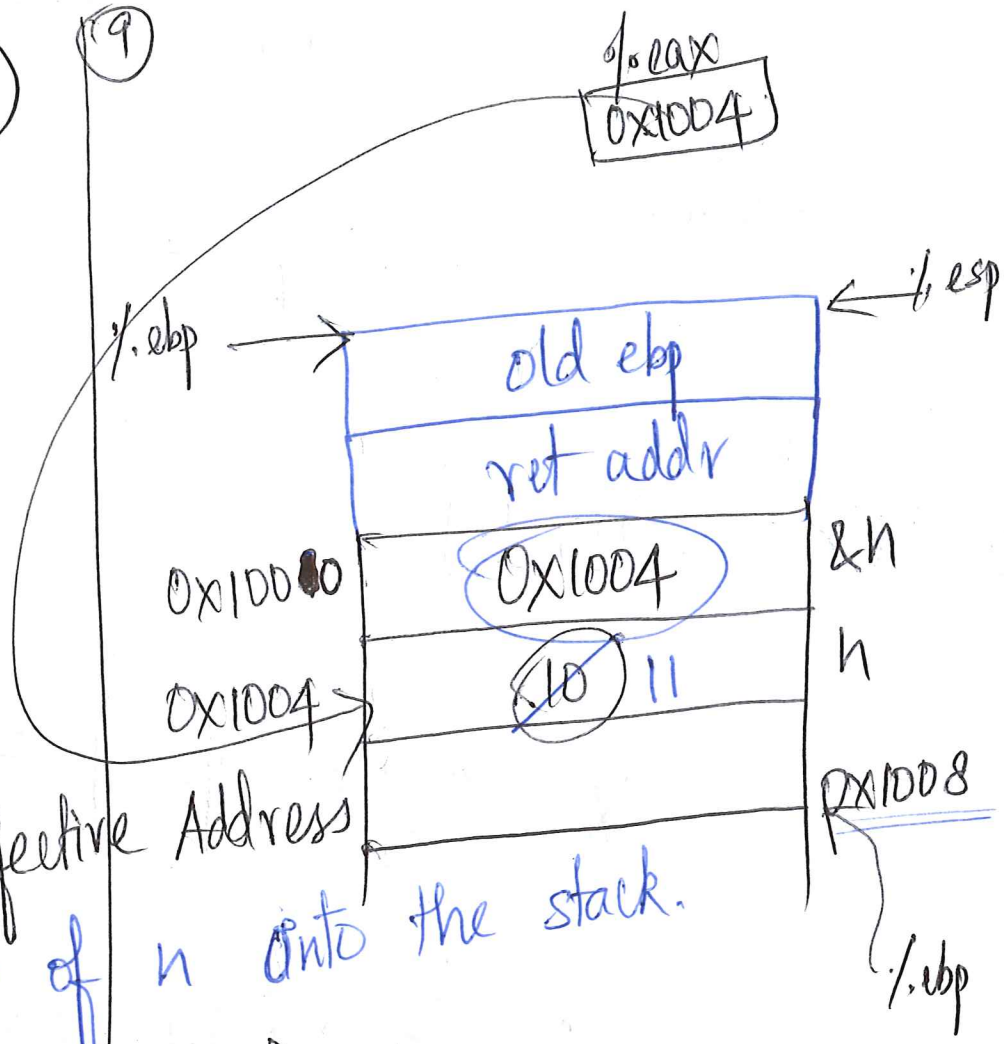
pincr:
```
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %eax
addl $1, (%eax)
movl %ebp, %esp
popl %ebp
ret
```

```
int main() {
    int n = 10;
    pincr(&n);
    return 0;
}
```

#Assume n is at -4(%ebp)
```
subl $4, %esp
movl $10, -4(%ebp)
movl %ebp, %ecx
subl $4, %ecx
~~subl $4, %esp~~
~~mov~~ pushl %ecx
call pincr
```

replace with leal -4(%ebp), %ecx

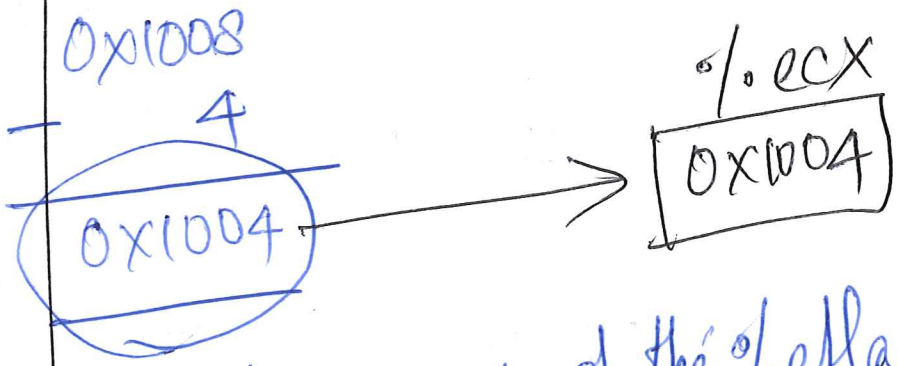
void incr(int *x)



LEAL – Load Effective Address

move the addr of n onto the stack.

leal -4(%ebp), %ecx



LEAL doesn't update any of the %eflags/CCs.