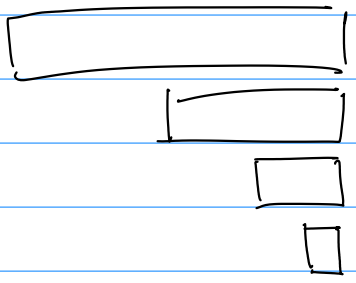
80% — x86_64 — asm → C
- assembler 1
- linker 2
- emulator 3
- macro processor 4

ISA , 4 + 2

64-bit	32-bit	16-bit	8-bit
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

x86 — 64



word = 16 bit

rip

pc = ip

CISC

Tip	Zapis	Vrednost
Neposredni	`Imm`	`Imm`
Registarski	`Rx`	`R[Rx]`
Memorijski	`[Rx]`	`M[R[Rx]]`
Memorijski	`[Rx + Imm]`	`M[R[Rx] + Imm]`
Memorijski	`[Rx + Ry]`	`M[R[Rx] + R[Ry]]`
Memorijski	`[Rx + Ry + Imm]`	`M[R[Rx] + R[Ry]+ Imm]`
Memorijski	`[Rx + s*Ry]`	`M[R[Rx] + s*R[Ry]]`
Memorijski	`[Rx + s*Ry + Imm]`	`M[R[Rx] + s*R[Ry] + Imm]`

intel, att

0x AB, 10, 0777, 0b1001

[rax], [r6x], ...

1, 2, 4, 8

[rax + 10]

[rax + 8 · r6x]

[rax + 4r6x + 17] — 15

Instrukcija	Efekat	Opis
`mov D, S`	`D := S`	move
`movsx D, S`	`D := signExtend(S)`	move sign extend
`movzx D, S`	`D := zeroExtend(S)`	move zero extend
`movabs R, Imm`	`R := Imm`	mov 64-bit immediate to a register

unsigned $1_0 \ldots 0 \; 0 \; 1 \; 1 \; 1 \; 0 \; 1$

$1 \ldots 1 \; 1$

$2^5 \quad 2^1 \; 2^0$

signed $1 \; 1 \; 1 \; 1 \; 1 \; 0 \; 1 \; 1$ — 5

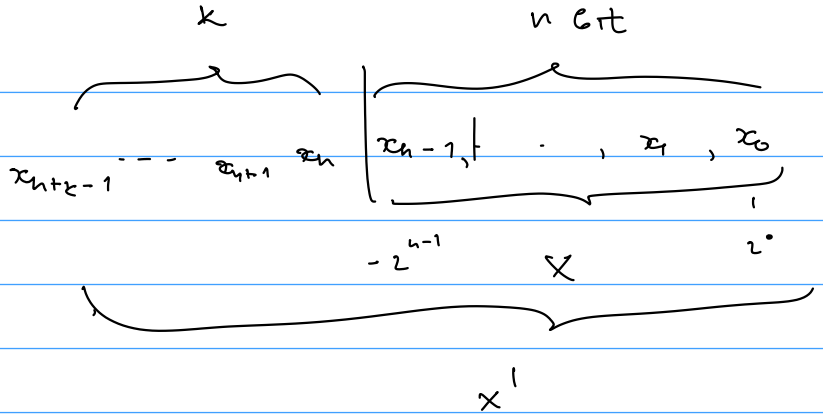
$-2^7 \; 2^5 \; 2^3 \; 2^2 \; 2^1 \; 2^0$

$-2^3 \; 2^2 \; 2^1 \; 2^0$

$-8 + 2 + 1 = -5$

$x_n, x_{n-1}, \ldots, x_1, x_0$

$-2^n \quad 2^{n-1} \quad 2^1 \; 2^0$



$$val(x) = -2^{n-1} \cdot x_{n-1} + \sum_{i=0}^{n-2} x_i 2^i$$

$$val(x') = -2^{n+k-1} x_{n+k-1} + \sum_{i=0}^{n+k-2} x_i 2^i$$

$x_{n-1} = 0$

$$val(x) = \sum_{i=0}^{n-2} x_i 2^i$$

$$val(x') = \sum_{i=0}^{n-2} x_i 2^i$$

$$val(x) = val(x')$$

2^k

$1\,0\,\ldots\,0$

$1\,\ldots\,1 + 1$

k

$2^k - 1$

$x_{n-1} = 1$

$$val(x') = -2^{n+k-1} + \sum_{i=n-1}^{n+k-2} 2^i + \sum_{i=0}^{n-2} x_i 2^i$$

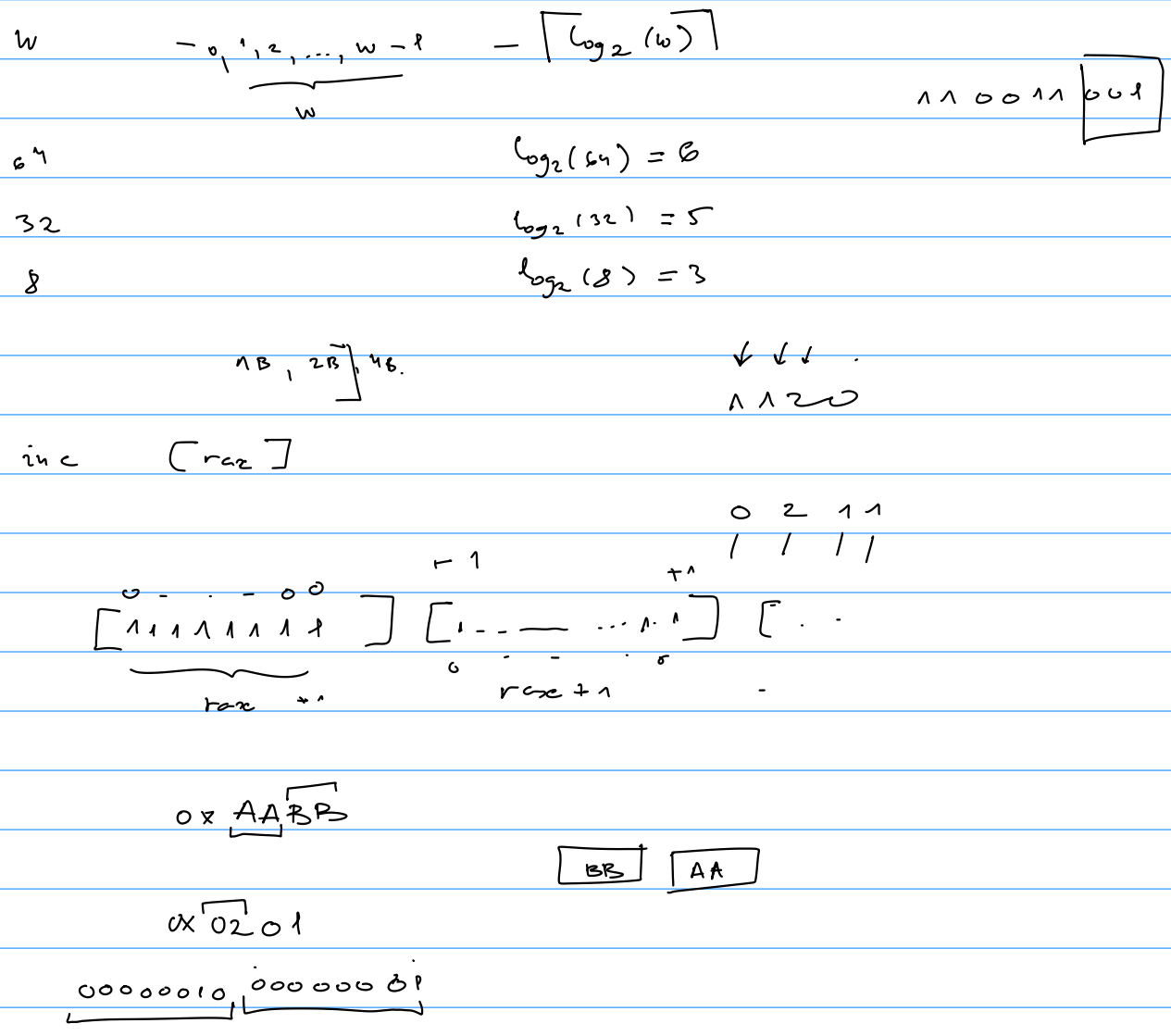
$$val(x') = -2^{n+k-1} + 2^{n-1} \boxed{\sum_{i=0}^{k-1} 2^i} + \sum_{i=0}^{n-2} x_i 2^i$$

$$val(x') = -2^{n+k-1} + 2^{n-1}(2^k - 1) + \sum_{i=0}^{n-2} x_i 2^i$$

$$= \underbrace{-2^{n-1}} + \sum_{i=0}^{n-2} x_i 2^i$$

$$= -x_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i = val(x)$$

Instrukcija	Efekat	Opis
`inc D`	`D := D+1`	Increment
`dec D`	`D := D-1`	Decrement
`neg D`	`D := -D`	Negate
`not D`	`D := ~D`	Complement
`add D, S`	`D := D+S`	Add
`sub D, S`	`D := D-S`	Subtract
`imul D, S`	`D := D*S`	Multiply
`xor D, S`	`D := D^S`	Exclusive-or
`or D, S`	`D := D|S`	Or
`and D, S`	`D := D&S`	And
`sal D, k`	$D := D \ll_A k$	Arithmetic left shift
`shr D, k`	$D := D \ll_L k$	Logic left shift
`sar D, k`	$D := D \gg_A k$	Arithmetic right shift
`shr D, k`	$D := D \gg_L k$	Logic right shift



DWORD PTR

```
add   eax , [r6x]
```

32 bit

Instrukcija	Efekat	Opis
`imul S`	`R[rdx]:R[rax] := S*R[rax]`	Signed full multiply
`mul S`	`R[rdx]:R[rax] := S*R[rax]`	Unsigned full multiply
`idiv S`	`R[rax] := R[rdx]:R[rax] div S` `R[rdx] := R[rdx]:R[rax] mod S`	Signed divide
`div S`	`R[rax] := R[rdx]:R[rax] div S` `R[rdx] := R[rdx]:R[rax] mod S`	Unigned divide
`cqo`	`R[rdx]:R[rax] := signeExtend(R[rax])`	Convert to oct word

[rdx][rax]

```
lea   rax , [r6x]              rax ← r6x
lea   rax , [r6x + 10]         rax ← r6x + 10
lea   rax , [r6x + 8*rcx + 10] rax ← r6x + 8*rcx + 10
```

```
mov   rax , rcx    1
imul  rax , 8      3
add   rax , 10     1
add   rax , r6x    1
```

signed overflow

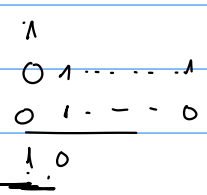
ZF, SF, OF, CF

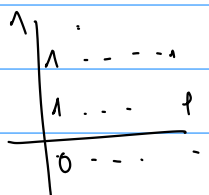
```
sub   eax , 10
```

unsigned overflow

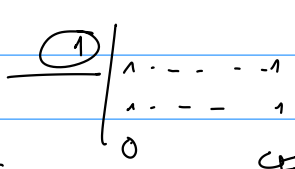
```
rax , r6x
add   rax , r6x
```

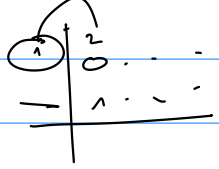
[+ , + → −]
[− , − → +]

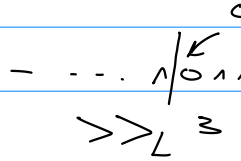


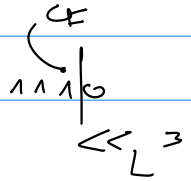


```
add   rax , r6x
```









setg D	setnle	D := ~(SF^OF)&~ZF	Greater (signed >)
setge D	setnl	D := ~(SF^OF)	Greater or equal (signed >=)
setl D	setnge	D := SF^OF	Less (signed <)
setle D	setng	D := (SF^OF)|ZF	Less or equal (signed <=)

setg D D ← 1 cmp a, b a - b > 0 a > b

(+) − (−) → (−)

(−) − (+) → (+)

SF	OF	SF ^ OF
1	1	0
1	0	1
0	1	1
0	0	0

a < b

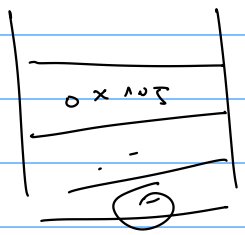
seta D	setnbe	D := ~CF&~ZF	Above (unsigned >)
setae D	setnb	D := ~CF	Above or equal (unsigned >=)
setb D	setnae	D := CF	Below (unsigned <)
setbe D	setna	D := CF|ZF	Below or equal (unsigned <=)

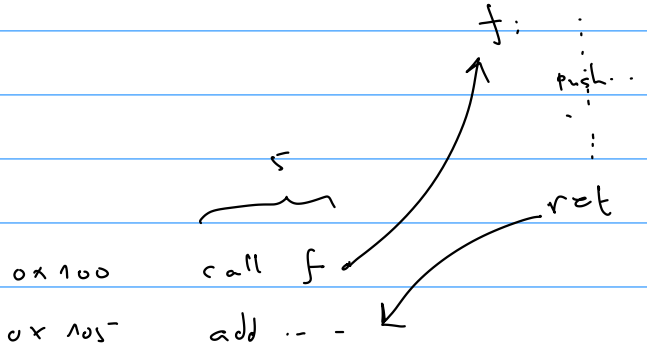
cmp a, b a - b

a < b

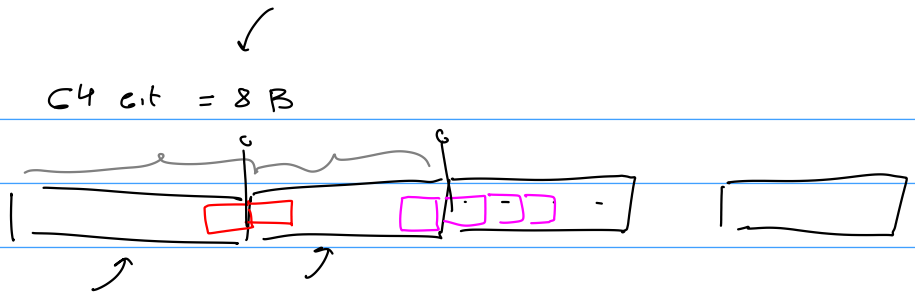
≥

≤






```
f() {
    :
    g();
    :
}
```



short x; → 2B

int x; → 4B

2B → align(2)

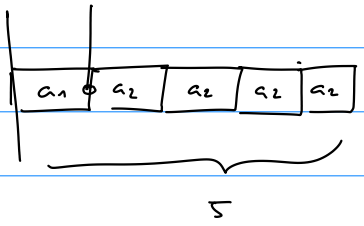
4B → align(4)

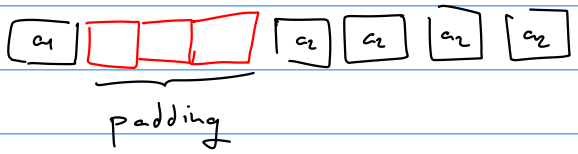
8B → align(8)

```
struct A {
    char a1;    align(1)
    int  a2;    align(4)
};
```

size(A) = ~~?~~ 8

align(A) = ~~?~~ 4






```
struct X {          sizeof(X)
    :               align(x).
};
```

struct X arr[N];

i ∈ {0, 1, ..., N-1}

arr mod align(x) = 0

(arr + i sizeof(x)) mod align(x) = 0

(i · sizeof(x)) mod align(x) = 0

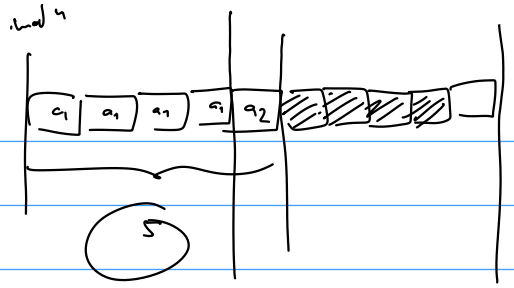
sizeof(x) mod align(x) = 0

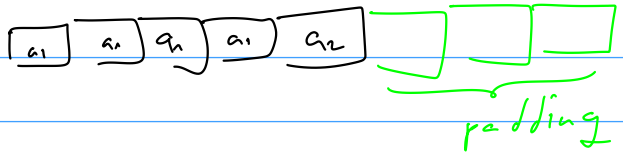
struct A {

int a1; align(4)

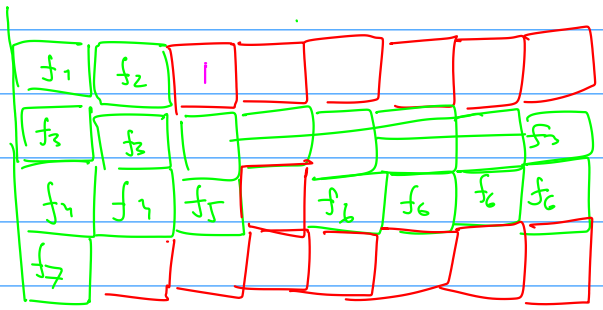
char a2; align(1)

}; align(A) = 4






```
 3 typedef struct
 4 {
 5     char field1;     // size:  1, alignment: 1
 6     char field2;     // size:  1, alignment: 1
 7     long field3;     // size:  8, alignment: 8
 8     short field4; // size:  2, alignment: 2
 9     char field5;     // size:  1, alignment: 1
10     int field6;      // size:  4, alignment: 4
11     char field7;     // size:  1, alignment: 1
12 } StructType;        // size: 32, alignment: 8
13
```



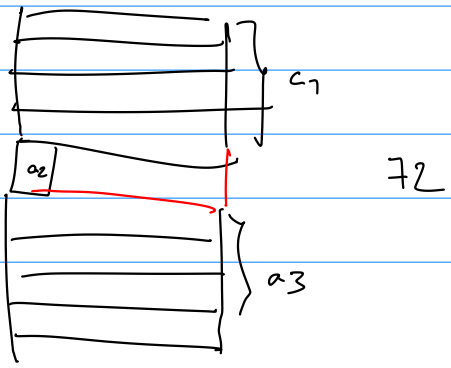
struct B {

struct A a1; a: 8

char a2; a: 1

struct A a3; a: 8

}; s: 72 a: 8



72

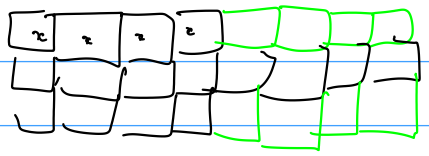
struct A_1 {

int x; 4

long y; 8

int z; 4

} 24 8



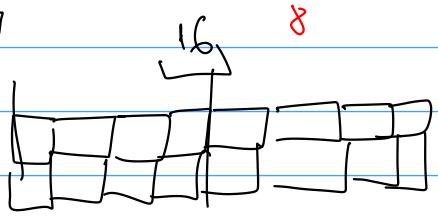
struct A_2 {

int x; 4

int z; 4

long y; 8

} 16 8

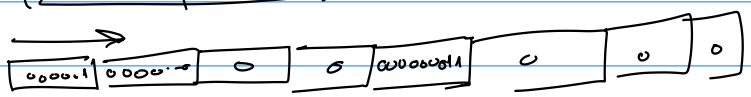


rax := 1

rax <<= 32;

rax += 3

.LC0



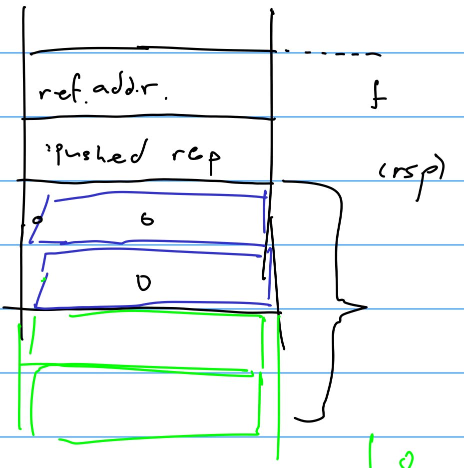
rax: 

long – 8B, 0

f:

```
push rbp
mov rbp, rsp
sub rsp, 16                      add rsp, 16
mov QWORD PTR (rbp-8), 0         rsp+8, rsp+24
mov QWORD PTR (rbp-16), 0        rsp, rsp+16
  :
sub rsp, 16                      add rsp, 16
  :
push — pop

mov rsp, rbp ]
pop rbp      ] leave
ret
```

base pointer



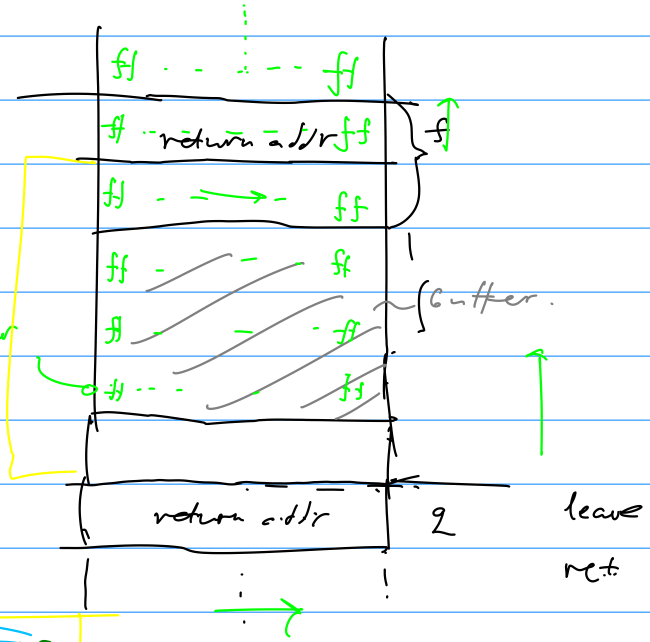
[enter n, 0]

```
push rbp
mov rbp, rsp
sub rsp, n
```

0xffeeddccbbaa

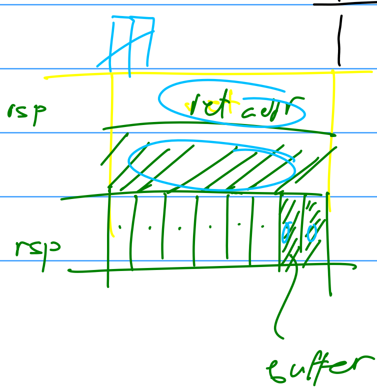
```
f() {
  :
  char buffer[24];
  :
  g(buffer);
}
```

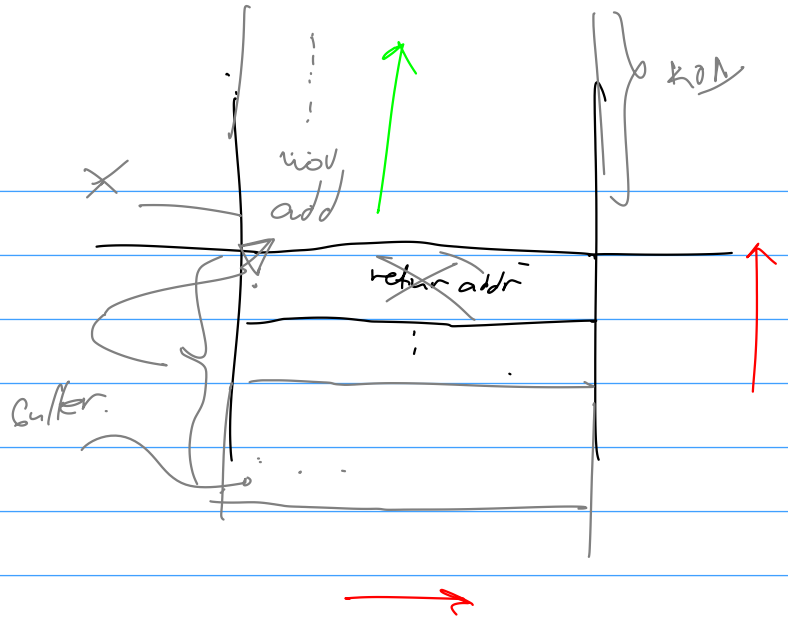
0xff

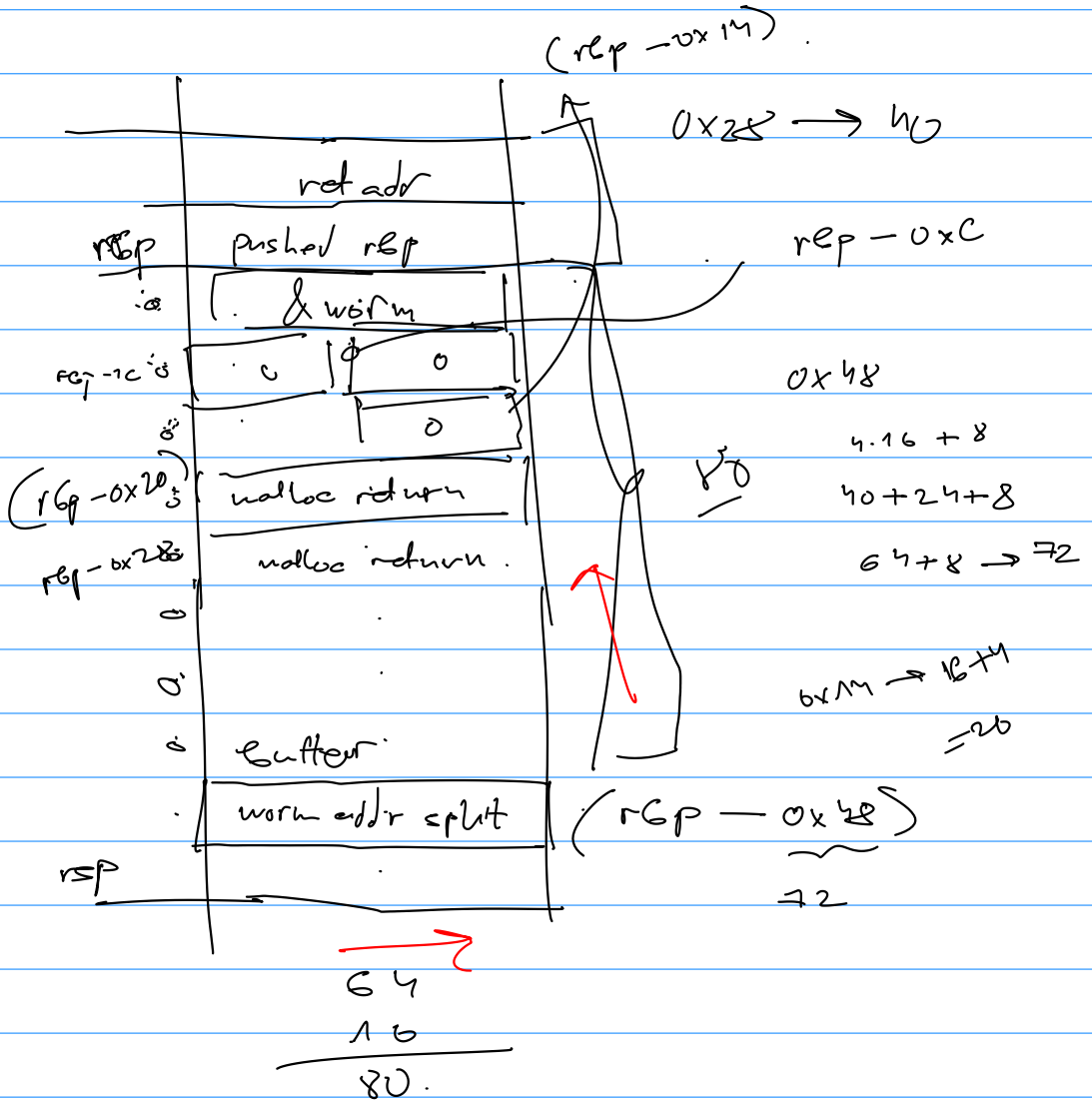


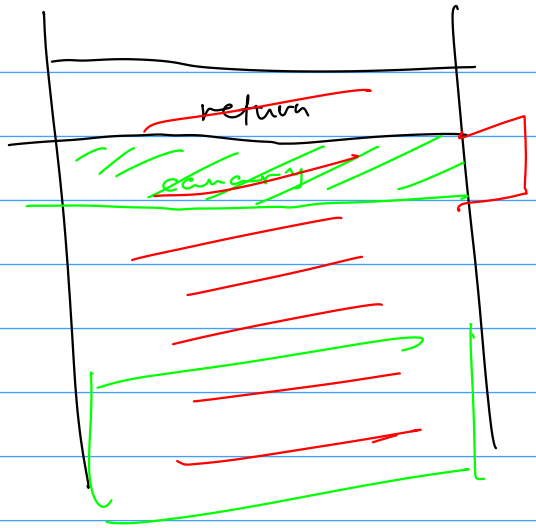
username[24]

... (username)

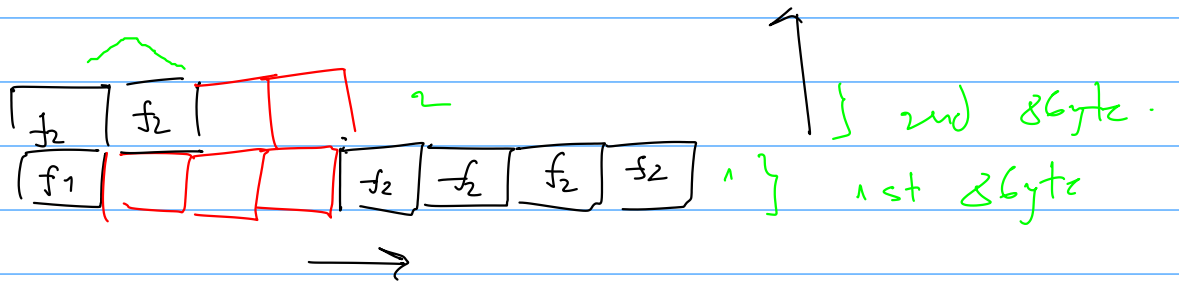


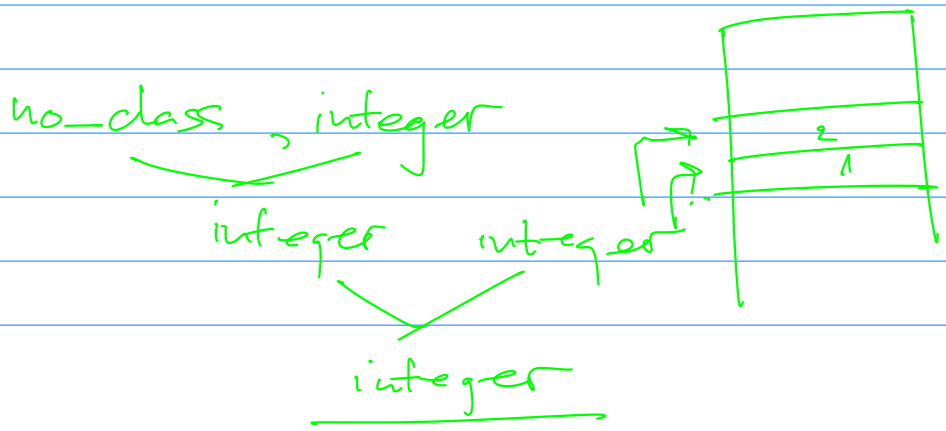

mov
add
return addr
buffer


ret addr
pushed rbp
& worm
0
0
malloc return
malloc return
buffer
worm addr split
rbp
rsp
rbp - 0x14
rbp - 0x20
rbp - 0x28
rbp - 0xC
0x28 → 40
0x48
4.16 + 8
40 + 24 + 8
64 + 8 → 72
0x14 → 16+4
= 20
64
16
80.



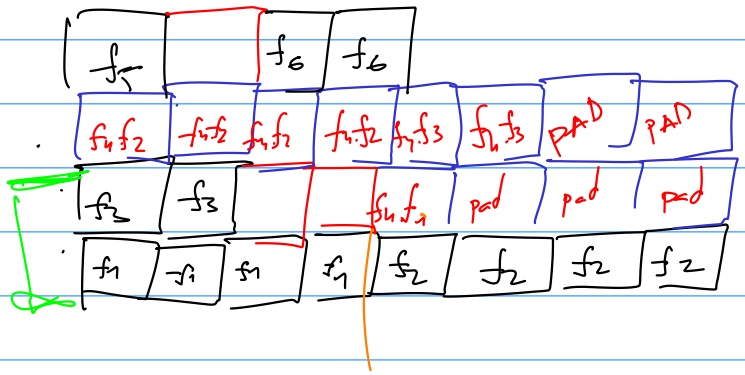

```
typedef struct
{
    char f1;    // size:  1, alignment: 1
    int f2;     // size:  4, alignment: 4
    short f3;   // size:  2, alignment: 2
} StructType1; // size: 12, alignment: 4
```



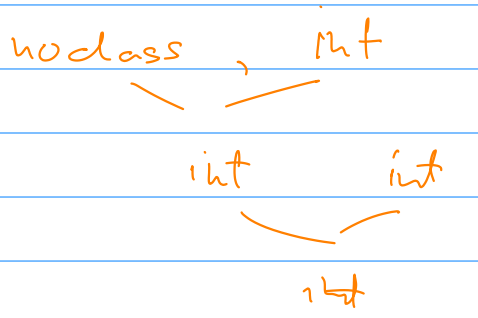


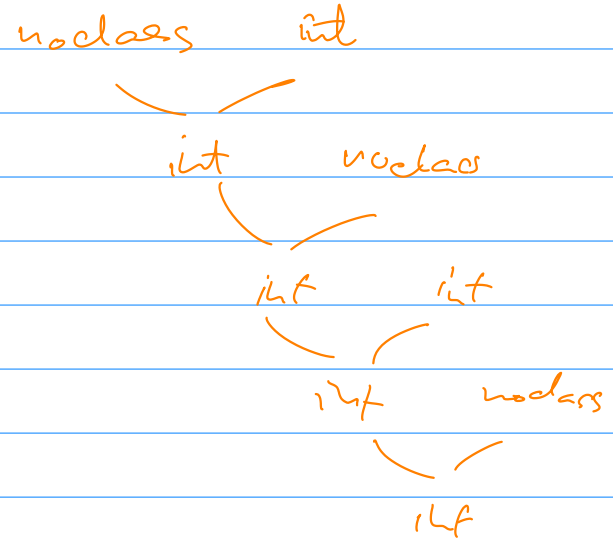
```
3 typedef struct
4 {
5     char f1;    // size:  1, alignment: 1
6     int f2;     // size:  4, alignment: 4
7     short f3;   // size:  2, alignment: 2
8 } StructType1; // size: 12, alignment: 4
9
10 /*
```

```
78 typedef struct
79 {
80     int f1;          // size:  4, alignment: 4
81     int f2;          // size:  4, alignment: 4
82     short f3;        // size:  2, alignment: 2
83     StructType1 f4; // size: 12, alignment: 4
84     char f5;         // size:  1, alignment: 1
85     short f6;        // size:  2, alignment: 2
86 } StructType2;       // size: 28, alignment: 4
87
88 /*
```

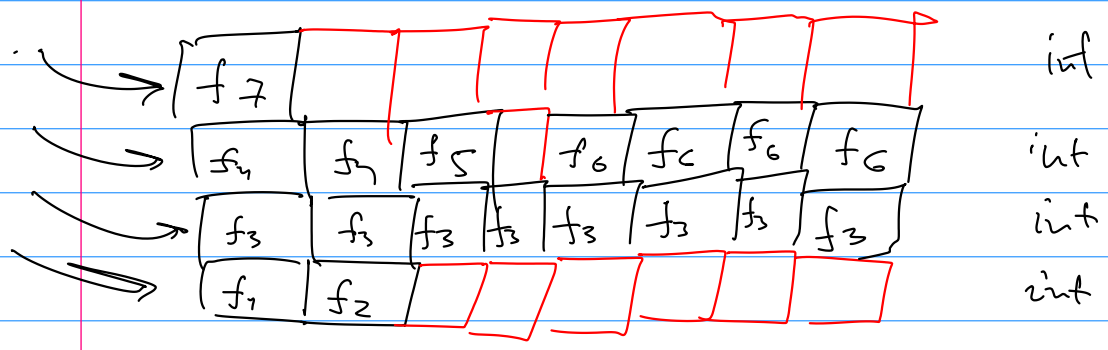


4 int
3 int
2 int
1 int



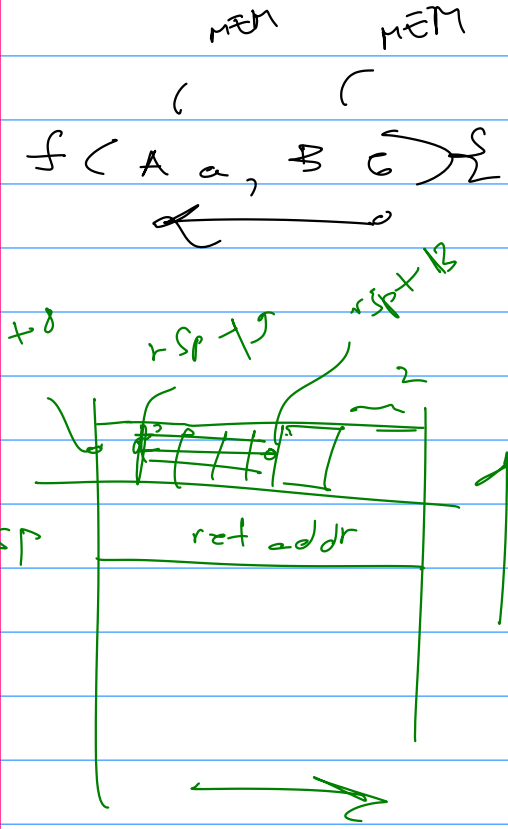


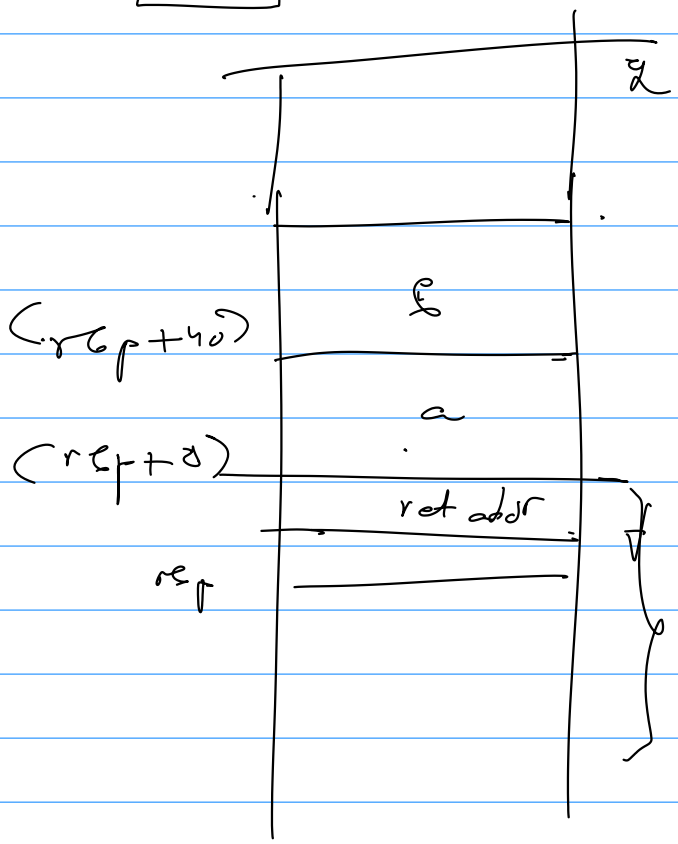
```
38 typedef struct
39 {
40     char f1;    // size:  1, alignment: 1
41     char f2;    // size:  1, alignment: 1
42     long f3;    // size:  8, alignment: 8
43     short f4;   // size:  2, alignment: 2
44     char f5;    // size:  1, alignment: 1
45     int f6;     // size:  4, alignment: 4
46     char f7;    // size:  1, alignment: 1
47 } StructType3; // size: 32, alignment: 8
48
```

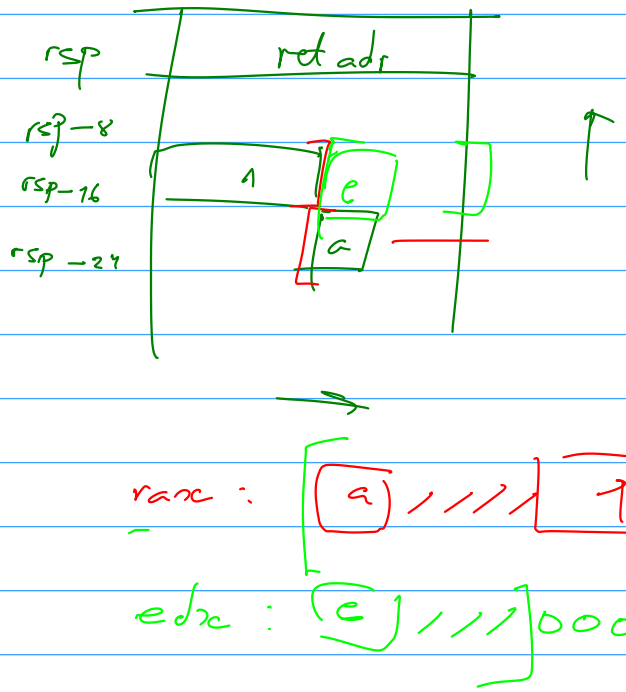


25 — 32

16B

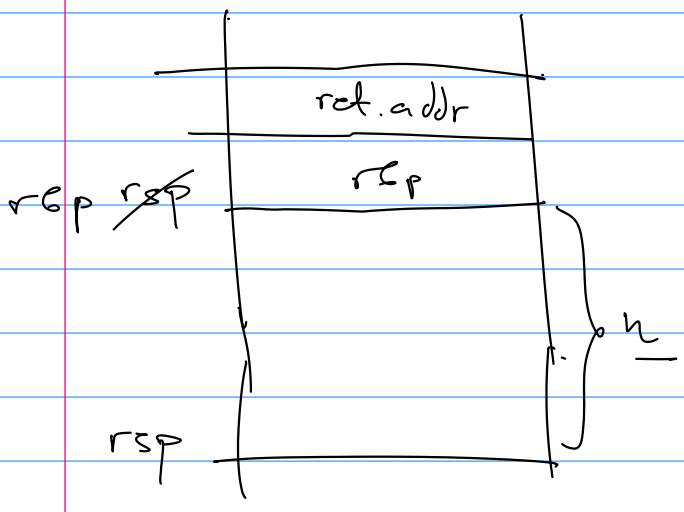






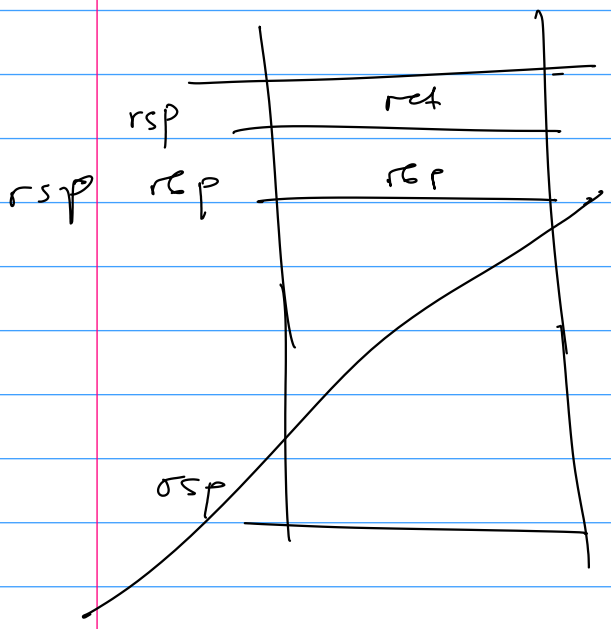
enter n, 0

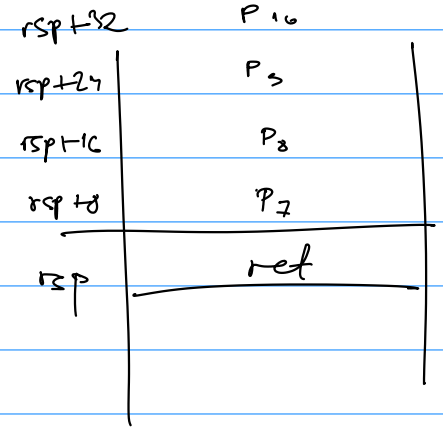
push rbp
mov rbp, rsp
sub rsp, n



leave
[ret]

. mov rsp, rbp
pop rbp



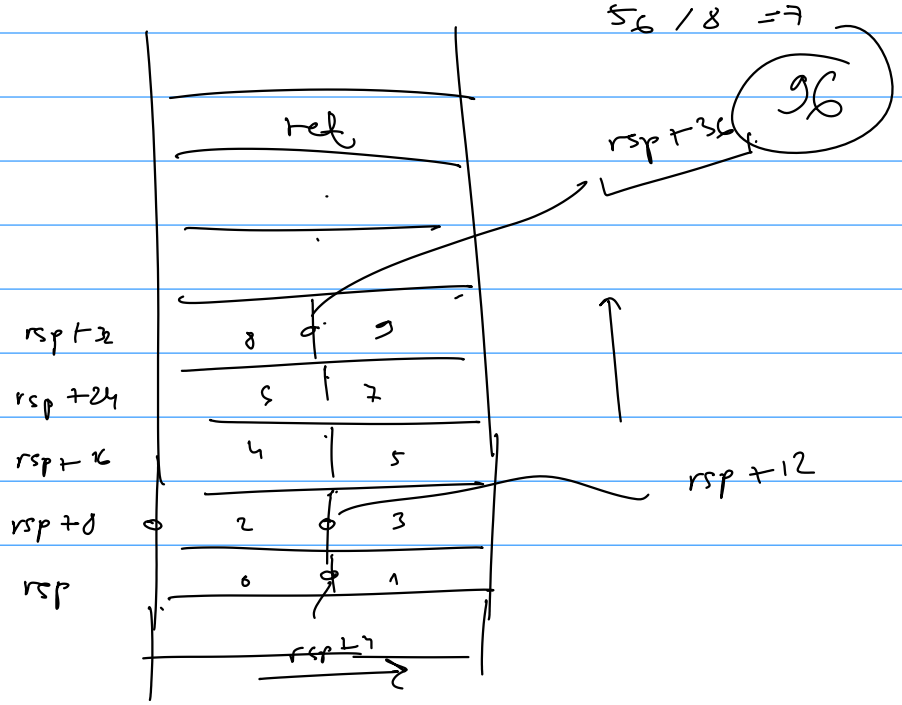


56 + 4.8
56 + 32
88

16
x2
48

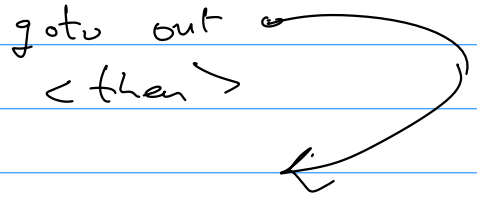
56 / 8 = 7

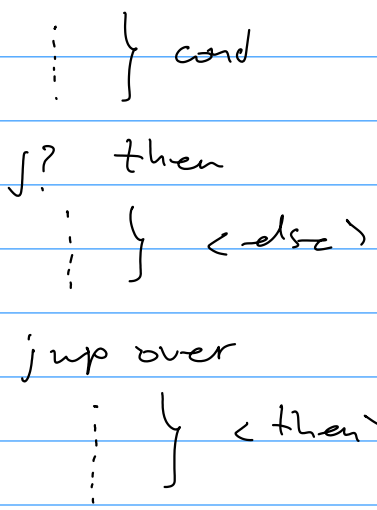
96



```
if (<cond>) {
    <then>
} else {
    <else>
}
```

```
if(<cond>)   goto then

    <else>
    goto out
then:  <then>
out:
```





```
do {
    <body>
} while (<cond>);
```

```
loop:
        <body>
        if (<cond>) goto loop
```

```
loop:  ⋮  } body
       ⋮  } cond
       j? loop
```

rare — cmov

```
x = <cond> ? val1 : val2;
```

```
if (cond) x = val1
else      x = val2
```

```
⋮  } <cond>
mov  rax, val1
cmov? rax, val2
⋮
```

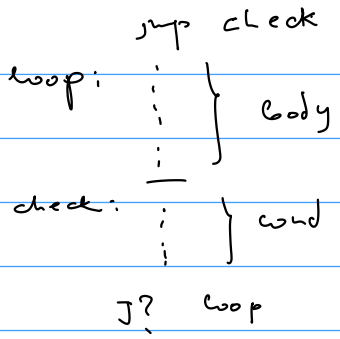
```
while (<cond>){
        <body>
}.
```

```
if (!<cond>) goto over
loop: <body>
      if (<cond>) goto loop
over:
```

goto check

loop : <body>

check: if(<cond>) goto loop.



for (init ; cond ; inc) {

body

}.

init

while (cond) {

body

inc

}.

init

jmp check

loop body

inc

check: cond

J? loop.

```
switch (<expr>) {
    case val1: <case1> ; break;
    case val2: <case2>
    default: <default>
}
```

```
x = <expr>
if (x == val1) goto case1
if (x == val2) goto case2
goto def
case1: <case1>
       goto over
case2: <case2>
def:   <default>
over:
```

≡

```
switch (<expr>) {
case 0:
case 1:
case 2:
.
.
.
case 10:
```

L = {case0, case1, ..., case10}

```
if (0 ≤ E && E ≤ 10) goto L[E];
goto default
```

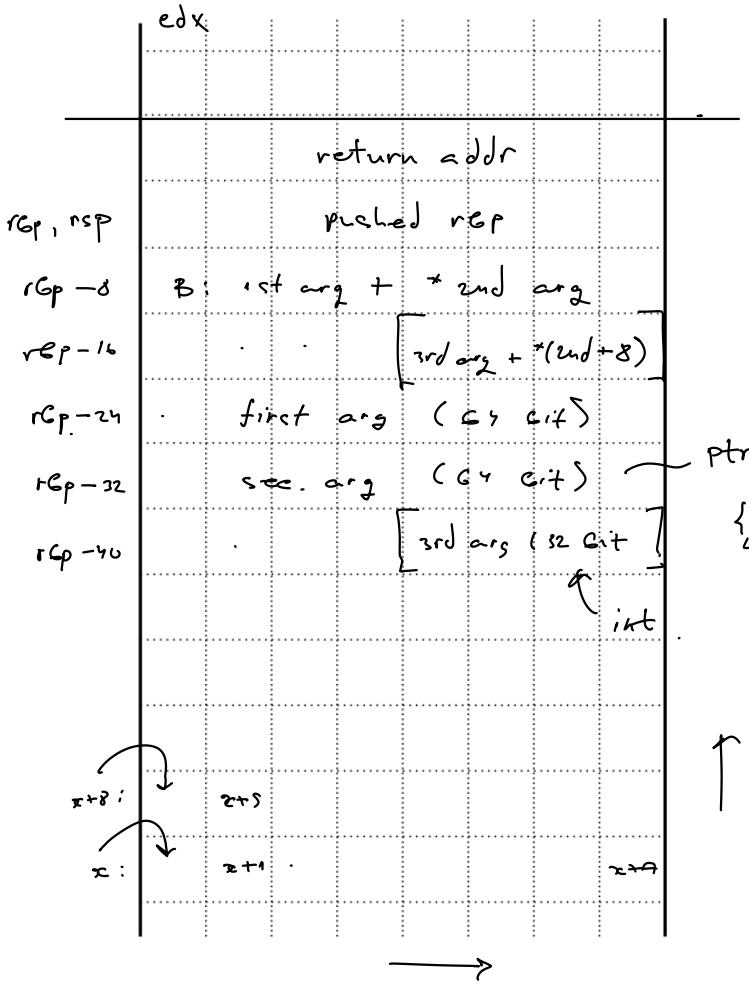
```
case 0:
case 1:
.
.
.
case 10:
```

```
.intel_syntax noprefix
.text
.global foo
.type foo, @function
foo:
    push rbp
    mov rbp, rsp
    mov QWORD PTR -24[rbp], rdi
    mov QWORD PTR -32[rbp], rsi
    mov DWORD PTR -36[rbp], edx
    mov rax, QWORD PTR -32[rbp]
    mov rdx, QWORD PTR [rax]
    mov rax, QWORD PTR -24[rbp]
    add rax, rdx
    mov QWORD PTR -8[rbp], rax
    mov rax, QWORD PTR -32[rbp]
    mov edx, DWORD PTR 8[rax]
    mov eax, DWORD PTR -36[rbp]
    add eax, edx
    mov DWORD PTR -12[rbp], eax
    mov eax, DWORD PTR -12[rbp]
    movsx rdx, eax                  # MOVe with Sign eXtend
    mov rax, QWORD PTR -8[rbp]
    add rax, rdx
    pop rbp
    ret
```

(eaf, 128

rax := 2nd arg
rdx := * 2nd arg
rax := 1st arg

(rdi, rsi, rdx, rcx, r8, r9)



rax := 2nd arg

edx := *(2nd arg + 8)

eax := 3rd arg

A = 3rd arg + *(2nd arg + 8)

rdx := signExten(A).

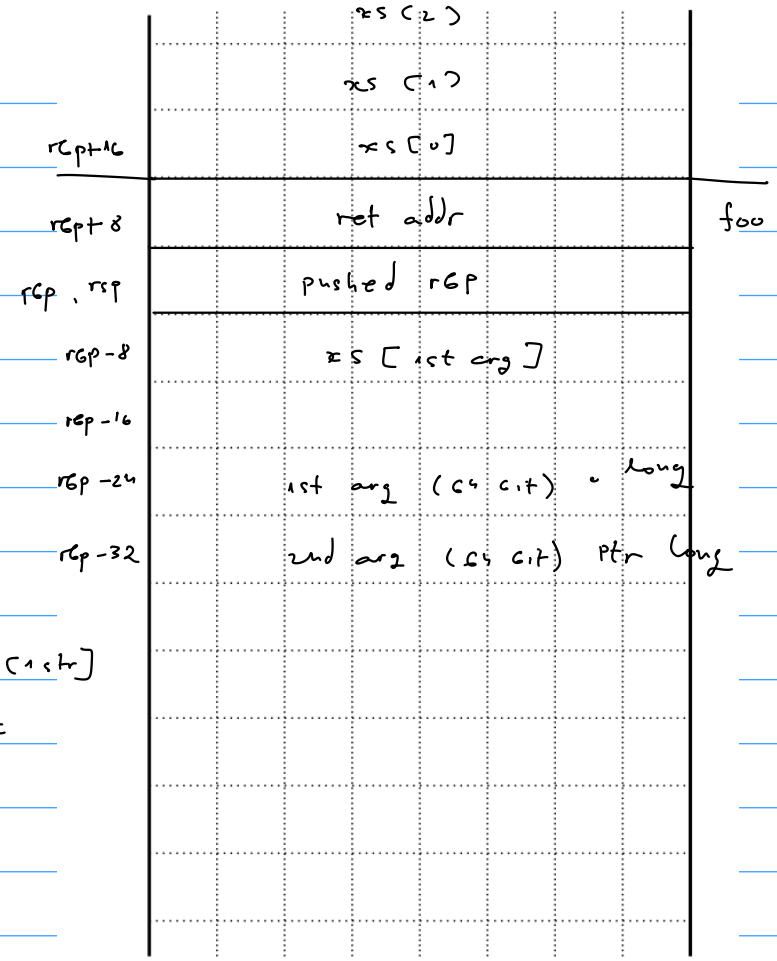
rax := A + B

```
long foo(long x, struct A *p, int y) {
    long A = y + p -> a2;
    long B = x + p -> a1;
    retur A + B;
}.
```

```
struct A {
    long a1;
    int a2;
}.
```

(rdi, rsi, rdx, rcx, r8, r9)

```
	.intel_syntax noprefix
	.text
	.global foo
	.type foo, @function
foo:
	push rbp
	mov rbp, rsp
	mov QWORD PTR -24[rbp],.rdi
	mov QWORD PTR -32[rbp],.rsi
	mov rax, QWORD PTR -24[rbp]
	mov rax, QWORD PTR 16[rbp+rax*8]
	mov QWORD PTR -8[rbp], rax
	mov rax, QWORD PTR -32[rbp]
	mov rdx, QWORD PTR [rax]
	mov rax, QWORD PTR -8[rbp]
	add rdx, rax
	mov rax, QWORD PTR -32[rbp]
	mov QWORD PTR [rax], rdx
	mov rax, QWORD PTR -8[rbp]
	pop rbp
	ret
```



*2nd += xs[1st] rax: = 1st

rax += 2nd

rdx: = *2nd. rax: = xs[1st]

rdx: = *2nd + xs[1st] = A

*2nd = A txs [1st]

rax: = xs [1st arg]

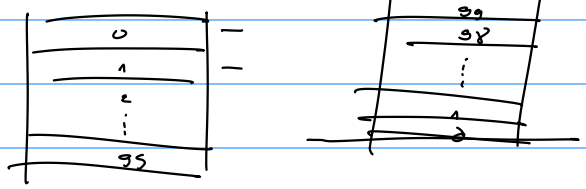
```
struct A {
    long xs [3];
};

long foo (struct A a, long idx, long *p) {
    long val = a.xs [idx];
    *p += val;
    return val;
}
```

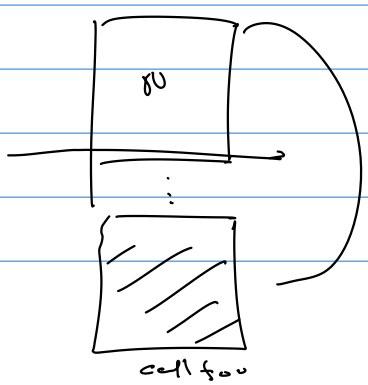
int xs[100] struct X {

foo (xs) int xs [100];

} s;

400, foo(s)





```
        .intel_syntax noprefix
        .text
        .globl foo
        .type foo, @function
foo:
        push rbp
        mov rbp, rsp
        sub rsp, 16
        mov QWORD PTR -8[rbp], rdi
        cmp QWORD PTR -8[rbp], 0
        je label_1
        mov rax, QWORD PTR -8[rbp]
        sub rax, 1
        push QWORD PTR 88[rbp]
        push QWORD PTR 80[rbp]
        push QWORD PTR 72[rbp]
        push QWORD PTR 64[rbp]
        push QWORD PTR 56[rbp]
        push QWORD PTR 48[rbp]
        push QWORD PTR 40[rbp]
        push QWORD PTR 32[rbp]
        push QWORD PTR 24[rbp]
        push QWORD PTR 16[rbp]
        mov rdi, rax
        call foo
        add rsp, 80
        # imul reg64, mem64
        # signed multiply
        # reg64 ← truncate(reg64 * mem64)
        imul rax, QWORD PTR -8[rbp]
        jmp label_2
label_1:
        mov rax, QWORD PTR 16[rbp]
label_2:
        leave
        ret
```

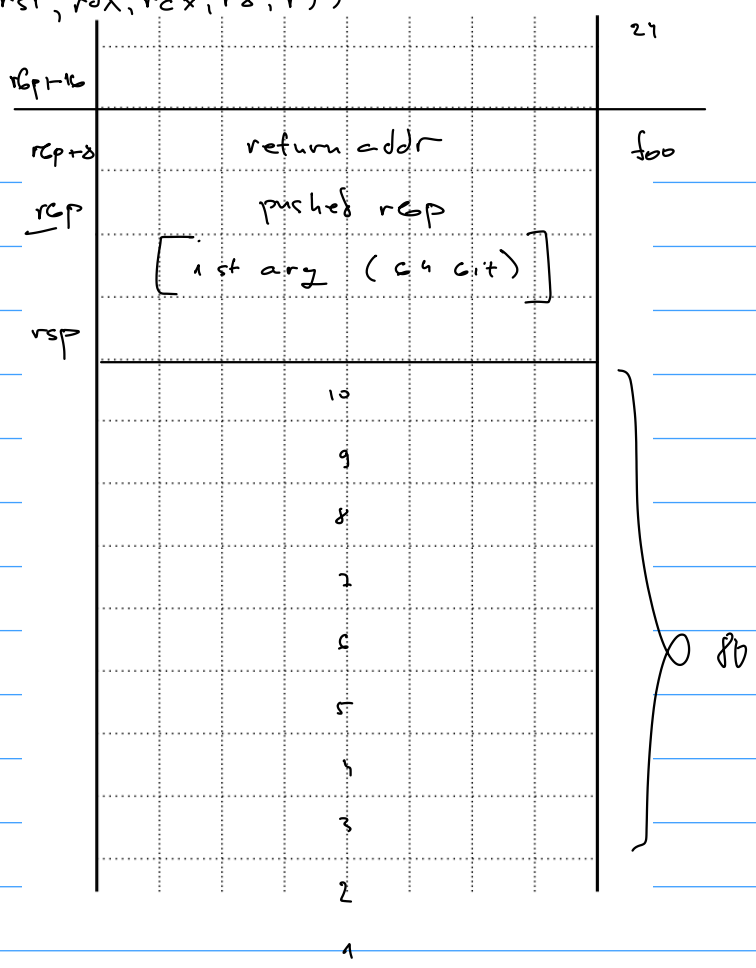
if (1st == 0) goto label

10 x 8

label:

foo(n-1,

(rdi, rsi, rdx, rcx, r8, r9)




```
struct A {
    long xs[10];
};

long foo(long n, struct A a) {
    if (n == 0) {
        return a.xs[0];
    } else {
        return n * foo(n-1, a);
    }
}
```