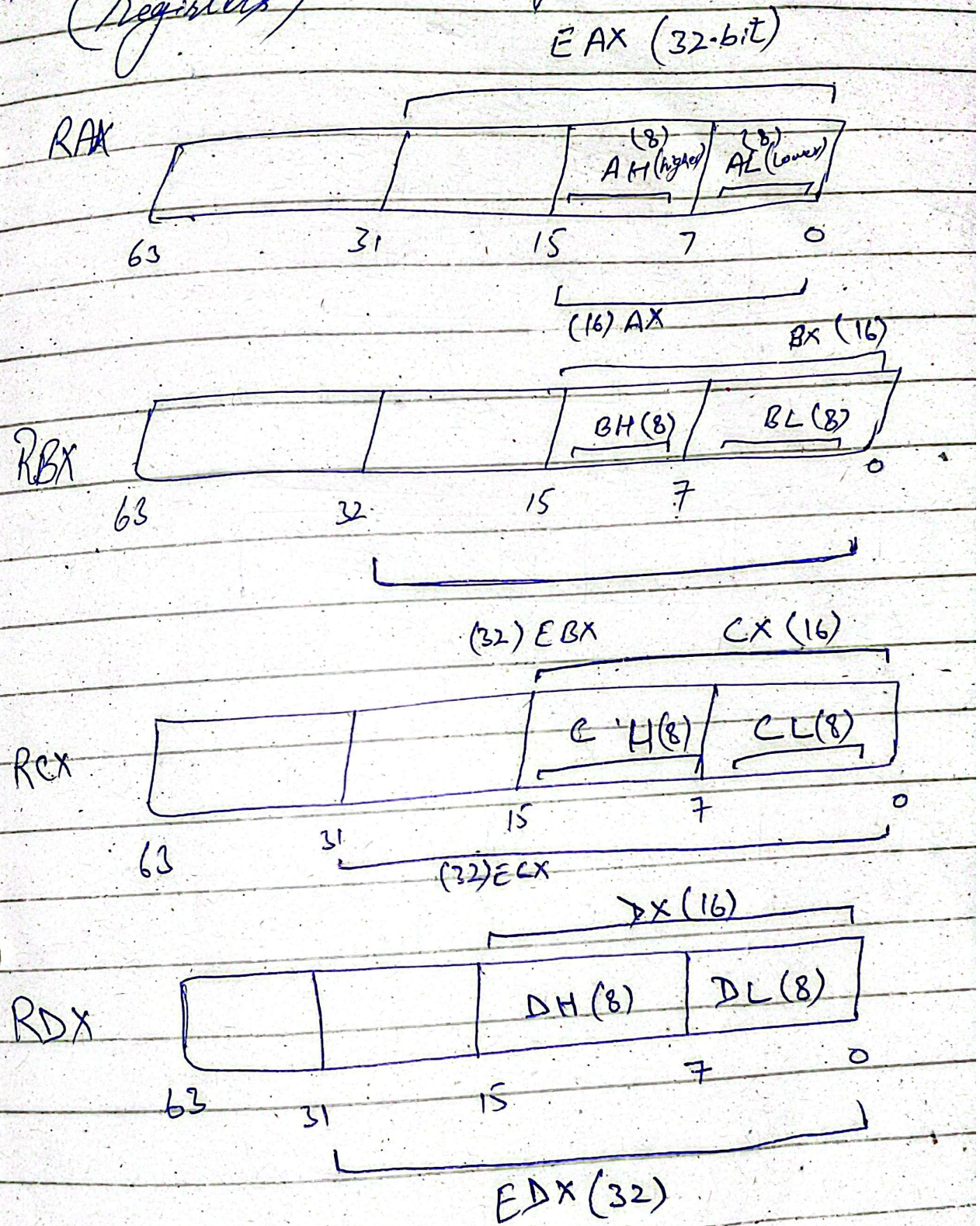
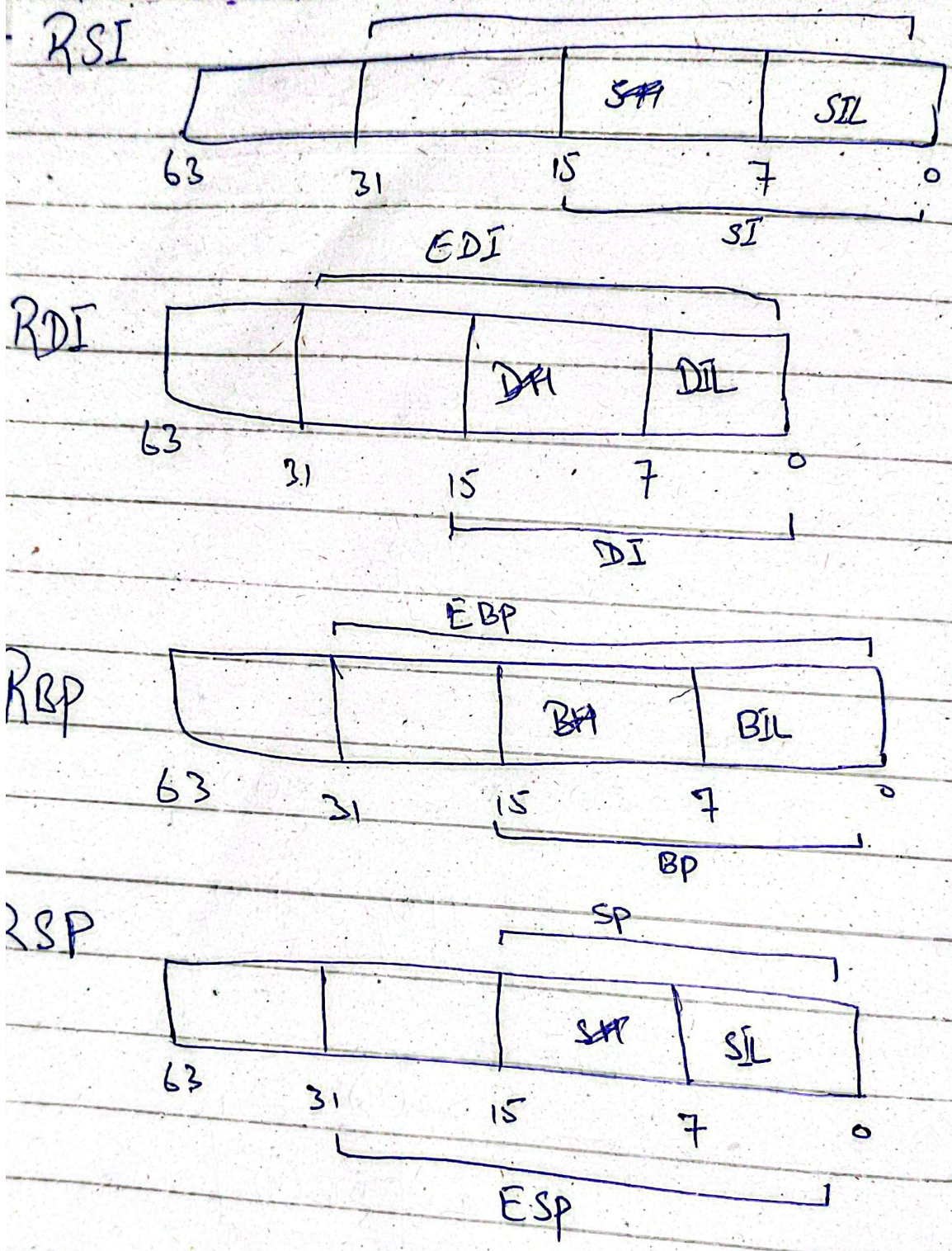
Reverse Enginnering
(Registers)



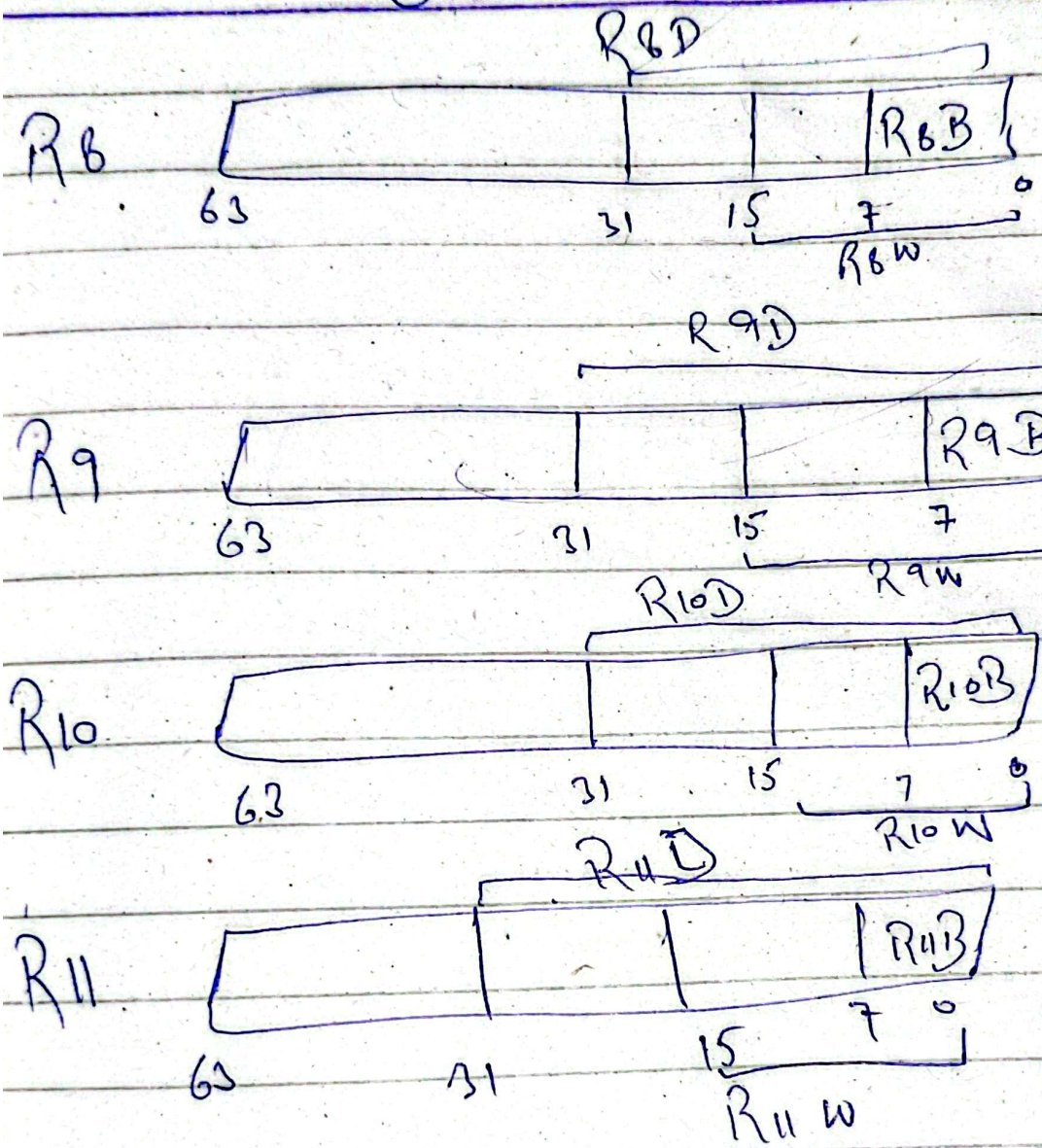
Same in 32 and 64 bit (OS)

Same in 64 and 32 bit (OS)



Higher Part of these registers Cannot be used so they have no names.

Additional in 64 bit (OS)



Same as

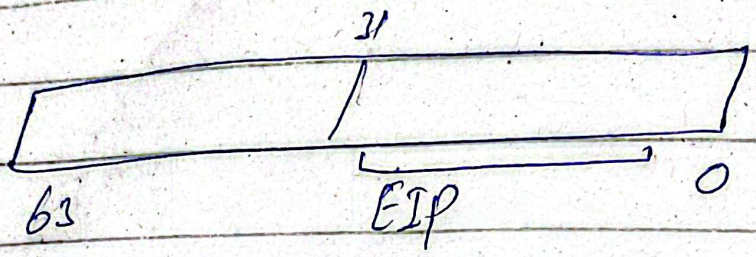
R12

R13

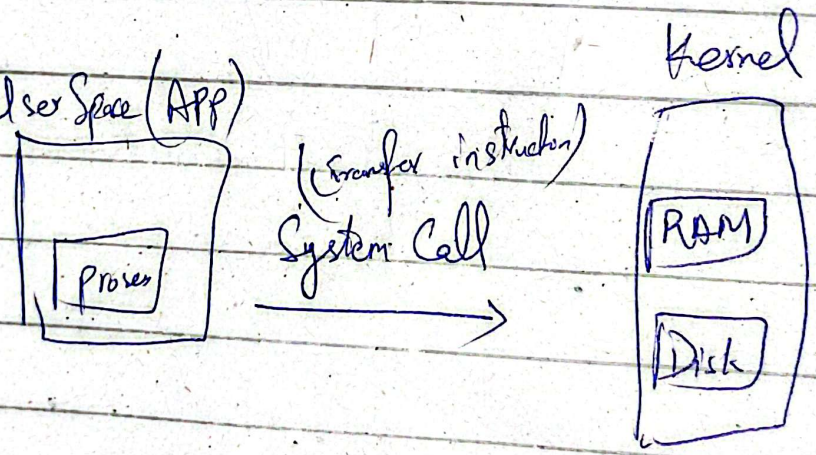
R14

R15

RIP (instruction Pointer / Program Counter)



» Tell whats about to execute in CPU
» Not general purpose Register



/usr/include/x86_64/asm/unistd_64.h

In OS develop we used C because we can easily provide instruction to kernel.

Three Types of STD in Linux

file descryptors

Std. input (0)
Std Output (1)
Std errow (2)
} file Descryptor

```
#include <stdio>
#include <stdlib.h>
main()
{
    write (1, "Hellow", 6);
    Exit (11); //(return 4 if Sucess)
}
```

```
$ gcc file.c -o write
$ ./write
```

Linux 64 bit Inted Syntex

1) intel Assembly
2) ATX Assembly

Assembly

Mov

Register	Value
RAX	0x0
RBX	0x0
RDX	0x0
RCX	0x0

Mov Destination, Source

Mov rax, 0x1 // move 1 to rax (register)

Mov rbx, 0xa // Hex a (10) move 10

Mov rcx, ~~0x0~~ rbx // ~~Hex b (11)~~ move rbx to rcx

~~Mov rax, 0xc~~ ~~Hex c (12)~~

Mov rax, rdx // move rdx to rax (overwrite)

Register	Value
RAX	0x1 0x0 (overwrite)
RBX	0xa
RDX	0xa
RCX	0x0

Add

ADD Designation, Source

ADD rax, 0x2

ADD rbx, 0x21 (decimal 33)

ADP rcx, rbx

ADD rax, rdx

Register	value
RAX	0x0
RBX	0x0
RCX	0x0
RDX	0x0

RAX	~~0x2~~ 0x2
RBX	0x21
RCX	0x21
RDX	0x0

In addition value remain (2)

SUB

SUB Destination, Source

Register	value
RAX	0x10
RBX	0x20
RCX	0x30
RDX	0x40

```
sub rax, 0x2
sub rbx, 0x9
sub rcx, rbx
```

0x20 (Hex(16)) ⇒ 32

$2 \times 16^1 + 0 \times 16^0 \Rightarrow 32 + 0 \Rightarrow 32_{(Dec)}$

```
sub rdx, rax
```

RAX	0x8E (0x16)
RBX	0x16
RCX	0x1a
RDX	0x38

0x16

$1 \times 16^1 + 6 \times 16^0$

⇒ 22

0x30

$3 \times 16^1 + 0 \times 16^0$

⇒ 48

~~$16 + 0 \times 16^0$~~

~~48 ⇒ 64~~

0x08

$0 \times 16^1 + 8 \times 16^0$

8 ⇒ 64 − 8 ⇒ 56

56 ÷ 16

⇒ 0x38

CMP (Compare)

CMP Dest, Source

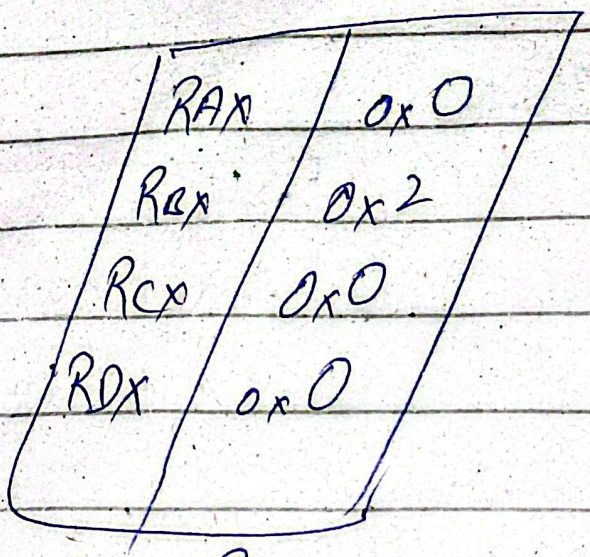
Here we can save value of one var to another and store Output (T, F) to flags (register)

Regrts	
RAX	0x1
RBX	0x2
RCX	0x3
RDX	0x4

flags help and work as Zero flag = value
Carry flag to carry/borrow any value./s.
Zero flag help to store Compared result.

CMP rax, 0x1 1=1 ZF = 1 (flag value)
CMP rbx, 0x3 2=3 ZF=(0)
CMP rcx, rbx 3=2 ZF=0
CMP rdx, 0x4 4=4 ZF=1

Test



Test Desgnation , Source

It work as (bit wise logic and)

Test rax, rax 0=0 ZF = 1

Test rbx, rbx 2=0 ZF = 0

~~Test~~ RC*

same as next

JMP (Jump bit. Code)

Unconditional Jump

JMP Designation (address)

JM 0x0804d6e2

JMP RAX

Register	Value
RAX	0x0804d6e2
RBX	0x0
RCX	0x0
RDX	0x0

JE / JZ (Conditional Jump)

(Jump if equal / Jump if zero)

If zero flag value is 1 (two value zero) it will Jump

Register	value
RAX	0x0804d6e2
RBX	0x0
RCX	0x0
RDX	0x0

JE 0x0804de2 = 0

Jump if zero ZF = 0

JZ. 0x0804de2 = 0

if zero flag set ZF = 0

JNE / JNZ (Jump if not zero / Equal)

JNE / JNZ Distination

1. JNE 0x0804d6e2

0≠0x0804d6e2

ZF = 0 (Jump)

jump if ZF value is zero

	0
RAX	0x0804
RBX	0x0
RCX	0x0
RDX	0x0

Jump

Not Jump

2. JNZ (Jump if not zero)

lets ZF=1 if zero flag value zero No jump

~~0≠0x0804d6e2~~ (No jump)

CALL

Jump from one to another Code

Used to call functions

→ CALL Designation

It will Jump execute and return on Previous (original) place.

(CALL ~~0x~~0804d6e2

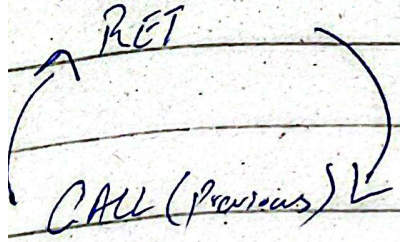
0x0804d6e2 ↓)

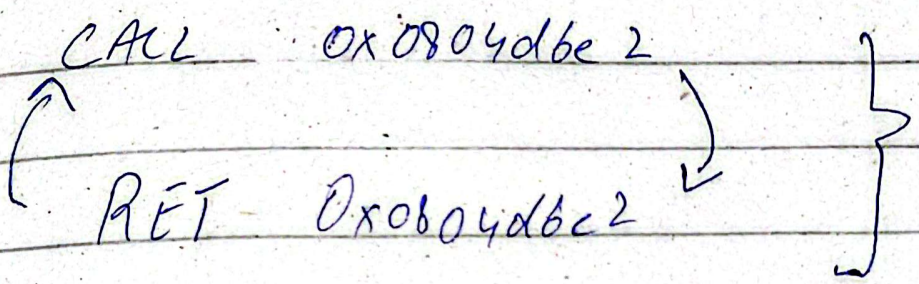
(CALL eax

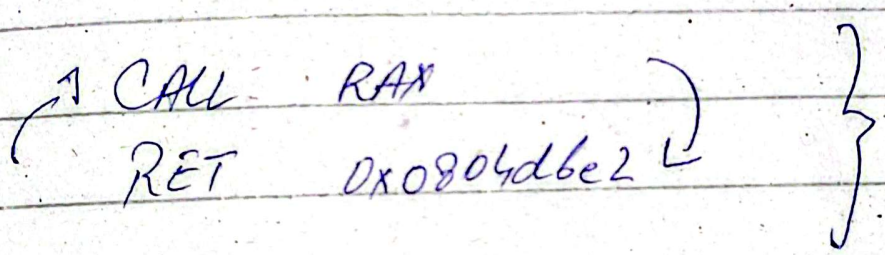
0x0804d6e2 ↓)

RET (return)

Return Write back Call to original place







SYS CLALL

Call kernel to perform action using sys call

Reg	value	
RAX	0x3c	(60)
RDI	0x0	(11)
RSI	0x0	
RDX	0x0	

Kernel known about sys call using values.

SYSCALL RAX (60 N) (Exit call)

only one argument

if argument provide to first

RDI, RSI, RDX, RCX, R8.....

Syscall exit (11) => exit RDI