Reversing(2)

~アセンブリツールとプログラミングを使う問題編~

2年 小俣直史

チャレンジ1:Rev2a(実行ファイル)

復習:

r2 [ファイル名]

-> aaaaaa(解析)

-> VV@[関数名] (命令実行マップ)

　　(最初は main を見て、それから

　　目星をつけた関数に入るのがおすすめ)

(Esc キーで操作取り消し・戻る)

↑と前回のReversingのパワポ資料を参照しながらやってみましょう

[注意]ファイルは最初に sudo chmod +x で権限を付与してあげましょう!!

var_8 = input_code():
入力用の関数で受け付けた値(整数値)を代入

check_code(var_8):
先の整数値をフラグと比較して分岐させる関数

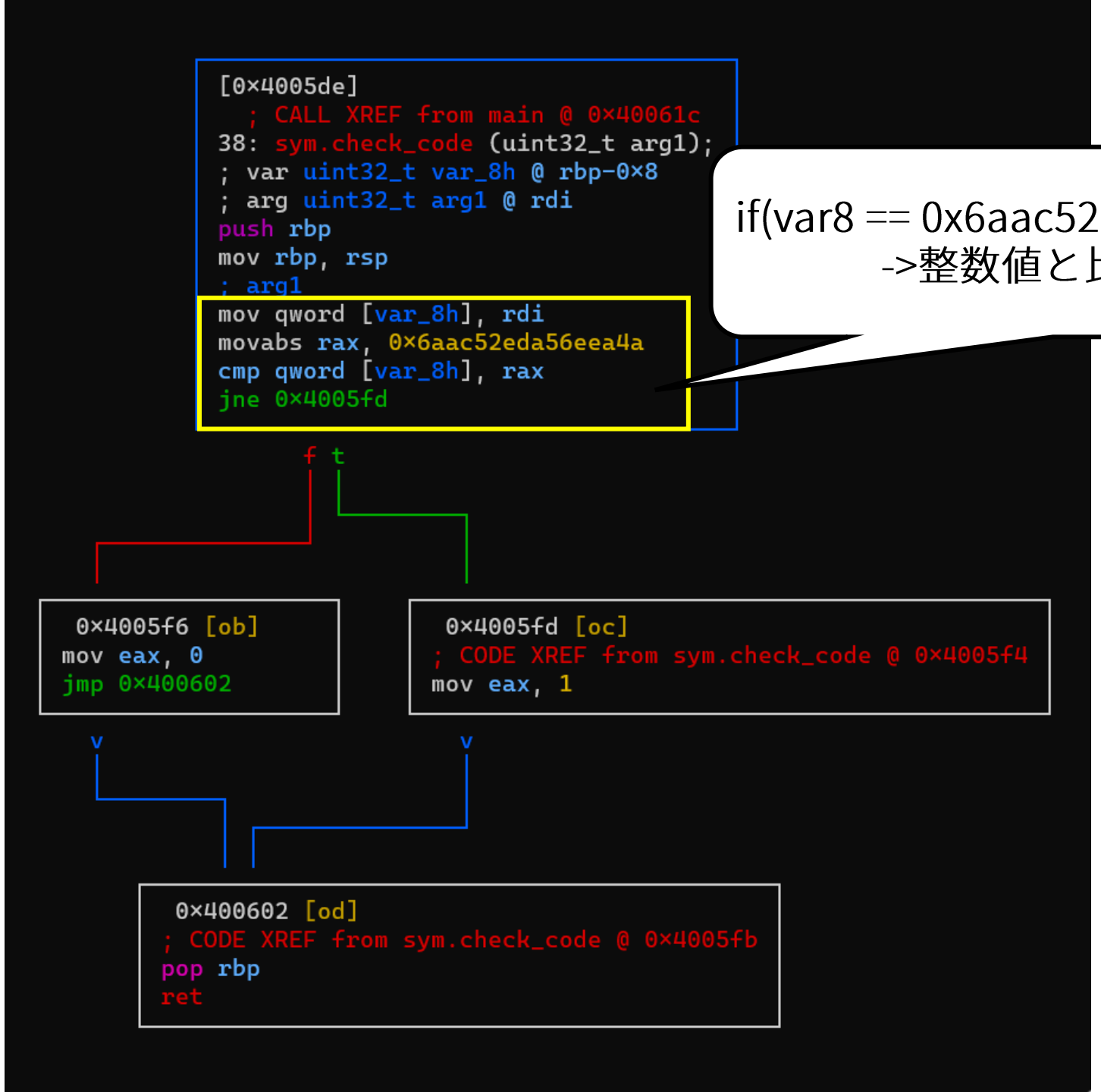
```
[0x4005a7]
  ; CALL XREF from main @ 0x40060c
55: sym.input_code ();
; var int64_t var_8h @ rbp-0x8
push rbp
mov rbp, rsp
sub rsp, 0x10
; const char *format
; 0x4006e4
; "LISENCE CODE: "
lea rdi, qword str.LISENCE_CODE:
mov eax, 0
; int printf(const char *format)
call sym.imp.printf;[oa]
lea rax, qword [var_8h]
mov rsi, rax
; const char *format
; 0x4006f3
; "%llu"
lea rdi, qword str.llu
mov eax, 0
; int scanf(const char *format)
call sym.imp.__isoc99_scanf;[ob]
mov rax, qword [var_8h]
leave
ret
```

= printf(“LISENCE CODE: ”)

```
ubuntu@GLANGZEL:~$ ./rev2a
LISENCE CODE:
```

= scanf(“%llu”, &var_8)
(var_8:符号なし整数値 int)

Return var_8:



if(var8 == 0x6aac52eda56eea4a)
->整数値と比較して命令を分岐

入力した値をそのまま比較しているだけ

ただ、scanfで%lluを受け付けているため、16進数を 10進数に変換してから入力(%x であれば16進数値なのでそのまま入力してよい。要確認)

0x6aac52eda56eea4a = 76866098446508304l0

```
ubuntu@GLANGZEL:~$ ./rev2a
LISENCE CODE: 768660984465083041O
Correct!
The flag is ctf4b{768660984465083041O}
ubuntu@GLANGZEL:~$
```

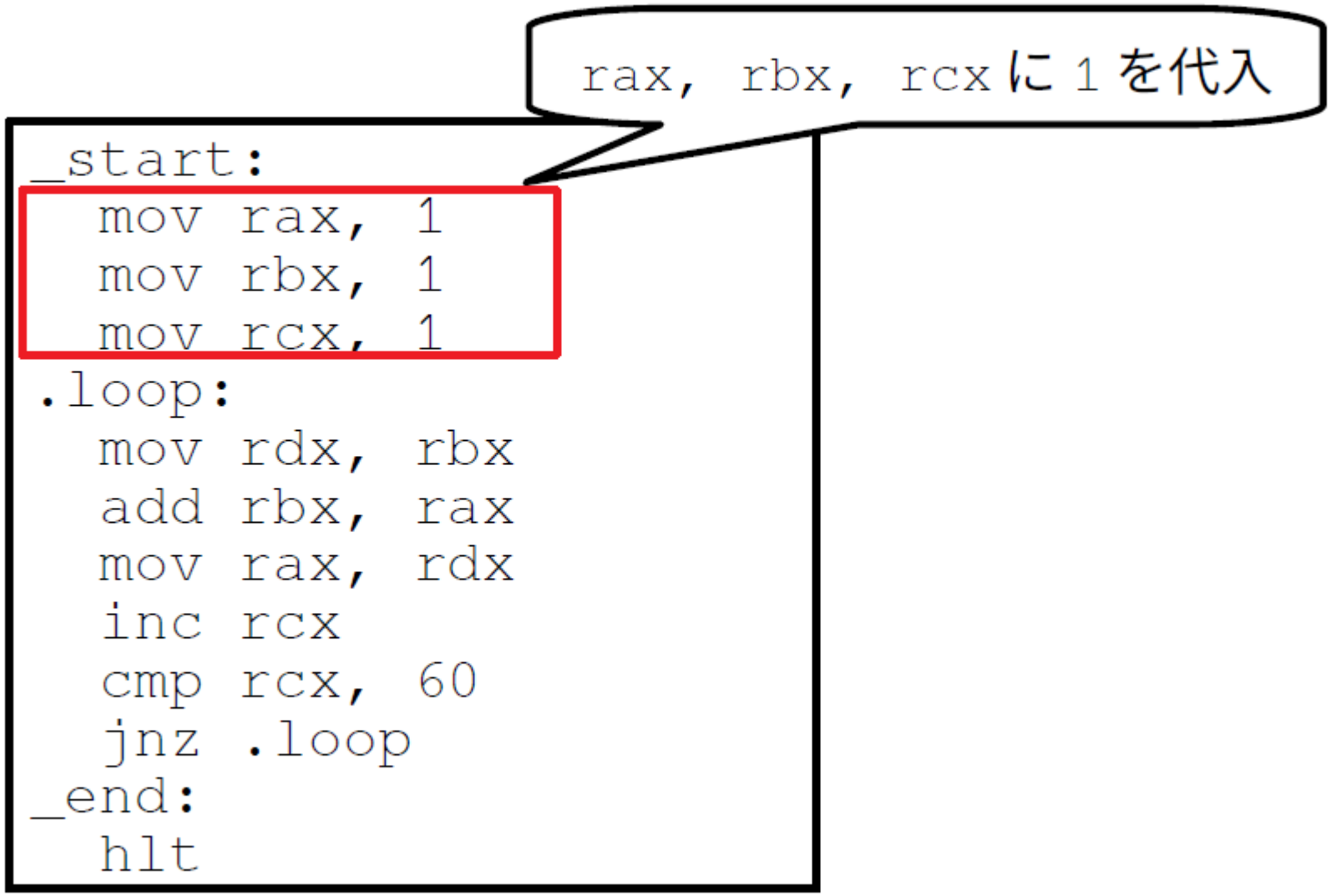
チャレンジ1:Rev2b(アセンブラファイル)

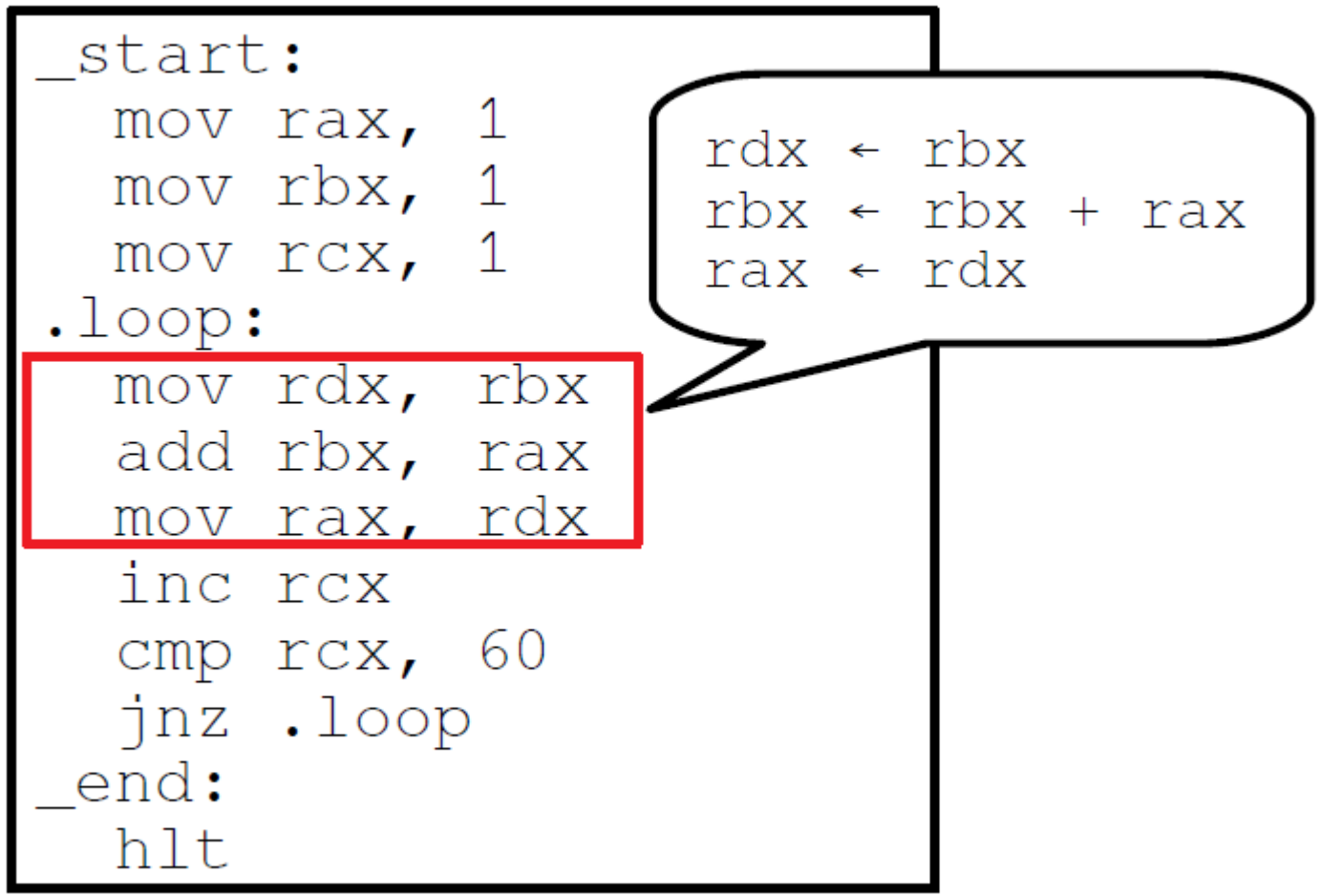
FLAG形式は ctf4b{処理後のrax レジスタの値}

[注意]
_[関数]:
　　処理
　　…
　　…

前回のReversingのパワポ資料を参照しながらやってみましょう
[注意]ファイルは最初に sudo chmod +x で権限を付与してあげましょう!!

```
_start:
  mov rax, 1
  mov rbx, 1
  mov rcx, 1
.loop:
  mov rdx, rbx
  add rbx, rax
  mov rax, rdx
  inc rcx
  cmp rcx, 60
  jnz .loop
_end:
  hlt
```


rax, rbx, rcx に 1 を代入
_start:
mov rax, 1
mov rbx, 1
mov rcx, 1
.loop:
mov rdx, rbx
add rbx, rax
mov rax, rdx
inc rcx
cmp rcx, 60
jnz .loop
_end:
hlt


_start:
mov rax, 1
mov rbx, 1
mov rcx, 1
.loop:
mov rdx, rbx
add rbx, rax
mov rax, rdx
inc rcx
cmp rcx, 60
jnz .loop
_end:
hlt
rdx ← rbx
rbx ← rbx + rax
rax ← rdx

```
_start:
  mov rax, 1
  mov rbx, 1
  mov rcx, 1
.loop:
  mov rdx, rbx
  add rbx, rax
  mov rax, rdx
  inc rcx
  cmp rcx, 60
  jnz .loop
_end:
  hlt
```

rcx をインクリメント
rcx が 60 ならループを終了

つまり、

rax,

rbx = rbx,

rax + rbx

を59回繰り返す

-> 59番目の rax を計算する

```
# Python 3

a, b = 1, 1
for i in range(59): #59回繰り返す
        a, b = b, a + b
print(“ctf4b{{{}}}”.format(a))  #ctf4b{[aの値]}のように出力
```

つまり

```
>>> print("ctf4b{{{}}}".format(a))
ctf4b{1548008755920}
>>>
```

CTF Reversingの問題には、プログラムを使う問題も結構あります。
「ctf reversing write up」な感じで調べると各種CTFの解説がみられるのでぜひ参考にしてみてください
(大体Python3 を使ってるイメージ?)