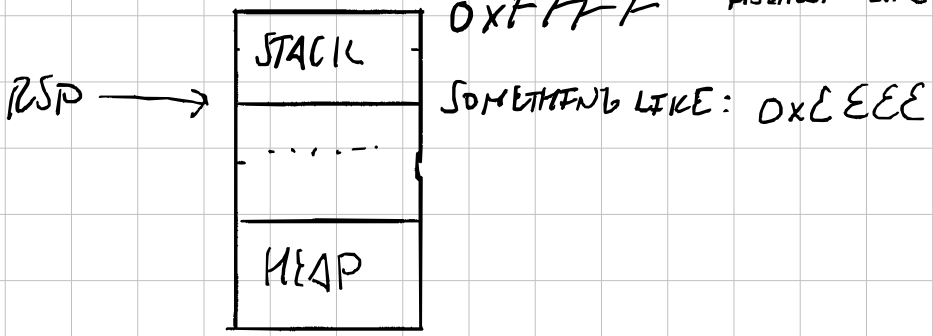
1	SUB	RSP, 0x8
2	CMP	EDI, 0x2
3	JNZ	LAB_00101lcf
4	MOV	RSI, qword Ptr [RSI+0x8]
5	MOV	EAX, 0x70
6	MOV	ECX, 0x0
7	LEA	RDI, [s_password1_00102037]

1. RSP REGISTER IS THE STACK POINTER. IT POINTS TO THE TOP OF STACK.



SUB RSP, 0x8 will decrement RSP by 8 BYTES. IN THE DIAGRAM, WE WOULD SEE RSP THEN BE MOVED DOWN CLOSER TOWARDS THE HEAP. THE STACK GROWS DOWNWARD. SO, WE HAVE 8 BYTES AVAILABLE ON STACK FOR SOMETHING.

2. CMP EDI, 0x2

Here, we compare the EDI register with the Hex #2.
This will set some flags for us to use.

3. JNZ LAB_001011cf

JNZ is short for JUMP IF NOT ZERO. It is equivalent to JNE (JUMP IF NOT EQUAL). So, if the arguments in 2. are equal, the program WILL NOT JUMP. If they are not equal, we will JUMP to LAB_001011cf

LOOKING AT LAB_001011cf NOW.

```
1 LEA   RDI,[s_Need_exactly_one_argument._001020004]
2 CALL  <EXTERNAL>::puts
3 MOV   EAX, 0xffffffff
4 JMP   LAB_001011ca
```

1 HERE WE LOAD THE ADDRESS OF A VALUE into RDI, RDI IS A REGISTER FOR STORING THE FEIRST ARGUMENT TO A FUNCTION

2 WE THEN CALL THIS FUNCTION
- ↳ FMAGE
- ↳ TAKES a pointer to A STRING
- ↳ THAT ARGUMENT IS THE THING WE JUST PUT IN RDI
- ↳ RETURNS An Int
- ↳ EAX HOLDS THE RETURN VALUE

CALL ALSO PUSHES ADDY OF NEXT INSTRUCTION ON STACK

3. THIS MOVES THE VALUE 0xFFFFFFFF into EAX

↳ WHY EVEN THOUGH WE JUST GOT SOMETHING RETURNED VIA THAT REGISTER? 2's COMP

↳ I don't know. ↙

↳ That Hex is -1, presumably to be returned

4. WE JMP TO LAB_00110ca

1 ADD RSP, 0x8
2 RET

1. we allocated 8 Bytes on stack Earlier, we now need to CLEAN UP. RSP now has RETURN ADDRESS: which is the return Address That was placed on the stack by the OS when the program started

2. NOW we POP VALUE FROM TOS WHICH IS RETURN ADDRESS. THAT IS THEN MADE RIP: our instruction pointer.
   - THAT VALUE IN EAX IS RETAINED, AND somehow RETURNED.

<TEXT discard>

1	SUB	RSP, 0x8
2	CMP	EDI, 0x2
3	JNZ	LAB_00101lcf
4	MOV	RSI, qword ptr [RSI+0x8]
5	MOV	EAX, 0x70
6	MOV	ECX, 0x0
7	LEA	RDI, [s_password_1_00102037]

PICK UP FROM 3, THE JUMP IF NOT EQUAL. SO, WHAT IF THEY'RE EQUAL?

Line 4: WE MOVE A VALUE INTO RSI. RSI IS FOR THE 2nd ARGUMENT.

- qword is Quad word
  - ↳ 64 Bit

MOVES A QUADWORD STARTING @ RSI +0x8 TO RSI.

→ THIS LINE essentially WE move THE USERS argument into RSI.

→ +0x8 because argv sized
  - ↳ argv[1] is what we want.

```
1  SUB     RSP, 0x8
2  CMP     EDI, 0x2
3  JNZ     LAB_00101cf
4  MOV     RSI, qword ptr [RSI + 0x8]
5  MOV     EAX, 0x70
6  MOV     ECX, 0x0
7  LEA     RDI, [s_password1_00102037]
```

Line 5: moving decmal value of 112 into EAX reg... OKAY!

Line 6: moving decmal value of 0 into ECX reg...

Line 7: loads effective address of "password1" into RDI...

↳ no! thats not the pass!

8	MOVZX	EDX, byte ptr [RSI + RCX * 0x1]
9	TEST	DL, DL
10	JZ	LAB_00101lac
11	MOVSX	EAX, AL
12	SUB	EAX, 0x1
13	MOVSX	EDX, DL
14	CMP	EAX, EDX
15	JNZ	LAB_00101le2

8: EDX REG CONTAINS

MOVZX is Move w Zero-Extend

RSI is 2nd Arg: our user's argument

RCX IS 4th Arg, we don't have!

RSI + (RCX * 0x1)

= ? + (0 * 1)

= ? + 0

= ?

? = THE VALUE TO BE COPIED

→ we read it as a single byte and extend to dword (32 bits)

ex: 0x3A BECOMES: 0x0000003A

- SO RSI IS 64 BITS, which is user's input
- FIRST BYTE OF THAT IS TAKEN AND EXTENDED to 32 bits: 24 bits added

```
8   MOVZX   EDX, byte ptr [RSI + RCX * 0x1]
9   TEST    DL, DL
10  JZ      LAB_001011ac
11  MOVSX   EAX, AL
12  SUB     EAX, 0x1
13  MOVSX   EDX, DL
14  CMP     EAX, EDX
15  JNZ     LAB_001011e2
```

9: THIS WILL SET FLAGS
DL IS THE LOWER 8bits of EDX. WE CHECK IF= Zero or neg + set flags
↳ THIS DOES AND operation

10: JZ, JUMP if equal/0.
↳ so if flag set by 9 for being zero, we will jump

Assuming the And is ==0 WE GO TO LAB_001011ac

TALLEN JUMP

```
1 MOV   RDX, RSI
2 LEA   RSI, [s_Yes_%s_is_correct!_00102041]
3 MOV   EDI, 0x2
4 MOV   EAX, 0x0
5 CALL  <EXTERNAL>::printf_chk
6 MOV   EAX, 0x0
```

1: RDX GETS RSI's VALUE
User's argument

2: LOADS EFFECTIVE ADDRESS OF THAT FCN
s_Yes_%s_...~ . NOT YET CALLED

3: EDI == 0x2
decimal 2

4: EAX == 0x0
decimal 0

5: CALL FCN, SAFE TO ASSUME RSI GETS PRINTED

6: EAX == 0 again?
Y That'll BE OUR RETURN VALUE

7. ADD RSP, 0x8
8. RET

7: From 9, when called, it pushes return ADDRESS ONTO STACK SO to do that it had to decrement RSP by 0x8, to undo we increment RSP BY 0x8

8: Exit with EAX being EAX & 0x50

So we want user's input that when AND'ed it'll be zero.

Try "" empty

works!

LET'S ICE'EM GOING. FF JZ IS NOT TAKEN

1	MOVSX	EAX, AL
2	SUB	EAX, 0x1
3	MOVSV	EDX, DL
4	CMP	EAX, EDX
5	JNZ	LAB_00101ed

1: MOVSX IS MOVE W SIGN EXTEND
VAL OF AL IS SIGN-EXTENDED
AND PUT IN EAX
↳ 01110000 → 00000000 01110000

EAX HAS 0x70 0x70$_{16}$ or 112$_{10}$
AL is Lower BYTE OF THAT REG.

2: WE SUBTRACT 1 FROM EAX...
MMM. 112 → 111
Probably ASCII.

3: Like 1, But sign extending sub-regs
DL (8-bits) to EDX (32-bits)

1	MOVSX	EAX, AL
2	SUB	EAX, 0x1
3	MOVSX	EDX, DL
4	CMP	EAX, EDX
5	JNZ	LAB_00101d1d

4: compare EAX to EDX

EAX = 111_{10}

EDX = one byte from EDX's lowest sub-reg, DL

↳ this is one letter from the user argument

we compare 111_{10} to $?_{10}$

? = user argument first char

5: when comparing; if it does not ==> we force jump.

IT APPEARS IT WANTS USERS FIRST CHAR TO MATCH WITH THAT ASCII VALUE

THAT JUMP, LAB_0011ed, IS AN EXIT COND. I'M GONNA IGNORE THAT JNZ. WE DO NOT TAKE JUMP.

1	ADD	RCX, 0x1
2	MOVZX	EAX, byte ptr [RDI + RCX*0x1]
3	TEST	AL, AL
4	JNZ	LAB_0010118b

1: INCREMENTING RCX BY 1
→ RCX IS NOW 1

2: RDI + RCX*1
RDI + RCX
GETS NEXT BYTE in "password"
C THEO

3: TEST IF ITS ZERO

4: IF ITS NOT ZERO, THEN CALL THIS FCN AGAIN

EAX IS ONE OF THOSE CHARACTERS FOR "password1"

ECX IS Incrementer, probably in C This would be 'i'

EDX IS IS ONE OF THOSE CHARACTERS FOR use's argument

IF IT DOES NOT JUMP, we've reached the end and do not COM for again.

we Then Move to LAB_00101ac