**Assembly Challenges Writeup**

**Username:** aga8870

**Unique ID:** 52621

**Level1**

To complete this challenge, I had to set rdi = 0x1337. I used mov to do this. Thus, the challenge was solved.

Assembly instructions:

.intel\_syntax noprefix

mov rdi, 0x1337

**Level2**

To complete this challenge, I had to set rax = 0x1337, r12 = 0xCAFED00D1337BEEF, and rsp = 0x31337. I used mov to do this. Thus, the challenge was solved.

Assembly instructions:

.intel\_syntax noprefix

mov rax, 0x1337

mov r12, 0xCAFED00D1337BEEF

mov rsp, 0x31337

**Level3**

To complete this challenge, I had to compute rdi + 0x331337. I used add to calculate the answer. Thus, the challenge was solved.

Assembly instructions:

.intel\_syntax noprefix

add rdi, 0x331337

**Level4**

To complete this challenge, I had to compute f(x) = mx + b, where m = rdi, x = rsi, and b = rdx. I used imul to calculate rdi\*rsi and add to calculate the answer. Thus, the challenge was solved.

Assembly instructions:

.intel\_syntax noprefix

mov rax, rdi

imul rax, rsi

add rax, rdx

**Level5**

To complete this challenge, I had to compute speed = distance / time, where distance = rdi, time = rsi, and speed = rax. I used div to calculate this and obtained the flag.

Assembly instructions:

.intel\_syntax noprefix

mov rdx, 0

mov rax, rdi

div rsi

**Level6**

To complete this challenge, I had to compute rdi % rsi and place the answer in rax. I did this by using div. Div calculates rax/reg, and the remainder is placed in edx (the lower 4 bytes of rdx). So, I just moved the remainder from rdx to rax and obtained the flag.

Assembly instructions:

.intel\_syntax noprefix

mov rax, rdi

div rsi

mov rax, 0

mov eax, edx

**Level7**

To complete this challenge, I had to set the upper 8 bits of the ax register to 0x42. The register encompassing the upper 8 bits of the ax register is the ah register. So, if we set ah to 0x42, we obtain the flag.

Assembly instructions:

.intel\_syntax noprefix

mov ah, 0x42

**Level8**

To complete this challenge, I was asked to calculate rax = rdi % 256 and rbx = rsi % 65536 using partial register access. Consider "x % y". When y = 2^n, the answer is the lower n bits of x. Therefore, we can use that to calculate rdi % 256, since the answer is the lower 8 bits of rdi. We can also use it to calculate rsi % 65536, since the answer is the lower 16 bits of rsi. Once I implemented this, I obtained the flag.

Assembly instructions:

.intel\_syntax noprefix

mov rax, 0

mov rbx, 0

mov al, dil

mov bx, si

**Level9**

To complete this challenge, I needed to set rax to the 5th least significant byte of rdi. To do this, I shifted rdi to the left by 24 bits (or 3 bytes), which got rid of everything to the left of the 5th least significant byte. Then, I shifted rdi to the right by 56 bits (or 7 bytes), which got rid of everything to the right of the 5th least significant byte and shifted the 5th least significant byte to the location of the least significant byte. This allowed me to move the 5th least significant byte into rax, which allowed me to obtain the flag.

Assembly instructions:

.intel\_syntax noprefix

shl rdi, 24

shr rdi, 56

mov rax, rdi

**Level10**

To complete this challenge, I needed to implement the following logic without using mov or xchg:

rax = rdi AND rsi

To do this, I first used XOR to calculate rax XOR 18446744073709551615. 18446744073709551615 is the number for which all bits are set to 1, so this operation would result in 0 being stored in rax. Next, I OR’d rax and rdi together to store rdi in rax (since A OR 0 = A). Next, I AND’d rax and rsi to calculate rdi AND rsi. Thus, the challenge is solved.

Assembly instructions:

.intel\_syntax noprefix

XOR rax, 18446744073709551615

OR rax, rdi

AND rax, rsi

**Level11**

To complete this challenge, I needed to use only the and, or, and xor instructions to implement the following logic:

if x is even then

  y = 1

else

  y = 0

where x = rdi and y = rax. To do this, I first used OR to calculate rax OR 18446744073709551615. 18446744073709551615 is the number for which all bits are set to 1. Therefore, this instruction will store 18446744073709551615 into rax. Next, I used XOR to calculate rax XOR 0xfffffffffffffffe. 0xfffffffffffffffe is the number for which all bits are set to 1 except for the last bit. This operation will store 1 into rax. Next, I used AND to calculate rax AND rdi. For rax, all values except the last one are 0, so this operation will result in 1 being stored in rax for odd numbers and 0 being stored in rax for even numbers. Lastly, I used XOR to calculate rax XOR 0x1. This will flip the value in rax so that if it was 1 before, now it will be 0, and if it was 0 before, now it will be 1. Thus, this challenge is complete.

Assembly instructions:

.intel\_syntax noprefix

OR rax, 18446744073709551615

XOR rax, 0xfffffffffffffffe

AND rax, rdi

XOR rax, 0x1

**Level12**

To complete this challenge, I placed the value stored in the location pointed to by address 0x404000 into rax. Once I completed this, I obtained the flag.

Assembly instructions:

.intel\_syntax noprefix

mov rax, [0x404000]

**Level13**

To complete this challenge, I placed the value stored in rax into the location pointed to by address 0x404000. Once I completed this, I obtained the flag.

Assembly instructions:

.intel\_syntax noprefix

add QWORD PTR [0x404000], rax

**Level14**

To complete this challenge, I was asked to do the following:

1. Place the value stored at 0x404000 into rax
2. Increment the value stored at the address 0x404000 by 0x1337

Once I completed this, I obtained the flag.

Assembly instructions:

.intel\_syntax noprefix

mov rax, [0x404000]

add QWORD PTR [0x404000], 0x1337

**Level15**

To complete this challenge, I was asked to set rax to the byte at 0x404000. Once I completed this, I obtained the flag.

Assembly instructions:

.intel\_syntax noprefix

mov al, [0x404000]

**Level16**

For this challenge, I was asked to

1. Set rax to the byte at 0x404000
2. Set rbx to the word at 0x404000
3. Set rcx to the double word at 0x404000
4. Set rdx to the quad word at 0x404000

Once I completed these steps, I obtained the flag.

Assembly instructions:

.intel\_syntax noprefix

mov al, [0x404000]

mov bx, [0x404000]

mov ecx, [0x404000]

mov rdx, [0x404000]

**Level17**

For this challenge, I was asked to set [rdi] = 0xdeadbeef00001337 and [rsi] = 0xc0ffee0000. Once I completed that, I obtained the flag.

Assembly instructions:

.intel\_syntax noprefix

mov rbx, 0xdeadbeef00001337

mov [rdi], rbx

mov rbx, 0xc0ffee0000

mov [rsi], rbx

**Level18**

This challenge taught me about accessing values in memory. I was asked to add two consecutive quad words from the address stored in rdi, calculate the sum of those two quad words, and store the sum at the address location in rsi.

Assembly instructions:

.intel\_syntax noprefix

mov rax, [rdi]

add rax, [rdi+8]

mov [rsi], rax

**Level19**

This challenge taught me about the pop and push instructions associated with the stack. I was asked to take the top value of the stack, subtract rdi from it, and push it back onto the stack. Once I did this, the challenge was solved.

Assembly instructions:

.intel\_syntax noprefix

pop rax

sub rax, rdi

push rax

**Level20**

This challenge taught me about the LIFO property of the stack. I was asked to swap rdi and rsi using only push and pop.

Assembly instructions:

.intel\_syntax noprefix

push rdi

push rsi

pop rdi

pop rsi

**Level21**

This challenge taught me how to access the stack directly using the stack pointer. I calculated the average of 4 consecutive quad words on the stack using the stack pointer. Thus, the challenge was solved.

Assembly instructions:

.intel\_syntax noprefix

mov rax, 0

mov rbx, 4

add rax, [rsp]

add rax, [rsp+8]

add rax, [rsp+16]

add rax, [rsp+24]

div rbx

push rax

**Level22**

This challenge taught absolute jumps. I had to jump to the address 0x403000. Once I did that, the challenge was solved.

Assembly instructions:

.intel\_syntax noprefix

mov rax, 0x403000

jmp rax

**Level23**

This challenge taught relative jumps. I was asked to implement a relative jmp that jumps to 0x51 bytes from the current code location. After the jump, I was asked to set rax to 0x1. Thus, the challenge was solved.

Assembly instructions:

.intel\_syntax noprefix

jmp HERE

.rept 0x51

nop

.endr

HERE:

mov rax, 0x1

**Level24**

This challenge asked me to implement the following:

1. Perform a relative jmp to 0x51 bytes from the current code position
2. Once there, place the top value on the stack into register rdi
3. jmp to the absolute address 0x403000

Once I completed these steps, I was able to get the flag.

Assembly instructions:

.intel\_syntax noprefix

jmp HERE

.rept 0x51

nop

.endr

HERE:

mov rdi, [rsp]

mov rbx, 0x403000

jmp rbx

**Level25**

The goal of this challenge was to teach how to implement jmps. For this challenge, I implemented the following using assembly code:

if [x] is 0x7f454c46:

  y = [x+4] + [x+8] + [x+12]

else if [x] is 0x00005a4d:

  y = [x+4] - [x+8] - [x+12]

else:

  y = [x+4] \* [x+8] \* [x+12]

where:

x = rdi, y = rax.

In other words, if the value at rdi was 0x7f454c46, I added the values at rdi+4, rdi+8, and rdi+12 together and put the answer in rax. If the value at rdi was 0x00005a4d, I subtracted the values at rdi+8 and rdi+12 from rdi+4 and put the answer in rax. If rdi was neither 0x00005a4d or 0x7f454c46, I multiplied the values at rdi+4, rdi+8, and rdi+12 together and put the answer in rax.

Assembly instructions:

.intel\_syntax noprefix

mov rax, 0

mov rbx, 0

cmp DWORD PTR [rdi], 0x7f454c46

jne SECOND

mov rbx, rdi

add rbx, 4

add rax, [rbx]

mov rbx, rdi

add rbx, 8

add rax, [rbx]

mov rbx, rdi

add rbx, 12

add rax, [rbx]

jmp DONE

SECOND:

cmp DWORD PTR [rdi], 0x00005a4d

jne THIRD

mov rbx, rdi

add rbx, 4

add rax, [rbx]

mov rbx, rdi

add rbx, 8

sub rax, [rbx]

mov rbx, rdi

add rbx, 12

sub rax, [rbx]

jmp DONE

THIRD:

mov rbx, rdi

add rbx, 4

mov rax, [rbx]

mov rbx, rdi

add rbx, 8

mul DWORD PTR [rbx]

mov rbx, rdi

add rbx, 12

mul DWORD PTR [rbx]

DONE:

mov rbx, 0

mov ebx, eax

mov rax, rbx

**Level26**

The goals of this challenge was to teach about jump tables. I implemented the following:

if rdi is 0:

  jmp 0x40305b

else if rdi is 1:

  jmp 0x4030d2

else if rdi is 2:

  jmp 0x4031d8

else if rdi is 3:

  jmp 0x40325f

else:

  jmp 0x403375

Where the table looks like:

[0x404059] = 0x40305b (addrs will change)

[0x404061] = 0x4030d2

[0x404069] = 0x4031d8

[0x404071] = 0x40325f

[0x404079] = 0x403375

rsi is the base address of the jump table. To complete this challenge, I jumped to rdi\*8+rsi for any rdi < 4. If rdi >= 4, then I jumped to 4\*8+rsi. Thus, I completed the challenge.

Assembly instructions:

.intel\_syntax noprefix

mov rax, 8

mov rbx, 4

mov rcx, 0

cmp rdi, 0x4

jge END

mul rdi

add rax, rsi

mov rcx, [rax]

jmp rcx

END:

mul rbx

add rax, rsi

mov rcx, [rax]

jmp rcx

**Level27**

The goal of this challenge was to teach how to implement a for loop. For this challenge, I looped over n qwords and added them altogether. Then, I divided the sum by n to compute the average and stored the average in rax. Thus, the challenge was solved.

Assembly instructions:

.intel\_syntax noprefix

mov rax, 0

mov rbx, 0

BEGIN:

cmp rbx, rsi

jge END

add rax, [rdi]

add rdi, 8

inc rbx

int3

jmp BEGIN

END:

div rsi

**Level28**

The goal of this challenge was to teach how to implement a while loop. For this challenge, I looped over contiguous memory from rdi to rdi+x and incremented a counter until I encountered a null byte. This allowed me to count the number of nonzero bytes for a contiguous region of memory. Thus, I solved the challenge.

Assembly instructions:

.intel\_syntax noprefix

mov rax, 0

mov rbx, 0

cmp rdi, 0

je END

BEGIN:

cmp BYTE PTR [rdi], 0

je END

inc rbx

inc rdi

jmp BEGIN

END:

mov rax, rbx

**Level29**

For this challenge, I checked that the src\_addr was not 0 and that the value at source\_addr wasn’t null. This means that the string length is greater than 0 (each character is 1 byte, so the string length is greater than or equal to 1 byte). Now, I kept on looping through bytes until I encountered the null byte. For every byte I iterated over, if the char was not already lowercase (byte value was not already >5a), then 0x20 was added to the byte’s hex value. I added 0x20 (converted the char from upper to lowercase) using the function foo. This would convert any uppercase character to lowercase. Thus, the challenge was completed.

Assembly instructions:

.intel\_syntax noprefix

mov rcx, rdi

mov rax, 0

mov rbx, 0

mov rdi, 0

mov rdx, 0

int3

cmp rcx, 0

je END

BEGIN:

mov dl, BYTE PTR [rcx]

cmp dl, 0

je END

cmp dl, 0x5a

jg LOOP\_END

mov dil, dl

mov rsi, 0x403000

call rsi

mov BYTE PTR [rcx], al

inc rbx

LOOP\_END:

inc rcx

jmp BEGIN

END:

mov rax, rbx

ret

**Level30**

For this challenge, I first subtracted 0xff from rsp to make space for the counts of all the bytes. Then, I counted the number of times each byte found at the address locations src\_addr to (src\_addr+size) is repeated. Then, I stored the number of times each byte was repeated at the location rbp-byte. Now, for every byte in (0-0xff), I checked rbp-byte and found the largest number there. This is the byte that was most repeated at the address locations src\_addr to (src\_addr+size). I then returned this most repeated byte.

Assembly instructions:

.intel\_syntax noprefix

mov rbp, rsp

sub rsp, 0xff

mov rax, 0

mov rbx, 0

mov rcx, 0

sub rsi, 1

FORLOOP:

cmp rbx, rsi

jg FORLOOP\_END

mov rax, rdi

add rax, rbx

mov cl, [rax]

mov rax, rbp

sub rax, rcx

add BYTE PTR [rax], 1

inc rbx

jmp FORLOOP

FORLOOP\_END:

mov rdx, 0xff

mov rcx, 0

mov rbx, 0

mov rax, 0

BEGIN:

cmp rdx, 0

je END

mov rax, rbp

sub rax, rdx

cmp BYTE PTR [rax], cl

jl END\_LOOP

mov cl, BYTE PTR [rax]

mov bl, dl

END\_LOOP:

dec rdx

jmp BEGIN

END:

mov rax, rbx

mov rsp, rbp

ret