## Bit-O-Asm-1

We're asked to figure out the value of the `eax` register, given the following assembly dump:

```
<+0>:     endbr64
<+4>:     push   rbp
<+5>:     mov    rbp,rsp
<+8>:     mov    DWORD PTR [rbp-0x4],edi
<+11>:    mov    QWORD PTR [rbp-0x10],rsi
<+15>:    mov    eax,0x30
<+20>:    pop    rbp
<+21>:    ret
```

I highlighted the line that changes `eax`. The "immediate" hexadecimal value `0x30` is equal to `48` in decimal. That makes the flag `picoCTF{48}`.

## Bit-O-Asm-2

Again, we need to figure out the value of the `eax` register. Now we are given this program:

```
<+0>:     endbr64
<+4>:     push   rbp
<+5>:     mov    rbp,rsp
<+8>:     mov    DWORD PTR [rbp-0x14],edi
<+11>:    mov    QWORD PTR [rbp-0x20],rsi
<+15>:    mov    DWORD PTR [rbp-0x4],0x9fe1a
<+22>:    mov    eax,DWORD PTR [rbp-0x4]
<+25>:    pop    rbp
<+26>:    ret
```

Notice there are two important lines here, because the program uses a pointer. First the value `0x9fe1a` is moved to a memory address. Then the value at that memory address is loaded to `eax`. So `eax` will be equal to `0x9fe1a`, that's `654874` in decimal. That makes the flag `picoCTF{654874}`.

## Bit-O-Asm-3

Same challenge here, now for this program:

```
<+0>:     endbr64
<+4>:     push   rbp
<+5>:     mov    rbp,rsp
<+8>:     mov    DWORD PTR [rbp-0x14],edi
<+11>:    mov    QWORD PTR [rbp-0x20],rsi
<+15>:    mov    DWORD PTR [rbp-0xc],0x9fe1a
```

```
<+22>:    mov    DWORD PTR [rbp-0x8],0x4
<+29>:    mov    eax,DWORD PTR [rbp-0xc]
<+32>:    imul   eax,DWORD PTR [rbp-0x8]
<+36>:    add    eax,0x1f5
<+41>:    mov    DWORD PTR [rbp-0x4],eax
<+44>:    mov    eax,DWORD PTR [rbp-0x4]
<+47>:    pop    rbp
<+48>:    ret
```

The following happens in the highlighted lines:

- The register is made equal to `0x9fe1a`.
- It's multiplied by `0x4`.
- `0x1f5` is added.

The operations on lines `<+41>` and `<+44>` don't really change the register: its value is just stored somewhere and immediately fetched back.

That makes the result equal to `0x9fe1a * 0x4 + 0x1f5 = 0x27fa5d`. That's `2619997` in decimal. That makes the flag `picoCTF{2619997}`.

## Bit-O-Asm-4

Same goal, for the following program:

```
<+0>:     endbr64
<+4>:     push   rbp
<+5>:     mov    rbp,rsp
<+8>:     mov    DWORD PTR [rbp-0x14],edi
<+11>:    mov    QWORD PTR [rbp-0x20],rsi
<+15>:    mov    DWORD PTR [rbp-0x4],0x9fe1a
<+22>:    cmp    DWORD PTR [rbp-0x4],0x2710
<+29>:    jle    0x55555555514e <main+37>
<+31>:    sub    DWORD PTR [rbp-0x4],0x65
<+35>:    jmp    0x555555555152 <main+41>
<+37>:    add    DWORD PTR [rbp-0x4],0x65
<+41>:    mov    eax,DWORD PTR [rbp-0x4]
<+44>:    pop    rbp
<+45>:    ret
```

This is where it gets more complicated. As there are jumps in this program, we really have to follow the control flow step-per-step. Starting from line <+22>, the following happens:

- The contents of `[rbp-0x4],` being the value `0x9fe1a,` is compared with the value `0x2710.` It is clearly greater than the given immediate value.
- The `jle` (less than or equal) call on `<+29>` is skipped.

- The value of `[rbp-0x4]` is subtracted by `0x65`, so it becomes `0x9fdb5`.
- We jump to the instruction `<main+41>`.
- We move the value at `[rbp-0x4]` to the `eax` register.

Meaning the value of `eax` at the end of the program is `0x9fdb5`. That's `654773` in decimal, making the flag `picoCTF{654773}`.