1 Dividing integers with *DIV* and *IDIV*

- **DIV** → Divides unsigned numbers.
- **IDIV** → Divides signed numbers.

1.1 The DIV instruction

Consider the following assembly program:

```
section .data

section .text
        global _start

        _start:
                MOV eax, 11
                MOV ecx, 2

                DIV ecx

                MOV eax, 1
                INT 80h
```

The **DIV** and **IDIV** instructions work just like the **MUL** and **IMUL**. The **eax** register is automatically used by the **DIV** and **IDIV** instructions.

In the program, **eax** is the dividend and **ecx** is the divisor. The quotient gets stored in **eax** and the remainder gets stored in the register **edx** as shown in the image below:

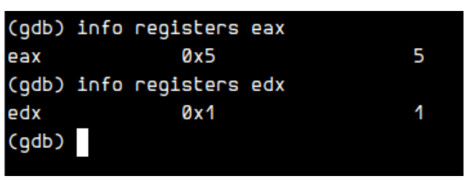


Figure 1: Values in **eax** and **edx**

1.2 The IDIV instruction

Program:

```
section .data

section .text
        MOV eax, -6
        MOV ecx, 2

        IDIV ecx

        MOV eax, 1
        INT 80h
```

Here, we are dividing a negative number(-6) by 2.

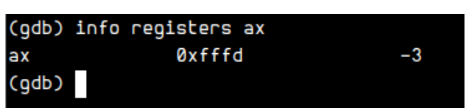
These values are stored in the registers **eax** and **ebx**:

```
(gdb) info registers eax
eax            0xfffffffa          -6
(gdb) si
0x0804900a in _start ()
(gdb) si
0x0804900c in _start ()
(gdb) info registers eax
eax            0x7ffffffd          2147483645
(gdb) info registers edx
edx            0x0                 0
(gdb)
```

Figure 2: Values of **eax** before and after dividing.

So, we can see that **edx** stores the remainder(as we know) which is 0. **eax** stores the the quotient. We can see that **eax** stores a rather larger value, it was expected to store -3.

Now, if we check the value in the sub-register **ax** then we get -3:



```
(gdb) info registers ax
ax             0xfffd              -3
(gdb)
```

Figure 3: -3 in **ax**

Now, we also check in the registers **ah** and **al**. We get the following coutput:

```
(gdb) info registers ah
ah             0xff                -1
(gdb) info registers al
al             0xfd                -3
(gdb) 
```

Figure 4: Values in **ah** and **al**

al stores -3 and **ah** stores -1.
On converting the value of **eax** to binary we get this:

01111111 11111111 11111111 11111101

ax stores the last 16-bits, out of those 16-bits, the higher 8-bits are in **ah** and the lower 8-bits are stored in **al**. We also know that 11111101 is -3.

1.2.1 Comparision

If we take -6 and -3, this how their binary representation is in 32-bit:

- -6

11111111 11111111 11111111 11111010

- -3

11111111 11111111 11111111 11111101

Notice that in the binary representation of -3 and -6 the last 0 in -6 is missing from -3. It almost looks like -6 was shifted towards the right by 1 bit(but that's not the case here).

Now, when we were debugging our program, **eax** had the following values in it. Binary representation has also been provided(in 32-bit binary):

1^{st} value in **eax**: -6

11111111 11111111 11111111 11111010

2^{nd} value in **eax**: 2147483645

01111111 11111111 11111111 11111101

Now, here the second value in **eax** seems to be a result of shifting -6(the previous value of **eax**) to the right by 1-bit.