1 Data section

We define variables inside the **data** section of the assembly program.

The format is like so:

```
section .data
        num DD 5
section .text
        global _start
        ....
        ....
        ....
```

So while declaring variables in the **data** section, provide these three things:

1. Name of the variable(in this case it's **num**).
2. Type of the variable(in this case it's **DD**).
3. Value for the variable(in this case it's 5).

Here are the various types that we can give to variables:

1. **DB** → Define byte(1 byte).
2. **DW** → Define word(2 bytes).
3. **DD** → Double word(4 bytes).
4. **DQ** → Double-precision floating-point constants(8 bytes).
5. **DT** → Extended-precision floating-point constants(10 bytes).

This is the complete program:

```
section .data
        num DD 5
section .text
        global _start

        _start:
                MOV eax, 1
                MOV ebx, num
                INT 80h
```

Here, we tried to move the value of **num** into the register **ebx**.

Notice what happens when we run this program:
Compilation

```
$ nasm -f elf -o asm2.o asm2.asm
```

Linking

```
$ ld -m elf_i386 -o asm2 asm2.o
```

Now, we will run the executable:

```
$ ./asm2
```

The program runs without any errors, but what value does **$?** store????

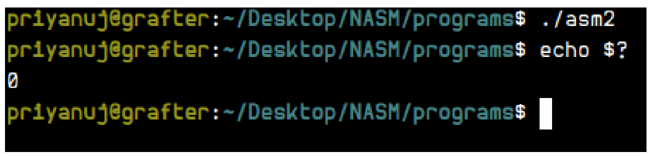

Figure 1: Unexpected value of **$?**

We can see that **$?** stores 0 when it should have stored 5 in it because we stored the value **num** into **ebx**. **DIDN'T WE??**

2 Debugging time

Let's see what GDB has to tell us. Everything is the same as section 3.

In the figure below:

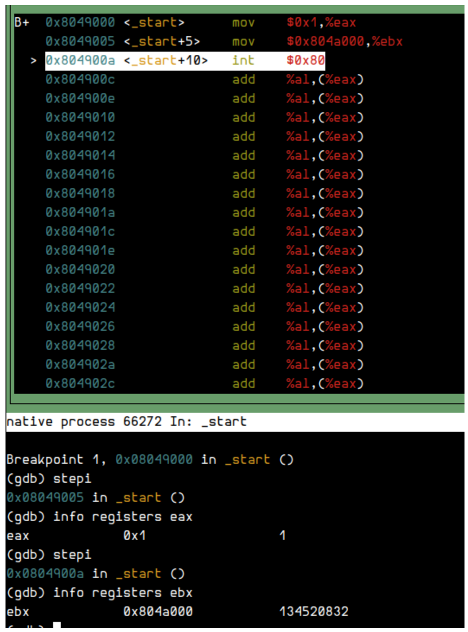

Figure 2: Surprise!

we can see a value getting stored into the register **ebx** but we are sure that it's not 5 because when we use the **info registers** command we can see the decimal form of that hexadecimal number and we can see that it's not 5.

What is it?? That is actually the address that is getting stored into register **ebx**.

Whose address?? The address of the data for the variable **num**. This address is stored by the variable **num**. So, **num** is basically storing the location on the stack where the number 5 is located. Oh no! Where is 5!!! No worries, we can view the value 5 via the following command:

```
(gdb) x/x $ebx
```

This will give us the following result:

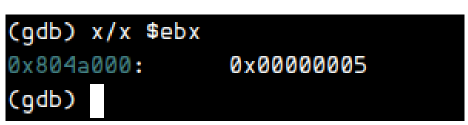

Figure 3: Value at address stored in **ebx**

Wow! It's the same hexadecimal address **0x804a000** but there is also another value in hexadecimal along with it and that is 5. So this tells us that the location(**0x804a000** is where 5 is located.

2.1 Then how do we get the value 5 out of that address??

Following changes are made:

```
MOV ebx, [num]
```

The [ ] are kind of like the dereference operator(*) in C/C++. Basically what it does is that it goes to the address stored in **num** gets the value stored in that address and moves it to the register **ebx**.

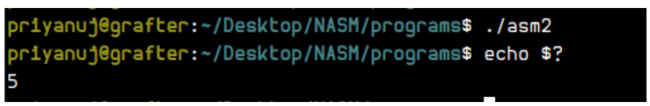

Figure 4: Value of **$?** as expected

Let's see with GDB.

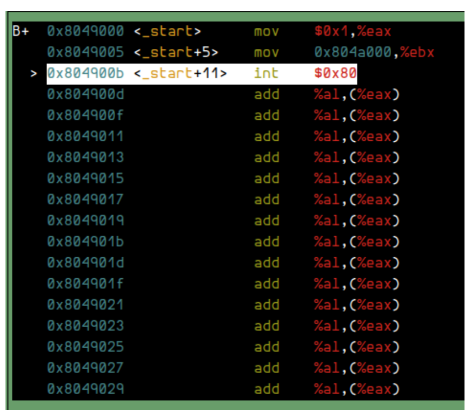

Figure 5: Notice the difference

Notice the difference between this figure and figure 2. There is no $ before **0x804a000** in this figure but there is one on the other figure.

Also, the command **info registers** command also gives the following output:

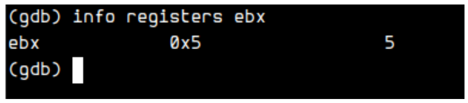

Figure 6: **ebx** stores 5