# 1 Characters, strings and lists

## 1.1 Characters

Consider the following program:

```
section .data
        char DB 'A'

section .text
        global _start
        _start:
                MOV bl, [char]
                MOV eax, 1
                INT 80h
```

The value of **$?** is as follows:

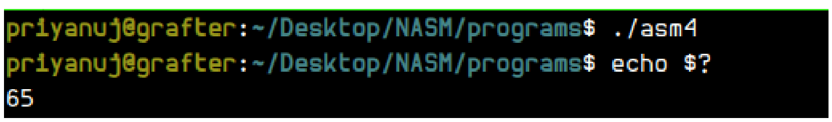


Figure 1: Value of **$?**

Since **bl** is a sub-register of the register **ebx** and the register **ebx** is used to store status code which is stored in **$?** that's why we can see 65 in **$?**.

We know that 65 is the ASCII value of the character **A**. When we work with anyting in assembly, it's stored as numeric(mostly as binary) data. So, when storing the character **A**(or any other character) it is encoded in a way so that it has a numeric value(there are many encodings like ASCII, UTF-8, etc) that represents **A**(or any other character). So, here 65 represents the character **A**.

## 1.2 Lists

Consider this program:

```
section .data
        list DB 1, 2, 3, 4

section .text
        global _start
        _start:
                MOV bl, [list]
                MOV eax, 1
                INT 80h
```

Here, **list** is a list of bytes.

Let's see how the bytes are stored in the memory.

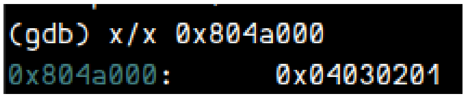


Figure 2: Bytes in **0x804a000**

We can see the bytes **0x01**, **0x02**, **0x03** and **0x04**. So, basically it's a list of bytes. Declaring a list of bytes is similar to declaring separate bytes(as seen from here). They are stored in the same manner.

Now, we will run the following commands:

```
(gdb) x/x 0x804a000
(gdb) x
```

Here is the output:

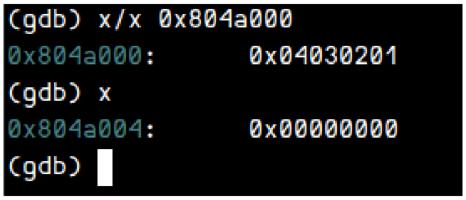


Figure 3: **x/x 0x804a000**

From the output it's clear that **x/x 0x804a000** showed the contents of the memory block starting from **0x804a000** till **0x804a003**(Visit this link).

When we use only the **x**(no address expression) command, it basically displays the memory contents from the address following the contents of the last memory address. In this case, we can see that **x/x 0x804a000** printed the contents of memory addresses **0x804a000**, **0x804a001**, **0x804a002** and **0x804a003**. Next when we run the only the **x** command then it displays the contents inside memory addresses starting from **0x804a004** all the way till **0x804a007**.

The following command:

```
(gdb) x/2x 0x804a000
```

will display the contents of 2 units of memory (4 bytes/unit). This means the contents of the first memory unit(from **0x804a000** to **0x804a003**) will be displayed and then the contents of second memory unit(from **0x804a004** to **0x804a007**) will be displayed.

### 1.3 Strings

The following x86 assembly program uses strings:

```
section .data
        string1 DB "ABA", 0
        string2 DB "CDE", 0

section .text
        global _start
        _start:
                MOV bl, [string1]
                MOV eax, 1
                INT 80h
```

The 0 at the end of each string is actually the ***null terminator***.

We run the following command:

```
(gdb) x/x 0x804a000
```

The following output is generated:

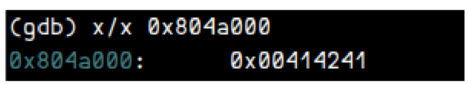


Figure 4: Storing strings

We can see the hex value of **A**(**0x41**), of **B**(**0x42**) and of **A**(**0x41**).

Now, again, we will use the **x/2x** command on **0x804a000**. We get the following output:

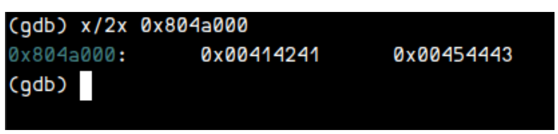


Figure 5: Output of **x/2x 0x804a000**

We have two strings and each has a null terminator.

From memory blocks starting from **0x804a000** to **0x804003** we have stored the string **"ABA\0"**. The **0x00** is actually the null terminator(not an empty or uninitialized byte). The next hexadecimal value **0x00454443** is actually the string **"CDE\0"**(**0x00** is for the null terminator("\0")).

These two strings are stored in two different units. 1 unit starts from **0x804a000** to **0x804a003**(4 bytes) and the other unit starts from **0x804a004** to **0x804a007**(4 bytes).