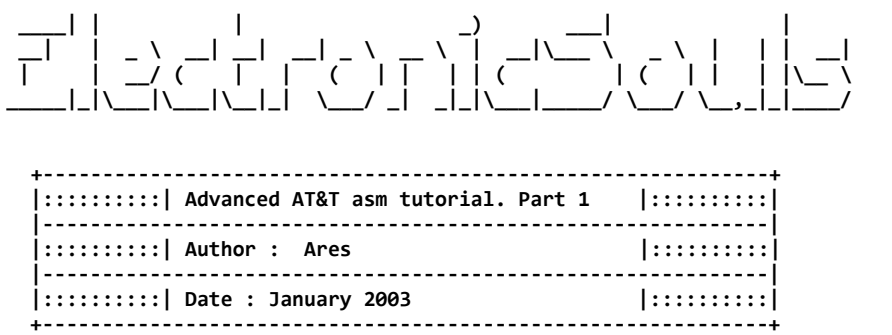

1 - [Introduction]:

A few words before, about why i wrote this paper...

When i start learning assembler in linux,
i faced with a lot of problems, because there are too few documents describing AT&T syntax,
most of them contain just information about difference between AT&T and Intel syntax + "Hello world" example.

Ofcourse if you pretty know Intel syntax then it's much more easy to write programms in asm for linux,
but i don't know it such as many people trying to learn linux assembler.

Programming for linux (AT&T) more easy than programming for DOS,
because of particularity of syntax,it looks more familiar for user.
This is one of the reasons i start learn asm for linux,
but main reason that i prefer Unix for developments.

I won't talking about CPU's registers, memory etc...
that's why you need to have basic knowledges in asm, if you don't,
try google.com "at&t asm basic tutorial" or some shellcoding tutorials...

All text below will be questions and answers,
basically it will description of writing common functions (strcmp,strcat etc...) without libc.

2 - [Overview]:

[The questions, which will describes]:

{1}. How to compare 2 string variables ?
{2}. How to level string variables ?
{3}. How to create a counter ?
{4}. How to calculate the length of string dynamically ?
{5}. How to work with arrays ?
{6}. How to convert number(integer) to it's ASCII equivalent(string) ?
{7}. How to convert integer variable to network byte order ?

[So lets begin...]

[1]. How to compare 2 string variables ?:

You can face with need this function in programs where you checking input information from user,
authorization for example.
There is a special instruction to compare string variables - cmpsb,
it takes variables from %esi, %edi and %ecx:

```
---------------------------------------------------------------------------
/* strcmp.s */
.globl _start
_start:

movl $str1,%esi         # we put first string to %esi
movl $str2,%edi         # second string to %edi
movl $4,%ecx            # number of bytes to compare
cld                     # clear flag D, if we won't clear it, the compare will occur in inverse order
rep  cmpsb              # compareing
jz   yes                # if our strings are equal then we jmp to yes, if not then exit
```

```
exit:
movl $1,%eax            # exit
int  $0x80

yes:
movl $4,%eax            # write
movl $1,%ebx            # stdout
movl $msg,%ecx          # out message
movl $msglen,%edx       # length of message
int  $0x80

jmp exit

.data
str1:    .string "asdf"
str2:    .string "asdf"
msg:     .string "The strings are equal! Quiting...\n"
msglen = . -msg         # calculate the length of msg
/* End */
```

```
 Compiling:
as strstr.s -o strstr.o
ld strstr.o
You can use gcc: gcc strstr.s , but first replace "_start" to "main".
```

[2]. How to level string variables ?:

```
 We will use instruction movsb to do it,
we should put the string we want add to %esi and a buffer where we will get the result to %edi.
Now look at example:
```

```
/* strcat.s */
.globl main
main:

movl $str1,%esi         # we put string "AT&T" to %esi
movl $buf,%edi          # using for contain result after movsb executed
movl $6,%ecx            # size of str1
cld                     # clear direction flag
rep  movsb              # first time we add str1 to $buf.
movl $str2,%esi         # here we prepare str2 before adding to $buf
movl $9,%ecx            # size of str2
cld
rep  movsb              # after it we have str1+str2 in $buf

movl $4,%eax            # write
movl $1,%ebx            # stdout
movl $buf,%ecx          # "AT&T is cool\n"
movl $15,%edx           # size of $buf
int  $0x80

movl $1,%eax            # exit
int  $0x80

.data
str1: .string "AT&T "
str2: .string "is cool\n"

buf: .byte 15           # define static buffer
/* End */
```

---

[3]. How to create a counter ?:

```
 What program without a counter ? Heh, yeah it's very often using function.
To show that our counter realy work we'll use write syscall,
that's why registers %eax,%ebx,%ecx,%edx are busy, but %esi and %edi are remained, let's do it:
```

```
/* counter.s */
.globl main
main:

movl $5,%esi            # repeat 5 times, use %esi or %edi doesn't matter

loop:

# write (stdout,str,5);
```

```
movl $4,%eax
movl $1,%ebx
movl $str,%ecx
movl $5,%edx
int  $0x80

decl %esi               # decrease %esi by 1
tesl %esi,%esi          # if %esi equal 0
jz   exit               # then jmp exit

jmp  loop               # jmp to start of counter

exit:
movl $1,%eax
int  $0x80
/* End */
```

---

[4]. How to calculate the length of string dynamically?:

To realize this function we need resort to a small trick.
We will use lodsb to check string byte by byte, simultaneously we'll increment our counter to calculate the length.
First we should move the effective address of string to %esi, register %edi is our counter.

```
/* strlen.s */
.globl main
main:

xorl %edi,%edi
leal str,%esi                   # effective address of string to %esi

abc:
lodsb                           # read byte of string, then automaticaly increment position in string (next byte)
cmpb $0,%al                     # if readed byte equal 0, so end of string reached
jz   continue
incl %edi                       # counter
jmp  abc                        # repeat procedure

continue:
decl %edi

...                             # now %edi contain length

.data
str: .string "CheckMylength"    # if there is '\n', then it increment length by 1
/* End */
```

Take a small break and i'll give you a few advices. Why not replace our variables to more familiar view ?
int $sysint - it looks better than int $0x80, so we can define it like sysint = 0x80, and everthing you want:

```
int = 1
char = 97       # character "a" (dec)
char = 0x61     # character "a" (hex)
etc...
```

But don't forget ! It is static variables, sets in time of compilation.
So you can't increment or decrease this variables in code.

If you want use dynamic buffer, you can define it like:

```
.comm variable,_bytes_,_bits_
_bytes_ - length of variable
_bits_ - type of variable: 8 - char, 16 - short, 32 - long etc...
```

or use memory, for example: -4(%ebp),-8(ebp)...

[Continue...]

[5]. How to work with arrays ?:

```
/* Small C example */
int main()
{
char a[10];
int b;
```

```
for (b=0;b!=10;b++)
 {
  a[b]='a';
  printf("%c",a[b]);
 }
}
/* End */
```

```
This program move character 'a' to every elements of our array, output of program:aaaaaaaaaa
Think it's good example to show how arrays work in asm.
```

```
/* array.s */
.text                           # text segment
count  = 10                     # number for counter
a      = -12                    # offset in %ebp, place for array - like char a[10];
b      = -16                    # int b;
char   = 97                     # character 'a'
sysint = 0x80

.globl main
main:

movl $0,b(%ebp)                 # for(b=0;

loop:
cmpl $count,b(%ebp)             # b != 10;
jne  lp                         # jump if not equal
jmp  exit                       # jump if counter reached

lp:
leal a(%ebp),%edi               # a[
movl b(%ebp),%esi               # b]
movb $char,(%esi,%edi)          # ='a';
incl b(%ebp)                    # b++);

# write(stdout,a[b],1);
movl $4,%eax
movl $1,%ebx
leal (%esi,%edi),%ecx
movl $1,%edx
int  $sysint

jmp loop

exit:
movl $1,%eax
int  $sysint
/* End */
```

As you saw the array is in %edi and elements in %esi, whole array is (%esi,%edi).

[6]. How to convert number(integer) to it's ASCII equivalent(string)?:

Why we need it ? Because syscall write work only with strings, so if we'll write(stdout,_numeric_,length),
it put a symbol from ascii table with number _numeric_. The numerals starts in ASCII table from 48 (decimal 0),
then we just enlarge our _numeric_ by 48 and get it ascii equivalent.
Well, before do it in asm, we will take a look at procedure in C to understand what to do...:

```
/* num2ascii.c */
int  main()
 {
char a[10];
int x = 1234,b=0;

do                              # separating every digits from whole number
        {
          a[b]=48 + x % 10;     # and put it in array, the first element contain "4", second "3" etc... - inverse order
          b++;
        } while (x /= 10);
do                              # without it we'll get "4321"
        {
          b--;
          printf("%d\n",a[b]);
        } while(b!=0);
}
/* End */
```

```
------------------------------------------------------------------------
 This is approximately function, if copy it to asm step by step, we get large code ~1kb.
------------------------------------------------------------------------
/* num2ascii.s */
int = 1234

.globl main
main:

xorl %esi,%esi
movl $int,%eax

loop:
movl $0,%edx
movl $10,%ebx
divl %ebx               # dividing, result in %eax (123) and remainder in %edx (4)
addb $48,%dl            # +48
pushl %edx              # on stack
incl %esi               # counter of digits
cmpb $0,%al
jz   next

jmp loop

next:                   #
popl (%ecx)             # get from stack
testl %esi,%esi
jz   exit
decl %esi
movl $4,%eax
movl $1,%ebx
movl $2,%edx
int  $0x80

jmp  next
exit:
movl $1,%eax
int $0x80
/* End */
------------------------------------------------------------------------
```

[7]. How to convert integer variable to network byte order ?:

```
 We need it for example in creating socket, exactly in htons(port).
------------------------------------------------------------------------
/* htons.s */
int = 10
.globl main
main:

movl $int,%edi
movl %edi,%eax
rol $8,%ax              # rotate

# now %eax contain 10 in network byte order type
...

/* End */
------------------------------------------------------------------------
```

3 - [Last words...]:

Hm, think it's enough for the first part... With this new knowledges you can write more difficult shellcodes, payload for any infectors or viruses, also this is good way to optimize some functions in big C code with inline assembler. Next time i gonna describe more interesting things, like working with fork and others...

All examples were writed for linux, but it's easy to port it for example in *BSD.
And for the end i coded port scanner for educational purpose, you'll find a few other tricks there like converting IP to network byte order, working with command line etc..
Well this is end...enjoy.