Meshari Alhammadi

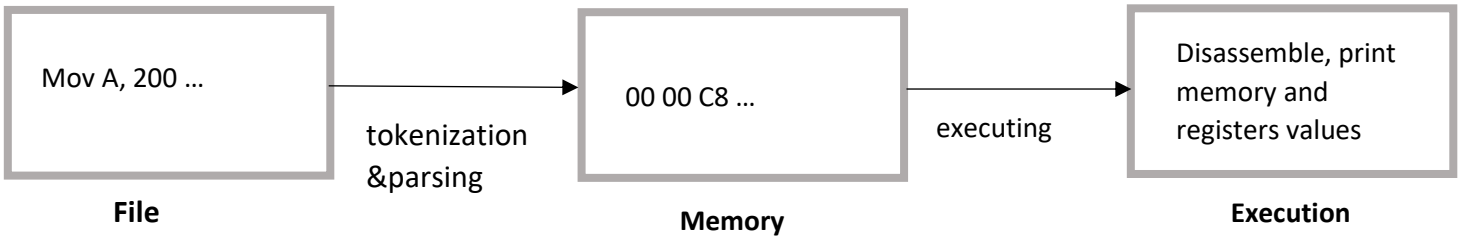
mesharialhmadi@outlook.sa

A.1

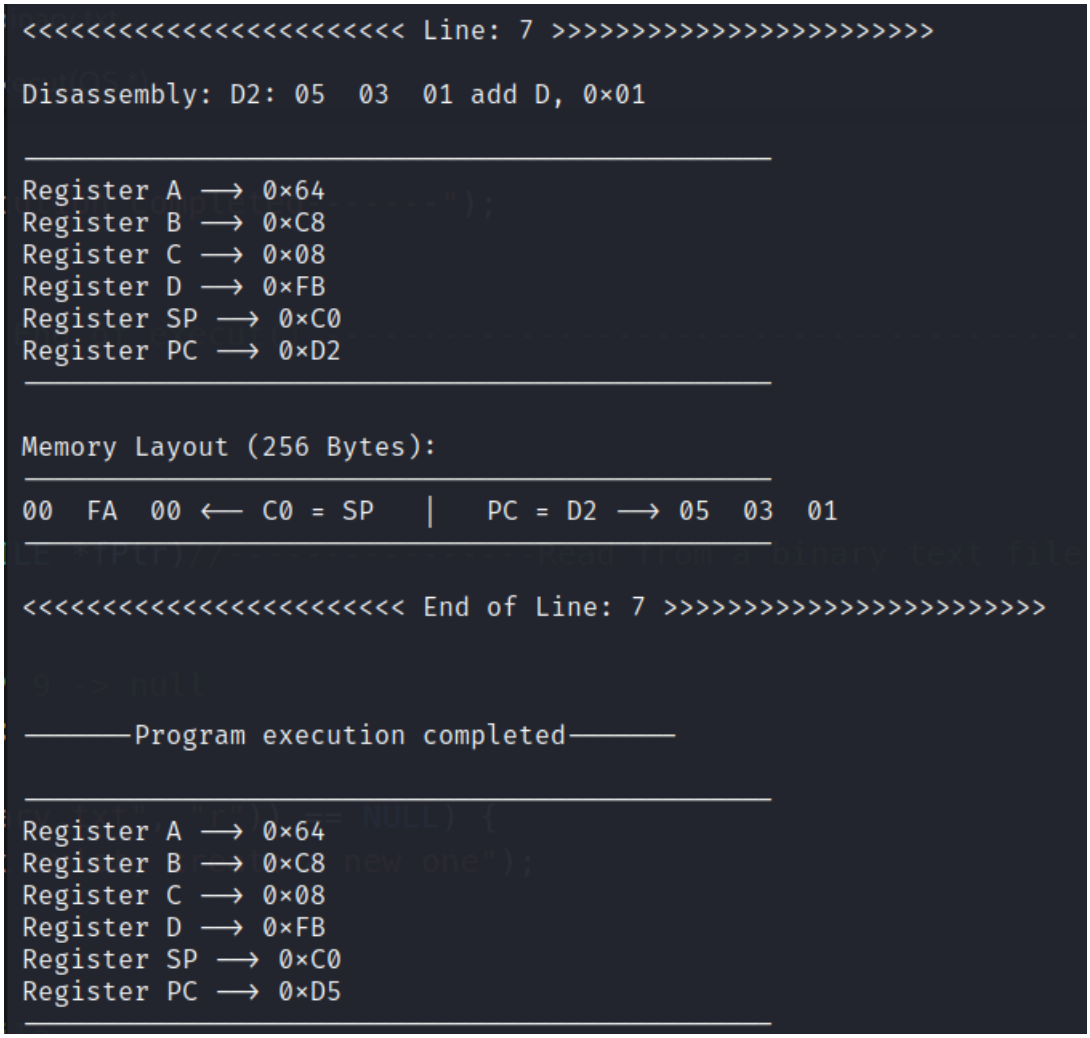
Here I challenged myself and I did two methods

1. Read the code as assembly code

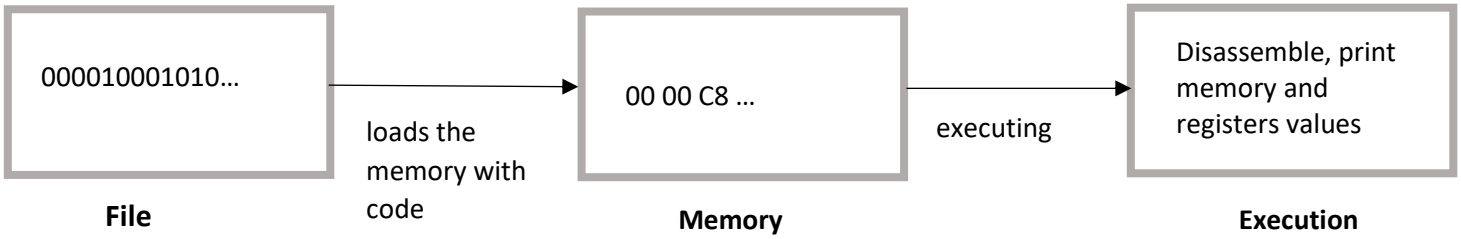
Tokenization, parsing and loads the memory with code then execute



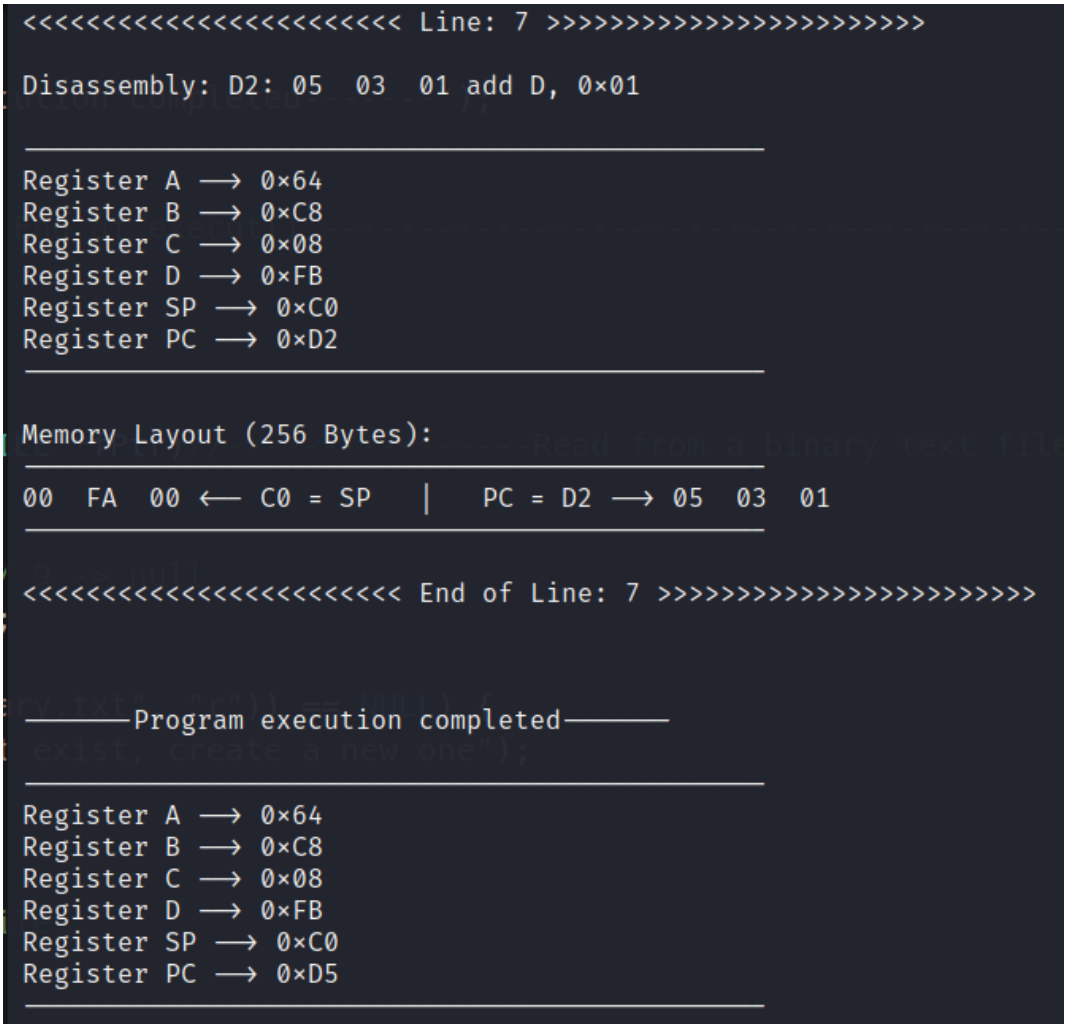

```
┌──(kali㉿kali)-[~/Desktop/c]
└─$ ./a.out
Enter 1 to read the code as binary from binary.txt
Enter 2 to read the code as assembly code from code.txt
——→2
Code raw bytes
| 00006403C80004010003FA000102BF040300050301 |

<<<<<<<<<<<<<<<<<<<<<<< Line: 1 >>>>>>>>>>>>>>>>>>>>>>>

Disassembly: C0: 00  00  64 mov A, 0×64

_____________________________________________

Register A ⟶ 0×64
Register B ⟶ 0×00
Register C ⟶ 0×00
Register D ⟶ 0×00
Register SP ⟶ 0×C0
Register PC ⟶ 0×C0
_____________________________________________

Memory Layout (256 Bytes):
_____________________________________________
00  00  00 ⟵ C0 = SP   |   PC = C0 ⟶ 00  00  64
_____________________________________________

<<<<<<<<<<<<<<<<<<<<<<< End of Line: 1 >>>>>>>>>>>>>>>>>>>>>>>
```


<<<<<<<<<<<<<<<<<<<<<<<< Line: 7 >>>>>>>>>>>>>>>>>>>>>>>>
Disassembly: D2: 05 03 01 add D, 0×01
Register A ⟶ 0×64
Register B ⟶ 0×C8
Register C ⟶ 0×08
Register D ⟶ 0×FB
Register SP ⟶ 0×C0
Register PC ⟶ 0×D2
Memory Layout (256 Bytes):
00 FA 00 ⟵ C0 = SP | PC = D2 ⟶ 05 03 01
<<<<<<<<<<<<<<<<<<<<<<<< End of Line: 7 >>>>>>>>>>>>>>>>>>>>>>>>
———Program execution completed———
Register A ⟶ 0×64
Register B ⟶ 0×C8
Register C ⟶ 0×08
Register D ⟶ 0×FB
Register SP ⟶ 0×C0
Register PC ⟶ 0×D5

2. Read the code as binary




```
┌──(kali㉿kali)-[~/Desktop/c]
└─$ gcc q1.c

┌──(kali㉿kali)-[~/Desktop/c]
└─$ ./a.out
Enter 1 to read the code as binary from binary.txt
Enter 2 to read the code as assembly code from code.txt
——→1
Code raw bytes
| 00006403C80004010003FA000102BF040300050301 |

<<<<<<<<<<<<<<<<<<<<<<<< Line: 1 >>>>>>>>>>>>>>>>>>>>>>>>

Disassembly: C0: 00  00  64 mov A, 0×64

_____________________________________________
Register A ⟶ 0×64
Register B ⟶ 0×00
Register C ⟶ 0×00
Register D ⟶ 0×00
Register SP ⟶ 0×C0
Register PC ⟶ 0×C0
_____________________________________________

Memory Layout (256 Bytes):
_____________________________________________
00  00  00 ⟵ C0 = SP   |  PC = C0 ⟶ 00  00  64
_____________________________________________

<<<<<<<<<<<<<<<<<<<<<<<< End of Line: 1 >>>>>>>>>>>>>>>>>>>>>>>>
```



I uploaded the files on my GitHub account: source

A.2

What I did is this

Push 0x00 as null, then reverse the ascii string and push it

→

Print the string to the console with prn [m]

I did that because it will print from last address and stop at null

[m] will be the stack pointer [B6]

Source code:

push 0x00

push 0x22

push 0x69

push 0x72

push 0x61

push 0x68

push 0x73

push 0x65

push 0x4d

push 0x22

prn [0xB6]

```
<<<<<<<<<<<<<<<<<<<<<<<<< Line: 10 >>>>>>>>>>>>>>>>>>>>>>>>>

Disassembly: DB: 03  22  00 push 0×22

_____________________________________________________

Register A ⟶ 0×00
Register B ⟶ 0×00
Register C ⟶ 0×00
Register D ⟶ 0×00
Register SP ⟶ 0×B6
Register PC ⟶ 0×DB
_____________________________________________________

Memory Layout (256 Bytes):
_____________________________________________________
00  00  22 ⟵ B6 = SP   |   PC = DB ⟶ 03  22  00
_____________________________________________________

<<<<<<<<<<<<<<<<<<<<<<<<< End of Line: 10 >>>>>>>>>>>>>>>>>>>>>>>>>


String ————⟶"Meshari"

————Program execution completed————
```

Code raw bytes

| 03000003220003690003720003610003680003730003650034D0003220007B600 |

also I did it as x86

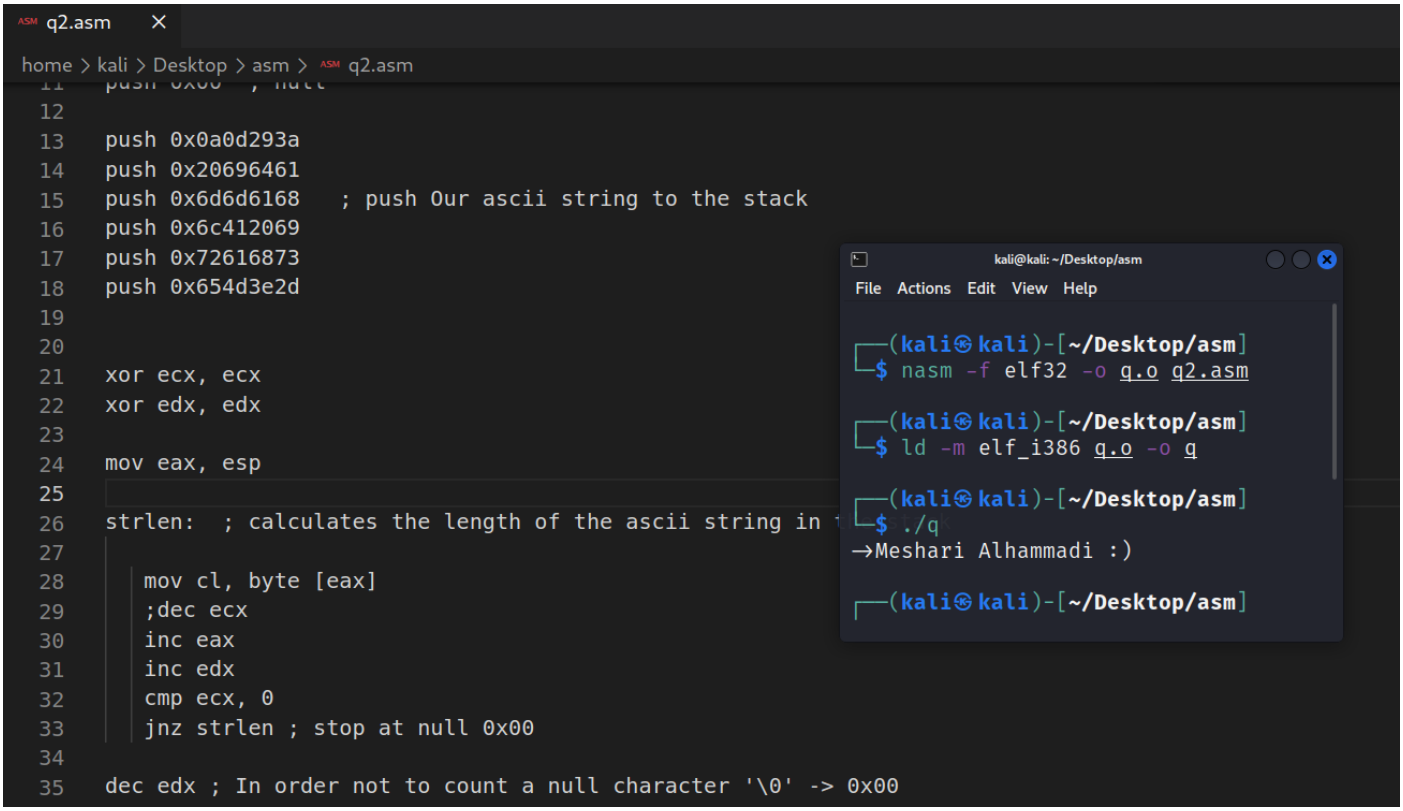
Push 0x00 as null, then reverse the ascii string and push it → Calculates the length of the ascii string in the stack → Print the string to the console

I did that because it will print from last address and stop at null

Mov byte by byte to a register in a loop and compare it with a zero and increment edx

At the end of the loop edx is the length of the ascii string

Call print and ecx will be the stack pointer



I uploaded the code on my GitHub account: source

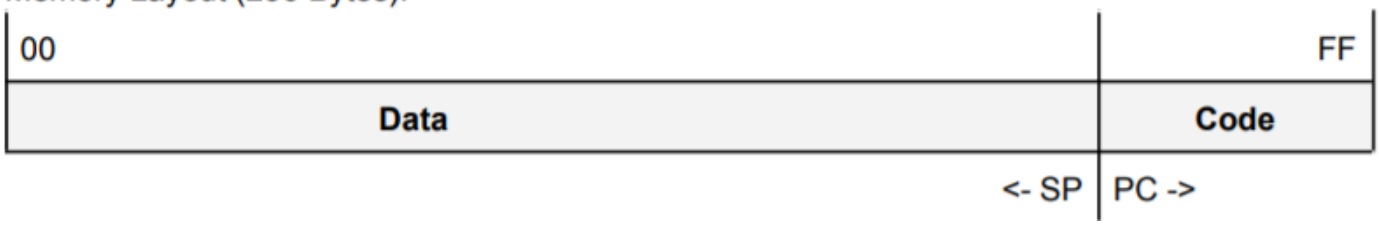
A.3

```
push 200
push A
add SP, 20
push 259
push 243
dec SP, 20
push 80
```

At “push A” will push A ID 000 → 0x00 as immediate byte

At “add SP, 20” the stack pointer will point at instruction supposed to be done later on (will point at “push 80”) !!!

Memory Layout (256 Bytes):



At “push 259” will increment stack pointer (will point at “dec SP,20”)

and rewrite the value (instruction)

with 03 not 259 because it is 8-bit system (0000 0001 0000 0011)

At “push 243” will increment stack pointer (will point at “dec SP,20”)

and rewrite the value (instruction)

Now “dec SP,20” (06 04 14) turn into “dec 243,3” (06 F3 03)

unpredictable!!

At “push 80” will increment stack pointer (will point at “dec SP,20”)

and rewrite the value (instruction) 06 → 50 it already executed

A.4

```
00006403C80004010003FA000102BF040300050301
```

C0: 00 00 64 mov A, 0x64

C3: 03 C8 00 push 0xC8

C6: 04 01 00 pop B

C9: 03 FA 00 push 0xFA

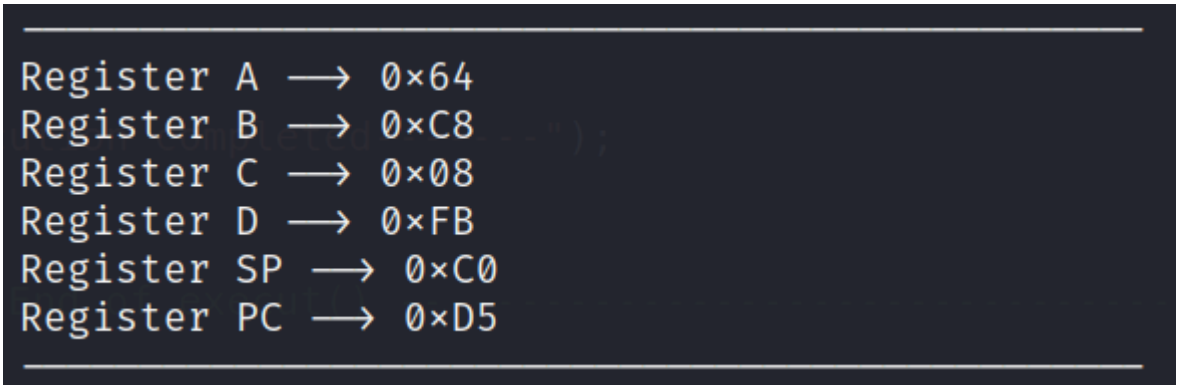
CC: 01 02 BF mov C, [0xBF]

CF: 04 03 00 pop D

D2: 05 03 01 add D, 0x01

```
└─$ ./a.out
Enter 1 to read the code as binary from binary.txt
Enter 2 to read the code as assembly code from code.txt
———→1
Code raw bytes
| 00006403C80004010003FA000102BF040300050301 |


Disassembly: C0: 00  00  64 mov A, 0×64

Disassembly: C3: 03  C8  00 push 0×C8

Disassembly: C6: 04  01  00 pop B

Disassembly: C9: 03  FA  00 push 0×FA

Disassembly: CC: 01  02  BF mov C, [0×BF]

Disassembly: CF: 04  03  00 pop D

Disassembly: D2: 05  03  01 add D, 0×01

———————Program execution completed———————
```

A.5


Register A ⟶ 0×64
Register B ⟶ 0×C8
Register C ⟶ 0×08
Register D ⟶ 0×FB
Register SP ⟶ 0×C0
Register PC ⟶ 0×D5