# NASM Quick Tutorial

695.744
T. McGuire
Johns Hopkins University

## Introduction

It may be useful to write your own assembly programs, assemble them, and then inspect the output as an aid when learning the (dis)assembly process. The Netwide Assembler (NASM) is recommended as it's cross-platform and freely available from the NASM site: https://www.nasm.us/pub/nasm/stable/.

## Tutorial

1. Use your favorite text editor to start writing assembly. The top of your file should include the directive **[BITS 32]** to let **nasm** know it should compile using 32-bit code.

2. Follow the BITS directive with a directive to indicate that the next section should be considered "text", which indicates executable code in this context. Use the **section .text** directive to indicate this. This is not entirely necessary, but it shows you how to move data into different sections.

3. Begin writing valid assembly! See below for an example.

```
[BITS 32]        ; BITS directive

section .text    ; section directive to indicate executable 'code'

    push    ebp
    mov     ebp, esp
    lea     esi, [ string ]

string:
db  'hello world',0xd,0x0
end_string:
```

**Figure 1:** View of the assembly that was hand-written

4. (Assuming Linux but Windows is similar). To assemble your newly created assembly to machine code, run the following:
   **nasm ex2.S -o ex2**

5. Assuming your syntax was correct, a new file named **ex2** should be created. It is also created in a format that your disassembler will be able to handle.

6. You can then use the **ndisasm** utility to disassemble your code to see what it looks like. You can use the **-u** or **-b 32** to force 32-bit disassembly. See below for an example. Notice that the disassembly from address 0x00000009 and beyond is jibberish. This is because ndisasm is a linear sweep disassembler and it is trying to disassemble the string "hello world".

```
user@reva:~/$ ndisasm -u ex2
00000000  55                push ebp
00000001  89E5              mov ebp,esp
00000003  8D3509000000      lea esi,[dword 0x9]
00000009  68656C6C6F        push dword 0x6f6c6c65
0000000E  20776F            and [edi+0x6f],dh
00000011  726C              jc 0x7f
00000013  64                fs
00000014  0D                db 0x0d
00000015  00                db 0x00
```

**Figure 2:** Output of **ex2** disassembled by **ndisasm**

7. With **binutils** installed (e.g. **sudo apt-get install binutils**), you can use **objdump** to have it disassemble the code for you so you can compare your disassembly with a known good disassembler engine. Note by default **objdump** uses AT&T syntax, whereas we will be using Intel syntax. To force objdump to use Intel syntax, use the **-M intel** option. Use the **-D** option to indicate you want to disassemble all. Since there is no header information, we must tell **objdump** this is a binary file. To do so, use the **-b binary** option. Again, without the header, we need to inform **objdump** of the processor. In this case, we want the Intel x86 processor so we supply **-mi386**.

```
user@reva:~/$ objdump -M intel -D -b binary -mi386 ex2

ex2:     file format binary


Disassembly of section .data:

00000000 <.data>:
   0:   55                      push   ebp
   1:   89 e5                   mov    ebp,esp
   3:   8d 35 09 00 00 00       lea    esi,ds:0x9
   9:   68 65 6c 6c 6f          push   0x6f6c6c65
   e:   20 77 6f                and    BYTE PTR [edi+0x6f],dh
  11:   72 6c                   jb     0x7f
  13:   64                      fs
  14:   0d                      .byte 0xd
```

**Figure 3:** Output of **ex2** disassembled by **objdump**