Nasm Installation and Compilation Instructions

Linux

1. First we want to install the actual `nasm` assembler. Run `apt-get install nasm` or `yum install nasm`.

    a. On systems that use rpm packages (i.e. uses `yum …`), you may need to download this file https://www.nasm.us/nasm.repo and copy it into your `/etc/yum/yum.repos.d` directory

2. (optional) Download nasmx from this link: https://sourceforge.net/projects/nasmx/ (you can also find this link by googling nasmx). This downloads some of the common higher level macro definitions that make programming assembly easier. The download file should be a gzip tar archive. We assume you downloaded the version 1.4 of nasmx. If they change suddenly, just change the commands wherever you see 1.4.

    a. In the directory where you downloaded the nasmx tar archive, type the command into terminal to extract it:

    ```
    tar -xf nasmx-1.4.tar.gz -C ~
    ```

    This will extract a nasmx directory directly into your home directory. If you want to extract it somewhere else, then change the ~ part of the command. The following steps assume you extracted the contents into your home directory.

    b. Now you have to edit your `~/.bashrc` file (assuming you are using bash). In your favorite editor, open up this file and scroll to the very bottom of the file. Then add the following line to it:

    ```
    cd ~/nasmx-1.4; . ./setpaths.sh > /dev/null
    ```

    This imports the nasmx to the include paths for nasm. YOU MUST OPEN A NEW TERMINAL BEFORE USING NASMX!

3. To assemble a file, type the command below. We are assuming you are assembling a file named hello.asm

    ```
    nasm -f elf32 hello.asm
    ```
    (for 32-bit)

    ```
    nasm -f elf64 hello.asm
    ```
    (for 64-bit)

This will generate a file named `hello.o` (where the "hello" part is just the name of the assembly source file minus the .asm extension)

4. To link the file after assembling it, type in the command. This invokes the gcc-linker and will automatically link you with the standard C library (i.e. printf, puts, strcmp, strcpy...)

   ```
   gcc hello.o
   ```

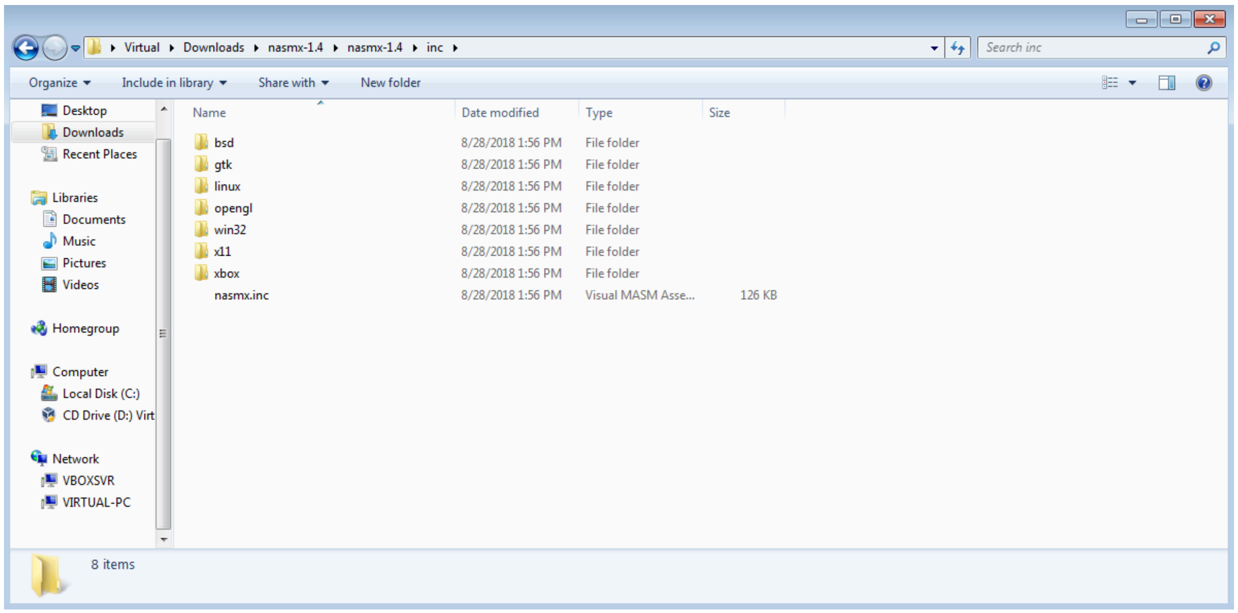
   If you do not need standard C library (rare that this happens), you can directly link with this command:

   ```
   ld hello.o
   ```

Windows

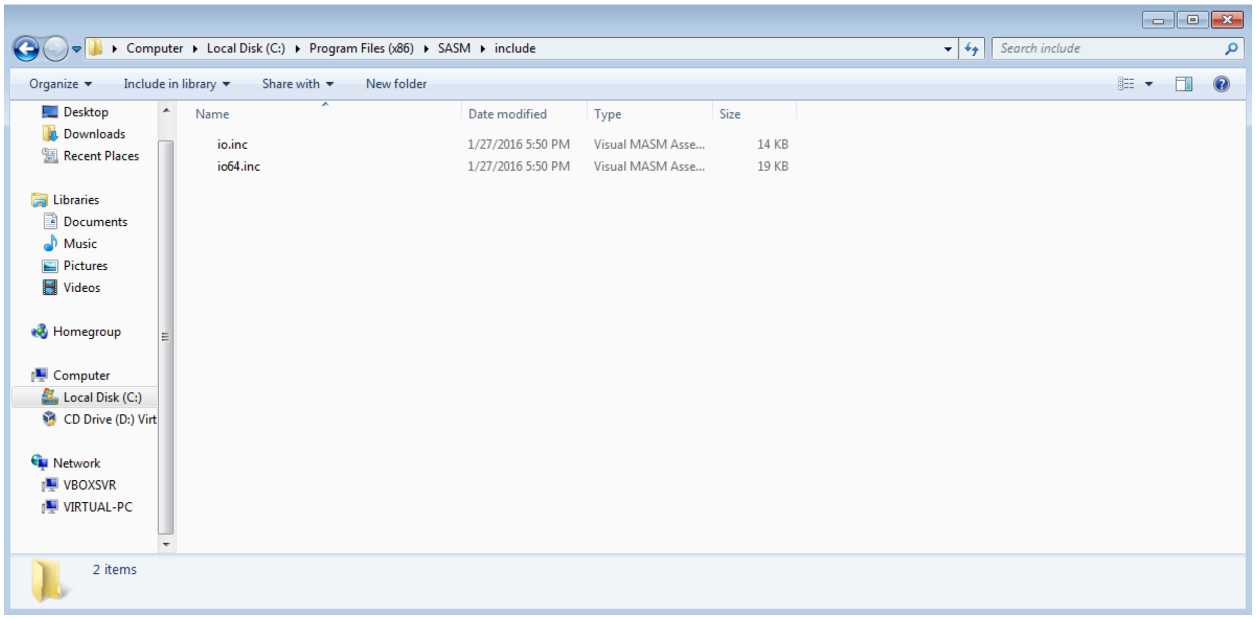
1. On windows, we are just going to download an IDE called SASM. This also works on Linux, but people in Linux should use terminal. First go to this link: https://dman95.github.io/SASM/english.html and click "Download for Windows". This will download a EXE setup file that will install SASM. Open up this installer and follow through the installer instructions.
2. (optional) Download nasmx from this link: https://sourceforge.net/projects/nasmx/ (you can also find this link by googling nasmx). This downloads some of the common higher level macro definitions that make programming assembly easier. When downloaded, you should get a .zip file. Extract this file.

   a. Within the folder, navigate to the folder `nasmx-1.4\inc`. It should look like the following image:



Name	Date modified	Type	Size
bsd	8/28/2018 1:56 PM	File folder	
gtk	8/28/2018 1:56 PM	File folder	
linux	8/28/2018 1:56 PM	File folder	
opengl	8/28/2018 1:56 PM	File folder	
win32	8/28/2018 1:56 PM	File folder	
x11	8/28/2018 1:56 PM	File folder	
xbox	8/28/2018 1:56 PM	File folder	
nasmx.inc	8/28/2018 1:56 PM	Visual MASM Asse...	126 KB

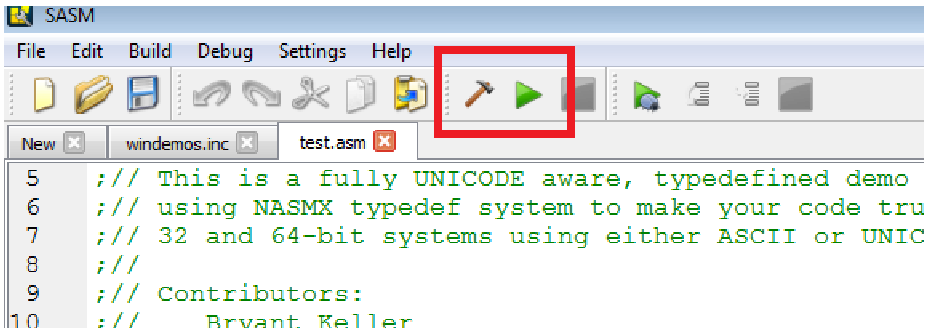
b. Then copy all the files. Navigate to the directory `C:\Program Files (x86)\SASM\include`. It should have two files in it, and it looks like this:



Name	Date modified	Type	Size
io.inc	1/27/2016 5:50 PM	Visual MASM Asse...	14 KB
io64.inc	1/27/2016 5:50 PM	Visual MASM Asse...	19 KB

Paste the files we copied from nasmx into this directory. If prompted with a message requesting administrator access, click "Continue".

3. Open up SASM. When you finish writing in assembly, to assemble and link the program, press the "Hammer" button to build it. Then the button next to it will build and run the program. To save an .exe file, go to `File->Save .exe`.



```
;// This is a fully UNICODE aware, typedefined demo
;// using NASMX typedef system to make your code tru
;// 32 and 64-bit systems using either ASCII or UNIC
;//
;// Contributors:
;//     Bryant Keller
```

4. To switch from building 32-bit and 64-bit, navigate to Settings->Settings->Build->Mode. This area allows you to select between x86 and x64 (32-bit and 64-bit respectively).

MacOS

1. Make sure that you have homebrew installed. (This is a tool for easily downloading and installing other important packages) To install homebrew, go to https://brew.sh and copy-paste the very long command into your terminal.
2. Run the command `brew install nasm` in the terminal to install the nasm assembler.

3. (optional) Download nasmx from this link: https://sourceforge.net/projects/nasmx/ (you can also find this link by googling nasmx). This downloads some of the common higher level macro definitions that make programming assembly easier. The download file should be a gzip tar archive. We assume you downloaded the version 1.4 of nasmx. If they change suddenly, just change the commands wherever you see 1.4.

   a. In the directory where you downloaded the nasmx tar archive, type the command into terminal to extract it:

      ```
      tar -xf nasmx-1.4.tar.gz -C ~
      ```

      This will extract a nasmx directory directly into your home directory. If you want to extract it somewhere else, then change the ~ part of the command. The following steps assume you extracted the contents into your home directory.

   b. Now you have to edit your `~/.bashrc` file (assuming you are using bash). In your favorite editor, open up this file and scroll to the very bottom of the file. Then add the following line to it:

      ```
      cd ~/nasmx-1.4; . ./setpaths.sh > /dev/null
      ```

      This imports the nasmx to the include paths for nasm. YOU MUST OPEN A NEW TERMINAL BEFORE USING NASMX!

4. To assemble a file, type the command below. We are assuming you are assembling a file named hello.asm

   ```
   nasm -f macho32 hello.asm
   ```
   (for 32-bit)

   ```
   nasm -f macho64 hello.asm
   ```
   (for 64-bit)

   This will generate a file named `hello.o` (where the “hello” part is just the name of the assembly source file minus the .asm g++ hextension)

5. To link the file after assembling it, type in the command. This invokes the linker and will link you with the standard C library (i.e. printf, puts, strcmp, strcpy...)

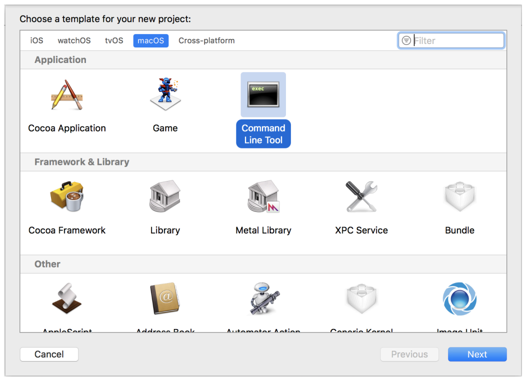
   ```
   gcc -Wl,-no_pie -o hello hello.o
   ```

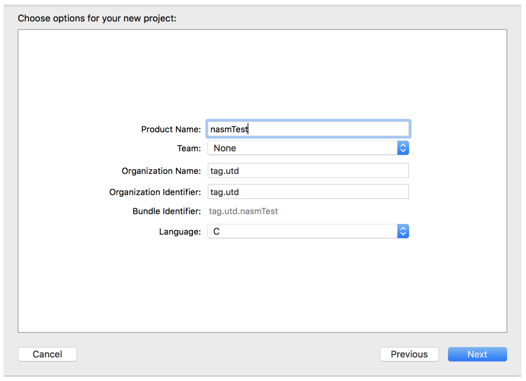
   If you do not need standard C library (rare that this happens), you can directly link with this command:

   ```
   ld hello.o -o hello -macosx_version_min 10.7.0 -lSystem
   ```

MacOS (Optional): Assembling NASM in XCode

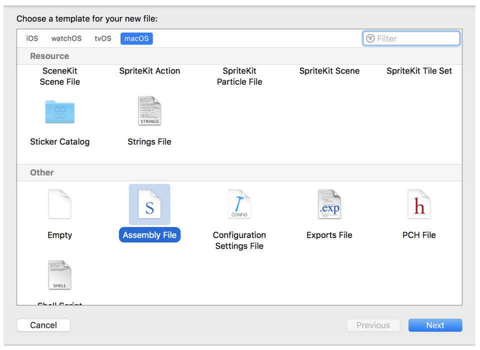
- Start a new XCode project, selecting “Command Line Tool”. Choose C as the language.

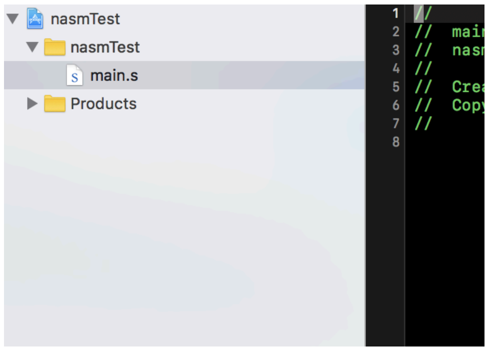



Choose options for your new project:

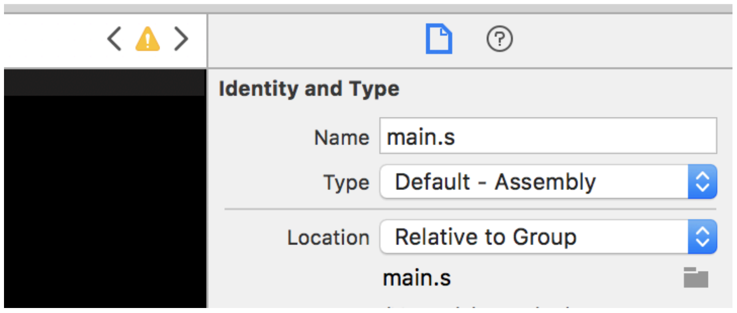
Product Name: nasmTest
Team: None
Organization Name: tag.utd
Organization Identifier: tag.utd
Bundle Identifier: tag.utd.nasmTest
Language: C

- Remove your main.c file, and replace it with an assembly file. In this case, main.s

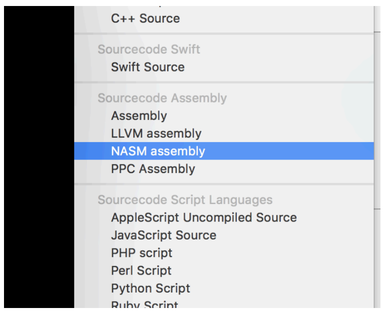




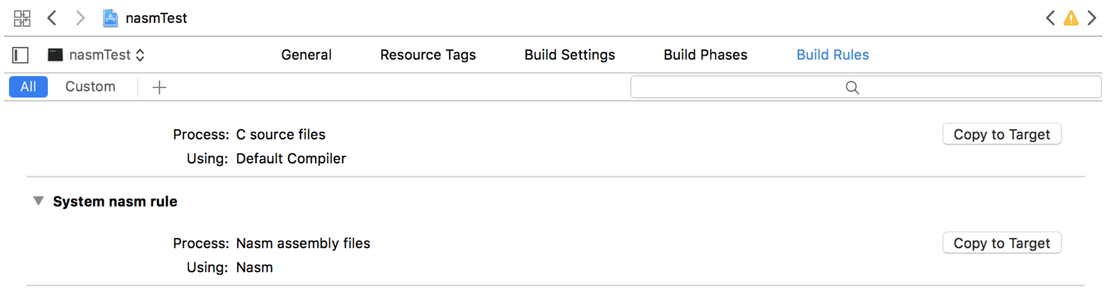
- With main.s open, go to the right side of the page and change the type to NASM


Identity and Type

Name: main.s
Type: Default - Assembly
Location: Relative to Group
main.s



- Now, go into build rules, and select “Copy to Target” under “System nasm rule”

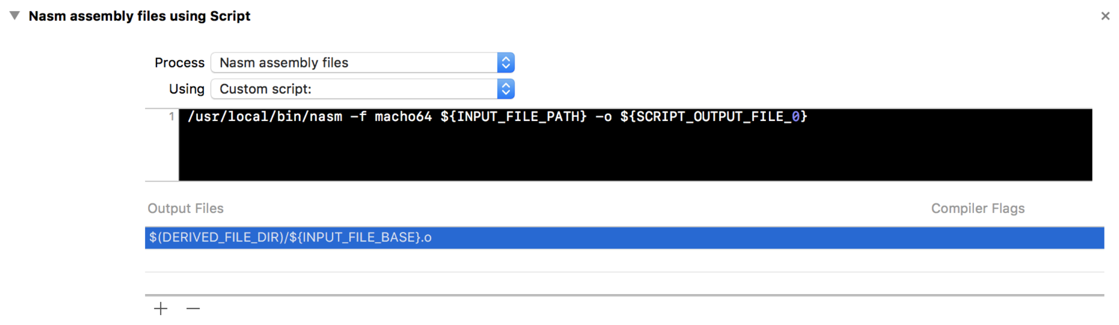


Process: C source files
Using: Default Compiler
Copy to Target

System nasm rule

Process: Nasm assembly files
Using: Nasm
Copy to Target

- In your new NASM system rule, enter the following text...

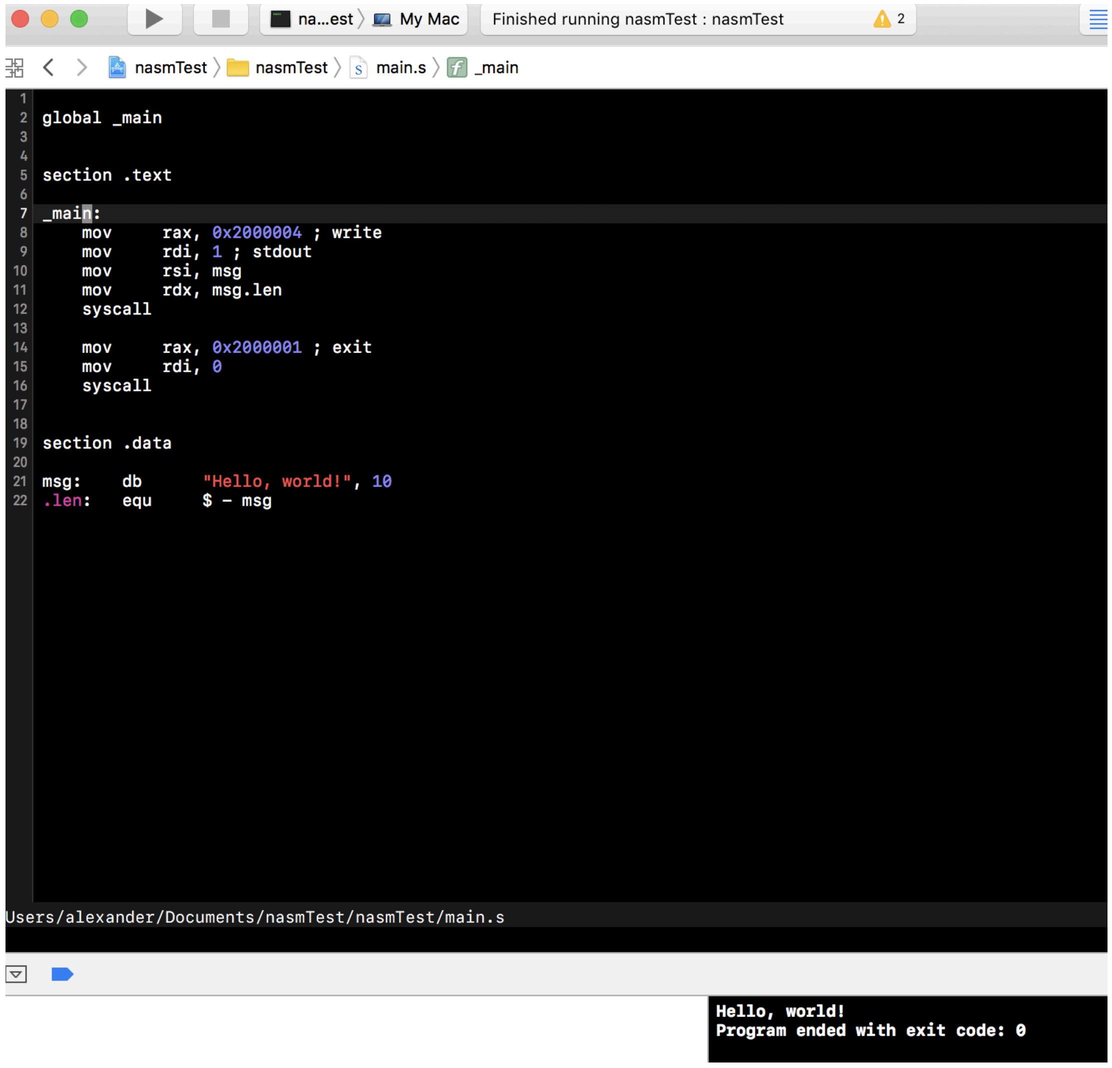


Process: Nasm assembly files
Using: Custom script:

```
/usr/local/bin/nasm -f macho64 ${INPUT_FILE_PATH} -o ${SCRIPT_OUTPUT_FILE_0}
```

Output Files: $(DERIVED_FILE_DIR)/$(INPUT_FILE_BASE).o

- Now, XCode should assemble NASM code for you! Just make sure that you include a symbol for _main somewhere in your code.



```

global _main


section .text

_main:
    mov     rax, 0x2000004 ; write
    mov     rdi, 1 ; stdout
    mov     rsi, msg
    mov     rdx, msg.len
    syscall

    mov     rax, 0x2000001 ; exit
    mov     rdi, 0
    syscall


section .data

msg:    db      "Hello, world!", 10
.len:   equ     $ - msg
```

```
Users/alexander/Documents/nasmTest/nasmTest/main.s
```

```
Hello, world!
Program ended with exit code: 0
```