

MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF REPUBLIC OF MOLDOVA
TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Computer Architecture

Laboratory work 4 : Simple problems in Assembly

Elaborated:

st.gr. FAF-211

Grama Alexandru

Verified:

asist.univ.

Vladislav Voitcovich

Chişinău, 2023

Content

Introduction	3
Objectives	4
Task 1	5
Task 2	5
Task 3	8
Task 4	13
Conclusions	14

Introduction

Assembly language, also known as "asm," is a low-level programming language that is used to write instructions for a computer's processor. It is a symbolic representation of the machine code that the processor can understand and execute. Assembly language is considered a low-level language because it is very close to the actual hardware of the computer, making it fast and efficient, but also very difficult to write and read compared to high-level languages.

In assembly language, each instruction represents a specific operation that the processor can perform, such as adding two numbers or moving data from one memory location to another. The instructions are written using mnemonics, which are short codes that represent the actual machine code instructions.

Assembly language is commonly used in system programming, device drivers, and embedded systems, where performance and efficiency are critical. It requires a good understanding of computer architecture and can be challenging to write and debug, but it can also be very rewarding and powerful for the programmer who masters it.

Assembly language is an important tool for programmers who need to work with low-level hardware and software interfaces. It allows them to write code that can interact directly with the hardware of a computer, such as the processor, memory, and input/output devices.

Because assembly language is so closely tied to the hardware of a computer, it is often used in operating system development, as well as in the development of device drivers, firmware, and other system-level software. It can also be used to optimize performance in high-performance computing applications, such as scientific simulations or video game engines.

In addition to its use in system programming, assembly language is also commonly used in the development of embedded systems, which are small computers or microcontrollers that are integrated into larger systems, such as appliances, vehicles, and medical devices. Because these systems often have limited processing power and memory, assembly language can be a useful tool for writing efficient and compact code.

Assembly language programming requires a strong understanding of computer architecture and the instruction set of the target processor. It can be challenging to write and debug, and requires a great deal of attention to detail. However, it can also be a very powerful and rewarding programming language, allowing programmers to write code that is highly optimized for specific hardware and software environments.

Objectives

1. Low-level hardware programming: Assembly language allows developers to write code that interacts directly with the hardware of a computer or device. This is useful for low-level programming tasks such as device drivers or system-level software.
2. Embedded systems programming: Assembly language is commonly used for programming small computers or microcontrollers that are integrated into larger systems, such as vehicles or medical devices.
3. Performance optimization: Assembly language provides a high degree of control over how code is executed, which can be used to optimize performance for specific hardware or software environments.
4. Reverse engineering: Assembly language is often used for reverse engineering, which involves analyzing software or hardware to understand how it works. This can be useful for security research or to gain a better understanding of legacy software or hardware.
5. Academic research: Assembly language can be a useful tool for academic research in fields such as computer science or engineering, where low-level programming tasks are required.
6. PPersonal learning: Learning assembly language can be a valuable personal development goal for programmers who want to gain a deeper understanding of computer architecture and low-level programming concepts.
7. Porting software: Assembly language can be used to port software from one platform to another, particularly in cases where the original source code is not available or is difficult to modify.
8. Debugging and optimization of high-level code: Assembly language is often used to debug and optimize high-level code by examining the machine code generated by the compiler and making adjustments to improve performance or fix bugs.
9. Operating system development: Assembly language is a key tool for operating system development, particularly for low-level tasks such as interrupt handling and memory management.
10. Real-time systems programming: Assembly language is commonly used in real-time systems programming, where precise timing and fast execution are critical. This includes applications such as control systems, robotics, and aviation systems.

Task 1

Familiarize yourself with the basics of Assembly .

Assembly Language is a low-level programming language that is used to write programs for computers, microcontrollers, and other devices. In Assembly Language, the programmer writes code that corresponds directly to the machine code instructions that the computer can execute. This makes it a powerful language for writing highly optimized and efficient code, but also makes it more complex and difficult to work with than higher-level programming languages.

At the heart of Assembly Language are registers and memory. Registers are small, fast, on-chip storage locations that can be used to store data or perform arithmetic and logical operations. Memory, on the other hand, is a larger, slower storage area that can hold much more data than registers, but is more expensive to access.

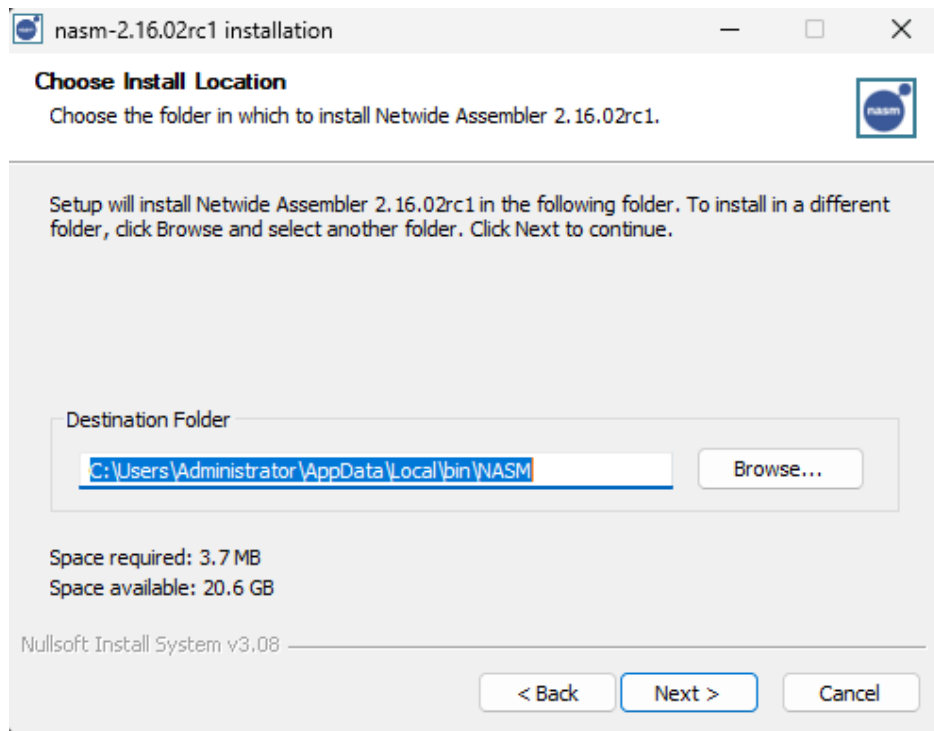
Assembly Language instructions are written using a specific syntax that varies depending on the processor architecture and the specific assembly language being used. Instructions typically consist of an operation code (opcode) that specifies the operation to be performed, along with one or more operands that provide the data for the operation.

To write Assembly Language code, a programmer must have a deep understanding of the underlying hardware architecture, including the registers, memory, and instruction set of the processor being used. They must also be able to read and write machine code in hexadecimal format.

Overall, Assembly Language is a powerful and important tool for programming low-level systems, but it requires a high level of technical expertise and a deep understanding of computer hardware.

Task 2

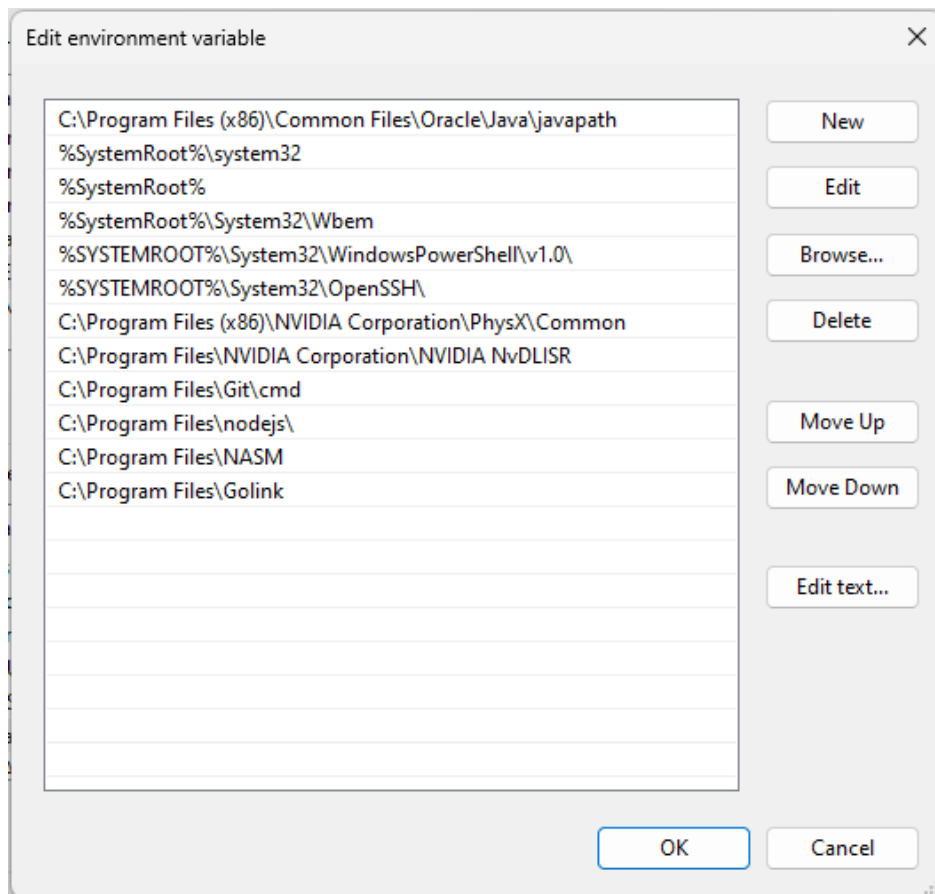
NASM is a popular assembler used to write code in assembly language. Sublime Text is a versatile text editor that is commonly used for programming, and Golang is a linker used to create executable files from object files. Here are the steps to install these tools on a Windows system:



NASM installing

The first step is to download the NASM installer for Windows from the official website. Run the installer and follow the instructions to install NASM on your system. Once installed, you can check if it's properly installed by opening a command prompt and typing "nasm -v". If NASM is installed correctly, it will display the version number.

To install Sublime Text, visit the official website and download the installer for your version of Windows. Run the installer and follow the instructions to install Sublime Text on your system. Once installed, you can launch Sublime Text from the Start menu.

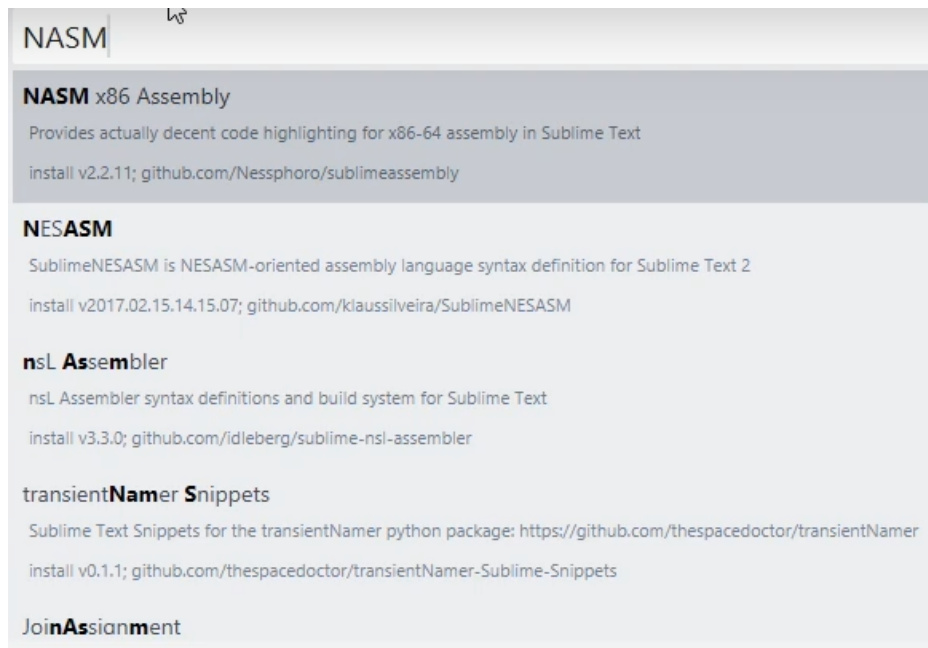


Setting administrator rights for both GoLink and NASM

To install Golink, download the zip file from the official website and extract it to a folder on your system. You can add the folder to your system PATH to make it easier to use. To do this, right-click on "This PC" in Windows Explorer and select "Properties". Click on "Advanced system settings", then click on "Environment Variables". Under "System Variables", select "Path" and click "Edit". Add the path to the Golink folder to the list of paths and click "OK" to save your changes.

Configuring Sublime Text for Assembly Language Programming

To configure Sublime Text for assembly language programming, you can install a package called "Easy Assembly". To do this, open Sublime Text and go to "Preferences" > "Package Control". In the search bar, type "Assembly x86" and select it from the list. Click "Install" to install the package. Once installed, you can use it to write and edit assembly language code.



NASM installing into Sublime Text

Problem 1

This program uses NASM assembly language to write a message "Hello from NASM on Windows!" to the console output using the Windows API functions `GetStdHandle@4` and `WriteFile@20`. It then exits the program using the `ExitProcess@4` function. The program defines constants and variables to make the code more readable and maintainable. Overall, this program demonstrates a simple but effective use of assembly language for writing to the console output in Windows.

```

NULL    EQU 0

STD_OUTPUT_HANDLE    EQU -11

extern _GetStdHandle@4
extern _WriteFile@20
extern _ExitProcess@4


global Start


section .data
    Message      db "Hello from NASM on Windows!", 0Dh, 0Ah
    MessageLength EQU $-Message
section .bss
    StandardHandle    resd 1
    Written    resd 1

```



```

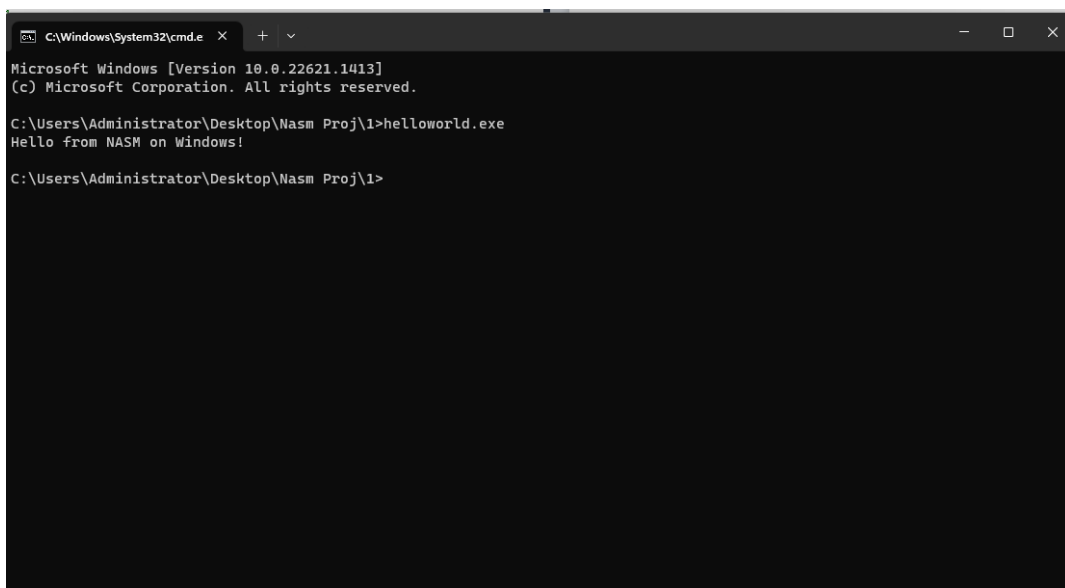
section .text

Start:

push  STDOUT_HANDLE
call  _GetStdHandle@4
mov   dword [StandardHandle], EAX

push  NULL
push  Written
push  MessageLength
push  Message
push  dword [StandardHandle]
call  _WriteFile@20
push  NULL
call  _ExitProcess@4

```



```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Administrator\Desktop\Nasm Proj\1>helloworld.exe
Hello from NASM on Windows!

C:\Users\Administrator\Desktop\Nasm Proj\1>

```

Output for Hello-World program

Problem 2

The program calculates the sum of two numbers (10 and 20) and displays the result (30) as a string on the console. It does this by first getting the standard output handle, adding the two numbers, converting the result to a string using a `DecimalToString` function, and then writing the message and result to the console using the `WriteFile` function. Finally, it exits the program using the `ExitProcess` function.

```

NULL EQU 0

STDOUT_HANDLE EQU -11

```

```

extern _GetStdHandle@4
extern _WriteFile@20
extern _ExitProcess@4

global Start

section .data
    Message db "The result is:", 0
    MessageLength EQU $-Message
    num1 dd 10
    num2 dd 20
    result dd 0

section .bss
    StandardHandle resd 1
    Written resd 1
    result_str resb 11

section .text
    Start:
        ; Get the standard output handle
        push STD_OUTPUT_HANDLE
        call _GetStdHandle@4
        mov dword [StandardHandle], EAX

        ; Add the two numbers and store in result
        mov EAX, [num1]
        add EAX, [num2]
        mov [result], EAX

        ; Convert the result to a string
        mov EAX, [result]
        call DecimalToString

```

```

; Write the message and the result to standard output
push NULL
push Written
push MessageLength
push Message
push dword [StandardHandle]
call _WriteFile@20

push NULL
push Written
push 11
push result_str
push dword [StandardHandle]
call _WriteFile@20

; Exit the program
push NULL
call _ExitProcess@4

```

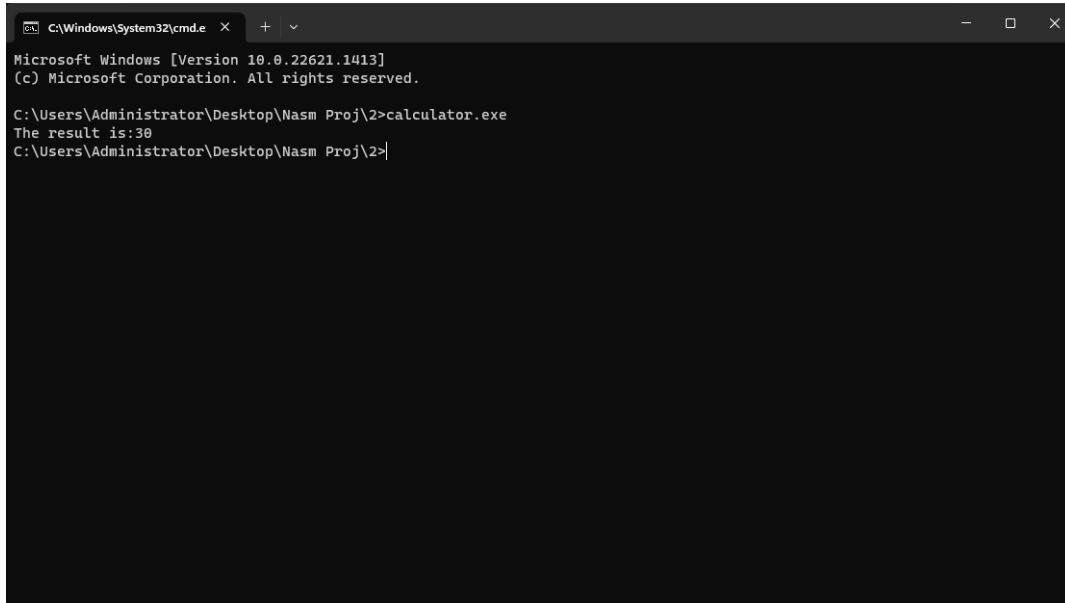
DecimalToString:

```

push EBP
mov EBP, ESP
sub ESP, 8
mov dword [EBP-4], 0
mov dword [EBP-8], 10
.Loop:
xor EDX, EDX
div dword [EBP-8]
add DL, '0'
mov byte [result_str+eax], DL
inc dword [EBP-4]
test EAX, EAX
jnz .Loop
mov EAX, dword [EBP-4]

```

```
add ESP, 8
pop EBP
ret
```



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Administrator\Desktop\Nasm Proj\2>calculator.exe
The result is:30
C:\Users\Administrator\Desktop\Nasm Proj\2>
```

Output for the addition program

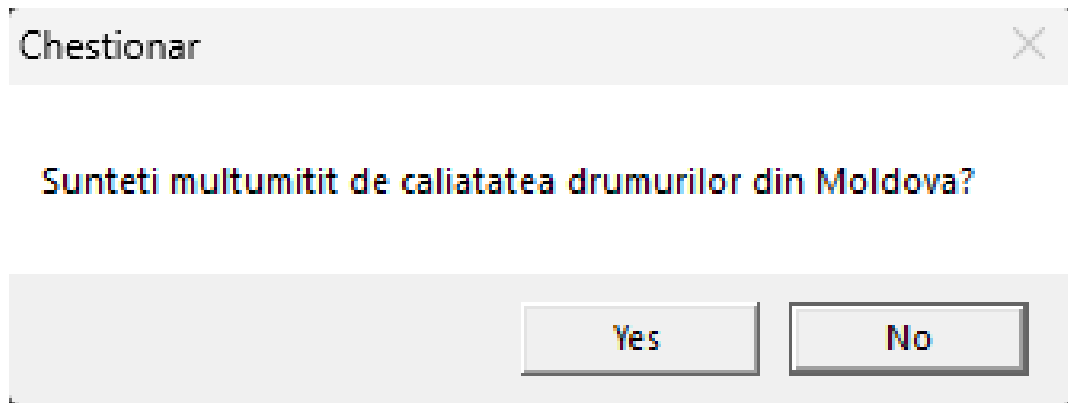
Problem 3

The code displays a message box with question about the quality of the roads in Moldova, giving the user the option to choose "Yes" or "No". If the user selects "No", the program goes back to the start of the loop and displays the message box again. If the user selects "Yes", the program exits. The program uses the MessageBoxA and ExitProcess functions from the Windows API to display the message box and exit the program, respectively.

Code:

```
NULL EQU 0
MB_DEFBUTTON EQU 100h
IDNO EQU 7
MB_YESNO EQU 4
extern _MessageBoxA@16
extern _ExitProcess@4
global Start
section .data
MessageBoxMessage db "Sunteti multumitit de caliatatea drumurilor din Moldova?", 0
MessageBoxTitle db "Chestionar", 0
```

```
section .text
Start:
push MB_YESNO | MB_DEFBUTTON
push MessageBoxTitle
push MessageBoxMessage
push NULL
call _MessageBoxA@16
cmp EAX, IDNO
je Start
push NULL
call _ExitProcess@4
```



Programs's message window

Task 4

Debugging in NASM can be a challenging process, but there are some tools and techniques that can make it easier. One of the most commonly used debugging tools is the GNU Debugger (GDB), which is a command-line tool that can be used to step through a program one instruction at a time, examine the values of variables and memory locations, and set breakpoints to pause the program at specific points. Another popular debugger is the OllyDbg, a graphical debugger for Windows, which can also be used to step through a program and examine variables and memory. In addition to using debuggers, it's also important to write clear and well-structured code, use descriptive variable names, and include comments to make it easier to understand and debug the code. It can also be helpful to test the code in small parts and to print out values or messages to the console to track the progress of the program. Overall, debugging in NASM requires patience, attention to detail, and a good understanding of the code being written.

Conclusions

In conclusion, using NASM with Sublime Text as the code editor and GoLink as the linker provides a simple and efficient way to write and debug x86 assembly language programs on Windows. Sublime Text's syntax highlighting and code completion features help make the code writing process faster and more efficient, while GoLink's easy-to-use interface allows for seamless linking of object files and generation of executable files. Additionally, the use of NASM's built-in debugging tools, such as the gdb debugger and the strace system call tracer, provides developers with powerful tools for identifying and resolving issues in their code. Overall, this combination of tools provides a reliable and streamlined environment for developing x86 assembly language programs on Windows. One important aspect of programming with NASM is the ability to work with external libraries and functions. In order to do this, you need to use the "extern" directive to declare the external symbols. Once you have declared the symbols, you can use them in your code just like any other symbol. However, it is important to make sure that the arguments you pass to the external functions are in the correct order and format. If you are not sure, you can consult the documentation for the library or function you are using.

Another important aspect of programming with NASM is debugging. Debugging is the process of finding and fixing errors in your code. NASM provides several tools that can help you debug your code, including the ability to print messages to the console, step through your code one instruction at a time, and set breakpoints to stop your code at specific points. These tools can be very helpful in tracking down and fixing errors in your code.

When debugging NASM code, it can be very helpful to use an integrated development environment (IDE) or text editor that supports NASM syntax highlighting and debugging. Sublime Text is one such editor that provides support for NASM syntax highlighting and debugging. Golink is another tool that can be used to link your NASM code with external libraries and functions. By using these tools, you can make your programming experience with NASM more efficient and effective.