

**“AZƏRBAYCAN HAVA YOLLARI” CJSC NATIONAL AVIATION ACADEMY**

**Course Work**

**Speciality:Computer Engineering**

**Faculty:Aerokosmic**

**Group: 1459i**

**Course:3**

**Subject: System programming and operating systems**

**Teacher: Sahmaliyev Mehemmed**

**Student: Hesenova Firuze**

## **Topic:** Developing device drivers in assembly language

**Baku 2022**

**Contents**

Introduction...............................................................................................1

## §1.What are Device Drivers?....................................................................2

§2.How do Device Drivers Work?............................................................4

## §3.Purpose of Device Drivers...................................................................7

## §4.Types of Device Drivers......................................................................8

## §5.What Is an Assembly Language?.......................................................11

## §6.How Assembly Languages Work?.....................................................12

## §7.Why is learning assembly language still important?..........................14

§8.Assembly Code...................................................................................16

§9.C++ code.............................................................................................17

§10.**Literature**...........................................................................................18

**Introduction**

When a user adds a new component to the computer system, for instance a printer, the device drivers will allow the communication and recognition of the component with the operating system, so that the new piece of hardware can function properly.

In the early days of programming and computers, device drivers would be created using assembly language, which refers to a low level language that can access hardware and CPU instructions quickly and directly.

Though nowadays, most programmers who write code for device drivers work with program languages such as C or C++.

These programming languages offer great access to low level instructions as well as extremely complex program instructions and data structures.

Programmers must not only create program the functionality of device drivers, they must also make sure these drivers incorporate a high leave of security.

This is because such software operates at high levels of privilege, thus leading to system compromise or vulnerabilities from a security point of view, it may also cause operational faults such as crashing or freezing, and therefore decreasing the overall performance of the system.

## §1.What are Device Drivers?

In computing a **device driver** refers to a computer program, which controls or operates a particular type of device that is attached to the computer. A device driver provides a software interface to hardware devices, thus enabling operating systems and computer programs to access and communicate with the hardware functions without requiring to acknowledge details about the hardware that is being used.

This communication occurs through a computer bus, or a communications subsystem to which the hardware connects.

A ****computer bus****refers to a communication system that transfers data between components inside a computer or between computers.

When a device driver calls a program, it invokes a routine in the driver, and thus the driver then will issue commands to the device.

When the device sends data back to the driver, the driver then can invoke routines in the original calling program.

A ****routine**** refers to is a sequence of program instructions that perform a specific task, packaged as a unit, moreover, this unit can be used in programs wherever that particular task should be performed. Routines can be defined within programs, or even in libraries that can be used by many programs.

Drivers can be recognized to be hardware dependent and operating system specific, furthermore, device drivers provide the interrupt handling required for any necessary asynchronous time dependent hardware interface.

In an operating system, device drivers are required because some computer components do not use standard commands.

For instance, two video cards from different manufacturers have the same functionality, however each video will have to require its own driver, as different hardware require different commands. Furthermore, operating systems also require different drivers, a driver that is written for Microsoft Windows cannot be used by a driver that is written for macOS.

Many computer components require device drivers, and some common examples are:

Graphic cards, computer printers, scanners, modems, sound cards, network cards etc.

Likewise, there are other computer peripherals, which don’t require device drivers, as driver is already built in the operating system, therefore when those peripherals are required, the operating system recognizes them and knows how to use them.

Examples of such parts include:

USB sound cards, USB hubs (a component which turns one USB more to two or more), hard drives, floppy drives, mouse, keyboard, RAM’s, processors, CD, DVD etc.

It is essential that the correct device driver is used, as a wrong device driver can prevent the hardware from working correctly, furthermore, keeping drivers updated can avoid potential problems when using programs with new piece of hardware.

## §2.How do Device Drivers Work?

Within the operating system runtime environment, device drivers run at a high level of privilege. As a matter of fact, some device drivers are directly linked to the operating system kernel, which is a portion of an operating system, such as Mac OS, Windows, or Linux that remains memory resident.

Device drivers function by relaying requests for device access and actions from the operating system and any active application to the respective hardware devices. Furthermore, device drivers also deliver outputs or messages/status from the hardware devices to the operating system (and thus to the applications themselves).

In the instance of a scanner or a printer, both of these systems have device drivers, moreover, both of these are accompanied by a compact disk, which includes the software code for losing the device driver.

In order to establish a connection with the PC, it is mandatory to load the software into your PC, so the PC can identify the new peripheral and communicate with it. Although some of the new systems don’t require the user to go through the process of installing the driver.

Anyhow, when the driver is installed it creates what is referred to is a ****device object****, which is deigned to control the specific component, which was attached to the system. The driver represents the peripheral device and consists of a physical structure of modes that will make up the process of allowing your operating system to control the peripheral device.

The device driver communicates with the computer system (i.e. your PC) through the computer bus which is used to connect the device with computer.

Device drivers operate within the kernel of the operating system. The kernel is a part of the of the operating system that has direct interactions with the physical structure of system.

Rather than interacting and accessing the new device directly, the operating system will load the device drivers and it will call the specific functions within the driver software to execute specific tasks on the device, for instance in our case, printing a sheet of paper or scanning a document, thus meaning that each driver has the device specific content codes which are required to carry out the actions on the device.

**Architecture of Device Drivers**

At the basic level, device drivers can be split into two layers, logical layer, and a physical layer.

****Logical Layer:****

Logical layers process data for a class of devices.

****Physical Layer:****

Physical layers communicate with specific device instances.

In order for them to function together, the logical layer need to communicate with a particular serial port chip, and thus the physical layer will address this.

Every device driver consists of two important data structures: The device information structure and the static structure.

These structures are used to install the device driver as well as to share information among the entry point routines.

****Device information structure:****

The device information structure is a static file, which is passed to the install entry point, and its functionality is to pass along any information required to install a major device into the install entry point where it is going to be used to initialise the static structure.

****Static structure:****

The static structure is used to pass along information between the different entry points and is initialised with the information stored in the information structure.

In order for the operating system to communicate with the driver, it uses the entry point routines.

## §3.Purpose of Device Drivers

The main purpose and functionality of device drivers is to provide a level of abstraction by acting as a translator between a hardware device (i.e. a printer or a scanner) and the applications or the operating system that use it.

Though programmers can independently write higher-level code of whatever specific hardware the end user is using.

For instance, a high level application, which interacts with a serial port, may have two functions for ****send data**** and ****receive data.****

On the contrary, device drivers at low-level applications implement these function with the need to communicate to the specific serial port controller installed on the computer.

## §4.Types of Device Drivers

Device drivers are preset in almost every device associated with a computer system, whether it was BIOS (Basic Input Output System) or a virtual machine, they all require device drivers.

These device drivers can be broadly classified into two categories:

****Kernel Device Drivers:****

A kernel device driver is a generic device driver that will load with the operating system into the memory as part of the operating system, furthermore, as soon as the driver is required, it can be invoked using a pointer.

These drivers are applicable to the motherboard, BIOS, processor, and similar hardware, which are part of the kernel software.

Though one of the problem that arise from kernel device drivers is that when they are required and invoked, thy are loaded into the RAM (Random Access Memory)  thus they cannot manage to move to the page file (i.e. virtual memory like a hard disk SSD). Therefore is there are a number of device drivers which are operating at the same time, they can overall slow down the machine, as they will be heavily relying on the RAM, hence why there is a minimum system requirement for each operating system.

****Random Access Memory (RAM):****Random Access Memory refers to the computer memory that can be read and changed in any order. This type of memory is used to store working data and machine code. A computer system comes with built in RAM.

****User Mode Device Drivers:****

User mode device drivers are device drivers that are triggered usually during a users session on a computer.

Drivers for “plug and play” services fall in to this category. User device drivers can be written to the disk in order for it to not strain the resources on the computer.

Though device drivers, which are relating to gaming devices, it is recommended to keep them in the main memory (RAM).

## Other types of drivers include

****Block Drivers and Character Drivers:****

These two device drivers can be categorised to belonging to the data reading and writing category.

****Character Drivers:****

These drivers are used as serial buses, they write data one character at a time (meaning a byte in a generic terms). Therefore if a device/component is connected to the computer system via a serial port, then they are characterised as being a character driver, for instance a keyboard or a mouse.

****Block Drivers:****

Block drivers are drivers that refer to writing and reading of more than one character at a time. These drivers create blocks and retrieve as much information as the block can withstand.

****Generic and OEM (Original Equipment Manufacturer) Drivers:****

Generic device drivers are can be generic or even OEM related.

If the device driver comes with its own operating software, then it is categorised as being generic.

Generic drivers can be utilised with a number of different brands of a particular device type.

For instance, Windows 10 works with a number of generic drivers that function without having the need to install any other software manually.

On the other original equipment manufacturers (OEM) create their own device drivers. These drivers have to be installed separately after installing the operating system.

****Virtual Drivers:****

Virtual drivers refer to software that is used to emulate hardware and other devices so that multiple applications running in protected mode can have access hardware interrupt channels, hardware resources, and memory without causing conflicts.

## §5.What Is an Assembly Language?

You'll hear it said: "I thought assembly was a dead language, why waste my time?" Well, you might not find yourself writing your next app in assembly, but there is still much to gain from learning assembly. Today, assembly language is used primarily for direct hardware manipulation, access to specialized processor instructions, or to address critical performance issues. Typical uses are device drivers, low-level embedded systems, and real-time systems

An ***assembly language*** is a type of low-level [programming](https://www.investopedia.com/career-advice-4689809) language that is intended to communicate directly with a computer’s hardware. Unlike machine language, which consists of binary and hexadecimal characters, assembly languages are designed to be readable by humans.

Low-level programming languages such as assembly language are a necessary bridge between the underlying hardware of a computer and the higher-level programming languages—such as Python or JavaScript—in which modern software programs are written.

## §6.How Assembly Languages Work?

Fundamentally, the most basic instructions executed by a computer are binary codes, consisting of ones and zeros. Those codes are directly translated into the “on” and “off” states of the electricity moving through the computer’s physical circuits. In essence, these simple codes form the basis of “machine language”, the most fundamental variety of programming language.

Of course, no human would be able to construct modern software programs by explicitly programming ones and zeros. Instead, human programmers must rely on various layers of abstraction that can allow themselves to articulate their commands in a format that is more intuitive to humans. Specifically, modern programmers issue commands in so-called “high-level languages”, which utilize intuitive syntax such as whole English words and sentences, as well as logical operators such as “And”, “Or”, and  “Else” that are familiar to everyday usage.

**Real World Example of an Assembly Language**

Today, assembly languages remain the subject of study by computer science students, in order to help them understand how modern software relates to its underlying hardware platforms. In some cases, programmers must continue to write in assembly languages, such as when the demands are performance are especially high, or when the hardware in question is incompatible with any current high-level languages.

One such example that is relevant to finance are the [high-frequency trading](https://www.investopedia.com/terms/h/high-frequency-trading.asp) **[(HFT)](https://www.investopedia.com/terms/h/high-frequency-trading.asp)** platforms used by some financial firms. In this marketplace, the speed and accuracy of transactions is of paramount importance in order for the HFT trading strategies to prove profitable. Therefore, in order to gain an edge against their competitors, some HFT firms have written their trading software directly in assembly languages, thereby making it unnecessary to wait for the commands from a higher-level language to be translated into machine language.

Ultimately, however, these high-level commands need to be translated into machine language. Rather than doing so manually, programmers rely on assembly languages whose purpose is to automatically translate between these high-level and low-level languages. The first assembly languages were developed in the 1940s, and although modern programmers spend very little time dealing with assembly languages, they nevertheless remain essential to the overall functioning of a computer.

Today, assembly languages remain the subject of study by computer science students, in order to help them understand how modern software relates to its underlying hardware platforms. In some cases, programmers must continue to write in assembly languages, such as when the demands are performance are especially high, or when the hardware in question is incompatible with any current high-level languages.

# §7.Why is learning assembly language still important?

An [assembly language](https://www.techopedia.com/definition/3903/assembly-language), also known as an assembler language, is a [low-level programming language](https://www.techopedia.com/definition/3933/low-level-language). Assembly language has a very strong correspondence with the architecture’s [machine code](https://www.techopedia.com/definition/8179/machine-code-mc) instruction and is specific only to that machine. Therefore, different machines have different assembly languages. This type of language makes use of symbols to represent an operation or instruction. Hence, it is also often known as symbolic machine code.

Despite the prevalence of [high-level languages](https://www.techopedia.com/definition/3925/high-level-language-hll) that are mainly used for the development of applications and software programs, the importance of assembly language in today’s world cannot be understated. A [programmer](https://www.techopedia.com/definition/4813/programmer) can still gain a lot if he/she can learn to code in assembly language and implement it. These days, assembly language makes it possible to manipulate hardware directly, address critical issues concerning performance and also provide access to special instructions for processors. Uses of assembly language include coding device drivers, real-time systems, low-level embedded systems, boot codes, reverse engineering and more.

The following are some of the reasons why learning assembly language is still important and relevant.

## Complete control over a system’s resources

As a programmer, the closest you can come to the [processor](https://www.techopedia.com/definition/28254/processor) of a machine is by using assembly language. Here, you can write code to access the registers and even deal with [memory addresses](https://www.techopedia.com/definition/323/memory-address) directly for retrieving values and pointers. So, if you are writing a program that has a great [algorithm](https://www.techopedia.com/definition/3739/algorithm), you are going to benefit greatly. This is mainly because assembly language is the gateway to optimization in speed, thereby offering great efficiency and performance.

## Understanding processor and memory function

If you are writing a program that is meant to be a compiler or a device driver, then a complete understanding of the processor’s function is a big plus. So, in this case the best option is to write some code in assembly language and see how the processor and the memory work. However, one thing to keep in mind is that assembly language is symbolic, so it may appear cryptic. Furthermore, the source code in assembly language is always larger than that of a high-level language. However, putting in time and effort to master it can benefit one greatly in terms of understanding.

## Direct access to hardware

Assembly language is the only language that speaks to the computer/machine directly. It is the language that a certain [CPU](https://www.techopedia.com/definition/2851/central-processing-unit-cpu) recognizes and different CPUs recognize different types of them. However, since every section of binary caters to a certain meaning, it can be somewhat easy to comprehend.

## Assembly language is transparent

When compared to high-level languages, which are mostly in the form of abstract data types, assembly language is bare and transparent. This is largely since it has a small number of operations. So, this is very helpful for algorithm analysis, consisting of semantics and flow of control. It also makes it easier for [debugging](https://www.techopedia.com/definition/16373/debugging), as it is less complex. Overall, there is less overhead as compared to high-level languages.

**§8.Assembly Code**

I will add the asm\_func function that will call a [MessageBox](https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-messageboxa" \t "https://dennisbabkin.com/blog/_blank) API, but before that, it will call our C++ function GetMsgBoxType. So let's code it all:

.data

msgCaption db"Message box text",0

.code

ALIGN 16

EXTERN GetMsgBoxType : PROC;

EXTERN MessageBoxA : PROC

EXTERN \_\_imp\_MessageBoxA : QWORD

asm\_func PROC

; RCX = address for the string for the message box

sub rsp, 28h ; shadow stack

mov [rsp], rcx

call GetMsgBoxType

mov r9, rax

mov r8, [rsp]

lea rdx, [msgCaption]

xor ecx, ecx

call [\_\_imp\_MessageBoxA]

add rsp, 28h 2 ; restoring shadow stack

ret

asm\_func ENDP

END

**§9.C++ code**

Lastly, we need to write our C++ code to call our Assembly function. In that case I will use the auto-generated .cpp file to write the following:

#include <iostream>

#include <Windows.h>

extern "C" void \_\_fastcall asm\_func(const char\* lpText);

int main()

{

asm\_func("Hello world!");

}

extern "C" UINT GetMsgBoxType()

{

return MB\_YESNOCANCEL | MB\_ICONINFORMATION;

}

**§10.Literature**

1. *[https://www.sciencedirect.com/topics/engineering/device-driver](https://www.sciencedirect.com/topics/engineering/device-driver" \t "https://teachcomputerscience.com/device-drivers/_blank)*
2. *[spamlaws.com/how-device-drivers-work.html](https://spamlaws.com/how-device-drivers-work.html)*
3. *[https://www.engineersgarage.com/how\_to/how-device-drivers-work/](https://www.engineersgarage.com/how_to/how-device-drivers-work/" \t "https://teachcomputerscience.com/device-drivers/_blank)*
4. *[https://en.wikipedia.org/wiki/Subroutine](https://en.wikipedia.org/wiki/Subroutine" \t "https://teachcomputerscience.com/device-drivers/_blank)*
5. *[https://searchenterprisedesktop.techtarget.com/definition/device-driver](https://searchenterprisedesktop.techtarget.com/definition/device-driver" \t "https://teachcomputerscience.com/device-drivers/_blank)*
6. *[spamlaws.com/how-device-drivers-work.html](https://spamlaws.com/how-device-drivers-work.html)*
7. *[https://www.thewindowsclub.com/what-is-device-driver/](https://www.thewindowsclub.com/what-is-device-driver/" \t "https://teachcomputerscience.com/device-drivers/_blank)*
8. *[https://www.slideshare.net/suhassr/device-drivers](https://www.slideshare.net/suhassr/device-drivers" \t "https://teachcomputerscience.com/device-drivers/_blank)*
9. *[https://www.techopedia.com/definition/6824/device-driver](https://www.techopedia.com/definition/6824/device-driver" \t "https://teachcomputerscience.com/device-drivers/_blank)*
10. *[https://simple.wikipedia.org/wiki/Device\_driver](https://simple.wikipedia.org/wiki/Device_driver" \t "https://teachcomputerscience.com/device-drivers/_blank)*
11. *[https://www.computerhope.com/jargon/d/driver.htm#:~:text=More%20commonly%20known%20as%20a,devices%2C%20such%20as%20a%20printer.](https://www.computerhope.com/jargon/d/driver.htm" \l ":~:text=More commonly known as a,devices, such as a printer." \t "https://teachcomputerscience.com/device-drivers/_blank)*
12. *[https://www.techwalla.com/articles/device-driver-examples](https://www.techwalla.com/articles/device-driver-examples" \t "https://teachcomputerscience.com/device-drivers/_blank)*
13. *[https://www.computerscience.gcse.guru/theory/software-device-drivers](https://www.computerscience.gcse.guru/theory/software-device-drivers" \t "https://teachcomputerscience.com/device-drivers/_blank)*
14. *<https://www.techopedia.com/why-is-learning-assembly-language-still-important/7/32268>*
15. *<https://teachcomputerscience.com/device-drivers/>*
16. *<https://www.investopedia.com/terms/a/assembly-language.asp>*
17. *<https://dennisbabkin.com/blog/?i=AAA10700>*
18. *Windows NT Device Driver Development [Peter G. Viscarola](https://bookauthority.org/author/Peter-G.-Viscarola" \t "https://bookauthority.org/books/_blank), [W. Anthony Mason](https://bookauthority.org/author/W.-Anthony-Mason" \t "https://bookauthority.org/books/_blank) 1998*
19. *Programming the Microsoft Windows Driver Model (Developer /2nd edition (26 Dec. 2002))*
20. *Designing a Driver Vigilance Monitoring Device[Wafik Moussa](https://bookauthority.org/author/Wafik-Moussa" \t "https://bookauthority.org/books/_blank)*

### *Application Driven Development of an FPGA Based Multiprocessor System [Wafik Moussa](https://bookauthority.org/author/Wafik-Moussa" \t "https://bookauthority.org/books/_blank) 2009*

# **21*.****Writing Windows V*x*Ds and Device Drivers By [Karen Hazzah](https://www.routledge.com/search?author=Karen Hazzah" \o "Search for more titles by Karen Hazzah) Copyright Year 1997(2nd edition)*