Process Report | Digital Techniques

Introduction

In this document, I will discuss the most important concepts I learned in my Digital Techniques course, accompanied by relevant proof of my learning.

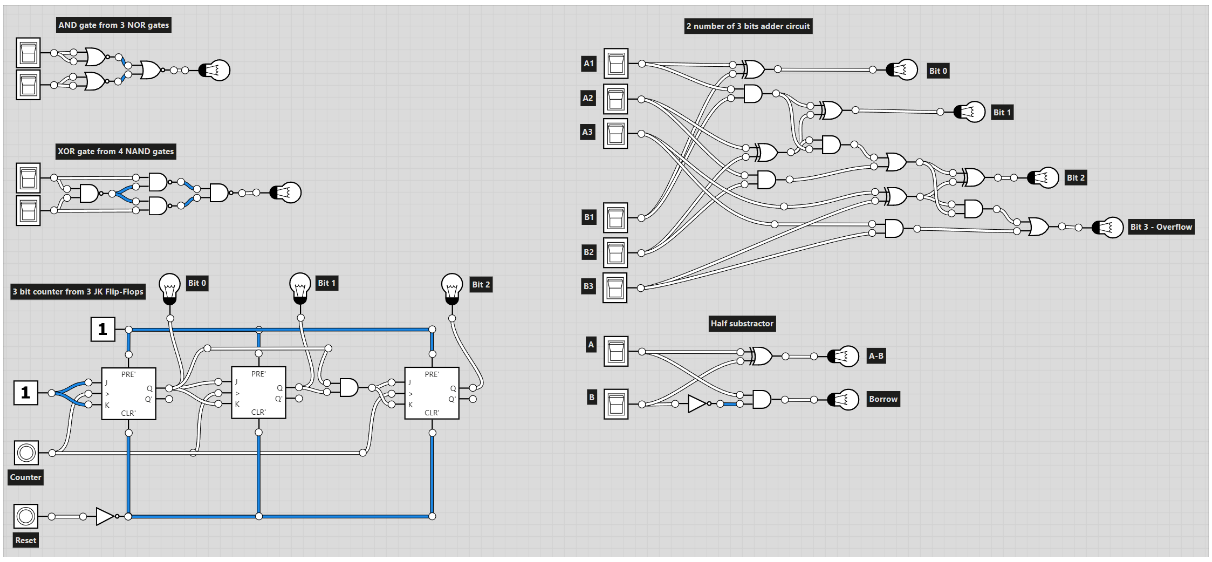
Workshop 1 - Digital Building Blocks

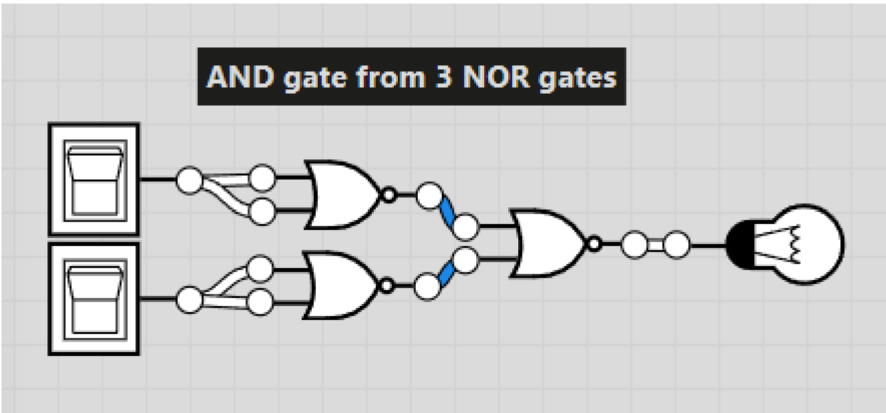
Key Takeaways

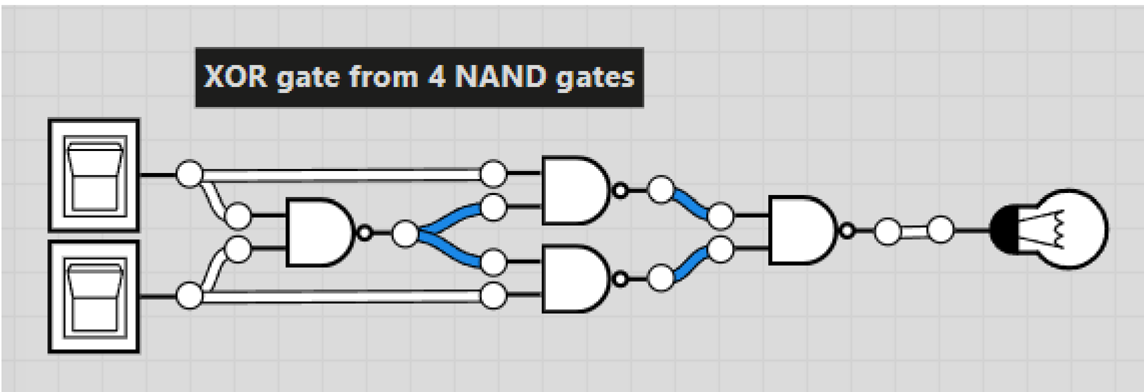
- **Boolean Algebra and Logic Gates:** Learned the fundamental principles of Boolean algebra and how they are used to design and analyze digital circuits. Understood the truth tables, symbols, and functions of basic logic gates (AND, OR, NOT, NAND, NOR, XOR, XNOR).
- **Combinational Logic Design:** Explored how to combine logic gates to create more complex circuits that perform specific functions, such as adders, multiplexers, decoders, and encoders.
- **Digital Circuit Analysis:** Practiced analyzing existing digital circuits to determine their functionality and troubleshoot potential issues.

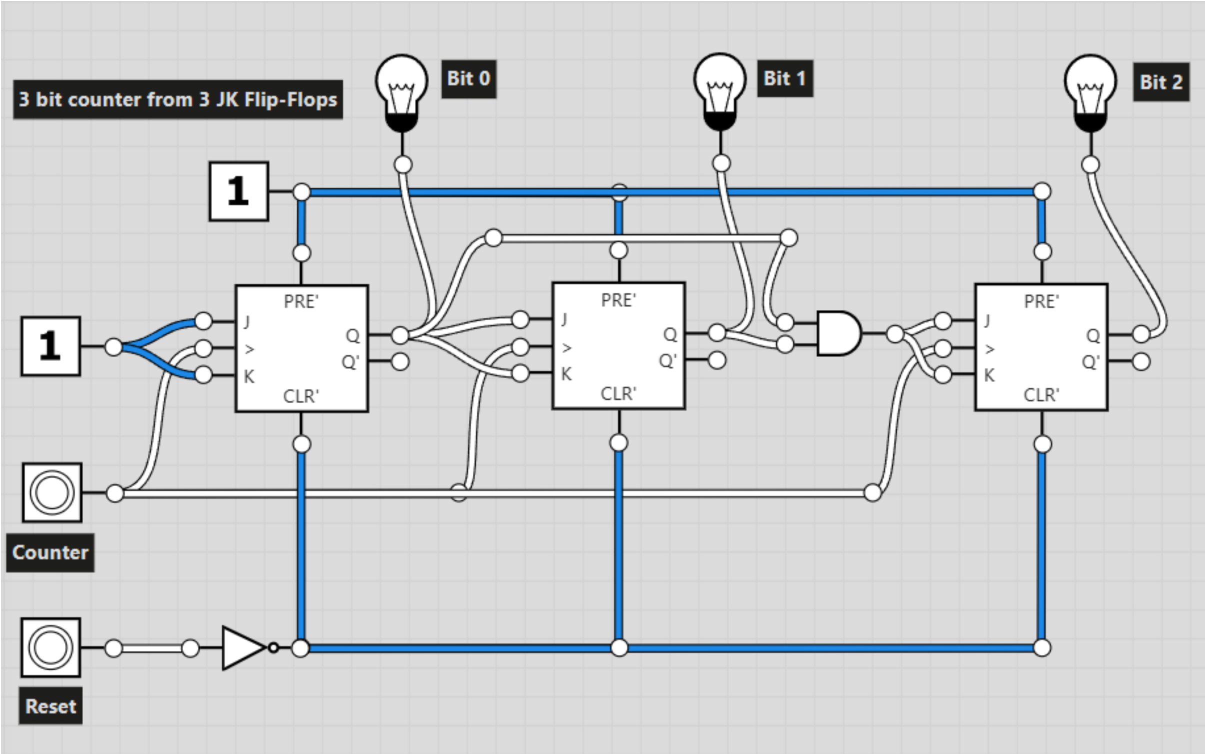
Challenges

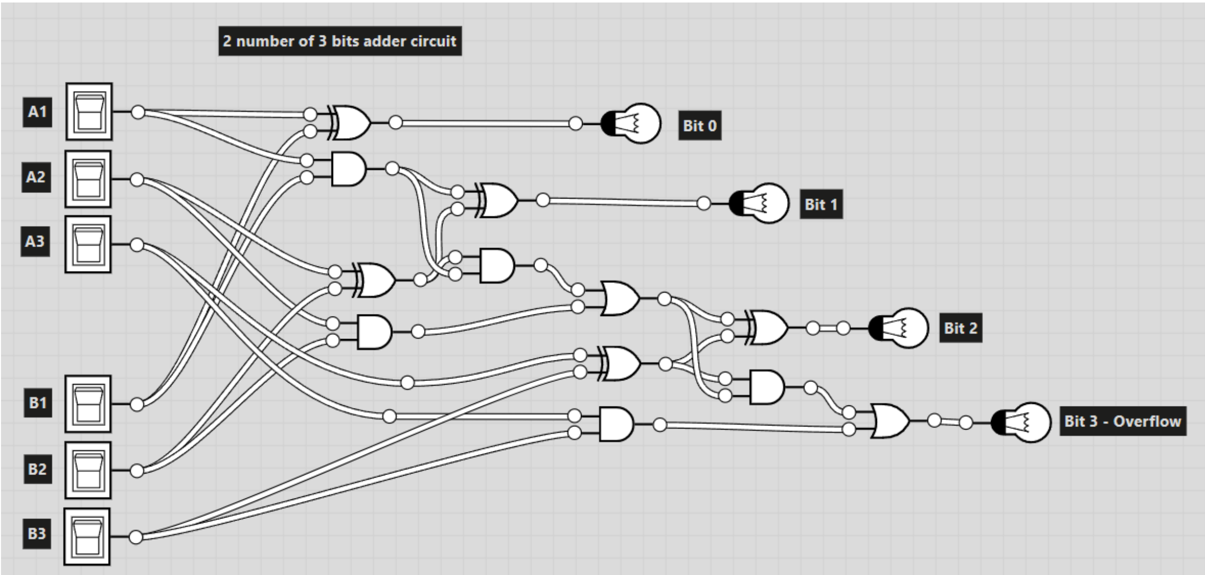
I successfully implemented all the exercises provided in the presentation using the Logic.ly software. In addition I wrote a document going thorough and taking a look at the basic components of digital logic as well as some more complicated arrangements. The document can be found in the dedicated for this workshop directory. This way I was able to better understand how modern digital electronics are made and work.

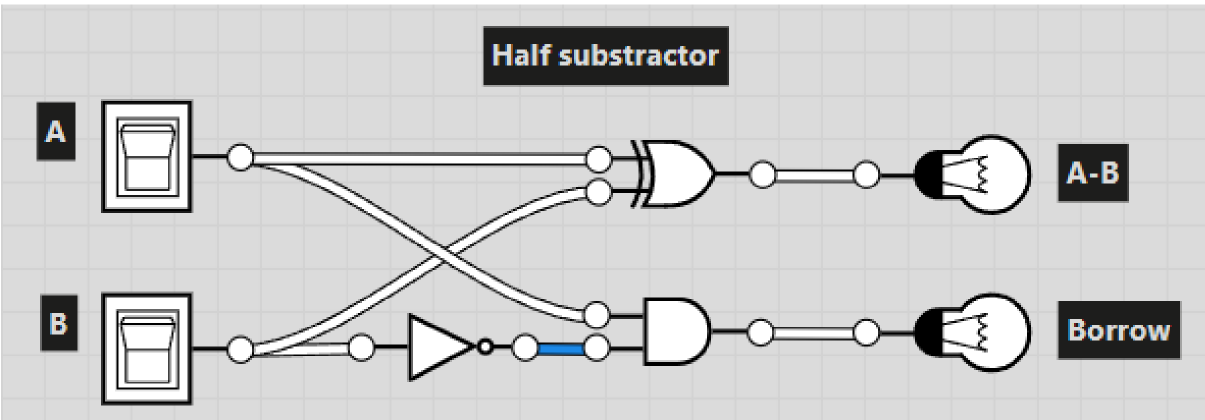

AND gate from 3 NOR gates
XOR gate from 4 NAND gates
3 bit counter from 3 JK Flip-Flops
Bit 0
Bit 1
Bit 2
Counter
Reset
2 number of 3 bits adder circuit
A1
A2
A3
B1
B2
B3
Bit 0
Bit 1
Bit 2
Bit 3 - Overflow
Half substractor
A
B
A-B
Borrow


AND gate from 3 NOR gates


XOR gate from 4 NAND gates


3 bit counter from 3 JK Flip-Flops
Bit 0
Bit 1
Bit 2
1
1
PRE'
J
>
K
Q
Q'
CLR'
Counter
Reset


2 number of 3 bits adder circuit
A1
A2
A3
B1
B2
B3
Bit 0
Bit 1
Bit 2
Bit 3 - Overflow


Half substractor
A
B
A-B
Borrow

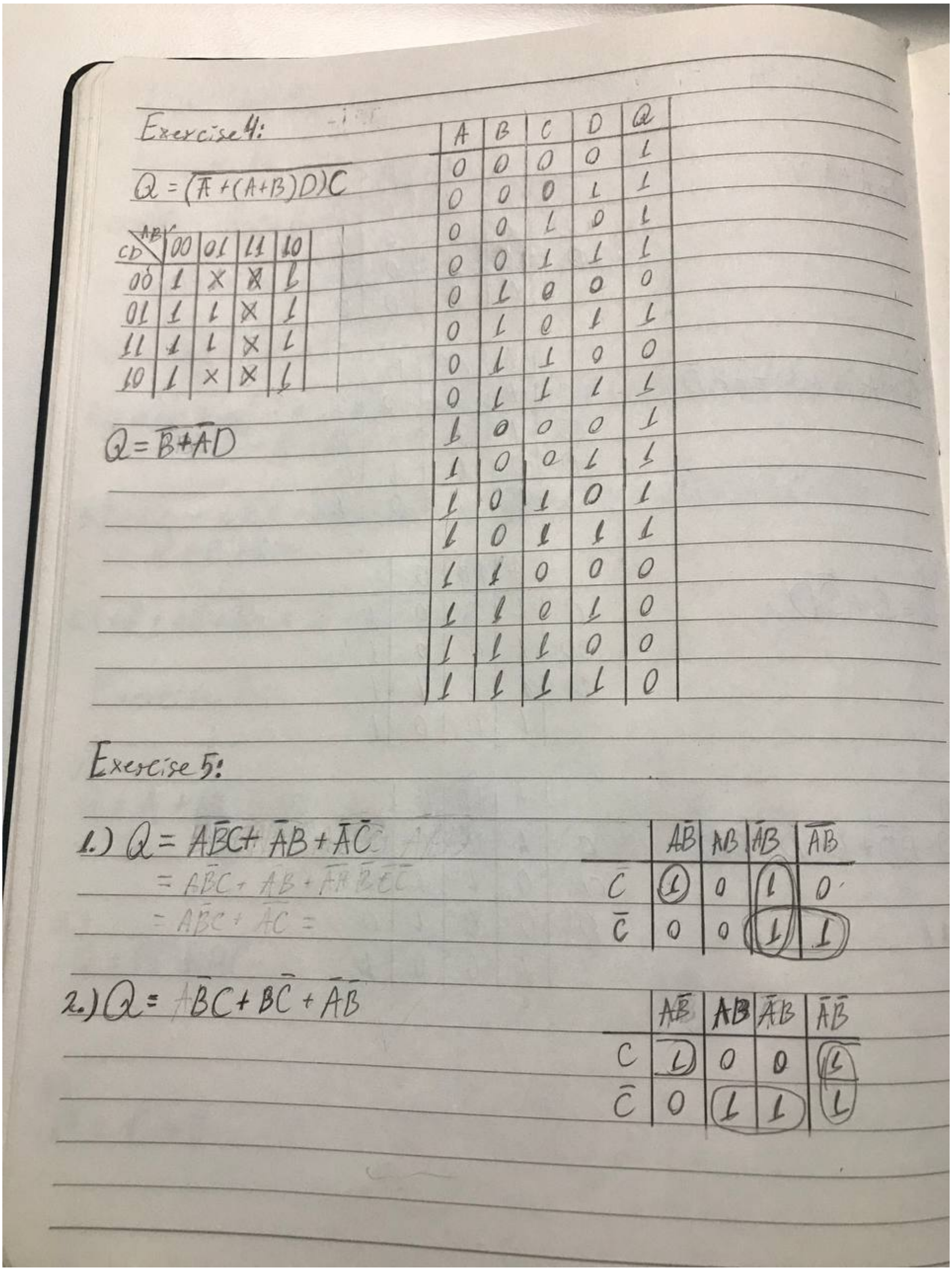
Workshop 2 - Karnaugh Maps

Key Takeaways

- **Karnaugh Map (K-Map) Technique:** Learned how to use K-Maps to simplify Boolean expressions visually, leading to more efficient circuit designs.
- **Don't Care Conditions:** Understood the concept of "don't care" conditions in K-Maps and how they can further optimize circuit designs.

Challenges

I solved all the challenges provided in the presentation despite initially having difficulties working with K-Maps for expressions with more than four variables and sometimes struggling to find the most efficient groupings in K-Maps for maximum simplification. Eventually I managed to get the hang of it and solve the rest of the exercises with ease. Here are some of the example I did the rest can be found in the dedicated for this workshop directory.



Exercise 4:

$Q = \overline{(\overline{A} + (A+B)D)C}$

CD \ AB	00	01	11	10
00	1	X	X	1
01	1	1	X	1
11	1	1	X	1
10	1	X	X	1

$Q = \overline{B} + \overline{A}D$

A	B	C	D	Q
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Exercise 5:

1.) $Q = A\overline{B}C + \overline{A}B + \overline{A}\overline{C}$

$= A\overline{B}C + \overline{A}B + \overline{A}\,\overline{B}\,\overline{C}$

$= A\overline{B}C + \overline{A}C =$

	$A\overline{B}$	AB	$\overline{A}B$	$\overline{A}\overline{B}$
C	1	0	1	0
$\overline{C}$	0	0	1	1

2.) $Q = \overline{A}BC + B\overline{C} + \overline{A}\overline{B}$

	$A\overline{B}$	AB	$\overline{A}B$	$\overline{A}\overline{B}$
C	1	0	0	1
$\overline{C}$	0	1	1	1

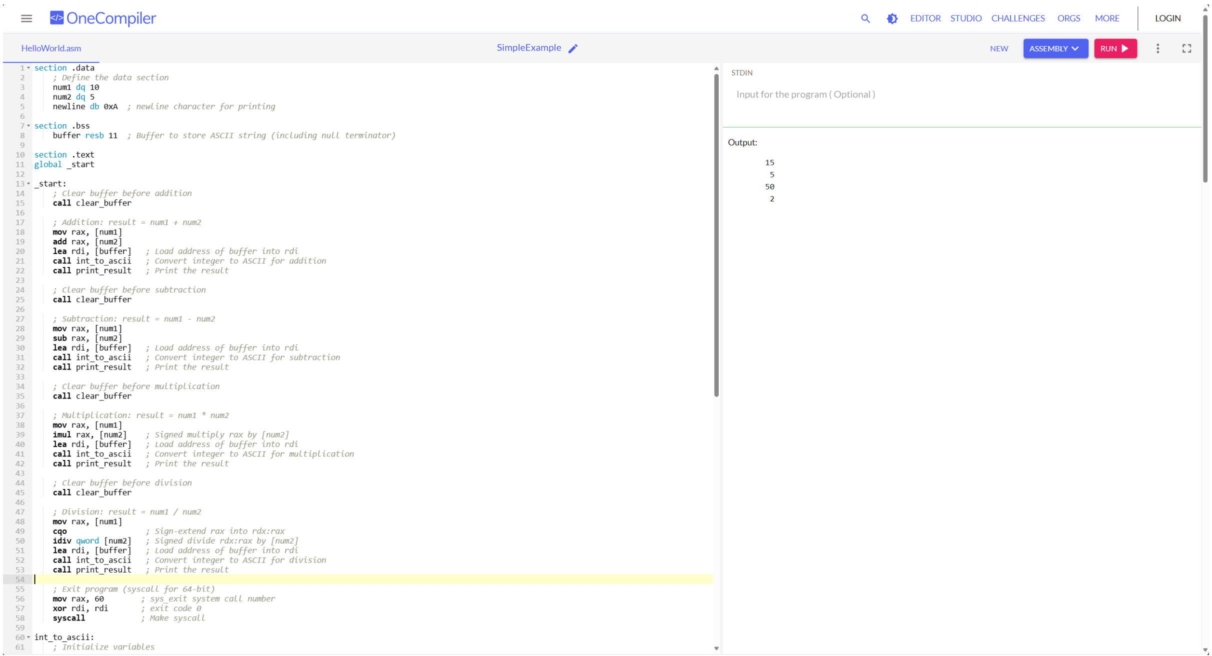
Workshop 3 - Processors

Key Takeaways

- **Processor Architecture:** Studied the basic structure of a microprocessor, including its components like the ALU, control unit, registers, and buses.
- **Instruction Set Architecture (ISA):** Understood the concept of ISA and how it defines the instructions that a processor can execute.
- **Assembly Language Programming:** Learned the basics of assembly language programming, including how to write simple programs that control a processor's operations.

Challenges

The workshop itself did not provide us with any challenges to complete, however I decided to make my own. I was interested by the assembly language and how it work so I make a simple example code that performs addition, subtraction, multiplication and division of 2 number and printing the result on the terminal. Initially I thought it will be easy to make this program but I quickly realized how different and also difficult it is to write code in Assembly than C/C++. Eventually I managed to complete the self made challenge and tested it using the free online platform OneCompiler here are the results.




```
section .data
    ; Define the data section
    num1 dq 10
    num2 dq 5
```

```
    newline db 0xA  ; newline character for printing

section .bss
    buffer resb 11  ; Buffer to store ASCII string (including

section .text
global _start

_start:
    ; Clear buffer before addition
    call clear_buffer

    ; Addition: result = num1 + num2
    mov rax, [num1]
    add rax, [num2]
    lea rdi, [buffer]   ; Load address of buffer into rdi
    call int_to_ascii   ; Convert integer to ASCII for additi
    call print_result   ; Print the result

    ; Clear buffer before subtraction
    call clear_buffer

    ; Subtraction: result = num1 - num2
    mov rax, [num1]
    sub rax, [num2]
    lea rdi, [buffer]   ; Load address of buffer into rdi
    call int_to_ascii   ; Convert integer to ASCII for subtra
    call print_result   ; Print the result

    ; Clear buffer before multiplication
    call clear_buffer

    ; Multiplication: result = num1 * num2
    mov rax, [num1]
    imul rax, [num2]    ; Signed multiply rax by [num2]
    lea rdi, [buffer]   ; Load address of buffer into rdi
    call int_to_ascii   ; Convert integer to ASCII for multip
    call print_result   ; Print the result
```

```
    ; Clear buffer before division
    call clear_buffer

    ; Division: result = num1 / num2
    mov rax, [num1]
    cqo                  ; Sign-extend rax into rdx:rax
    idiv qword [num2]    ; Signed divide rdx:rax by [num2]
    lea rdi, [buffer]    ; Load address of buffer into rdi
    call int_to_ascii    ; Convert integer to ASCII for divisi
    call print_result    ; Print the result

    ; Exit program (syscall for 64-bit)
    mov rax, 60         ; sys_exit system call number
    xor rdi, rdi        ; exit code 0
    syscall             ; Make syscall

int_to_ascii:
    ; Initialize variables
    mov rcx, 10             ; Base 10
    mov rsi, rdi            ; Store the original buffer addres
    add rsi, 10             ; Point to the end of the buffer (
    mov byte [rsi], 0       ; Null-terminate the string

convert_loop:
    dec rsi                 ; Move backwards in the buffer
    xor rdx, rdx            ; Clear rdx for the division
    div rcx                 ; Divide rax by 10
    add dl, '0'             ; Convert remainder to ASCII
    mov [rsi], dl           ; Store ASCII character in buffer

    ; Check if we are done
    test rax, rax           ; Check if rax is 0
    jnz convert_loop        ; If not, continue loop

    ; Adjust the pointer to the start of the string
    mov rdi, rsi
```

```
    ret                     ; Return from the function

clear_buffer:
    xor rdi, rdi        ; Clear rdi
    lea rdi, [buffer]   ; Load address of buffer into rdi
    mov rcx, 11         ; Number of bytes in the buffer
    mov al, 0           ; Value to set (zero)
    rep stosb           ; Store AL (zero) in [RDI], repeat RC

    ret

print_result:
    ; Print the converted string (syscall for 64-bit)
    mov rax, 1         ; sys_write system call number
    mov rdi, 1         ; file descriptor 1 (stdout)
    mov rdx, 10        ; Length of the string to print (adjus
    mov rsi, buffer    ; Pointer to the string
    syscall            ; Make syscall

    ; Print newline after multiplication result
    mov rax, 1         ; sys_write system call number
    mov rdi, 1         ; file descriptor 1 (stdout)
    mov rdx, 1         ; number of bytes to write (newline)
    mov rsi, newline   ; pointer to newline
    syscall            ; Make syscall

    ret
```

Workshop 4 - Soldering

Key Takeaways

- **Soldering Techniques:** Acquired practical skills in soldering, including preparing components, applying heat, and ensuring proper connections.
- **Soldering Safety:** Learned about the importance of safety precautions when working with soldering irons and molten solder.

- **De-soldering and Rework:** Practiced de-soldering techniques to remove and replace faulty components on printed circuit boards (PCBs).

Challenge

I soldered the available MADLAB electronic kit (Christmas tree). However, I did encounter a problem, namely damaging one of the contact pads of the PCB. I attempted to solve the issue by exposing one of the traces and soldering on leg of the component to the trace itself. Unfortunately this did not work and some of the LEDs don’t light up.

Workshop 5 - KiCAD

Key Takeaways

- **Schematic Design:** Learned how to use KiCAD to create electronic schematics, capturing the circuit's design and component connections.
- **PCB Layout:** Explored the PCB layout features of KiCAD, including placing components, routing traces, and generating manufacturing files.
- **Design Rules Check (DRC):** Understood the importance of DRC to ensure that the PCB layout adheres to manufacturing constraints and electrical requirements.

Challenges

Conclusion

Throughout this Digital Techniques course, I gained a strong foundation in the fundamentals of digital electronics. I learned about the building blocks of digital circuits, how to design and analyze them, and how to implement them in physical form. I also acquired practical skills in soldering and PCB design using KiCAD. These skills will be invaluable as I continue my studies and career in electronics and related fields.