ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION
PROJECT 2 REPORT

CRN : 21336

LECTURER : Prof. Dr. Mustafa Ersel Kamaşak

GROUP MEMBERS:

150200097 : Mustafa Can Çalışkan

150200016 : Yusuf Şahin

SPRING 2024

Contents

1 INTRODUCTION

This project, part of the BLG 222E Computer Organization course, focuses on designing and implementing a functional Central Processing Unit (CPU) system. The CPU is capable of executing a comprehensive set of instructions, including various arithmetic and logical operations.

Key components such as decoders, counters, and an arithmetic logic unit (ALU) were developed and integrated to form the complete CPU system. The design process emphasized modularity and hierarchical structuring for scalability and ease of maintenance.

Extensive simulations were conducted to validate the functionality and performance of the CPU. While the system successfully executes all instructions, this report specifically highlights the results for instructions like BRA, ADDS, POP, LSL, INC, NAND, MOVH, and LDR.

This report details the materials and methods used, the results obtained, and the challenges addressed during the project.

2 MATERIALS AND METHODS

2.1 Task Distribution

The task distribution has been carried out as outlined below.

1. The implementation of the ARF, IR, and MUX selects, as well as the Counter and Decoder modules, was carried out by Can Çalışkan.

2. The implementation of the RF and MEM selects, the initial conditions of the computer, the creation of the overall skeleton of the module, and the correction of errors remaining from Project 1 were carried out by Yusuf Şahin.

3. The report was written collaboratively. The tasks and responsibilities for the entire project were shared equally.

2.2 Decoders & Counters

The implementation of the TCounter, TDecoder, and TDecoder modules, which decode the opcode from the given instruction and ensure the sequential execution of operations, are illustrated in Figures 1 and 2.

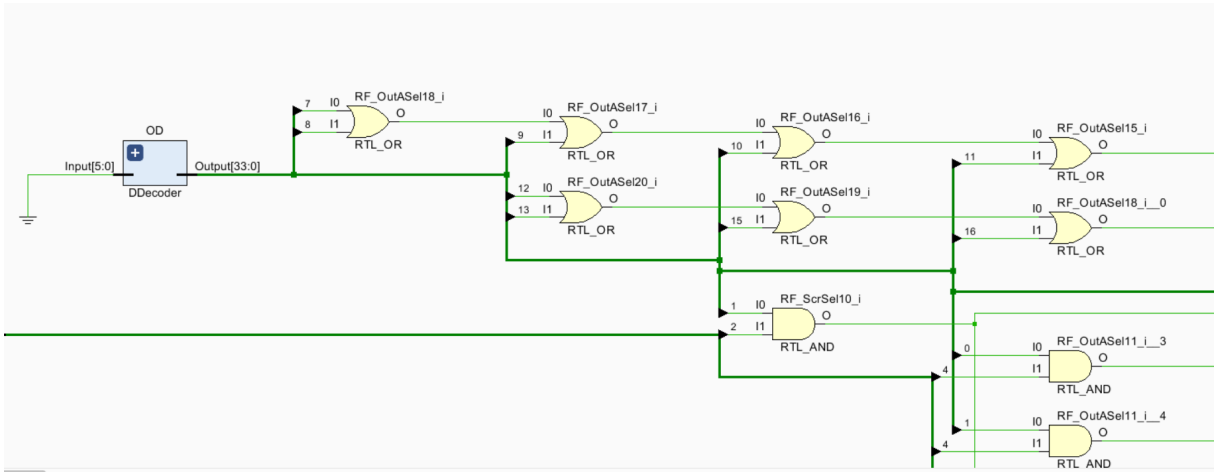

Figure 1: Opcode (D) Decoder RTL Schema

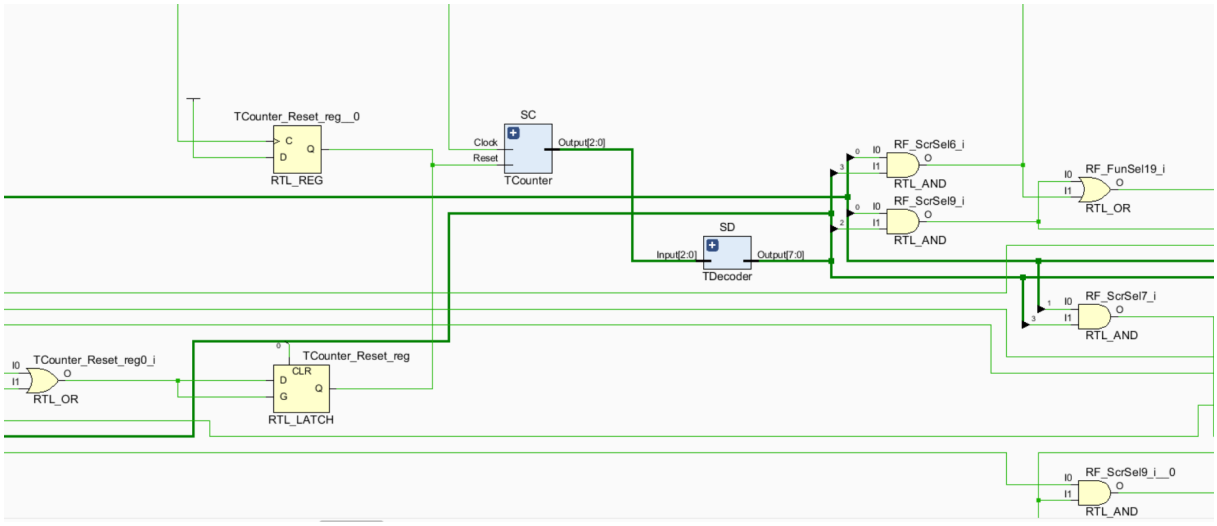

Figure 2: Sequence Counter (T) and T Decoder RTL Schema

2.3 Control Unit

The connection of the select signals obtained from the control unit to the ALU System module is illustrated in Figure 3.

2.4 CPU System

The complete CPU System RTL schematic is shown in Figure 4. On the left side of the figure, the Decoder modules and the T Counter module can be seen, while the ALU System module is visible on the right side.

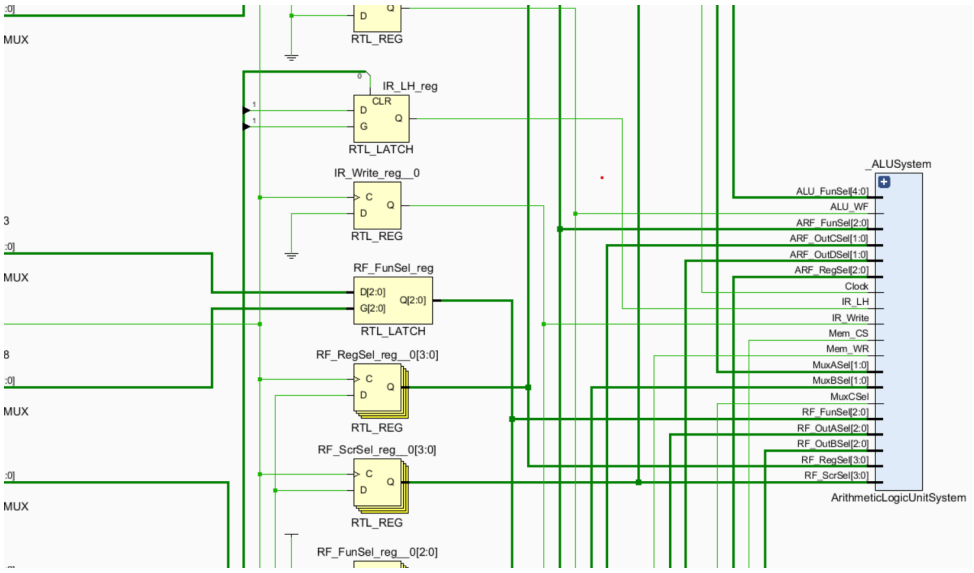

Figure 3: Control Unit RTL Schema

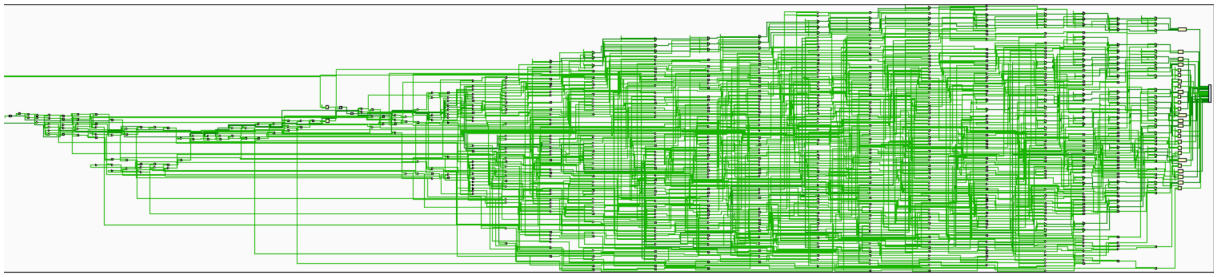

Figure 4: CPU System RTL Schema

3 RESULTS

All instructions were implemented and tested. The results of a selected few can be seen below. Due to the results not fitting into a single figure, they have been numbered as 1, 2, and so on.

To assign initial values to the registers that will be used, the instructions are written in pairs.

3.1 Initial Condition

At the start of the simulation, all select values begin as 'x' before the T[0] time signal. Once the T[0] time signal arrives, the select values transition to their normal states. The initial state can be observed in Figure 5.

```
T:   x
Address Register File: PC:     x, AR:     x, SP:     x
Instruction Register :     x
Register File Registers: R1:     x, R2:     x, R3:     x, R4:     x
Register File Scratch Registers: S1:     x, S2:     x, S3:     x, S4:     x
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   1
Address Register File: PC:     0, AR:     x, SP:     x
Instruction Register :     x
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x
```

Figure 5: Simulation Initial Condition

3.2 BRA Instruction

When the 16-bit 0x0030 instruction is written to RAM.MEM, as expected, the decimal value of 0x30, which is 48, is added to the old value of the PC, which is 2, resulting in 50 being written to the PC. The results can be seen below.

```
Output Values:
T:   1
Address Register File: PC:     0, AR:     x, SP:     x
Instruction Register :     x
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   2
Address Register File: PC:     1, AR:     x, SP:     x
Instruction Register :     X
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0


Output Values:
T:   4
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :    48
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0
```

Figure 6: BRA Instruction 1

```
Output Values:
T:   8
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :    48
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:    48, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:    48


Output Values:
T:  16
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :    48
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:    48, S2:     2, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:    50


Output Values:
T:   1
Address Register File: PC:    50, AR:     x, SP:     x
Instruction Register :    48
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0
```

Figure 7: BRA Instruction 2

3.3 ADDS and POP Instructions

In the ADDS and POP instructions, initially, with the 0x6680 instruction, the PC is added to itself and written to the SP register. Subsequently, the POP operation is performed with the 0x0C00 instruction from memory.

Using the values 0x00 and 0x12 located in the 4th and 5th words of memory, the R1 register is populated as follows: R1(7-0) <= 0x00 and R1(15-8) <= 0x12.

In the final step, the SP is incremented to point to the next empty address and is set to 6. The results can be seen below.

```
Output Values:
T:   1
Address Register File: PC:     0, AR:     x, SP:     x
Instruction Register :     x
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   2
Address Register File: PC:     1, AR:     x, SP:     x
Instruction Register :     X
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0


Output Values:
T:   4
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register : 26240
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0


Output Values:
T:   8
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register : 26240
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     2, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     2
```

Figure 8: ADDS Instruction 1

```
Output Values:
T:  16
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register : 26240
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     2, S2:     2, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     4


Output Values:
T:   1
Address Register File: PC:     2, AR:     x, SP:     4
Instruction Register : 26240
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     2, S2:     2, S3:     0, S4:     0
ALU Flags: Z: 0, N: 0, C: 0, O: 0
ALU Result: ALUOut:     4
```

Figure 9: ADDS Instruction 2

```
Output Values:
T:   2
Address Register File: PC:     3, AR:     x, SP:     4
Instruction Register : 26112
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, N: 0, C: 0, O: 0
ALU Result: ALUOut:     0


Output Values:
T:   4
Address Register File: PC:     4, AR:     x, SP:     4
Instruction Register :  3072
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, N: 0, C: 0, O: 0
ALU Result: ALUOut:     0


Output Values:
T:   8
Address Register File: PC:     4, AR:     x, SP:     5
Instruction Register :  3072
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, N: 0, C: 0, O: 0
ALU Result: ALUOut:     0


Output Values:
T:   1
Address Register File: PC:     4, AR:     x, SP:     6
Instruction Register :  3072
Register File Registers: R1:  4608, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, N: 0, C: 0, O: 0
ALU Result: ALUOut:     0
```

Figure 10: POP Instruction

3.4 LSL and INC Instructions

First, with the 0x1D00 instruction, the value in the PC register is left-shifted (LSL) and written to the R1 register. Then, with the 0x1560 instruction, the value in R1 is written to R2 and incremented. The results can be seen below.

```
Output Values:
T:   1
Address Register File: PC:     0, AR:     x, SP:     x
Instruction Register :     x
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   2
Address Register File: PC:     1, AR:     x, SP:     x
Instruction Register :     X
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0


Output Values:
T:   4
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :  7424
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0
```

Figure 11: LSL Instruction 1

```
Output Values:
T:   8
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :  7424
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     2, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     4


Output Values:
T:   1
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :  7424
Register File Registers: R1:     4, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     4, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     8
```

Figure 12: LSL Instruction 2

```
Output Values:
T:   2
Address Register File: PC:     3, AR:     x, SP:     x
Instruction Register :  7520
Register File Registers: R1:     4, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0


Output Values:
T:   4
Address Register File: PC:     4, AR:     x, SP:     x
Instruction Register :  5472
Register File Registers: R1:     4, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     4


Output Values:
T:   8
Address Register File: PC:     4, AR:     x, SP:     x
Instruction Register :  5472
Register File Registers: R1:     4, R2:     4, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0
```

Figure 13: INC Instruction 1

```
Output Values:
T:   1
Address Register File: PC:     4, AR:     x, SP:     x
Instruction Register :  5472
Register File Registers: R1:     4, R2:     5, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0
```

Figure 14: INC Instruction 2

3.5 NAND and MOVH Instructions

With the 0x440F instruction, the operation R1(15-8) <= IR(7-0) is performed. Subsequently, the value in the R1 register is NANDed with the value 0x0 in R2 and written to the R3 register. The results can be seen below.

```
Output Values:
T:   1
Address Register File: PC:     0, AR:     x, SP:     x
Instruction Register :     x
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   2
Address Register File: PC:     1, AR:     x, SP:     x
Instruction Register :     X
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0


Output Values:
T:   4
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register : 17423
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0


Output Values:
T:   1
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register : 17423
Register File Registers: R1:  3840, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0
```

Figure 15: MOVH Instruction

```
Output Values:
T:   2
Address Register File: PC:     3, AR:     x, SP:     x
Instruction Register : 17580
Register File Registers: R1:  3840, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0


Output Values:
T:   4
Address Register File: PC:     4, AR:     x, SP:     x
Instruction Register : 16812
Register File Registers: R1:  3840, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0


Output Values:
T:   8
Address Register File: PC:     4, AR:     x, SP:     x
Instruction Register : 16812
Register File Registers: R1:  3840, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:  3840


Output Values:
T:  16
Address Register File: PC:     4, AR:     x, SP:     x
Instruction Register : 16812
Register File Registers: R1:  3840, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:  3840, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut: 65535
```

Figure 16: NAND Instruction 1

```
Output Values:
T:   1
Address Register File: PC:     4, AR:     x, SP:     x
Instruction Register : 16812
Register File Registers: R1:  3840, R2:     0, R3: 65535, R4:     0
Register File Scratch Registers: S1:     0, S2:  3840, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut: 65535
```

Figure 17: NAND Instruction 2

3.6 MOVS and LDR Instructions

First, with the `0x60E0` instruction (MOVS), the value 0 in the R1 register is moved to AR. Then, with the `0x4800` (LDR) instruction, $R1(7\text{-}0) \leftarrow M[AR]$ and $R1(15\text{-}8) \leftarrow M[AR+1]$ are written. Subsequently, the AR register is decremented to return to its original value.

```
Output Values:
T:   1
Address Register File: PC:     0, AR:     x, SP:     x
Instruction Register :     x
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   2
Address Register File: PC:     1, AR:     x, SP:     x
Instruction Register :     X
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0


Output Values:
T:   4
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register : 24800
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0
```

Figure 18: MOVS Instruction 1

```
Output Values:
T:   1
Address Register File: PC:     2, AR:     0, SP:     x
Instruction Register : 24800
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0
```

Figure 19: MOVS Instruction 2

```
Output Values:
T:   2
Address Register File: PC:     3, AR:     0, SP:     x
Instruction Register : 24576
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0


Output Values:
T:   4
Address Register File: PC:     4, AR:     0, SP:     x
Instruction Register : 18432
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0


Output Values:
T:   8
Address Register File: PC:     4, AR:     1, SP:     x
Instruction Register : 18432
Register File Registers: R1:   224, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0
```

Figure 20: LDR Instruction 1

```
Output Values:
T:   1
Address Register File: PC:     4, AR:     0, SP:     x
Instruction Register : 18432
Register File Registers: R1: 24800, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     0
```

Figure 21: LDR Instruction 2

4 DISCUSSION

In the project, it was necessary to make certain assumptions in cases where some situations were unclear, as outlined below.

1. In POP and PSH operations, as demonstrated in the POP example above, the SP register is designed to point to the first empty slot by nature. Therefore, in the implementation, the SP register was incremented twice during the POP operation and decremented twice during the PSH operation.

2. It was assumed that the IMMEDIATE and VALUE symbols refer to IR(7-0).

3. For instructions containing 'S' such as ADDS and SUBS, it was assumed that the flags would change based on the state of 'S' (1 if it changes, 0 if it doesn't). For other operations, it was assumed that no flags would change regardless of the state of 'S' (because the instruction name does not contain 'S').

4. In the MOVH and MOVL operations, it was assumed that DSTREG would always be one of the R registers in RF.

5. Since each memory word is 8 bits and the registers are 16 bits, operations involving memory refer to memory twice: first to write/read bits 7-0 of the target/source register, and then to write/read bits 15-8 of the target/source register. For example, in the LDR instruction, this is done first as Rx(7-0) <= M[AR], then as Rx(15-8) <= M[AR + 1]. Here, the value of AR remains constant.

6. To make the outputs of certain instructions resemble the sample outputs, operations that were not directly necessary for execution were performed. (For example, clearing the S registers in RF when the BRA instruction is completed.) These operations, albeit rarely, extended the duration of the instruction's execution time by T + 1.

5 CONCLUSION

In this project, we successfully designed and implemented a functional Central Processing Unit (CPU) system capable of executing a predefined set of instructions and performing various arithmetic and logical operations. The development process involved creating key components such as decoders, counters, and an arithmetic logic unit (ALU) system, which were then integrated to form the complete `CPUSystem` module.

Throughout the project, we adhered to best practices in digital design, ensuring modularity and hierarchical structuring to maintain scalability and ease of maintenance. The

CPU system was rigorously tested through extensive simulation to validate its functionality and performance.

The results demonstrated that the CPU system could accurately execute instructions such as BRA, ADDS, POP, LSL, INC, NAND, MOVH, and LDR, meeting the project objectives. Specific challenges, such as making assumptions about the behavior of the SP register in POP and PSH operations and the handling of immediate values and flags, were addressed through careful design decisions.

In conclusion, this project provided a valuable learning experience in digital design and CPU architecture, reinforcing our understanding of the principles and practices necessary for developing complex digital systems. The successful implementation and testing of the CPU system underscore our ability to apply theoretical knowledge to practical design tasks, paving the way for further exploration and innovation in the field of computer engineering.

REFERENCES