ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION
PROJECT 2 REPORT

CRN	:	21335
LECTURER	:	Prof. Dr. Deniz Turgay Altılar

GROUP MEMBERS:

150210077	:	Emre Yamaç
150210088	:	Alper Tutum

SPRING 2024

Contents

1 INTRODUCTION

In the second project of the computer organization course, we took on the task of designing a basic Central Processing Unit (CPU). The main purpose requested from us in this project was to carry out the instructions given to us by applying the necessary operations at the appropriate time, by using the Address Register File, Register File, Instruction Register, ALU, Memory and necessary Registers that we created in the previous project. After following the instructions, we simulated our instructions and compared the situations as a result of the appropriate operations in the RAM.mem file.

2 MATERIALS AND METHODS

In this section, the implementation details of the CPU System are explained.

2.1 Sequence Counter

A sequence counter for the CPU is implemented by creating a 3-bit register 'SC' and by initializing its value to 0. SC is incremented with each negative edge of the clock and the next time signal is obtained by decoding SC to an 8 bit 'T' register.

SC	T
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000

Table 1: Decoding of Sequence Counter[1]

The sequence counter counts from 0 to 7 as long as it is not cleared by two particular signals. One of the signals is 'Reset'. It is an external signal which stops the sequence counter when its value is 0. The other signal comes from the 'CLR' flip-flop which is set when an instruction reaches its end.

2.2 Arithmetic Logic Unit System

The Arithmetic Logic Unit System implemented in the previous project is used without making any changes to the module definition. To summarize the ALU System consists of an ALU, Register File, Address Register File, Instruction Register, Memory Unit, and three multiplexers.

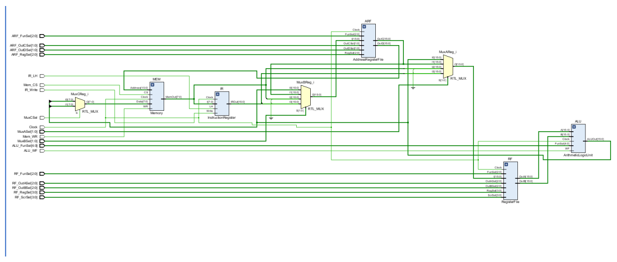


Figure 1: Schematic of ALU System[1]

Every select input of the module is connected to reg variables in the CPU for controlling the flow of data in the system. The data output of the Instruction Register inside the ALU System is assigned to a variable 'IR' since it will later be used for the fetch decode phase. Every register in the Register File and Address Register File is initialized to 16-bit 0 except for SP in ARF. The Stack Pointer starts from 255 which is the bottom address of the given RAM.

2.3 Fetch and Decode

The fetch and decode procedure is designed to happen in two clock cycles. This is because the memory unit holds 8 bit data. In the first clock cycle, the ram data that the Program Counter (PC) is pointing to is written on the LSB 8 bits of the Instruction Register and PC is incremented by one. In the second clock cycle the memory data is written to MSB 8 bits of IR and PC is incremented again so that it points to the next instruction.

The instruction format of the given design is explained below:

OPCODE (6-bit)	Rsel (2-bit)	Address (8-bit)

Table 2: Memory Reference Instruction Format[1]

Memory reference instructions select a general purpose register from RF and use it for reading or writing. Some instructions don't use the selected register at all.

Rsel	Selected Register
00	R1
01	R2
10	R3
11	R4

Table 3: Register select of memory reference instructions[1]

OPCODE (6-bit)	S	DSTREG	SREG1	SREG2

Table 4: Register Reference Instruction Format[1]

DSTREG/SREG1/2	Selected Register
000	PC
001	PC
010	SP
011	AR
100	R1
101	R2
110	R3
111	R4

Table 5: Register select of register reference instructions[1]

In this project's design, a flip-flop 'M' is defined for distinguishing between memory reference instructions and non-memory reference instructions in the decoding phase. The decoding of IR happens during every cycle after the first two.

If memory reference: Either OutASel or RegSel of RF should be set depending on the instruction. In the proposed design, Mem WR is checked in the decode block in order to decide if a memory read or write operation is taking place. If the operation is a memory write, OutASel is set to give the selected register into the ALU and ALU FunSel is set to directly feed the contents of the selected register to its output which is connected to Memory via a MUX. If the operation is a memory read, the contents of the register is likely to change so RegSel of RF is set depending on the Rsel bits of IR before going to the execution phase. If the operation is a memory read operation but the Rsel bits are unnecessary, the RegSel is blocked exclusively inside the definition of the instruction.

If register reference: DSTREG bits of IR are decoded and RegSel of RF and ARF are set accordingly. SREG1 and SREG2 bits are decoded to select what is fed into OutA and

OutB of RF respectively. Additionally, an S bit exists in this type of instruction. It is directly connected to the WF input of ALU. If 1, it means the flags of ALU can change.

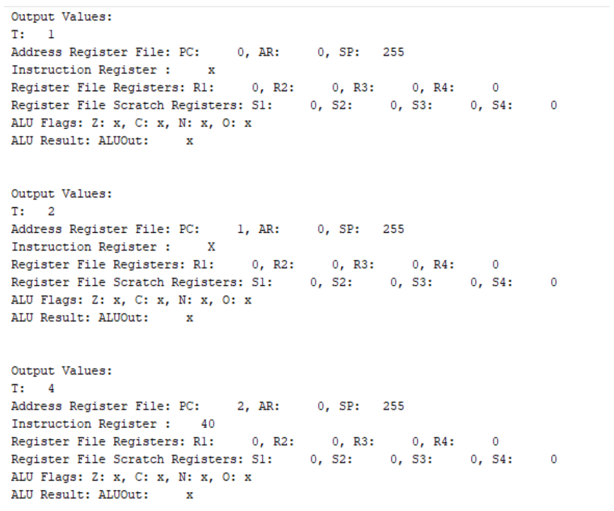
2.4 Execution of Instructions

The OPCODE part from IR was read and a case was written for every defined instruction. Each case was then divided into its clock signals. Inside the cases, every control signal that allows the required register transfer is set. Since the decoding part deals with choosing which registers to write or read, there is no need for extra cases inside of the OPCODE cases.

3 RESULTS

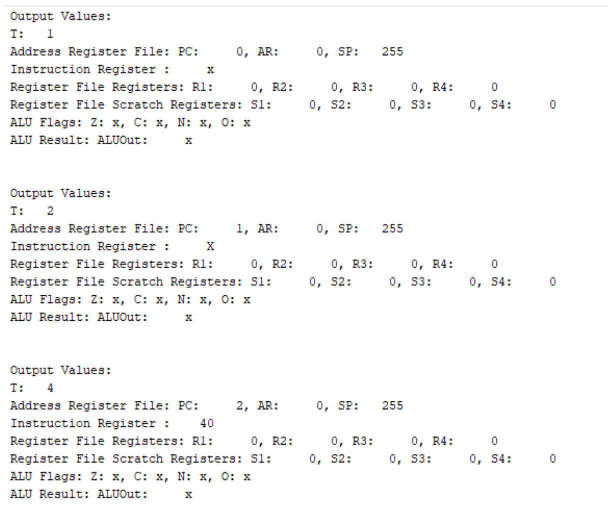
The results for some instructions are given below. The instructions are chosen from the example outputs given to us. As shown below, the outputs we received match the example outputs.

3.1 Fetch Decode

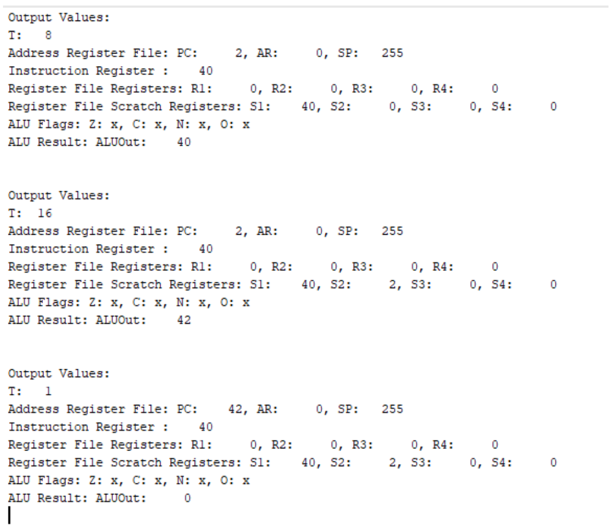


```
Output Values:
T:   1
Address Register File: PC:     0, AR:     0, SP:   255
Instruction Register :     x
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   2
Address Register File: PC:     1, AR:     0, SP:   255
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, C: x, N: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   4
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :    40
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, C: x, N: x, O: x
ALU Result: ALUOut:     x
```

3.2 BRA

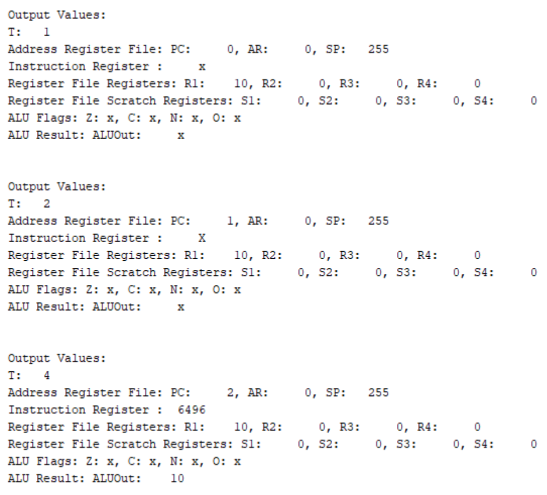


```
Output Values:
T:   1
Address Register File: PC:     0, AR:     0, SP:   255
Instruction Register :     x
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   2
Address Register File: PC:     1, AR:     0, SP:   255
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, C: x, N: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   4
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :    40
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, C: x, N: x, O: x
ALU Result: ALUOut:     x
```

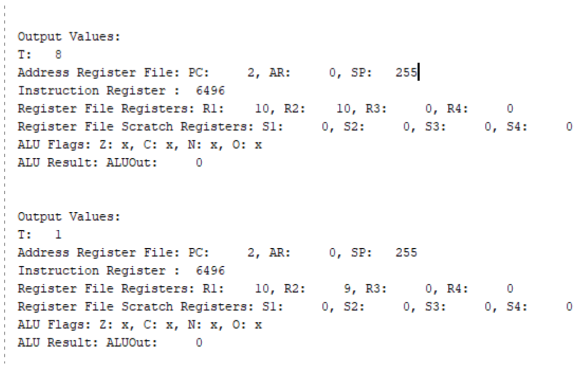


```
Output Values:
T:   8
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :    40
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:    40, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:    40


Output Values:
T:  16
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :    40
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:    40, S2:     2, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:    42


Output Values:
T:   1
Address Register File: PC:    42, AR:     0, SP:   255
Instruction Register :    40
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:    40, S2:     2, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     0
```

3.3 DEC

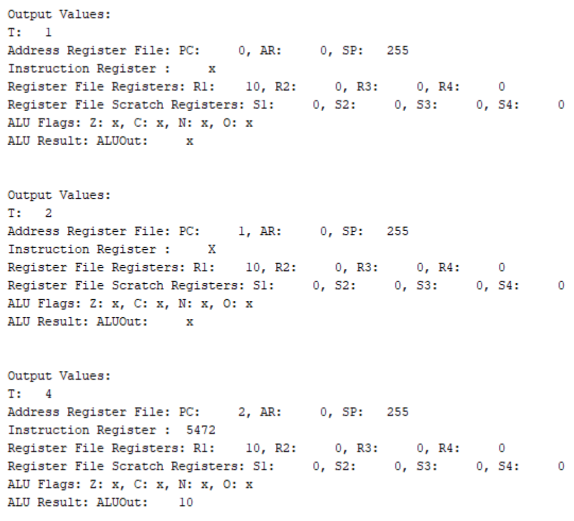


```
Output Values:
T:   1
Address Register File: PC:     0, AR:     0, SP:   255
Instruction Register :     x
Register File Registers: R1:    10, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   2
Address Register File: PC:     1, AR:     0, SP:   255
Instruction Register :     X
Register File Registers: R1:    10, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   4
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :  6496
Register File Registers: R1:    10, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:    10
```

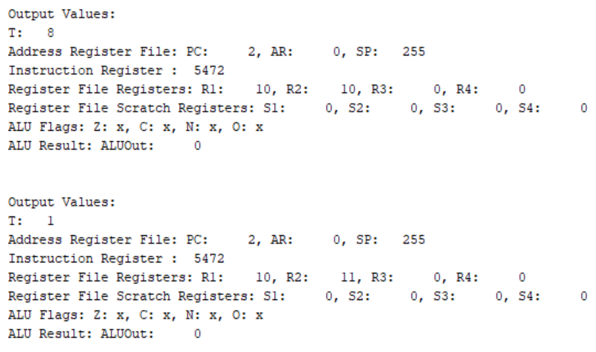


```
Output Values:
T:   8
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :  6496
Register File Registers: R1:    10, R2:    10, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     0


Output Values:
T:   1
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :  6496
Register File Registers: R1:    10, R2:     9, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     0
```

3.4 INC

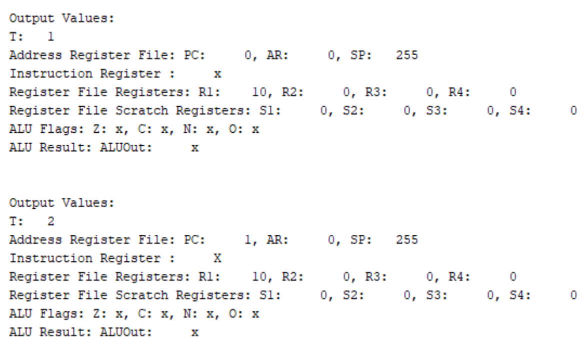


```
Output Values:
T:   1
Address Register File: PC:     0, AR:     0, SP:   255
Instruction Register :     x
Register File Registers: R1:    10, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   2
Address Register File: PC:     1, AR:     0, SP:   255
Instruction Register :     X
Register File Registers: R1:    10, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   4
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :  5472
Register File Registers: R1:    10, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:    10
```

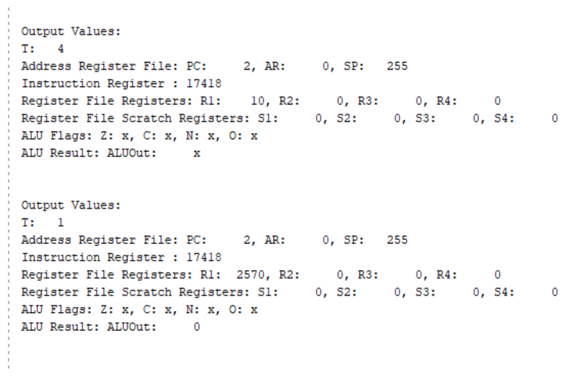


```
Output Values:
T:   8
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :  5472
Register File Registers: R1:    10, R2:    10, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     0


Output Values:
T:   1
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :  5472
Register File Registers: R1:    10, R2:    11, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     0
```

3.5 MOVH

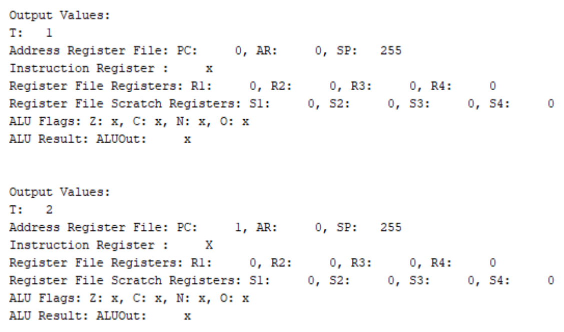


```
Output Values:
T:   1
Address Register File: PC:     0, AR:     0, SP:   255
Instruction Register :     x
Register File Registers: R1:    10, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   2
Address Register File: PC:     1, AR:     0, SP:   255
Instruction Register :     X
Register File Registers: R1:    10, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     x
```

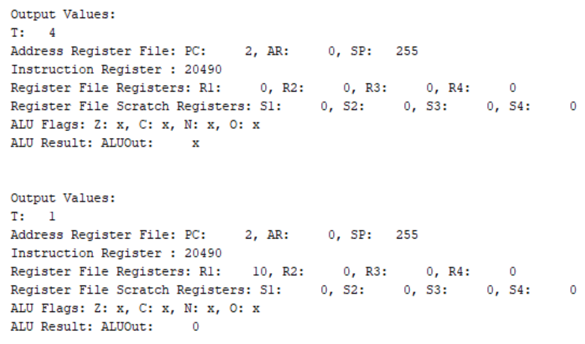


```
Output Values:
T:   4
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register : 17418
Register File Registers: R1:    10, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   1
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register : 17418
Register File Registers: R1:  2570, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     0
```

3.6 MOVL

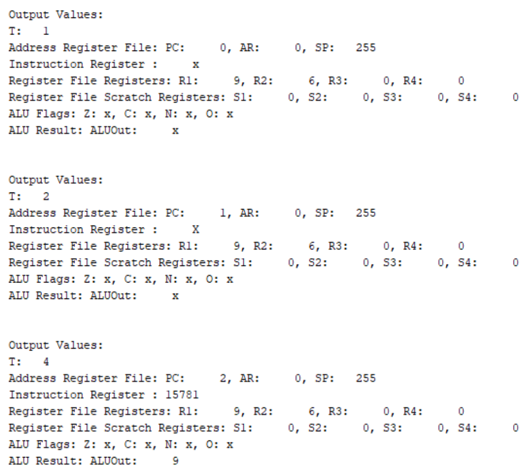


```
Output Values:
T:   1
Address Register File: PC:     0, AR:     0, SP:   255
Instruction Register :     x
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   2
Address Register File: PC:     1, AR:     0, SP:   255
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, C: x, N: x, O: x
ALU Result: ALUOut:     x
```

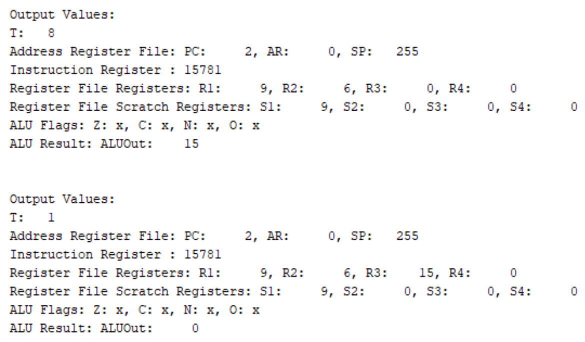


```
Output Values:
T:   4
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register : 20490
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   1
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register : 20490
Register File Registers: R1:    10, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     0
```

3.7 XOR



```
Output Values:
T:   1
Address Register File: PC:     0, AR:     0, SP:   255
Instruction Register :     x
Register File Registers: R1:     9, R2:     6, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   2
Address Register File: PC:     1, AR:     0, SP:   255
Instruction Register :     X
Register File Registers: R1:     9, R2:     6, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   4
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register : 15781
Register File Registers: R1:     9, R2:     6, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     9
```



```
Output Values:
T:   8
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register : 15781
Register File Registers: R1:     9, R2:     6, R3:     0, R4:     0
Register File Scratch Registers: S1:     9, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:    15


Output Values:
T:   1
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register : 15781
Register File Registers: R1:     9, R2:     6, R3:    15, R4:     0
Register File Scratch Registers: S1:     9, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, C: x, N: x, O: x
ALU Result: ALUOut:     0
```

4 DISCUSSION

In this project, we successfully managed to implement our own CPU design using the ALU System we created before. Our first step was to solve the remaining issues in the previous project so that the errors aren't carried into this project. Then we designed the CPU on paper and noted every single register transfer in each time cycle of every single instruction. This made things a lot easier for us. By looking at the paper we wrote our verilog code for each instruction. We specified every control for every operation and it took great time. Since there was a big chunk of code we faced lots of problems with our coding and debugged to solve the errors that didn't let us compile. Then we tested fetch decode and every instruction given in the pdf one by one. As a result we had all of the instructions working as expected, at least on their own. However we faced some issues when using the instructions in sequence. The example program worked only partially, the first iteration of the program loop went smoothly but after that it was broken. The problem is most likely with the ram.mem file since we didn't have much time for testing the example program. At least that's what we assume since the first iteration of the loop goes perfectly and then different operations take place in the second iteration. As a conclusion, the biggest difficulty in this project for us was not allocating enough time for testing.

5 CONCLUSION

To summarize, in our project, we first wrote on a piece of paper which actions the instructions would perform according to time, and drafted the necessary control signals by drawing it to the template of Part 4 we made in the last project. Afterwards, we defined the necessary cables and registers in Verilog and wrote the process of uploading to IR. We defined the opcodes according to the time cycle sketch we created, and after controlling the necessary signals, we finally simulated and compared our answers with the example outputs.

REFERENCES

[1] Emre Yamaç Alper Tutum. Computer organization. *ITU Computer Engineering*, 2(1):1–9, May 2024.