ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION
PROJECT 1 REPORT

CRN : 21335

LECTURER : Prof. Dr. Deniz Turgay Altılar

GROUP MEMBERS:

150220079 : AHMET ENES ÇİĞDEM

SPRING 2024

Contents

1 INTRODUCTION

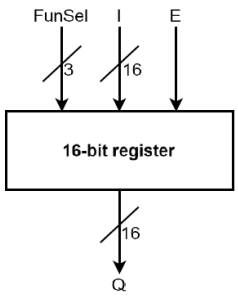
In our computer organization project, we constructed a basic computer system from fundamental components like the Instruction Register (IR), Register File, Address Register, and Arithmetic Logic Unit (ALU). By interconnecting these elements and integrating them with memory, we created a functional computational unit.

2 MATERIALS AND METHODS

We built all hardware part of this project by using Verilgo HDL via Xilinx Vivado.

2.1 Part 1

In this part, We implemented a basic register. This register has 8 functionalities that are controlled by 3-bit control signals (FunSel) and an enable input (E). We will use this register in other part of this project.



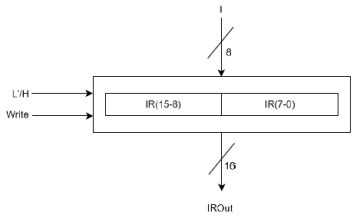
E	FunSel	Q^+
0	ϕ	Q (Retain value)
1	000	Q-1 (Decrement)
1	001	Q+1 (Increment)
1	010	I (Load)
1	011	0 (Clear)
1	100	Q (15-8) ← Clear, Q (7-0) ← I (7-0) (Write Low)
1	101	Q (7-0) ← I (7-0) (Only Write Low)
1	110	Q (15-8) ← I (7-0) (Only Write High)
1	111	Q (15-8) ← Sign Extend (I (7)) Q (7-0) ← I (7-0) (Write Low)

Figure 1: Function table for part 1 register.

2.2 Part 2

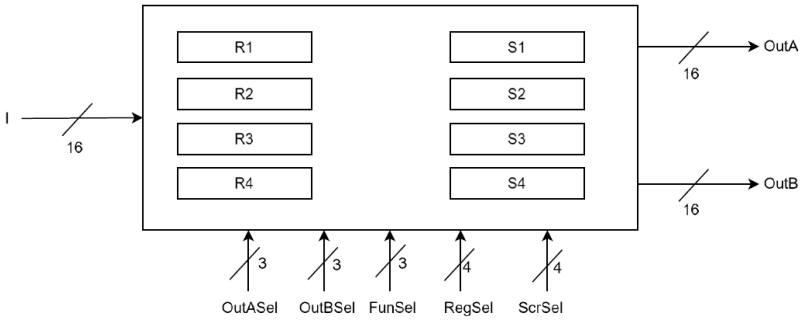
(Part-2a)In this part, we designed a 16-bit IR register whose block diagram and function table are given in Figure 2. This register can store 16-bit binary data. However, the input of this register file is only 8 bits. Therefore, using the 8-bit input bus, you can load either the lower (bits 7-0) or higher (bits 15-8) half. This decision is made by the L'H signal.

(Part-2b)In this part, we design the system shown in Figure 3 which consists of four 16-bit general purpose registers: R1, R2, R3, R4, and four 16-bit scratch registers: S1, S2, S3, and S4. The details of inputs and outputs are as follows.



L'H	Write	IR+
ϕ	0	IR (retain value)
0	1	IR (7-0) ← I (Load LSB)
1	1	IR (15-8) ← I (Load MSB)

Figure 2: Function table for instruction register.



OutASel and OutBSel are used to feed to output lines OutA and OutB, respectively. 16 bits of selected registers are output to OutA and OutB. Table 1 shows the selection of output registers based on the OutASel and OutBSel control inputs.

OutASel	OutA
000	R1
001	R2
010	R3
011	R4
100	S1
101	S2
110	S3
111	S4

OutBSel	OutB
000	R1
001	R2
010	R3
011	R4
100	S1
101	S2
110	S3
111	S4

RegSel and ScrSel are 4-bit signals that select the registers to apply the function that is determined by the 3-bit FunSel signal which is given in Table 2. The selected registers by RegSel and ScrSel are shown in Tables 3 and 4, respectively.

Table 3: RegSel Control Input

RegSel	Enable General Purpose Registers	RegSel	Enable General Purpose Registers
0000	All general purpose registers are enabled. (Function selected by FunSel will be applied to R1, R2, R3 and R4.)	1000	R2, R3, and R4 are enabled. (Function selected by FunSel will be applied to R2, R3, and R4.)
0001	R1, R2 and R3 are enabled. (Function selected by FunSel will be applied to R1, R2, and R3.)	1001	R2 and R3 are enabled. (Function selected by FunSel will be applied to R2 and R3.)
0010	R1, R2, and R4 are enabled. (Function selected by FunSel will be applied to R1, R2, and R4.)	1010	R2 and R4 are enabled. (Function selected by FunSel will be applied to R2 and R4.)
0011	R1 and R2 are enabled. (Function selected by FunSel will be applied to R1 and R2.)	1011	Only R2 is enabled. (Function selected by FunSel will be applied to R2.)
0100	R1, R3, and R4 are enabled. (Function selected by FunSel will be applied to R1, R3, and R4.)	1100	R3 and R4 are enabled. (Function selected by FunSel will be applied to R3 and R4.)
0101	R1 and R3 are enabled. (Function selected by FunSel will be applied to R1 and R3.)	1101	Only R3 is enabled. (Function selected by FunSel will be applied to R3.)
0110	R1 and R4 are enabled. (Function selected by FunSel will be applied to R1 and R4.)	1110	Only R4 is enabled. (Function selected by FunSel will be applied to R4.)
0111	Only R1 is enabled. (Function selected by FunSel will be applied to R1.)	1111	NO general purpose register is enabled. (All registers retain their values.)

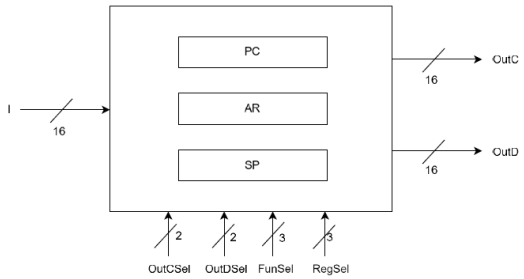
Table 4: ScrSel Control Input

ScrSel	Enable General Purpose Registers	ScrSel	Enable General Purpose Registers
0000	All general purpose registers are enabled. (Function selected by FunSel will be applied to S1, S2, S3, and S4.)	1000	S2, S3, and S4 are enabled. (Function selected by FunSel will be applied to S2, S3, and S4.)
0001	S1, S2, and S3 are enabled. (Function selected by FunSel will be applied to S1, S2, and S3.)	1001	S2 and S3 are enabled. (Function selected by FunSel will be applied to S2 and S3.)
0010	S1, S2, and S4 are enabled. (Function selected by FunSel will be applied to S1, S2, and S4.)	1010	S2 and S4 are enabled. (Function selected by FunSel will be applied to S2 and S4.)
0011	S1 and S2 are enabled. (Function selected by FunSel will be applied to S1 and S2.)	1011	Only S2 is enabled. (Function selected by FunSel will be applied to S2.)
0100	S1, S3, and S4 are enabled. (Function selected by FunSel will be applied to S1, S3, and S4.)	1100	S3 and S4 are enabled. (Function selected by FunSel will be applied to S3 and S4.)
0101	S1 and S3 are enabled. (Function selected by FunSel will be applied to S1 and S3.)	1101	Only S3 is enabled. (Function selected by FunSel will be applied to S3.)
0110	S1 and S4 are enabled. (Function selected by FunSel will be applied to S1 and S4.)	1110	Only S4 is enabled. (Function selected by FunSel will be applied to S4.)
0111	Only S1 is enabled. (Function selected by FunSel will be applied to S1.)	1111	NO general purpose register is enabled. (All registers retain their values.)

FunSel	R_x^+ (Next State)
000	R_x-1 (Decrement)
001	R_x +1 (Increment)
010	I (Load)
011	0 (Clear)
100	R_x (15-8) ← Clear, R_x (7-0) ← I (7-0) (Write Low)
101	R_x (7-0) ← I (7-0) (Only Write Low)
110	R_x (15-8) ← I (7-0) (Only Write High)
111	R_x (15-8) ← Sign Extend (I (7)) R_x (7-0) ← I (7-0) (Write Low)

(Part-2c) In this part we design the address register file (ARF) system shown in Figure 4 which consists of three 16-bit address registers: program counter (PC), address register (AR), and stack pointer (SP). FunSel and RegSel work as in Part-2b.

RegSel	Enable Address Registers
000	All address registers are enabled. (Function selected by FunSel will be applied to PC, AR, and SP.)
001	PC and AR are enabled. (Function selected by FunSel will be applied to PC and AR.)
010	PC and SP are enabled. (Function selected by FunSel will be applied to PC and SP.)
011	PC is enabled. (Function selected by FunSel will be applied to PC.)
100	AR and SP are enabled. (Function selected by FunSel will be applied to AR and SP.)
101	AR is enabled. (Function selected by FunSel will be applied to AR.)
110	SP is enabled. (Function selected by FunSel will be applied to SP.)
111	NO address register is enabled. (All registers retain their values.)



OutCSel and OutDSel are used to feed output lines OutC and OutD, respectively. 16 bits of the selected registers are output to OutC and OutD. Table below shows the selection of output registers based on the OutCSel and OutDSel control inputs.

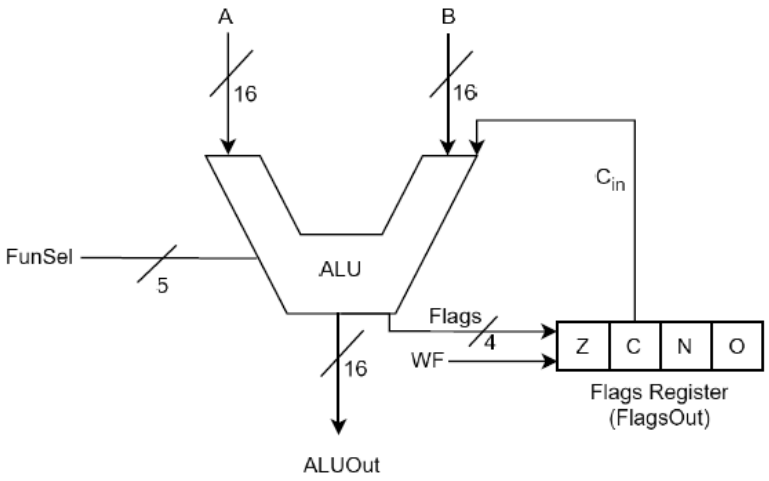
OutCSel	OutC
00	PC
01	PC
10	AR
11	SP

OutDSel	OutD
00	PC
01	PC
10	AR
11	SP

2.3 Part 3

(Part-3) In this part, we designed an Arithmetic Logic Unit (ALU) that has two 16-bit inputs, a 16-bit output, and a 4-bit output for zero, negative, carry, and overflow flags. The ALU is shown in Figure 5. The ALU functions and the flags that will be updated (i.e., - means that the flag will not be affected and + means that the flag changes based on the ALUOut) are given in Table below when WF (Write Flag) is 1.

- FunSel selects the function of the ALU.
- ALUOut shows the result of the operation that is selected by FunSel and applied on A and/or B inputs.
- Arithmetic operations are done using 2's complement logic.
- Z (zero) bit is set if ALUOut is zero (e.g. when NOT B is zero).
- C (carry) bit is set if ALUOut sets the carry (e.g. when LSL A produces carry)
- N (negative) bit is set if the ALU operation generates a negative result (e.g. when A-B results in a negative number).
- O (overflow) bit is set if an overflow occurs (e.g. when A+B results in an overflow).
- Note that Z—C—N—O flags are stored in a register! The Z flag is the MSB of the register, and the O flag is the LSB of the register.

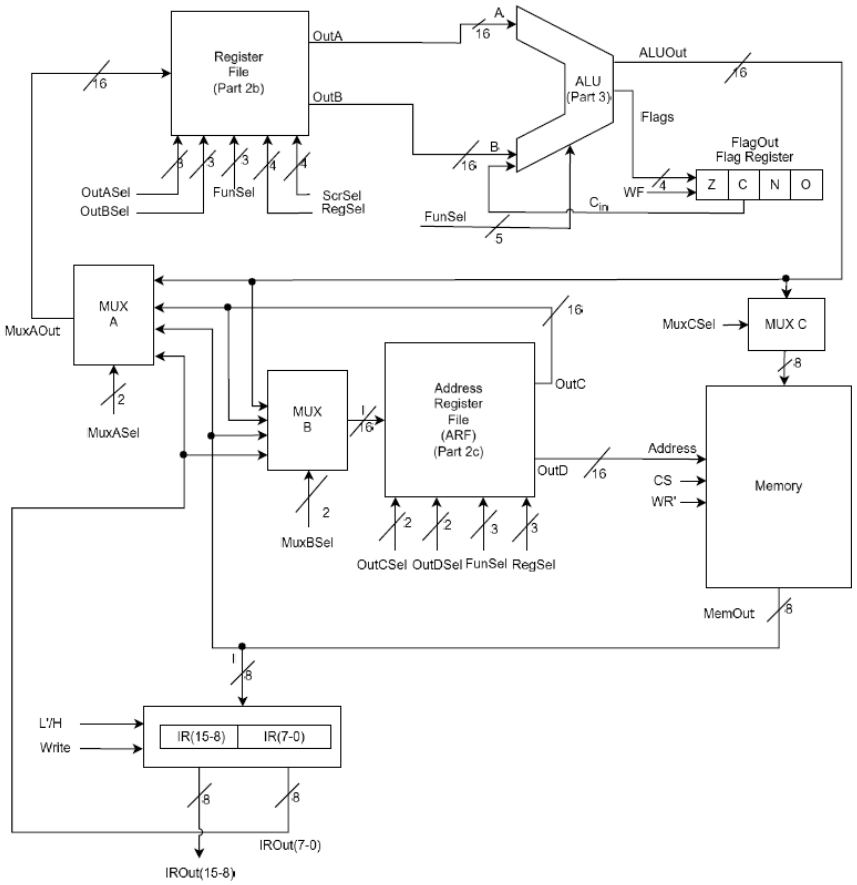


FunSel	ALUOut	Z	C	N	O
00000	A (8-bit)	+	-	+	-
00001	B (8-bit)	+	-	+	-
00010	NOT A (8-bit)	+	-	+	-
00011	NOT B (8-bit)	+	-	+	-
00100	A + B (8-bit)	+	+	+	+
00101	A + B + Carry (8-bit)	+	+	+	+
00110	A – B (8-bit)	+	+	+	+
00111	A AND B (8-bit)	+	-	+	-
01000	A OR B (8-bit)	+	-	+	-
01001	A XOR B (8-bit)	+	-	+	-
01010	A NAND B (8-bit)	+	-	+	-
01011	LSL A (8-bit)	+	+	+	-
01100	LSR A (8-bit)	+	+	+	-
01101	ASR A (8-bit)	+	+	-	-
01110	CSL A (8-bit)	+	+	+	-
01111	CSR A (8-bit)	+	+	+	-

FunSel	ALUOut	Z	C	N	O
10000	A (16-bit)	+	-	+	-
10001	B (16-bit)	+	-	+	-
10010	NOT A (16-bit)	+	-	+	-
10011	NOT B (16-bit)	+	-	+	-
10100	A + B (16-bit)	+	+	+	+
10101	A + B + Carry (16-bit)	+	+	+	+
10110	A – B (16-bit)	+	+	+	+
10111	A AND B (16-bit)	+	-	+	-
11000	A OR B (16-bit)	+	-	+	-
11001	A XOR B (16-bit)	+	-	+	-
11010	A NAND B (16-bit)	+	-	+	-
11011	LSL A (16-bit)	+	+	+	-
11100	LSR A (16-bit)	+	+	+	-
11101	ASR A (16-bit)	+	+	-	-
11110	CSL A (16-bit)	+	+	+	-
11111	CSR A (16-bit)	+	+	+	-

2.4 Part 4

In this part we implemented the system that shown in the figure by using previous implementations.



3 RESULTS [15 points]

We are given some simulation files. This part shows the results of this test files for each implementation.

3.1 Part 1

Results of Register

```
-----------------------------------
Register Simulation Started
[PASS] Test No: 1, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 2, Component: Q, Actual Value: 0x0024, Expected Value: 0x0024
[PASS] Test No: 3, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 4, Component: Q, Actual Value: 0x0026, Expected Value: 0x0026
[PASS] Test No: 5, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 6, Component: Q, Actual Value: 0x0012, Expected Value: 0x0012
[PASS] Test No: 7, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 8, Component: Q, Actual Value: 0x0000, Expected Value: 0x0000
Register Simulation Finished
0 Test Failed
8 Test Passed
-----------------------------------
```

3.2 Part 2.1

Results of Instruction Register

```
-----------------------------------
InstructionRegister Simulation Started
[PASS] Test No: 1, Component: IROut, Actual Value: 0x2315, Expected Value: 0x2315
[PASS] Test No: 2, Component: IROut, Actual Value: 0x1567, Expected Value: 0x1567
InstructionRegister Simulation Finished
0 Test Failed
2 Test Passed
-----------------------------------
```

3.3 Part 2.2

Results of Register File

```
-----------------------------------
RegisterFile Simulation Started
[PASS] Test No: 1, Component: OutA, Actual Value: 0x1234, Expected Value: 0x1234
[PASS] Test No: 1, Component: OutB, Actual Value: 0x5678, Expected Value: 0x5678
[PASS] Test No: 2, Component: OutA, Actual Value: 0x1234, Expected Value: 0x1234
[PASS] Test No: 2, Component: OutB, Actual Value: 0x3548, Expected Value: 0x3548
RegisterFile Simulation Finished
0 Test Failed
4 Test Passed
-----------------------------------
```

3.4 Part 2.3

Results of Address Register File

```
-----------------------------------
AddressRegisterFile Simulation Started
[PASS] Test No: 1, Component: OutC, Actual Value: 0x1234, Expected Value: 0x1234
[PASS] Test No: 1, Component: OutD, Actual Value: 0x5678, Expected Value: 0x5678
[PASS] Test No: 2, Component: OutA, Actual Value: 0x1234, Expected Value: 0x1234
[PASS] Test No: 2, Component: OutB, Actual Value: 0x3548, Expected Value: 0x3548
AddressRegisterFile Simulation Finished
0 Test Failed
4 Test Passed
-----------------------------------
```

3.5 Part 3

Results of Arithmetic Logic Unit

```
-----------------------------------
ArithmeticLogicUnit Simulation Started
[PASS] Test No: 1, Component: ALUOut, Actual Value: 0x5555, Expected Value: 0x5555
[PASS] Test No: 1, Component: Z, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 1, Component: C, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 1, Component: N, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 1, Component: O, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 2, Component: ALUOut, Actual Value: 0x5555, Expected Value: 0x5555
[PASS] Test No: 2, Component: Z, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: C, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: N, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: O, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 3, Component: ALUOut, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 3, Component: Z, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 3, Component: C, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 3, Component: N, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 3, Component: O, Actual Value: 0x0000, Expected Value: 0x0000
ArithmeticLogicUnit Simulation Finished
0 Test Failed
15 Test Passed
-----------------------------------
```

3.6 Part 4

Results of Arithmetic Logic Unit System

```
-----------------------------------
ArithmeticLogicUnitSystem Simulation Started
[PASS] Test No: 1, Component: OutA, Actual Value: 0x7777, Expected Value: 0x7777
[PASS] Test No: 1, Component: OutB, Actual Value: 0x8887, Expected Value: 0x8887
[PASS] Test No: 1, Component: ALUOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 1, Component: Z, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: C, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: N, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: O, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: MuxAOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 1, Component: MuxBOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 1, Component: MuxCOut, Actual Value: 0x00fe, Expected Value: 0x00fe
[PASS] Test No: 1, Component: R2, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: S3, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: OutA, Actual Value: 0x7777, Expected Value: 0x7777
[PASS] Test No: 2, Component: OutB, Actual Value: 0x8887, Expected Value: 0x8887
[PASS] Test No: 2, Component: ALUOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 2, Component: Z, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: C, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: N, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 2, Component: O, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: MuxAOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 2, Component: MuxBOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 2, Component: MuxCOut, Actual Value: 0x00fe, Expected Value: 0x00fe
[PASS] Test No: 2, Component: R2, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 2, Component: S3, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 2, Component: PC, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 3, Component: OutC, Actual Value: 0x1254, Expected Value: 0x1254
[PASS] Test No: 3, Component: Address, Actual Value: 0x0023, Expected Value: 0x0023
[PASS] Test No: 3, Component: Memout, Actual Value: 0x0015, Expected Value: 0x0015
[PASS] Test No: 3, Component: IROut, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 4, Component: OutC, Actual Value: 0x1254, Expected Value: 0x1254
[PASS] Test No: 4, Component: Address, Actual Value: 0x0023, Expected Value: 0x0023
[PASS] Test No: 4, Component: Memout, Actual Value: 0x0015, Expected Value: 0x0015
[PASS] Test No: 4, Component: IROut, Actual Value: 0x0015, Expected Value: 0x0015
ArithmeticLogicUnitSystem Simulation Finished
0 Test Failed
33 Test Passed
-----------------------------------
```

4 DISCUSSION

4.1 Part 1

The project involves the design of a 16-bit register with versatile functionality controlled by 3-bit control signals (FunSel) and an enable input (E). The register's operation is determined by the combination of these control signals, allowing for eight distinct functionalities. The graphic symbol of the register, along with its characteristic table, facilitates understanding of its behavior. The register's flexibility and configurability make it a valuable component for subsequent stages of the project, enabling diverse operations within a digital system.

4.2 Part 2.1

The project entails designing a 16-bit Instruction Register (IR) with specified block diagram and function table, as depicted in the report. The IR serves as a storage unit for 16-bit binary data. Notably, while the register has the capacity to store 16 bits, the input bus is only 8 bits wide. As a result, the register incorporates functionality to load either the lower half (bits 7-0) or the higher half (bits 15-8) of the input data, determined by the L'H signal. This capability allows for versatile usage of the IR within digital systems, facilitating efficient data storage and manipulation in accordance with system requirements.

4.3 Part 2.2

The project involves designing a system, comprising four 16-bit general-purpose registers (R1, R2, R3, R4) and four 16-bit scratch registers (S1, S2, S3, S4). The system incorporates two output lines, OutA and OutB, driven by control signals OutASel and OutBSel, respectively. Based on the values of OutASel and OutBSel, 16 bits of data from selected registers are output to OutA and OutB. The selection of output registers is determined by output selection table given above. Furthermore, the system includes control signals RegSel and ScrSel, both 4-bit signals, used to select registers for executing functions specified by the 3-bit FunSel signal. Tables for RegSel and ScrSel outline the registers selected by RegSel and ScrSel, respectively. This design facilitates flexible data storage, retrieval, and manipulation within the system, catering to diverse operational requirements.

4.4 Part 2.3

The part involves designing an Address Register File (ARF) system, comprising three 16-bit address registers: Program Counter (PC), Address Register (AR), and Stack Pointer (SP). The system operates with control signals FunSel and RegSel, akin to Part-2b. Additionally, the system incorporates control signals OutCSel and OutDSel, utilized to direct data to output lines OutC and OutD, respectively. 16 bits of data from selected registers are output to OutC and OutD based on the values of OutCSel and OutDSel. This design facilitates efficient management and manipulation of address data within the system, supporting diverse functionalities and operational requirements.

4.5 Part 3

The part entails designing an Arithmetic Logic Unit (ALU) featuring two 16-bit inputs, a 16-bit output, and a 4-bit output for zero, negative, carry, and overflow flags. The ALU's operation is governed by control signals (FunSel), determining various arithmetic and logical functions as detailed in Table given in the report. These functions include arithmetic operations conducted using 2's complement logic, with corresponding flag updates stored in a register. The ALU's output provides essential data for subsequent processing within the system, contributing to efficient computation and decision-making.

4.6 Part 4

In this part we implemented a design that includes all of previous implementation of this project. This part includes MuxA, MuxB and MuxC that enable us to manipulate corresponding inputs of each part thus, we can process some predetermined programs in the memory.

5 CONCLUSION

In conclusion, the design and implementation of the various components—ranging from the 16-bit register systems to the Arithmetic Logic Unit (ALU)—have culminated in a robust and versatile digital system. Each component, meticulously crafted to fulfill specific functional requirements, contributes to the overall functionality and efficiency of the system. The Address Register File (ARF) system facilitates seamless management and manipulation of address data, while the register systems provide essential storage capabilities. The ALU, with its array of arithmetic and logical functions, empowers the system with the ability to perform complex computations and decision-making tasks. Together, these components form a cohesive and adaptable digital system capable of executing a diverse range of operations, paving the way for further exploration and development in the realm of digital systems design.

REFERENCES