ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION
PROJECT 2 REPORT

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

GROUP MEMBERS:

1500220079 : AHMET ENES ÇİĞDEM

SPRING 2024

Contents

1 INTRODUCTION

In this raport I will basicly explain a CPU system with Arithmetic Logic Unit System and Contro Unit. ALU System is already impelmented befor. This raport will be mostly about Control Unit and CPU System.

2 MATERIALS AND METHODS

Implementation of this project is made by using given instruction table. This table executed by an ALU System which is made before the Control Unit. Control Unit designed to control the inputs of ALU System and manipulate the RTL system according to corresponding operations in Figure 2. ALU System shown in Figure 1:

2.1 Programming Language

This project is implemented by using Verilog HDL. The operations and the ALU system is shown in figures

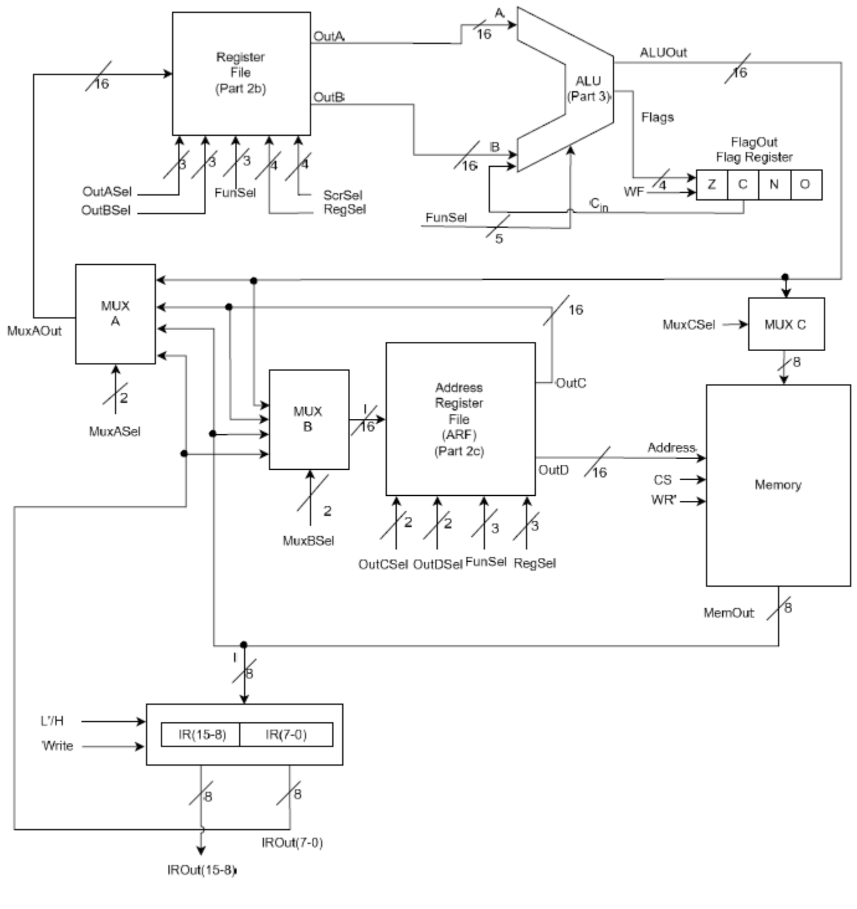
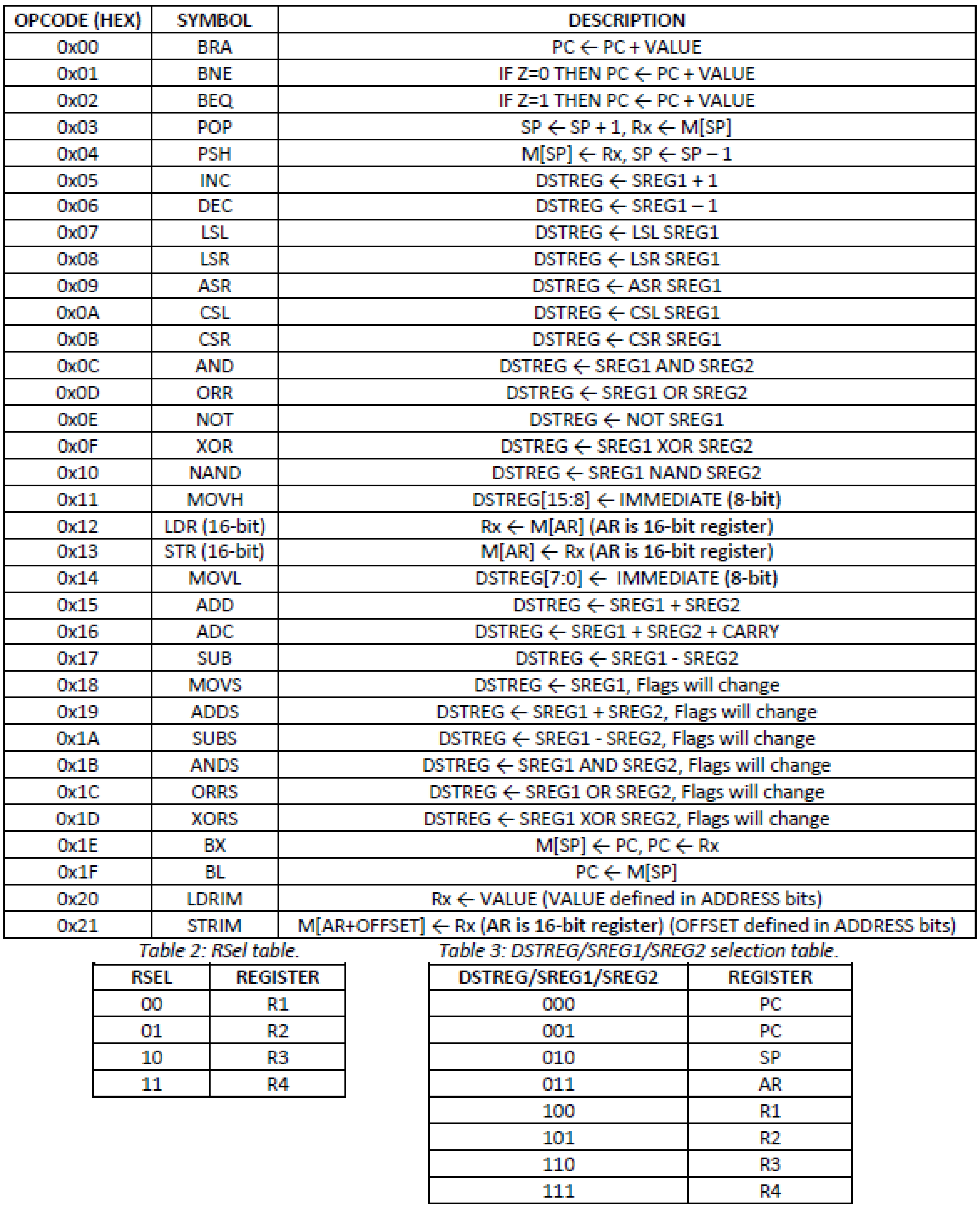


Figure 1: ALU System



OPCODE (HEX)	SYMBOL	DESCRIPTION
0x00	BRA	PC ← PC + VALUE
0x01	BNE	IF Z=0 THEN PC ← PC + VALUE
0x02	BEQ	IF Z=1 THEN PC ← PC + VALUE
0x03	POP	SP ← SP + 1, Rx ← M[SP]
0x04	PSH	M[SP] ← Rx, SP ← SP – 1
0x05	INC	DSTREG ← SREG1 + 1
0x06	DEC	DSTREG ← SREG1 – 1
0x07	LSL	DSTREG ← LSL SREG1
0x08	LSR	DSTREG ← LSR SREG1
0x09	ASR	DSTREG ← ASR SREG1
0x0A	CSL	DSTREG ← CSL SREG1
0x0B	CSR	DSTREG ← CSR SREG1
0x0C	AND	DSTREG ← SREG1 AND SREG2
0x0D	ORR	DSTREG ← SREG1 OR SREG2
0x0E	NOT	DSTREG ← NOT SREG1
0x0F	XOR	DSTREG ← SREG1 XOR SREG2
0x10	NAND	DSTREG ← SREG1 NAND SREG2
0x11	MOVH	DSTREG[15:8] ← IMMEDIATE **(8-bit)**
0x12	LDR (16-bit)	Rx ← M[AR] **(AR is 16-bit register)**
0x13	STR (16-bit)	M[AR] ← Rx **(AR is 16-bit register)**
0x14	MOVL	DSTREG[7:0] ← IMMEDIATE **(8-bit)**
0x15	ADD	DSTREG ← SREG1 + SREG2
0x16	ADC	DSTREG ← SREG1 + SREG2 + CARRY
0x17	SUB	DSTREG ← SREG1 - SREG2
0x18	MOVS	DSTREG ← SREG1, Flags will change
0x19	ADDS	DSTREG ← SREG1 + SREG2, Flags will change
0x1A	SUBS	DSTREG ← SREG1 - SREG2, Flags will change
0x1B	ANDS	DSTREG ← SREG1 AND SREG2, Flags will change
0x1C	ORRS	DSTREG ← SREG1 OR SREG2, Flags will change
0x1D	XORS	DSTREG ← SREG1 XOR SREG2, Flags will change
0x1E	BX	M[SP] ← PC, PC ← Rx
0x1F	BL	PC ← M[SP]
0x20	LDRIM	Rx ← VALUE (VALUE defined in ADDRESS bits)
0x21	STRIM	M[AR+OFFSET] ← Rx **(AR is 16-bit register)** (OFFSET defined in ADDRESS bits)

Table 2: RSel table.

RSEL	REGISTER
00	R1
01	R2
10	R3
11	R4

Table 3: DSTREG/SREG1/SREG2 selection table.

DSTREG/SREG1/SREG2	REGISTER
000	PC
001	PC
010	SP
011	AR
100	R1
101	R2
110	R3
111	R4

Figure 2: Operation Codes

2.2 Implementation

The general implementation of this project is made by using clock cycle. No matter what type operation is being executed, the system goes throughout T based time loop and each operation change the inputs of ALU System based on their op code and other bits of instruction. When instruction is idetifed the inputs of the system will be arranged according to our operation.

Fetch and Decode

First I want to explain the fetch and decode steps of this RTL. In the fetch side our IR is 16 bits but our Memory words are 8 bits long so we need to first put instructions to IR in 2 clock cycle. T_0 and T_1.

These instructions are read from memory whit program counter adress after the fetch we will have new adress if we need for operation.

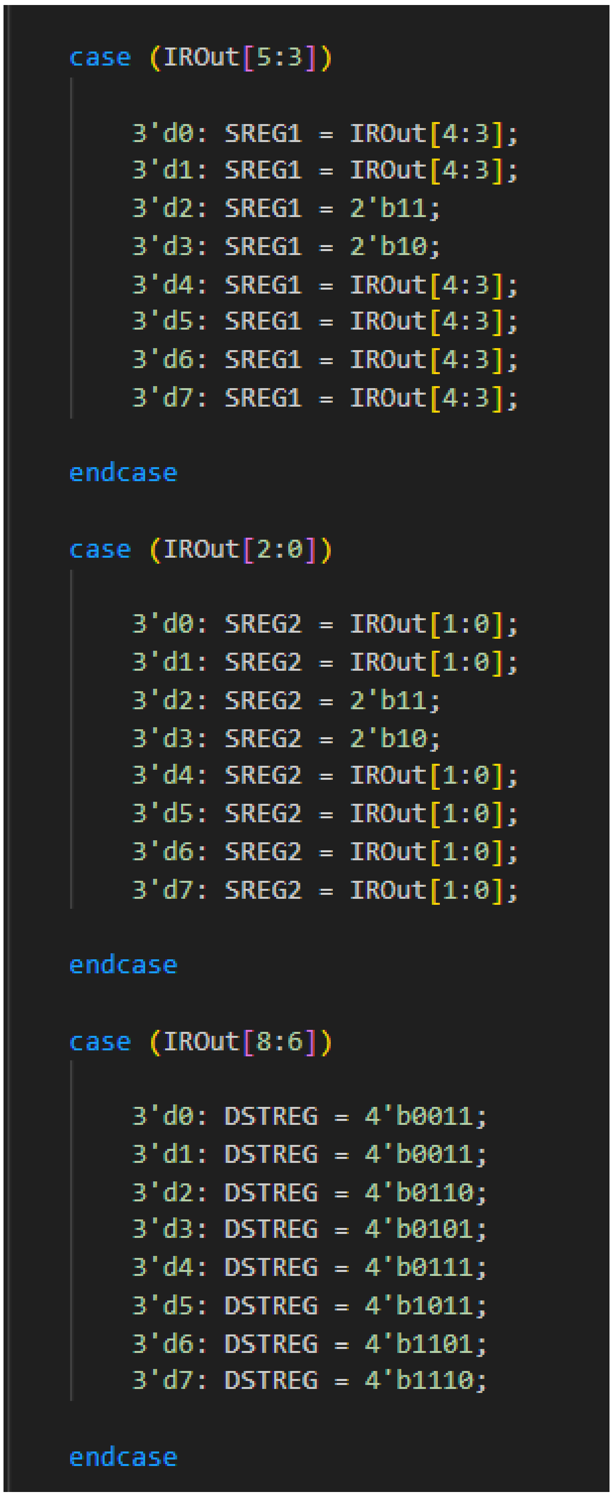
$\mathbf{T}_0$: IR[7:0] $\leftarrow M[PC], PC \leftarrow PC + 1$

$\mathbf{T}_1$: IR[15:8] $\leftarrow M[PC], PC \leftarrow PC + 1$

After the fetch operation, we need to determine DSTREG, SREG1 and SREG2 for register referance instructions and RSEL for address referance then put the address to AR if we need.

$\mathbf{T}_2$: AR $\leftarrow IR[7:0]$

And we need to identfy our input acoording to DSTREG, SREG1, SREG2 and RSEL. DSTREG is the register that will be written something so it must manipulate the RegSel of our ARF or RF and SREGs must determine the OutSel inputs. The basic match code is in the figure. When we need them, thanks to this code, we will just write these to corresponding inputs.



```
case (IROut[5:3])

    3'd0: SREG1 = IROut[4:3];
    3'd1: SREG1 = IROut[4:3];
    3'd2: SREG1 = 2'b11;
    3'd3: SREG1 = 2'b10;
    3'd4: SREG1 = IROut[4:3];
    3'd5: SREG1 = IROut[4:3];
    3'd6: SREG1 = IROut[4:3];
    3'd7: SREG1 = IROut[4:3];

endcase

case (IROut[2:0])

    3'd0: SREG2 = IROut[1:0];
    3'd1: SREG2 = IROut[1:0];
    3'd2: SREG2 = 2'b11;
    3'd3: SREG2 = 2'b10;
    3'd4: SREG2 = IROut[1:0];
    3'd5: SREG2 = IROut[1:0];
    3'd6: SREG2 = IROut[1:0];
    3'd7: SREG2 = IROut[1:0];

endcase

case (IROut[8:6])

    3'd0: DSTREG = 4'b0011;
    3'd1: DSTREG = 4'b0011;
    3'd2: DSTREG = 4'b0110;
    3'd3: DSTREG = 4'b0101;
    3'd4: DSTREG = 4'b0111;
    3'd5: DSTREG = 4'b1011;
    3'd6: DSTREG = 4'b1101;
    3'd7: DSTREG = 4'b1110;

endcase
```

Figure 3: Matching for registers

Then we need to identify what is our operation. I have 33 operations on this project but there could be more ,so for just to make it more flexible to more operation, I will make a 64 bits D register that will make the cooresponding binary cod 1 and others 0 according to operation code.

D[IR[15:10]] = 1.

This register show me what is the operation is according to op code written in IR[15:10]. All of these operations happen in clock T_2.

All of these for fetch and decode, now lets begin to execute operations.

Execution of some operations

BRA:

T_3: S1 $\leftarrow PC$

T_4: S2 $\leftarrow IR[7:0]$

T_5: PC $\leftarrow S1 + S2$

BNE and BNQ same as BRA just depend on Z as well.

POP:

T_3: M[SP] $\leftarrow Rx[7:0], SP \leftarrow SP + 1$

T_4: M[SP] $\leftarrow Rx[7:0], SP \leftarrow SP + 1$

INC:

T_3: DSTREG $\leftarrow SREG1$

T_4: DSTREG $\leftarrow DSTREG + 1$

In DEC just the register functin is changed.

LSL:

In LSL for the RF is source and destination it can be handled in just one clock cycle.

T_3: DSTREG $\leftarrow LSLSREG1$

The output of ALU is equal to LSL SREG1 so we just write it DSTREG.

LSR, ASR, CSL, CSR is same as this operation just ALU Funsel will be changed for each opeartin.

AND:

Same as the LSR if it's just for RF the result is directly writable to DSTREG.

T_3: DSTREG $\leftarrow SREG1 \wedge SREG2$

Operations OR NOT XOR NAND ADD SUB ADDS SUBS ANDS ORRS XORS is same as operation just ALU Funsel will be changed for corresponding operation.

STRIM:

T_3: S1 $\leftarrow AR$

T_4: S2 $\leftarrow IR[7:0]$

T_3: AR $\leftarrow S1 + S2$

T_3: M[AR] $\leftarrow Rx[7:0], AR \leftarrow AR + 1$

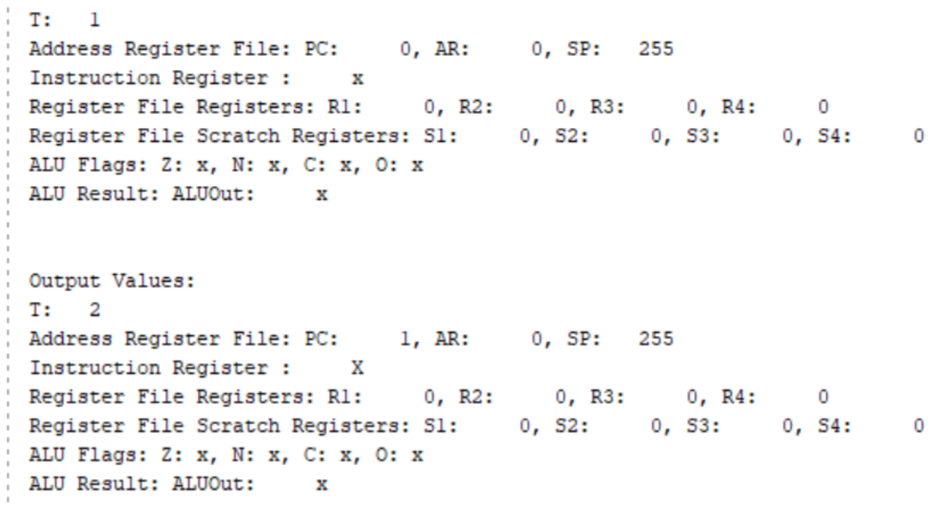
T_3: M[AR] $\leftarrow Rx[15:8]$

Example output for STRIM in the results section.

3 RESULTS

Example results for some operations:

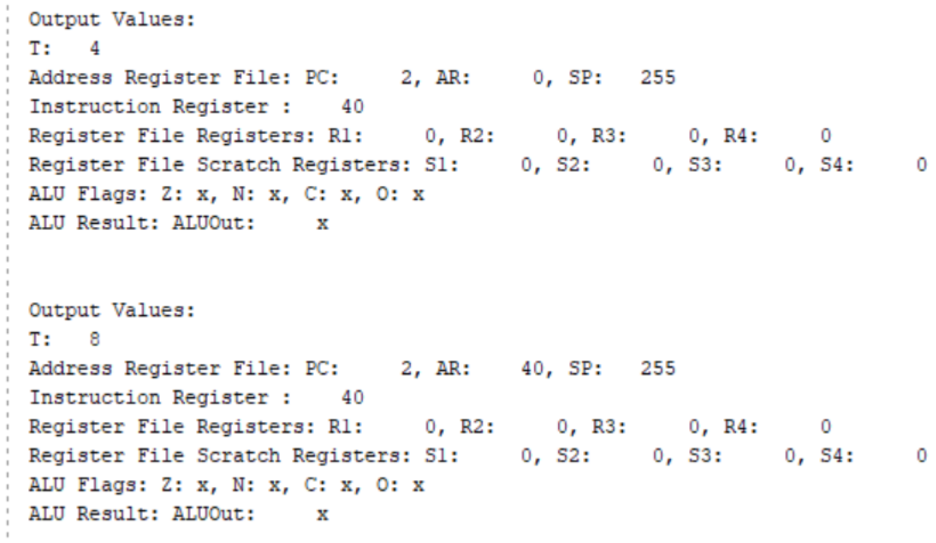
BRA 0x28



```
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, N: x, C: 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, N: x, C: x, O: x
ALU Result: ALUOut:     x
```

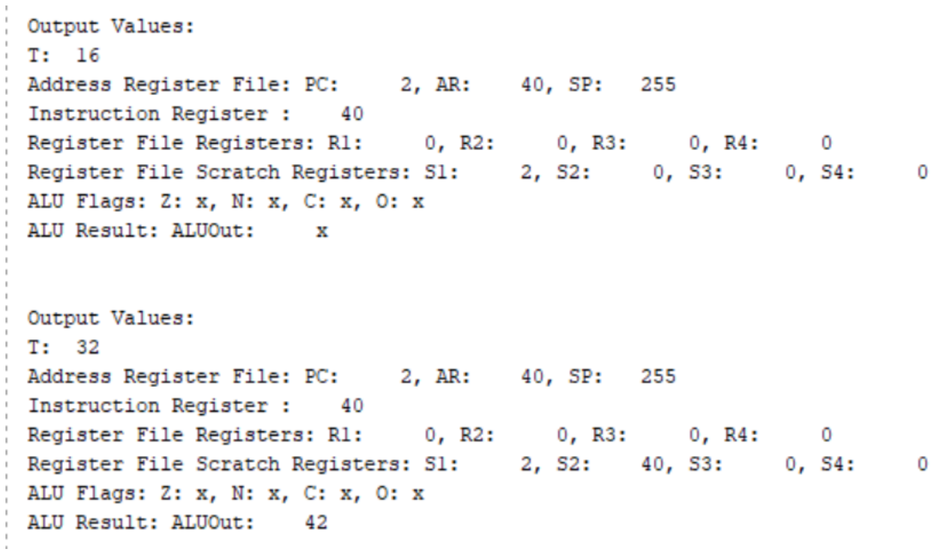
Figure 4: BRA 1



```
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, N: x, C: x, O: x
ALU Result: ALUOut:     x


Output Values:
T:   8
Address Register File: PC:     2, AR:    40, 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, N: x, C: x, O: x
ALU Result: ALUOut:     x
```

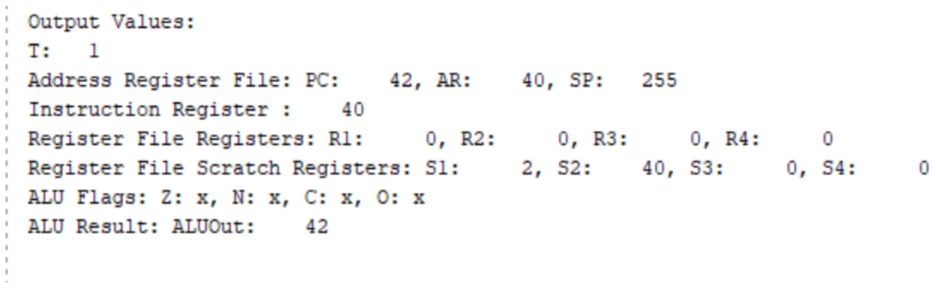
Figure 5: BRA 2



```
Output Values:
T:  16
Address Register File: PC:     2, AR:    40, SP:   255
Instruction Register :    40
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:     x


Output Values:
T:  32
Address Register File: PC:     2, AR:    40, SP:   255
Instruction Register :    40
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     2, S2:    40, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:    42
```

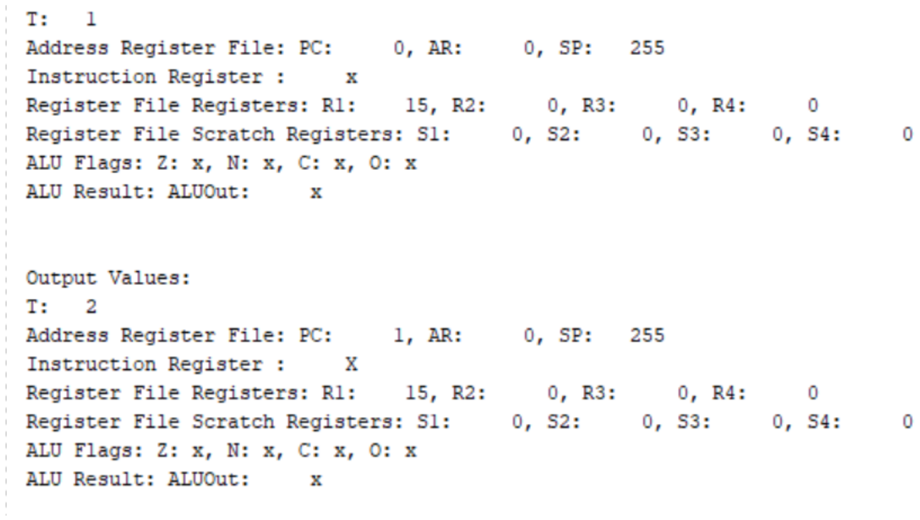
Figure 6: BRA 3



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

Figure 7: BRA 4

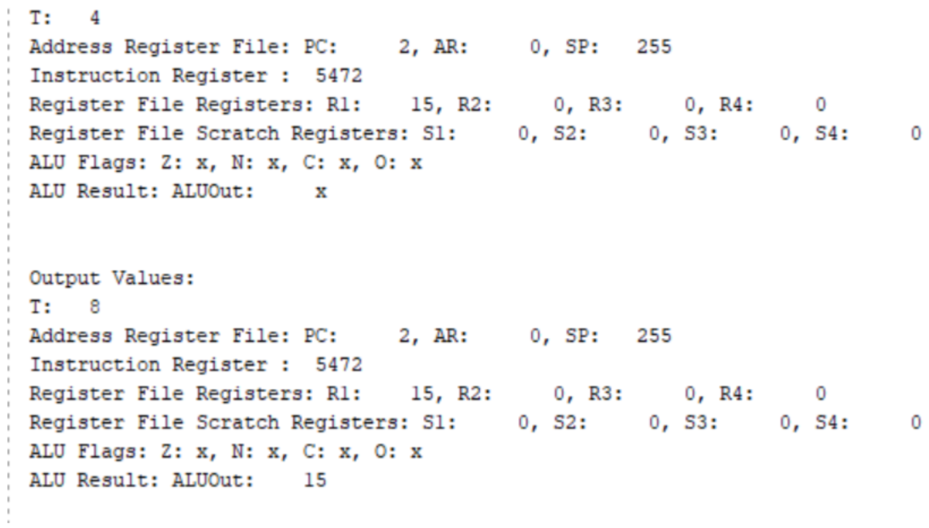
INC R1, R2



```
T:   1
Address Register File: PC:     0, AR:     0, SP:   255
Instruction Register :     x
Register File Registers: R1:    15, 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:     0, SP:   255
Instruction Register :     X
Register File Registers: R1:    15, 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
```

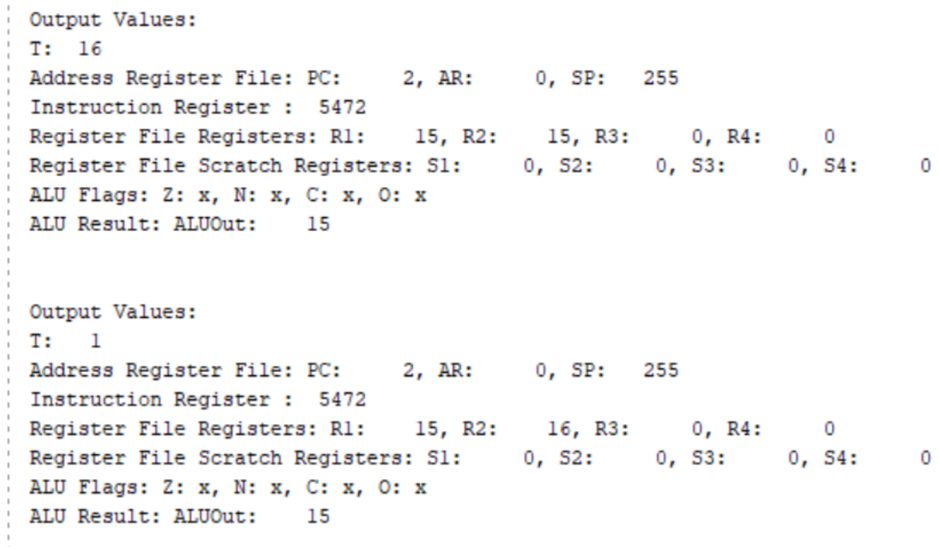
Figure 8: INC 1



```
T:   4
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :  5472
Register File Registers: R1:    15, 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:   8
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :  5472
Register File Registers: R1:    15, 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:    15
```

Figure 9: INC 2

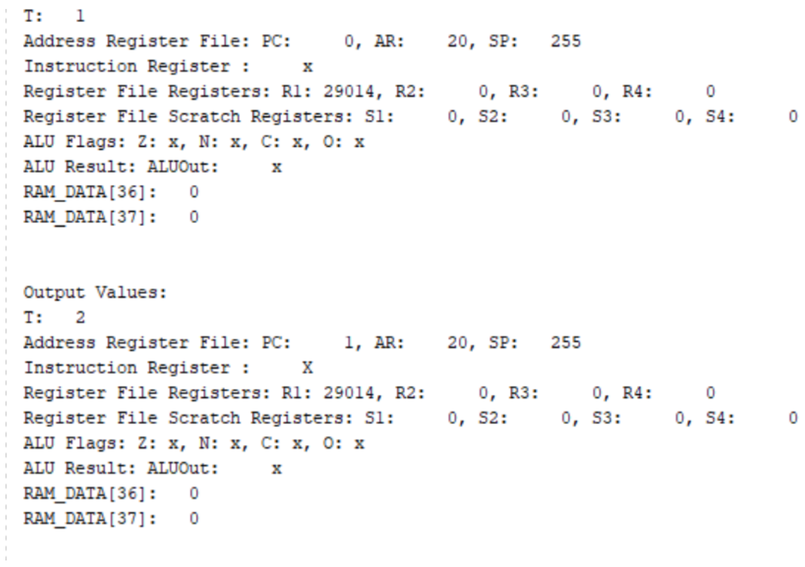


```
Output Values:
T:  16
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :  5472
Register File Registers: R1:    15, R2:    15, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:    15


Output Values:
T:   1
Address Register File: PC:     2, AR:     0, SP:   255
Instruction Register :  5472
Register File Registers: R1:    15, R2:    16, 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:    15
```

Figure 10: INC 3

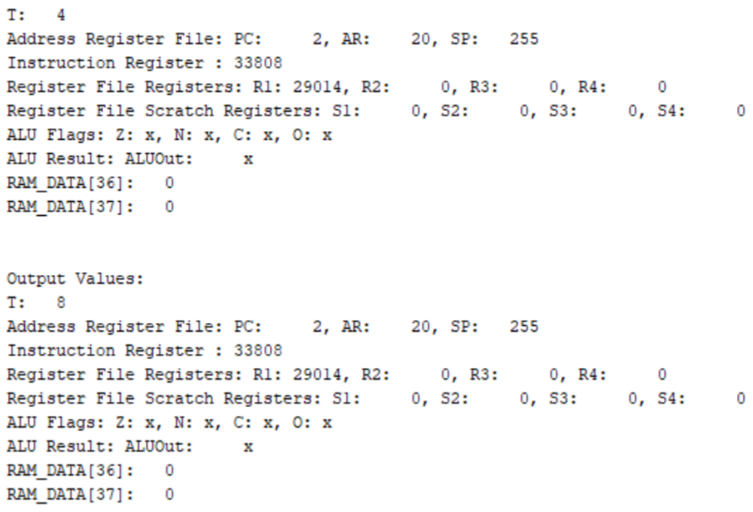
STRIM, R1, 0x10



```
T:   1
Address Register File: PC:     0, AR:    20, SP:   255
Instruction Register :     x
Register File Registers: R1: 29014, 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
RAM_DATA[36]:   0
RAM_DATA[37]:   0


Output Values:
T:   2
Address Register File: PC:     1, AR:    20, SP:   255
Instruction Register :     X
Register File Registers: R1: 29014, 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
RAM_DATA[36]:   0
RAM_DATA[37]:   0
```

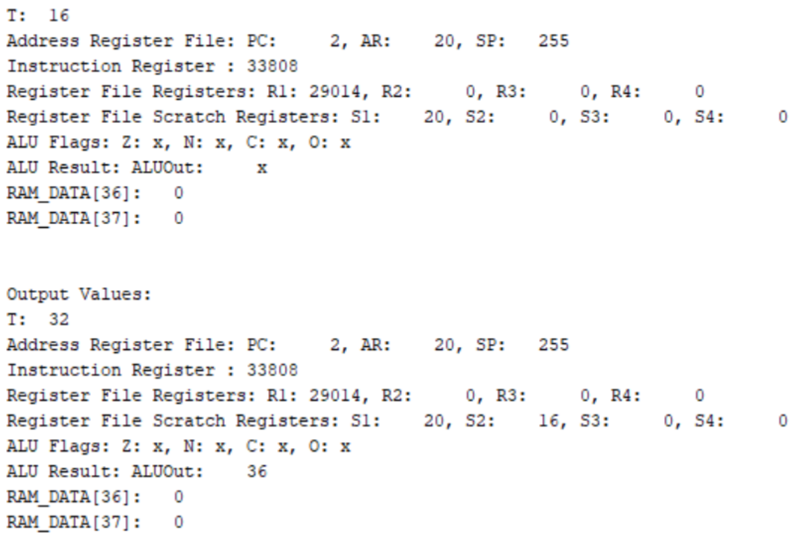
Figure 11: STRIM 1



```
T:   4
Address Register File: PC:     2, AR:    20, SP:   255
Instruction Register : 33808
Register File Registers: R1: 29014, 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
RAM_DATA[36]:   0
RAM_DATA[37]:   0


Output Values:
T:   8
Address Register File: PC:     2, AR:    20, SP:   255
Instruction Register : 33808
Register File Registers: R1: 29014, 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
RAM_DATA[36]:   0
RAM_DATA[37]:   0
```

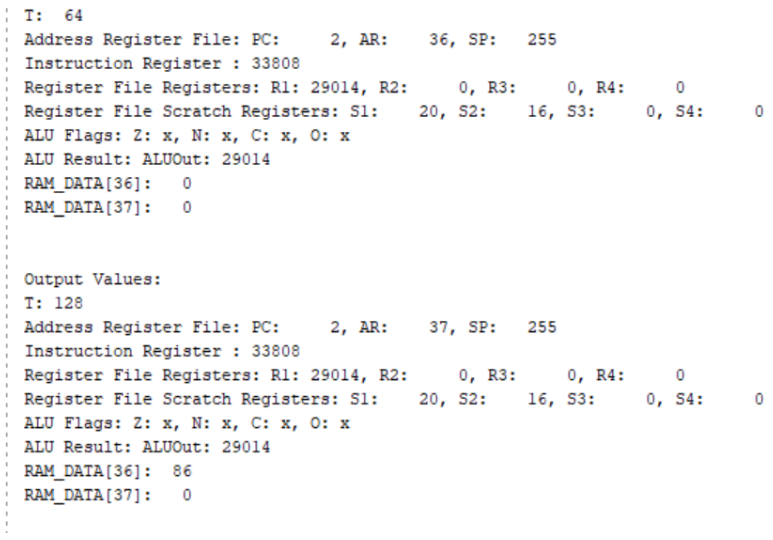
Figure 12: STRIM 2



```
T:  16
Address Register File: PC:     2, AR:    20, SP:   255
Instruction Register : 33808
Register File Registers: R1: 29014, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:    20, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x
RAM_DATA[36]:   0
RAM_DATA[37]:   0


Output Values:
T:  32
Address Register File: PC:     2, AR:    20, SP:   255
Instruction Register : 33808
Register File Registers: R1: 29014, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:    20, S2:    16, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:    36
RAM_DATA[36]:   0
RAM_DATA[37]:   0
```

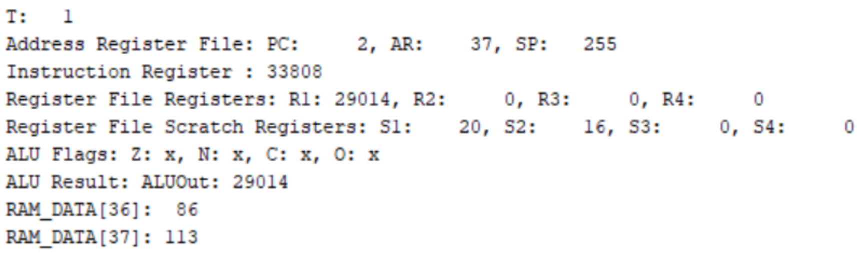
Figure 13: STRIM 3



```
T:  64
Address Register File: PC:     2, AR:    36, SP:   255
Instruction Register : 33808
Register File Registers: R1: 29014, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:    20, S2:    16, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut: 29014
RAM_DATA[36]:   0
RAM_DATA[37]:   0


Output Values:
T: 128
Address Register File: PC:     2, AR:    37, SP:   255
Instruction Register : 33808
Register File Registers: R1: 29014, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:    20, S2:    16, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut: 29014
RAM_DATA[36]:  86
RAM_DATA[37]:   0
```

Figure 14: STRIM 4



```
T:   1
Address Register File: PC:     2, AR:    37, SP:   255
Instruction Register : 33808
Register File Registers: R1: 29014, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:    20, S2:    16, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut: 29014
RAM_DATA[36]:  86
RAM_DATA[37]: 113
```

Figure 15: STRIM 5

4 DISCUSSION

In our CPU development, combining the ALU with the CU was a milestone. However, we faced a hurdle fetching ALU inputs through RTL with the CU. Coordinating CU control logic with ALU data processing proved challenging. Despite meticulous planning, routing and synchronizing inputs proved tricky due to data flow complexities and timing issues. Overcoming these challenges demanded a deeper understanding of hardware interactions. Through iterative refinement and experimentation, we navigated these hurdles, showcasing our team's adaptability and commitment to innovation.

5 CONCLUSION

In conclusion, our journey to integrate the ALU with the CU in our CPU architecture was filled with challenges and triumphs. Despite significant hurdles in fetching inputs for the ALU through RTL, our team's perseverance and collaboration prevailed. Through theoretical understanding, practical experimentation, and iterative refinement, we successfully addressed complexities in data routing and synchronization. This underscores our resilience and commitment to innovation, positioning us at the forefront of CPU architecture advancement. Moving forward, we are dedicated to pushing the boundaries of hardware design, driving continued progress and innovation in computing.

REFERENCES