KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
COMPUTER ENGINEERING DEPARTMENT

COE 301 Lab: Computer Organization

Term 211 (Fall 2021)

Single Cycle CPU Project

ABDULAZIZ ALTHOBAITI s201859320
SAUD ALKHULAIFI s201837340
ABDULAZIZ ALAMRI s201842560

Contents

Overview of Single-Cycle CPU Design 2

Components 3

Program Counter (PC) 3
Register File 4
ALU 5
Shifter 6
PC +2 6
Arithmetic Unit 7
Logic Unit 7
Instruction Memory 8
Data Memory 9

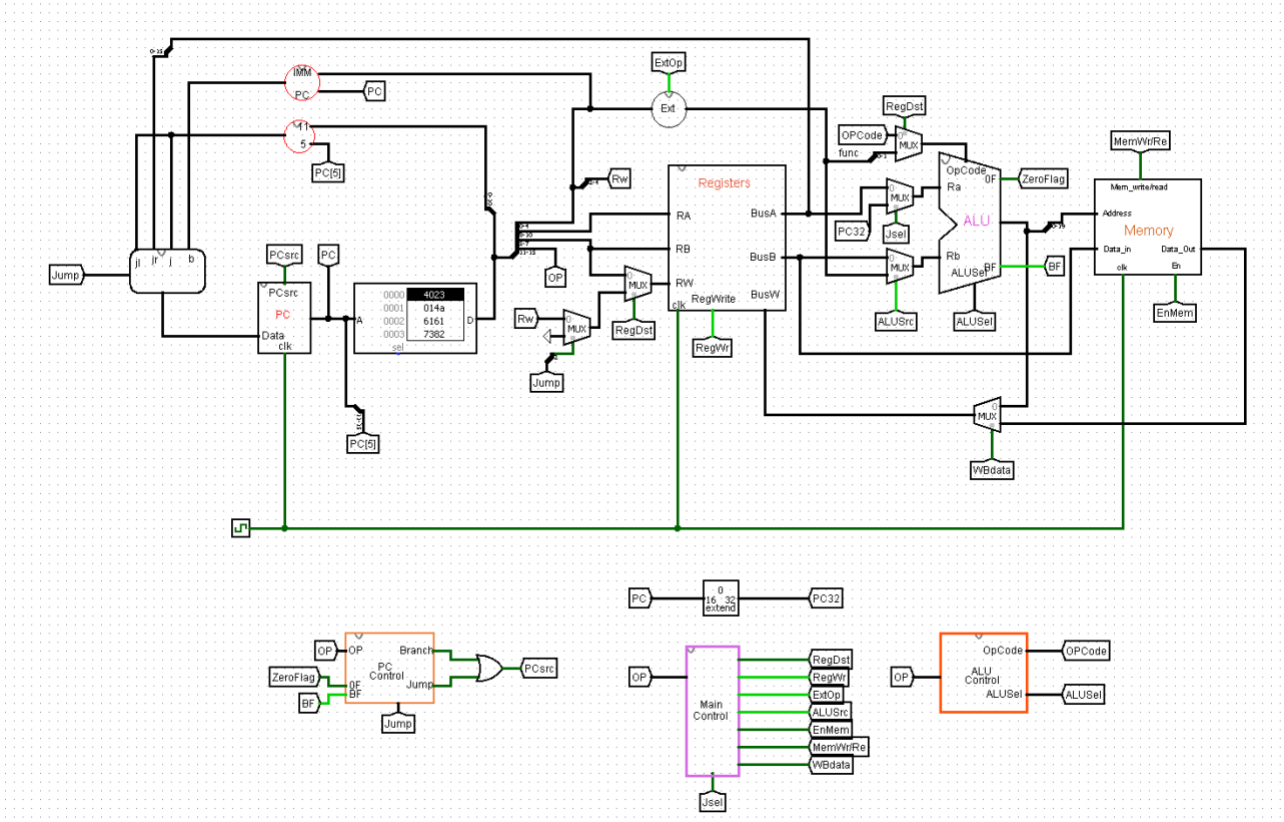
Control Signals 10

Main Control 10
ALU Control 11
PC Control 12

Problems Faced 12

Instruction Encoding 13

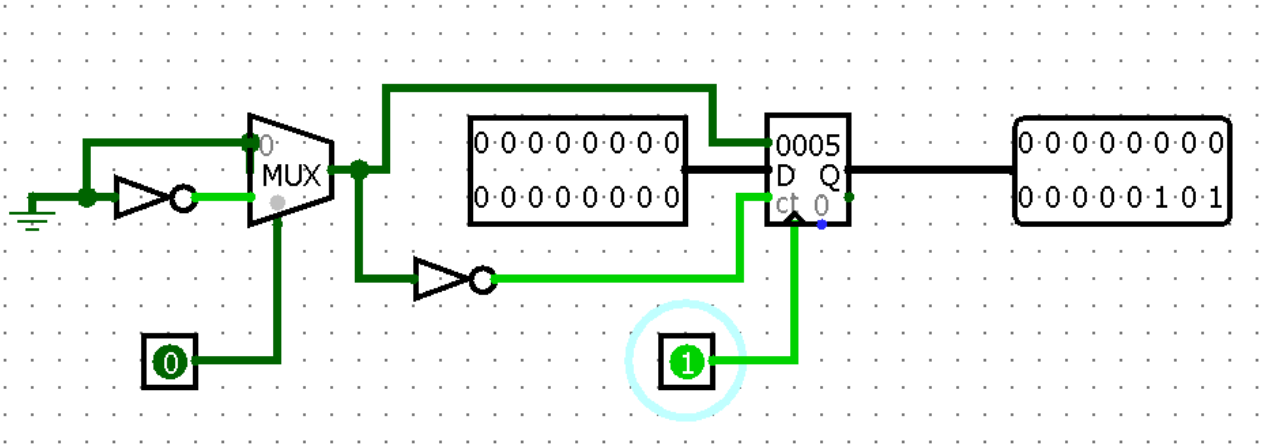
Overview of Single-Cycle CPU Design



For every tick, an instruction is passed from the Instruction Memory to its defined path. Every tick will cause the Program Counter to be updated according to the selected instruction. After that, the Register File Component will receive the specified registers and whether write to them or read from them. Furthermore, the ALU Component will execute the operation that was passed from The Instruction Memory. After finishing the ALU operation, the data memory component will act according to the control signals that was passed from the Main Control Component.

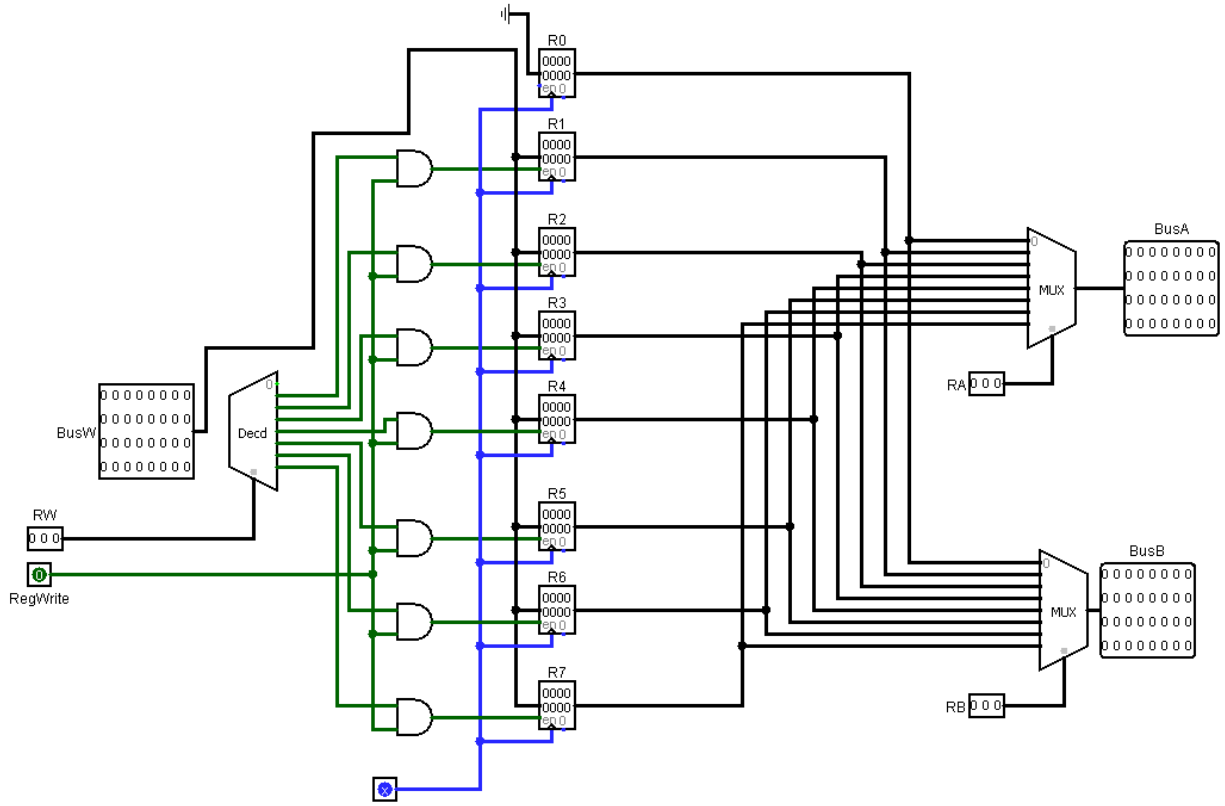
Components

Program Counter (PC)



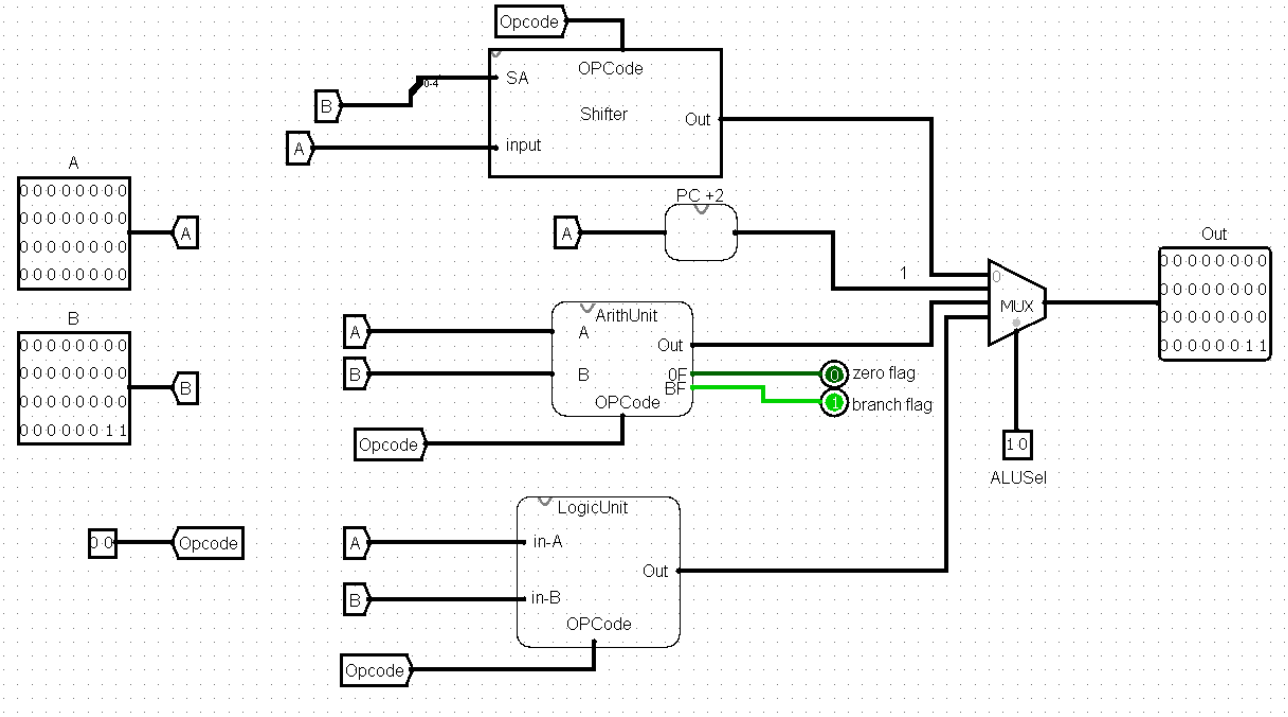
The Program Counter (PC) will be updated based on what type of instruction is passed from the instruction Memory Component. If the MUX is 0, then the PC will increment by 1 each tick. If the MUX is 1, it will load the address of the branch/jump instruction.

Register File



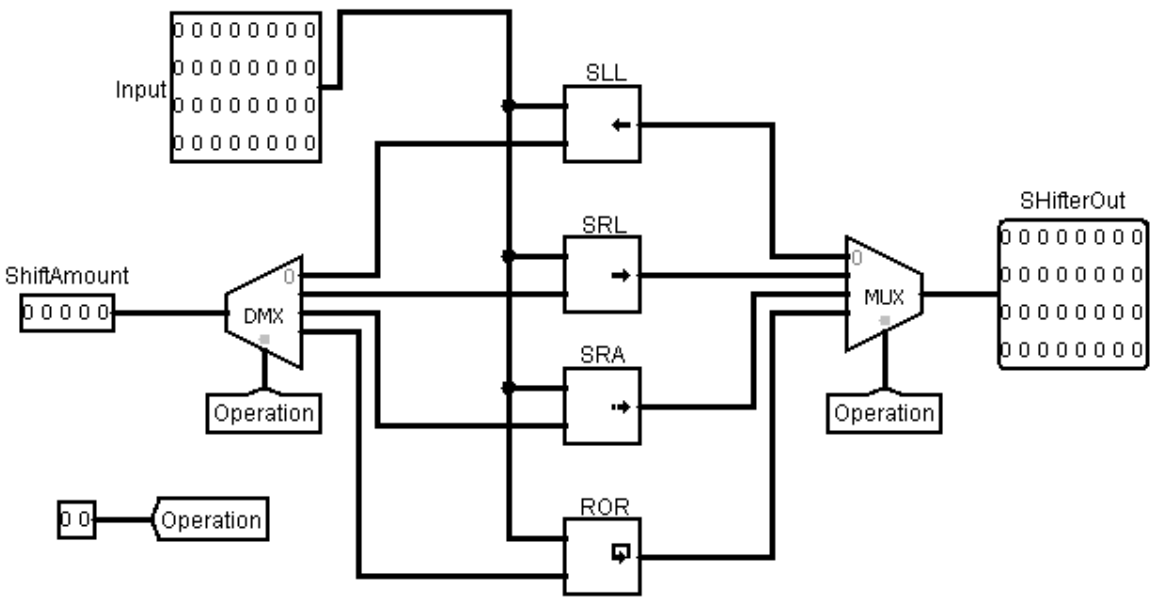
For every tick, RW will select the register to write in or read from using a decoder. After that, BusW will have the input data that will be written in the register selected by RW.

ALU



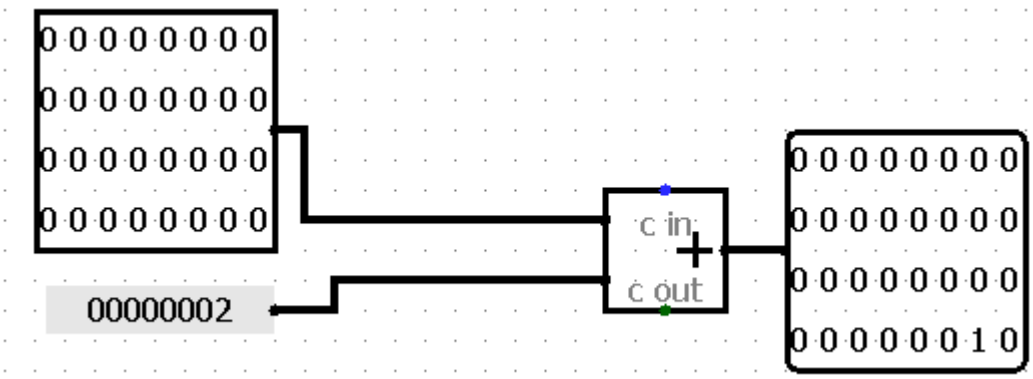
We built the ALU circuit using four main circuits that will be illustrated shortly.
In the ALU, the inputs A, B, Opcode, and ALUSel will be delivered to each component. A MUX will be used to choose which operation to output based on the ALUSel control signal.

Shifter



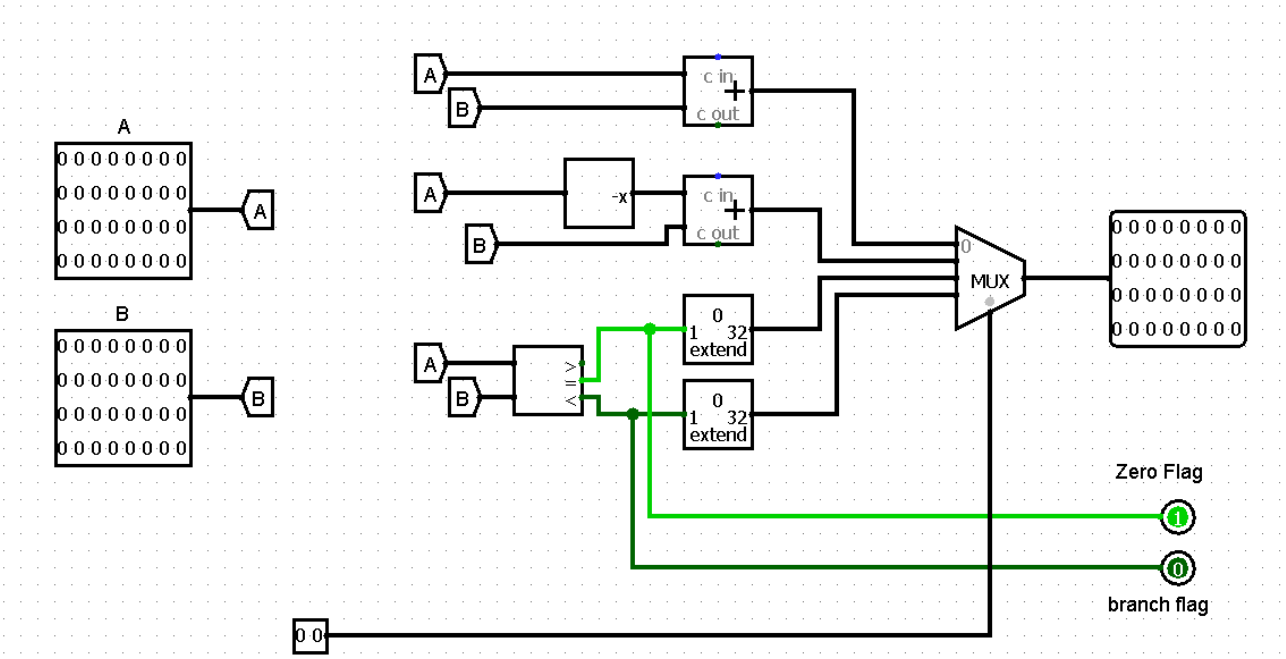
The Job of the Shifter circuit is to execute the shifting operations (SLL , SRL , SRA , ROR).

PC +2



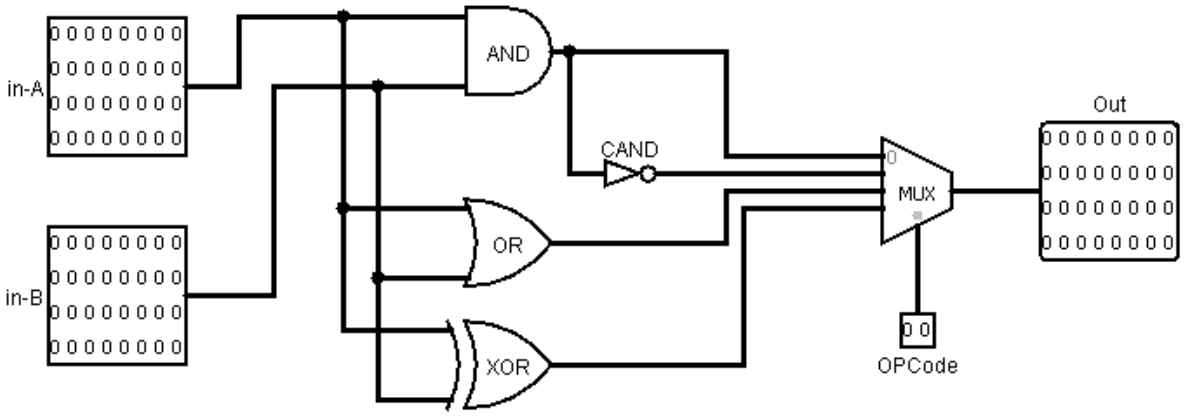
PC +2 Circuit is a simple circuit that increments the PC by 2. This circuit is used by JAL and JALR operations.

Arithmetic Unit



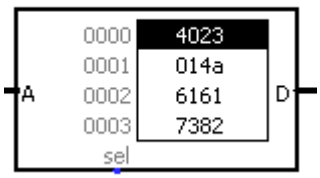
The Arithmetic Unit is concerned with the arithmetic operations that will be selected using a MUX. If the MUX selects 0 or 1, then the operations are ADD or SUB. On the other hand, when the MUX selects 2 or 3, it will execute a comparator. Also, both the Zero Flag and the Branch Flag are used as a control signals for the branching operations.

Logic Unit



The Logic Unit is used for for the logical operations such as AND and OR. A MUX will select the operation according to the provided OPCode.

Instruction Memory

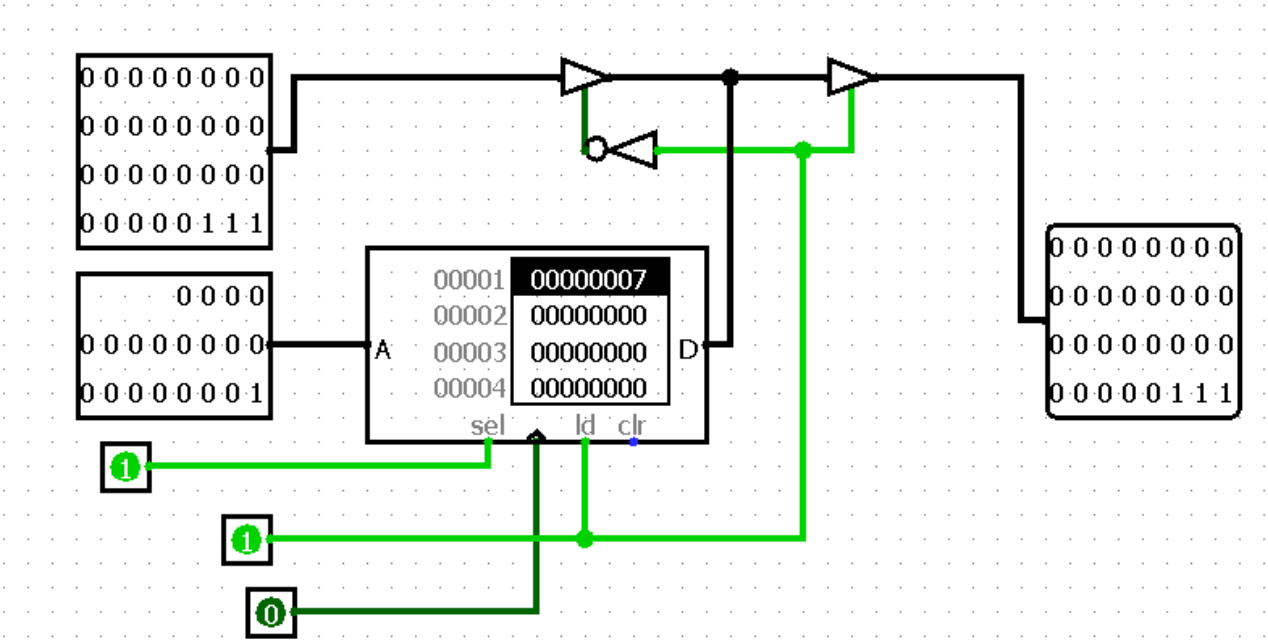


Instruction Memory was implemented as a simple ROM that contains all the necessary instructions to test the CPU.

Below are some of the test cases for the CPU :

Inst. Code	Inst.	Hex
01000 - 000-001-00011	addI R0, R1,3	4023
00000 - 001-010-010-10	or R1, R2, R2	014a
01100 - 001-011-00001	sll R1, R3,1	6161
01110 - 011-100-00010	sra R3, R4,2	7382
01101 - 011-101-00010	srl R3, R5,2	6ba2
01011-011-110-00011	slti R3,R6,3	5bc3
01011-011-111-00111	slti R3,R7,7	5be6
10101-000-001-00000	sw R0 ,R1 , 0	a820
10100-000-011-00000	Lw R0,R3,0	a060
10000-000-000-00010	Beq R0,R0, 2	8002
10010-011-010-00111	Blt R3,R2,7	9347
10010-010-011-00111	Blt R2,R3,7	9267
11100-00000000001	J 1	E001
11011-001-110-00000	JALR R1, R6	D9C0
11101-00000000011	JAL 3	E803

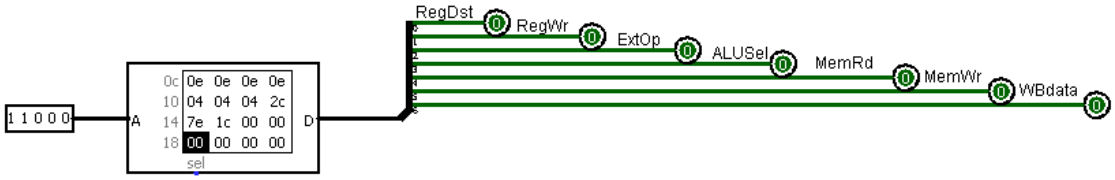
Data Memory



Data Memory was Implemented using a RAM that would be fed the address in the memory and the data that would be inserted in the memory.
Sel is used to enable reading from or writing to the memory.
ld is used to load data to memory.

Control Signals

Main Control

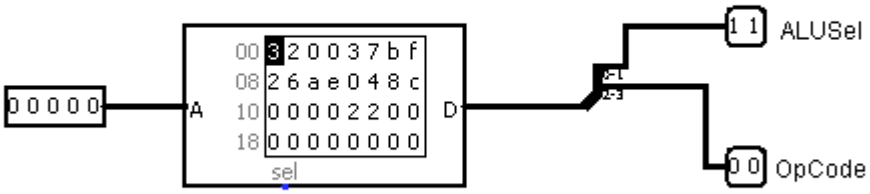


The Main Control Unit will enable signals that are used by the components to reach the correct output according to the table below:

Op	RegDst	RegWr	ExtOp	ALUSrc	EnMem	MemWr/Re	WBdata	Jsel
R-type	1 = Rd	1	X	0 = BusB	0	0	0 = ALU	0
ADDI	0 = Rt	1	1 = sign	1 = Imm	0	0	0 = ALU	0
SLTI	0 = Rt	1	1 = sign	1 = Imm	0	0	0 = ALU	0
ANDI	0 = Rt	1	0 = zero	1 = Imm	0	0	0 = ALU	0
ORI	0 = Rt	1	0 = zero	1 = Imm	0	0	0 = ALU	0
XORI	0 = Rt	1	0 = zero	1 = Imm	0	0	0 = ALU	0
LW	0 = Rt	1	1 = sign	1 = Imm	1	1	1 = Mem	0
SW	X	0	1 = sign	1 = Imm	1	0	X	0
BEQ	X	0	1 = sign	0 = BusB	0	0	X	0
BNE	X	0	1 = sign	0 = BusB	0	0	X	0
BLT	X	0	1	0	0	0	0	0
J	X	0	X	X	0	0	X	0
JAL	1	1	X	X	0	0	X	1
JALR	0	1	0	1 = imm	0	0	X	1

X is a Don’t Care (can be 0 or 1).

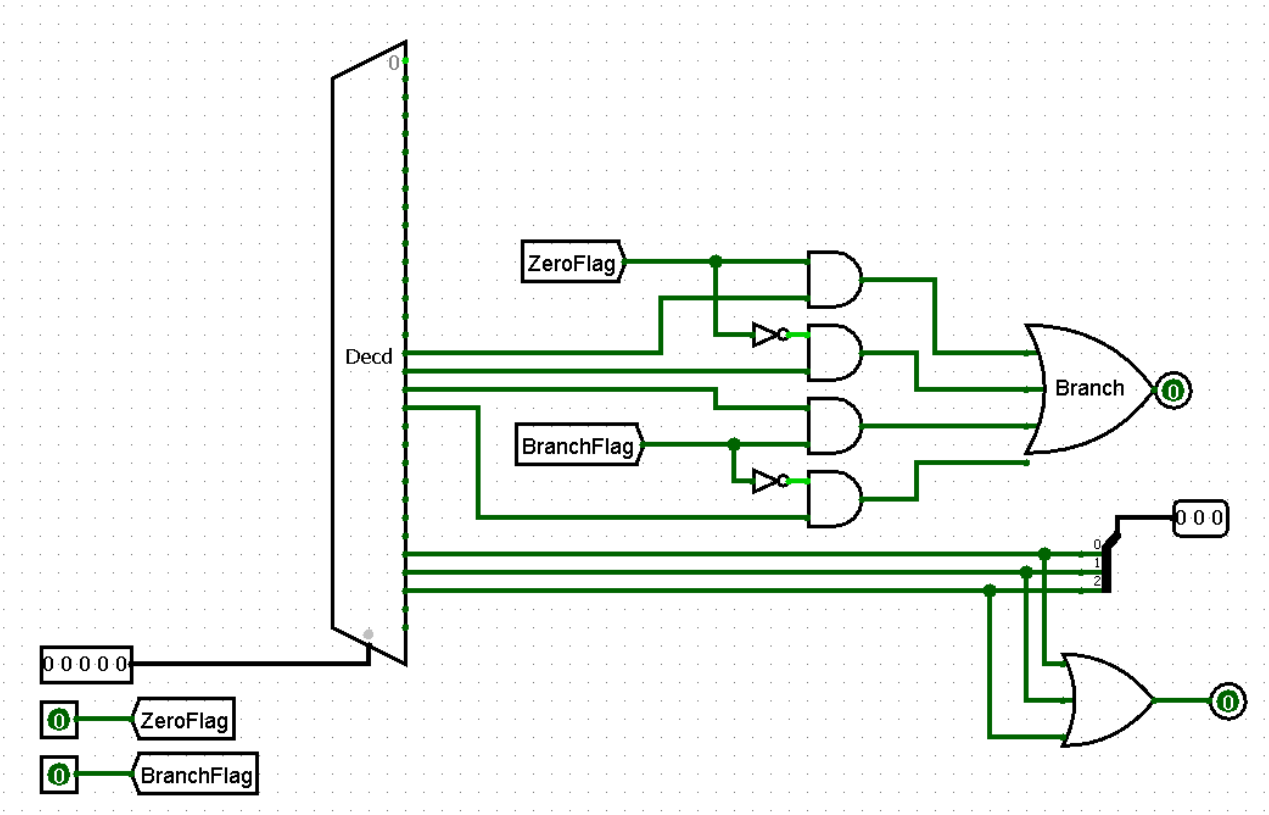
ALU Control



The ALU Control Unit is used to control the signals that are passed to the ALU circuit, and it will control the signals using the Opcode.

OPREATION	OPCODE	OPCODE/ALUSel
AND	00000	XX 11
CAND	00000	XX 11
OR	00000	XX 11
XOR	00000	XX 11
ADD	00001	XX 10
NADD	00001	XX 10
SEQ	00001	XX 10
SLT	00001	XX 10
ANDI	00100	00 11
CANDI	00101	01 11
ORI	00110	10 11
XORI	00111	11 11
ADDI	01000	00 10
NADDI	01001	01 10
SEQI	01010	10 10
SLTI	01011	11 10
SLL	12	00 00
SRL	13	01 00
SRA	14	10 00
ROR	15	11 00
BEQ	10000	01 10
BNE	10001	01 10
BLT	10010	11 10
BGE	10011	11 10
LW	10100	0010
SW	10101	0010
JALR	11011	0001
J	11100	0000
JAL	11101	0001

PC Control



The PC Control Unit is used to control the signals that are passed to the PC circuit, which will determine the operation that will be executed (BEQ , BNE , BLT , BGE , JAL , JALR , J) by using the Opcode.

Problems Faced

We faced a problem in the register file component, that it will update all the registers with a value that was meant to be loaded in one register only. We solved this issue by using AND Gates to enable the register write instead of using an enable in a decoder.

We faced a problem in R-Type instructions which is having the same opcode with different Function codes. We solved it by passing the 2 least significant bits from the instruction code to the ALU Opcode and control its signal using the main control unit

Instruction Encoding

Instruction	Meaning	Encoding				
AND	Rd = Ra & Rb	Op = 0	a	b	d	f = 0
CAND	Rd = ~Ra & Rb	Op = 0	a	b	d	f = 1
OR	Rd = Ra | Rb	Op = 0	a	b	d	f = 2
XOR	Rd = Ra ^ Rb	Op = 0	a	b	d	f = 3
ADD	Rd = Ra + Rb	Op = 1	a	b	d	f = 0
NADD	Rd = –Ra + Rb	Op = 1	a	b	d	f = 1
SEQ	Rd = (Ra == Rb) (result is 0 or 1)	Op = 1	a	b	d	f = 2
SLT	Rd = (Ra < Rb) (result is 0 or 1)	Op = 1	a	b	d	f = 3
ANDI	Rb = Ra & Imm	Op = 4	a	b	Imm5	
CANDI	Rb = ~Ra & Imm	Op = 5	a	b	Imm5	
ORI	Rb = Ra | Imm	Op = 6	a	b	Imm5	
XORI	Rb = Ra ^ Imm	Op = 7	a	b	Imm5	
ADDI	Rb = Ra + Imm	Op = 8	a	b	Imm5	
NADDI	Rd = –Ra + Imm	Op = 9	a	b	Imm5	
SEQI	Rb = (Ra == Imm) (result is 0 or 1)	Op = 10	a	b	Imm5	
SLTI	Rb = (Ra < Imm) (result is 0 or 1)	Op = 11	a	b	Imm5	
SLL	Rb = Shift_Left_Logical(Ra, Imm5)	Op = 12	a	b	Imm5	
SRL	Rb = Shift_Right_Logical(Ra, Imm5)	Op = 13	a	b	Imm5	
SRA	Rb = Shift_Right_Arithmetic(Ra, Imm5)	Op = 14	a	b	Imm5	
ROR	Rb = Rotate_Right(Ra, Imm5)	Op = 15	a	b	Imm5	
BEQ	Branch if (Ra == Rb) (PC += Imm<<1)	Op = 16	a	b	Imm5	
BNE	Branch if (Ra != Rb) (PC += Imm<<1)	Op = 17	a	b	Imm5	
BLT	Branch if (Ra < Rb) (PC += Imm<<1)	Op = 18	a	b	Imm5	
BGE	Branch if (Ra >= Rb) (PC += Imm<<1)	Op = 19	a	b	Imm5	
LW	Rb ←4byte MEM[Ra + Imm]	Op = 20	a	b	Imm5	
SW	MEM[Ra + Imm] ←4byte Rb	Op = 21	a	b	Imm5	
JALR	Rb = PC+2; PC = Ra	Op = 27	a	b	0	
J	PC = PC + signed(Imm11<<1)	Op = 28	Imm11			
JAL	R7 = PC + 2; PC=PC+signed(Imm11<<1)	Op = 29	Imm11			