CO224 –LAB 05
PART 5 – EXTENDED ISA

GROUP 09
DISSANAYAKE D.M.D.R (E/17/072)
DEVINDI G.A.I (E/17/058)

INTRODUCTION

The processor was built for an Instruction Set in which each instruction has a fixed length of 32-bits. The instructions were encoded in the following format.

OP-CODE (bits 31-24)	RD/ IMM (bits 23-16)	RT (bits 15-8)	RS / IMM (bits 7-0)

- Bits (31-24): OP-CODE field identifies the instruction's operation.
- Bits (23-16): A register (RD) to be written to in the register file, or an immediate value (jump or branch target offset).
- Bits (15-8): A register (RT) to be read from in the register file.
- Bits (7-0): A register (RS) to be read from in the register file, or an immediate value (shift value for shift and rotate instructions, immediate value for loading instruction)

The processor designed in lab 05 was built to give support to the following 14 functionalities.

1. LOADI :- load immediate value in bits 7-0 to the RD register
2. MOV :- move value in RS register to RD register
3. ADD :- add the value in RT register to RS register and place the result in RD register
4. SUB :- subtract value in RS register from value in RT register and place the result in RD register
5. AND :- perform bitwise AND on values in RS and RT registers and place the result in RD register
6. OR :- perform bitwise OR on values in RS and RT registers and place the result in RD register
7. J :- jump the number of instructions given in bits 23-16 (RD/IMM field)
8. BEQ :- if values in RT and RS registers are equal, branch the number of instructions given in bits 23-16 (RD/IMM field).
9. BNE :- if values in RT and RS registers are not equal, branch the number of instructions given in bits 23-16(RD/IMM field).
10. MULT :- Multiply the values in RS and RT registers and place the result in RD register
11. SLL :- Logical shift left the value in RT by the value given in bits 7-0 and place the result in RD register
12. SLR :-Logical shift right the value in RT by the value given in bits 7-0 and place the result in RD register
13. SRA :-Arithmetic shift right the value in RT by the value given in bits 7-0 and place the result in RD register
14. ROR :- Rotate right the value in RT by the value given in bits 7-0 and place the result in RD register

Different 8-bit encoding values were given to the OP-CODE in the instruction to identify the above functions. It was decoded inside the control unit of the CPU and generated the 3-bit ALUOP signal to identify the operation to be done inside the ALU.

Table 1: OPCODE of different instructions and their ALUOP Signals

OPCODE	**INSTRUCTION**	**ALUOP**
00000000	loadi	000
00000001	mov	000
00000010	add	001
00000011	sub	001
00000100	and	010
00000101	or	011
00000110	j	000
00000111	beq	001
00001000	bne	001
00001001	mult	100
00001010	sll	101
00001011	srl	110
00001100	sra	110
00001101	ror	111

IMPLEMENTATION OF ADDITIONAL FUNCTIONAL UNITS INSIDE THE ALU

Figure 1 shows the interface of the 8-bit ALU implementation. The values for Operand 1 and 2 are obtained either from registers or taken as immediate values depending on the function performed. The ZERO output port is used to indicate whether the Operands are equal or not.
(If Operand1=Operand2 ZERO=1)

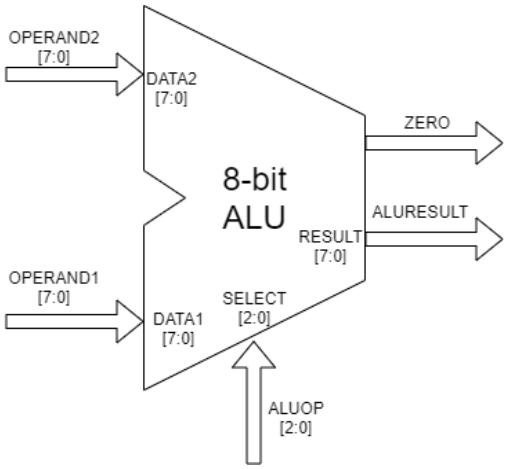


Figure 1: Interface of the ALU

The following table shows the behavior of the ALU with respect to the ALUOP Signal (SELECT)

Table 2: ALU Functional units

SELECT	FUNCTIONAL UNIT	DESCRIPTION	SUPPORTED INSTRUCTION	UNIT DELAY
000	FORWARD	DATA2-> RESULT	Loadi ,mov	#1
001	ADD	DATA1+DATA2 ->RESULT	add,sub,beq,bne	#2
010	AND	DATA1 && DATA2 ->RESULT	and	#1
011	OR	DATA1 | DATA2 ->RESULT	or	#1
100	MULT	DATA1 * DATA2 ->RESULT	mult	#3
101	SHIFT_LEFT	DATA1<<DATA2	sll	#1
110	SHIFT_RIGHT	DATA1>>DATA2	srl,sra	#1
111	ROR	Rotate DATA1 by DATA2 amount	ror	#1

MODIFICATIONS MADE TO THE ALU

LEFT SHIFT MODULE

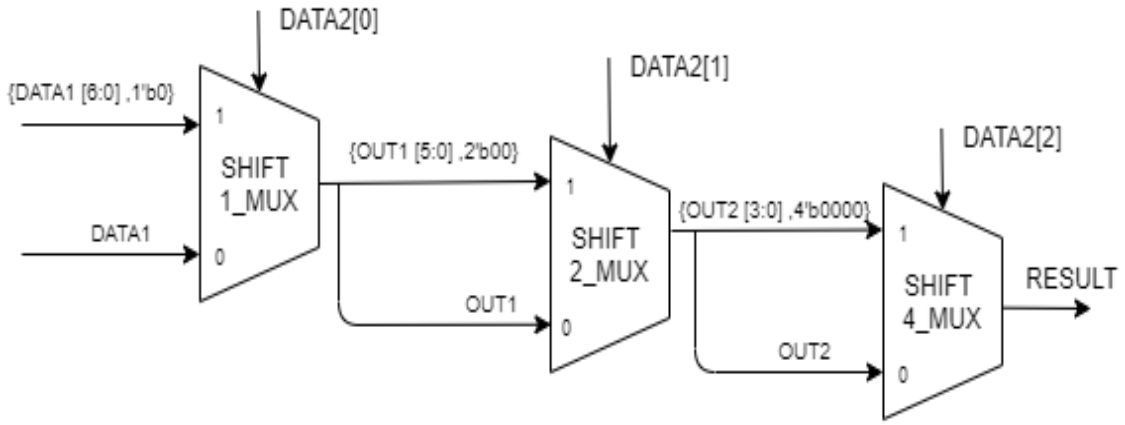


Figure 2: Left shift module

- Left shifting is done inside the ALU using 3 MUX modules as shown in figure 2.
- The binary representation of the shift value given in DATA2 is considered to get the select signals of each MUX.

The shifting procedure is as follows.

1. SHIFT1_MUX takes the DATA2[0] bit (LSB) as the select signal and DATA1 left shifted by 1 and DATA1 as inputs. If,

    a. DATA2[0] = 1, OUT1= {DATA1[6:0],1'b0} (DATA1 left shifted by 1)
    b. DATA2[0] = 0, OUT1= DATA1

2. SHIFT2_MUX takes the DATA2[1] bit as the select signal and OUT1 left shifted by 2 and OUT1 as inputs. If,

    a. DATA2[1] = 1, OUT2= {OUT1[5:0],2'b00} (OUT1 left shifted by 2)
    b. DATA2[1] = 0 , OUT2= OUT1

3. SHIFT4_MUX takes the DATA2[2] bit as the select signal and OUT2 left shifted by 4 and OUT2 as inputs. If,

    a. DATA2[2] = 1, OUT4= {OUT2[3:0],4'b0000} (OUT2 left shifted by 4)
    b. DATA2[2] = 0 , OUT4= OUT2

- Ultimately, this module is capable of performing any shift combination which can be given by 3 bits (Up to 7 shifts) . Since any 8 bit value shifted more than 7 times will result in 0, this module can perform left shift for 8 bit data inputs.
- The latency of this module is 1 time unit.

RIGHT SHIFT MODULE

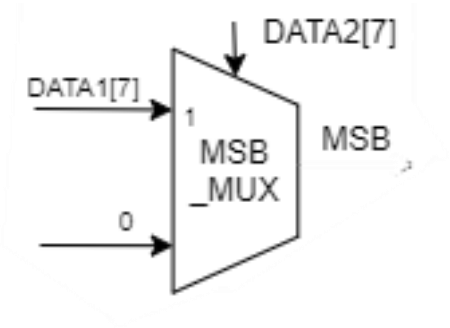


Figure 3: Circuit to get the MSB replacing bit

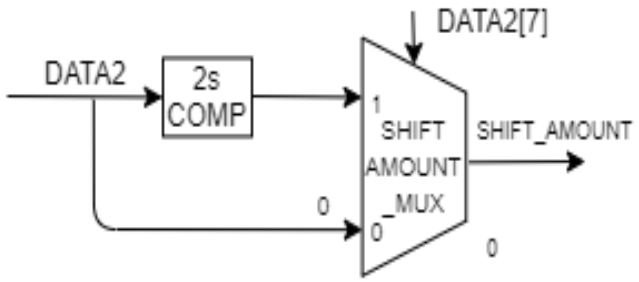


Figure 4: Circuit to get the shift amount

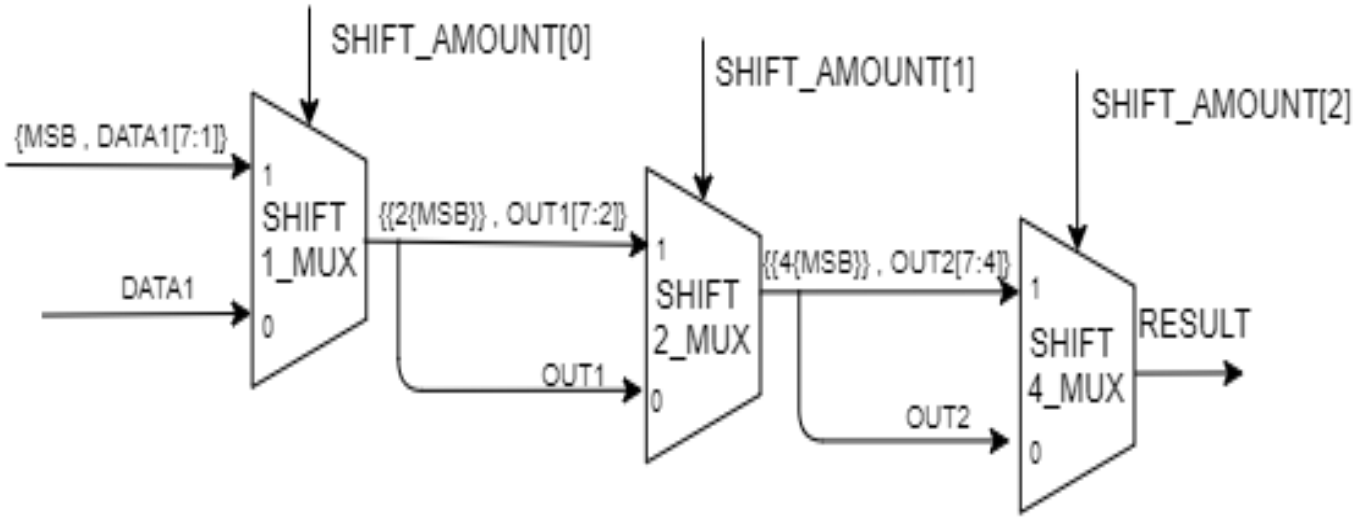


Figure 5: Right shift module

- The right shift module in the ALU is used to perform both arithmetic and logic right shift.
- To identify which right shift to perform, the shift amount given in DATA2 is fed to the ALU as a negative value if the operation to be performed is arithmetic right shift.
- The right shift is done using 5 MUX modules as shown in figures 3,4 and 5.

The shifting procedure is as follows.

1. To obtain the sign of the bit used to fill the vacant bits of the right shifted value, a MUX module represented in figure 3 is used. Since the DATA2 is fed to the ALU as a negative value if the shift is arithmetic, DATA2[7] (MSB) is given as the select signal to the MUX. (DATA2[7] is 1 for arithmetic shift since DATA2 is negative)

    a. If DATA2[7] = 1, MSB = DATA1[7] (for Arithmetic shift)
    b. If DATA2[7] = 0, MSB = 0 (for Logical shift)

2. Since the shift amount (DATA2) is negative for the arithmetic shift, a 2s complement component is used to get the positive value of the shift value if the shifting is arithmetic. The MUX module along with the 2s complement component shown in figure 4 is used for this task. The select signal is DATA2[7].

    a. If DATA2[7] = 1, SHIFT_AMOUNT = ~DATA2+1 (for Arithmetic shift, 1 time unit delay)
    b. If DATA2[7] = 0, SHIFT_AMOUNT = DATA2 (for Logical shift)

3. SHIFT1_MUX takes the SHIFT_AMOUNT [0] bit (LSB) as the select signal and DATA1 right shifted by 1 and DATA1 as inputs. If,
    a. SHIFT_AMOUNT [0] = 1, OUT1= {MSB, DATA1[7:0]} (DATA1 right shifted by 1)
    b. SHIFT_AMOUNT [0] = 0, OUT1= DATA1

4. SHIFT2_MUX takes the SHIFT_AMOUNT [1] bit as the select signal and OUT1 right shifted by 2 and OUT1 as inputs. If,

    a. SHIFT_AMOUNT [1] = 1, OUT2= {{2{MSB}}, OUT1[7:2]} (OUT1 right shifted by 2)
    b. SHIFT_AMOUNT [1] = 0, OUT2= OUT1

5. SHIFT4_MUX takes the SHIFT_AMOUNT [2] bit as the select signal and OUT2 right shifted by 4 and OUT2 as inputs. If,

    a. SHIFT_AMOUNT [2] = 1, OUT4= {{4{MSB}OUT2[7:4],4'b0000} (OUT2 right shifted by 4)
    b. SHIFT_AMOUNT [2] = 0 , OUT4= OUT2

- This module is capable of performing any shift combination which can be given by 3 bits (Up to 7 shifts). Since any 8 bit value shifted more than 7 times will result in 00000000 in logical right shift or 11111111 in arithmetic shift, this module can perform logical or arithmetic right shift for 8 bit data inputs.
- The latency of arithmetic right shift is 2 time units which includes the delay to get the 2s complement of the shift amount.
- The latency of logical right shift is 1 time unit.

ROTATE RIGHT MODULE

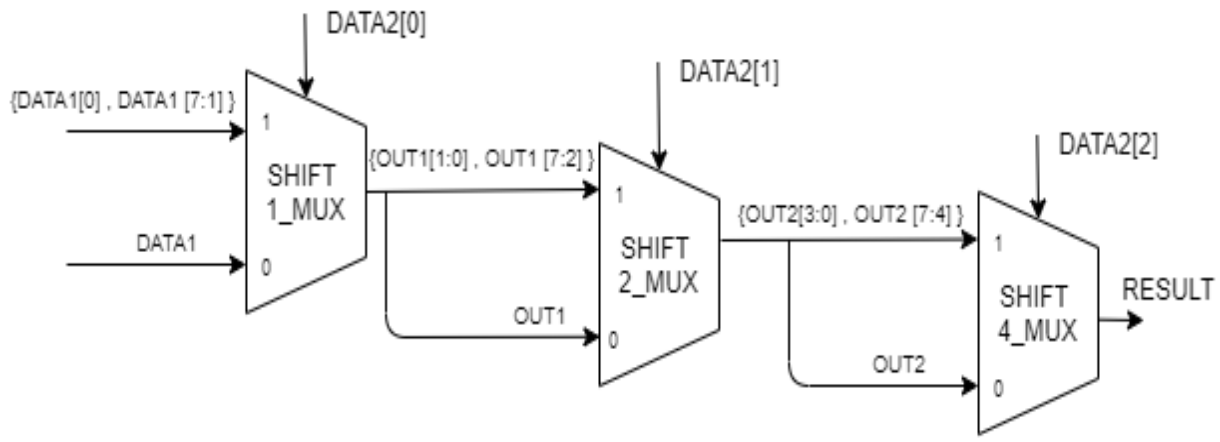


Figure 6: Rotate right module

- Right rotate is done inside the ALU using 3 MUX modules as shown in figure 6.
- The binary representation of the rotate value given in DATA2 is considered to get the select signals of each MUX.

The shifting procedure is as follows.

1. SHIFT1_MUX takes the DATA2[0] bit (LSB) as the select signal and DATA1 rotated by 1 and DATA1 as inputs. If,

    a. DATA2[0] = 1, OUT1= {DATA1[0], DATA1[7:1]} (DATA1 rotated right by 1)
    b. DATA2[0] = 0, OUT1= DATA1

2. SHIFT2_MUX takes the DATA2[1] bit as the select signal and OUT1 rotated by 2 and OUT1 as inputs. If,

    a. DATA2[1] = 1, OUT2= {OUT1[1:0],OUT1[7:2]} (OUT1 rotated right by 2)
    b. DATA2[1] = 0 , OUT2= OUT1

3. SHIFT4_MUX takes the DATA2[2] bit as the select signal and OUT2 rotated by 4 and OUT2 as inputs. If,

    a. DATA2[2] = 1, OUT4= {OUT2[3:0],OUT2[7:4]} (OUT2 rotated right by 4)
    b. DATA2[2] = 0 , OUT4= OUT2

- Since any 8-bit value rotated 8 times will result in the value itself, this module can perform any rotation amount represented by 3 bits.
- The latency is 1 time unit.

MULTIPLICATION MODULE

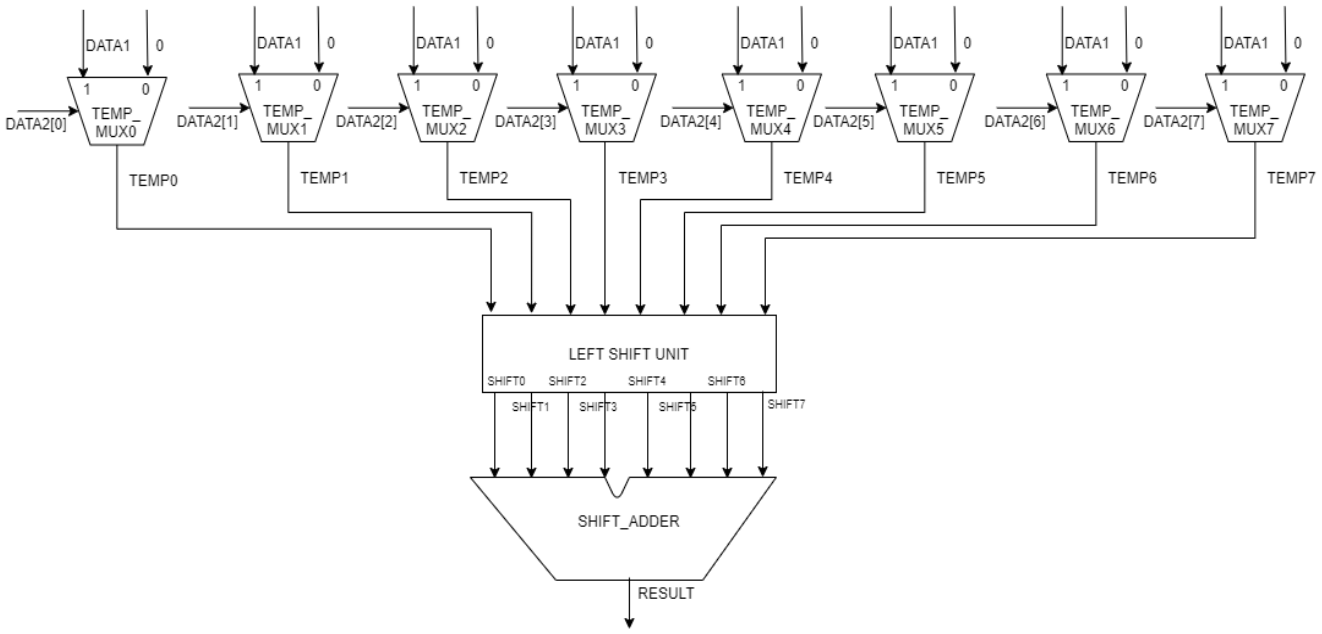


Figure 7: Multiplication module

- The module shown in figure 7 is used to perform the multiplication of DATA1 and DATA2.
- Binary multiplication algorithm is used in the following manner.

1. Consider each bit of the multiplier. If the bit is 1, store the multiplicand in the index relevant to the index of the bit in an array named as temp. If the bit is 0, store 0 in the index relevant to the index of the bit. This is done using 7 MUXs where the select signal for each MUX is a bit of the multiplier, and the inputs are multiplicand and 0.

2. Shift left all the elements in the array by an amount equal to the element index and store each shifted element in an array named shift.

3. Add all the shifted elements together to get the final result.

Eg: 3*4 = ?

3=0011 (Multiplicand)
5=0101 (Multiplier)
temp = [0011,0000,0011,0000]
shift = [0011,0000,1100,0000]
Result =1111 = 15 = 3 * 5

3= 0 0 1 1 ← Multiplicand
5= 0 1 0 1 ← Multiplier

0 0 1 1
0 0 0 0
0 0 1 1
0 0 0 0

1 1 1 1 = 15

- The latency of multiplication is 3 time units which include 1 time unit for left shifting and 2 time units for addition.

OVERVIEW OF THE CPU

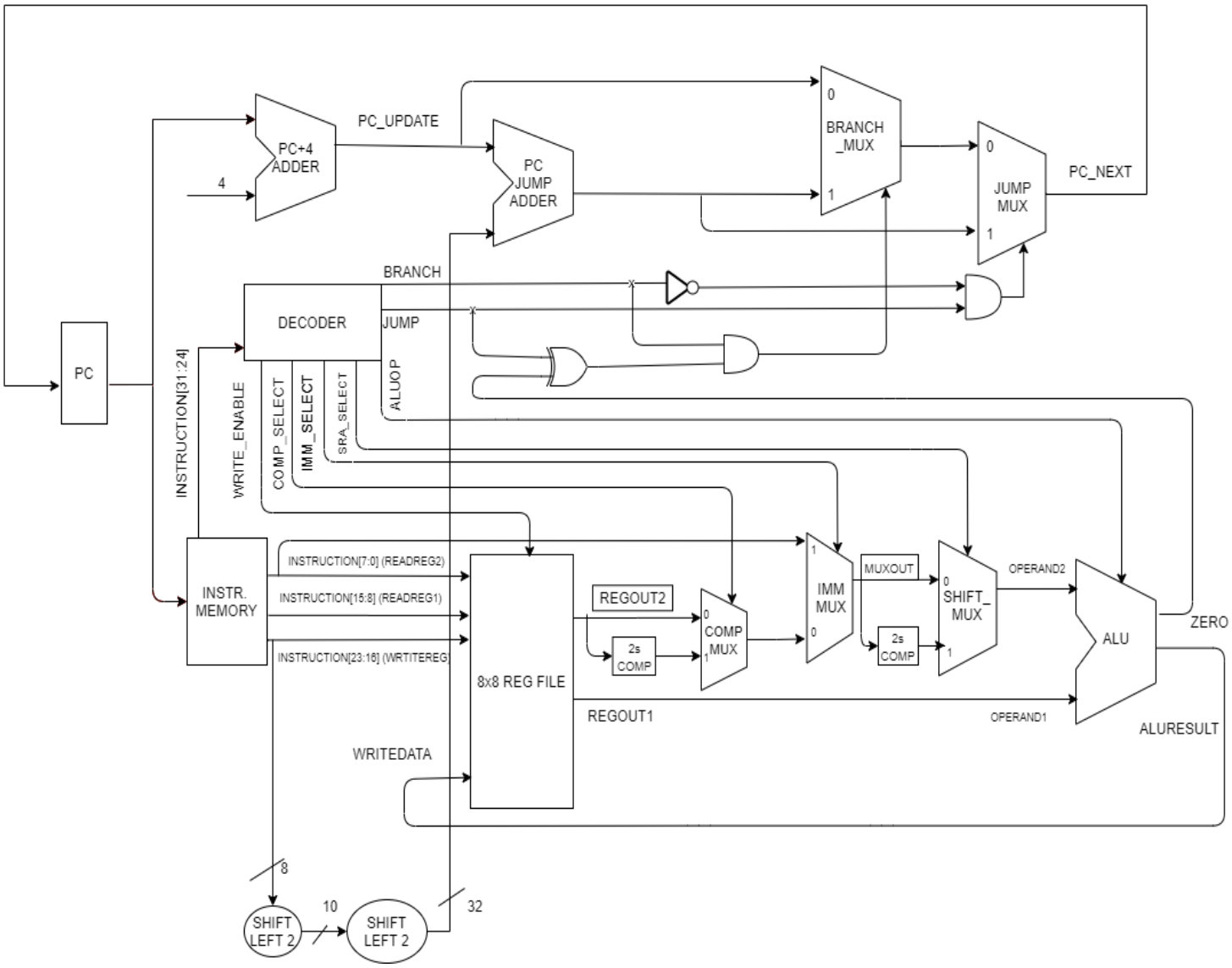


Figure 8: Overview of the modified CPU

According to the OPCODE encoded in the instruction several control signals are generated inside the decoder in the Control Unit. This signal generation consumes a time delay of 1 time unit.

1. WRITE_ENABLE :- Write enable is set to high when the instruction requires writing the result to a destination register.
   When this signal is high, the data available at the WRITEDATA port in the ALU will be written to the register indicated by the WRITEREG at the positive edge of the clock.
   WRITE_ENABLE = 1 for the following instructions.
   - Loadi, mov,add,sub,and,or,mult,sll,srl,sra,ror

   WRITE_ENABLE = 0 for the following instructions.
   - j,beq,bne

2. COMP_SELECT :- This signal is set to high when the instruction requires the complement of operand2 (REGOUT2).
   If this signal is high, the 2s complement of data in the register indicated by Instruction [7:0] will be taken as the Operand2 input of the ALU.
   COMP_SELECT = 1 for the following instructions.
   - Sub,beq,bne

3. IMM_SELECT :- Is set to high when the instruction requires the second operand to be an immediate value.
   When this signal is high, the immediate value indicated by the value in Instruction[7:0] is selected as the Operand2 input to the ALU.
   IMM_SELECT = 1 for the following instructions.
   - Loadi ,sll,srl,sra,ror

4. SRA_SELECT :- Is set to high when the instruction is arithmetic shift right.
   If this signal is high, the immediate value obtained from IMM_MUX will be converted to a negative value using 2s compliment to identify whether the right shift is arithmetic.
   If the Shift_right select signal is given to the ALUOP, the SHIFT_RIGHT functional unit checks whether the MSB of Operand2=1. If it's 1, the unit will perform arithmetic right shift. Otherwise, it will perform a logical right shift.
   SRA_SELECT = 1 for the following instructions.
   - sra

5. BRANCH :- Is set to high when the instruction requires branching to another instruction.
   When BRANCH=1, it will be used to compute relevant select signals used to obtain the next PC value.
   BRANCH = 1 for the following instructions.
   - beq,bne

6. JUMP :- Is set to high when the instruction requires jumping a certain number of instructions.
   JUMP is used to compute the relevant select signals used to obtain the next PC value.
   JUMP = 1 for the following instructions.
   - J,bne

7. ALUOP :- ALUOP relevant to each OPCODE is assigned according to the values shown in table1.

   These decoding signals are used to make decisions along the datapath for each instruction and the final ALU result is obtained with the help of these signals.

DATAPATH AND EXECUTION PROCEDURE OF INSTRUCTIONS

BNE (BRANCH IF NOT EQUAL)

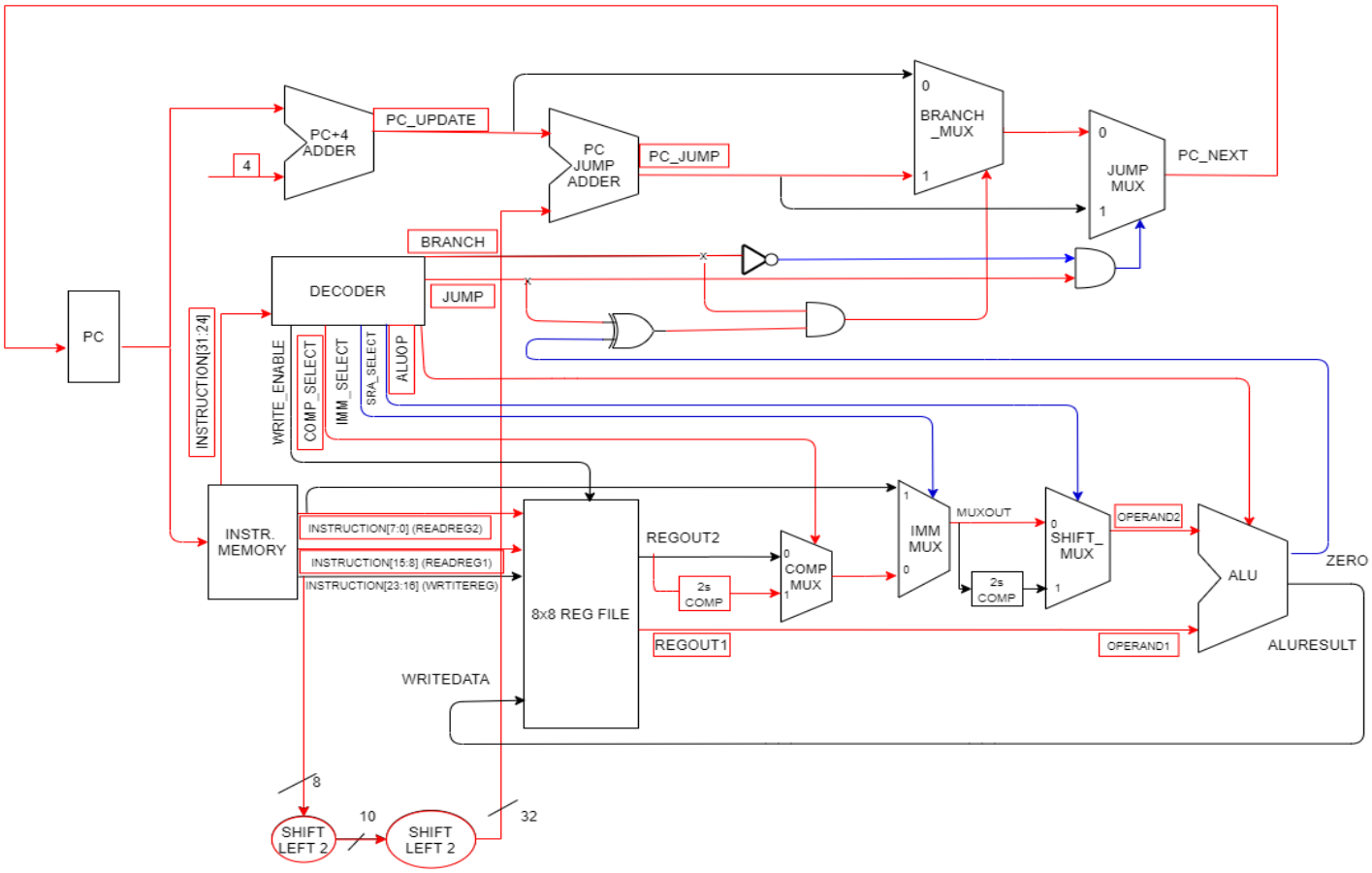


Figure 9: Data path of the bne instruction

PC Update	Instruction Memory Read		Register Read	2's Comp	ALU
#1	#2		#2	#1	#2
	PC+4 Adder		Branch/Jump Target Address		
	#1		#2		
			Decode		
			#1		

Figure 10: Timing details of the bne instruction critical datapath

Instruction format

bne IMM RT RS

Steps of instruction execution

1. After the 2 time unit delay of instruction memory reading, the Instruction [15:0] which contains the addresses of RT and RS registers are fed to the register file.
2. After a 2 time unit delay of register file reading the content of RT and RS registers are available at REGOUT1 and REGOUT2 ports.
3. Parallel to register reading, when the bne instruction OPCODE 00001000 is detected by the decoder it sets the decoding signals to following values after 1 time unit delay.
   ALUOP = 001
   IMM_SELECT = 0
   COMP_SELECT = 1
   BRANCH = 1
   JUMP = 1
   SRA_SELECT =0
   WRITE_ENABLE = 0
4. Parallel to register reading, the PC_JUMP target address is calculated using the current PC+4 (delay is 1 time unit) value and the offset value given in the INSTRUCTION [23:16].
   This takes a delay of 2 time units.
5. To check whether the contents of the registers are equal or not the subtract operation has to be performed. For that purpose, the 2s complement of the REGOUT2 data is obtained after 1 time unit delay and directed to the COMP_MUX.
6. Since the select signal of the mux is set to high at the decoder, the 2s complement of REGOUT2 is taken as the output of COMP_MUX.
7. As the IMM_SELECT and SRA_SELECT signals are 0, the COMP_MUX output is taken as the output of IMM_MUX and SHIFT_MUX as well.
8. As a result, the data stored in RT and RS registers are taken as OPERAND1 and OPERAND2 inputs of the ALU and the ALU performs the sub operation (2 time unit delay) because the ALUOP signal is 001.
9. The values in RS and RT are not equal if the ALURESULT is not 0. This sets the ZERO signal to 0.
10. As ZERO is set to 0 and both JUMP and BRANCH signals are 1, the select signal of the BRANCH_MUX is 1 according to the logic shown in figure 3 and the PC_JUMP value available to the input of this mux is taken as the output.
11. Since both JUMP and BRANCH signals are 1 , the select signal of the JUMP_MUX is set to 0 from the logic provided in figure 3 data path. Therefore. the PC_JUMP value, which is the output of the BRANCH_MUX, is directed to the output of JUMP_MUX.
12. This process will set the PC_NEXT value to the PC_JUMP value calculated from the offset given in the bne instruction.
    If the values in RS and RT registers are equal, the select signal of BRANCH_MUX will be set to 0 and the PC_UDPATE value (PC+4) is directed to the PC_NEXT value.

MULT (MULTIPLICATION)

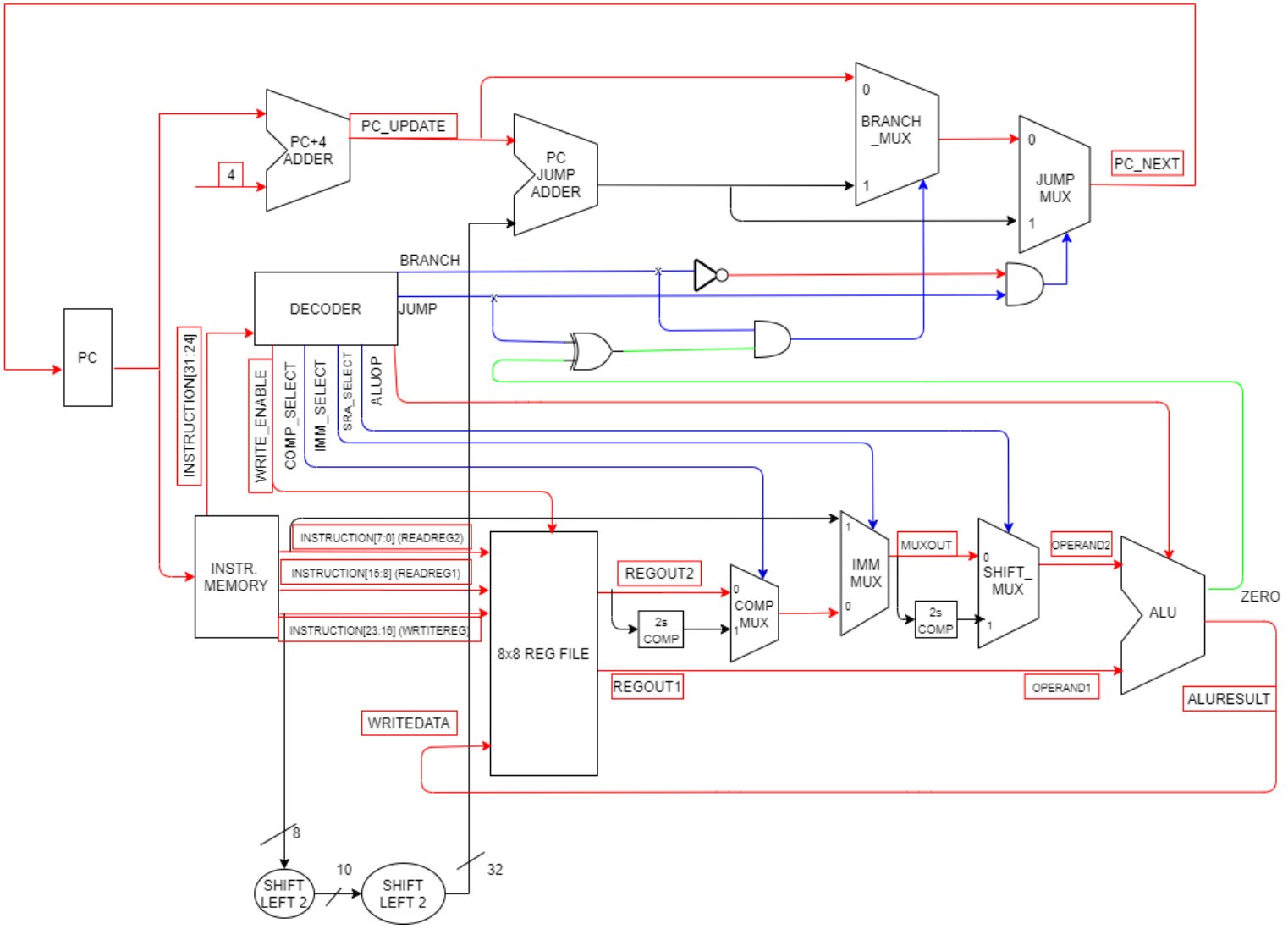


Figure 11: Datapath of the mult instruction

PC Update	Instruction Memory Read		Register Read		ALU #3 shifting(#1), adding(#2)
#1	#2		#2		
	PC+4 Adder		Decode		
	#1		#1		
Register Write					
#1					

Figure 12: Timing details of the mult instruction critical datapath

Instruction format

mult RD RT RS

Steps of instruction execution

1. After the 1 time unit PC delay, the instruction is loaded from the instruction memory with a delay of 2 time units.

2. After a 2 time unit delay of register file reading the content of RT and RS registers are available at REGOUT1 and REGOUT2 ports.

3. Parallel to register reading, when the mult instruction OPCODE 00001001 is detected by the decoder it sets the decoding signals to following values after 1 time unit delay.
   ALUOP = 100
   IMM_SELECT = 0
   COMP_SELECT = 0
   BRANCH = 0
   JUMP = 0
   SRA_SELECT =0
   WRITE_ENABLE = 1

4. Since the select signals of COMP_MUX, IMM_MUX and SHIFT_MUX are 0, the REGOUT2 is directed to the OPERAND2 input of the ALU.

5. The ALU then performs binary multiplication on OPERAND1 and OPERAND2 with 1 time unit delay to left shift the OPERAND1 according to the algorithm and 2 time unit delay to add all the shifted values and get the final multiplication result (details operation performed inside the ALU is give in page 8).

6. After a total of 3 time unit delay the multiplication result is available at ALURESULT and it is directed to the WRITEDATA input of the register file. The multiplication of values in RS and RT will be then written to the RD register at the next positive clock edge with a delay of 1 time unit because the WRITE_ENABLE signal is 1.

- Even though the output of the ZERO signal is indefinite, since BRANCH and JUMP control signals are 0, the PC_UPDATE value is directed to the PC_NEXT wire without considering any jump offsets.

SLL (LOGICAL SHIFT LEFT), SRL (LOGICAL SHIFT RIGHT), ROR (ROTATE RIGHT)

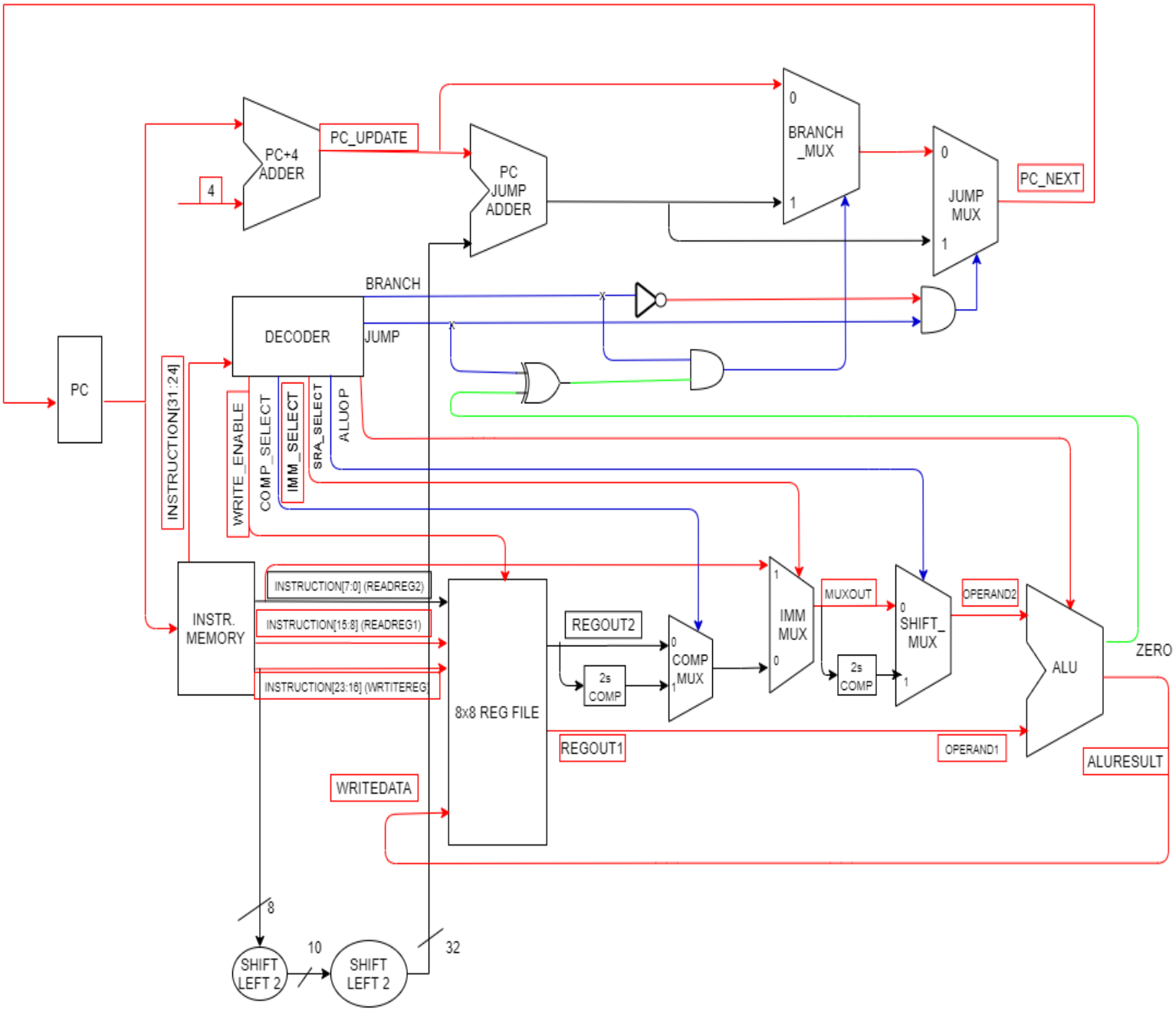


Figure 13: Datapath of the sll,srl,ror instructions

SLL (LOGICAL SHIFT LEFT)

PC Update	Instruction Memory Read		Register Read		ALU(sll)	
#1	#2		#2		#1	
	PC+4 Adder		Decode			
	#1		#1			
Register Write						
#1						

SRL (LOGICAL SHIFT RIGHT)

PC Update	Instruction Memory Read		Register Read		ALU(srl)	
#1	#2		#2		#1	
	PC+4 Adder		Decode			
	#1		#1			
Register Write						
#1						

ROR (ROTATE RIGHT)

PC Update	Instruction Memory Read		Register Read		ALU(ror)	
#1	#2		#2		#1	
	PC+4 Adder		Decode			
	#1		#1			
Register Write						
#1						

Figure 14: Timing details of the sll,srl,ror instruction critical datapath

Instruction format

sll/srl/ror RD RT IMM

Steps of instruction execution

1. After the 1 time unit PC delay, the instruction is loaded from the instruction memory with a delay of 2 time units.

2. After a 2 time unit delay of the register file reading the content of RT is available at REGOUT1 ports.

3. Parallel to register reading, when the sll/srl/ror instruction OPCODEs 00001010 /00001011/ 00001101 are detected by the decoder, it sets the decoding signals to following values after 1 time unit delay.

   ALUOP = 101 (sll) / 110(srl) / 111(ror)
   IMM_SELECT = 1
   COMP_SELECT = 0
   BRANCH = 0
   JUMP = 0
   SRA_SELECT =0
   WRITE_ENABLE = 1

4. Since the select signal of IMM_MUX is set to 1, the immediate value given in Instruction[7:0] is taken as the output of the IMM_MUX after the decoding delay and as SRA_SELECT is set to 0, this immediate value is given as the OPERAND2 to the ALU.

5. The ALU then either logically shifts left, right or rotate right OPERAND1 by the amount indicated by OPERAND2 , depending on the ALUOP and the result is provided to ALURESULT after a delay of 1 time unit.(details of the operations performed inside the ALU are given in pages 4-7)

6. The ALURESULT is then directed to the WRITEDATA input of the register file. The shifted or rotated value of RS register is written to the RD register at the next positive clock edge with a delay of 1 time unit because the WRITE_ENABLE signal is 1.

   - Even though the output of the ZERO signal is indefinite, since BRANCH and JUMP control signals are 0, the PC_UPDATE value is directed to the PC_NEXT wire without considering any jump offsets.

SRA (ARITHMETIC SHIFT RIGHT)

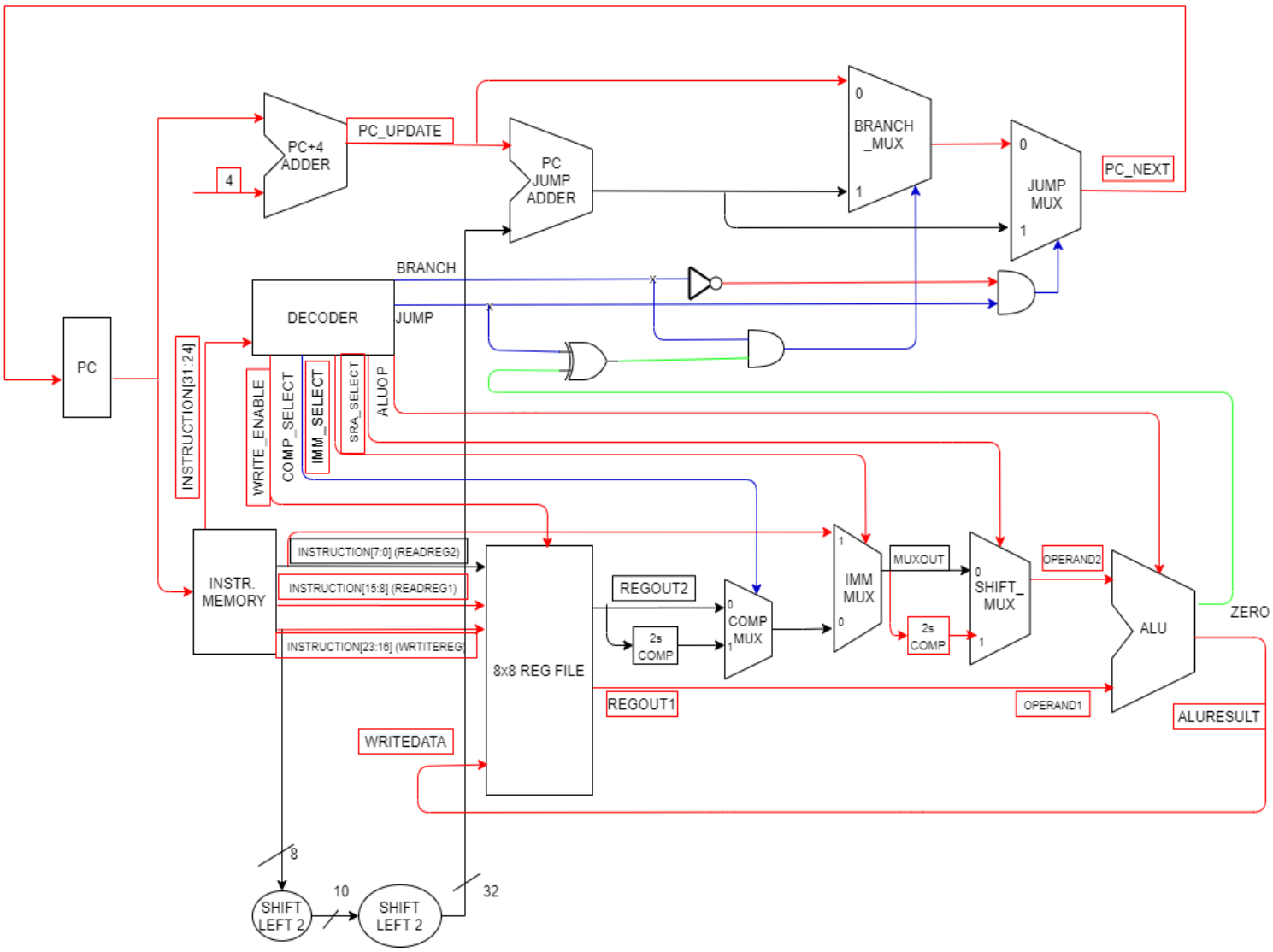


Figure 15: Datapath of the sra instruction

PC Update	Instruction Memory Read		Register Read		ALU #2 2's Comp(#1),Shift(#1)	
#1	#2		#2			
	PC+4 Adder		Decode	2's Comp		
	#1		#1	#1		
Register Write						
#1						

Figure 16:Timing details of the sra instruction critical datapath

Instruction format

sra RD RT IMM

Steps of instruction execution

1. After the 1 time unit PC delay, the instruction is loaded from the instruction memory with a delay of 2 time units.

2. After a 2 time unit delay of the register file reading the content of RT is available at REGOUT1 ports.

3. Parallel to register reading, when the sra instruction OPCODE 00001100 is detected by the decoder, it sets the decoding signals to following values after 1 time unit delay.
    i. ALUOP = 110
    ii. IMM_SELECT = 1
    iii. COMP_SELECT = 0
    iv. BRANCH = 0
    v. JUMP = 0
    vi. SRA_SELECT = 1
    vii. WRITE_ENABLE = 1

4. Since the select signal of IMM_MUX is set to 1, the immediate value given in Instruction[7:0] is taken as the output of the IMM_MUX after the decoding delay

5. Since SRA_SELECT is set to 1, the 2's complement of this immediate value is given as the OPERAND2 to the ALU.

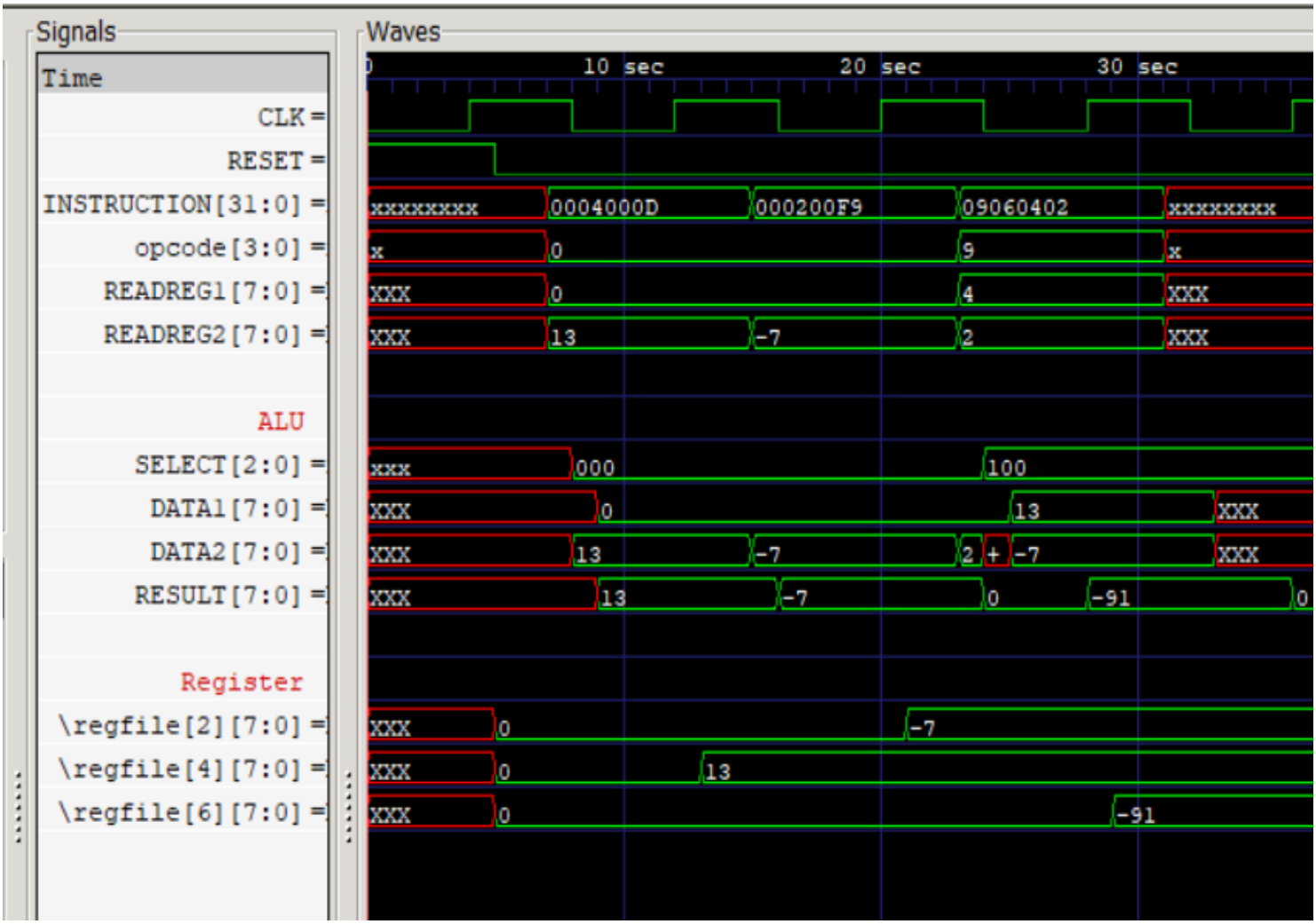
6. The shift right functional unit in the ALU then checks whether the MSB of OPERAND2 is 1. If it is 1, the module gets the positive value represented by the OPERAND2 (OPERAND2 was negated using 2's complement in step 5) which takes a delay of 1 time unit. Then, the functional unit performs arithmetic right shift on OPERAND1 by the positive value of OPERAND1, with a delay of 1 time unit.

7. The ALURESULT is then directed to the WRITEDATA input of the register file. The arithmetic right shift of the value in the RS register is written to the RD register at the next positive clock edge with a delay of 1 time unit because the WRITE_ENABLE signal is 1.

    - Even though the output of the ZERO signal is indefinite, since BRANCH and JUMP control signals are 0, the PC_UPDATE value is directed to the PC_NEXT wire without considering any jump offsets.

APPENDIX

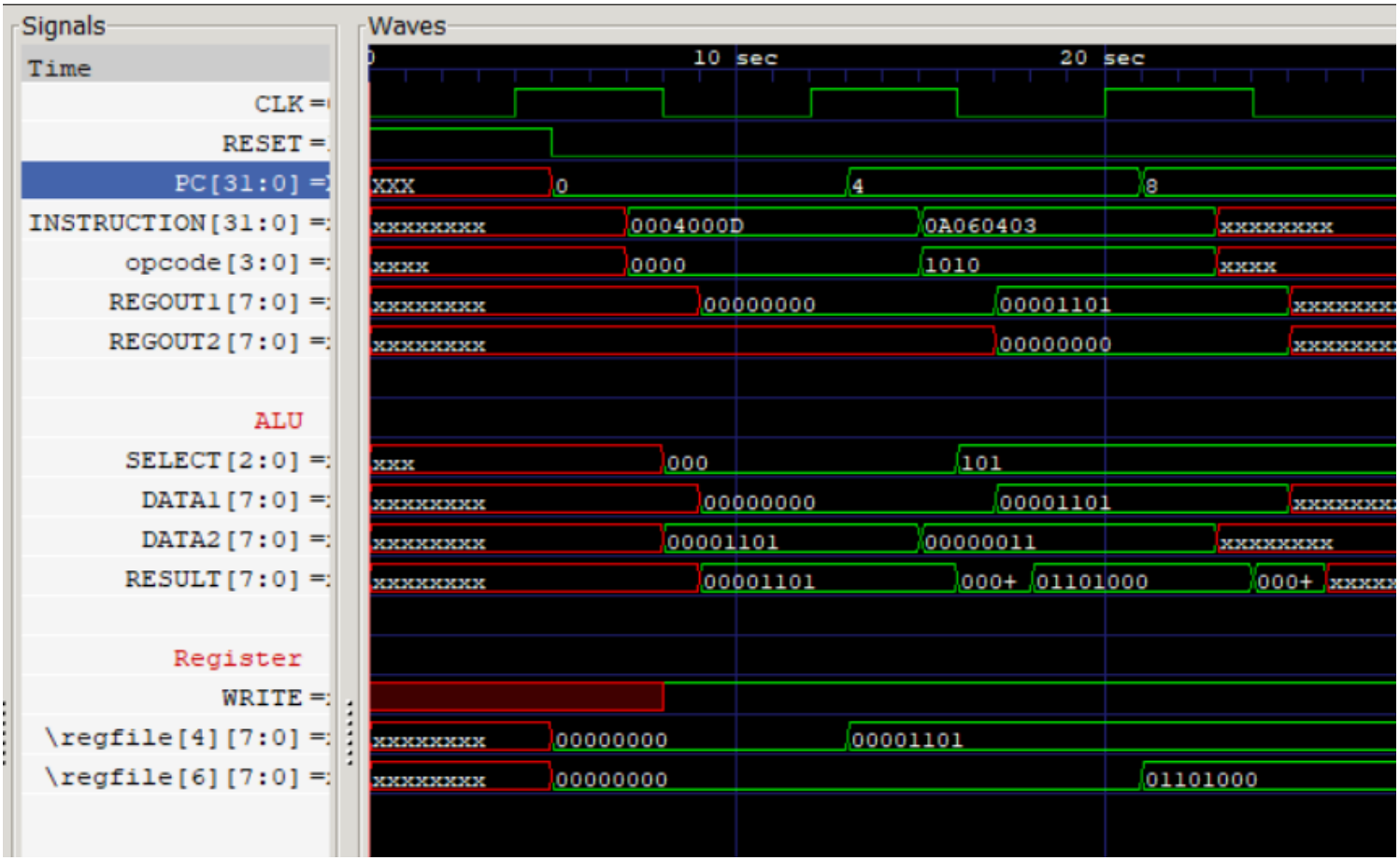
1.Timing diagram for multiplication:-
loadi 4 0x0D
loadi 2 0xF9
mult 6 4 2



2.Timing diagram for logical shift left:-

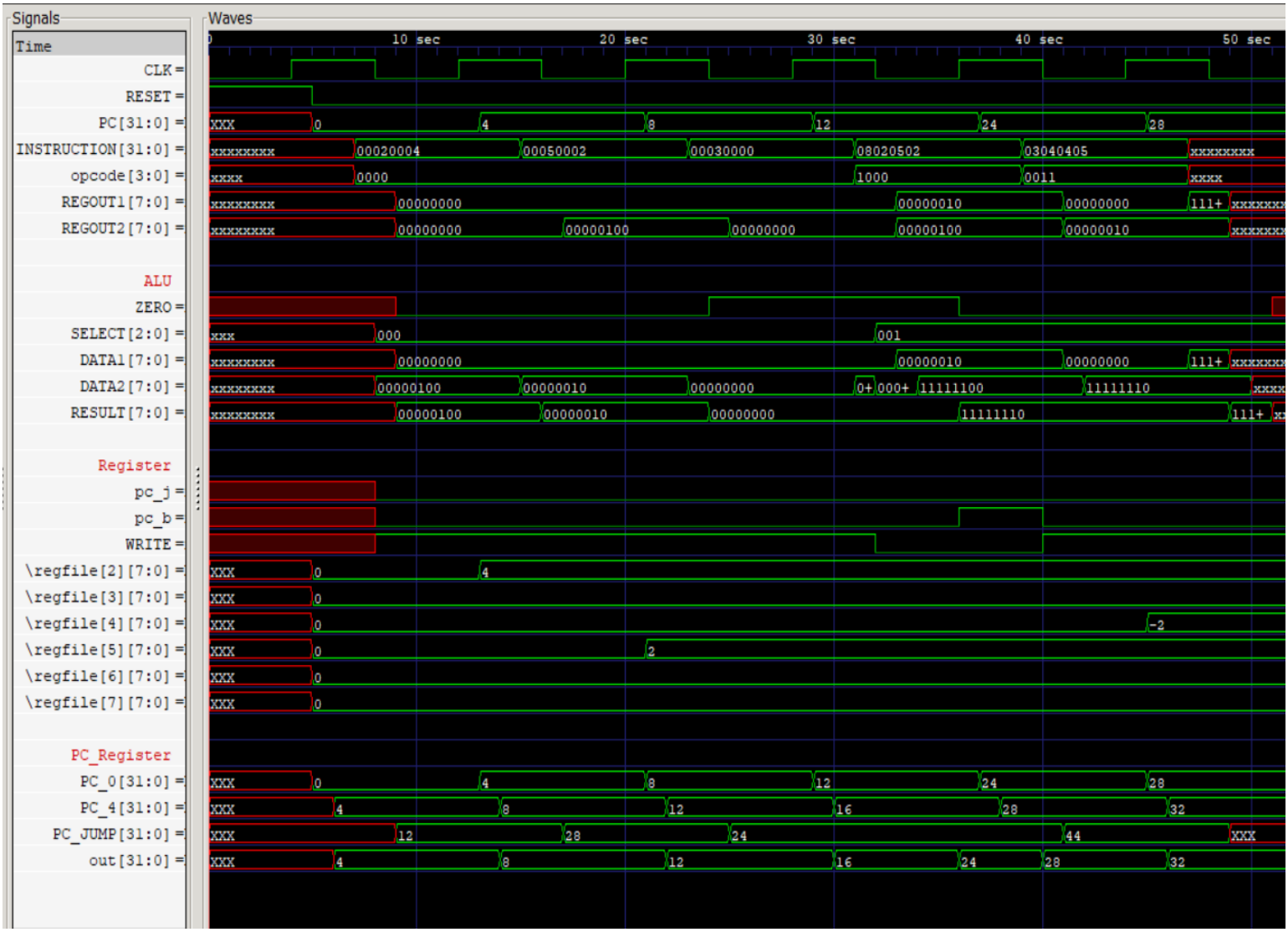
loadi 4 0x0D

sll 6 4 0x03



3.Timing diagram for branch not equal :

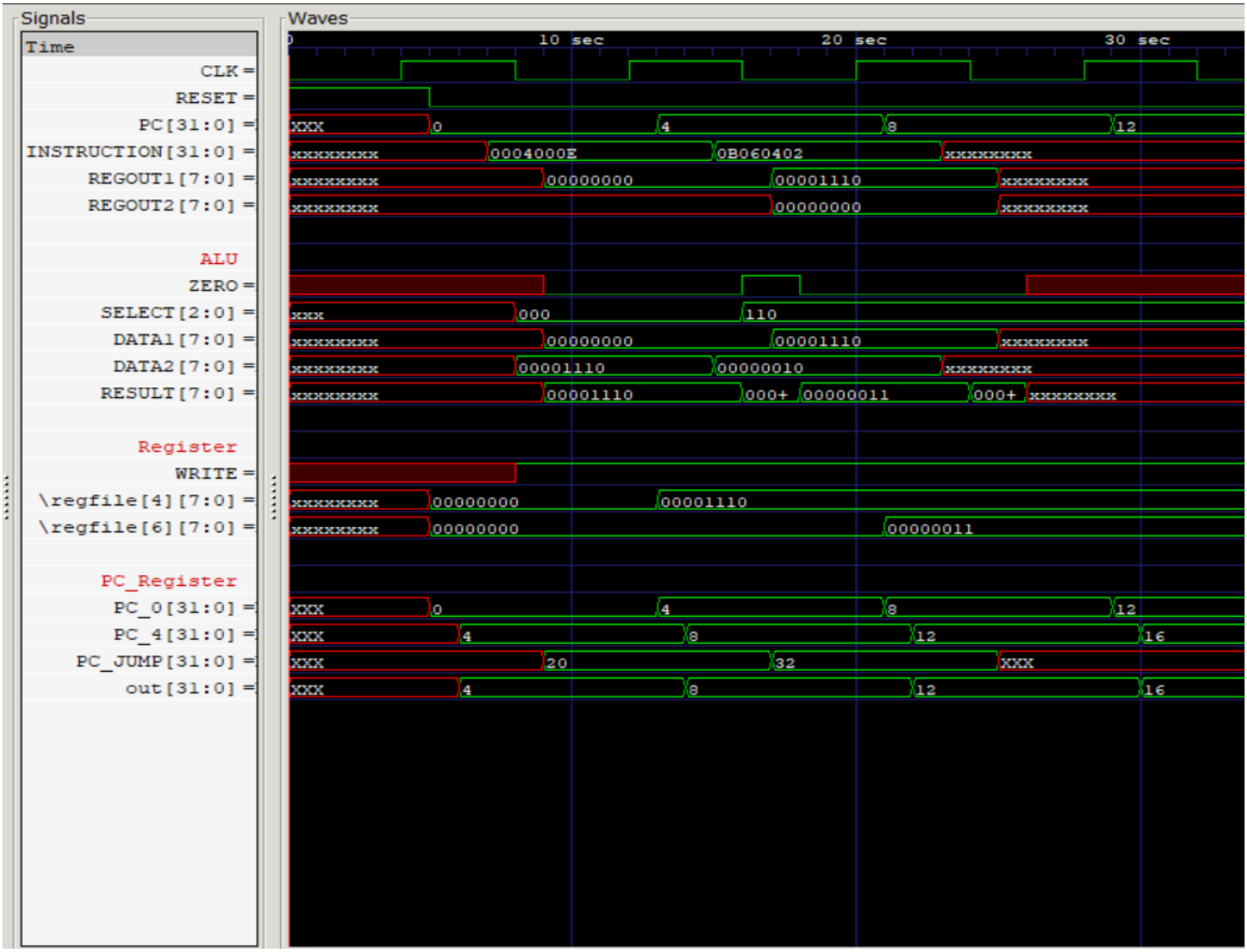
loadi 2 0x04
loadi 5 0x02
loadi 3 0x00
bne 0x02 5 2
loadi 6 0x01
loadi 7 0x09
sub 4 4 5



4.Timing diagram for logical shift right :

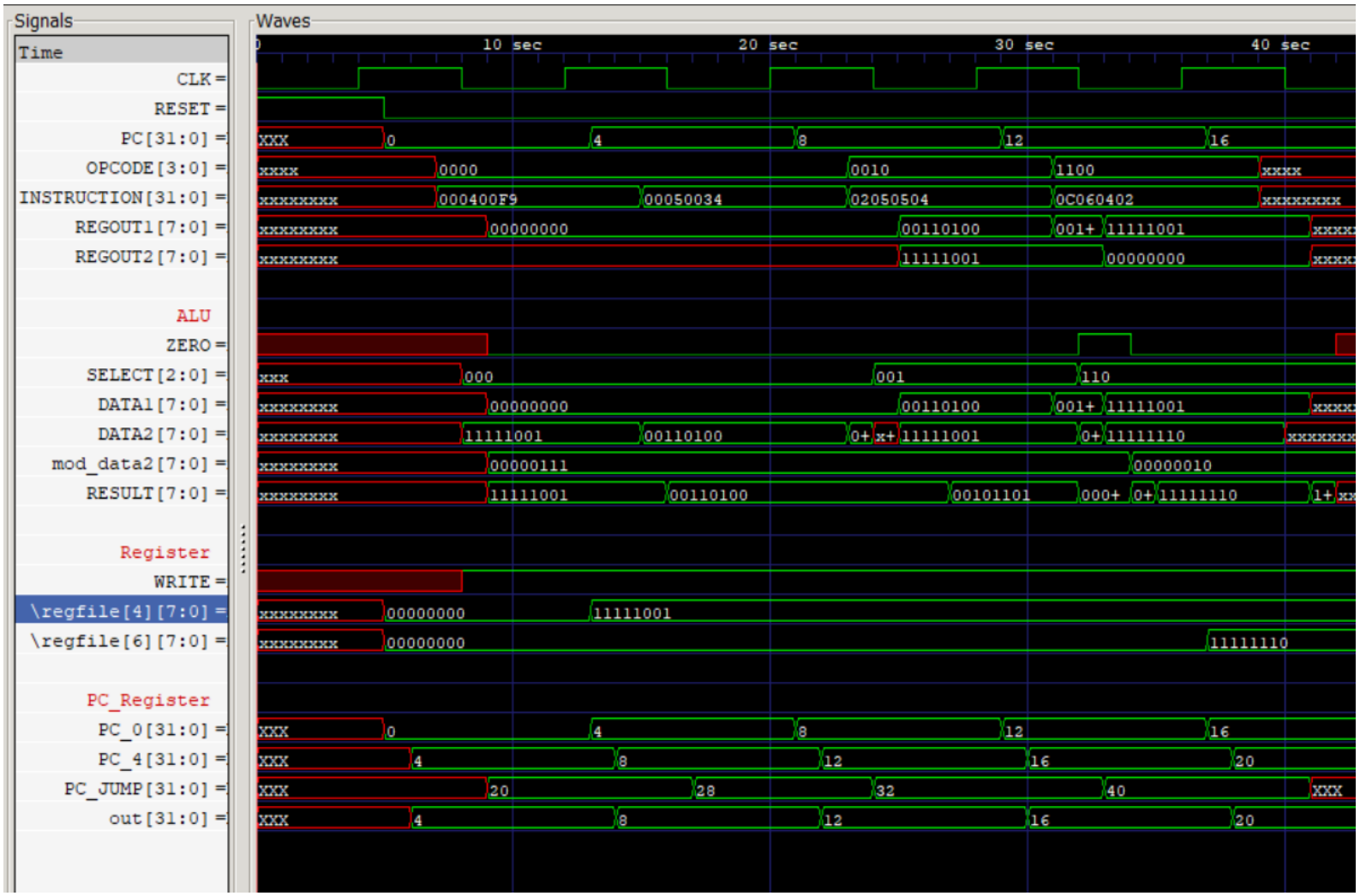
loadi 4 0x0E

srl 6 4 0x02



5.Timing diagram for arithmetic shift right:

loadi 4 0xF9
loadi 5 0x34
add 5 5 4
sra 6 4 0x02



6.Timing diagram for rotate:

loadi 4 0xF7

ror 6 4 0x04