CO224 – Computer Architecture

Group 49 : Nawanjana A.L.D. (E/19/256)

Bandara M.G.R.S. (E/19/038)

Lab 5 – Building a Simple Processor

Part 5 – Extended ISA – Report

In Part 5, the CPU's instruction set was expanded to incorporate six additional instructions. This involved various alterations to the original CPU hardware, along with the implementation of several changes and constraints in the CPU's assembler (assembler.py). The new instructions and their intended applications are outlined as follows.

mult <r1> <r2> <r3> : multiply value in r2 by value in r3, and place the result in r1

sll <r1> <r2> <offset>: apply logical shift left offset times on value in r2, and place the result in r1

srl <r1> <r2> <offset>: apply logical shift right offset times on value in r2, and place the result in r1

sra <r1> <r2> <offset>: apply arithmetic shift right offset times on value in r2, and place the result in r1

ror <r1> <r2> <offset>: apply rotate right offset times on value in r2, and place the result in r1

bne <offset> <r1> <r2>: if values in r1 and r2 are not equal, branch offset times instructions forward

(r1: Register 1, r2: Register 2)

OPCODEs Added to the Assembler File (assembler.py)

```
opcodes = {
    "loadi": "00000000",
    "mov": "00000001",
    "add": "00000010",
    "sub": "00000011",
    "and": "00000100",
    "or": "00000101",
    "j": "00000110",
    "beq": "00000111",
    "bne": "00001000",
    "mult": "00001001",
    "sll":"00001010",
    "srl":"00001011",
    "sra": "00001100",
    "ror": "00001101",
}
```

OPCODEs for Decoding Instructions

Table 4: CPU OPCODEs for Instruction Set

INSTRUCTION	OPCODE
loadi	00000000
mov	00000001
add	00000010
sub	00000011
and	00000100
or	00000101
j	00000110
beq	00000111
bne	00001000
mult	00001001
sll	00001010
srl	00001011
sra	00001100
ror	00001101

To accommodate these new instructions, a minor adjustment was made to the Branch/Jump hardware, and some additional functional units were integrated into the ALU of the CPU. These modifications are depicted in the CPU block diagram and the updated ALU functions table provided below.

CPU Block Diagram

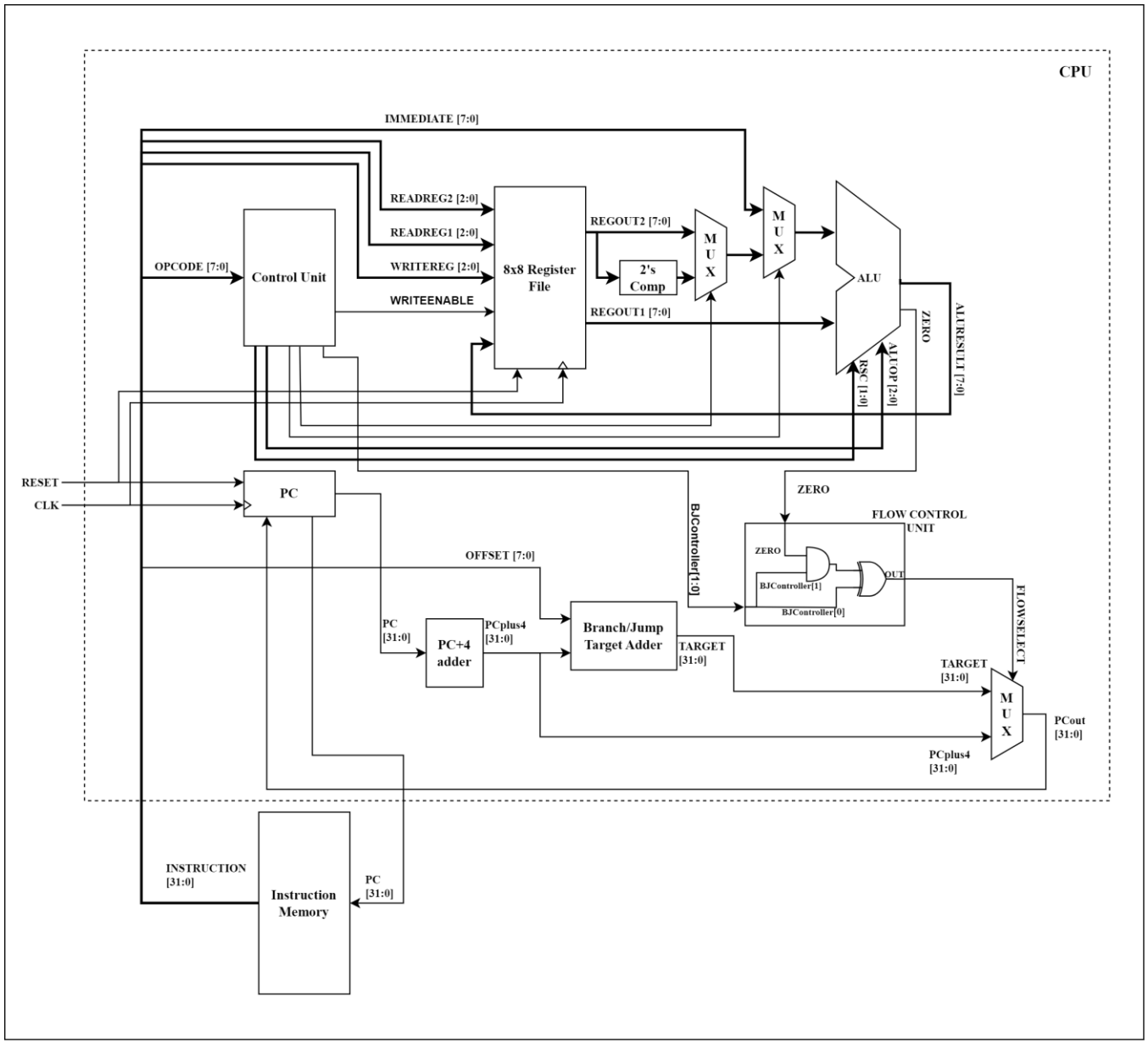


Figure 1: CPU Block Diagram

ALU Unit

Table 1: ALU Functions

SELECT	Function	Description	Supported Instructions	Unit's 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	MULTI	DATA1*DATA2→RESULT	mult	#3
101	LSHIFT	DATA1<<DATA2→RESULT	sll	#2
110	RSHIFT	(Logical Shift) DATA1>>DATA2→RESULT **Or** (Arithmetic Shift) DATA1>>>DATA2→RESULT **Or** Rotate Right DATA1 by DATA2 times (Operation chosen depends on first two MSB bits of DATA2)	srl, sra, ror	#2

MULTI Module

To implement the MULT instruction, a dedicated functional unit was added to the ALU. This unit comprises an array of Full Adders arranged in multiple layers to compute the result for each bit of the output. The Full Adder itself was designed as a module using basic combinational logic. The block diagram for the MULTI functional unit is presented below.

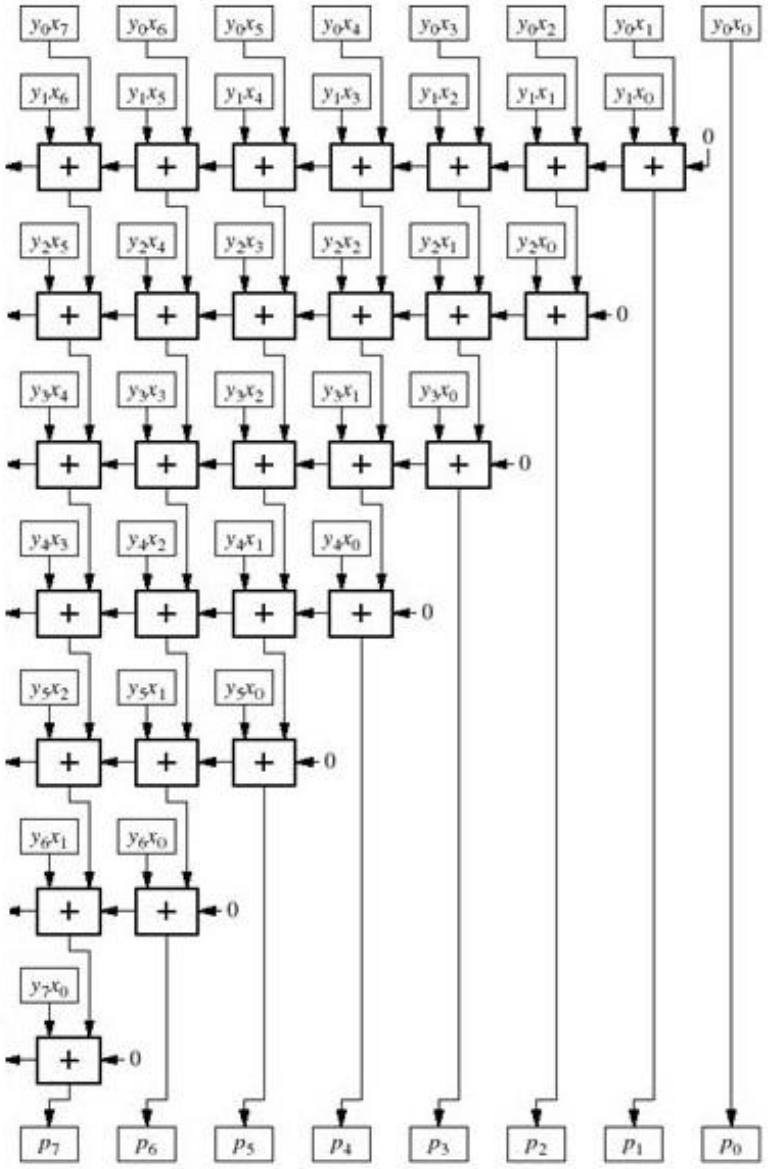


Figure 3: MULTI Functional Unit Block Diagram

The multiplier, similar to the ADD unit, is limited by the requirement to produce an accurate result within 8 bits. Therefore, any multiplication operation yielding a result greater than 255 will produce erroneous outcomes. Due to the MULTI module's reliance on a large array of Full Adders, it is assumed that the calculation takes a significantly longer time compared to other functional units in the ALU. Consequently, a simulation delay of #3 time units was added to this unit.

The timing details for the *MULT* instruction is as follows.

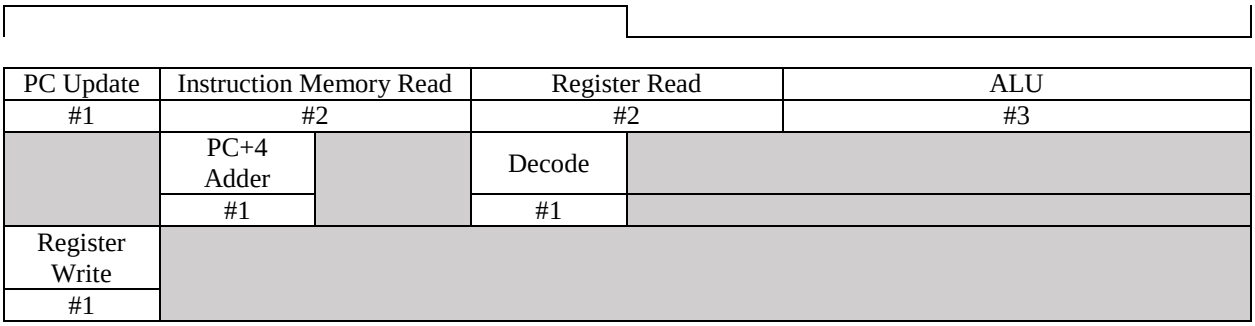


Figure 4: Timing details for MULT instruction datapath

BNE Instruction

As shown in the CPU block diagram, the Flow Control Unit has been updated by replacing the OR gate with an XOR gate to generate the OUT output. Additionally, the BRANCH and JUMP control signals from the Control Unit have been combined into a single 2-bit bus named BJController. These changes were made to support the BNE instruction, which triggers a PC jump to the TARGET value if the ZERO input from the ALU is not HIGH. The unit responds to the control signals from the Control Unit as follows.

Table 2: Behavior of Flow Control Unit

BJController	ZERO	FLOWSELECT	Flow Type
00	x	0	*Normal Flow*
01	x	1	*Jump Flow*
10	0	0	*Normal Flow*
	1	1	*Beq Flow*
11	0	1	*Bne Flow*
	1	0	*Normal Flow*

The behavior of the XOR gate can be utilized to implement the BNE instruction by setting the BJController control signal to 11 for BNE instructions. Therefore, after adding the necessary decoding logic to the Control Unit, no further modifications were required, allowing the BNE instruction to be seamlessly added to the instruction set.

The timing details for the *BNE* instruction is as follows.

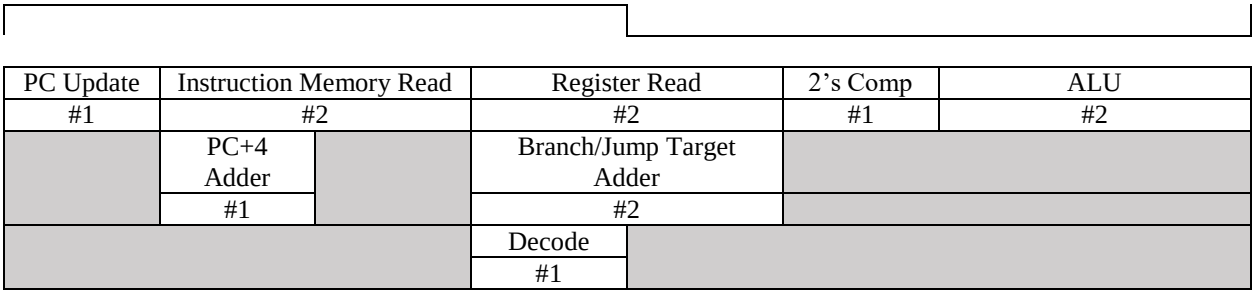


Figure 2: Timing details for BNE instruction datapath

LEFT SHIFT Unit

This functional unit was integrated into the ALU to enable bitwise left shift operations. This unit functions as a barrel shifter, using multiple layers of multiplexers (MUXes) to produce the shifted output. The MUXes were implemented as separate modules, and the block diagram for the LSHIFT functional unit is provided below.

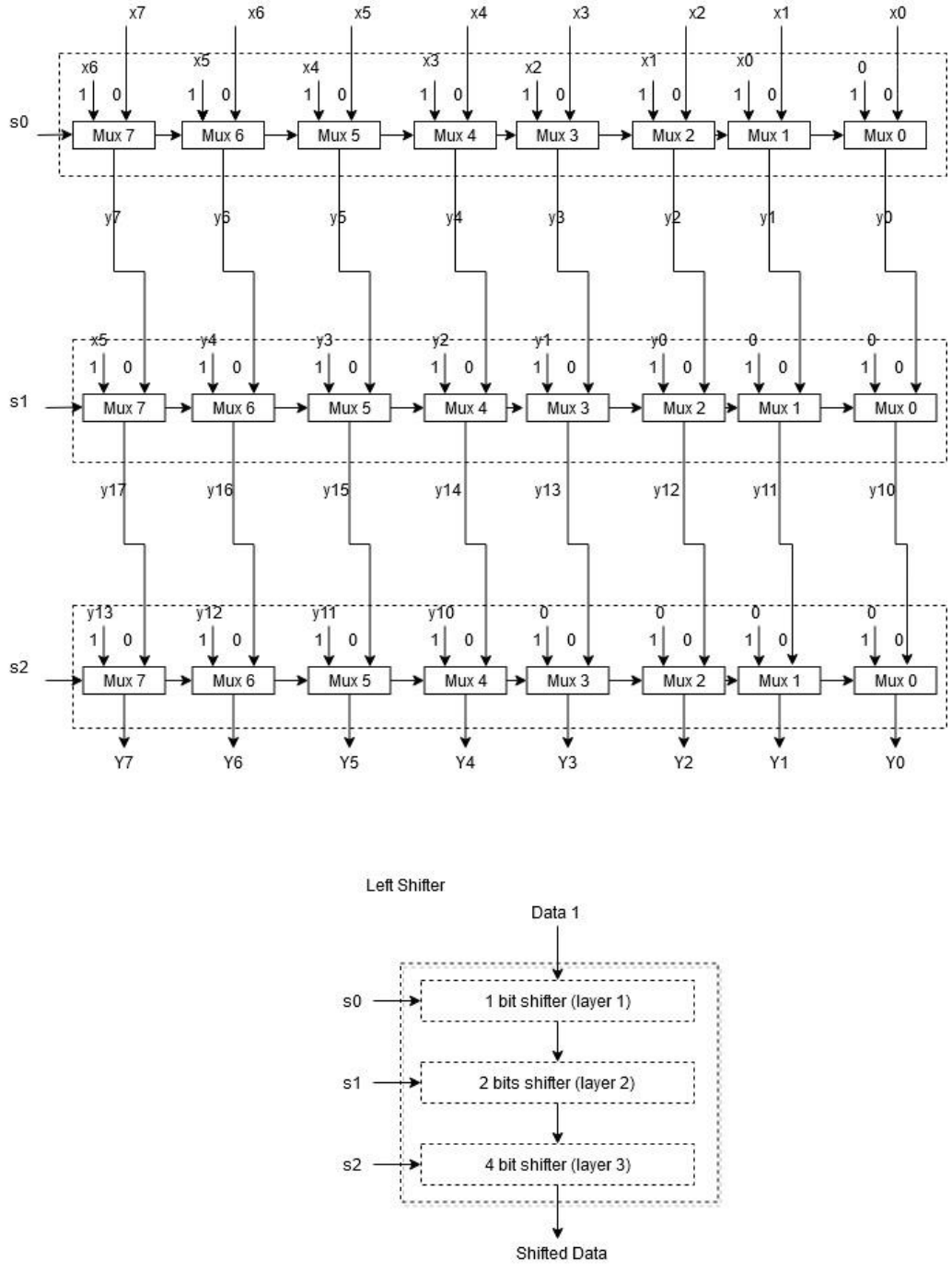


Figure 5: LSHIFT Unit Block Diagram

Since the data word size is 8 bits, shifting by a value greater than 8 produces the same result as shifting by 8. To reduce hardware complexity, the shifter supports only up to 8 shifts. For shifts larger than 8, the assembler detects these values and substitutes them with a shift value of 8, providing the expected behavior while maintaining a simple design. The LSHIFT unit handles one instruction; the SLL instruction. A simulation delay of #2 time units was set for this operation.

The timing details for the *SLL* instruction is as follows.

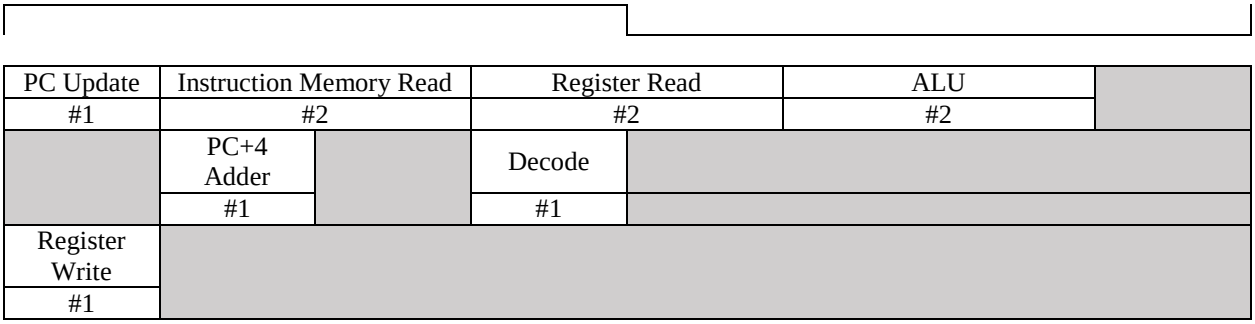


Figure 6: Timing details for SLL instruction datapath

RHIGHT SHIFT Unit

The three right shift instructions were consolidated into a single functional unit called RSHIFT, responsible for managing logic, arithmetic, and rotate right shift operations. This three types of shifting operations contain in 3 different modules. This unit operates similarly to the LSHIFT unit, employing a barrel shifter model.

MUX Layers for Right Shifting Operations

The module utilizes a series of 2x1 multiplexers (muxes) to implement the right shift operations. These muxes are organized into three layers for each type of shift operation (logical, arithmetic, and rotate right). Each layer performs partial shifts based on the corresponding bit in Loaded value (DATA1), enabling efficient calculation of the final shifted value (lOut, aOut or rOut).

1. Logical Right Shift

   In the logical right shift, bits are shifted right, and vacated leftmost bits are filled with zeros. Three layers of 2x1 multiplexers (muxes) perform this operation: the first layer shifts bits based on the least significant bit (LSB) of the shift amount (DATA2), the second layer shifts by up to 2 positions using the next bit of the shift amount (DATA2), and the third layer shifts by up to 4 positions, completing the logical shift.

2. Arithmetic Right Shift

   The arithmetic right shift also shifts bits right but preserves the sign by filling vacated leftmost bits with the original sign bit (MSB). The operation uses three mux layers: the first layer shifts bits based on the LSB of the shift amount (DATA2), the second layer allows shifts by up to 2 positions, and the third layer shifts by up to 4 positions, using the MSB to fill vacated bits if the original MSB was 1, ensuring the sign is maintained.

3. Rotate Right Shift

   In the rotate right shift, bits are circularly shifted right, with rightmost bits wrapping around to the leftmost positions. The first layer of muxes shifts bits based on the LSB of the shift amount (DATA2), the second layer shifts by up to 2 positions, and the third layer shifts by up to 4 positions, achieving the rotation by wrapping bits around to the start of the data.

Below is the block diagram for the Logical Right Shifting Unit.

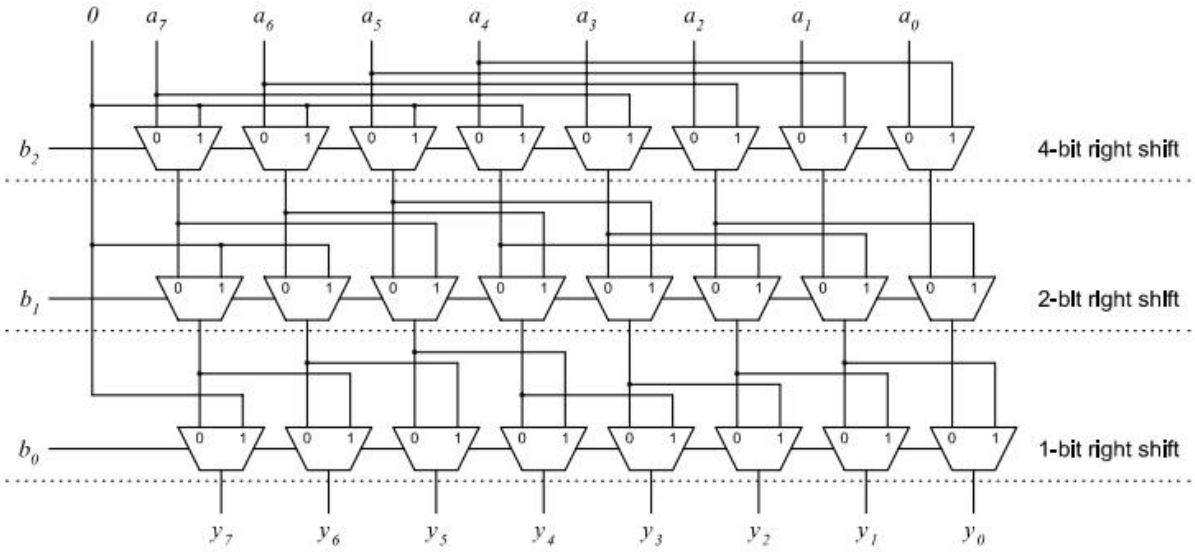


Figure 7: Logical Right Shifting Unit Block Diagram

Control Logic

An always block in the module determines which shift operation to perform based on the 2-bit RSC control signal.

Table 3: Behavior of Right Shift Control Signal

RSC	**Operation Performed**
00	*Other Operations*
01	*Logical Right Shift*
10	*Arithmetic Right Shift*
11	*Rotate Right*

For the arithmetic right shift, additional logic ensures that if the MSB of Loaded value (DATA1) is 1, indicating a negative number, the output bits are appropriately set to maintain the sign. If the shift amount (DATA2) is 8 or greater, the output is either all ones or all zeros depending on the MSB of Loaded value (DATA1). For the Logic right shift, additional logic ensures that if the shift amount (DATA2) is 8 or greater, the output is either all zeros due to the logic shift operation.

The RSHIFT unit has been given a simulation delay of #2 time units. Then, the timing details for the three instructions is as follows.

The timing details for the *srl*/*sra*/*ror* instruction is as follows.

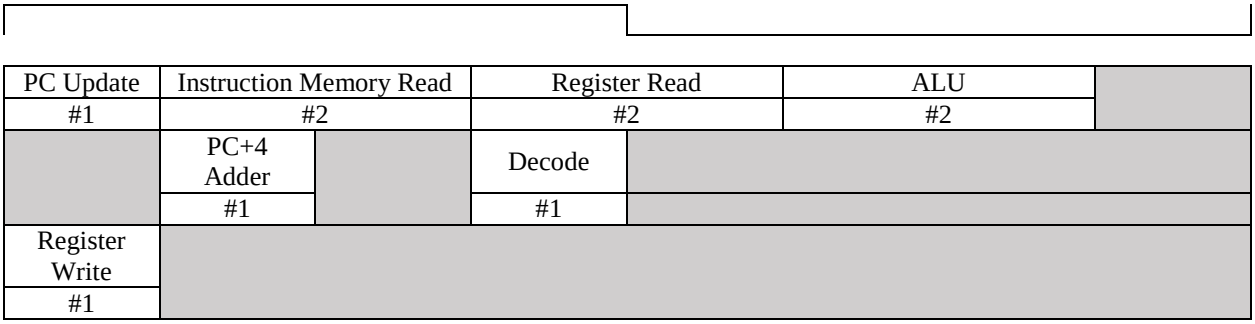


Figure 8: Timing details for right shift instruction datapath

Timing Diagrams

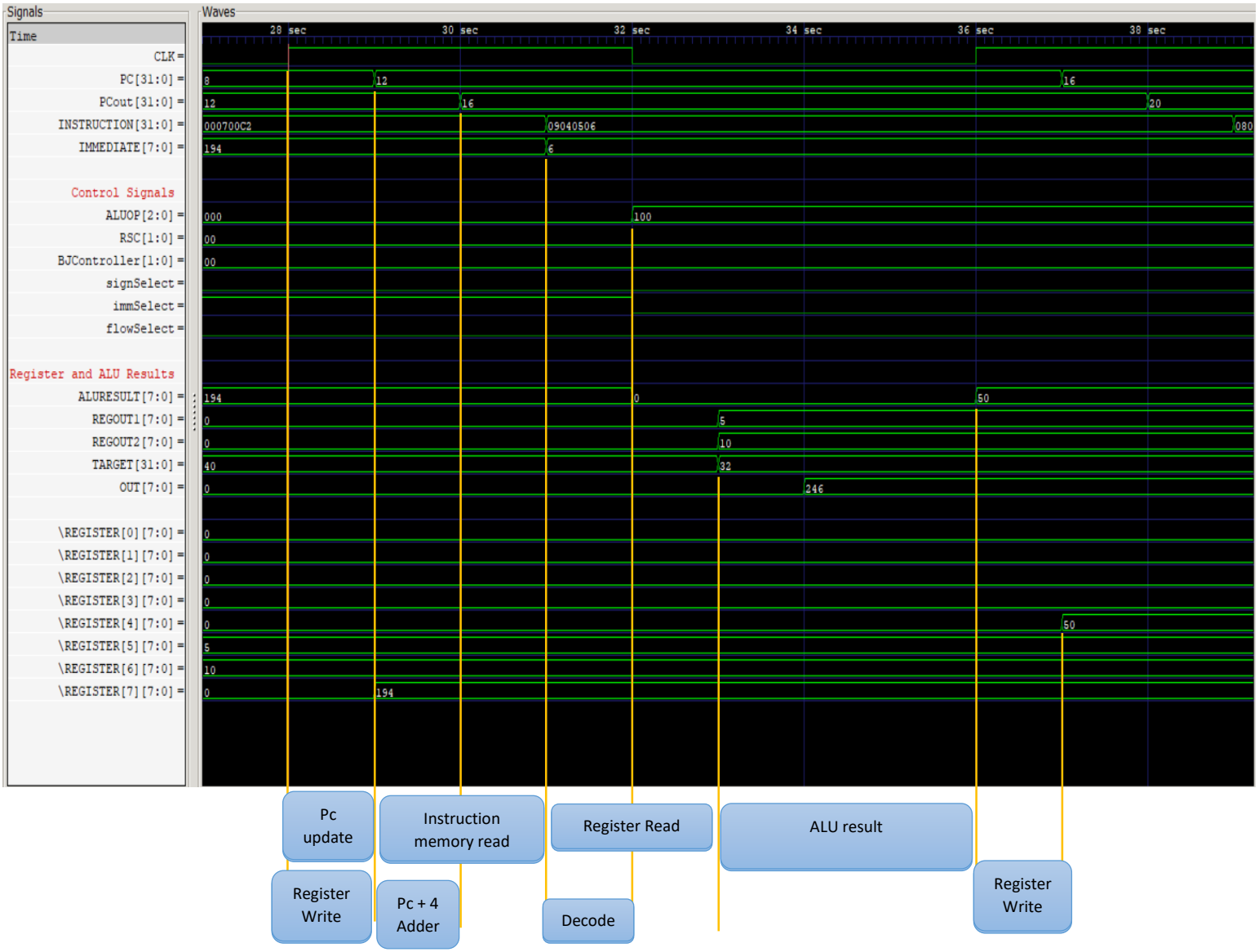
my_sample_program.s

```
1   //This is a sample assembly program for CO224 Lab 5
2   loadi 5 0x05
3   loadi 6 0x0A
4   loadi 7 0xC2
5   mult 4 5 6
6   bne 0x01 5 6
7   loadi 4 0x07
8   sll 3 7 0x03
9   srl 2 7 0x04
10  sra 1 7 0x05
11  ror 0 7 0x06
12
```

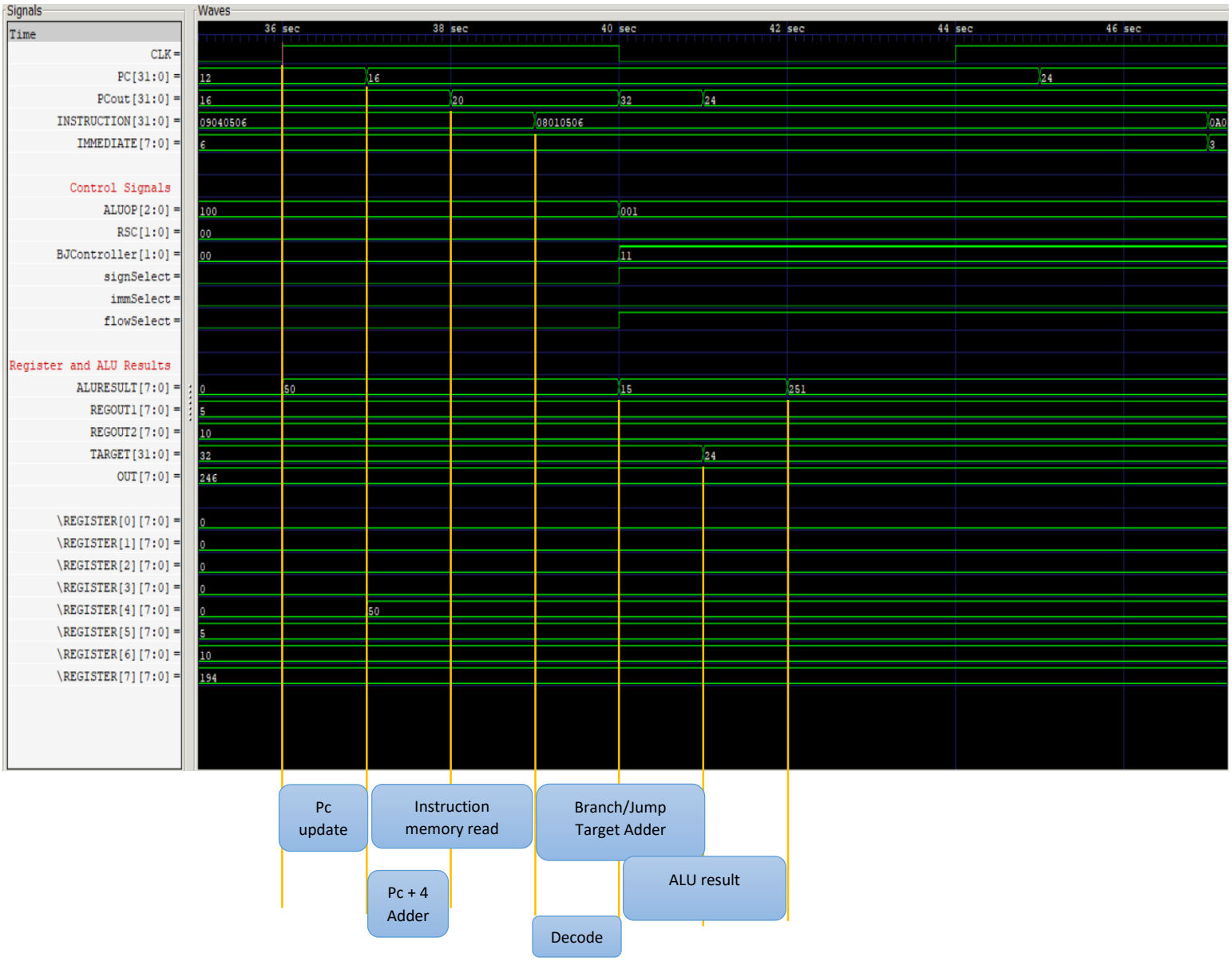
```
        ================================================
         Change of register Content Starting from Time #5
        ================================================

time    reg0    reg1    reg2    reg3    reg4    reg5    reg6    reg7
--------------------------------------------------------------------
   5       0       0       0       0       0       0       0       0
  13       0       0       0       0       0       5       0       0
  21       0       0       0       0       0       5      10       0
  29       0       0       0       0       0       5      10     194
  37       0       0       0       0      50       5      10     194
  53       0       0       0      16      50       5      10     194
  61       0       0      12      16      50       5      10     194
  69       0     254      12      16      50       5      10     194
  77      11     254      12      16      50       5      10     194
```

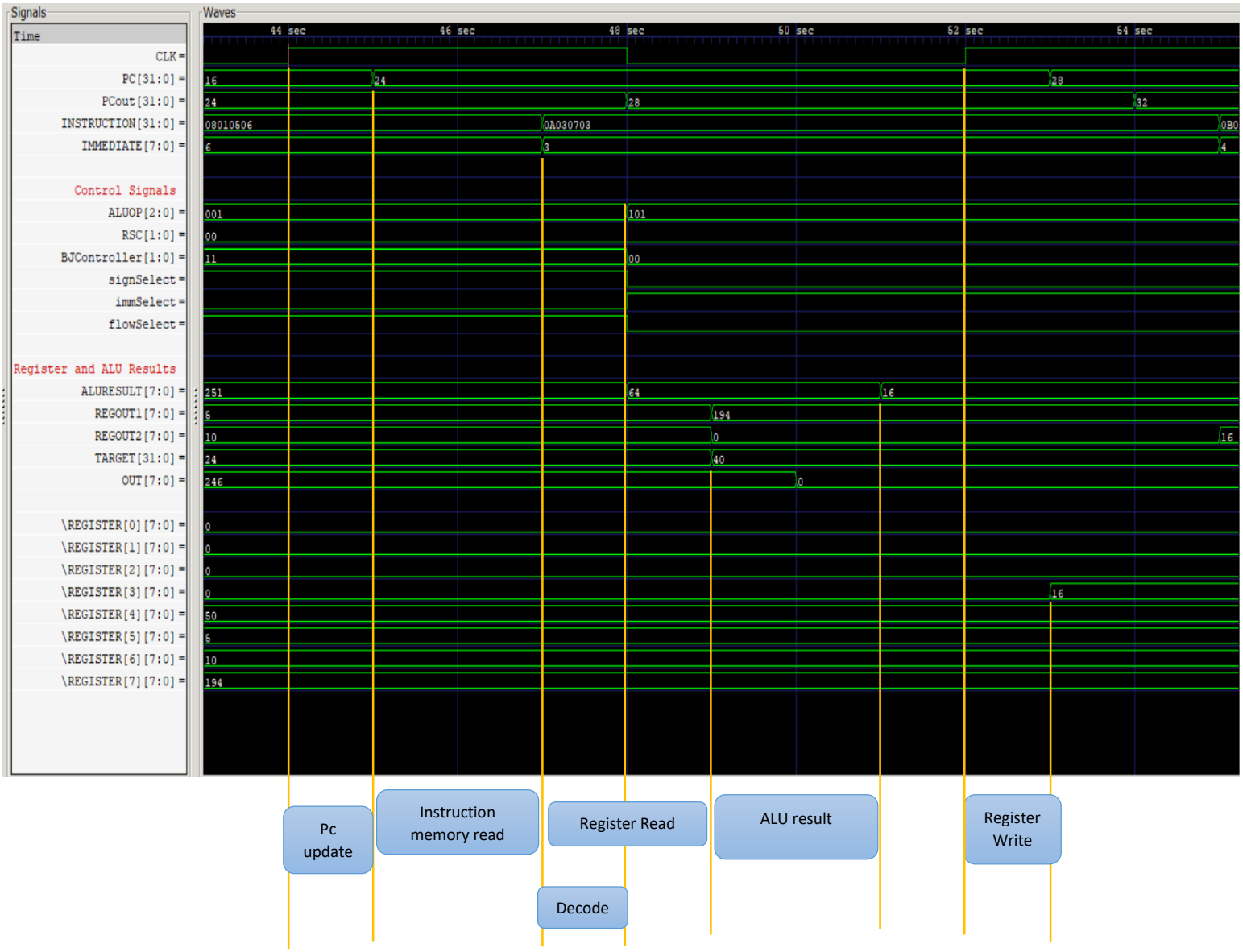
1. MULT Instruction



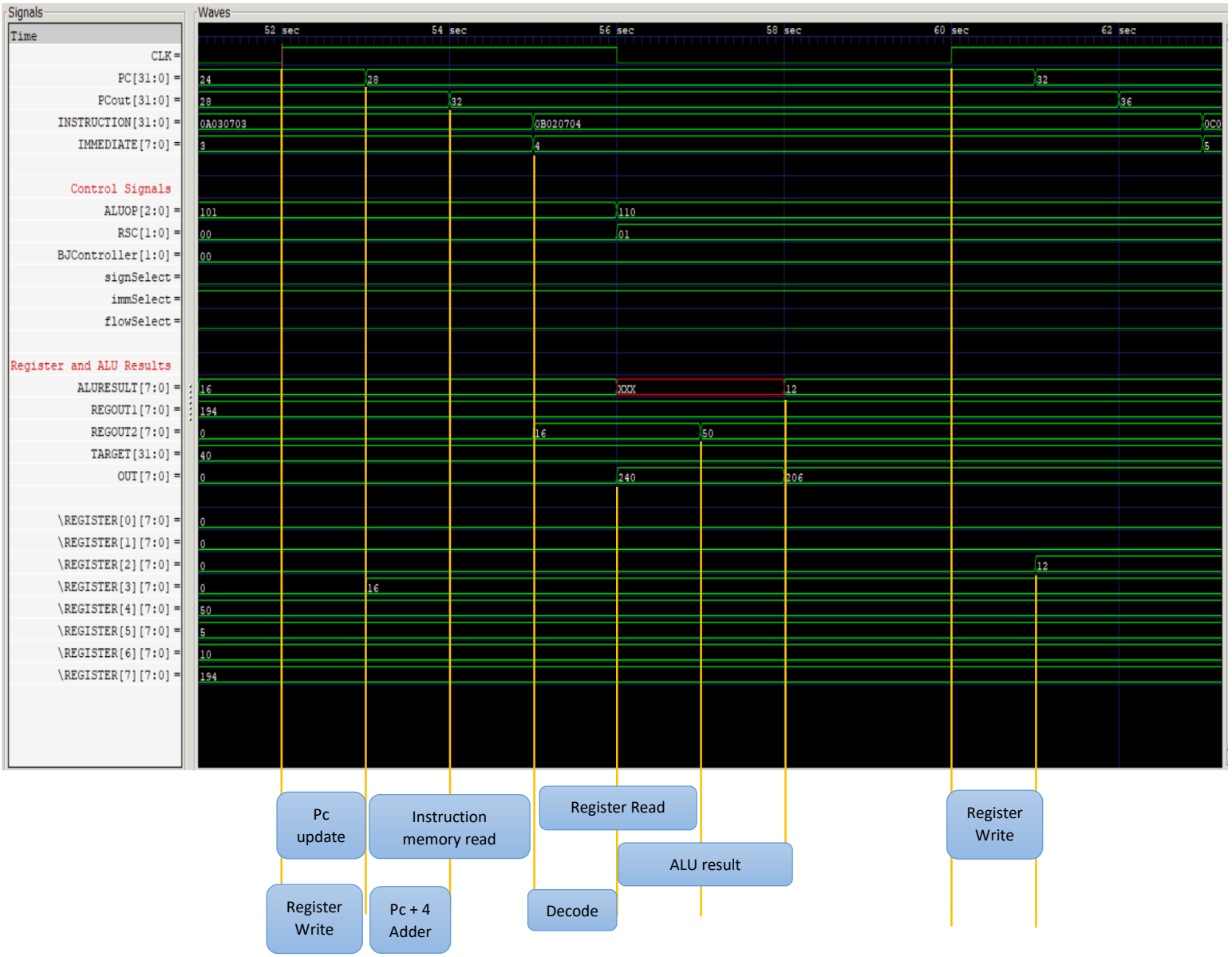
2. BNE Instruction



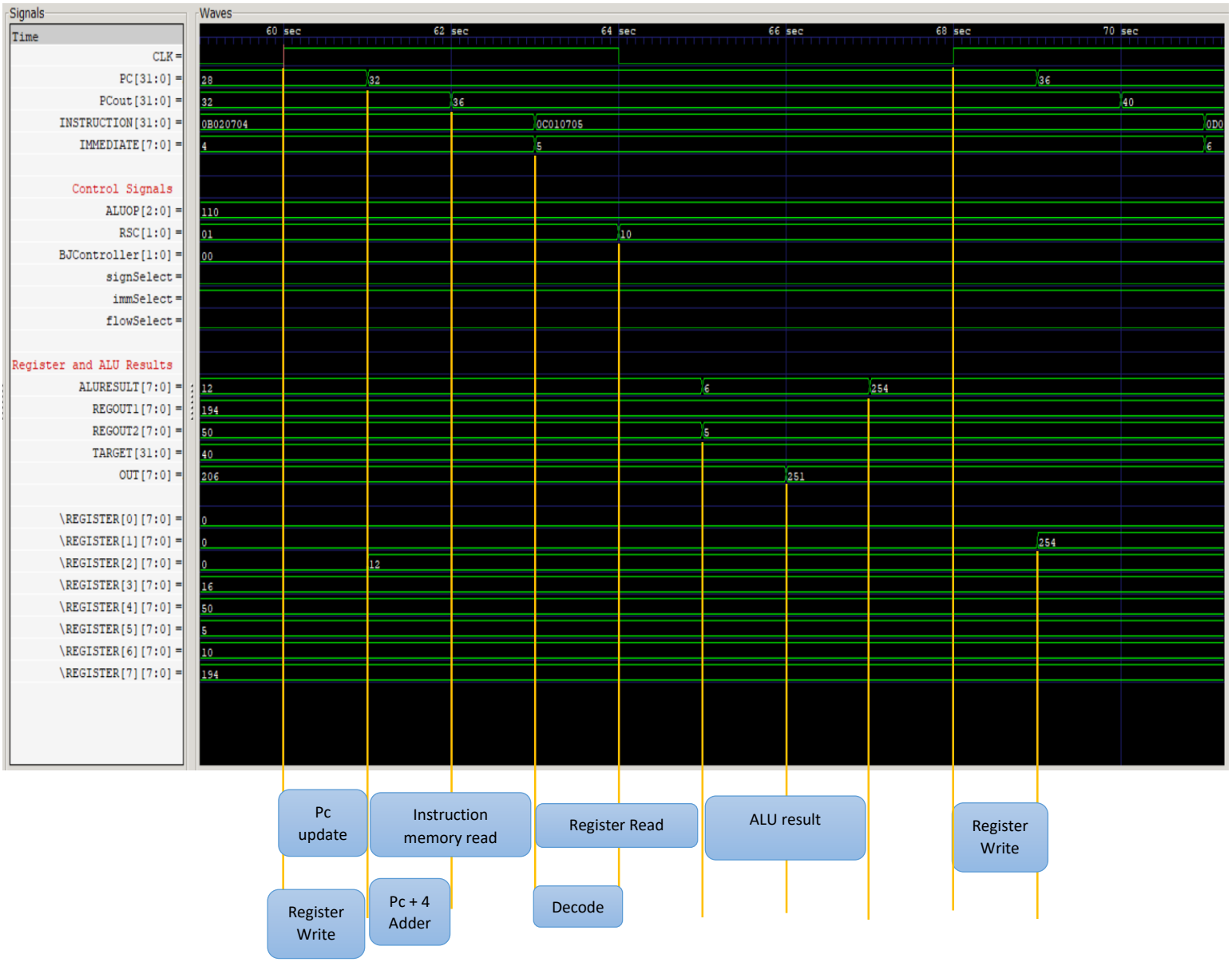
3. SLL Instruction



4. SRL Instruction



5. SRA Instruction



6. ROR Instruction