CO224 – Computer Architecture

Lab 5 – Building a Simple Processor

Part 5 – Extended ISA – Report

Group 20 – Balasuriya I.S. (E/17/018), Francis F.B.A.H. (E/17/090)

In Part 5, the instruction set of the CPU was extended to include five new instructions. This was achieved by making various modifications to the original CPU hardware as well as introducing several changes and restrictions in the assembler for the CPU. The newly added instructions as well as their expected usage is as follows.

`mult <r1> <r2> <r3>` : Multiplies the values in r2 and r3 and stores result in r1

`sll <r1> <r2> <offset>`: Logically left shifts r2 by offset value and stores result in r1

`srl <r1> <r2> <offset>`: Logically right shifts r2 by offset value and stores result in r1

`sra <r1> <r2> <offset>`: Arithmetically right shifts r2 by offset value and stores result in r1

`ror <r1> <r2> <offset>`: Rotates r2 right by offset value and stores result in r1

`bne <offset> <r1> <r2>`: Branches based on offset if values in r1 and r2 are not equal

To facilitate these new instructions, a minor modification was made to the Branch/Jump hardware and some additional functional units were implemented within the ALU of the CPU. These changes are reflected in the CPU block diagram and the ALU functions table given below.

CPU Block Diagram

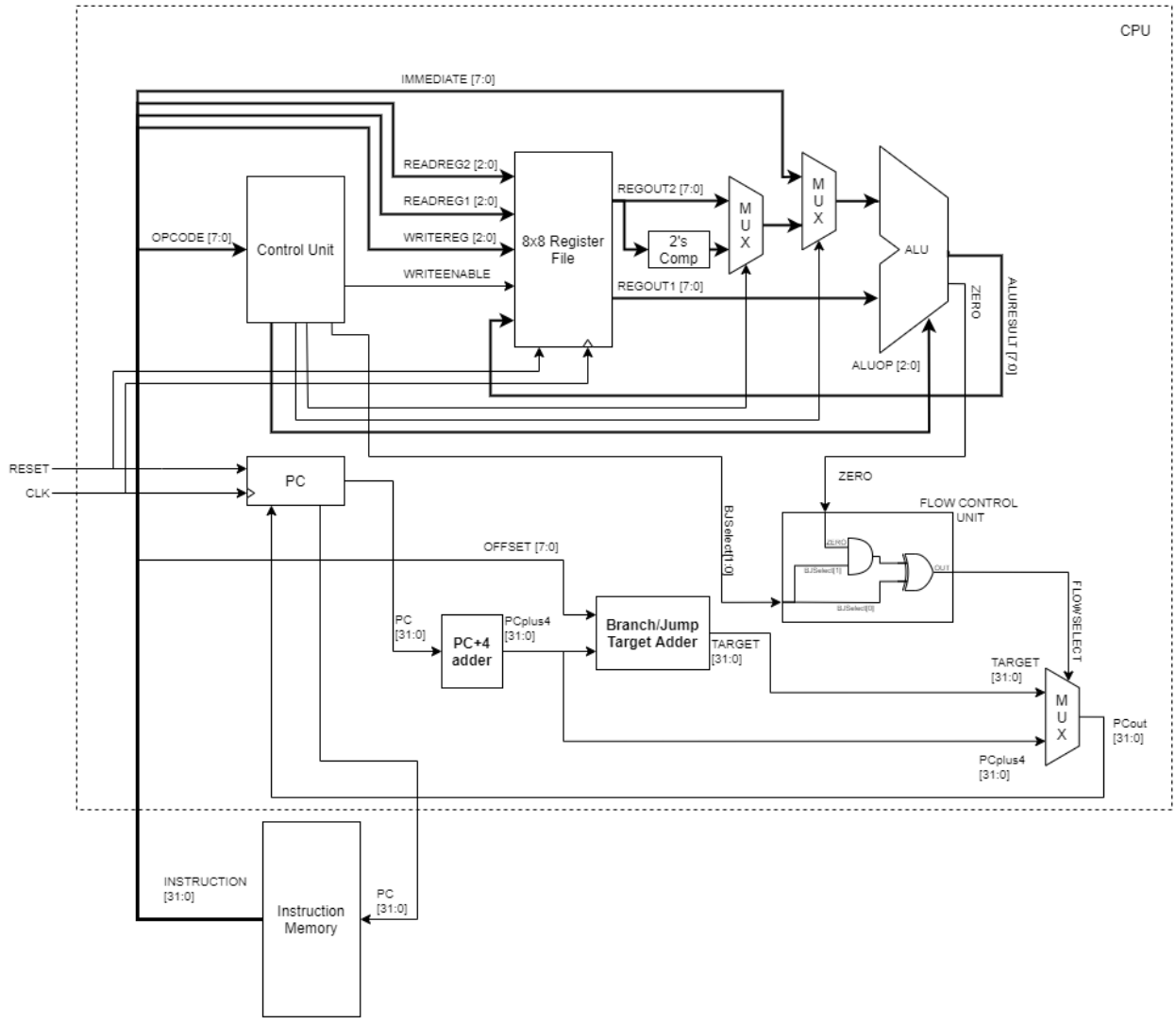


Figure 1: CPU Block Diagram

ALU

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	LSHIFT	DATA1<<DATA2→RESULT	sll	#2
101	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
110	MULT	DATA1*DATA2→RESULT	mult	#3

BNE Instruction

As can be seen in the CPU block diagram, the Flow Control Unit has been modified by replacing the OR gate with an XOR gate to provide the OUT output. In addition, the BRANCH and JUMP control signals from the Control Unit have been consolidated into a single 2-bit bus named BJSelect. These modifications were made to allow for the BNE instruction which should trigger a PC jump to TARGET value if the ZERO input from the ALU is not HIGH. The unit responds to the control signals from the CU as follows.

Table 2: Behaviour of Flow Control Unit

BJSelect	FLOWSELECT
00	0
01	1
10	1 if ZERO is HIGH
11	1 if ZERO is LOW

This behaviour of the XOR gate can be exploited to implement the BNE instruction by simply setting the BJSelect control signal to 11 for BNE instructions. Hence, once the decoding logic was added to the Control Unit, no further modifications were necessary and the BNE instruction could be added to the instruction set.

Since the underlying hardware is mainly the same, the timings of the BNE instruction remain identical to that of the BEQ instruction.

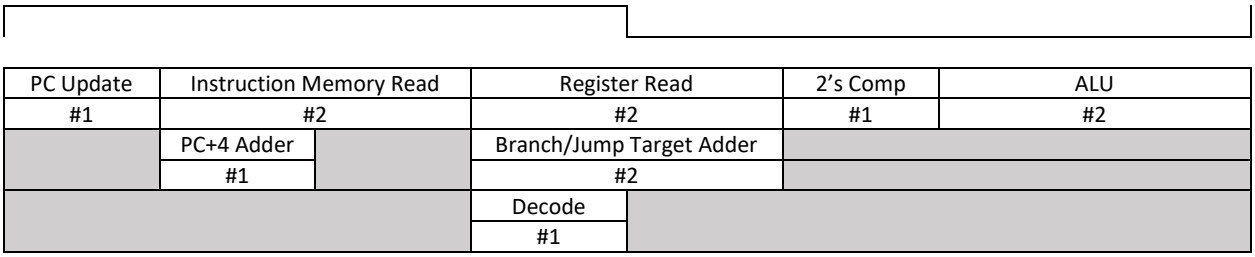


Figure 2: Timing details for BNE instruction datapath

MULT Unit

For the MULT instruction, a separate functional unit was added to the ALU. The unit consists of an array of Full Adders that work in several layers to generate the result for each bit of the output. The Full Adder was also implemented as a module using basic combinational logic. As such, the block diagram for the MULT functional unit is as follows.

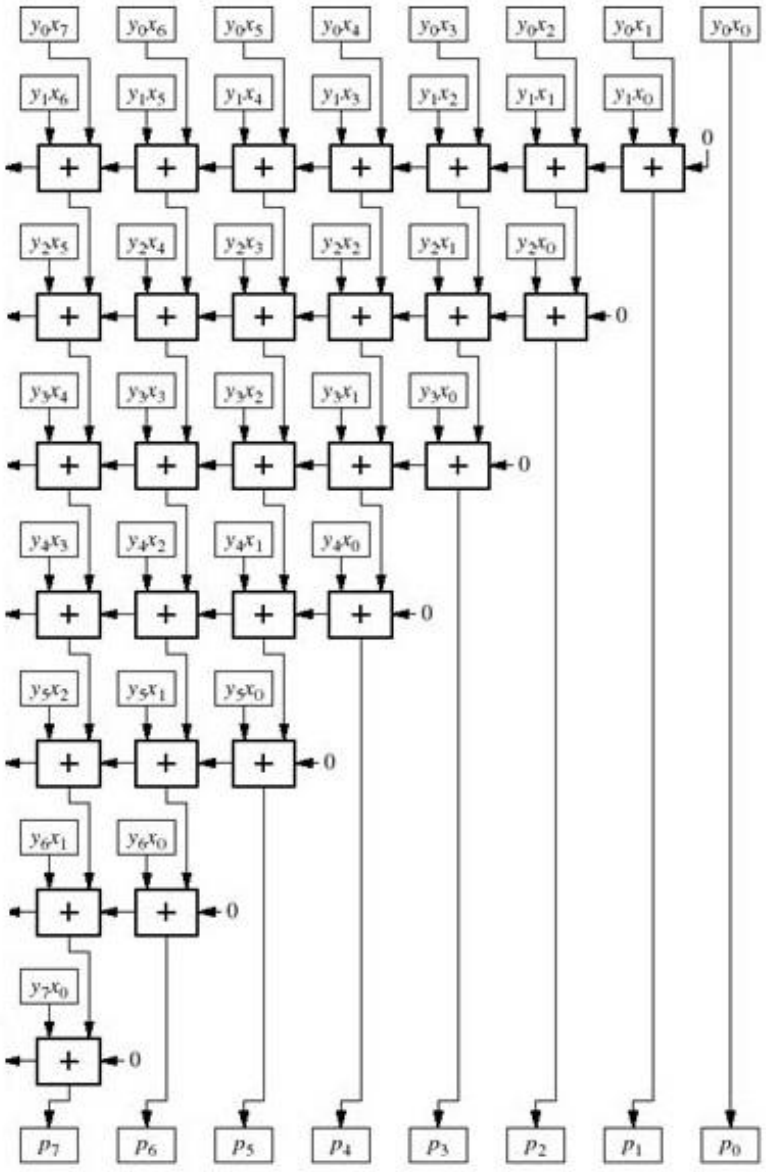


Figure 3: MULT Functional Unit Block Diagram

The multiplier, much like the ADD unit, has limitations on the maximum result that can be produced accurately since the result has to be contained within 8 bits. As such, any multiplication operation that produces a result that is greater than 255, the multiplier will produce erroneous results.

Since the MULT unit uses a large array of Full Adders, it can be assumed that a fairly large amount of time is taken for the calculation of the result when compared with the other functional units of the ALU. Hence, a simulation delay of #3 time units was added to this unit. Then, the timings for the MULT instruction are as follows.

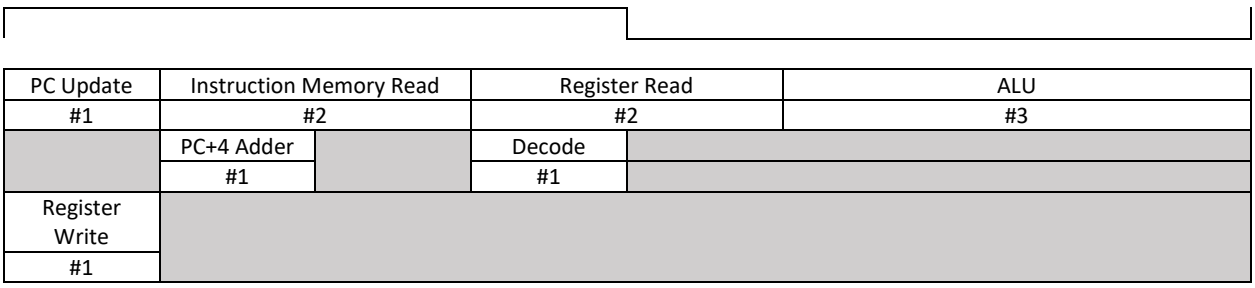


Figure 4: Timing details for MULT instruction datapath

LSHIFT Unit

Another functional unit was added to the ALU to facilitate bitwise left shift operations. This unit acts as a barrel shifter with several layers of MUXes to generate the shifted output. The MUXes for the unit were also implemented as separate modules and the block diagram for the LSHIFT functional unit is given below.

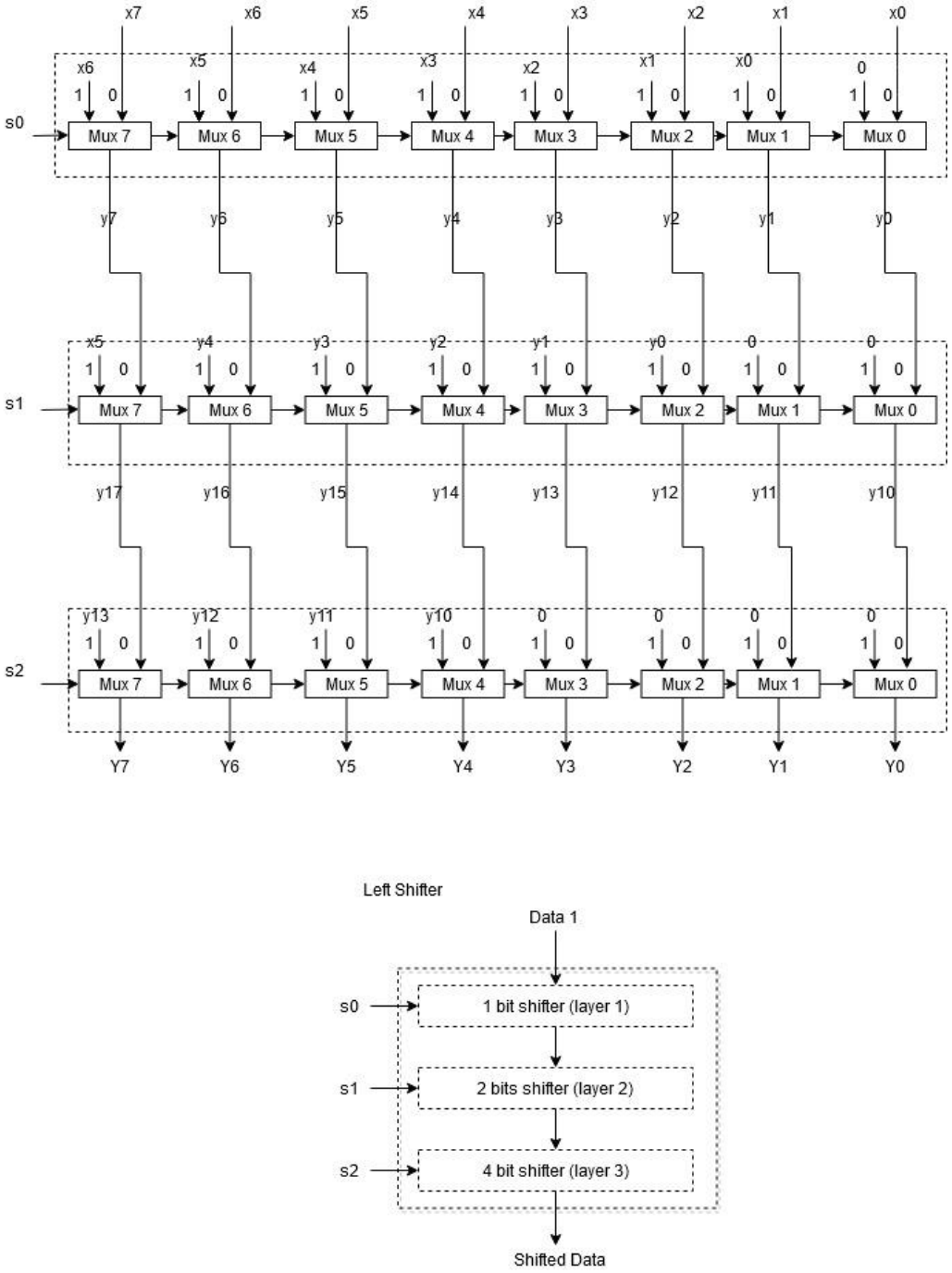


Figure 5: LSHIFT Unit Block Diagram

Since the data word size is 8 bits, shifting by a value larger than 8 will give the same result as shifting by 8. Therefore, to minimize hardware complexity, the shifter supports shift operations only up to 8 shifts. To handle shifts greater than 8, the assembler was modified to detect such large shifts. Then, in such cases, the assembler replaces the shift value with a shift value of 8 to provide the expected behaviour while maintaining simplicity in design.

The LSHIFT unit handles one instruction; the SLL instruction. A simulation delay of #2 time units was set for this operation. The timings for this instruction are as given below.

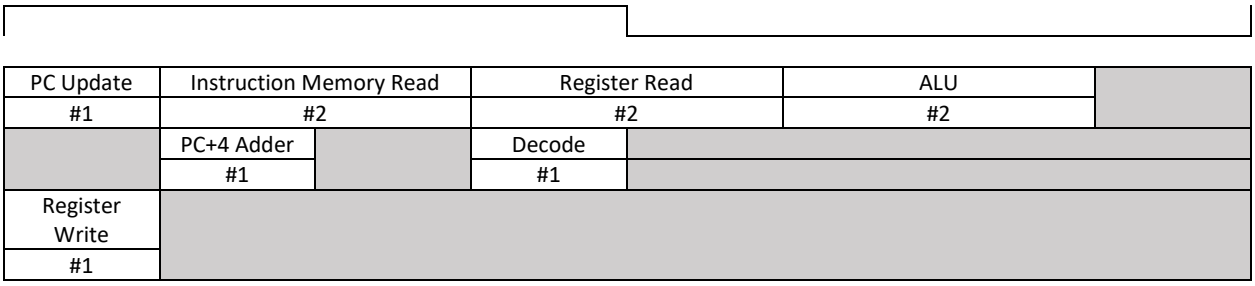
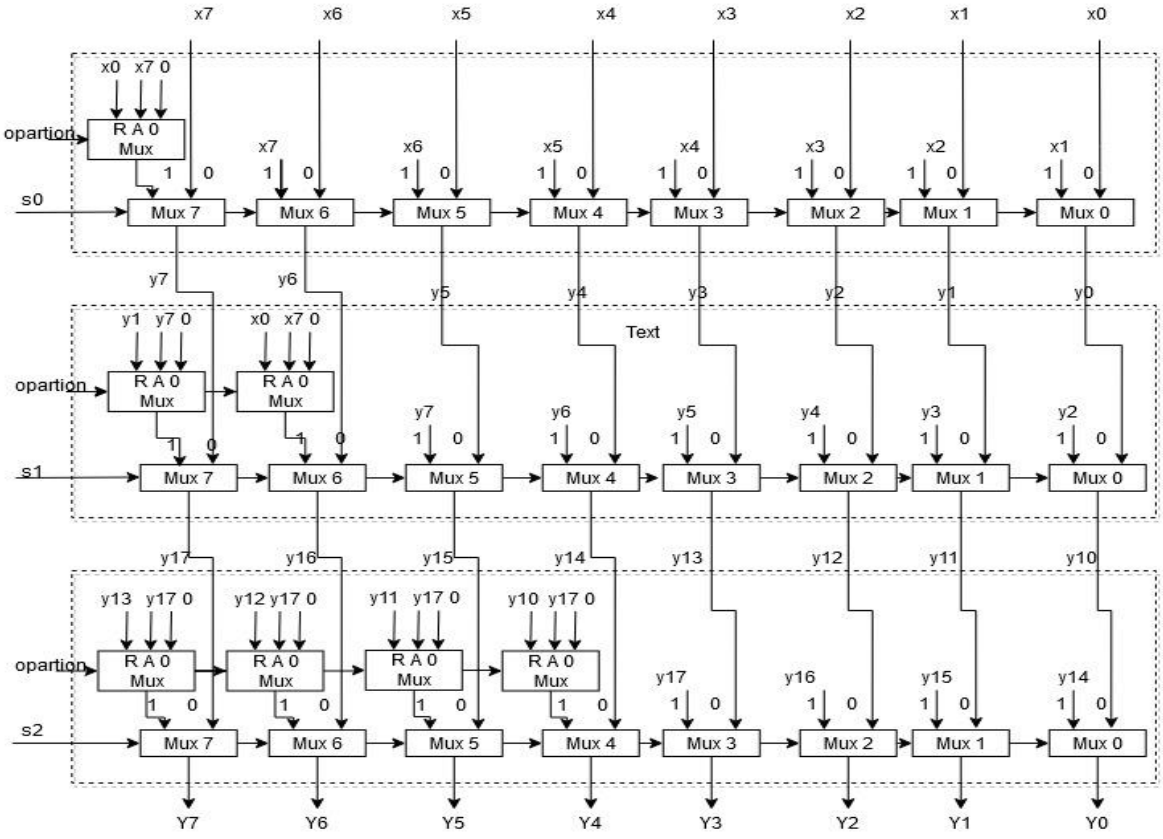


Figure 6: Timing details for SLL instruction datapath

RSHIFT Unit

The three right shift instructions were implemented using a single functional unit named RSHIFT which handles all right shift/rotate operations. This unit acts similar to the LSHIFT unit in that it utilizes a barrel shifter model. However, additional hardware is also present to allow for arithmetic and circular shift operations. The block diagram for the RSHIFT unit is as follows.



Right Shifter

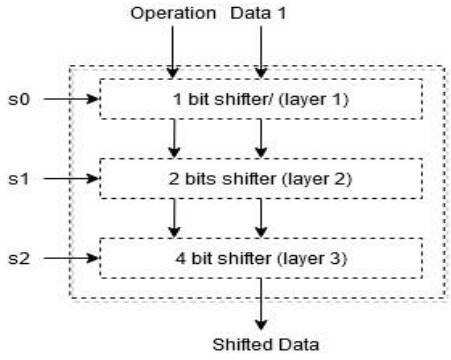


Figure 7: RSHIFT Unit Block Diagram

This unit chooses the correct operation to perform based on the first two MSBs of the shift value provided in DATA2. Like in the LSHIFT unit, only the first three bits of the shift value from the LSB side are taken since shifts greater than 8 will give identical results to shifts of 8. Once again, at the assembler, these larger shift values are replaced with shift value 8 to maintain correct outputs for these instructions. For the rotate right instruction, however, such replacement is not necessary since the output repeats its pattern every 8 shifts. So, in the case of rotate operations, the higher order bits of large shift values can simply be ignored and the result would still be accurate. The behaviour of the RSHIFT unit can be illustrated in the following manner.

Table 3: Behaviour of RSHIFT Unit

DATA[7:6]	Operation Performed
00	Logical Right Shift
01	Arithmetic Right Shift
10	Rotate Right

Then, the assembler was modified to inject these bits into the IMM/RS segment of the instruction when generating the machine code based on the instruction provided.

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

srl/sra/ror:

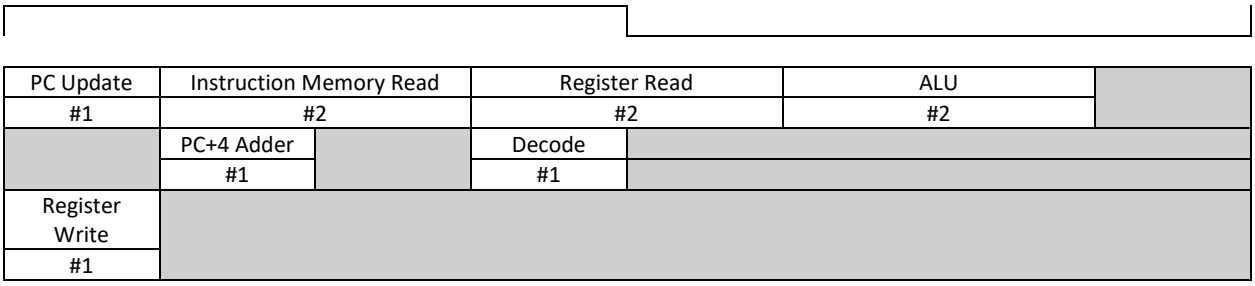


Figure 8: Timing details for right shift instruction datapath

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