Lab 5 Part 5 Report – Extended Instruction Set Implementation
Group 06
E/21/087-Dewagedara D.M.E.S.
E/21/302 Perera W.S.S.

# 1. OVERVIEW

In this bonus part, we extended our processor to support six new instructions:

1.mult (multiply)

2.sll (logical shift left)

3.srl (logical shift right)

4.sra (arithmetic shift right)

5.ror (rotate right)

6.bne (branch if not equal)

Each of these instructions was added without increasing the ALU control signal width (3-bit ALUOP), requiring efficient reuse of existing functional units and minimal additional control logic.

# 2. Instruction Encoding

Each instruction uses the following format:

| OPCODE | RD/IMM | RT | RS/IMM |
|---|---|---|---|

| INSTRUCTION | FORMAT | EXAMPLE | OPCODE(3-bit ALUOP) |
|---|---|---|---|
| mult | mult rd rs1 rs2 | mult 4 1 2 | 100 |
| sll | sll rd rs1 imm | sll 4 1 0x02 | 101 |
| srl | srl rd rs1 imm | srl 4 1 0x02 | 101 (same opcode-srl) |
| sra | sra rd rs1 imm | sra 4 1 0x02 | 110 |
| ror | ror rd rs1 imm | ror 4 1 0x02 | 111 |

We reused and reassigned ALUOPs wisely to avoid needing more than 3 bits.

## 3. ALU Functional Unit Modifications

To implement all six instructions using only 8 ALUOP values, we structured shared functional units as follows:

| ALUOP | FUNCTION |
|---|---|
| 000 | FORWARD |
| 001 | ADD/SUBTRACT |
| 010 | BITWISE AND |
| 011 | BITWISE OR |
| 100 | MULTIPLICATION |
| 101 | SHIFT LEFT LOGICAL |
| 110 | SHIFT RIGHT ARITHMETIC |
| 111 | ROTATE RIGHT |

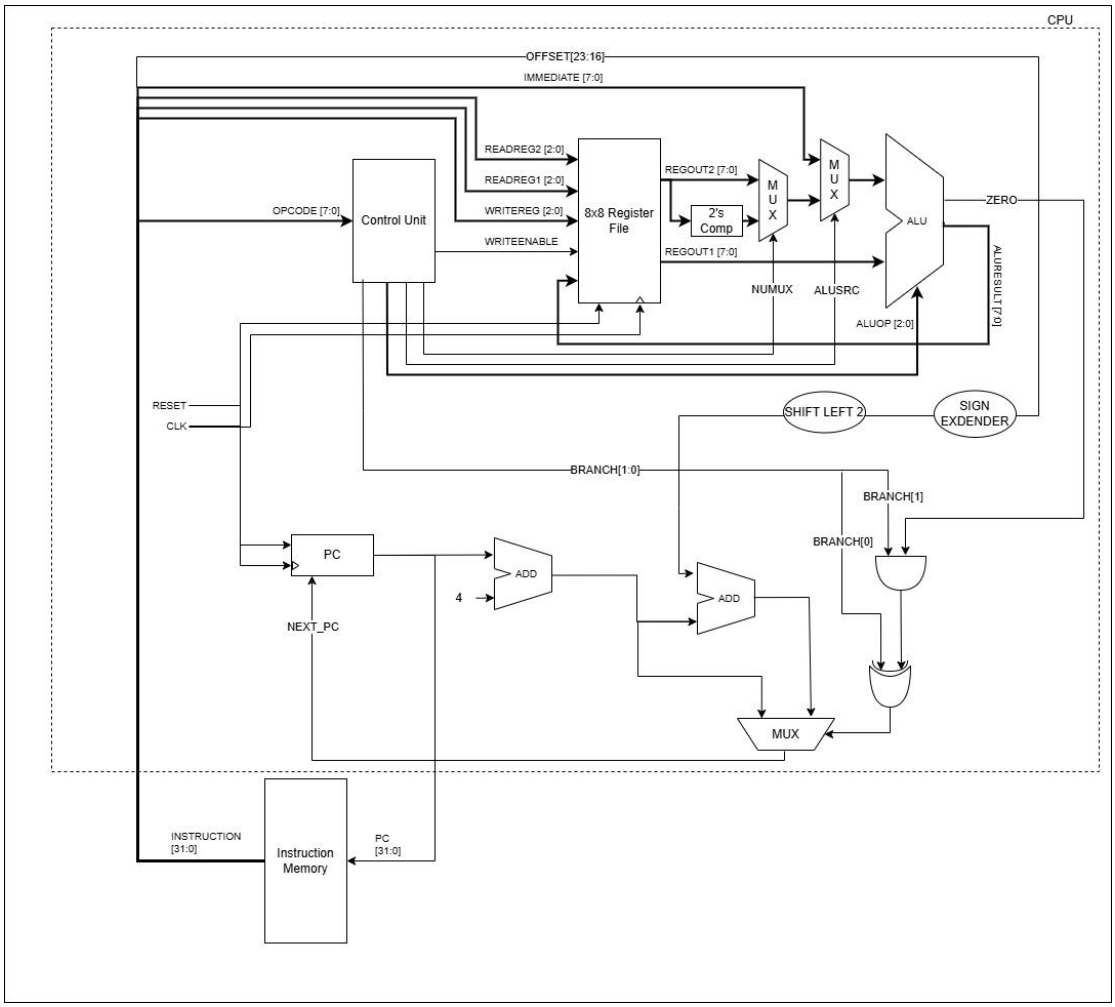
## 4. Instruction Encodings, Assigned Opcodes, and Control Signals

| INSTRUCTION | INSTRUCTION OPCODE | WRITE ENABLE | ALUOP | ALUSRC | NEMUX | BRANCH |
|---|---|---|---|---|---|---|
| ADD | 00000000 | 1 | 001 | 1 | 0 | 00 |
| SUB | 00000001 | 1 | 001 | 1 | 1 | 00 |
| AND | 00000010 | 1 | 010 | 1 | 0 | 00 |
| OR | 00000011 | 1 | 011 | 1 | 0 | 00 |
| MOV | 00000100 | 1 | 000 | 1 | 0 | 00 |
| LOADI | 00000101 | 1 | 000 | 0 | 0 | 00 |
| J | 00000110 | 0 | 000 | 0 | 0 | 01 |
| BEQ | 00000111 | 0 | 001 | 1 | 1 | 10 |
| BNE | 00001000 | 0 | 001 | 1 | 1 | 11 |
| MULT | 00001001 | 1 | 100 | 1 | 0 | 00 |
| SL | 00001010 | 1 | 101 | 0 | 0 | 00 |
| SRA | 00001100 | 1 | 110 | 0 | 0 | 00 |
| ROR | 00001101 | 1 | 111 | 0 | 0 | 00 |

Branch Encoding,

| BRANCH | FLOWSELECT |
|---|---|
| 00 | Normal sequential execution |
| 01 | Unconditional jump |
| 10 | Branch if equal (BEQ) |
| 11 | Branch if not equal (BNE) |

# 5. CPU BLOCK DIGRAM



## 6. New instruction

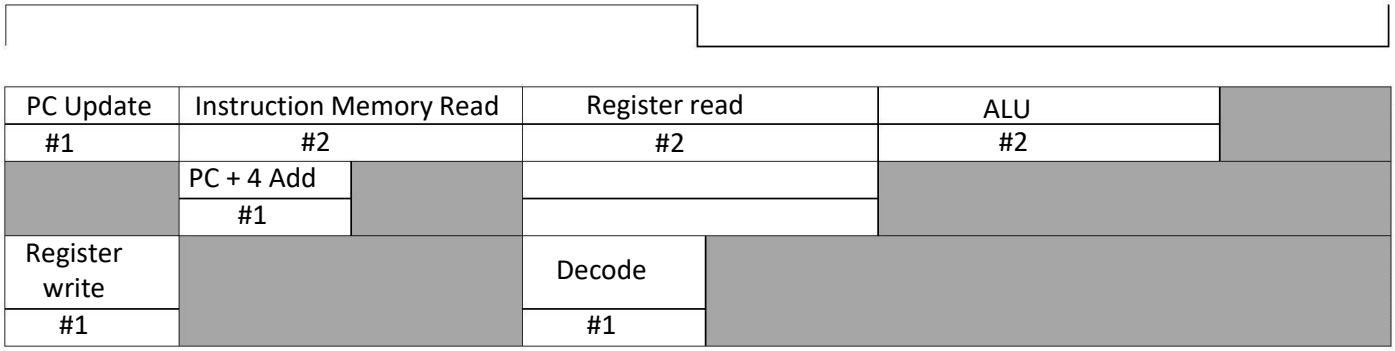
1. multiplication -mult instruction

The MULT instruction is supported using a dedicated 8×8 array multiplier built into the ALU. It uses layers of Full Adders made from basic logic gates to combine partial products, which are generated by ANDing each bit of the multiplicand with the multiplier.

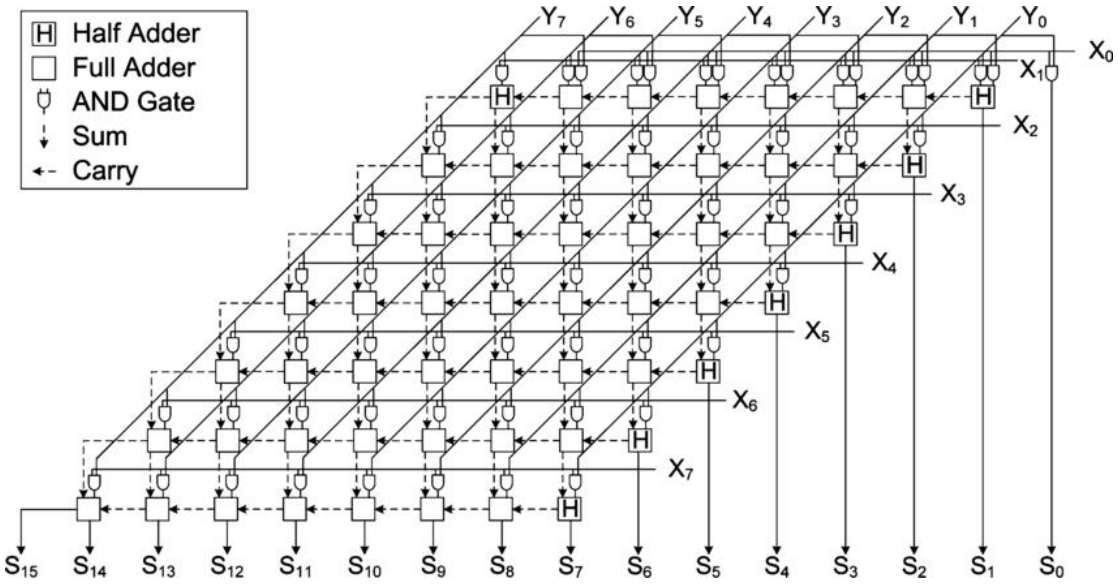
These partial products are added in a structured, layered way. The design follows a triangular shape: early layers process more bits, while later layers handle fewer, optimizing hardware usage and ensuring correct carry propagation using carry chains between Full Adders.

All operations are performed within an 8-bit system. So, if the result exceeds 255, it gets truncated, causing overflow and possible incorrect results. This is a trade-off made to keep the hardware simple and efficient.

TIMING DIAGRAM,



DESIGN,




## 2. Logical Shift Left and logical shift right – sll and srl instructions

Another functional unit was integrated into the ALU architecture to facilitate both bitwise left shift and right shift operations within a unified hardware structure. This consolidated unit operates as a barrel shifter employing multiple layers of multiplexers to generate the appropriately shifted output for both shift directions. The multiplexer components for this unit were implemented as discrete modules to maintain design modularity and enable systematic verification of the shift logic.

Given the 8-bit data word architecture, shifting by values exceeding 8 positions produces identical results to shifting by exactly 8 positions, as all original data bits are displaced beyond the representable range. To minimize hardware complexity while maintaining functional correctness, the shifter unit supports shift operations only up to 8 positions. For shift amounts greater than 8, the assembler incorporates detection logic to identify such oversized shift values and automatically replaces them with a shift value of 8, thereby providing the expected zero-fill behavior while preserving design simplicity.

The unified shift unit employs an innovative approach to distinguish between left and right shift operations without requiring additional control signals. This technique utilizes the most significant bit of the immediate shift amount value as a directional indicator, where a value of 0 designates left shift operations and a value of 1 indicates right shift operations. This directional bit is embedded during the compilation process, eliminating the need for separate control signal routing within the processor datapath.

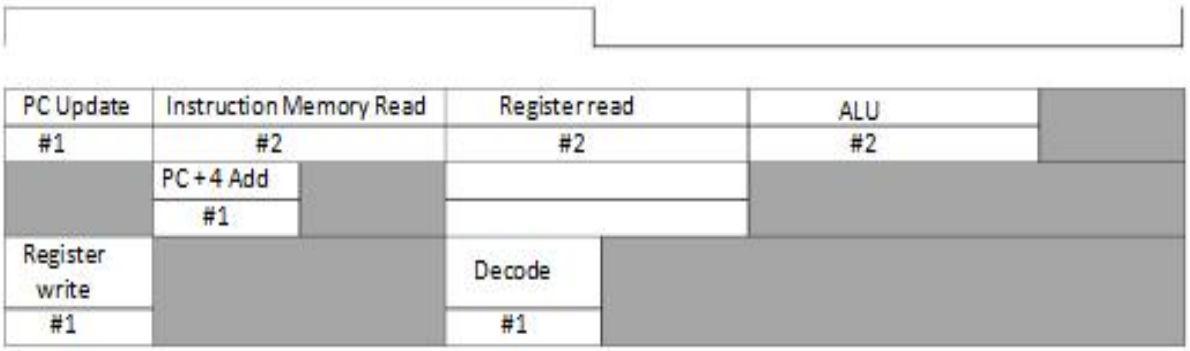
The barrel shifter architecture implements this dual-direction capability through conditional routing within the multiplexer network. When the directional bit indicates left shift operation (MSB = 0), the multiplexers route input bits toward higher-order output positions with zero-fill at the least significant positions. Conversely, when right shift operation is indicated (MSB = 1), the routing directs input bits toward lower-order positions with zero-fill at the most significant positions.

Both SRL and SLL operations implement consistent overflow behavior where shift amounts greater than 8 result in complete zero output, reflecting the logical consequence of shifting all data bits beyond the 8-bit representation boundary. This uniform overflow handling ensures predictable operation regardless of shift direction and simplifies the verification process for the combined shift unit.

However, this design approach introduces a specific ambiguity case that affects programmer expectations. When a programmer specifies a command such as "sll r3 r4 128" (binary 10000000), the compiler interprets this immediate value with the most significant bit already set to 1, causing the CPU to identify this as an SRL operation with a shift amount of 0 (derived from the remaining seven bits). Consequently, the CPU executes a right shift by 0 positions, returning the original value unchanged, whereas the programmer's intention was a left shift by 128 positions that should produce a zero result. While this represents a corner case in the instruction encoding scheme, all other shift operations within the normal range function correctly according to the intended directional semantics.

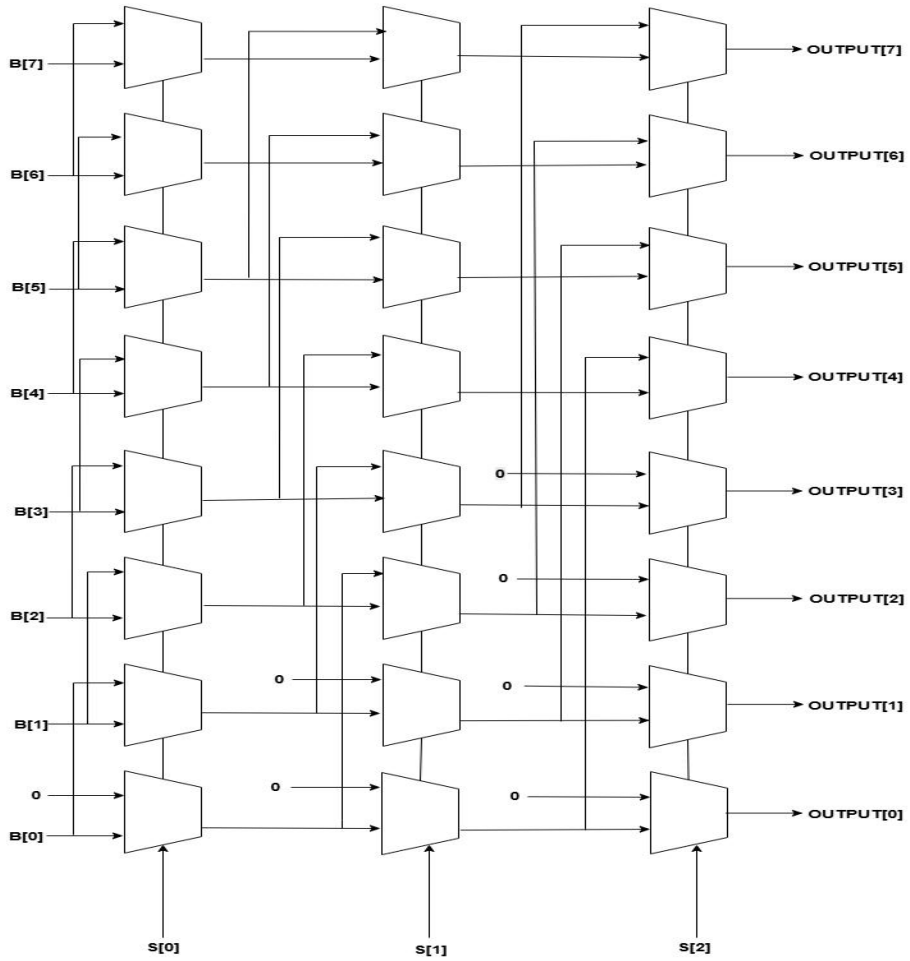
| MOST SIGNIFICANT BIT IN IMMEDIATE | OPERATION PERFORMED |
|---|---|
| 0 / shift amount remaining seven bits | sll |
| 1 / shift amount remaining seven bits | srl |

TIMING DIAGRAM,



DESIGN,

**LOGICAL LEFT SHIFT**



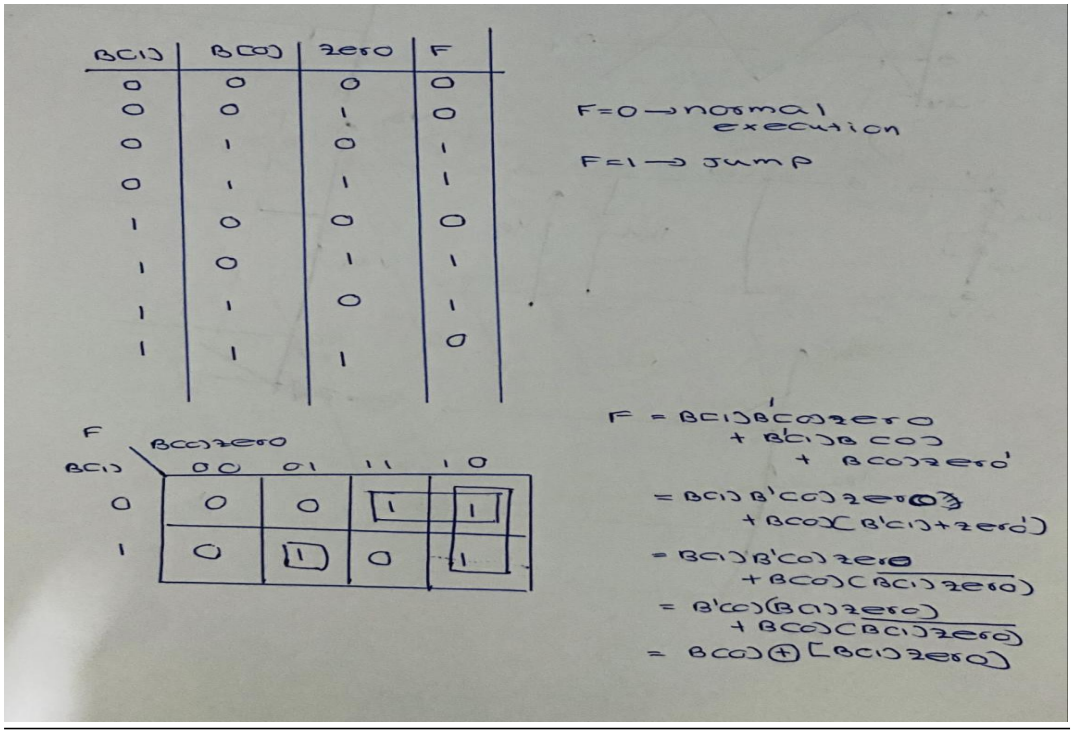
## 3. BNE

The BNE instruction implementation utilizes the existing 2-bit branch control signal without requiring additional control lines. The branch control signal encoding follows a systematic pattern where 00 represents normal sequential execution, 01 indicates jump operations, and both 10 and 11 are designated for BNE instruction variants. This encoding scheme maximizes the utilization of available control bits while maintaining compatibility with existing branch infrastructure.

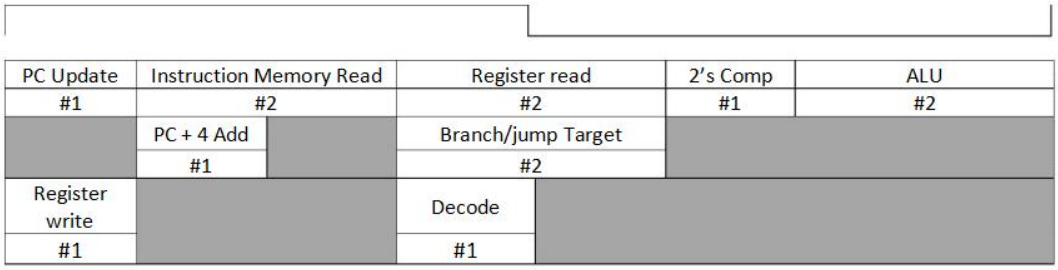
The Flow Control Unit implements BNE functionality through a combinational logic circuit that processes three inputs: branch[1], branch[0], and the zero flag from the ALU. The truth table for this implementation defines the branching behavior based on these input combinations. When branch control is set to 00, the output remains 0 regardless of the zero flag state, ensuring normal sequential execution. For branch control 01, the output is forced to 1, enabling unconditional jump operations.

The critical BNE functionality emerges when branch control is set to either 10 or 11. In these configurations, the output becomes dependent on the zero flag state, but with inverted logic compared to standard branch equal operations. When the zero flag is LOW (indicating operands are not equal), the circuit generates a HIGH output, triggering the branch to the target address. Conversely, when the zero flag is HIGH (indicating equal operands), the output remains LOW, continuing sequential execution.

This behavior is achieved through a carefully designed combinational circuit derived from Karnaugh map optimization.



TIME DIAGRAM,




---

## 4. ROR,SRA,SRL

The SRA and ROR instructions, along with the previously discussed SRL operation, are implemented within a unified barrel shifter circuit design that handles all three shift/rotate operations through a single hardware structure. This consolidated approach maximizes hardware efficiency while providing comprehensive bit manipulation capabilities through a systematic three-layer shifting architecture.

The barrel shifter employs a three-layer design where each layer provides specific shift capabilities in powers of two. The first layer implements single-bit shifting (shift by 1), the second layer handles two-bit shifting (shift by 2), and the third layer manages four-bit shifting (shift by 4). This logarithmic progression allows any shift amount from 0 to 7 to be achieved through appropriate combinations of the three layers, providing complete coverage of the 8-bit word shifting requirements.

The first layer represents the most critical component of the design, as it must handle the fundamental differences between the three operation types. For arithmetic right shift (SRA) operations, the most significant bit position receives the original sign bit to maintain proper sign extension for signed number representation. For logical right shift (SRL) operations, the most significant bit position is filled with zero to implement logical shifting behavior. For rotate right (ROR) operations, the most significant bit position receives the original least significant bit, creating the circular data path characteristic of rotation operations.

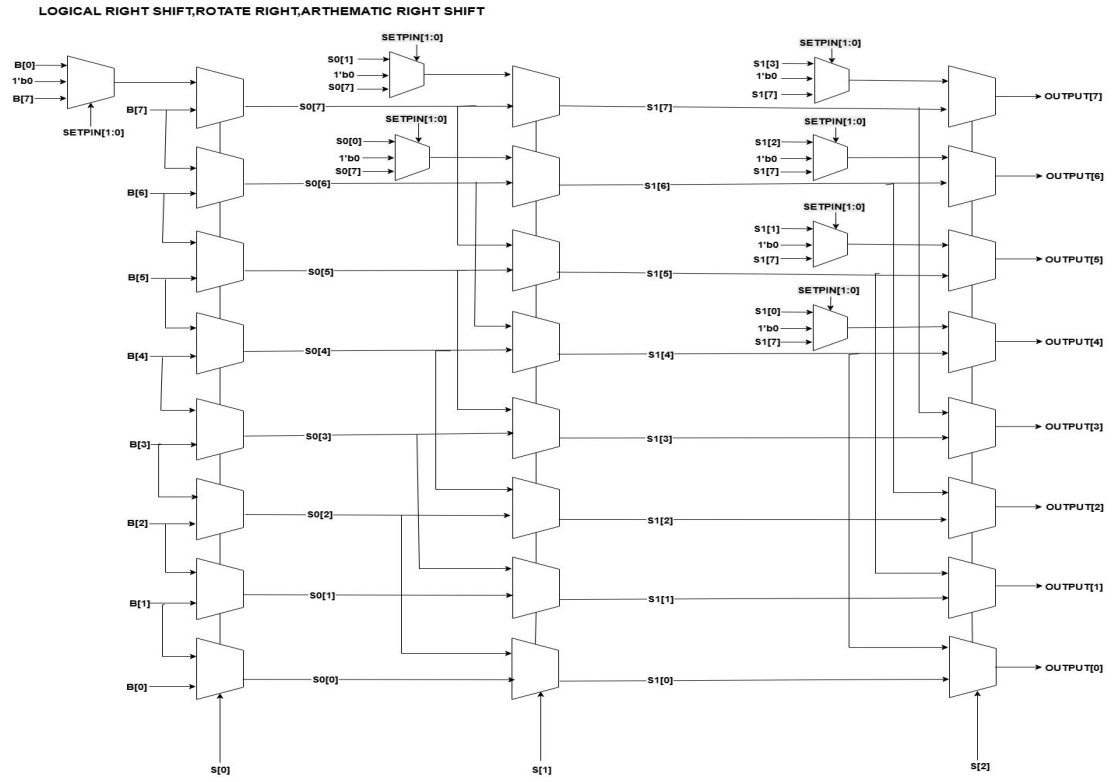
This differentiated bit-fill behavior is implemented through a 3:1 multiplexer network at each bit position in the first layer. The multiplexer selection inputs are derived from operation type control signals that distinguish between SRA, SRL, and ROR operations. When SRA is selected, the multiplexer routes the original MSB to the shifted MSB position. For SRL operations, the multiplexer selects a ground connection to provide zero-fill. For ROR operations, the multiplexer routes the original LSB to the MSB position, establishing the wraparound connection.

The second layer operates on the output of the first layer and implements shifting by two bit positions. The control for this layer utilizes the two most significant bits of the shift amount value to determine whether the two-position shift should be applied. When these control bits indicate a shift requirement, the layer routes input bits to positions two places to the right, with appropriate fill behavior determined by the operation type. The fill logic follows the same principles as the first layer, using the operation type signals to select between sign extension, zero-fill, or circular rotation.

The third layer completes the barrel shifter design by providing four-bit shifting capability. This layer processes the output from the second layer and applies shifting by four positions when the shift amount control bits indicate this requirement. The layer employs the same fill selection mechanism as the previous layers, ensuring consistent behavior across all shift amounts. The control signals for this layer are derived from the most significant bits of the shift amount, allowing precise control over the final shift distance.

The systematic three-layer approach ensures that any combination of 1, 2, and 4-bit shifts can be applied to achieve the desired total shift amount. For example, a shift by 5 positions is implemented by enabling both the first layer (shift by 1) and the third layer (shift by 4). The barrel shifter design maintains consistent timing characteristics regardless of the shift amount, as all signal paths traverse the same three-layer structure with only the multiplexer selections varying based on the control inputs.

DIAGRAM,



TIME DIGARM,

TIMING DIAGRAM,