# COA Lab Report

Assignment No. 7


Student 1: Aryan Singh Roll No: 19CS30007
Student 2: Abhinandan De Roll No: 19CS10069

# OBJECTIVES:

- To evolve a suitable instruction format for the given instruction set.
- To identify the data path elements and draw an architecture for ALU
- To draw the complete data path along with the control signals for a single cycle execution unit.
- To construct the truth table of the control signals as a function of the opcode and the function code in the instruction format.

# Instruction Format

Our design divides the operations into two classes R-Type and I-Type. We distinguish the operation by looking at the function code if the opcodes of two operations match with each other.

## Class 1 - R-type instructions (Op code: 0000X) 10 instructions

| Op Code | rs | rt | shamt | func | X |
|---|---|---|---|---|---|
| 5 | 5 | 5 | 5 | 5 | 7 |

| Operation | Opcode (5 bits) | Function code (5 bits) |
|---|---|---|
| Add | 0 | 0 |
| Comp | 0 | 1 |
| AND | 0 | 2 |
| XOR | 0 | 3 |
| shllv | 0 | 4 |
| shrlv | 0 | 5 |
| shrav | 0 | 6 |
| Shll | 1 | 4 |
| Shrl | 1 | 5 |
| shra | 1 | 6 |

In our design, the opcode tells the control unit to select the correct input to be fed to the input 2 slot for the ALU while the function code aids in selecting the module to be selected for performing the operation inside the ALU.

# Class 2 - I type instructions : 12 instructions

| Op Code | rs | rt/func | immediate |
|---|---|---|---|
| 5 | 5 | 5 | 17 |

| Operation | Opcode(5 bits) |
|---|---|
| lw | 2 |
| sw | 3 |
| addi | 4 |
| compi | 5 |
| branches | 6, 7, 8, 9 |

| Operation | Opcode (5 bits) | func(rt) (5 bits) |
|---|---|---|
| b | 6 | 0 |
| br | 7 | 0 |
| bcy | 6 | 1 |
| bncy | 6 | 2 |
| bz | 8 | 3 |
| bltz | 8 | 4 |
| bnz | 8 | 5 |
| bl | 9 | 6 |

In this case, the function code helps to choose the correct flag supplied by the ALU in case of conditional branch instructions and the opcode helps us to determine the following:

1. If the current instruction is a branch instruction
2. The target label.

We had 32 possible op code permutations, out of which we used 10.
So **we can add 22 more opCodes to the instruction set**.

**For R type instruction, we can add 25 more instructions** inside the first class (opcode = 0000X) using different function codes.

**For I type instructions we can add 244 more instructions** to the class

The Datapath elements that would be required in our processor are:

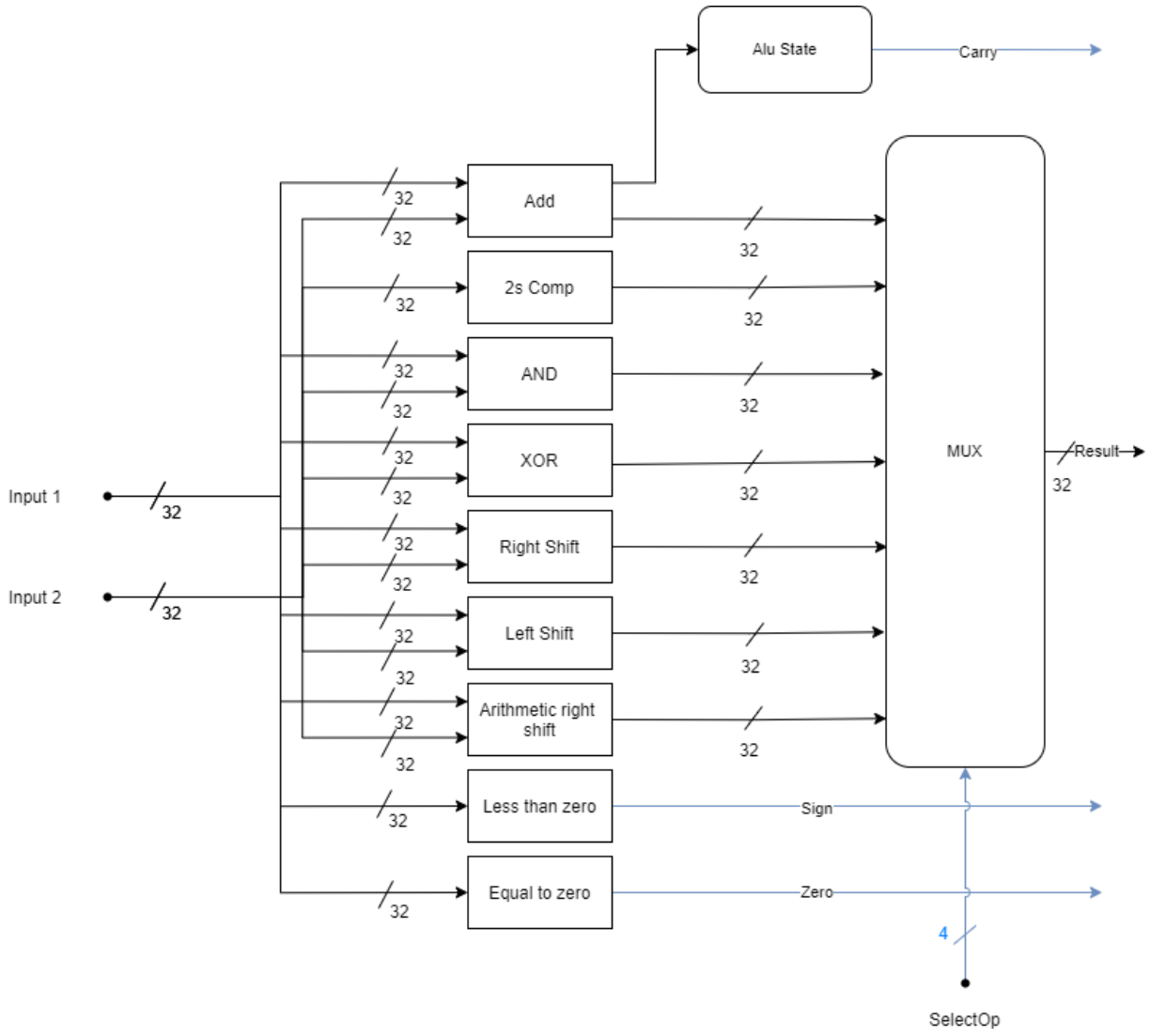
1. ALU
2. MUX
3. Register File
4. Data Memory
5. Instruction Memory
6. Program Counter
7. Sign Extend

# Designing the ALU

The ALU is capable of performing the following operations:

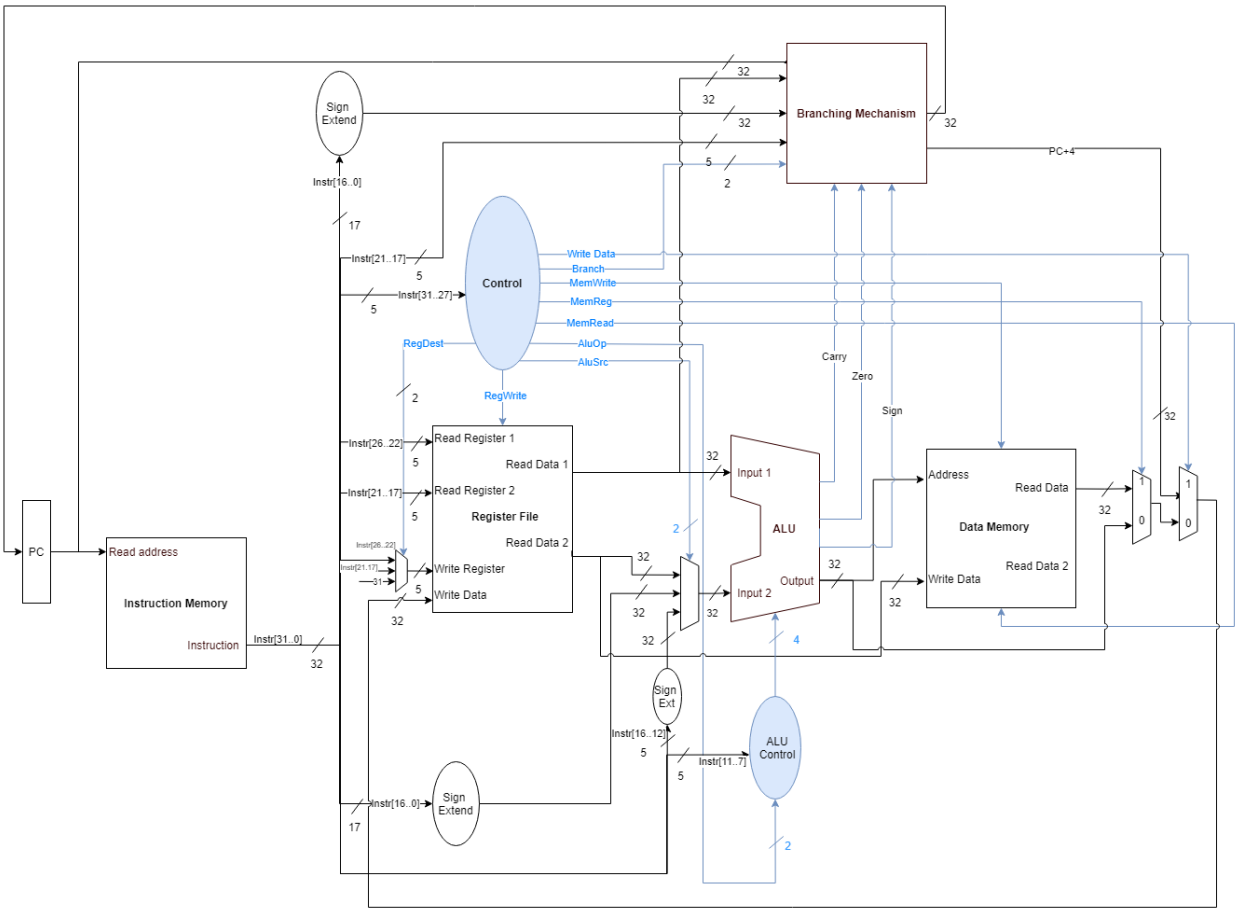
| Function | ALU control line |
|---|---|
| ADD | 0000 |
| COMP | 0001 |
| AND | 0010 |
| XOR | 0011 |
| LEFT SHIFT | 0100 |
| RIGHT SHIFT | 0101 |
| RIGHT SHIFT ARITHMETIC | 0110 |

# THE ALU



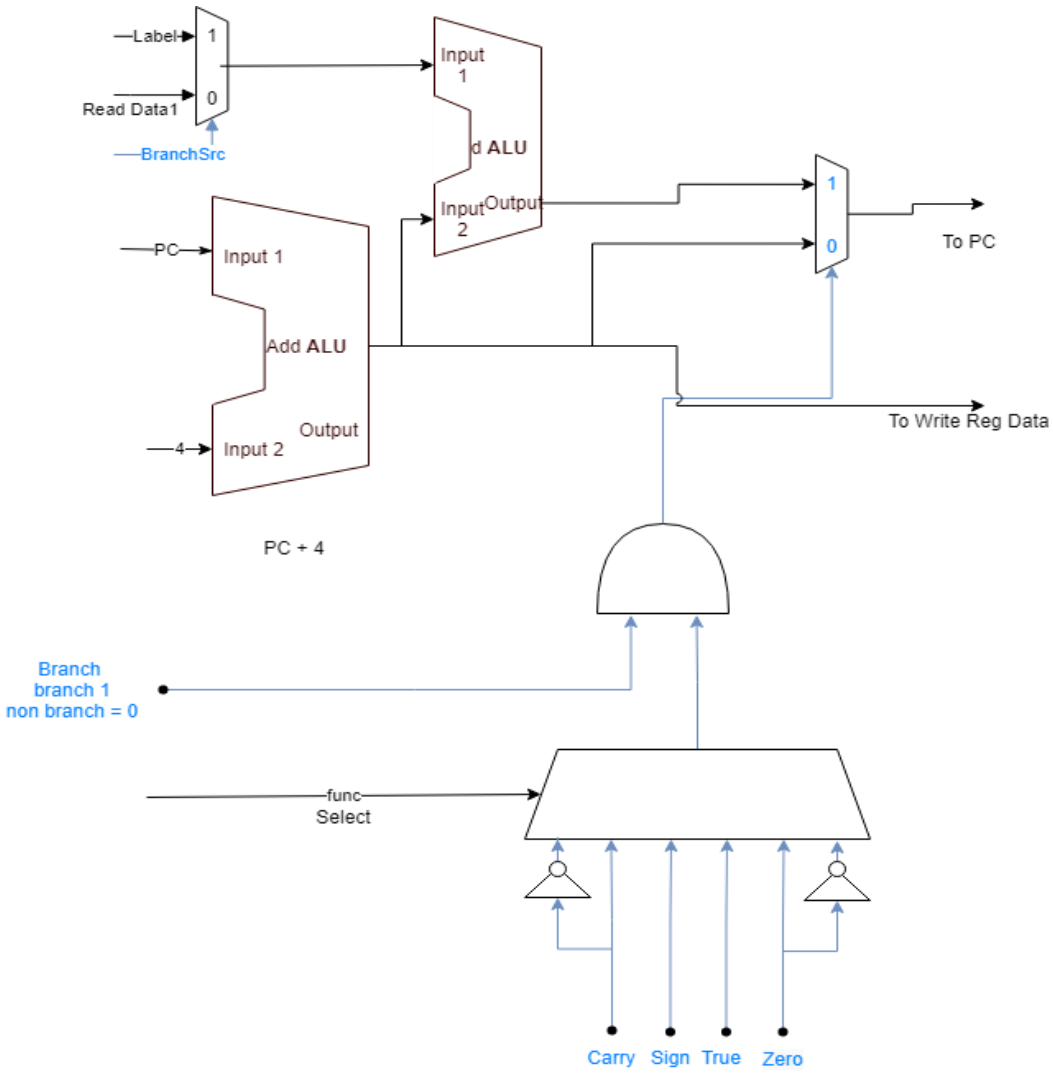

The ALU comprises nine submodules which work independently and the final output is selected using a multiplexor. It is also responsible for the Carry, Sign and the Zero signals. These aid in the implementation of the conditional branch instructions.

# Datapath




This figure shows the datapath of  our single cycle processor

# Branching Mechanism




There are two types of branches that we are required to handle in this assignment, unconditional and conditional. The conditional branches can be further divided into two categories owing to the branch and link instruction which requires an additional RegWrite operation.

The branching mechanism works as follows: If we have a conditional branch, we choose one out of the five outputs from the multiplexor at the bottom. If it's unconditional we choose the true line. After this we have an activate signal which indicates if we have a branch instruction in our program. These outputs are passed through an AND gate and we select the target label/register only if the output of the AND gate is 1.

# Truth Table for controls

| Instruction (OpCode) | Write Data | ALUSrc | MemTo Reg | Reg Write | Mem Read | Mem Write | Branch src,brnc | ALUOp | RegDest |
|---|---|---|---|---|---|---|---|---|---|
| R format(0) | 0 | 00 | 0 | 1 | 0 | 0 | X0 | 01 | 00 |
| Shift(0) (without sh) | 0 | 00 | 0 | 1 | 0 | 0 | X0 | 01 | 00 |
| Shift(1)(sh) | 0 | 11 | 0 | 1 | 0 | 0 | X0 | 01 | 00 |
| Load word(2) | 0 | 01 | 1 | 1 | 1 | 0 | X0 | 00 | 01 |
| Store word(3) | 0 | 01 | X | 0 | 0 | 1 | X0 | 00 | XX |
| Immediate(Add) (4) | 0 | 01 | 0 | 1 | 0 | 0 | X0 | 00 | 00 |
| Immediate (Comp)(5) | 0 | 01 | 0 | 1 | 0 | 0 | X0 | 10 | 00 |
| Branch w/o link (6)b,Cy,Ncy | 0 | XX | X | 0 | 0 | 0 | 11 | XX | XX |
| Br (7) | 0 | XX | X | 0 | 0 | 0 | 01 | XX | XX |
| Bz, blitz, bnz (8) | 0 | XX | X | 0 | 0 | 0 | 11 | XX | XX |
| Branchand link(9) | 1 | XX | X | 1 | 0 | 0 | 11 | XX | 10 |

In our design, the control signals are purely based on the opcode. The function codes are useful for generating controls in the submodules. Eg: choosing the correct flag for the branch module and choosing the correct submodule from the ALU.

| ALUOp | Function | Output |
|---|---|---|
| 1 | R format | last 4 bits of fcode, (function of fcode) |
| 0 | load word/store word (Add) | 0000 |
| 2 | Comp | 0001 |

This table represents the logic for the 4 bit output generated by the ALU Control depending on the ALUOp and the function codes.