Name - Thilakarathna W M D U

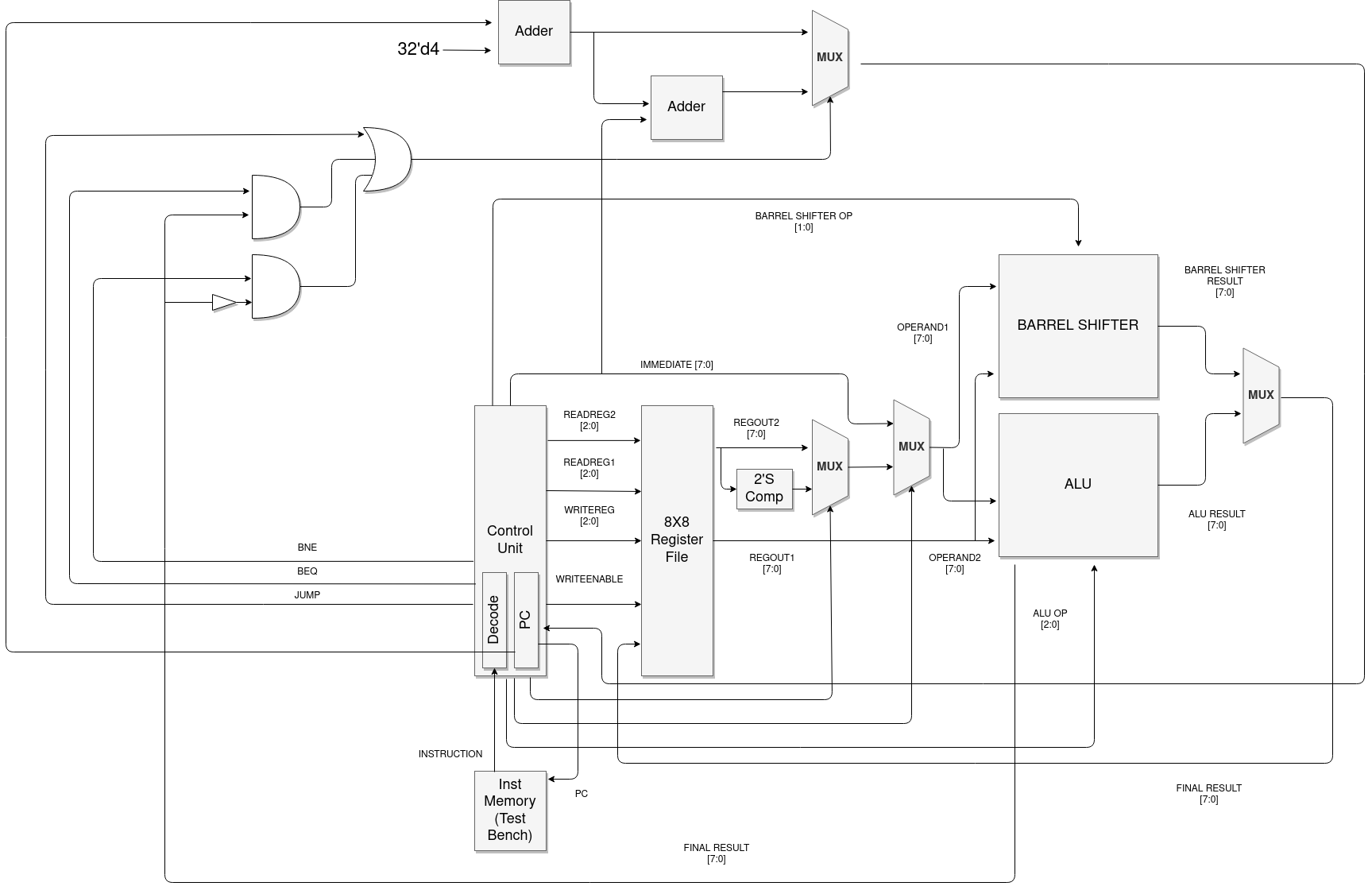
Reg No - E/16/366

**CO224 - Lab05 (Designing a CPU) Part05 (bonus)**

In this part, we have given to implement, Two or more instructions from the following instruction set,

1. mult (Multiply)
2. sll (Logical Left Shift)
3. srl (Logical Right Shift)
4. ror (Rotate Right)
5. bne (Branch if negative)

I have implemented all of the above commands the whole structure of the CPU looks as follows.



The whole CPU has created according to the following OPCode format.

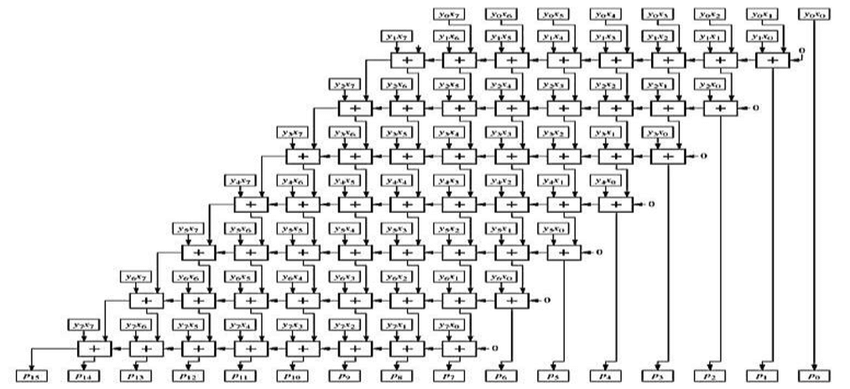
|  |  |  |
| --- | --- | --- |
| **Operation** | **Code** | **OpCode** |
| Addition | ADD | 8’h00 |
| Subtraction | SUB | 8’h01 |
| Bitwise AND | AND | 8’h02 |
| Bitwise OR | OR | 8’h03 |
| Move from one reg to another | MOV | 8’h04 |
| Load Immediate | LOADI | 8’h05 |
| Jump | J | 8’h06 |
| Branch if Equal | BEQ | 8’h07 |
| Branch if Not Equal | BNE | 8’h08 |
| Logical Shift Left | SLL | 8’h09 |
| Logical Shift Right | SRL | 8’h0A |
| Arithmetic Shift Right | SRA | 8’h0B |
| Rotate | ROR | 8’h0C |
| Multiply | MUL | 8’h0D |

**ALU OP Codes**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Code** | **OpCode** |
| Forward data 2 to the result | FORWARD | 3’b000 |
| Add the two operands | ADD | 3’b001 |
| Bitwise and the operands | AND | 3’b010 |
| Bitwise or the operands | OR | 3’b011 |
| Multiply the operands | MUL | 3’b100 |

1. **MUL Instruction**

For this operation, I used an 8-bit array multiplier because this process should be done within a clock cycle time.



I used an arrangement like shown in the picture. The eight adding layers can be reduced to three layers by doing some additions in parallel.

#2 add(layer1, layer2); add(layer3, layer4); add(layer5, layer6); add(layer7, layer8)

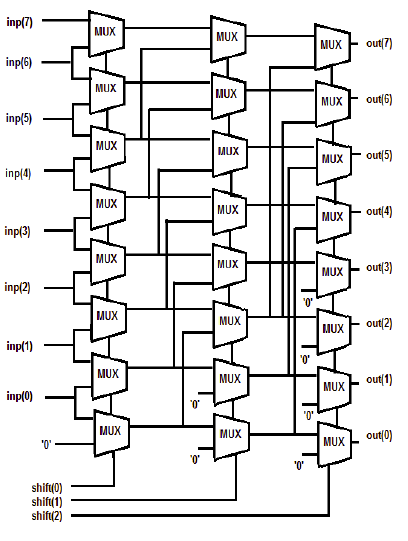
#2 add(2\_layer1, 2\_layer2); add(2\_layer3, 2\_layer4)

#2 add(3\_layer1, 3\_layer2);

When we take the adder gets #2 time unit delay then the multiplication process takes 6 time units.

1. **Barrel Shifter**

I used a barrel shifter module in parallel to the ALU to do the shift operations. I used this method because in our CPU we don’t do shifting operations in combination with the arithmetic operations. In our instruction encoding format, we can’t do such complicated operations. So I fixed the barrel shifter parallel to the CPU. the below figure shows the left shift barrel shifter, but I cave change that a little bit to accomplish the four types of shifts. I have used an additional 7 MUXs to the point where the below diagram gives static 0 inputs. By using that I have made **Logical Right Shift, Arithmetic Right Shift, Rotate Right**. And I have added two 2 to 1 8bit MUXs in order to do the **Left Shift Logical** operation.



I have generated a separate opcode to the barrel shifter.

**Barrel Shifter OP Codes**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Code** | **OpCode** |
| Logical Right Shift | SRL | 2’b00 |
| Arithmetic Right Shift | SRA | 2’b01 |
| Rotate Right | ROR | 2’b10 |
| Logical Left Shift | OR | 2’b11 |

1. **BNE Operation**

BNE operation data path is shown in the above data path figure. I inverted the ALU comparator signal and feed that to a and gate along with the BNE control signal. After that, I feed that signal to the or gate which I previously used to do the jump and BEQ operations.