Jooyoun Hong

Dec. 10. 2014

**EC413 Project Report**

The Multicycle CPU includes MUX, Registers, Instruction memory, Data Memory, Register File, Zero-Extension, Sign-Extension, ALU, Controller, and Datapath.

***Controller (Conroller4.v)***

***Control Signals***

***PCSource – Controls Next PC Source***

|  |  |
| --- | --- |
| **00** | Current PC + 1 is sent to the PC for writing |
| **01** | Jump Target Address is sent to the PC for writing |
| **10** | Branch Target Address is sent to the PC for writing |

***ALUSrcA – Controls ALU Operand1 Source***

|  |  |
| --- | --- |
| **00** | Current PC |
| **01** | Content of R2 |
| **10** | 0 |

***ALUSrcB – Controls ALU Operand2 Source***

|  |  |
| --- | --- |
| **00** | Content of R1 or R3 (Depends upon ReadSel) |
| **01** | 1 |
| **10** | Sign-Extended Immediate |
| **11** | Zero-Extended Immediate |

***ReadSel – Decides the Register Being Read for ReadData2***

|  |  |
| --- | --- |
| **0** | Read from R1 |
| **1** | Read from R3 |

***MemtoReg – Decides the Write Back Data***

|  |  |
| --- | --- |
| **00** | ALUOut |
| **01** | M [ZE (Imm)] |
| **10** | {R1 [31:16], ZE [15:0]} |
| **11** | {ZE [15:0], R1 [15:0]} |

***ALUOp***

|  |  |
| --- | --- |
| **0000** | MOV |
| **0001** | NOT |
| **0010** | ADD |
| **0011** | SUB |
| **0100** | OR |
| **0101** | AND |
| **0110** | XOR |
| **0111** | SLT |

***Enables***

|  |  |
| --- | --- |
| **PCWrite** | PC is written |
| **PCWriteCond** | PC is written only if the Zero wire is enabled |
| **IRWrite** | Instruction Register takes the new instruction |
| **RegWrite** | Write into register file |

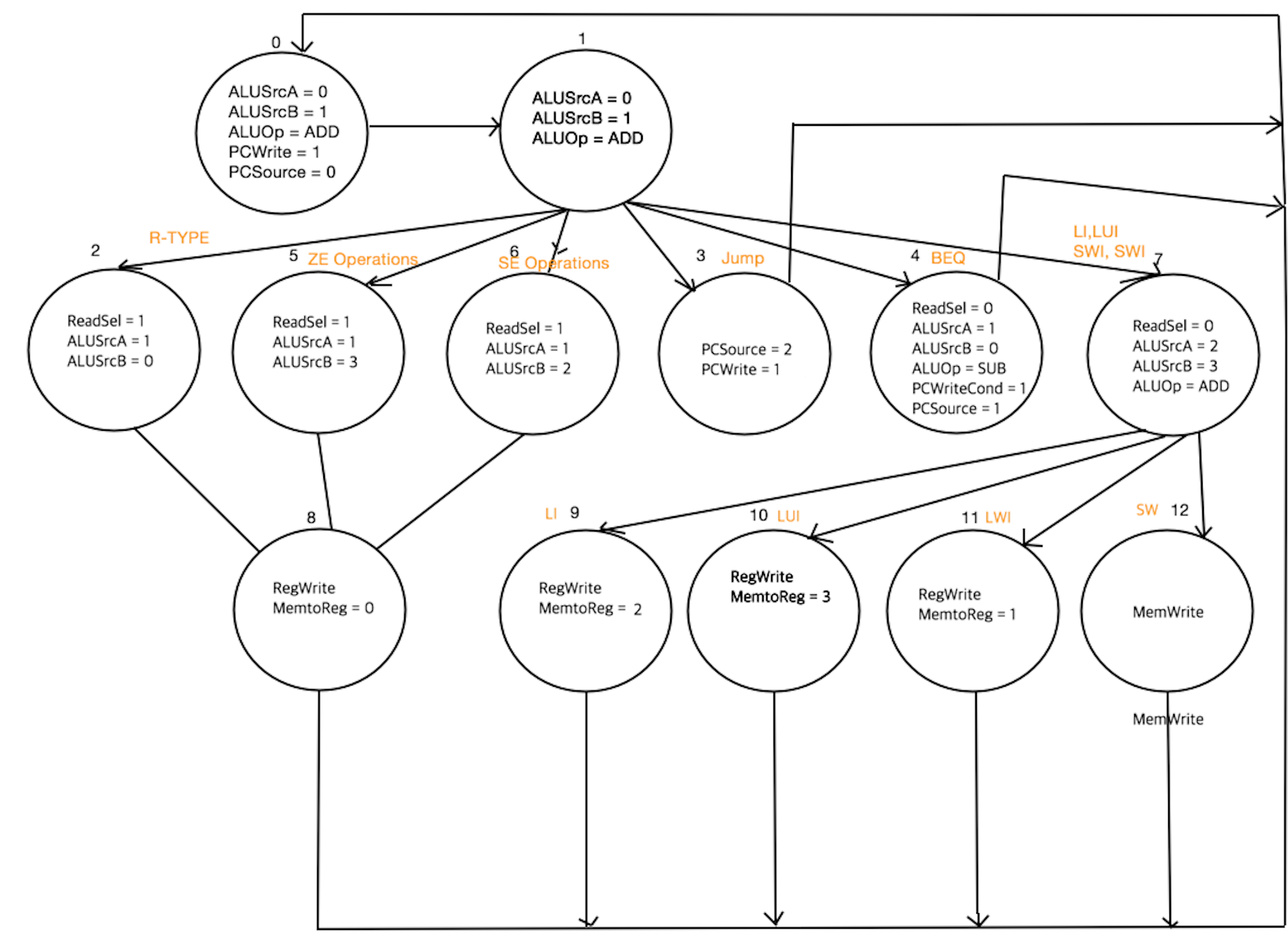
******

Figure 1. State Diagram

***State0***

Instruction is read from the instruction memory (IMem). Program Counter is incrementing by 1 at this state by setting ALUSrcA as 0, which is PC and setting ALUSrcB as 1, hard-wired 1 and setting the ALU Opcode as add (0010). PCWrite is enabled and PCSource is sending out the signal 0 to PCSource Mux to take Current PC + 1 as the next PC.

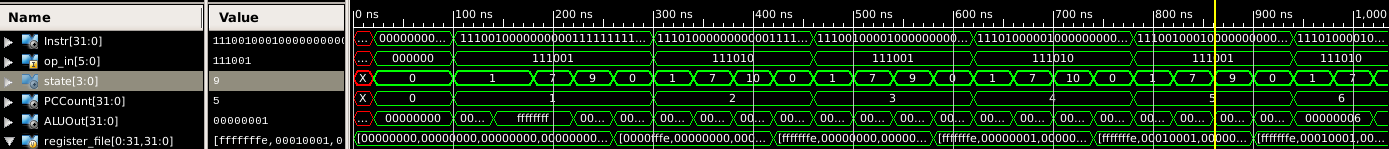
******

Figure 2. PC Increment

***State1***

The purpose of this state is to calculate Jump or Branch Target address calculation in case of the instruction is Jump or BEQ. The calculated address will be used if the current instruction is Jump or BEQ, otherwise it does not affect further processing. The Jump or Branch target address is calculated by adding current program counter and sign extended immediate. Unlike the original instructions in Multicycle CPU we learned in the class, whose instructions start at address that multiple of four, this project consists of sequenced 32-bit instructions. Therefore, there is no need of left-shift 2-bit (which is equivalent to multiplying by 4). ALUSrcA is 0, the current PC, ALUSrcB is 2, the sign-extended immediate, and ALUOp is 0010 referring addition.

***State2 (R-type Arithmetic and Logical Instructions)***

The purpose of this state is operation of R-type arithmetic and logical operations. All of these instructions are reading R2 and R3, so the Operand2 for ALU always be contents of R3, which is induced by setting ReadSel signal as 1. ALUSrcA is set to 1, getting contents from R2, and ALUSrcB is set to 0, getting contents from R3. ALUOp depends upon the instruction opcode. The next state is state8.

***State3 (Jump Instruction)***

The purpose of this state is operation of Jump Instruction. Unlike BEQ, there is no condition for this instruction to be taken, so ALUSrcA and ALUSrcB at this point do not matter that much. PCSource is set to 2 which is Jump target address and PCWrite is enabled, and therefore PCEn is enabled. The next state is state0.

***State4 (BEQ Instruction)***

The purpose of this state is operation of BEQ Instruction. It is more complicated instruction than jump instruction the condition for the branch to be taken depends upon the ALUOut. Since it is comparing contents from R2 and R1, so the ReadSel signal is set to 0. To check if the two values are the same, the ALU operates subtraction. If the ALUResult from subtraction is 0 then the Zero wire is set to 1. This Zero wire goes through the AND gate with PCWriteCond, which already set to be 1 by the controller. This gate system prevents the PCEn is enabled when the two contents are not the same. Only when the Zero wire is high, the system allows the new branch target address. PCSource is set to 1, which is branch target address that is calculated from state2. The next is state0.

***State5 (Arithmetic or Logical Operations with Zero-Extended Immediate)***

The purpose of this state is operation of Arithmetic or Logical operation taking zero-extended immediate as ALU Operand2. ALUSrcA is set to 1, taking the value from R2, and ALUSrcB is set to 3 taking zero-extended immediate. ALUop depends upon the instructions. Although the FSM diagram indicates ReadSel to be 1, it does not matter which to set, because the content from the second register is being used for these instructions. The next state is state8.

***State6 (Arithmetic or Logical Operations with Sign-Extended Immediate)***

The purpose of this state is operation of Arithmetic or Logical operation taking sign-extended immediate as ALU Operand2. ALUSrcA is set to 1, taking the value from R2, and ALUSrcB is set to 2, taking sign-extended immediate. ALUop depends upon the instructions. Although the FSM diagram indicates ReadSel to be 1, it does not matter which to set, because the content from the second register is being used for these instructions. The next state is state8.

***State7 (Load and Store)***

The purpose of this state is operation load and store. To satisfy the SWI reading from R1, and to concatenate half of R1’s data for LI and LUI at the next state, the ReadSel is set to 0. Since these instructions are not much related to the ALU Operations but they have to go through anyway, I decided to add the zero-extended immediate with hard-wired zero. ALUSrcA is set to 3, taking 0 as ALU Operand1, and ALUSrcB is set to 2, taking sign-extended immediate as ALU Operand2. ALUOp is set to ADD. The next states of each instruction are all separated from state9 to state12.

***State8 (Finishing up Arithmetic or Logical Operations)***

The purpose of this stage is to write back the value calculated by ALU to the register file. MemtoReg signal is set to 0, taking ALUOut as a write-back data. The signal RegWrite is also enabled to take the value. The next state is state0.

***State9 (LI)***

The purpose of this state is to keep process the LI instruction flowed from state7. MemtoReg is set to 2, taking concatenation of {R1 [31:16], ZE [15:0]} as write-back data. Also, RegWrite signal is enabled to take the value. The next state is state0.

***State10 (LUI)***

The purpose of this state is to keep process the LUI instruction flowed from state7. MemtoReg is set to 3, taking concatenation of {ZE [15:0], R1 [15:0]} as write-back data. Also, RegWrite signal is enabled to take the value. The next state is state0.

***State11 (LWI)***

The purpose of this state is to keep process the LUI instruction flowed from state7. MemtoReg is set to 1, taking M[ZE(Imm}] as a write-back data. Also, RegWrite signal is enabled to take the value. The next state is state0.

***State12 (SWI)***

The purpose of this state is to keep process the SWI instruction flowed from state7. ALUOut is being data memory address to write the data, and the MemWrite signal is enabled to store the value. The next state is state0.

***Instruction Executions***

***(Detailed explanations for each states and instructions are above)***

***R-type Logical and Arithmetic Instructions***

* State Flow: 0 1 2 8 0
* ALUOp is fed into ALU for specified computation.
* The result of computation is stored in R1.

***Jump***

* State Flow: 0 1 3 0
* At state1, the Jump Target address is calculated by adding PC and sign-extended Immediate.
* PCWrite is enabled to over write the current PC to (PC + SE(Imm)).

***BEQ***

* State Flow: 0 1 4 0
* At state1, the Branch Target address is calculated by adding PC and sign-extended Immediate.
* PCWrite is not enabled, but PCWriteCond is so that the computed new target address can overwrite the current PC only when the Zero wire is high (When the contents of 2 registers are the same).

***I-type Arithmetic and Logical Instructions***

* State Flow: Zero-Extended Immediate – 0 1 5 8 0

: Sign-Extended Immediate – 0 1 6 8 0

* ALUOp is fed into ALU for specified computation.
* The result of computation is stored in R1.

***Load Immediate***

* State Flow: 0 1 7 9 0
* Immediate is replacing original R1[15:0] bits.
* At Write Back state, the concatenation is stored in R1.

***Load Upper Immediate***

* State Flow: 0 1 7 10 0
* Immediate is replacing original R1[31:16] bits.
* At Write Back state, the concatenation is stored in R1.

***Load Word Immediate***

* State Flow: 0 1 7 11 0
* Loading data from data memory whose address is zero extended immediate.
* At Write Back state, the data is stored in R1.

***Store Word Immediate***

* State Flow: 0 1 7 12 0
* Storing data from R1 to data memory whose address is zero extended immediate.
* No Write Back to register file.

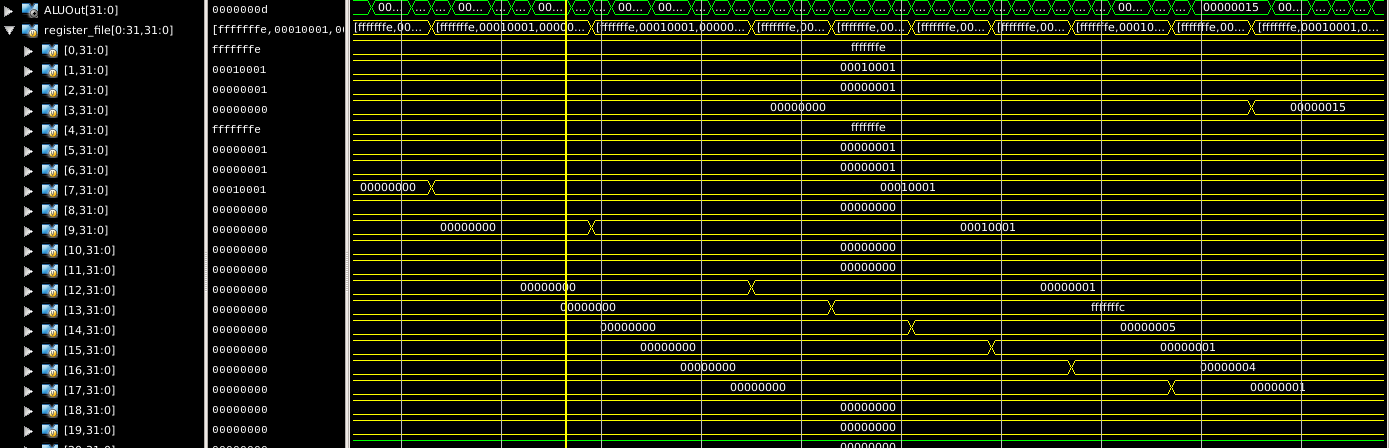


Figure 3. ALUOut stored in Register File



Figure 4. Jump Instruction

***Datapath (Datapath2.v)***

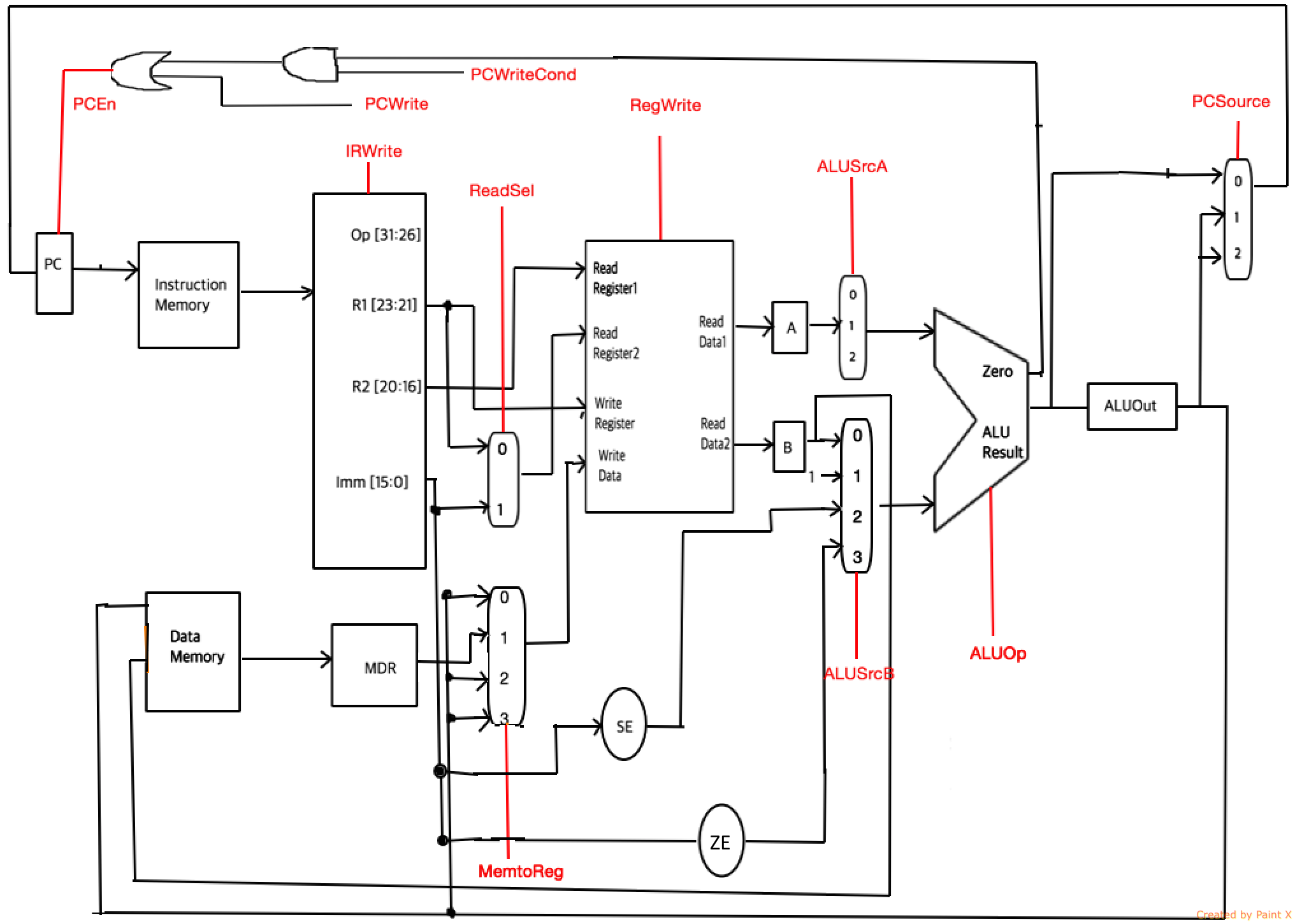
* MUX – Takes the control signals and send out the right output for current state.
* IMem – Storing 32-bits of Instructions.
* DMem – Storing 32-bits of Data.
* General Purpose Register – Holding the temporary data for a single cycle.

Figure . Datapath