

**CEE210/CEE216**

**Computer Engineering 2nd year computer/Commination Spring 2022**

**Faculty of Engineering Electrical Engineering Dep. Instructor: Dr.Gihan Naguib**

**Computer architecture project**

16-bit MIPS Processor Implementation

**Prepared by:**

* Diea Abdeltwab (EECE)
* Mostafa Mohamed Shaban (EECE)
* Hosni Abdelaziz (EECE)
* Shehab Mahmoud Kilany (CSE)

**Supervised by:**

***Dr.Gihan Naguib.***

***Eng.Jihad Awad.***

**Single Cycle**

***Table of contents***

***Introduction…………………………………………………………………………………***

***Main components………………………………………………………………………..***

***Register file:***

* *Design………………………………………………………………………………*
* *Description……………………………………………………………………….*

***ALU:***

* *Design………………………………………………………………………………*
* *Description……………………………………………………………………….*
* *Problems in design…………………………………………………………..*
* *Solutions…………………………………………………………………………..*

***Control unit:***

* *Design………………………………………………………………………………*
* *Description……………………………………………………………………….*
* *Truth table……………………………………………………………………….*

***ALU control:***

* *Design………………………………………………………………………………*
* *Description……………………………………………………………………..*

***PC controller:***

* *Design………………………………………………………………………………*
* *Description……………………………………………………………………..*

***Introduction:***

We will design a 16-bit mips processor which has many functionalities such as

doing arithmetic and logic operations and has instructions to compute or manipulate data values using registers ,change or retrieve values in read/write memory and preform a relational tests between data values and control program flow.

***Main components:***

|  |  |
| --- | --- |
| **component** | **role** |
| RAM | *Used to write or read data on it.* |
| ALU | *Used to do the arithmetic and logical operations.* |
| Register | *Used to save data in order to do operations on the data* |
| Control unit | *used to control the data paths by sending signals* |
| PC | *Used to determine which instruction to be fetched* |
| ALU control | *Used to send signals to ALU* |

***Register file:***

* Design:

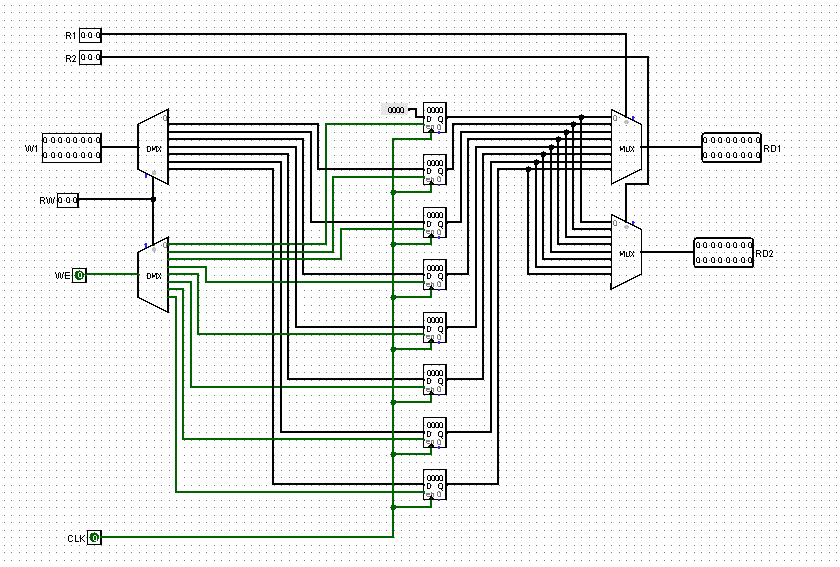


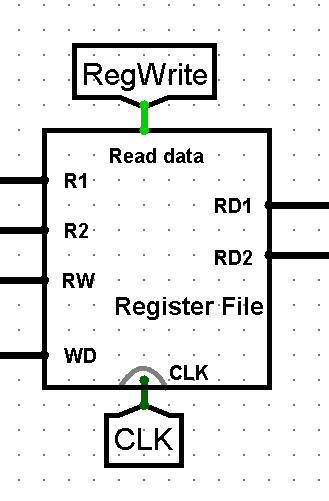
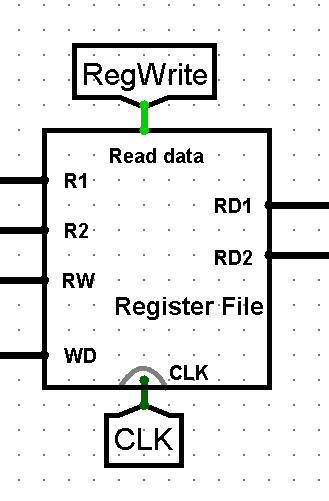
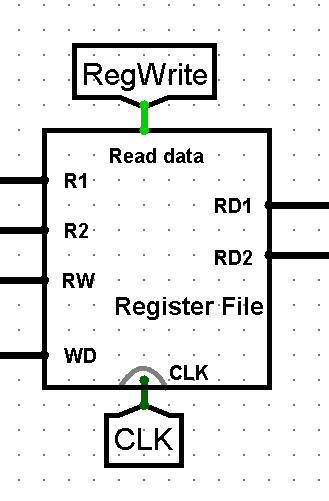
Figure (1.a) 

Figure (1.b)



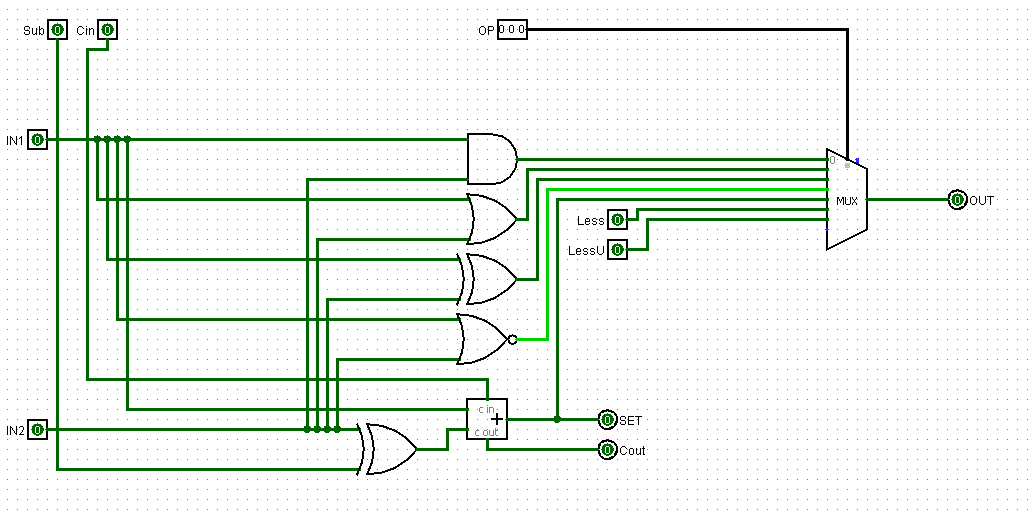
* Description:

There is a seven 16-bit generalpurpose registers: R1 through R7. R0 is hardwired to zero and cannot be written as shown in figure (1.a).

It takes 2 inputs from instruction to decide which register to be shown in the output also it take a register to write data on it when the register write is 1 as shown in figure (1.b).

***ALU:***

* Design:



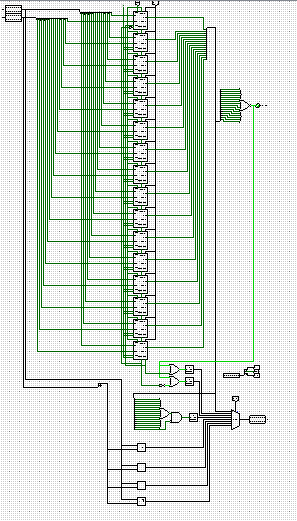


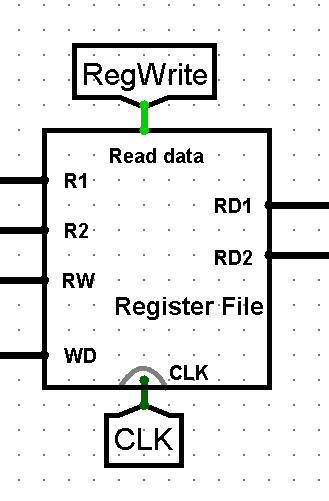
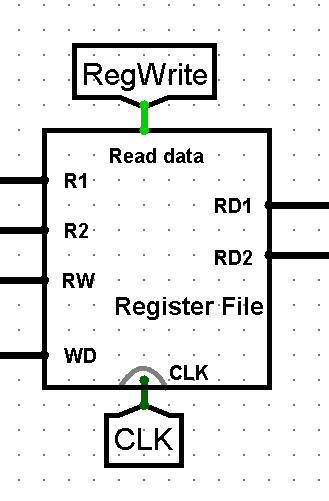
Figure (2.a)

Figure (2.b)

* Description:

The figure (2.a) is representing the logical, adding and subtracting operation.

The 16 bit ALU is responsible to do the operation of 1 bit ALU plus shifting and comparing as shown in figure (2.b).

* Problems:

How to take signals for 2 mux from ALU control and doing slteu.

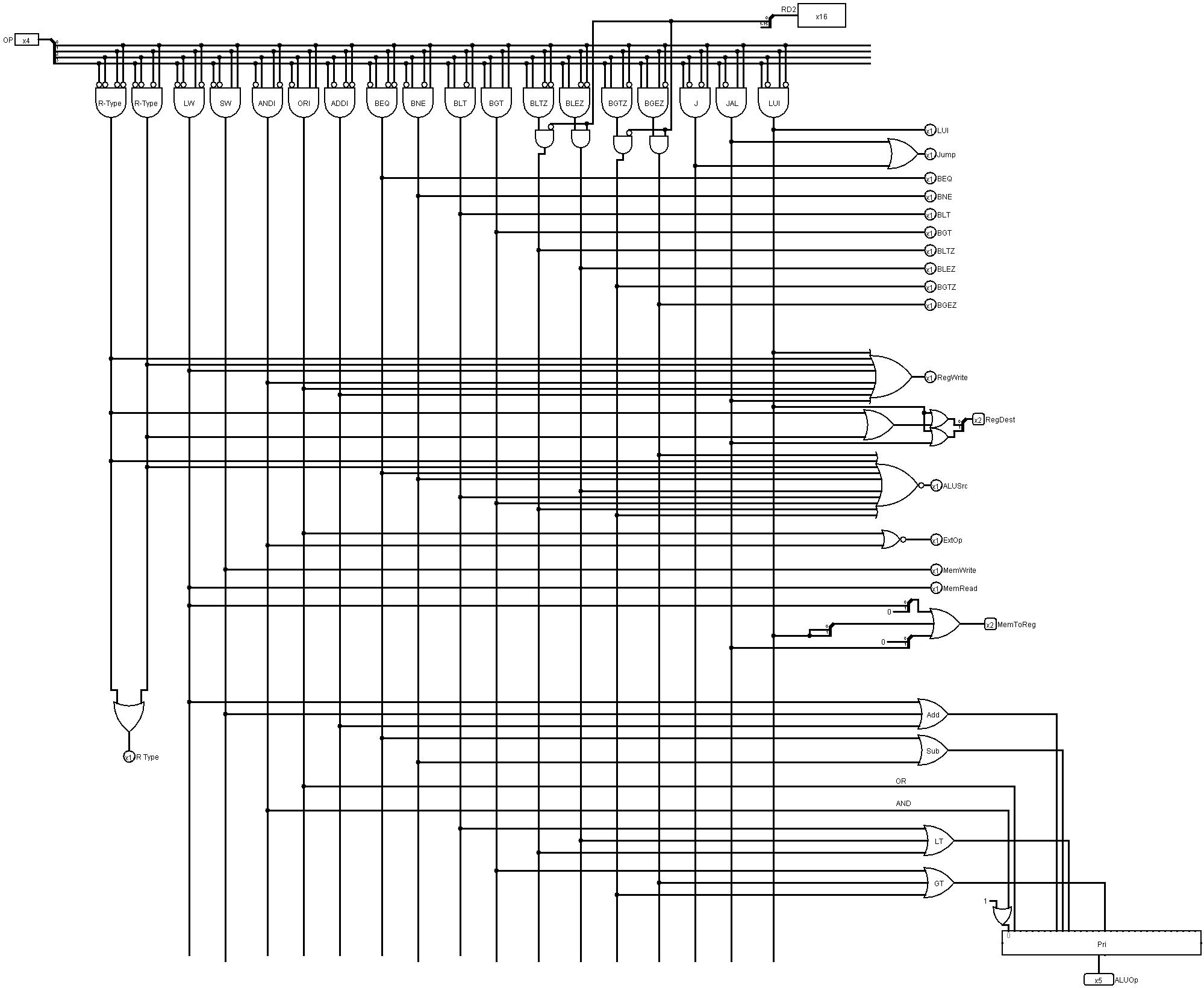
* solution:

We solved the 2 mux problem by using priority encoder in ALU control.

By trying different patterns we found out that the slteu is the negation of the carry of the last bit.

***Control unit:***

* Design:

******

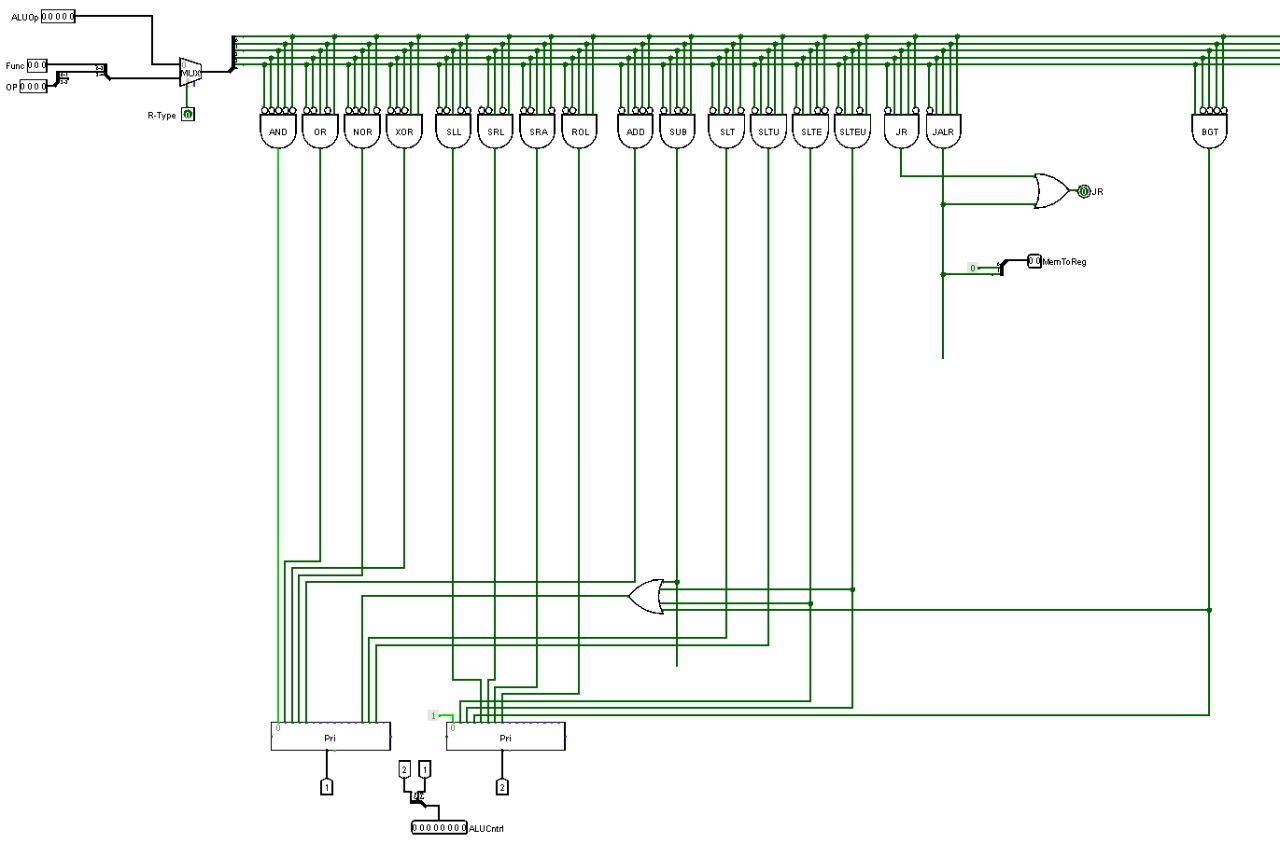
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | PC | RegDes | RegWrite | Extop | ALUsrc | MemR | MemW | Mem to Reg |
| AND | 000 | 01 | 1 | X | 0 | 0 | 0 | 00 |
| OR | 000 | 01 | 1 | X | 0 | 0 | 0 | 00 |
| NOR | 000 | 01 | 1 | X | 0 | 0 | 0 | 00 |
| XOR | 000 | 01 | 1 | X | 0 | 0 | 0 | 00 |
| SLL | 000 | 01 | 1 | X | 0 | 0 | 0 | 00 |
| SRL | 000 | 01 | 1 | X | 0 | 0 | 0 | 00 |
| SRA | 000 | 01 | 1 | X | 0 | 0 | 0 | 00 |
| ROL | 000 | 01 | 1 | X | 0 | 0 | 0 | 00 |
| ADD | 000 | 01 | 1 | X | 0 | 0 | 0 | 00 |
| SUB | 000 | 01 | 1 | X | 0 | 0 | 0 | 00 |
| SLT | 000 | 01 | 1 | X | 0 | 0 | 0 | 00 |
| SLTU | 000 | 01 | 1 | X | 0 | 0 | 0 | 00 |
| SLTE | 000 | 01 | 1 | X | 0 | 0 | 0 | 00 |
| SLTEU | 000 | 01 | 1 | X | 0 | 0 | 0 | 00 |
| JR | 100 | X | 0 | X | X | 0 | 0 | X |
| JALR | 100 | 01 | 1 | X | X | 0 | 0 | 10 |
| LW | 000 | 00 | 1 | 1 | 1 | 1 | 0 | 01 |
| SW | 000 | X | 0 | 1 | 1 | 0 | 1 | X |
| ANDI | 000 | 00 | 1 | 0 | 1 | 0 | 0 | 00 |
| ORI | 000 | 00 | 1 | 0 | 1 | 0 | 0 | 00 |
| ADDI | 000 | 00 | 1 | 1 | 1 | 0 | 0 | 00 |
| BEQ | 001 | X | 0 | 1 | 0 | 0 | 0 | X |
| BNE | 001 | X | 0 | 1 | 0 | 0 | 0 | X |
| BLT | 001 | X | 0 | 1 | 0 | 0 | 0 | X |
| BGT | 001 | X | 0 | 1 | 0 | 0 | 0 | X |
| BLTZ | 001 | X | 0 | 1 | 0 | 0 | 0 | X |
| BLEZ | 001 | X | 0 | 1 | 0 | 0 | 0 | X |
| BGTZ | 001 | X | 0 | 1 | 0 | 0 | 0 | X |
| BGEZ | 001 | X | 0 | 1 | 0 | 0 | 0 | X |
| J | 010 | X | 0 | X | X | 0 | 0 | X |
| JAL | 010 | 10 | 1 | X | X | 0 | 0 | 10 |
| LUI | 000 | 11 | 1 | X | X | 0 | 0 | 11 |

* Description:

The control unit is the brain of the CPU it takes the op code and decide which operation to be done and after that it decides the signals according to op code for R type and for I type we used the priority encoder to do R type operation we need.

***ALU control:***

* Design:

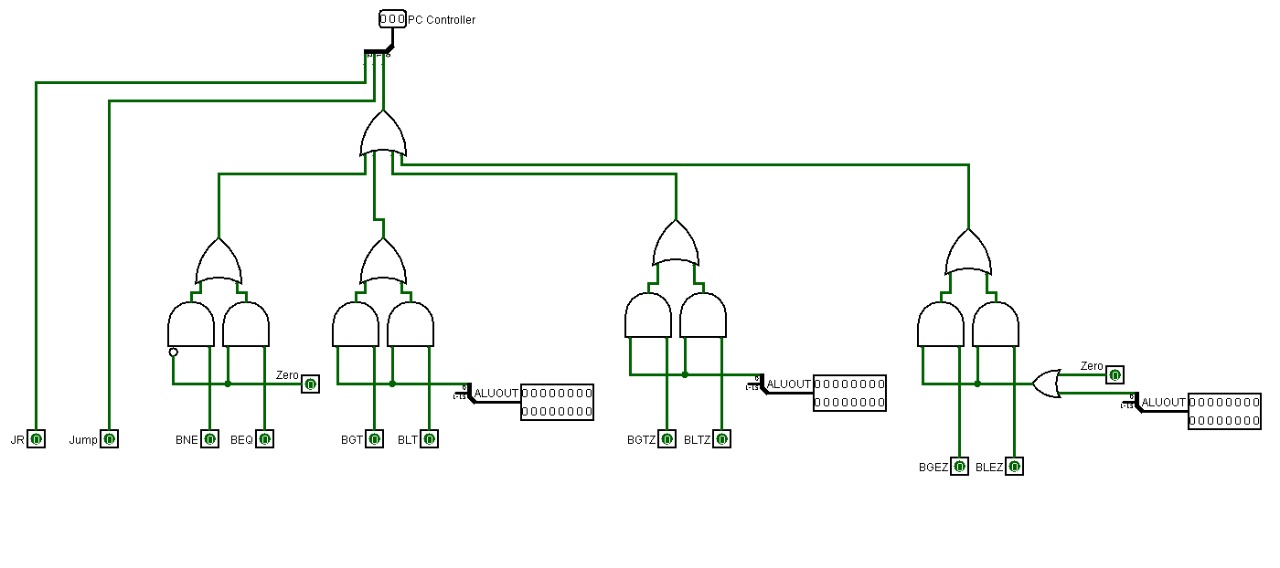


* Description:

ALU control takes the op code and function for R type and for the other types it takes the code which was generated by the control unit to decide which operation to be done and add them to 2 priority encoder to generate a new code to control the 2 mux in ALU.

***Pc control:***

* Design:



* Description:

It takes the signal from control unit and checking whether we branch by comparing it the less than or greater than bits from the ALU or with the zero flag and for jump it jumps when signal of jump is one.

**Pipeline**

***Table of contents***

***Introduction………………………………………………………………………………..***

***Pipeline Stages:***

* *Design…………………………………………………………………………………….*
* *Description……………………………………………………………………………..*

***Hazards:***

* *Hazards types……………………………………………………………………………*

*Solutions:*

* *Stalling ……………………………………………………………………………………..*
* *Forwarding………………………………………………………………………………..*

***PC controller…………………………………………………………………………………..***

* *Killing………………………………………………………………………………………...*

***Test code………………………………………………………………………………………..***

***Introduction:***

Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end. Pipelining increases the overall instruction throughput.

***Pipeline Stages:***

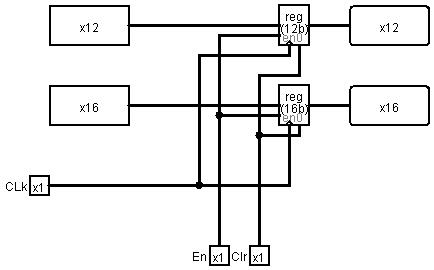
There are five stages in simple pipeline:

* **Fetching :** getting instruction from instructions memory.
* **Decoding :** starting to separate the instruction into op code, registers and immediate.
* **Executing :** the stage where the ALU works.
* **Memory :** the stage where the data memory could be accessed.
* **Write back :** the stage where the data is written in the register file.

In order to do different instructions in different stages we have to save them in registers and they are:

***IF/ID:***

* Design:

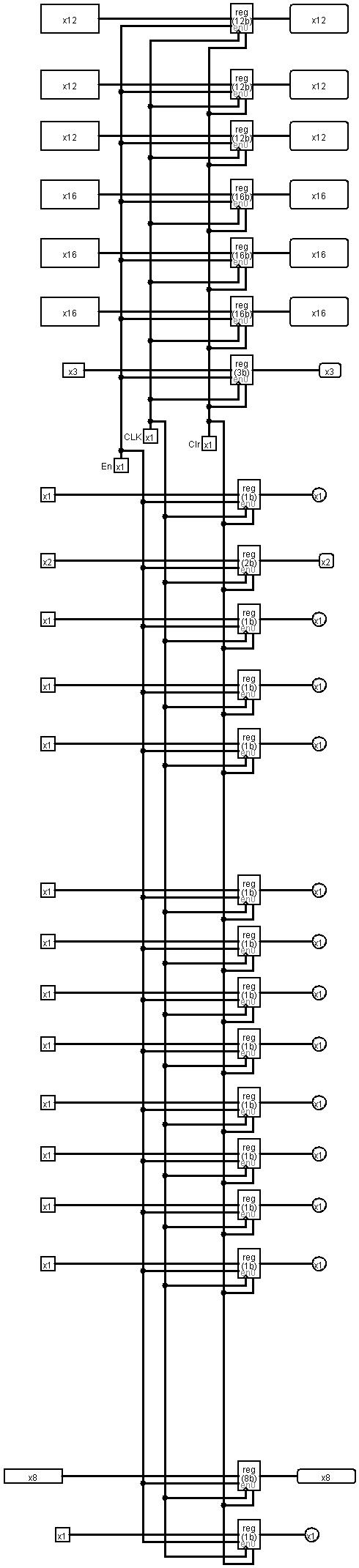


* Description:

These register saves the program counter in case we needed to do branch and saves the instruction as the computer doesn’t know which instruction would be executed.

***ID/EX:***

* Design:

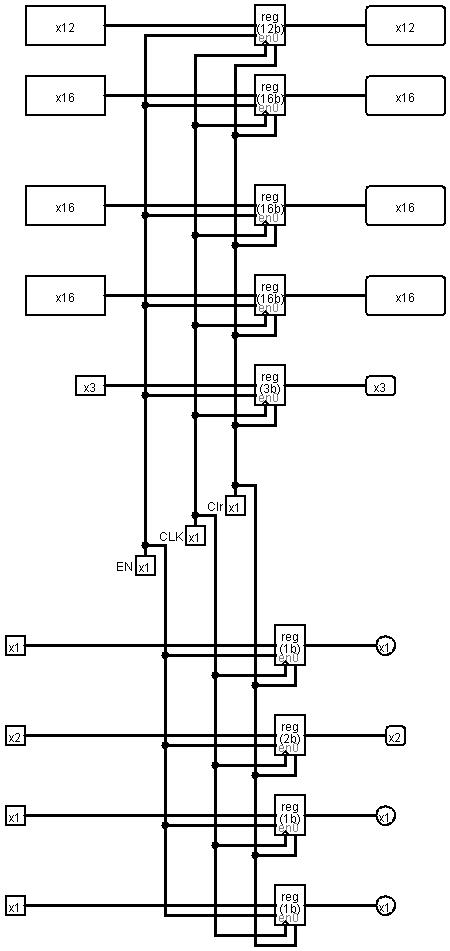


* Description:

This stage is used to store some of the signals of control unit that will be used in the next stages, immediate value, register destination, register values and PC+1.

***EX/MEM:***

* Design:

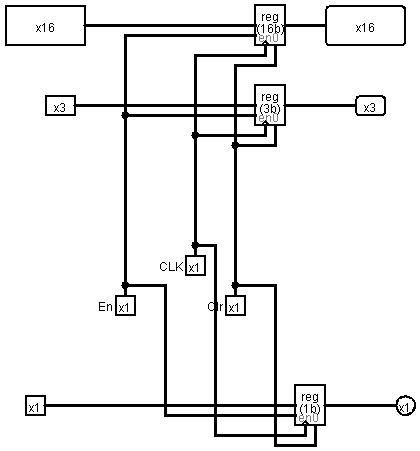


* Description:

This stage is used to store some signals from control unit, ALU result which could be the effective address in SW and LW or the logical and arithmetic operations, register destination ,rt value and, PC+1 and LUI result.

***MEM/WB:***

* Design:



* Description:

It is used to store the register write signal, register destination and the value of the register that will be written back.

***Hazards:***

* *Introduction:*

Ideally we expect a CPI value of 1 and a speedup equal to the number of stages in the pipeline. But, there are a number of factors that limit this. The problems that occur in the pipeline are called hazards. Hazards which shown in the pipeline prevent the next instruction from executing during its designated clock cycle. There are three types of hazards:

* Structural hazards: Hardware cannot support certain combinations of instructions (two instructions in the pipeline require the same resource).

Solution:

Using a separate instruction and data memory and making all instructions in five clock cycles.

* Data hazards: Instruction depends on result of prior instruction still in the pipeline

Solution:

Taking data whenever it is ready to be taken in any stage using (forwarding).

* Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).

Solution:

Using signals to insert no operation in the stages before the branch and jump.

***Forwarding:***

Conditions of forwarding:

* if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA=01 (Forward from EX/MEM pipe stage)
* if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01 (Forward from EX/MEM pipe stage) ϖ Detecting RAW hazard with Second Previous
* if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10 (Forward from MEM/WB pipe stage)
* if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10 (Forward from MEM/WB pipe stage)
* if (WB.RegWrite and after WB.RegisterRd ≠ 0) and (after WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 11 (Forward from after WB pipe stage)

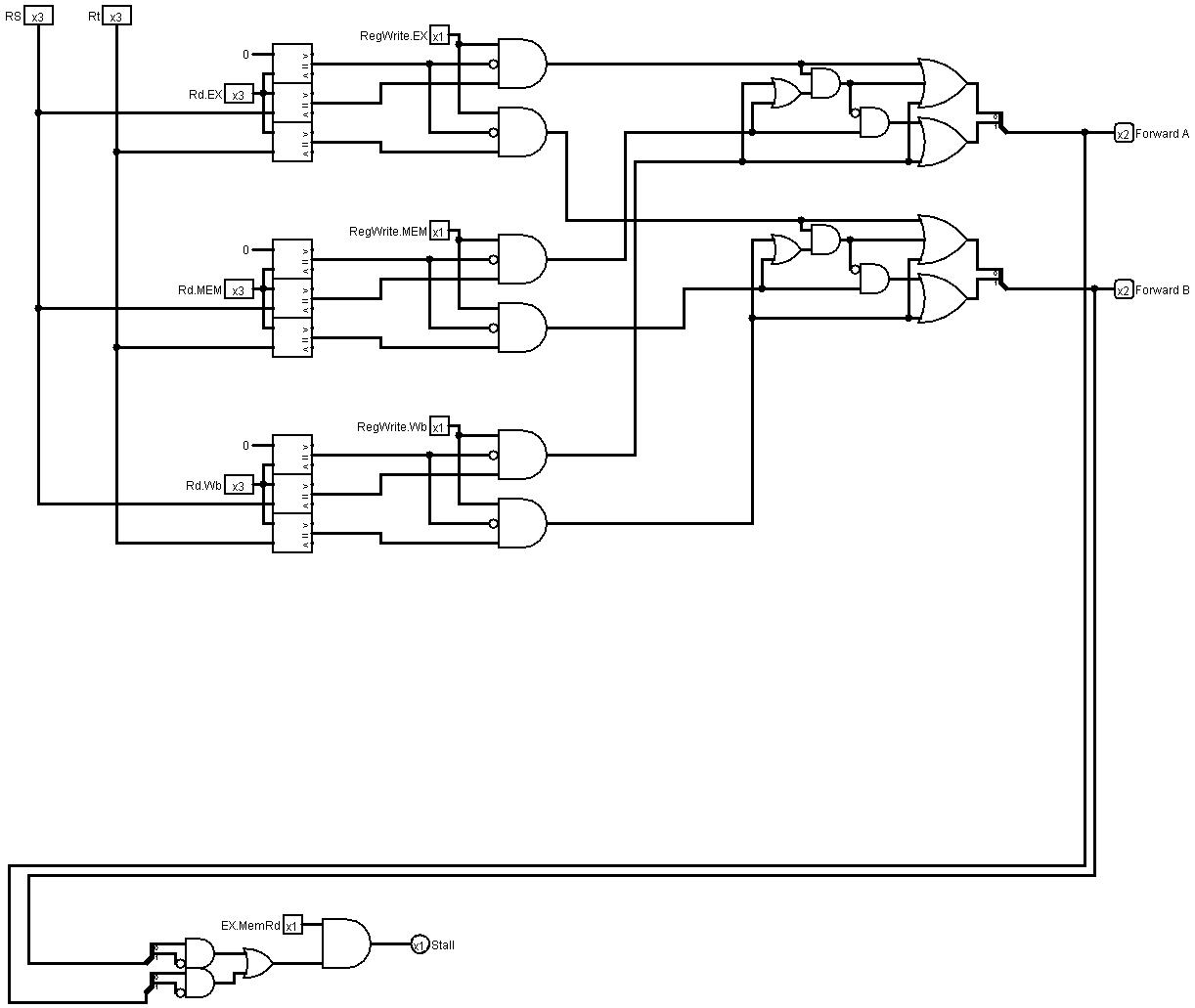
***Stalling:***

There was a problem in forwarding LW that it wasn’t arise till the memory stage so we need to stall for one cycle.

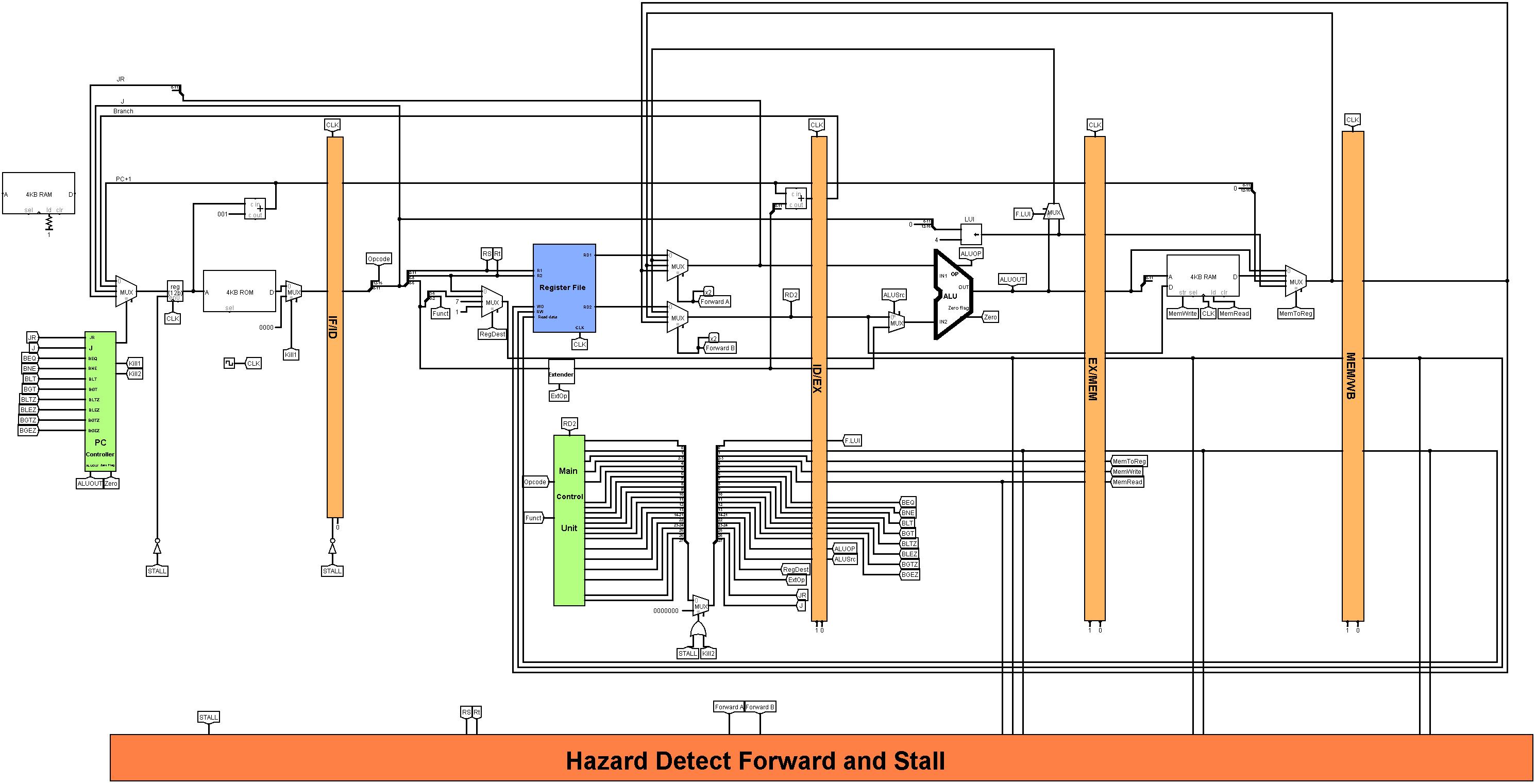
Condition for stalling:

The pipeline if ((EX.MemRd == 1) and (ForwardA==1 or ForwardB==1)) Stall.

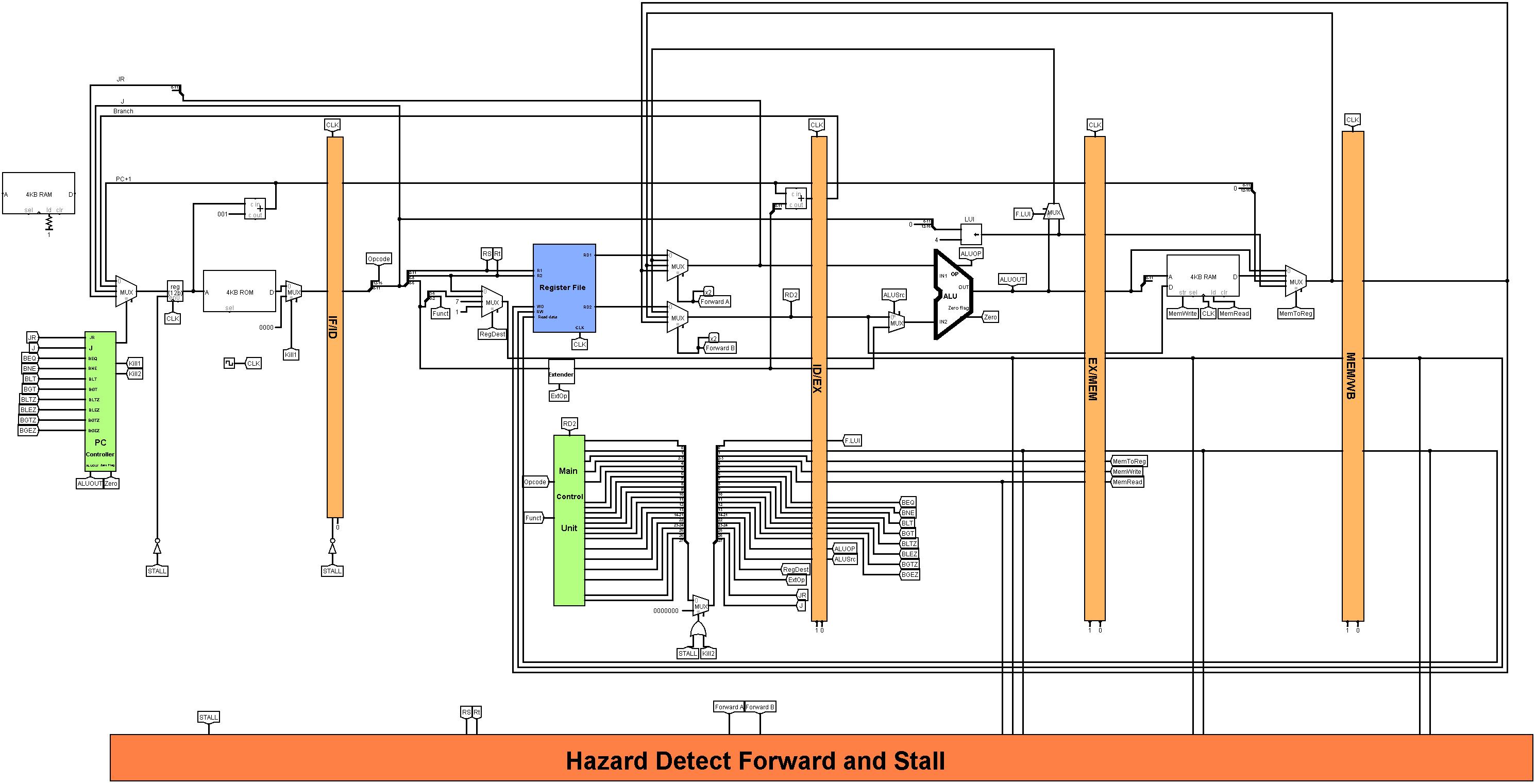
**Design of the forward and stalling unit:**

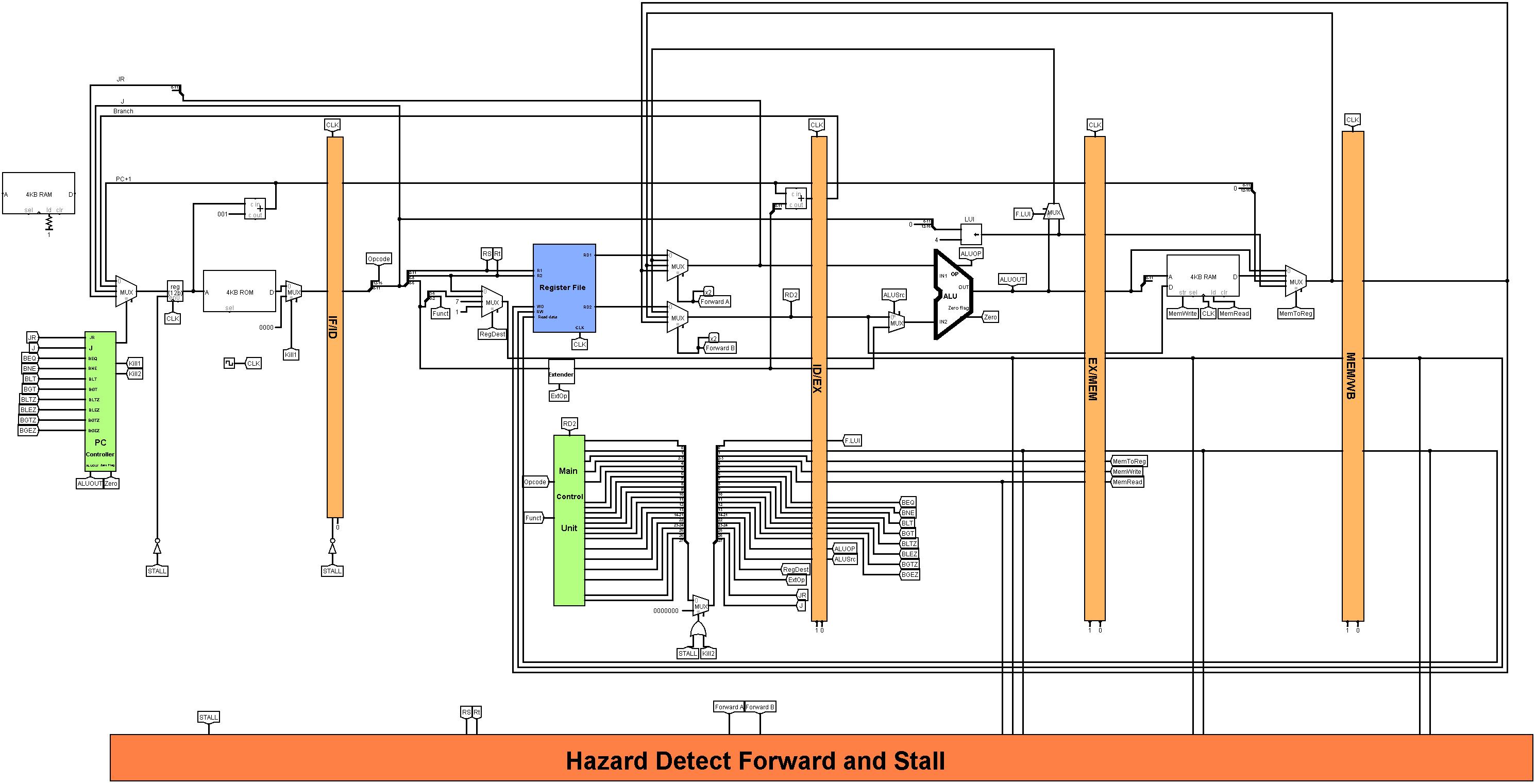


There was a problem in LUI that uses R1 and we needed to forward it to ALU.



Stall:

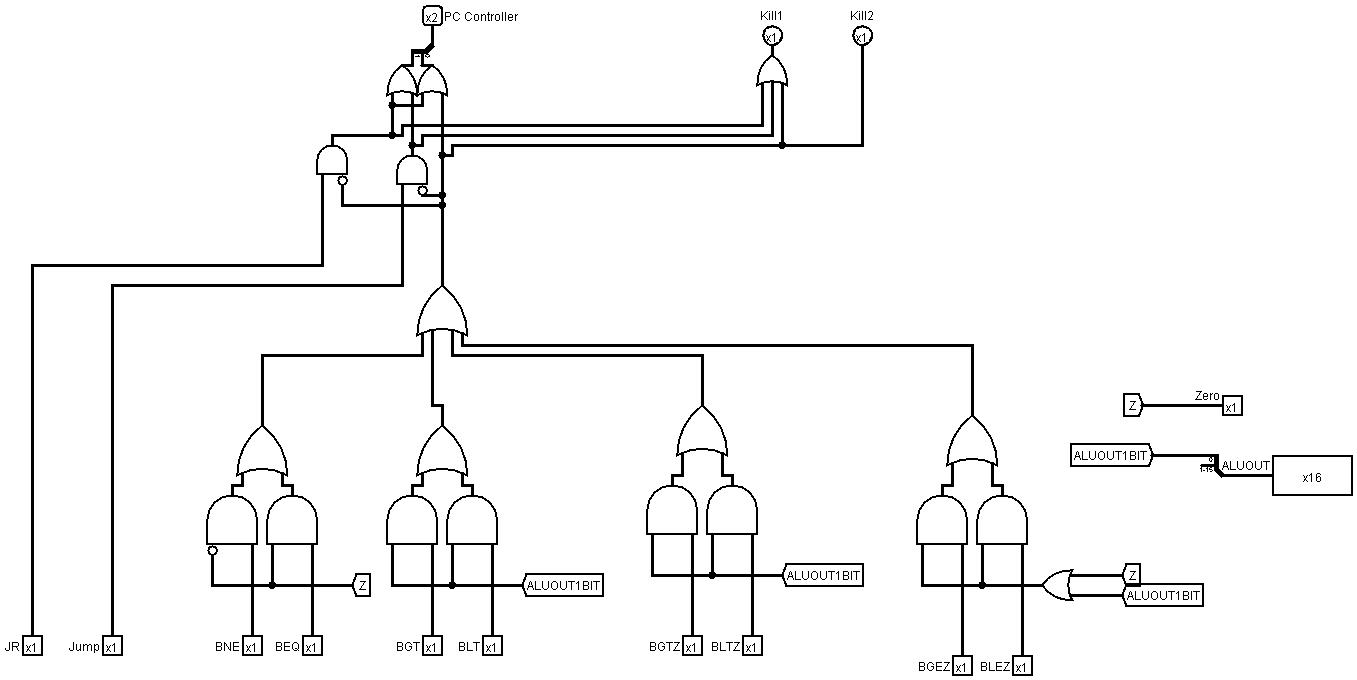


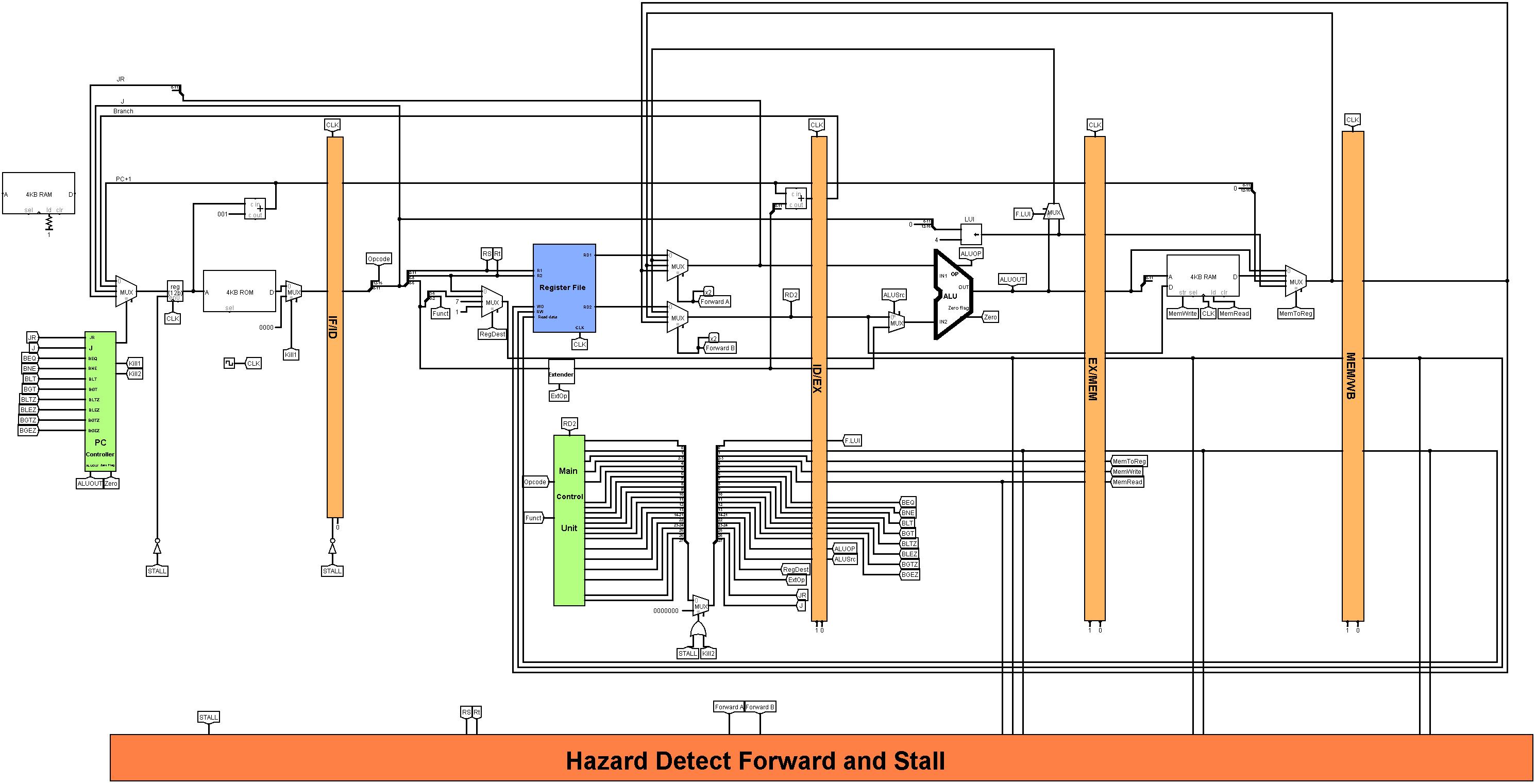


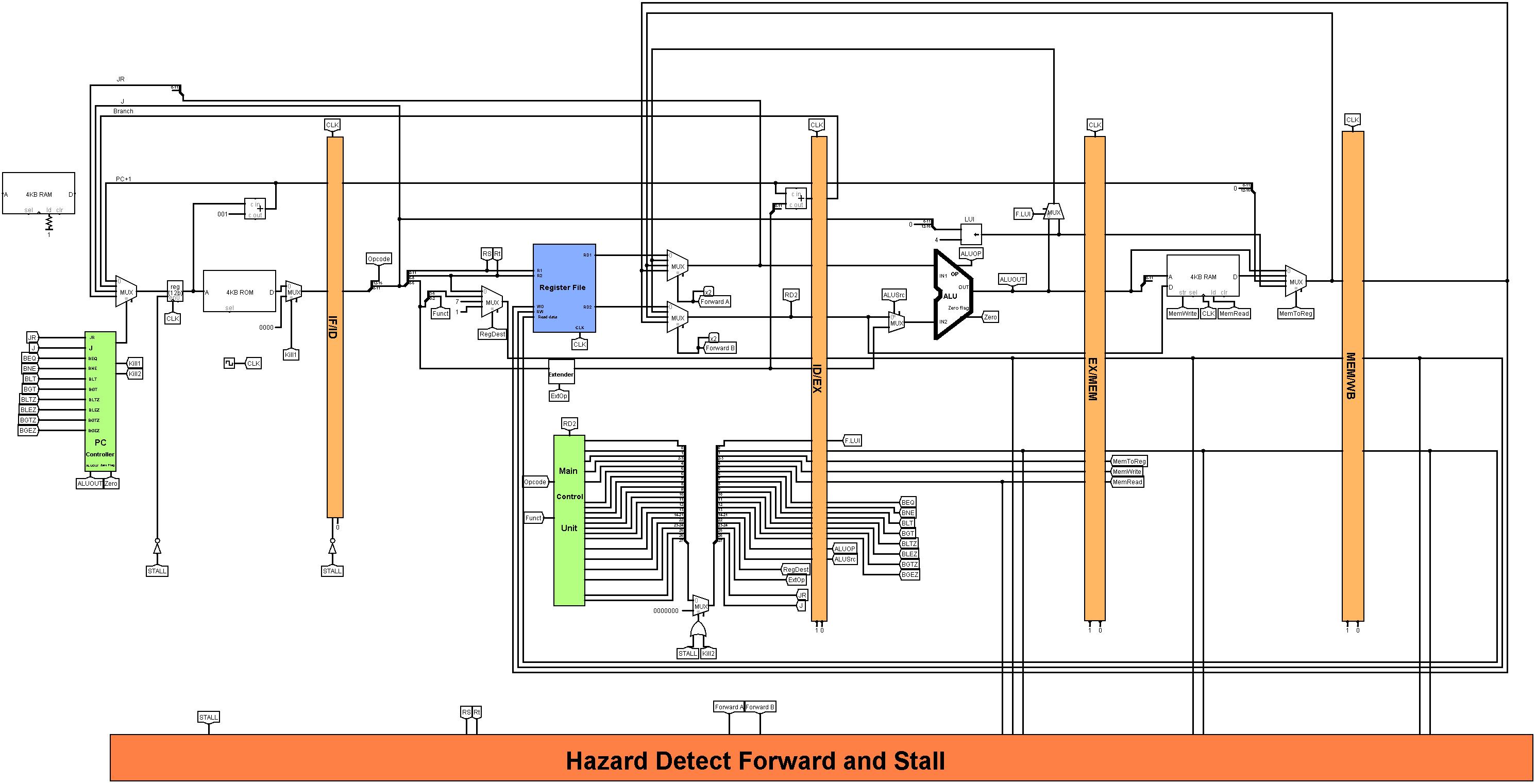
***PC controller:***

We needed to modify the circuit of the pc controller in order to do the killing to the instructions after taken branches and jump. The killing is aiming to insert no operation in the fetching and stop the signals of the control unit.

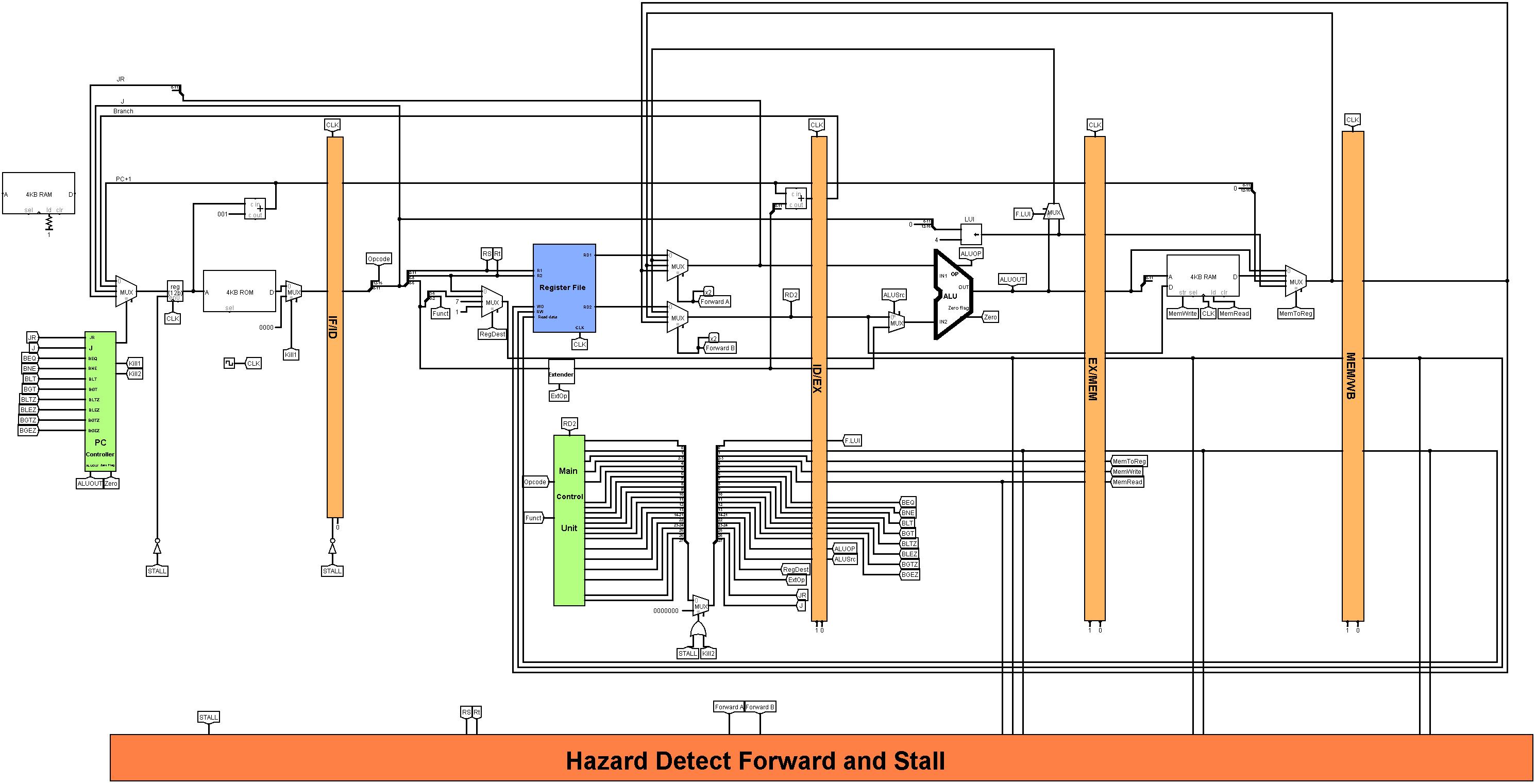
***Design:***







Full design



***Test code:***

|  |  |
| --- | --- |
| instructions | hexadecimal |
| Lui 0x384 | 0xA384 |
| Addi R5, R1,20 | 0x8354 |
| Xor R3, R1, R5 | 0x035B |
| Lw R1, 0(R0) | 0x2040 |
| Lw R2, 1(R0) | 0x2081 |
| Lw R3, 2(R0) | 0x20C2 |
| Addi R4, R4, 10 | 0x890A |
| Sub R4, R4, R4 | 0x1921 |
| Add R4, R2, R4 | 0x1520 |
| Slt R6, R2, R3 | 0x14F2 |
| Beq R6, R0, 2 | 0x4C02 |
| Add R2, R1, R2 | 0x1290 |
| Beq R0, R0, -5 | 0x403B |
| Sw R4, 0(R0) | 0x3100 |
| Jal func | 0xB012 |
| Sll R3, R2, R5 | 0x055C |
| Add R5, R5, R5 | 0x1B68 |
| beq r0,r0,-1 | 0x403F |
| Func: or R5, R2, R3 | 0x04E9 |
| Lw R1, 0(R0) | 0x2040 |
| Lw R2, 5(R1) | 0x2285 |
| Lw R3 ,6(R1) | 0x22C6 |
| And R4, R2, R3 | 0x04E0 |
| Sw R4, 0(R0) | 0x3100 |
| Jr R7 | 0x1E06 |

***Description:***

First loading 0x384 in R1 then add R1 to 20 in R5 then Xor R1 and R5 and put them in R3 which leads to get rid of the similar data then loading the data we have in the memory [0], memory [1] and memory [2] in R1, R2 and R3 then put in R4 10 in decimal and then put zero on it then make R4=R2 then see if R2 <= R3 then it will enter a loop till R2=10 and R4 has the cumulative sum to the numbers from 1 to 10 then the branch Beq R6, R0, 2 will be taken and put the value in the memory[0] then R5 will store 10 then load the value of the cumulative sum(55) in R1 and load memory[60] & memory[61] in R2 and R3 then and the values from R2 & R3 which leads to take the first and third bit from the hexadecimal in R2 take the second and fourth bit in R3 then store the value in the memory[0] then shift R2 with R5 value in R3 then multiply R5 by 2.

***Branching:***

*Taken and untaken branches:*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| branch | state | | | | | | | | | | |
| Beq R6, R0, 2 | N | N | N | N | N | N | N | N | N | N | T |
| Beq R6, R0, 2 | T | T | T | T | T | T | T | T | T | T | --- |

***Control hazard signals:***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instruction** | **Forward A** | **Forward B** | **Stall** | **Kill 1** | **Kill 2** |
| Lui 0x384 | 00 | 00 | 0 | 0 | 0 |
| Addi R5, R1,20 | 01 | 00 | 0 | 0 | 0 |
| Xor R3, R1, R5 | 10 | 01 | 0 | 0 | 0 |
| Lw R1, 0(R0) | 00 | 00 | 0 | 0 | 0 |
| Lw R2, 1(R0) | 00 | 00 | 0 | 0 | 0 |
| Lw R3, 2(R0) | 00 | 00 | 0 | 0 | 0 |
| Addi R4, R4, 10 | 00 | 00 | 0 | 0 | 0 |
| Sub R4, R4, R4 | 01 | 01 | 0 | 0 | 0 |
| Add R4, R2, R4 | 00 | 01 | 0 | 0 | 0 |
| Slt R6, R2, R3 | 00 | 00 | 0 | 0 | 0 |
| Beq R6, R0, 2 | 01 | 00 | 0 | 1 | 1 |
| Add R2, R1, R2 | 00 | 00 | 0 | 0 | 0 |
| Beq R0, R0, -5 | 00 | 00 | 0 | 1 | 1 |
| Sw R4, 0(R0) | 00 | 00 | 0 | 0 | 0 |
| Jal func | 00 | 00 | 0 | 1 | 0 |
| Sll R3, R2, R5 | 00 | 00 | 0 | 0 | 0 |
| Add R5, R5, R5 | 00 | 00 | 0 | 0 | 0 |
| beq r0,r0,-1 | 00 | 00 | 0 | 1 | 1 |
| Func: or R5, R2, R3 | 00 | 00 | 0 | 0 | 0 |
| Lw R1, 0(R0) | 00 | 00 | 0 | 0 | 0 |
| Lw R2, 5(R1) | 01 | 00 | 1 | 0 | 0 |
| Lw R3 ,6(R1) | 11 | 00 | 0 | 0 | 0 |
| And R4, R2, R3 | 10 | 01 | 0 | 0 | 0 |
| Sw R4, 0(R0) | 00 | 01 | 1 | 0 | 0 |
| Jr R7 | 00 | 00 | 0 | 1 | 0 |