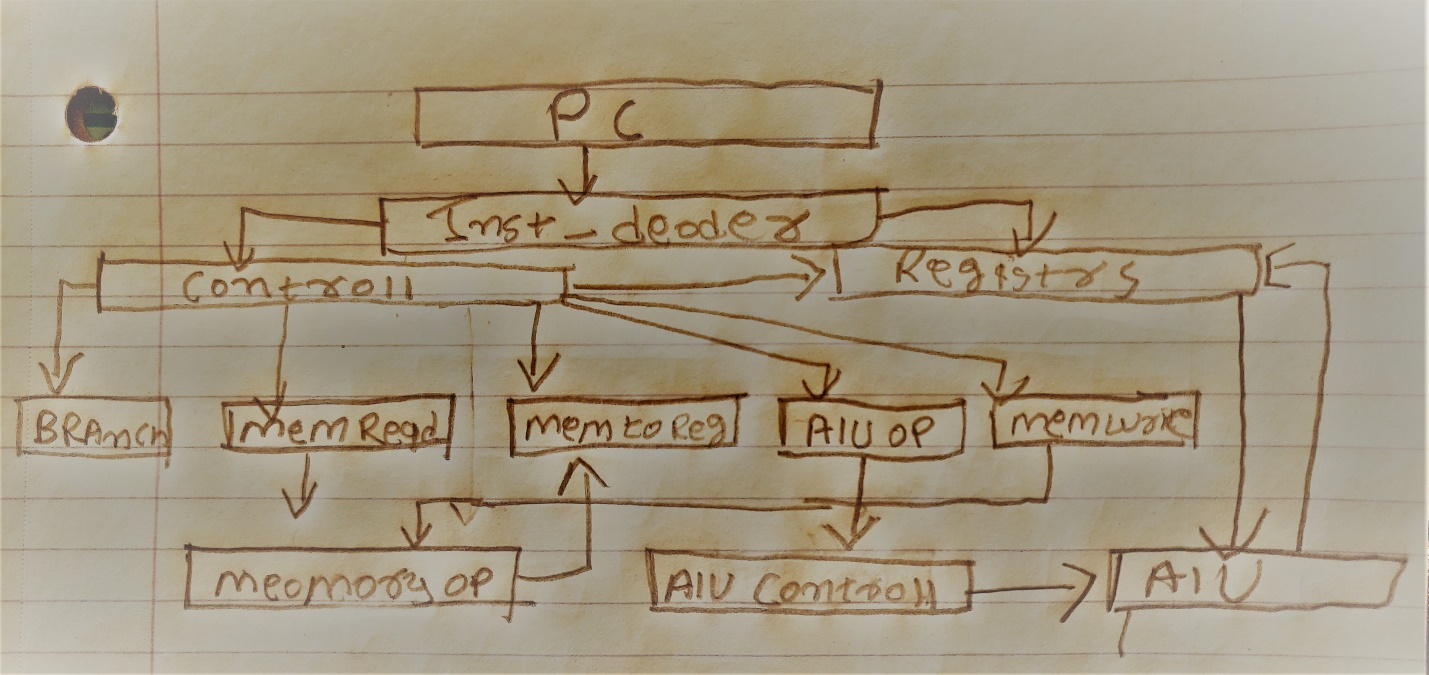
Part A:

 The Quartus project is a 32 bit processor. That computes all the R-Type, J-Type, and I-Type instructions which include load, Store, Add, Sub , Logical shifts, Logical operations such as And , Or and it also does the control flow instructions such as branch , and jump. The input of the processor is the instruction itself and the out of the processor is the performed instruction. For example, Assume the value of x3 = 0, x4, =1, x5=2 and you want to perform 2+1 so the you would provided the processor with the instruction Add x3, x5, x4 this would be your main input of the processor. Now the output of this would be to perform 1+2 and store into x3. This is project is divided into 7 sub modules. The modules are ALU unit, ALU control unit, control unit, register files unit, program counter unit, instruction fetch, and instruction decoder.

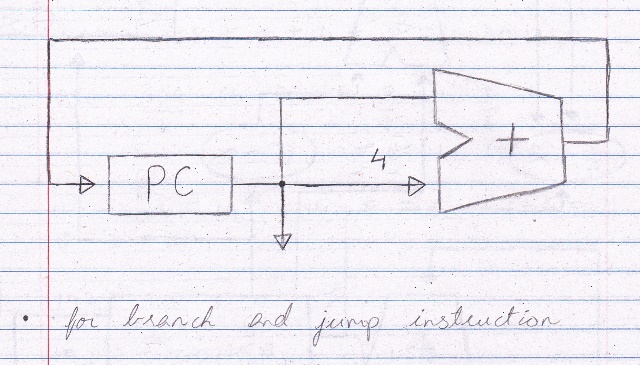
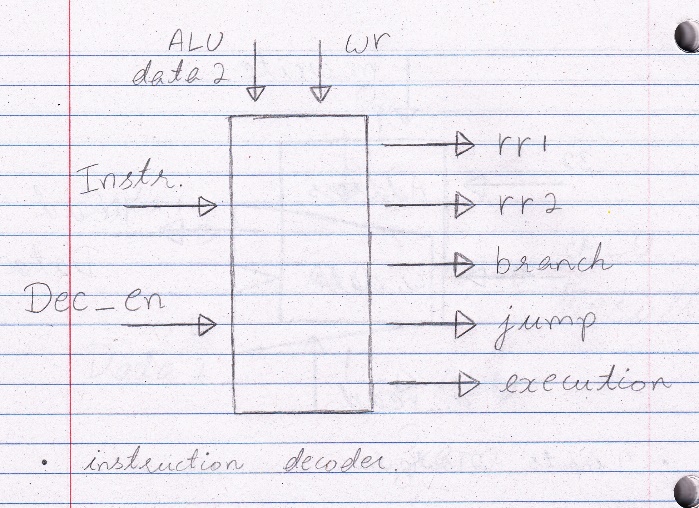
Part B:

1. The First module is PC. V which is a program counter. Which is The CPU's program counter that is set to the memory location where the first instruction in the program has been stored, and execution begins. It takes the current program counter and the instruction and if the type of instruction is j-type it takes the current offset and 32-bit type of instruction that can be either jump or a branch. The outputs of the PC are the updated address.
2. Second module is inst\_rom.v which fetches the instruction from the memory address provided by pc.v. The Input is the memory address from the program counter and outputs the instruction.
3. Next module is the inst\_decoder.v that takes the instruction outputted by inst\_rom. This module decodes the instruction and outputs the 32-bit machine code depending on the Type of the instruction.
4. fourth module is control unit which is the main/brain of the whole circuit it takes the bits [31-26] which is the 7-bit op code provided by the instruction decoder. The CU is made up of flip-flops, logic gates, digital circuits and encoder and decoder circuits that are wired in a specific and fixed way. Depending on the op code the Type is decided and specific outputs are triggered.
5. The fifth module is the register’s file unit that gets its input from the instruction decoder that provides the necessary 5 bit registers machine code. And it outputs the read registers.
6. The sixth module is the ALU process unit that reads the inputs given in from the registers file and the it performs the logical operations/ shifts/ and arithmetic operations. It outputs the result of the performed operation.
7. The Alu Control unit that receives and input of the opcode from the control unit and and passes the 3-bit function code to the ALU process unit.

***Part C:***

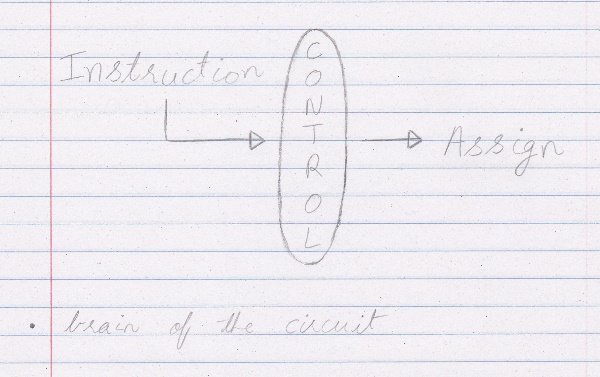
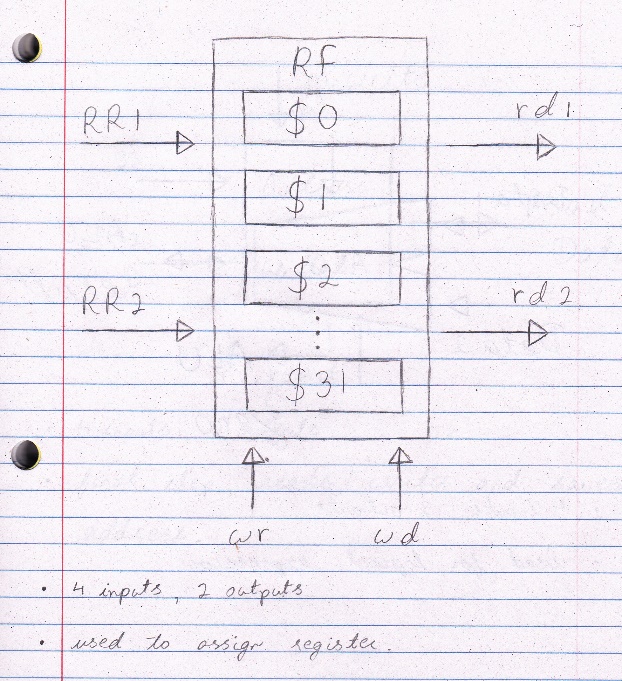
*PC and Instruction Decoder*

A PC (program counter) initially begins from 0000, it points at a certain location in the program memory which is then fetched by the CPU and placed in the instruction register. And at the same time, the PC is incremented by four. Then the process goes into the decoding cycle, which decodes the instruction and determines whether the instruction is executable or not. If the instruction is not executable, then the CPU goes back to the phase of fetching the instruction. If any instruction requires a jump instruction, then the CPU changes the content of the PC and jumps to the new location of the program memory and continues the process of fetching. This PC process continues until all the instructions in the program memory are fetched and decoded and executed. This process is also called the Fetch cycle*.*

**

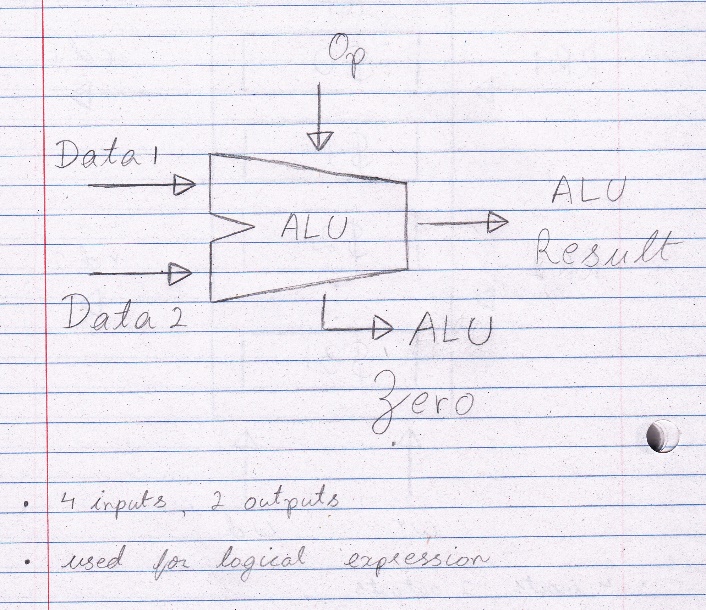
*Control Unit and Register Unit*

The CPU contains the control unit, where the control unit controls what the processor can access. The control unit uses circuitry to help the processor communicate with the software and the hardware available in the computer. It reads and interprets the instructions and determines the sequence for processing the data. The control unit simply generates control signals for different purposes. The register unit needs to take in the 5 bits inputs which are read-only, 5 bits register to write to, and 32 bits input source known as write data which holds the output and stores the data into the write to register. The register unit also outputs certain items, it outputs the 5 bits information that is stored in the read-only registers. Lastly the register unit also holds a 1 bit register called the RegWrite bit which determines whether or not it is okay to store the write data to the write register; if the RegWrite is 1, then it can be stored in the write register but if it is 0, then it cannot be stored in the write register. Register unit is a hardware component within a data path because while given some type of input, the data path lets the input flow through the hardware components and follows out computations.

**

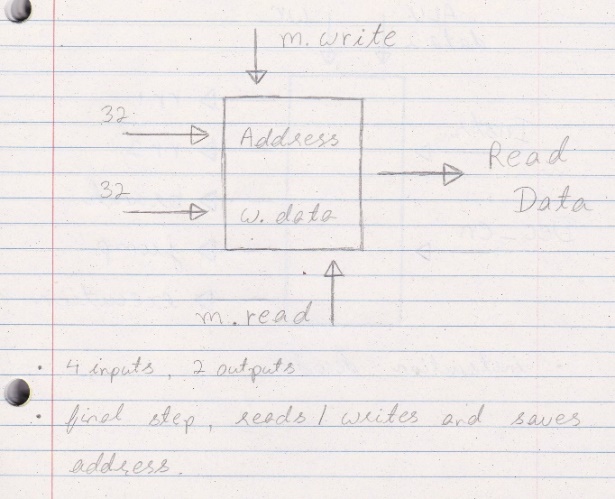
*ALU and ALU Control*

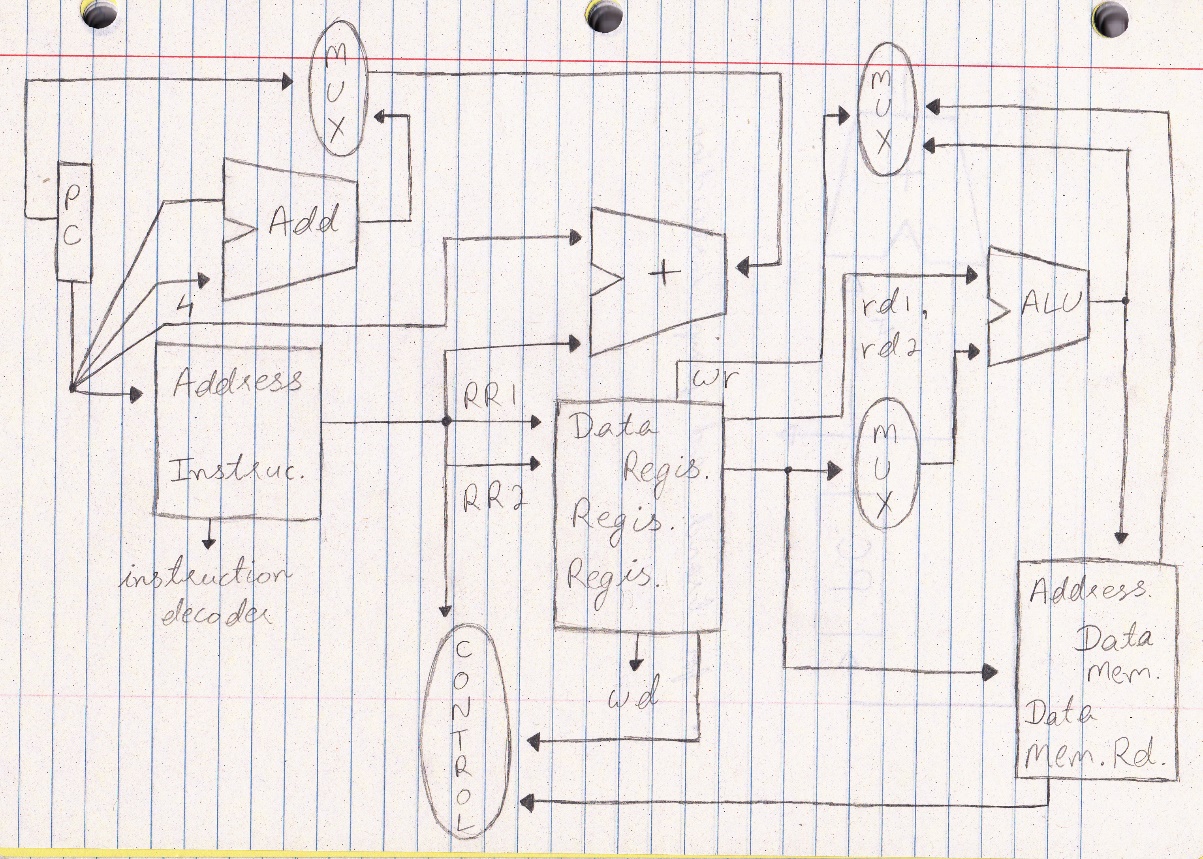
The op code in the R-type instruction, 7 bits [31:26] (always 0000000), goes to the control unit, which determines whether it needs to write to a register unit or not. Any function, 5 bits [4:0] (funct), of the R-type instruction goes through the ALU control (or ALU op), which tells the ALU (arithmetic logic unit) what type of R-type instruction it is. The ALU would then output the instruction that has been stated by the ALU control (add, sub, mul, etc.), that has been fixed earlier by the control unit which proves that it is a R-type instruction (which are the 7 bits [31:26]). The ALU which output the instruction will eventually go back to the register unit where it is supposed to write data. In the register unit, two registers are read from and one register that is written to, which are then read out within the register unit, then the ALU determines what he instruction is supposed to do, which is told by the ALU control (last 5 bits [5:0] (funct)); which at last goes back to write data and is then written into the write register in the register unit.

**

*Data Memory:*

Memory unit takes in, where in the memory (address) we want to read the address from, as the input which is 32 bits and the only output is simply the value from that address (read data) which is also 32 bits. The memory does not always just read information from the specific address, it sometimes writes data to a specific address. When the memory writes data, that is known as another input (write data), which is again 32 bits. There are two control lines used, one is MemWrite, which states that there will be some data that is going to be written to the memory address. The other one is MemRead which states that there will be some data that is just going to be read from the memory address. In simple words, memory stores information and the CPU calculate the stored information. The two typical components of a CPU includes the ALU and the control unit.

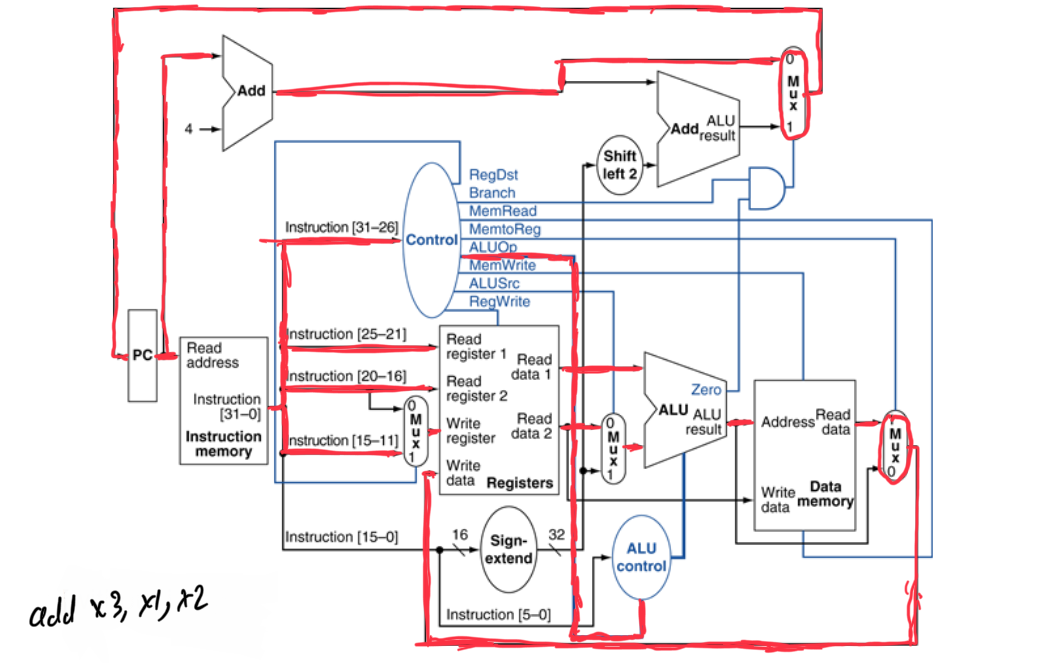
**

** When we combine all the parts explained above we get a 32 bit processor the rough diagram looks like the one drawn below.

**Part D:**

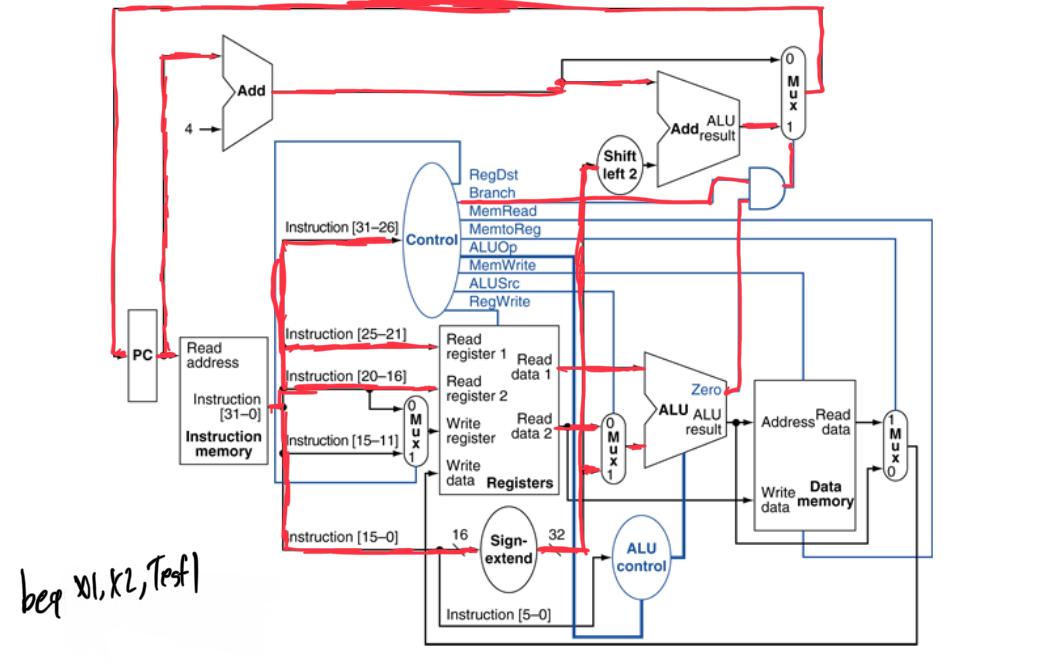
**ADD Instruction:** ADD x3, x1, x2 (This is a simple instruction to add two values)

The ADD instruction first goes through the PC (Program Counter), then through the Instruction memory which passes the 32-bit registers that are intimated in ADD instruction. After the instruction is passed through the Instruction memory it breaks down into three smaller registers bits to allocate structure for the Register and Control. The Control will read the instruction (in this case it’s ADD) after the control reads the instruction it will output the ALUOp code which will then continue to ALU control which will ADD the result and store the result into ALU as ALU result. When this previous process is completed the ALU result then moves to Address in Data memory and it reads it in MUX which then goes back to Registers and Writes the data back (Also known as Wd). While all this is happing, the PC goes to Add and add 4 to where this is a simple ADD instruction therefore the MUX will output zero and go back to the PC to fetch the next instruction. Science this is a simple instruction it usually take 1 clock cycle to complete which makes adding the fastest instruction to execute.



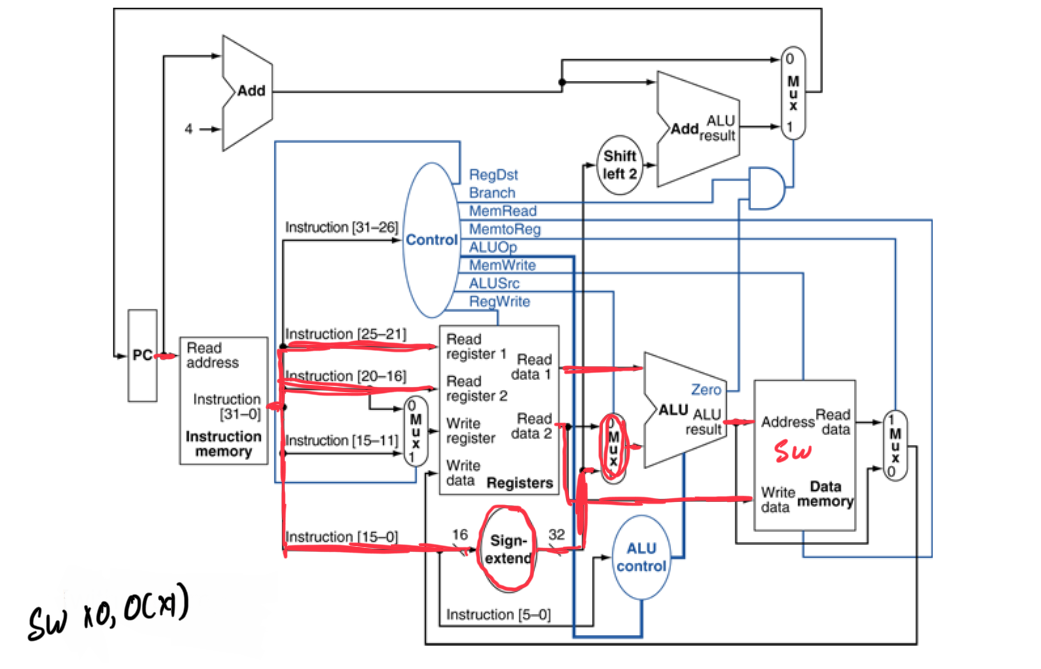
**BEQ Instruction:** BEQ x1, x2, Test1 (This instruction checks if x1 and x2 are equal)

The BEQ instruction first goes through the PC, and then it goes through the Instruction Memory which divided the code into OP, RS, RT or 16-bit immediate. The OP code will go to the Control, RS will go to the first read register and the RT will go to second read register as well as the MUX, the 16-bit immediate will also go through the Sign Extend to extend the sign from 16-bit to 32-bit which will then Shift Left two time to check if the adder is a branch OP code. The control will then tell the ALU to subtract the two x1 and x2 register and check if it is equal to zero if it is equal the zero the value will go through an and gate which is also connected to the control which will tell us if the And Gate is branch. The previous instruction will then go through MUX and if the different through the And Gate is not zero then the instruction will come back to PC to fetch a different set of value. Completing this instruction will take 3 cycles on average.

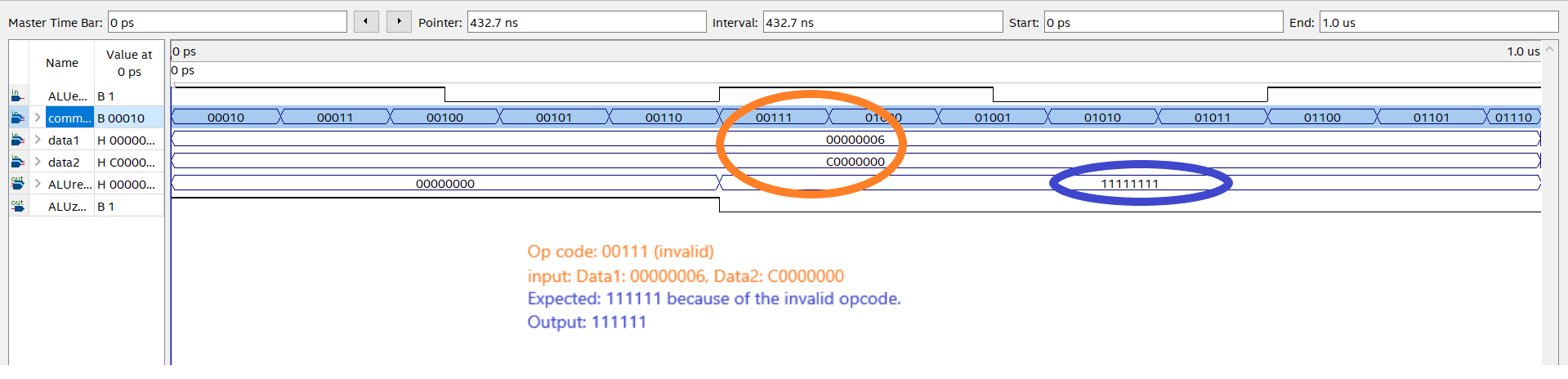
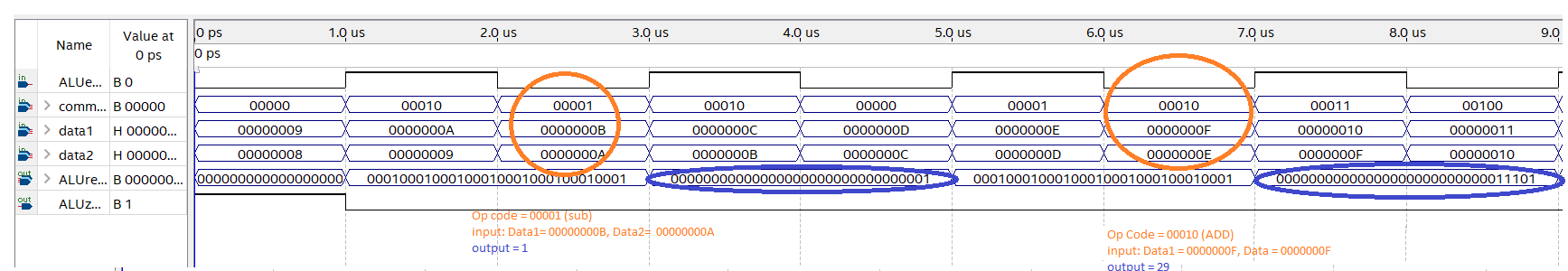


**SW Instruction**: SW x0, 0(x1) (This instruction is used for Storing a value in the CPU)

The SW instruction first goes through the PC, then the Instruction Memory which will brake down into RS, RT and offset/immediate value. The RS will go through the Read Register one, Rt will go through Read Register two, and offset will be enhanced through Sign Extend. The previous step will then go through the ALU to see if any operation needs to be done and if not, it will continue to Data Memory where the value of SW will be stored. Completing SW will take up toward off 4 cycles on average.



***Part E:***

******

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input in Hexadecimal | Input in binary | Input in Decimal | Expected Output | | | |
| 0000000B | 00000000 00000000 00000000 00001011 | 11 | 11-10 = 1 | |
| 0000000A | 00000000 00000000 00000000 00001010 | 10 |
| 0000000F | 00000000 00000000 00000000 00001111 | 15 | 15 + 14 = 29 | | |
| 0000000E | 00000000 00000000 00000000 00001110 | 14 |
| 00000006 | 00000000 00000000 00000000 00000110 | 6 | 11111111 |
| C0000000 | 1100000000000000000000000000000000 | 3221225472 |

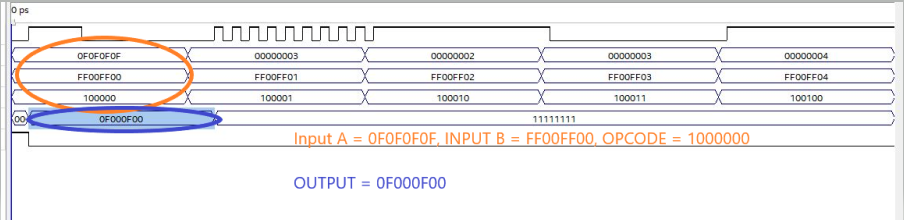
**Part-F:**

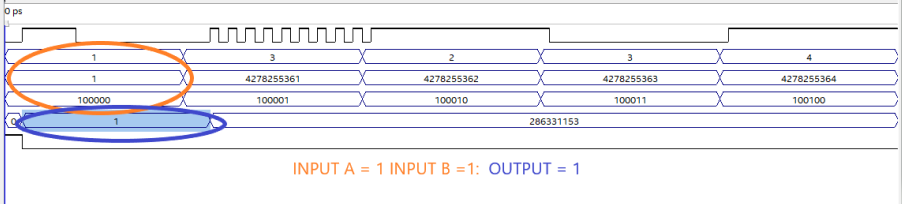
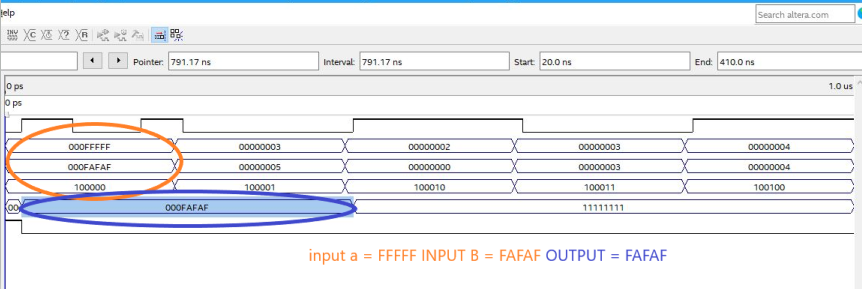
We as group decided to implement the AND instruction. In order to implement the AND instructions we had to change ALU and the control unit. We only need to change these two since instruction decoder already decodes that it is a R-type instruction. Now we needed to change the control unit so that it sets appropriate signals to 1 that includes the ALU control unit it provides the function to identify that the instruction and sends 0 to memory operations. Second we need to modify the ALU unit as the name suggests it’s an ARTHMETIC LOGIC UNIT. It needs the opcode as input we need to provide the and code opcode and as well as perform the and between two values.

|  |  |
| --- | --- |
| **BEFORE** | **AFTER** |
| **5 parameters for ALU operations in control.v** | **6 parameters for ALU operation. The sixth one is shown here: parameter ALUAND=6'b1000000;** |
| **No command input for AND in alu.v and control.v** | **Changed 5 bits to 6 bits of command input shown here: input[5:0] command to make room for and instruction.** |
| **No parameter for and in control.v** | **New parameter added in control.v parameter AND=12'b100000000000; Also as a result, we had to make all the other instructions’ parameters become 12 bits in size.** |
| **No parameter for AND in alu.v** | **6 parameters for And instruction. The sixth one is shown here: parameter ALUAND=6'b1000000; Also as a result, we had to make all the other instructions’ parameter becomes 6 bits in size.** |
| **AND operation not performed in alu.v** | **Added the instruction in ALU. ALUresult\_r<=data1&data2; which performs the AND instruction.** |
| **No control signals for AND operation control.v** | **Added the control signals in control.v As shown: ALUsrc<=1'b0; ALUcommand\_r<=ALUAND; state<= executing;** |

**Part G:**

**Input A is initialized in x3 , Input B is initialized in x4 and output is stored in x5.**

****

****

|  |  |  |
| --- | --- | --- |
| **I NPUTS** | **BINARY** | **EXPECTED OUTPUT** |
| **0F0F0F0F** | **00001111000011110000111100001111** | **11110000000000001111 00000000**  **HEX: (f000f00)** |
| **FF00FF00** | **11111111000000001111111100000000** |  |
| **1** | **00000000000000000000000000000001** | **000000000000000000 00000000000001**  **UNSGINED INTEGER: 1** |
| **1** | **00000000000000000000000000000001** |  |
| **FFFFF** | **00000000000011111111111111111111** | **00000000000011111010111110101111**  **HEX: FAFAF** |
| **FAFAF** | **00000000000011111010111110101111** |  |

**Part H:**

In milestone one our group has expanded our knowledge on how a CPU functions and the different parts it takes to make a low-level CPU. We also learned how a processor is designed and how different type of instructions move around the processor to output the inputted instruction. In this milestone we all learned how to use Quartus and the benefits on using a complex software like Quartus for better understanding the design of a CPU.

In milestone two my group has learned how to implement an extra operation such as AND, my group also learned how the extra operation moves through the CPU and gives us the correct output to our input.

**Part I:**

|  |  |
| --- | --- |
| Members | Tasks |
| Pranshu Vyas | * Worked on Part A , B E , F, G off the report |
| Kunj Patel | * Worked on Part D E, F , G off the report |
| Janki Jadeja | * Worked on Part C, E, F, G off the report |