**Nursena Köprücü- 59843**

**Hasan Can Aslan- 60453**

**COMP303 TERM PROJECT FALL 2019**

**Single Cycle Processor Design**

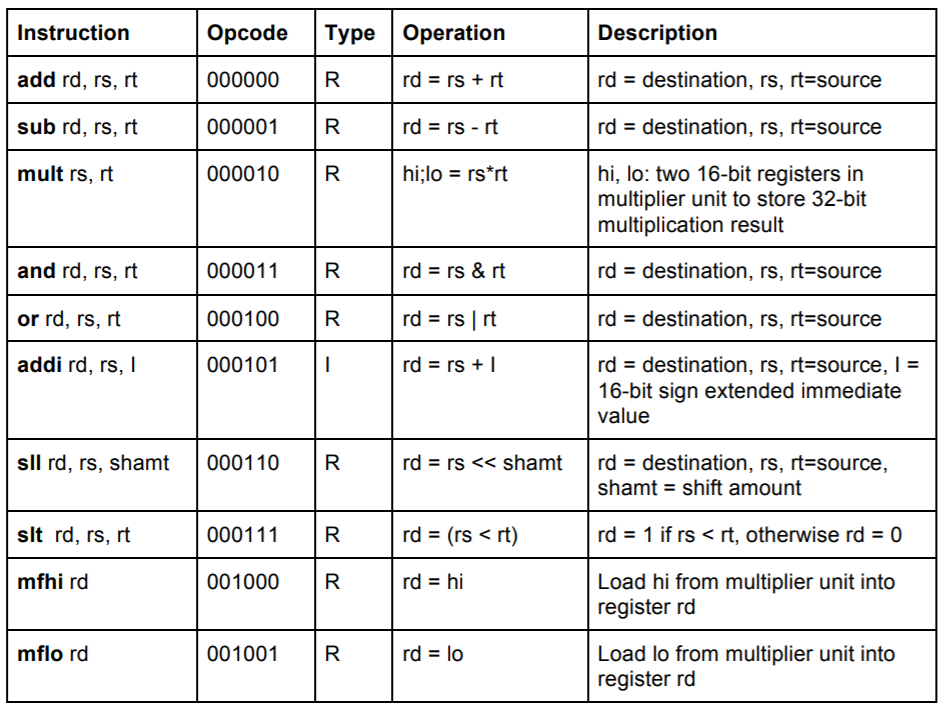
In this project, we have first designed 16-bit single cycle processor, implemented it by using Logisim, and tested the design whether it gives us the result as we expected.

1. **The ALU Design**

We have first designed and implemented our own Arithmetic Logic Unit (ALU) which supports 16-bit. However, we have 32-bit instructions as MIPS. So, the instruction formats should be as the following order:

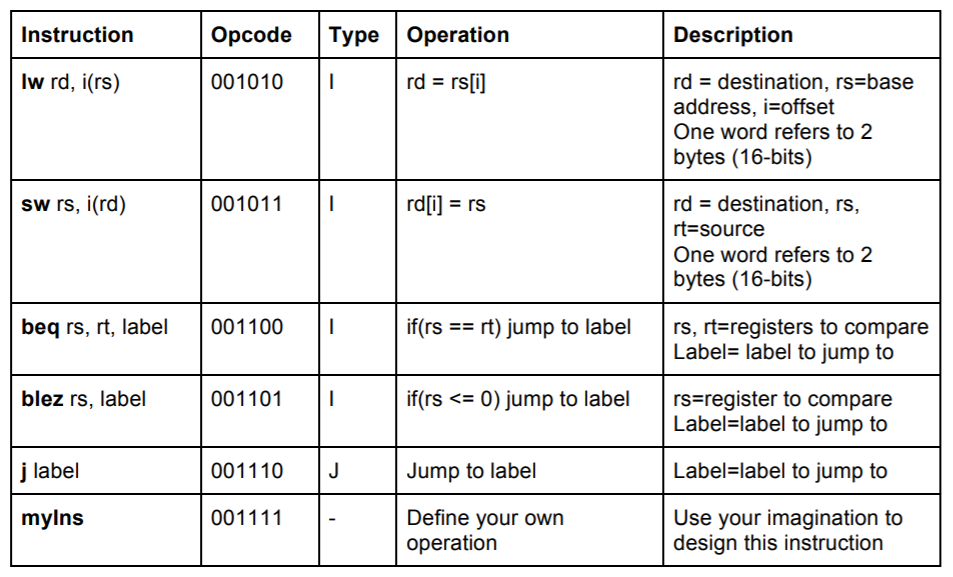
* R-type instruction: 6-bit for opcode, 5-bit for rs, 5-bit for rt, 5-bit for rd, 5-bit for shamt, 6-bit for function
* I-type instruction: 6-bit for opcode, 5-bit for rs, 5-bit for rt, 16-bit for immediate
* J-type instruction: 6-bit for opcode, 26-bit for address

The ALU supports the instructions as listed in the table below:

****

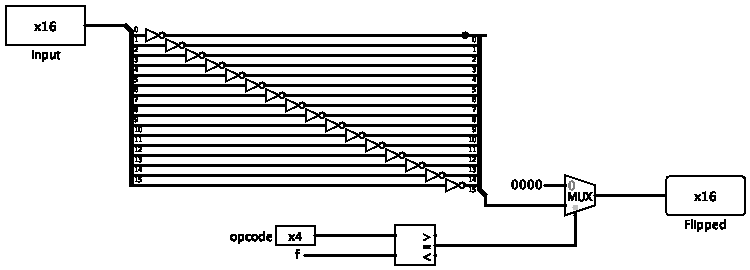
***Figure 1: ALU instructions***

In addition to these, we have added the following load-store, branch and jump instructions as listed in the table below.

******

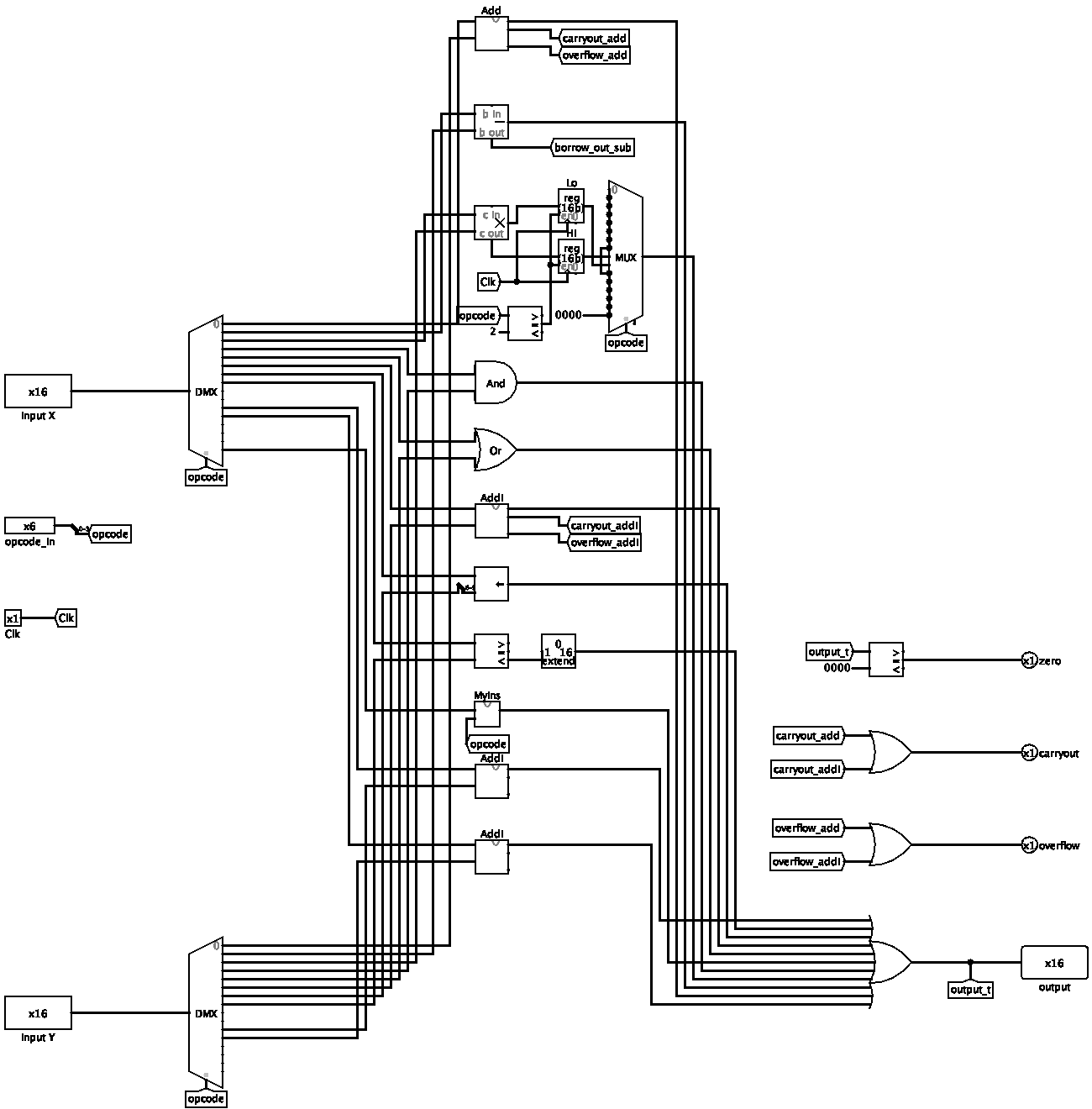
***Figure 2: Load-store, branch and jump instructions***

For myIns that is shown in the table, we have implemented the instruction that computes the 1s complement of a given input. It is an I-type instruction and simply inverts all bits with its complements as bitwise. Here is the design for this instruction:



***Figure 3: myIns Design***

In addition, we have specified the carryout, zero, and overflow bits to be able to display them when it is necessary. At the end, we have implemented all these instructions, and the final ALU design is as the following:

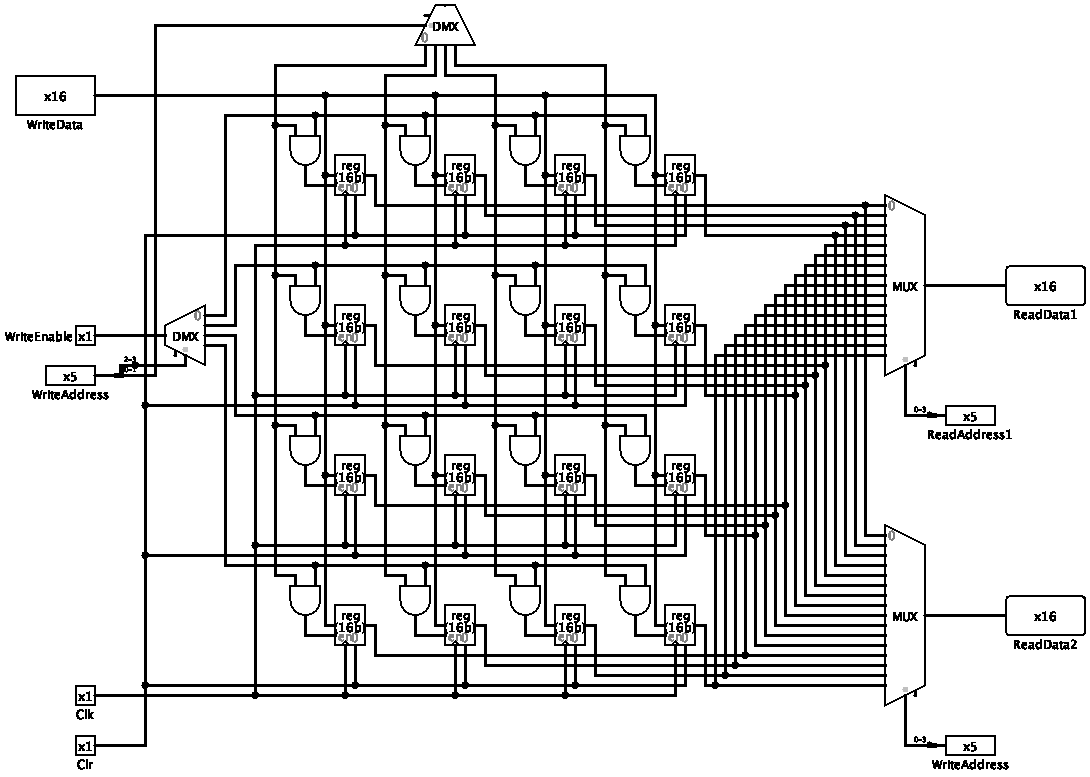


***Figure 4: The ALU Design***

1. **Register File**

We have designed our Register File by using 16 16-bit registers. It has two read register address ports, one write register address port, one write register data port, two read register output ports, a register write control signal, and a clock input.

At the end, it has the design as shown in the next page.

 ***Figure 5: Register File Design***

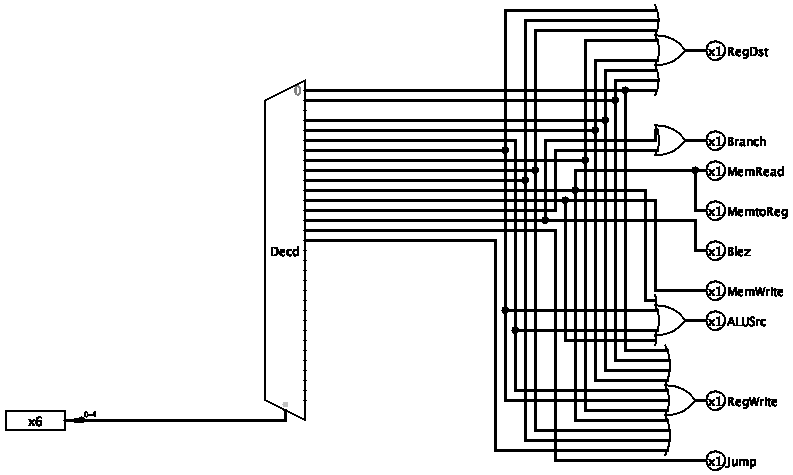
1. **Control Unit**

In the Control Unit, we have taken the opcode of a given instruction as an input; however, the higher two bits are redundant, since lower four bits are enough to decide what the operation is. We have nine control signals such that RegDst, Branch, MemRead, MemtoReg, Blez, MemWrite, ALUSrc, RegWrite, and Jump. In table below, we specify which control signal would be active for a given instruction type, and which one would not be active as 1 or 0, respectively. For example, when we have an addition operation, add, the Control Unit gives RegDst and RegWrite signals to do the instruction properly.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ins. Type | RegDst | Branch | Mem Read | MemtoReg | Blez | Mem Write | ALUSrc | Reg Write | Jump |
| add | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| sub | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| mult | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| and | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| or | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| addi | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| sll | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| slt | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| mfhi | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| mflo | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| lw | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| sw | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| beq | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| blez | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| j | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| myIns | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

***Figure 6: Table of Control Signals***

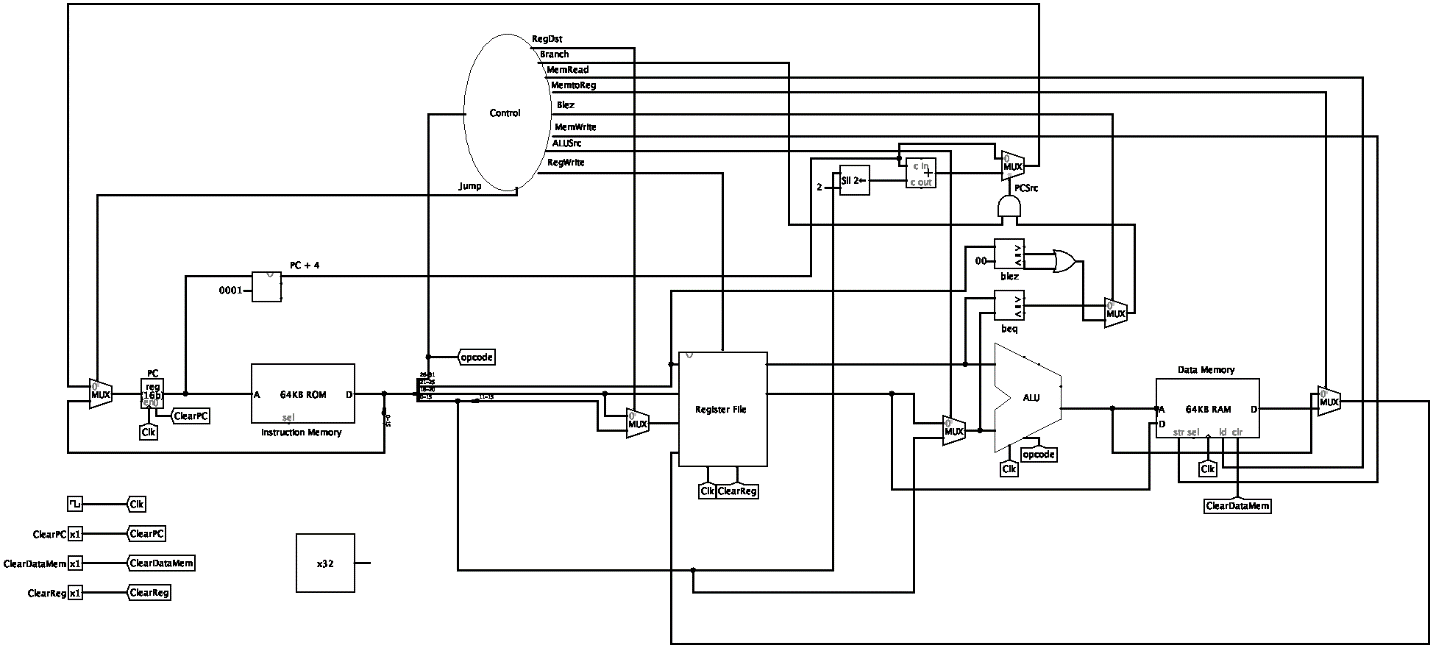
At the end, the Control Unit design is as the following:



***Figure 7: Control Unit Design***

1. **CPU Design**

In this part, we put the components that we have designed before like ALU and Register File all together to be able to implement the CPU design. For the memory part in the design, we have chosen to use a ROM for the instruction memory, but we have used a RAM for the data memory. Moreover, we give all the control signals into the correct destinations, since they are the key point for the process. The final CPU design is on the next page.



***Figure 8: CPU Design***

1. **Testing the Processor**

In this part, we have written an assembly code in a file named *test\_program\_asm.txt*. It contains the assembly instructions and its machine codes for each instruction, respectively. So, we can see which assembly instruction corresponds to which machine code. Here is the implementation:

**# MAIN**

**# addi $0, $0, 0**

**0: 20050001;**

**# addi $2, $2, 0**

**4: 20050001;**

**# addi $5, $5, 0**

**8: 20050001;**

**# addi $6, $6, 4**

**c: 20050001;**

**# addi $7, $7, 8**

**10: 20050001;**

**# beq $4, $5,44[exit-0x0000002c-4]**

**14: 20050001;**

**# lw $3, 0($5)**

**18: 20050001;**

**# lw $4, 0($6)**

**1c: 20050001;**

**# mult $3, $4**

**20: 20050001;**

**# mflo $1**

**24: 20050001;**

**# sll $8, $1, 1**

**28: 20050001;**

**# sw $8, 0($7)**

**2c: 20050001;**

**# addi $5, $5, 1**

**30: 20050001;**

**# addi $6, $6, 1**

**34: 20050001;**

**# addi $7, $7, 1**

**38: 20050001;**

**# addi $0, $0, 1**

**3c: 20050001;**

**# j 14 [loop-0x0000000e-4]**

**40: 20050001;**

**# DATA**

**0 : 00000005;**

**4 : 00000007;**

**8 : 0000fffe;**

**c : 00000028;**

**10: 00000041;**

**14: 0000ffe9;**

**18: 00000011;**

**1c: 00000400;**

**20: 00000000;**

**24: 00000000;**

**28: 00000000;**

**2c: 00000000;**

We have also written the corresponding machine instructions for the assembly program in the file named *test\_program\_code.txt.* It is used to load instructions into the instruction memory of the processor to test it.

We have also provided a separate file named *test\_program.txt* that contains the data of our program.

1. **Conclusion**

Finally, we have designed 16-bit single cycle processor and implemented it by using Logisim. We have also written the test files and checked whether it gives the expected result or not. We have first designed the ALU, and Register File, and use them to design a functional CPU. At the end, test trace shows that our design works properly.