

# CSE 331/503 Computer Organization Final Project – MiniMIPS Design

Hasan Mutlu – 1801042673

## Design Explanation:

- ➔ Top level entity MiniMIPS takes 16bit instruction as input. Its output is 32bit number in Program Counter. In testbench file, an array of instructions is created and read from file. Program Counter number is used as indexing for instructions as follows: instruction[PCounter].
- ➔ Unlike real MIPS, the PC is incremented by 1 instead of 4.
- ➔ Unlike real MIPS, branch instructions' immediate field isn't shifted left by 2. So, exact number of instructions to be jumped must be provided instead of 4\*( number of instructions to be jumped).
- ➔ ALU is changed from HW3. Instructions orders are changed and a comparator is included. Multiplier is excluded.

## Test Results

-> In R-Type instructions, two consecutive registers are operated and result is saved in second operand register. So there are 4 test results for each instruction type.

### 1.) 000 000 AND

```
registers.txt registers_output.txt instruction.txt
1 00000000000000000000000000000000 // 000 1 // memory data file (do not edit the following line)
2 11001010011101011100010101000111 // 001 2 // instance=/MiniMIPS_testbench/minimips/register
3 11011010111010011100010100011101 // 010 3 // format-bin addressradix=h dataradix=b version=1
4 101000111010111010111010111000 // 011 4 00000000000000000000000000000000
5 1101010100011101010101010101001 // 100 5 00000000000000000000000000000000
6 010110101010111010101110001110 // 101 6 11011010111010011100010100011101
7 1101010000111110101110001110010 // 110 7 1000001010101000100000000011000
8 11111111111111111111111111111111 // 111 8 1101010100011101010101010101001
9 01010000000001100000010100001000 9 01010000000001100000010100001000
10 1101010001111110101110001110010 10 1101010001111110101110001110010
11 1101010001111110101110001110010 11 1101010001111110101110001110010
12 12
```

### 2.) 0000 001 ADD

```
registers.txt registers_output.txt instruction.txt
1 00000000000000000000000000000000 // 000 1 // memory data file (do not edit the following line)
2 00000000100110001001010101111111 // 9,999,999 2 // instance=/MiniMIPS_testbench/minimips/register
3 00000000101010011000101011000111 // 5,555,555 3 // format-bin addressradix=h dataradix=b v
4 00000000100001111010001000111000 // 4,444,444 4 00000000000000000000000000000000
5 11111111111111111101010111010000 // -5400 5 00000000101010001011001111111111
6 00000000000000000000000000000000 // 5400 6 0000000010101001100010101100011
7 00111010101110011000101000110001 // 1,073,741,824 7 00000000100110001001011001111111
8 01000000000000000000000000000000 // 987,654,320 8 11111111111111111101011101000
9 00000000000000000000000000000000 9 00000000000000000000000000000000
10 00110101011100110010100010110001 10 00110101011100110010100010110001
11 01110101011100110010100010110001 11 01110101011100110010100010110001
12 12
```

### 3.) 0000 010 SUB

```
registers.txt registers_output.txt instruction.txt
1 00000000000000000000000000000000 // 000 1 // memory data file (do not edit the following line)
2 00000000100110001001010101111111 // 9,999,999 2 // instance=/MiniMIPS_testbench/minimips/register
3 00000000101010011000101011000111 // 5,555,555 3 // format-bin addressradix=h dataradix=b v
4 00000000100001111010001000111000 // 4,444,444 4 00000000000000000000000000000000
5 11111111111111111101010111010000 // -5400 5 111111111010011011010010000001 // -9,999,999
6 00000000000000000000000000000000 // 5400 6 0000000010101001100010101100011
7 00111010101110011000101000110001 // 1,073,741,824 7 000000000010000111010001000111
8 01000000000000000000000000000000 // 987,654,320 8 11111111111111111101011101000
9 00000000000000000000000000000000 9 111111111111111111010111010000 // -10,800
10 00110101011100110010100010110001 10 00110101011100110010100010110001
11 01110101011100110010100010110001 11 000001010010001100101101001111 // 86,087,504
12 12
```

#### 4.) 0000 011 XOR

```
registers.txt
1 00000000000000000000000000000000 // 000
2 11001010011101011100010101000111 // 001
3 11011010111010011100010100011101 // 010
4 101000111010111010111010111000 // 011
5 110101010001110101010101101001 // 100
6 0101101010101110101011010001110 // 101
7 11010100001111110101110001110010 // 110
8 11111111111111111111111111111111 // 111

registers_output.txt
1 // memory data file (do not edit the following l
2 // instance=/MiniMIPS_testbench/minimips/regist/
3 // format=bin addressradix=h dataradix=b version
4 00000000000000000000000000000000
5 11001010011101011100010101000111
6 11011010111010011100010100011101
7 0111001010001110111111110100101
8 110101010001110101010101010001
9 10001111111010011111100011100111
10 1101010000111110101110001110010
11 0010101111000001010001110001101
12

instruction.txt
1 // This program is XOR test
2 0000_000_001_001_011 // xor $s1, $s0, $s1
3 0000_010_011_011_011 // xor $s3, $s2, $s3
4 0000_100_101_101_011 // xor $s5, $s4, $s5
5 0000_111_110_111_011 // xor $s7, $s6, $s6
6 0000_000_000_000_000 // halt
```

#### 5.) 0000 100 NOR

```
registers.txt
1 00000000000000000000000000000000 // 000
2 11001010011101011100010101000111 // 001
3 11011010111010011100010100011101 // 010
4 101000111010111010111010111000 // 011
5 110101010001110101010101101001 // 100
6 0101101010101110101011010001110 // 101
7 11010100001111110101110001110010 // 110
8 11111111111111111111111111111111 // 111

registers_output.txt
1 // memory data file (do not edit the following l
2 // instance=/MiniMIPS_testbench/minimips/regist/
3 // format=bin addressradix=h dataradix=b version
4 00000000000000000000000000000000
5 00110101100010100011101010111000
6 11011010111010011100010100011101
7 00000100000100000000000000000010
8 110101010001110101010101010001
9 00100000001000000000000000000000
10 11010100001111110101110001110010
11 00000000000000000000000000000000
12

instruction.txt
1 // This program is NOR test
2 0000_000_001_001_100 // nor $s1, $s0, $s1
3 0000_010_011_011_100 // nor $s3, $s2, $s3
4 0000_100_101_101_100 // nor $s5, $s4, $s5
5 0000_111_110_111_100 // nor $s7, $s6, $s6
6 0000_000_000_000_000 // halt
```

#### 6.) 0000 101 OR

```
registers.txt
1 00000000000000000000000000000000 // 000
2 11001010011101011100010101000111 // 001
3 11011010111010011100010100011101 // 010
4 101000111010111010111010111000 // 011
5 110101010001110101010101101001 // 100
6 0101101010101110101011010001110 // 101
7 11010100001111110101110001110010 // 110
8 11111111111111111111111111111111 // 111

registers_output.txt
1 // memory data file (do not edit the following l
2 // instance=/MiniMIPS_testbench/minimips/regist/
3 // format=bin addressradix=h dataradix=b version
4 00000000000000000000000000000000
5 11001010011101011100010101000111
6 11011010111010011100010100011101
7 111110111101111111111111011101
8 110101010001110101010101010001
9 110111111101111111111101110111
10 1101010000111110101110001110010
11 11111111111111111111111111111111
12

instruction.txt
1 // This program is OR test
2 0000_000_001_001_101 // or $s1, $s0, $s1
3 0000_010_011_011_101 // or $s3, $s2, $s3
4 0000_100_101_101_101 // or $s5, $s4, $s5
5 0000_111_110_111_101 // or $s7, $s6, $s6
6 0000_000_000_000_000 // halt
```

-> In I-Type operation instructions, registers with even addresses (0,2,4,6) are operated with immediate field and the result is saved in next register. There are 4 test results for each instruction type.

#### 8.) 0001 ADDI

```
registers.txt
1 00000000000000000000000000000000 // 000
2 11001010011101011100010101000111 // 001
3 00000000000000000000000000000000 // 2048
4 101000111010111010111010111000 // 100
5 00000000000000000000000000000000 // 4071
6 0101101010101110101011010001110 // 1048
7 00000000000000000000000000000000 // 1048
8 11111111111111111111111111111111 // 111

registers_output.txt
1 // memory data file (do not edit the following l
2 // instance=/MiniMIPS_testbench/minimips/regist/
3 // format=bin addressradix=h dataradix=b version
4 00000000000000000000000000000000
5 000000000000000000000000000001101 // 13
6 000000000000000000000000000000000
7 000000000000000000000000000001101 // 2077
8 00000000000000000000000000000110111
9 000000000000000000000000000000000 // 4096
10 0000000000000000000000000000011000
11 000000000000000000000000000000000 // 1024
12

instruction.txt
1 // This program is ADDI test
2 0001_000_001_001101 // addi $s1, $s0, 13
3 0001_010_011_011101 // addi $s3, $s2, 29
4 0001_100_101_011001 // addi $s5, $s4, 25
5 0001_110_111_101000 // addi $s7, $s6, -24
6 0000_000_000_000_000 // halts
```

## 9.) 0010 ANDI

≡ registers.txt ×

C: > Users > Lenovo > Desktop > OKUL > 3 Fall > 331 Computer Organization > HW4 > testsources > andi test ≡ registers\_output.txt

Users > Lenovo > Desktop > OKUL > 3 Fall > 331 Computer Organization > HW4 > testsources > andi test ≡ registers\_output.txt

1 000000000000000000000000000000 // 000

2 11001010011101011100010101000111 // 001

3 11011010111010011100010100011101 // 010

4 10100011101011101011101010111000 // 011

5 1101010101000111010101011101001 // 100

6 01011010101011101010111010001110 // 101

7 11010100001111110101110001110010 // 110

8 11111111111111111111111111111111 // 111

≡ registers\_output.txt ×

C: > Users > Lenovo > Desktop > OKUL > 3 Fall > 331 Computer Organization > HW4 > testsources > andi test ≡ registers\_output.txt

1 // memory data file (do not edit the following line - required for mem load use)

2 // instance=/MiniMIPS\_testbench/minimips/register/registers

3 // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress

4 00000000000000000000000000000000

5 00000000000000000000000000000000

6 11011010111010011100010100011101

7 11011010111010011100010100011100

8 11010101010001110101010101010001

9 00000000000000000000000000001001

10 11010100001111110101110001110010

11 000000000000000000000000010000

12

≡ instruction.txt ×

C: > Users > Lenovo > Desktop > OKUL > 3 Fall > 331 Computer Organization > HW4 > testsources > andi test ≡ instruction.txt

1 // This program is ANDI test

2 0010\_000\_001\_001101 // andi \$s1, \$s0, 32'b00000000000000000000000000001101

3 0010\_010\_011\_101100 // andi \$s3, \$s2, 32'b111111111111111111111111111101100

4 0010\_100\_101\_011001 // andi \$s5, \$s4, 32'b000000000000000000000000000011001

5 0010\_110\_111\_010101 // andi \$s7, \$s6, 32'b000000000000000000000000000010101

6 0000\_000\_000\_000\_000 // halt

## 10.) 0011 ORI

registers.txt

C:\Users\Lenovo\Desktop> OKUL > 3 Fall > 331 Computer Organization > HW4 > testsources > ori test > registers\_output.txt

```
1 00000000000000000000000000000000 // 000
2 11001010011101011100010101000111 // 001
3 11011010111010011100010100011101 // 010
4 10100011101011101011101010111000 // 011
5 11010101010001110101010101101001 // 100
6 01011010101011110101011010001110 // 101
7 11010100001111110101110001100010 // 110
8 11111111111111111111111111111111 // 111
```

registers\_output.txt

C:\Users\Lenovo\Desktop> OKUL > 3 Fall > 331 Computer Organization > HW4 > testsources > ori test > registers\_output.txt

```
1 // memory data file (do not edit the following line - required for mem load use)
2 // instance=MiniMIPS_testbench/minimips/register/registers
3 // format-bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4 00000000000000000000000000000000
5 000000000000000000000000000000001101
6 11011010111010011100010100011101
7 1111111111111111111111111111111101
8 11010101000111010101010101010001
9 110101010001110101010101111001
10 1101010001111110101110001110010
11 1101010001111110101110001110111
12
```

instruction.txt

C:\Users\Lenovo\Desktop> OKUL > 3 Fall > 331 Computer Organization > HW4 > testsources > ori test > instruction.txt

```
1 // This program is ORITest
2 0011_000_001_001101 // ori $s1, $s0, 32'b00000000000000000000000000001101
3 0011_010_011_101100 // ori $s3, $s2, 32'b1111111111111111111111111101100
4 0011_100_101_011001 // ori $s5, $s4, 32'b000000000000000000000000000011001
5 0011_110_111_010101 // ori $s7, $s6, 32'b000000000000000000000000000010101
6 0000_000_000_000_000 // halt
```

## 11.) 0100 NORI

```
organization > HW4 > testsources > nori test > registers.txt  
1 00000000000000000000000000000000  
2 11001010011101011100010101000111  
3 11011010111010011100010100011101  
4 10100011101011101011101010111000  
5 11010101010001110101010101101001  
6 01011010101011101010111010001110  
7 11010100001111110101110001110010  
8 11111111111111111111111111111111
```

```
C:\> Users\Lenovo\Desktop> OKUL > 3 Fall > 331 Computer Organization > HW4 > testsources > nori test > registers_output.txt  
1 // memory data file (do not edit the following line - required for mem load use)  
2 // instance=/MiniMIPS_testbench/minimips/regist/registers  
3 // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress  
4 00000000000000000000000000000000  
5 111111111111111111111111111110010  
6 11011010111010011100010100011101  
7 00000000000000000000000000000010  
8 11010101010001110101010101101001  
9 00101010101110001010101010000110  
10 11010100001111110101110001110010  
11 00101011110000001010001110001000  
12
```

```
C:\> Users\Lenovo\Desktop> OKUL > 3 Fall > 331 Computer Organization > HW4 > testsources > nori test > instruction.txt  
1 // This program is NORI test  
2 0100_000_001_001101 // nori $s1, $s0, 32'b00000000000000000000000000001101  
3 0100_010_011_101100 // nori $s3, $s2, 32'b1111111111111111111111111101100  
4 0100_100_101_011001 // nori $s5, $s4, 32'b00000000000000000000000000011001  
5 0100_110_111_010101 // nori $s7, $s6, 32'b0000000000000000000000000010101  
6 0000_000_000_000_000 // halt
```

-> In branch tests, addi instructions are added to show branches work. Jumped instructions aren't evaluated.

## 12.) 0101 BEQ

```
registers.txt
1 00000000000000000000000000000000 //
2 11001010011101011100010101000111 //
3 11011010111010011100010100011101 //
4 11011010111010011100010100011101 // equal with above
5 1101010100011101010101010101001 //
6 1101010100011101010101010101001 // equal with above
7 11010100001111110101110001110010 //
8 11111111111111111111111111111111 //
```

```
registers_output.txt
1 // memory data file (do not edit the following line - required for mem load use)
2 // instance=/MiniMIPS_testbench/minimips/regist/registers
3 // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4 00000000000000000000000000000000
5 00000000000000000000000000001000
6 11011010111010011100010100011101
7 11011010111010011100010100011101
8 1101010100011101010101010101001
9 1101010100011101010101010101001
10 0000000000000000000000000000110
11 111111111111111111111111110000
12
```

```
instruction.txt
1 0101_010_011_000011 // beq $s2, $s3, 3 true
2 0001_000_001_000001 // addi $s3, $s0, 1 This will be jumped.
3 0001_000_010_000010 // addi $s2, $s0, 2 This will be jumped.
4 0001_000_001_000011 // addi $s1, $s0, 3 This will be jumped.
5 0001_000_001_001000 // addi $s1, $s0, 8 This won't be jumped.
6
7 0101_010_011_000010 // beq $s4, $s5, 2 true
8 0001_000_010_000100 // addi $s2, $s0, 4 This will be jumped.
9 0001_000_001_000101 // addi $s3, $s0, 5 This will be jumped.
10 0001_000_110_000110 // addi $s6, $s0, 6 This won't be jumped.
11
12 0101_000_111_000011 // beq $s0, $s7, 1 false
13 0001_000_111_100000 // addi $s7, $s0, -32 This won't be jumped.
14 0000_000_000_000000 //halt
```

## 13.) 0110 BNE

```
registers.txt
1 00000000000000000000000000000000 //
2 11001010011101011100010101000111 //
3 11011010111010011100010100011101 //
4 1010001110101110101110101011000 // not equal with above
5 1101010101000111010101010101001 //
6 01011010101011101010101010001110 // not equal with above
7 11010100001111110101110001110010 //
8 00000000000000000000000000000000 // equal with $s0
```

```
registers_output.txt
1 // memory data file (do not edit the following line - required for mem load use)
2 // instance=/MiniMIPS_testbench/minimips/regist/registers
3 // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4 00000000000000000000000000000000
5 00000000000000000000000000000000
6 11011010111010011100010100011101
7 1010001110101110101110101011000
8 1101010101000111010101010101001
9 01011010101011101010101010001110
10 0000000000000000000000000000110
11 111111111111111111111111110000
12 // same output with above but with BNE instruction
```

```
instruction.txt
1 0110_010_011_000011 // bne $s2, $s3, 3 true
2 0001_000_001_000001 // addi $s3, $s0, 1 This will be jumped.
3 0001_000_010_000010 // addi $s2, $s0, 2 This will be jumped.
4 0001_000_001_000011 // addi $s1, $s0, 3 This will be jumped.
5 0001_000_001_001000 // addi $s1, $s0, 8 This won't be jumped.
6
7 0110_010_011_000010 // bne $s4, $s5, 2 true
8 0001_000_010_000100 // addi $s2, $s0, 4 This will be jumped.
9 0001_000_001_000101 // addi $s3, $s0, 5 This will be jumped.
10 0001_000_110_000110 // addi $s6, $s0, 6 This won't be jumped.
11
12 0110_000_111_000011 // bne $s0, $s7, 1 false
13 0001_000_111_100000 // addi $s7, $s0, -32 This won't be jumped.
14 0000_000_000_000000 //halt
```



## 14.) 0111 SLTI

```
registers.txt X registers_output.txt X instruction.txt X
Fall > 331 Computer Organization > HW4 > testsources > slti test C: > Users > Lenovo > Desktop > OKUL > 3 Fall > 331 Computer Orgar C: > Users > Lenovo > Desktop > OKUL > 3 Fall > 331 Computer Organization > HW4 > testsources > slti test
1 00000000000000000000000000000000 // memory data file (do not edit the fo
2 11001010011101011100010101000111 // instance=/MiniMIPS_testbench/minimip
3 11011010111010011100010100011101 // format=bin addressradix=h dataradix=l
4 10100011101011101011101010111000 00000000000000000000000000000000
5 01010101010001110101010101010001 00000000000000000000000000000001
6 01011010101011101010101010001110 11011010111010011100010100011101
7 01010100001111110101110001110010 00000000000000000000000000000001
8 00000000000000000000000000000000 01010101010001110101010101010001
9 00000000000000000000000000000000 00000000000000000000000000000000
10 01010100001111110101110001110010 01010101000111110101110001110010
11 00000000000000000000000000000000 00000000000000000000000000000000
1 // This is SLTI test
2 0111_000_001_000110 // slti $s1, $s0, 6 (true)
3 0111_010_011_010000 // slti $s3, $s2, 16 (true)
4 0111_100_101_100100 // slti $s5, $s4, -28 (false)
5 0111_110_111_000000 // slti $s7, $s6, 0 (false)
6 0000_000_000_000000 // halt
```

-> In Load-Store tests, data in registers with odd addresses(1,3,5,7) are loaded or stored to memory.

## 15.) 1000 LW

```
registers.txt X registers_output.txt X memory.txt X
computer Organization > HW4 > testsources > lw test C: > Users > Lenovo > Desktop > OKUL > 3 Fall > 331 Computer C: > Users > Lenovo > Desktop > OKUL > 3 Fall > 331 Computer Organization > HW4 > testsources > lw test
1 00000000000000000000000000000000 // memory data file (do not edit the
2 11001010011101011100010101000111 // instance=/MiniMIPS_testbench/minimip
3 00000000000000000000000000000001 // format=bin addressradix=h dataradix=l
4 10100011101011101011101010111000 00000000000000000000000000000000
5 00000000000000000000000000000011 00000000000000000000000000000100
6 01011010101011101010101010001110 00000000000000000000000000000001
7 00000000000000000000000000000011 00000000000000000000000000000011
8 00000000000000000000000000000000 00000000000000000000000000000011
9 00000000000000000000000000000010 00000000000000000000000000000010
10 00000000000000000000000000000011 00000000000000000000000000000011
11 00000000000000000000000000000011 00000000000000000000000000000011
12 00000000000000000000000000000011 00000000000000000000000000000011
1 // This is LW test
2 1000_000_001_000001 // lw $s1, 1($s0) (001 in memory)
3 1000_010_011_000010 // lw $s3, 2($s2) (011 in memory)
4 1000_100_101_111111 // lw $s5, -1($s4) (010 in memory)
5 1000_110_111_000000 // lw $s7, 0($s6) (111 in memory)
6 0000_000_000_000000 // halt
```

## 16.) 1001 SW

```
registers.txt X memory.txt X memory_output.txt X
Fall > 331 Computer Organization > HW4 > testsources > sw test C: > Users > Lenovo > Desktop > OKUL > 3 Fall > 331 Computer Organiz C: > Users > Lenovo > Desktop > OKUL > 3 Fall > 331 Computer Organization > HW4 > testsources > sw test
1 00000000000000000000000000000000 // memory data file (do not edit the following line - require
2 11001010011101011100010101000111 // instance=/MiniMIPS_testbench/minimips/memory/memory
3 00000000000000000000000000000001 // format=bin addressradix=h dataradix=b version=1.0 wordsper
4 10100011101011101011101010111000 00000000000000000000000000000000
5 00000000000000000000000000000011 11001010011101011100010101000111
6 01011010101011101010101010001110 01011010101011101010101010001110
7 00000000000000000000000000000011 10100011101011101011101010111000
8 11111111111111111111111111111111 00000000000000000000000000000100
9 00000000000000000000000000000000 00000000000000000000000000000101
10 00000000000000000000000000000001 00000000000000000000000000000110
11 00000000000000000000000000000001 11111111111111111111111111111111
12 00000000000000000000000000000001 00000000000000000000000000000100
13 00000000000000000000000000000001 00000000000000000000000000000101
14 00000000000000000000000000000001 00000000000000000000000000000101
15 00000000000000000000000000000001 00000000000000000000000000000101
16 00000000000000000000000000000001 00000000000000000000000000000110
17 00000000000000000000000000000001 00000000000000000000000000000101
18 00000000000000000000000000000001 00000000000000000000000000000100
19 00000000000000000000000000000001 00000000000000000000000000000101
20 00000000000000000000000000000001 00000000000000000000000000000101
21 00000000000000000000000000000001 00000000000000000000000000000100
22 00000000000000000000000000000001 00000000000000000000000000000101
23 00000000000000000000000000000001 00000000000000000000000000000110
24 00000000000000000000000000000001 00000000000000000000000000000111
25 00000000000000000000000000000001 00000000000000000000000000000100
26 00000000000000000000000000000001 00000000000000000000000000000101
27 00000000000000000000000000000001 00000000000000000000000000000110
1 // This is SW test
2 1001_000_001_000001 // sw $s1, 1($s0) (001 in memory)
3 1001_010_011_000010 // sw $s3, 2($s2) (011 in memory)
4 1001_100_101_111111 // sw $s5, -1($s4) (010 in memory)
5 1001_110_111_000000 // sw $s7, 0($s6) (111 in memory)
6 0000_000_000_000000 // halt
```

## Bonus Test: Multiplier

-> I've written and run a simple multiplier program using ADDI and BNE instructions.

```

C:\Users\Lenovo\Desktop\OKUL\3 Fall\331 Computer Organization> registers_output.txt X
1 // memory data file (do not edit the follo
2 // instance=MiniMIPS_testbench/minimips/r
3 // format=bin addressradix=h dataradix=b v
4 00000000000000000000000000000000
5 00000000000000000000000000000101
6 0000000000000000000000000000010000 <-- Result
7 0000000000000000000000000000000101
8 0000000000000000000000000000000011
9 0101101010101110101010110001110
10 000000000000000000000000000000111
11 11111111111111111111111111111111
12

C:\Users\Lenovo\Desktop\OKUL\3 Fall\331 Computer Organization\HW4\testsources\multiplier> instruction.txt X
1 // This program is multiplier. It multiplies 5 with 8. Result is found in $s2
2 0001_000_001_000101 //addi $s1, $s0, 5 - multiplier is loaded in $s1
3 0001_000_010_000000 //addi $s2, $s0, 0 - $s2 is cleared. It will be the result register.
4 0001_000_011_000000 //addi $s3, $s0, 0 - $s3 is cleared. It will be the counter.
5 0001_010_010_001000 //addi $s2, $s2, 8 - $s2 is incremented by multiplicand
6 0001_011_011_000001 //addi $s3, $s3, 1 - $s3 is incremented by 1.
7 0110_011_001_111101 //bne $s3, $s1, -3 - If counter didn't reach multiplier's value, jump back 2 instructions.
8 0000_000_000_000000 //halt
  
```

## Controller:

Opcode:	_0000	Mixed	_0101	_0110	1000	1001
	R-Type	I-Type Arithmetic	BEQ	BNE	LW	SW
RegDst	1	0	x	x	0	x
ALUSrc	0	1	0	0	1	1
MemtoReg	0	0	0	0	1	0
RegWrite	1	1	0	0	1	0
MemRead	0	0	0	0	1	0
MemWrite	0	0	0	0	0	1
Branch	0	0	1	1	0	0
BranchSrc	x	x	0	1	x	x

## ALUControl:

Instr	Opcode				Func			ALUOp		
AND	0	0	0	0	0	0	0	0	0	0
ADD	0	0	0	0	0	0	1	0	0	1
SUB	0	0	0	0	0	1	0	0	1	0
XOR	0	0	0	0	0	1	1	0	1	1
NOR	0	0	0	0	1	0	0	1	0	0
OR	0	0	0	0	1	0	1	1	0	1
ADDI	0	0	0	1	X	X	X	0	0	1
ANDI	0	0	1	0	X	X	X	0	0	0
ORI	0	0	1	1	X	X	X	1	0	1
NORI	0	1	0	0	X	X	X	1	0	0
BEQ	0	1	0	1	X	X	X	X	X	X
BNE	0	1	1	0	X	X	X	X	X	X
SLTI	0	1	1	1	X	X	X	1	1	0
LW	1	0	0	0	X	X	X	0	0	1
SW	1	0	0	1	X	X	X	0	0	1