CSE 331/503 Computer Organization Final Project – MiniMIPS Design

Hasan Mutlu – 1801042673

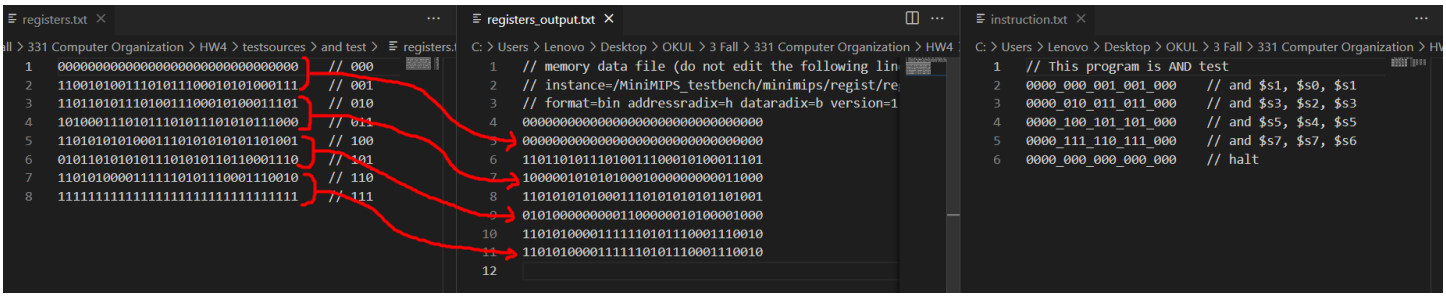
Design Explanation:

- ➔ 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.

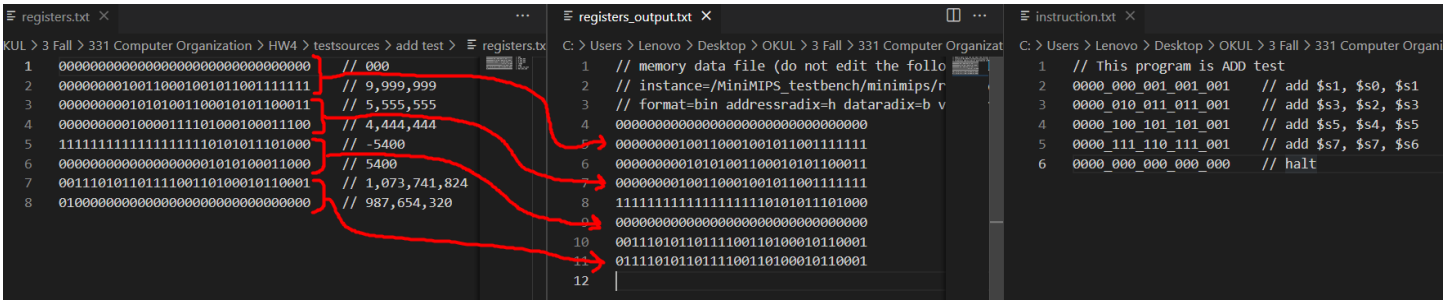
Test Results

-> 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.

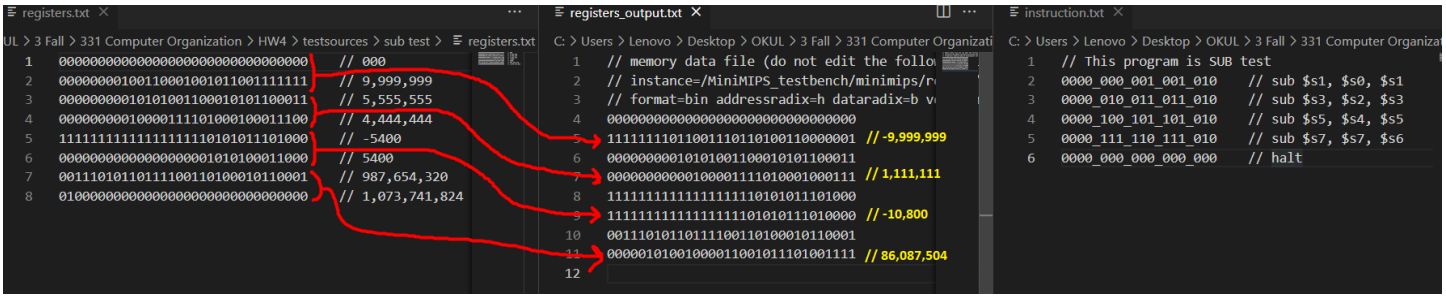
1.) 000 000 AND



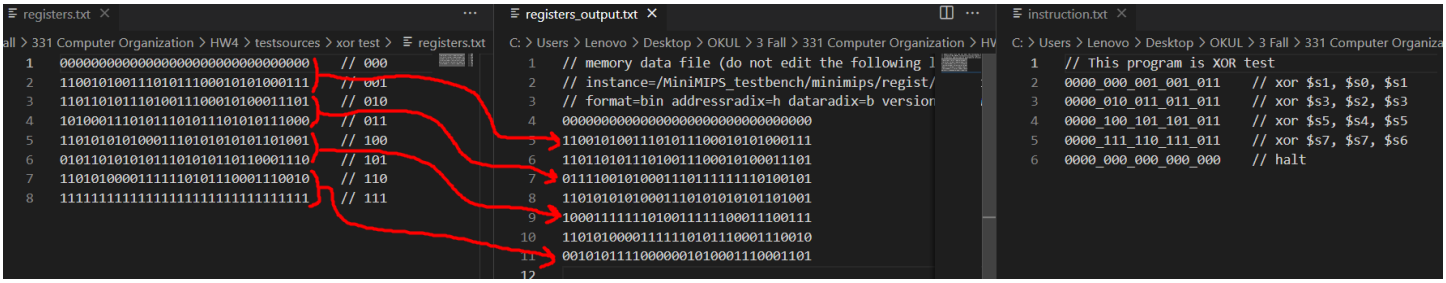
2.) 0000 001 ADD



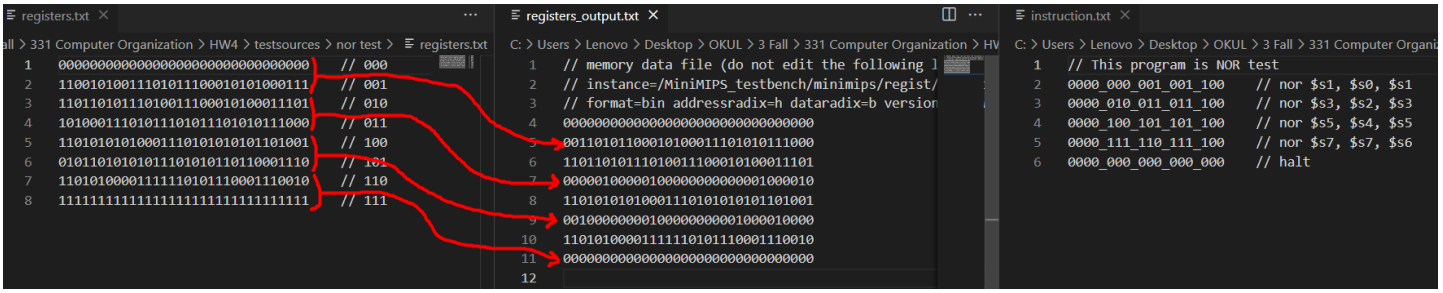
3.) 0000 010 SUB



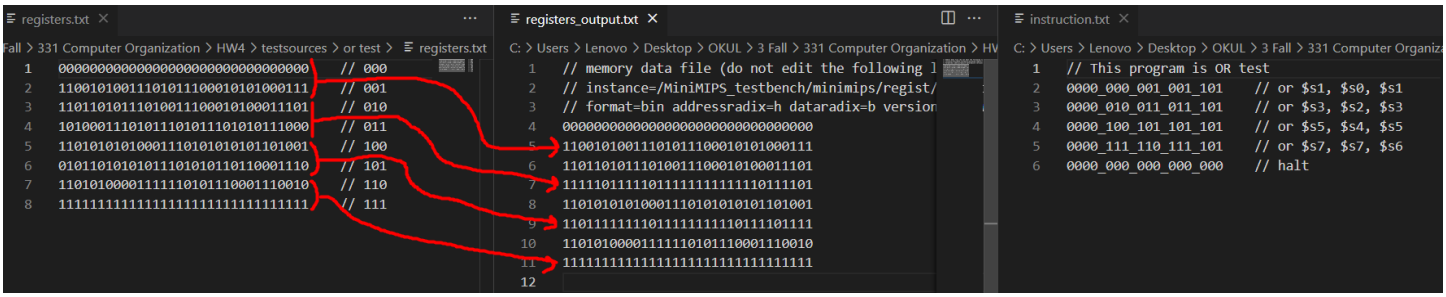
4.) 0000 011 XOR



5.) 0000 100 NOR

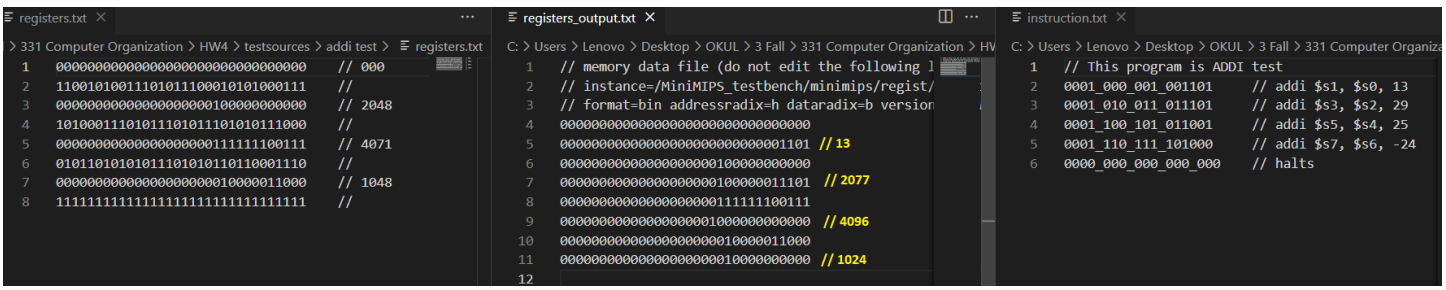


6.) 0000 101 OR

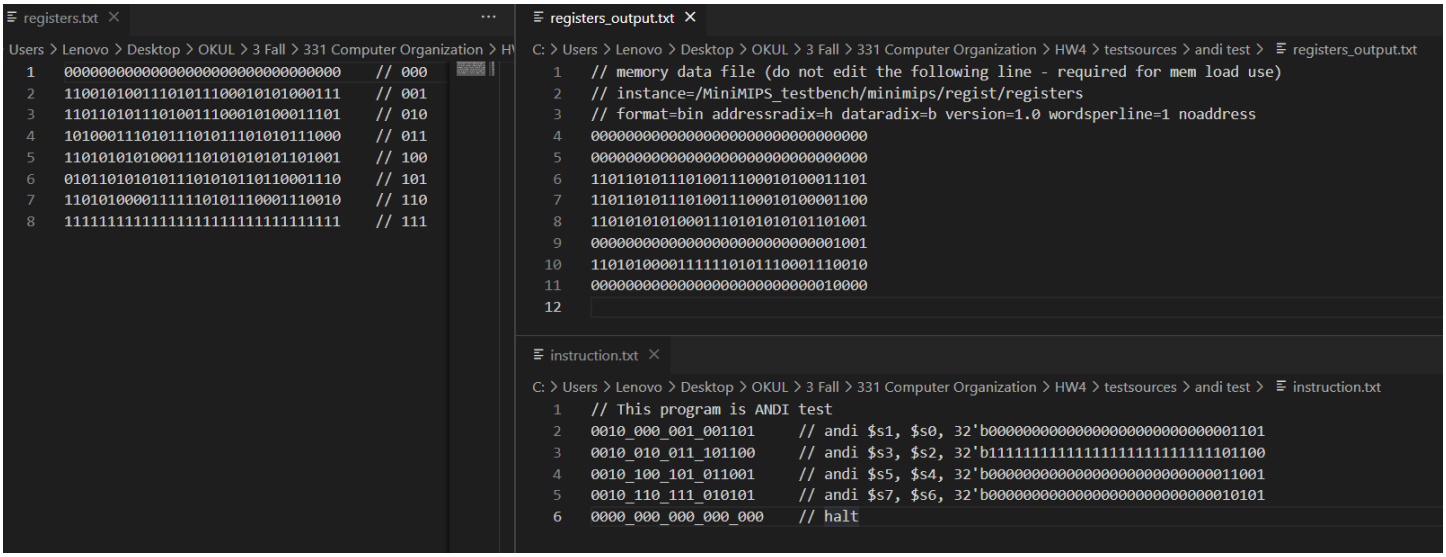


-> In I-Type operation instructions, registers with even addresses (0,2,4,6) are operated with immediate field and the result is saved in next register. There are 4 test results for each instruction type.

8.) 0001 ADDI



9.) 0010 ANDI



registers.txt

```
00000000000000000000000000000000    // 000
11001010011101011100010101000111    // 001
11011010111010011100010100011101    // 010
10100011101011101011101010111000    // 011
11010101010001110101010101101001    // 100
01011010101011101010110110001110    // 101
11010100001111110101110001110010    // 110
11111111111111111111111111111111    // 111
```

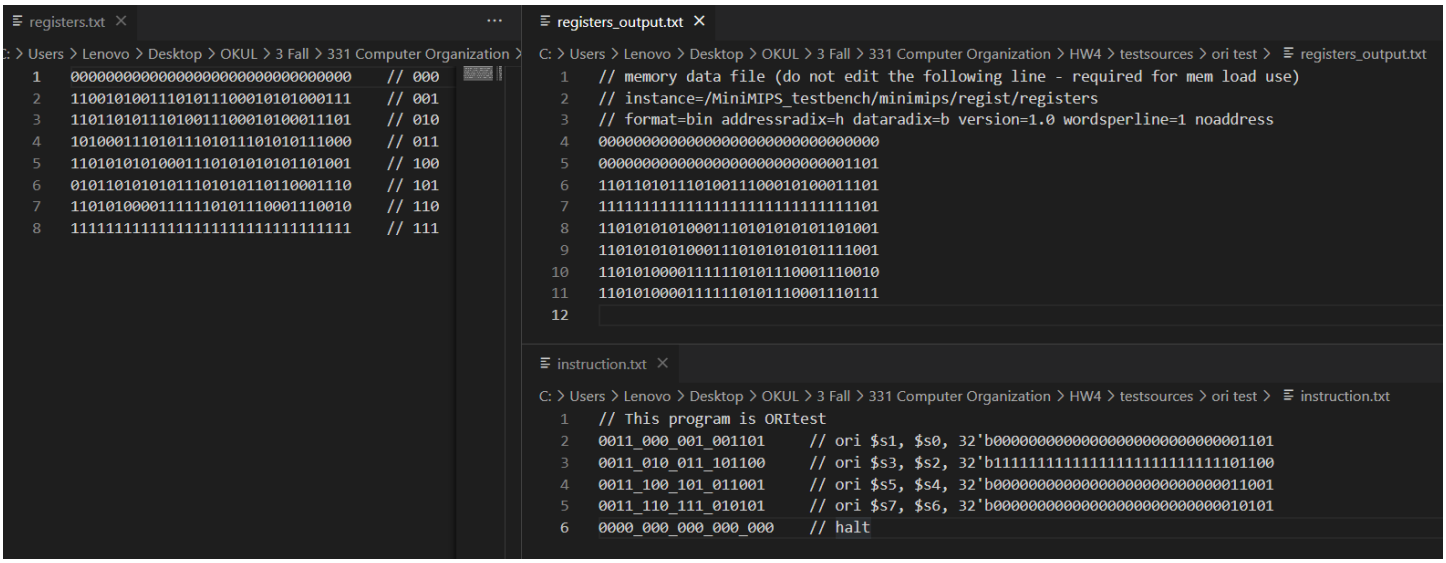
registers_output.txt

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/MiniMIPS_testbench/minimips/regist/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000000
00000000000000000000000000000000
11011010111010011100010100011101
11011010111010011100010100001100
11010101010001110101010101101001
00000000000000000000000000001001
11010100001111110101110001110010
00000000000000000000000000010000
```

instruction.txt

```
// This program is ANDI test
0010_000_001_001101     // andi $s1, $s0, 32'b00000000000000000000000000001101
0010_010_011_101100     // andi $s3, $s2, 32'b11111111111111111111111111101100
0010_100_101_011001     // andi $s5, $s4, 32'b00000000000000000000000000011001
0010_110_111_010101     // andi $s7, $s6, 32'b00000000000000000000000000010101
0000_000_000_000_000    // halt
```

10.) 0011 ORI



registers.txt

```
00000000000000000000000000000000    // 000
11001010011101011100010101000111    // 001
11011010111010011100010100011101    // 010
10100011101011101011101010111000    // 011
11010101010001110101010101101001    // 100
01011010101011101010110110001110    // 101
11010100001111110101110001110010    // 110
11111111111111111111111111111111    // 111
```

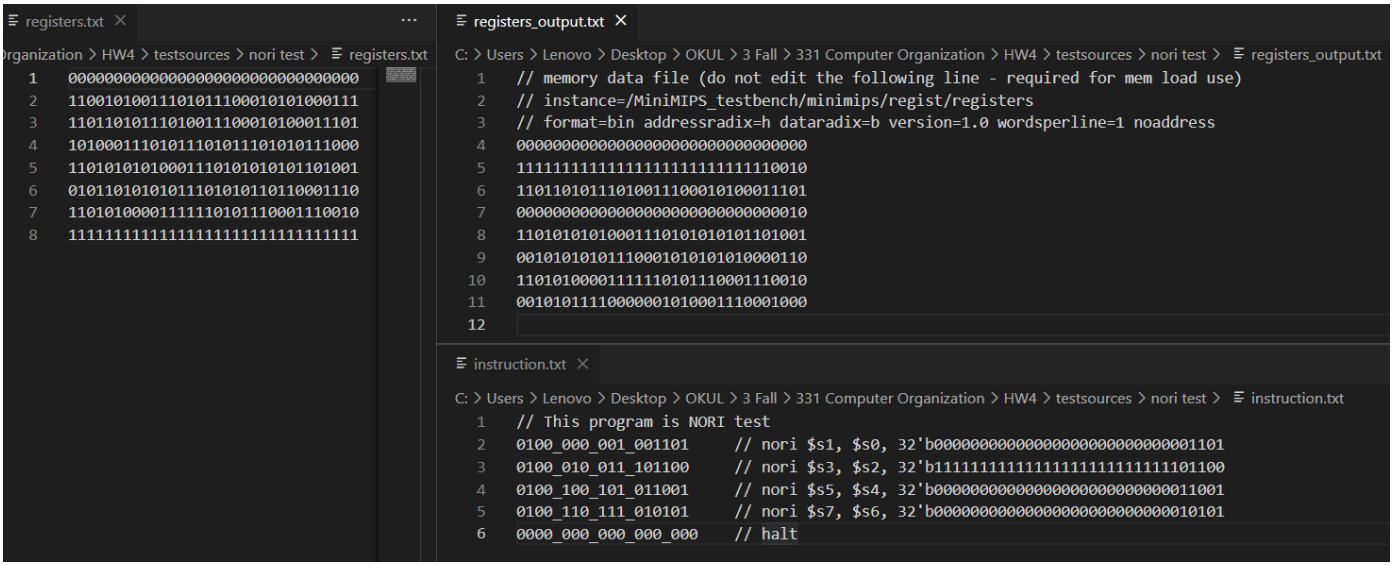
registers_output.txt

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/MiniMIPS_testbench/minimips/regist/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000000
00000000000000000000000000001101
11011010111010011100010100011101
11111111111111111111111111111101
11010101010001110101010101101001
11010101010001110101010101111001
11010100001111110101110001110010
11010100001111110101110001110111
```

instruction.txt

```
// This program is ORItest
0011_000_001_001101     // ori $s1, $s0, 32'b00000000000000000000000000001101
0011_010_011_101100     // ori $s3, $s2, 32'b11111111111111111111111111101100
0011_100_101_011001     // ori $s5, $s4, 32'b00000000000000000000000000011001
0011_110_111_010101     // ori $s7, $s6, 32'b00000000000000000000000000010101
0000_000_000_000_000    // halt
```

11.) 0100 NORI



registers.txt

```
00000000000000000000000000000000
11001010011101011100010101000111
11011010111010011100010100011101
10100011101011101011101010111000
11010101010001110101010101101001
01011010101011101010110110001110
11010100001111110101110001110010
11111111111111111111111111111111
```

registers_output.txt

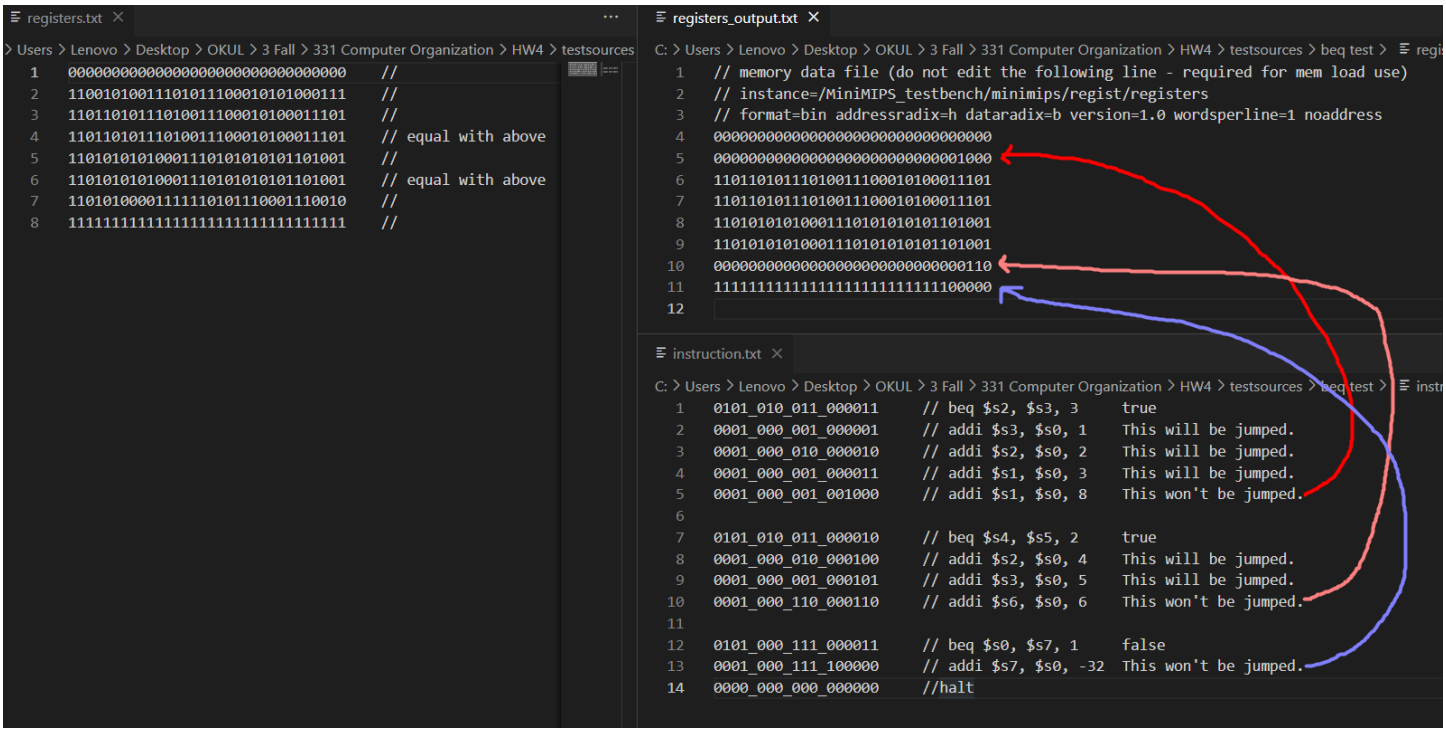
```
// memory data file (do not edit the following line - required for mem load use)
// instance=/MiniMIPS_testbench/minimips/regist/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000000
11111111111111111111111111110010
11011010111010011100010100011101
00000000000000000000000000000010
11010101010001110101010101101001
00101010101110001010101010000110
11010100001111110101110001110010
00101011110000001010001110001000
```

instruction.txt

```
// This program is NORI test
0100_000_001_001101     // nori $s1, $s0, 32'b00000000000000000000000000001101
0100_010_011_101100     // nori $s3, $s2, 32'b11111111111111111111111111101100
0100_100_101_011001     // nori $s5, $s4, 32'b00000000000000000000000000011001
0100_110_111_010101     // nori $s7, $s6, 32'b00000000000000000000000000010101
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

```
00000000000000000000000000000000    //
11001010011101011100010101000111    //
11011010111010011100010100011101    //
11011010111010011100010100011101    // equal with above
11010101010001110101010101101001    //
11010101010001110101010101101001    // equal with above
11010100001111110101110001110010    //
11111111111111111111111111111111    //
```

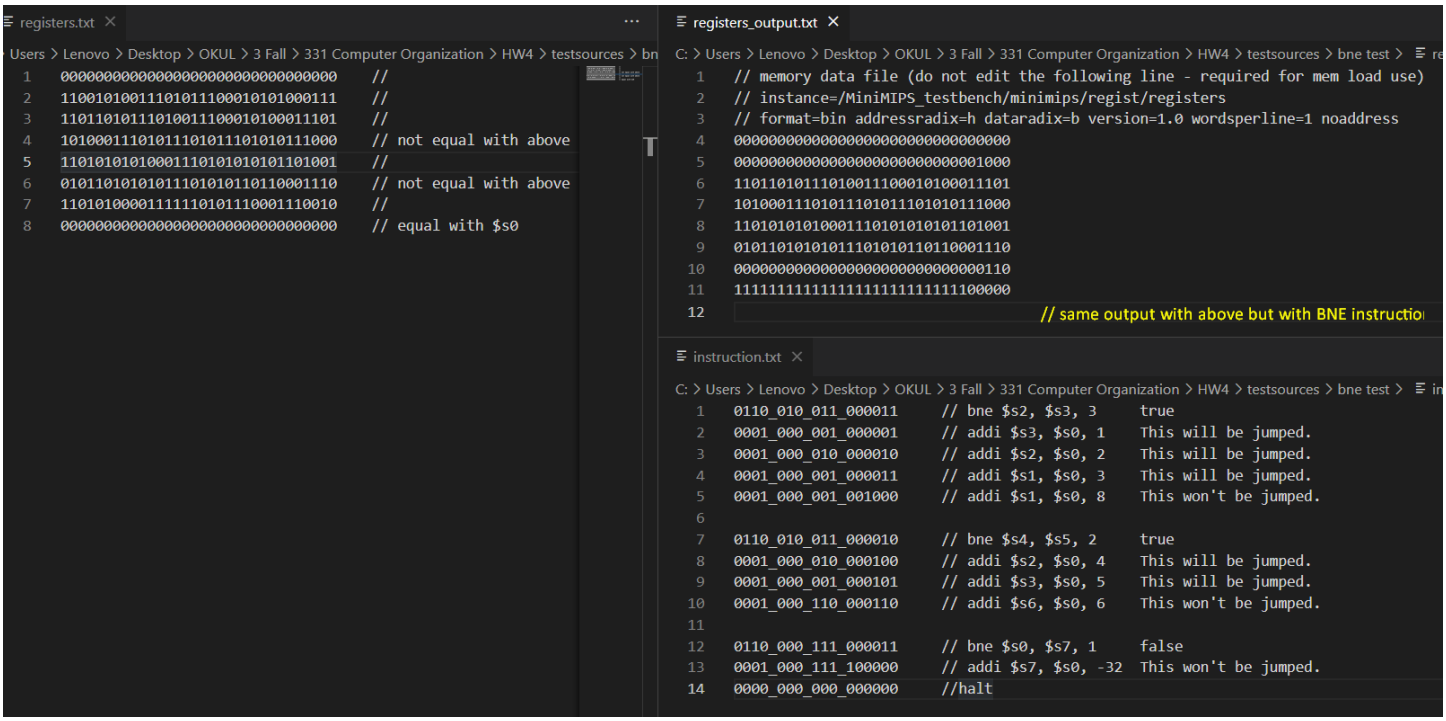
registers_output.txt

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/MiniMIPS_testbench/minimips/regist/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000000
00000000000000000000000000001000
11011010111010011100010100011101
11011010111010011100010100011101
11010101010001110101010101101001
11010101010001110101010101101001
00000000000000000000000000000110
11111111111111111111111111100000
```

instruction.txt

```
0101_010_011_000011     // beq $s2, $s3, 3     true
0001_000_001_000001     // addi $s3, $s0, 1    This will be jumped.
0001_000_010_000010     // addi $s2, $s0, 2    This will be jumped.
0001_000_001_000011     // addi $s1, $s0, 3    This will be jumped.
0001_000_001_001000     // addi $s1, $s0, 8    This won't be jumped.

0101_010_011_000010     // beq $s4, $s5, 2     true
0001_000_010_000100     // addi $s2, $s0, 4    This will be jumped.
0001_000_001_000101     // addi $s3, $s0, 5    This will be jumped.
0001_000_110_000110     // addi $s6, $s0, 6    This won't be jumped.

0101_000_111_000011     // beq $s0, $s7, 1     false
0001_000_111_100000     // addi $s7, $s0, -32  This won't be jumped.
0000_000_000_000000     //halt
```

13.) 0110 BNE



registers.txt

```
00000000000000000000000000000000    //
11001010011101011100010101000111    //
11011010111010011100010100011101    //
10100011101011101011101010111000    // not equal with above
11010101010001110101010101101001    //
01011010101011101010110110001110    // not equal with above
11010100001111110101110001110010    //
00000000000000000000000000000000    // equal with $s0
```

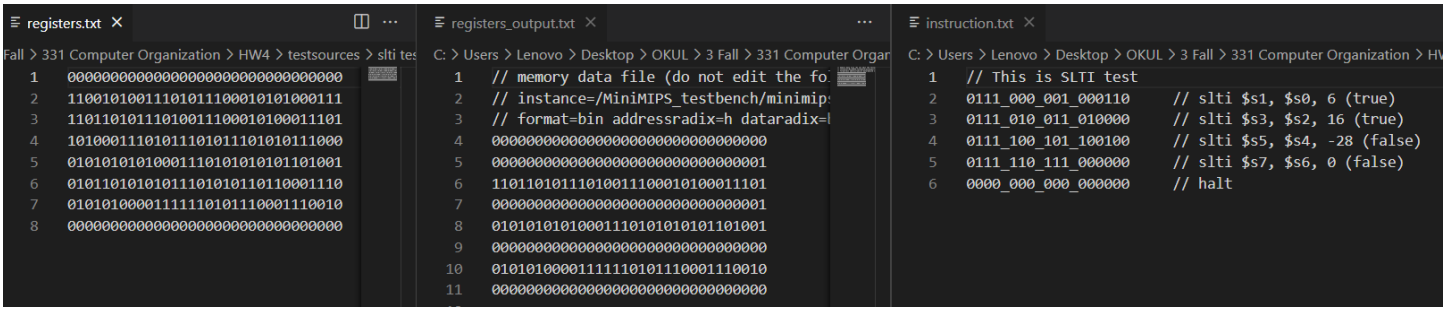
registers_output.txt

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/MiniMIPS_testbench/minimips/regist/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000000
00000000000000000000000000001000
11011010111010011100010100011101
10100011101011101011101010111000
11010101010001110101010101101001
01011010101011101010110110001110
00000000000000000000000000000110
11111111111111111111111111100000
                                  // same output with above but with BNE instructio
```

instruction.txt

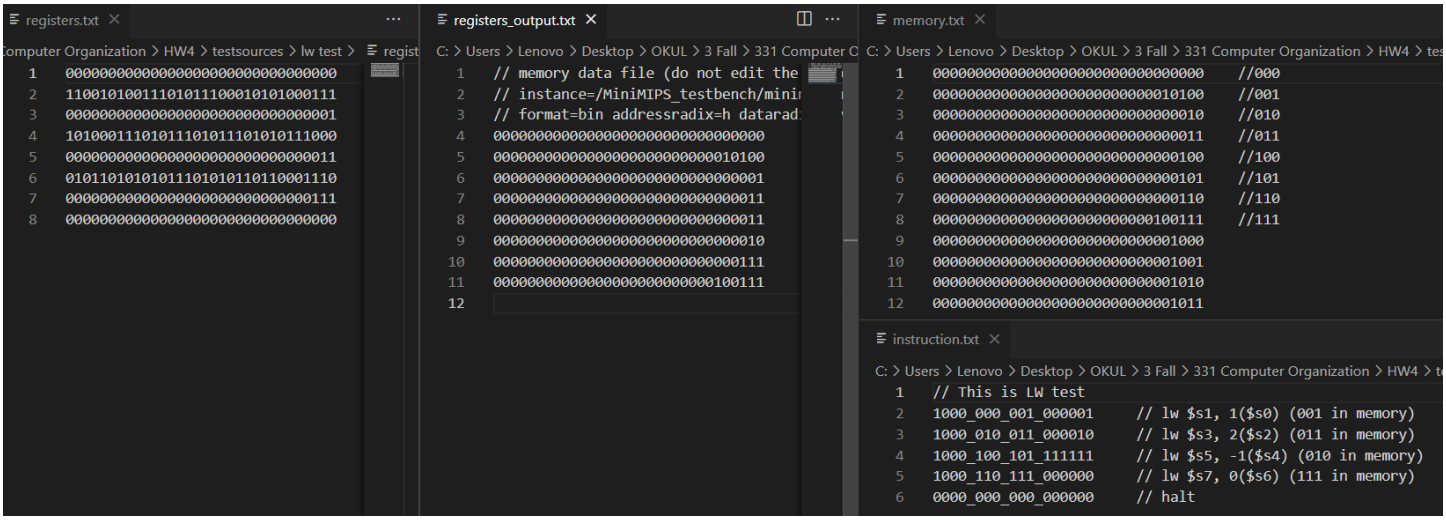
```
0110_010_011_000011     // bne $s2, $s3, 3     true
0001_000_001_000001     // addi $s3, $s0, 1    This will be jumped.
0001_000_010_000010     // addi $s2, $s0, 2    This will be jumped.
0001_000_001_000011     // addi $s1, $s0, 3    This will be jumped.
0001_000_001_001000     // addi $s1, $s0, 8    This won't be jumped.

0110_010_011_000010     // bne $s4, $s5, 2     true
0001_000_010_000100     // addi $s2, $s0, 4    This will be jumped.
0001_000_001_000101     // addi $s3, $s0, 5    This will be jumped.
0001_000_110_000110     // addi $s6, $s0, 6    This won't be jumped.

0110_000_111_000011     // bne $s0, $s7, 1     false
0001_000_111_100000     // addi $s7, $s0, -32  This won't be jumped.
0000_000_000_000000     //halt
```

14.) 0111 SLTI

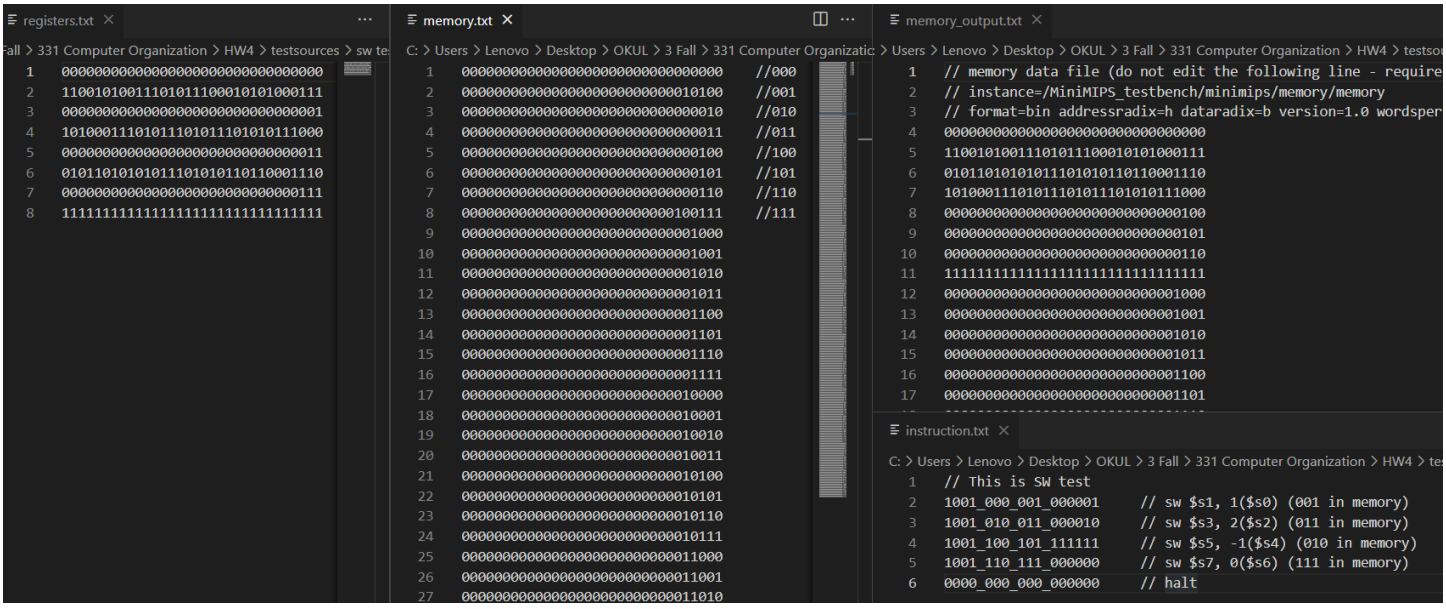


-> In Load-Store tests, data in registers with odd addresses(1,3,5,7) are loaded or stored to memory.

15.) 1000 LW

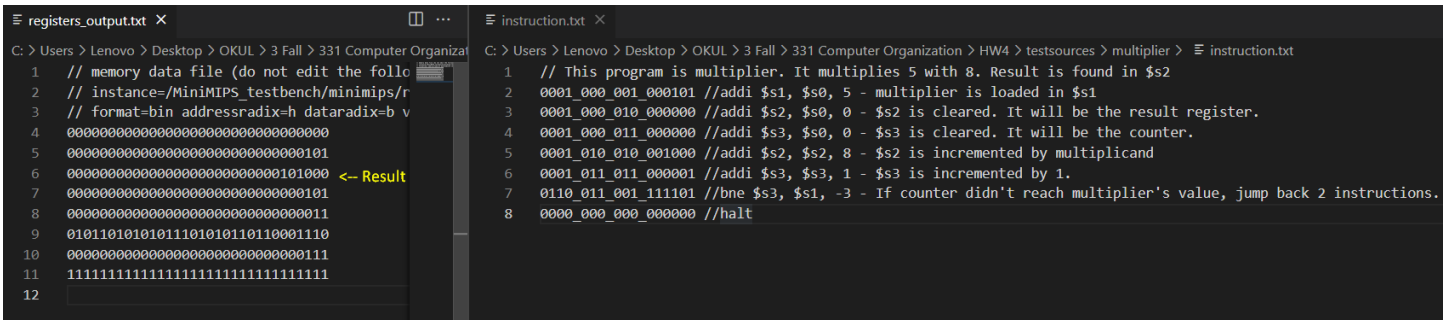


16.) 1001 SW



Bonus Test: Multiplier

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



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