Name: Enes Abdülhalik

Student No: 1901042803

CSE331 – Final Project Report

Module Schematics:

- Main Processor: (without test signals)

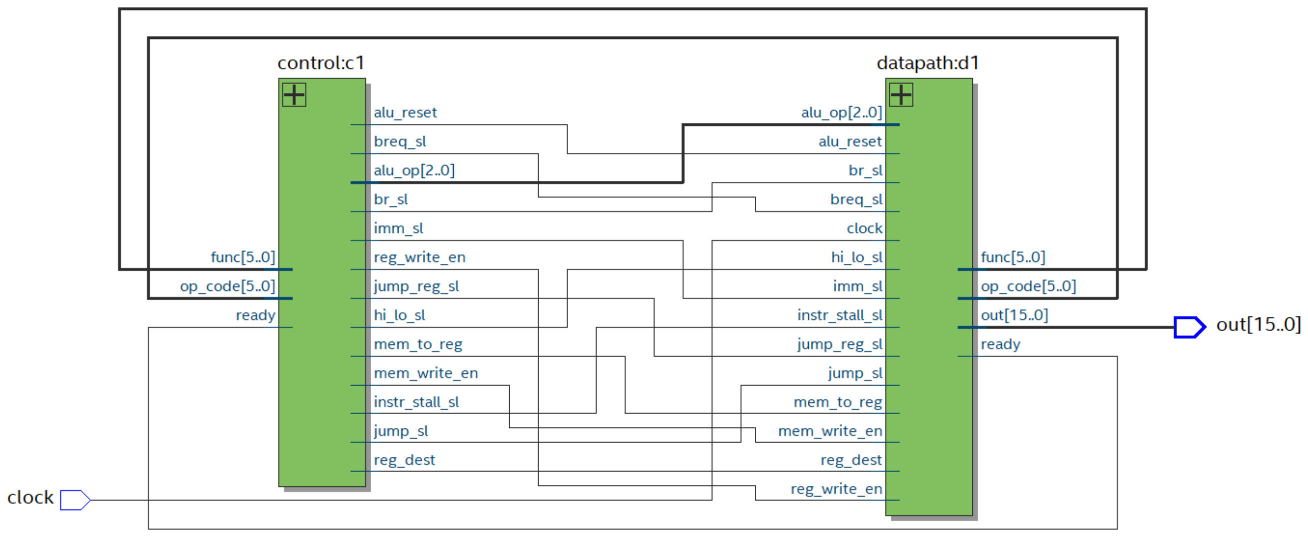

Figure 1The layout of the main processor without the test signals used in test benches. The design separates the datapath from the control unit completely.

- Datapath:

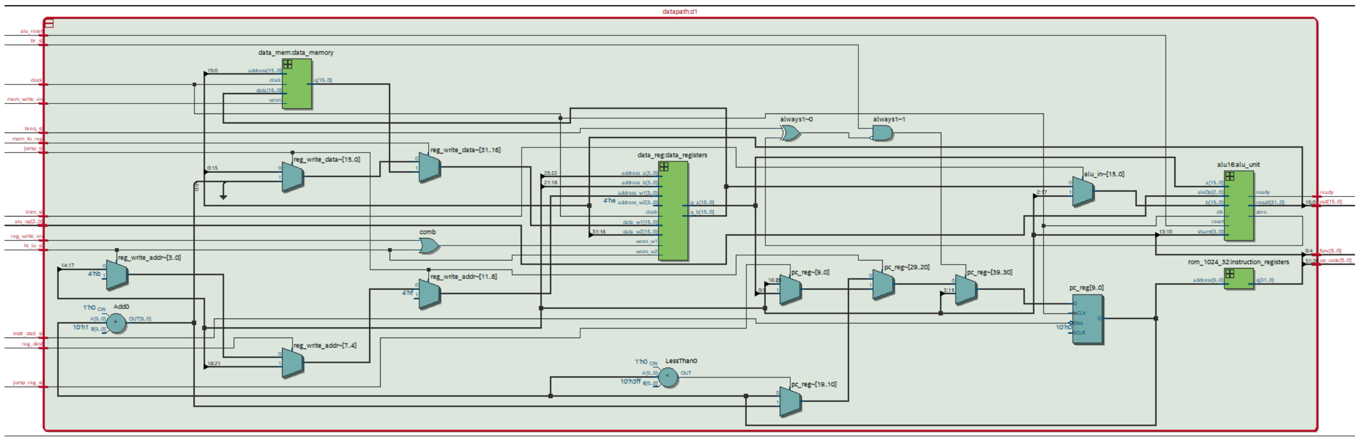

Figure 2The layout of the datapath.

- Control Unit:

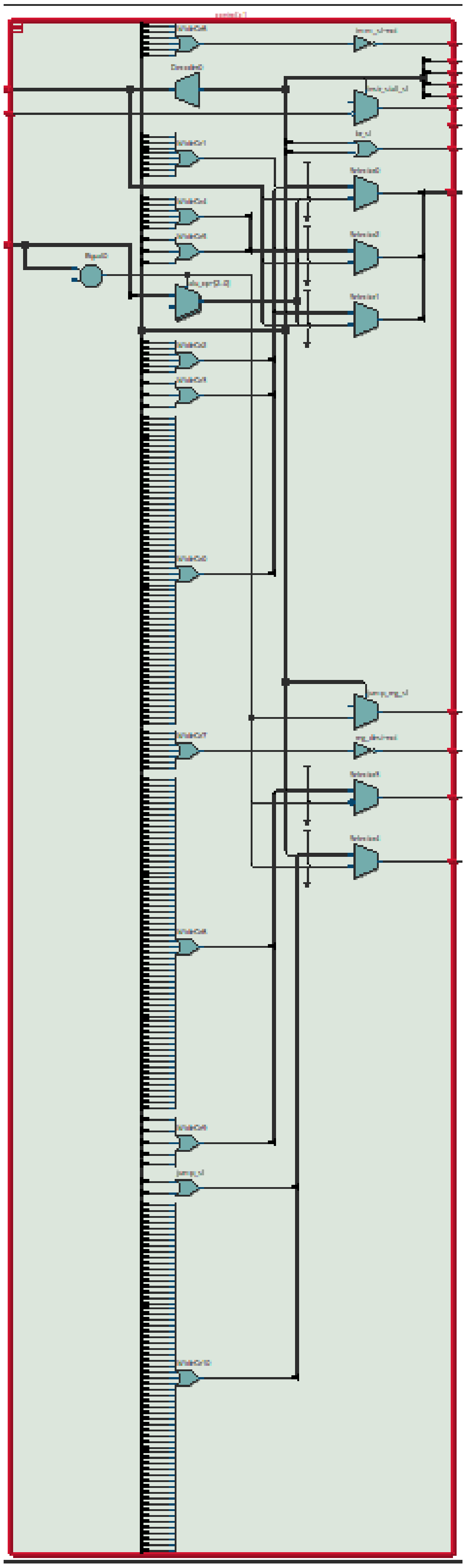

Figure 3The layout of the control unit

- Data registers:

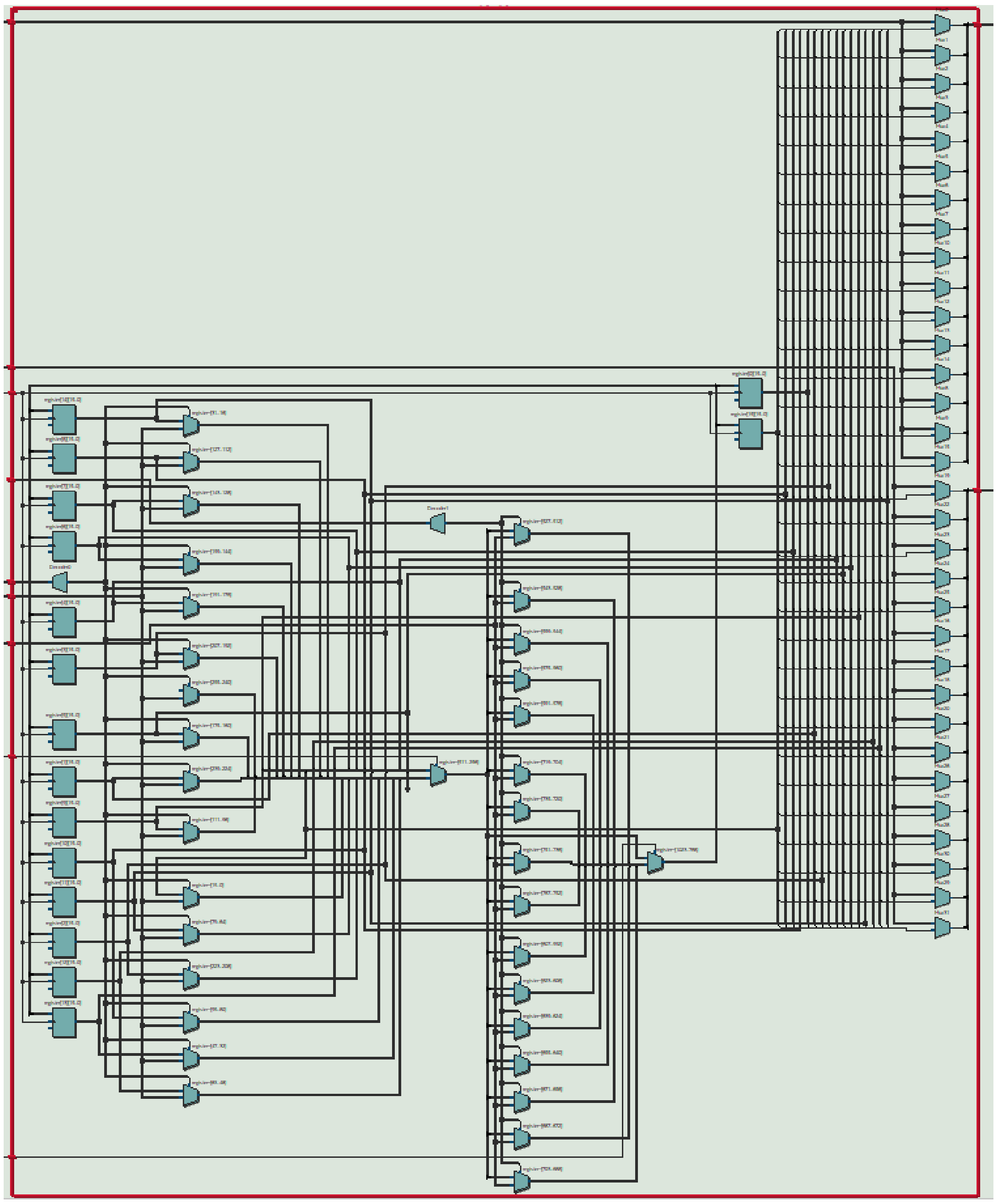

Figure 4The layout of the data registers. These registers do not infer memory on the FPGA, to get rid of any unnecessary registers in the way.

- ALU Unit:

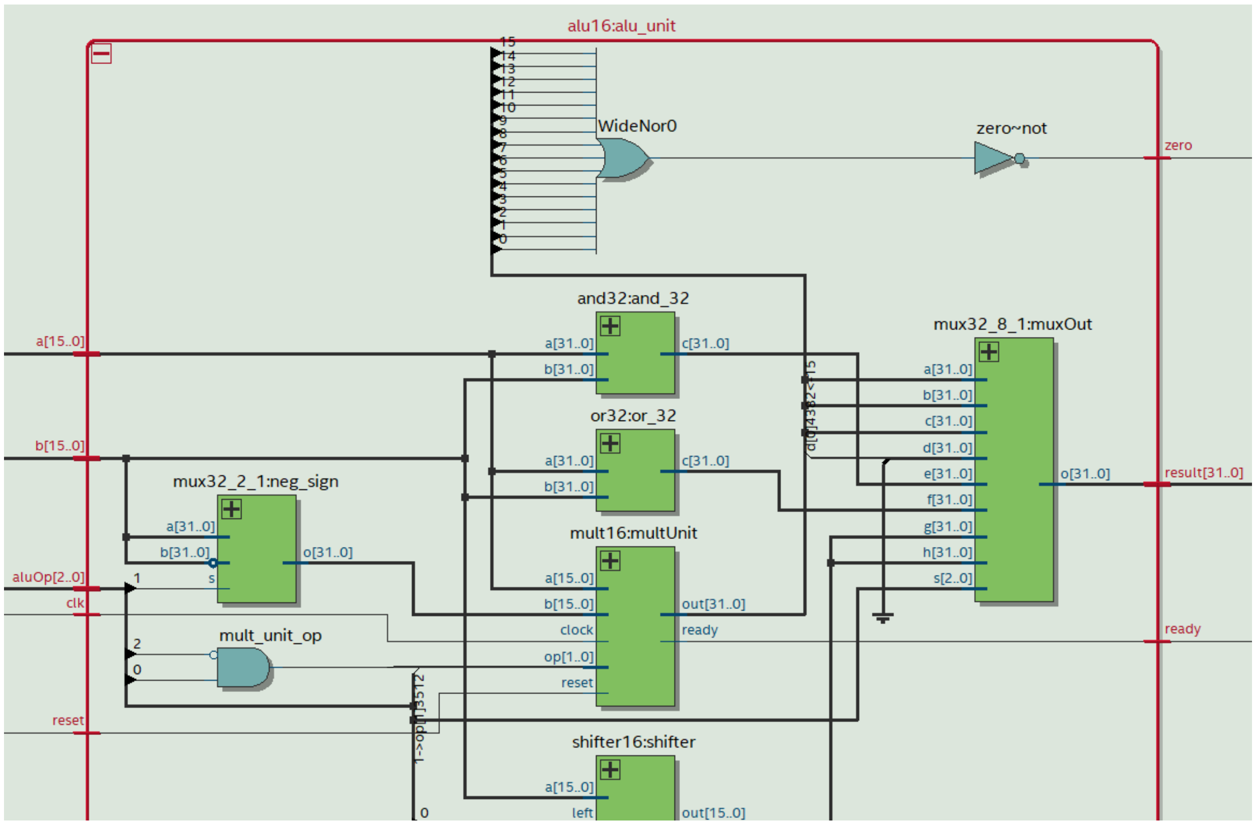

Figure 5The ALU layout. this 16-bit ALU supports addition, subtraction, set-on-less-than, left and right shifting, and, or, and 32-bit multiplication. It only needs a 3-bit ALU operation signal.

- Multiplication Unit:

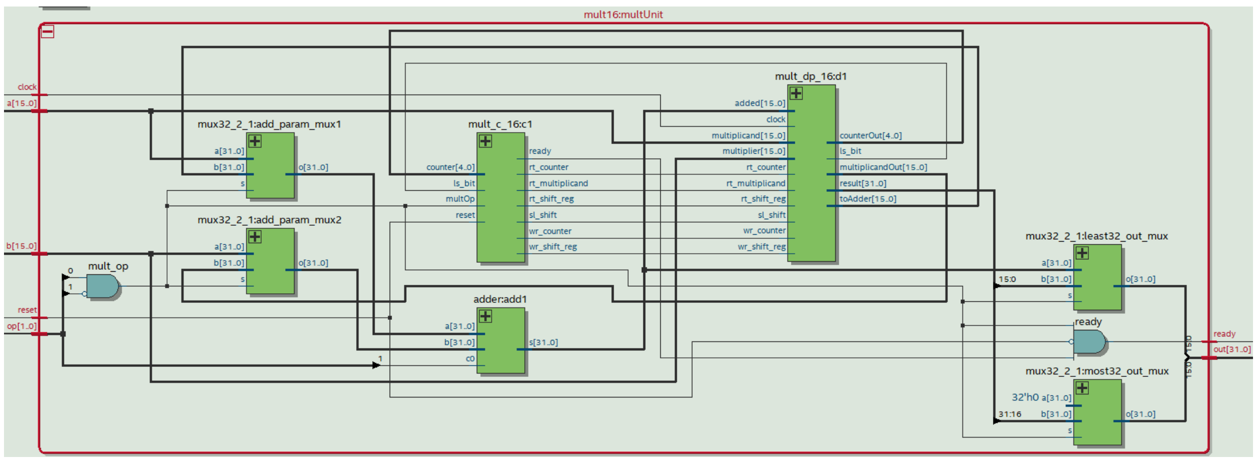

Figure 6The multiplication unit inside the ALU.

- Multiplication Unit Datapath:

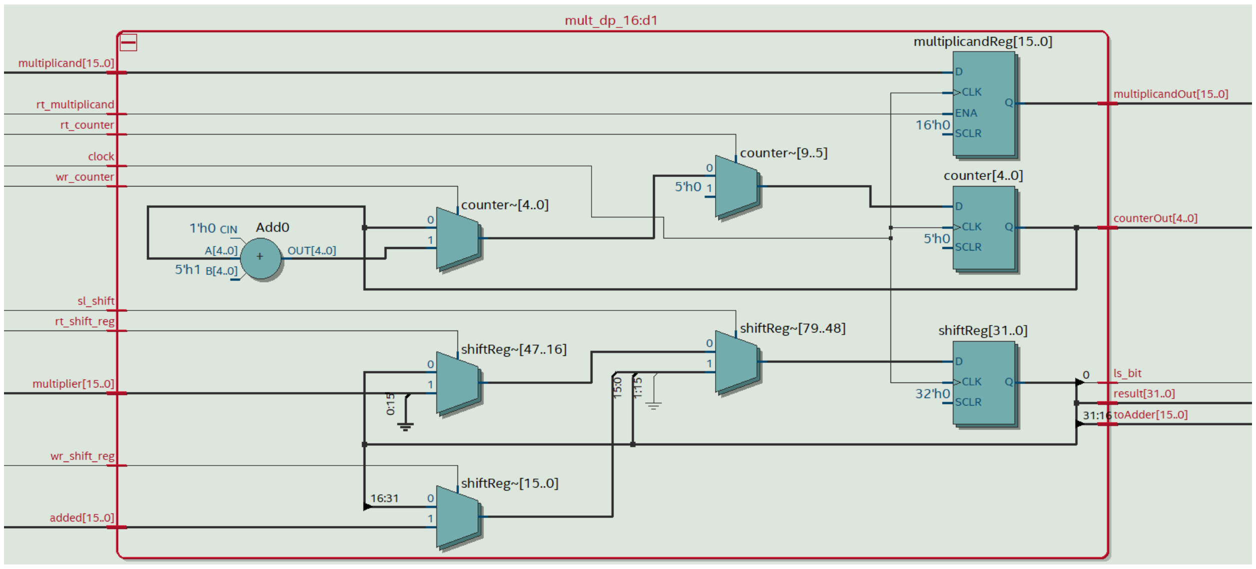

Figure 7The datapath of the multiplication unit inside the ALU.

- Multiplication Unit Controller:

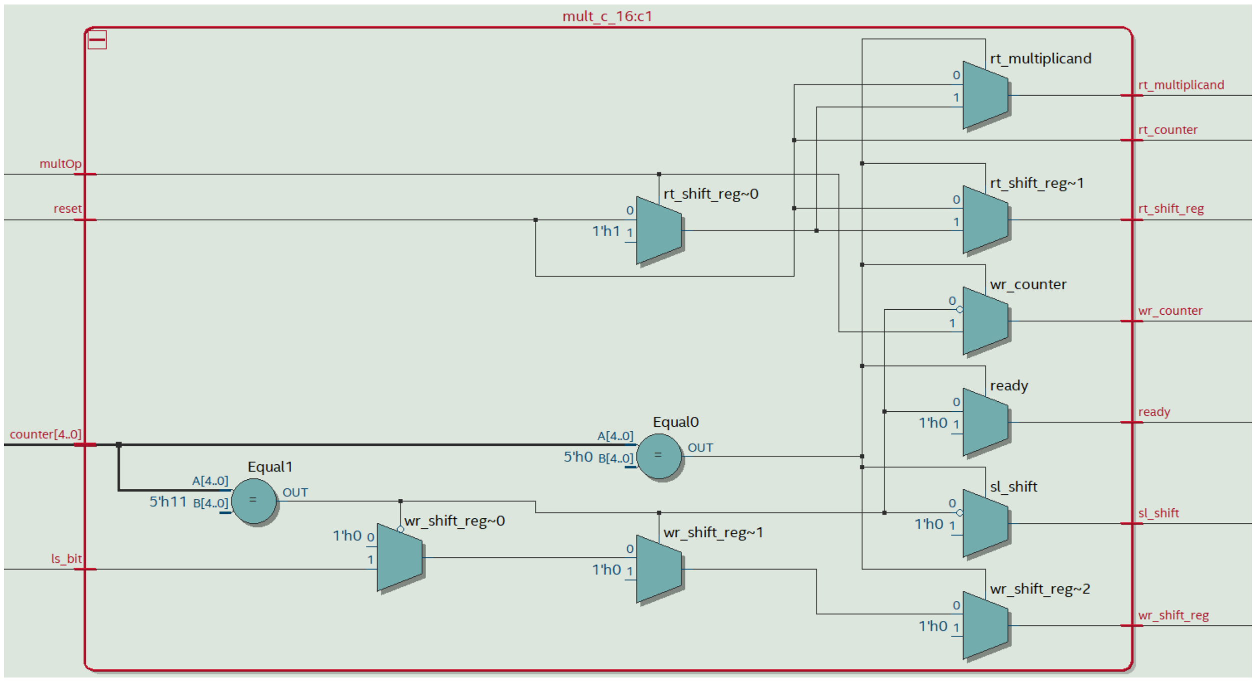

Figure 8The control unit of the multiplication unit inside the ALU.

Supported instructions:

name	format	operation	Opcode/func	Type
Load immediate	Li $1, 100	$1 = 100	001111	I
add	add $1, $2, $3	$1=$2+$3	100000	R
Add immediate	addi $1, $2, 100	$1=$2+100	001000	I
subtract	sub $1, $2, $3	$1=$2-$3	100010	R
and	and $1,$2,$3	$1=$2&$3	100100	R
And immediate	andi $1,$2,100	$1=$2&100	001100	I
Or	or $1,$2,$3	$1=$2|$3	100101	R
Or immediate	ori $1,$2,100	$1=$2|100	001101	I

Set less than	slt $1,$2,$3	if($2<$3) $1=1;
		else $1=0
			101011	R

Set less than imm	slti $1,$2,100	if($2<100) $1=1;
		else $1=0
			001011	I

Shift left logical	sll $1,$2,10	$1=$2<<10	000111	R
Shift right logical	srl $1,$2,10	$1=$2>>10	000110	R
Multiply	mult $1, $2	{$14, $13} = $1 * $2	011000	

Branch on equal	beq $1,$2,100	if($1==$2) pc = 100
		else pc = pc + 1
			000100	I

Branch on not equal	bne $1,$2,100	if($1!=$2) pc = 100
		else pc = pc + 1
			000101	I

Jump	j 1000	Pc = 1000	000010	J
Jump and link	jal 1000	$15 = pc + 1; pc = 1000;	000011	J
Jump register	jr $1	Pc = $1	001000	R
Load word	lw $1,100($2)	$1=Memory[$2+100]	100011	I
Store word	sw $1,100($2)	Memory[$2+100]=$1	101011	I

Registers:

Number	Use
0	Zero register; always 0.
1-12	General purpose
13	lo register; stores least significant 16-bits of multiplication result
14	hi register; stores most significant 16-bits of multiplication result
15	ra register; saves a return value after jump and link.

Test benches:

Note: all of the test benches and their memory init. and assembly and Verilog files are in “test_benches” folder in the main directory.

1- Load immediate:

- Assembly code:

```
li $1, 1
```

- Output in Questasim:

```
# The decimal value in register 1 is:     x
# finished after                  0 cycles
#
# The decimal value in register 1 is:     1
# finished after                  1 cycles
```

2- Add and Add Immediate:

- Assembly code:

```
li $1, 2
li $2, 3
add $3, $1, $2
addi $4, $3, 4
```

- Output in Questasim:

```
# The value in register 1 is:     2
# The value in register 2 is:     3
# The value in register 3 (before assignment) is:     x
# The value in register 4 (before assignment) is:     x
#
# The value in register 3 ($1 + $2) is:     x
# The value in register 4 ($3 + 1) is:     x
# finished after                  0 cycles
#
# The value in register 3 ($1 + $2) is:     5
# The value in register 4 ($3 + 1) is:     x
# finished after                  1 cycles
```

```
#
# The value in register 3 ($1 + $2) is:     5
# The value in register 4 ($3 + 1) is:     9
# finished after                    2 cycles
```

3- Subtract:

- Assembly code:

```
li $1, 7
li $2, 3
sub $3, $1, $2
```

- Output in Questasim:

```
# The value in register 1 is:     7
# The value in register 2 is:     3
# The value in register 3 (before assignment) is:     x
#
# The value in register 3 ($1 - $2) is:     x
# finished after                    0 cycles
#
# The value in register 3 ($1 - $2) is:     4
# finished after                    1 cycles
```

4- And and And Immediate:

- Assembly code:

```
li $1, 3
li $2, 2
and $3, $1, $2
andi $4, $1, 2
```

- Output in Questasim:

```
# The value in register 1 is: 0000000000000011
# The value in register 2 is: 0000000000000010
# The value in register 3 (before assignment) is: xxxxxxxxxxxxxxxx
# The value in register 4 (before assignment) is: xxxxxxxxxxxxxxxx
```

```
#
# The value in register 3 ($1 and $2) is: xxxxxxxxxxxxxxxx
# The value in register 4 ($1 andi 2) is: xxxxxxxxxxxxxxxx
# finished after                   0 cycles
#
# The value in register 3 ($1 and $2) is: 0000000000000010
# The value in register 4 ($1 andi 2) is: xxxxxxxxxxxxxxxx
# finished after                   1 cycles
#
# The value in register 3 ($1 and $2) is: 0000000000000010
# The value in register 4 ($1 andi 2) is: 0000000000000010
# finished after                   2 cycles
```

5- OR and OR Immediate:

- Assembly code:

```
li $1, 3
li $2, 2
or $3, $1, $2
ori $4, $1, 2
```

- Output in Questasim:

```
# The value in register 1 is: 0000000000000011
# The value in register 2 is: 0000000000000010
# The value in register 3 (before assignment) is: xxxxxxxxxxxxxxxx
# The value in register 4 (before assignment) is: xxxxxxxxxxxxxxxx
#
# The value in register 3 ($1 or $2) is: xxxxxxxxxxxxxxxx
# The value in register 4 ($1 ori 2) is: xxxxxxxxxxxxxxxx
# finished after                   0 cycles
#
# The value in register 3 ($1 or $2) is: 0000000000000011
# The value in register 4 ($1 ori 2) is: xxxxxxxxxxxxxxxx
# finished after                   1 cycles
#
```

```
# The value in register 3 ($1 or $2) is: 0000000000000011
# The value in register 4 ($1 ori 2) is: 0000000000000011
# finished after            2 cycles
```

6- Set on Less Than and Set on Less Than Immediate:

- Assembly code:

```
li $1, 1
li $2, 2
slt $3, $1, $2
slti $4, $2, 1
```

- Output in Questasim:

```
# The value in register 1 is:    1
# The value in register 2 is:    2
# The value in register 3 (before assignment) is:    x
# The value in register 4 (before assignment) is:    x
#
# The value in register 3 ($1 is smaller than $2) is:    x
# The value in register 4 ($2 is smaller than immediate 1) is:    x
# finished after            0 cycles
#
# The value in register 3 ($1 is smaller than $2) is:    1
# The value in register 4 ($2 is smaller than immediate 1) is:    x
# finished after            1 cycles
#
# The value in register 3 ($1 is smaller than $2) is:    1
# The value in register 4 ($2 is smaller than immediate 1) is:    0
# finished after            2 cycles
```

7- Shift left and Shift right:

- Assembly code:

```
li $1, 8
sll $3, $1, 2
srl $4, $1, 2
```

- Output in Questasim:

```
# The value in register 1 is: 0000000000001000
# The value in register 3 (before assignment) is: xxxxxxxxxxxxxxxx
# The value in register 4 (before assignment) is: xxxxxxxxxxxxxxxx
#
# The value in register 3 (shift left $1 2 bits) is: xxxxxxxxxxxxxxxx
# The value in register 4 (shift right $1 2 bits) is: xxxxxxxxxxxxxxxx
# finished after                   0 cycles
#
# The value in register 3 (shift left $1 2 bits) is: 0000000000100000
# The value in register 4 (shift right $1 2 bits) is: xxxxxxxxxxxxxxxx
# finished after                   1 cycles
#
# The value in register 3 (shift left $1 2 bits) is: 0000000000100000
# The value in register 4 (shift right $1 2 bits) is: 0000000000000010
# finished after                   2 cycles
```

8- Multiply:

- Assembly code:

```
li $1, 3000
li $2, 50000
mult $1, $2
```

- Output in Questasim:

```
# The value in register 1 (multiplier) is:  3000
# The value in register 2 (multiplicand) is: 50000
#
# The value in register 14 (hi) is:     x
# The value in register 13 (lo) is:     x
# finished after                   0 cycles
#
# The value in register 14 (hi) is:     0
# The value in register 13 (lo) is:     0
# finished after                   1 cycles
# The value in register 14 (hi) is:     0
# The value in register 13 (lo) is: 50000
# finished after                  2 cycles
#
# The value in register 14 (hi) is:     0
# The value in register 13 (lo) is: 25000
# finished after                  4 cycles
#
# The value in register 14 (hi) is:     0
# The value in register 13 (lo) is: 12500
# finished after                  5 cycles
#
# The value in register 14 (hi) is:     0
# The value in register 13 (lo) is:  6250
# finished after                  6 cycles
#
# The value in register 14 (hi) is:     0
# The value in register 13 (lo) is:  3125
# finished after                  7 cycles
#
# The value in register 14 (hi) is:  1500
# The value in register 13 (lo) is:  1562
# finished after                  8 cycles
#
# The value in register 14 (hi) is:   750
# The value in register 13 (lo) is:   781
# finished after                  9 cycles
#
# The value in register 14 (hi) is:  1875
# The value in register 13 (lo) is:   390
# finished after                 10 cycles
#
# The value in register 14 (hi) is:   937
# The value in register 13 (lo) is: 32963
# finished after                 11 cycles
```

```
#
# The value in register 14 (hi) is:  1968
# The value in register 13 (lo) is: 49249
# finished after                12 cycles
#
# The value in register 14 (hi) is:  2484
# The value in register 13 (lo) is: 24624
# finished after                13 cycles
#
# The value in register 14 (hi) is:  1242
# The value in register 13 (lo) is: 12312
# finished after                14 cycles
#
# The value in register 14 (hi) is:   621
# The value in register 13 (lo) is:  6156
# finished after                15 cycles
#
# The value in register 14 (hi) is:   310
# The value in register 13 (lo) is: 35846
# finished after                16 cycles
#
# The value in register 14 (hi) is:   155
# The value in register 13 (lo) is: 17923
# finished after                17 cycles
#
# The value in register 14 (hi) is:  1577
# The value in register 13 (lo) is: 41729
# finished after                18 cycles
#
# The value in register 14 (hi) is:  2288
# The value in register 13 (lo) is: 53632
# finished after                19 cycles
```

9- Branch on not equal:

- Assembly code:

```
li $1, 1
li $2, 10
bne $1, $2, 6
li $3, 1
li $3, 2
li $3, 3
li $3, 4
li $3, 5
```

- Output in Questasim:

```
# The value in register 3 is:     x
# finished after                  0 cycles
#
# The value in register 3 is:     4
# finished after                  2 cycles
#
# The value in register 3 is:     5
# finished after                  3 cycles
```

10- Branch on equal

- Assembly code:

```
li $1, 1
li $2, 10
beq $1, $2, 6
li $3, 1
li $3, 2
li $3, 3
li $3, 4
li $3, 5
```

- Output in Questasim:

```
# The value in register 3 is:     x
# finished after                  0 cycles
#
# The value in register 3 is:     1
# finished after                  2 cycles
#
# The value in register 3 is:     2
# finished after                  3 cycles
#
# The value in register 3 is:     3
# finished after                  4 cycles
#
# The value in register 3 is:     4
# finished after                  5 cycles
#
# The value in register 3 is:     5
# finished after                  6 cycles
```

11- Jump:

- Assembly code:

```
j 4
li $3, 1
li $3, 2
li $3, 3
li $3, 4
li $3, 5
```

- Output in Questasim:

```
# The value in register 3 is:     x
# finished after                  0 cycles
#
```

```
# The value in register 3 is:     4
# finished after                  2 cycles
#
# The value in register 3 is:     5
# finished after                  3 cycles
```

12- Jump and link:

- Assembly code:

```
jal 4
li $3, 1
li $3, 2
li $3, 3
li $3, 4
li $3, 5
```

- Output in Questasim:

```
# The value in register 3 is:     x
# The value in register ra is:     x
# finished after                  0 cycles
#
# The value in register 3 is:     x
# The value in register ra is:     1
# finished after                  1 cycles
#
# The value in register 3 is:     4
# The value in register ra is:     1
# finished after                  2 cycles
#
# The value in register 3 is:     5
# The value in register ra is:     1
# finished after                  3 cycles
```

13- Jump register:

- Assembly code:

```
jal 5
li $3, 1
li $3, 2
li $3, 3
j 8
li $3, 4
li $3, 5
jr $15
```

- Output in Questasim:

```
# The value in register 3 is:     x
# The value in register ra is:     x
# finished after                  0 cycles
#
# The value in register 3 is:     x
# The value in register ra is:     1
# finished after                  1 cycles
#
# The value in register 3 is:     4
# The value in register ra is:     1
# finished after                  2 cycles
#
# The value in register 3 is:     5
# The value in register ra is:     1
# finished after                  3 cycles
#
# The value in register 3 is:     1
# The value in register ra is:     1
# finished after                  5 cycles
#
# The value in register 3 is:     2
# The value in register ra is:     1
# finished after                  6 cycles
```

```
#
# The value in register 3 is:     3
# The value in register ra is:     1
# finished after                  7 cycles
```

14- Store word:

- Assembly code:

```
li $1, 2
li $2, 45
sw $2, 0($1)
```

- Output in Questasim:

```
# The value in register 1 (address to memory) is:     2
# The value in register 2 (to store in memory) is:    45
# The value in address     2 in memory is:     0
# finished after                  0 cycles
#
# The value in register 1 (address to memory) is:     2
# The value in register 2 (to store in memory) is:    45
# The value in address     2 in memory is:    45
# finished after                  1 cycles
```

15- Store word:

- Assembly code:

```
li $1, 2
li $2, 45
sw $2, 0($1)
lw $3, 0($1)
```

- Output in Questasim:

```
# The value in register 1 (address to memory) is:     2
# The value in address     2 in memory is:    45
# The value in register 3 (loaded from memory) is:     x
# finished after                  0 cycles
```

```
#
# The value in register 1 (address to memory) is:     2
# The value in address     2 in memory is:    45
# The value in register 3 (loaded from memory) is:    45
# finished after                    1 cycles
```

General notes:

- .mem files are used for instruction register initialization, and there are no other memories to initialize.
- The assembler is a python script (asm_to_mem.py). It takes the path of the file to translate and the path to the destination, and whether the file should be a mif or a mem file.
- The multiplier stores the the most significant 16-bits in register 14, and the least significant 16-bits in register 13.
- The multiplier stalls the processor for 19 clocks in total.
- Load immediate is converted to or immediate with register 0.
- All of the module designs are in the module directory in their respective packages.