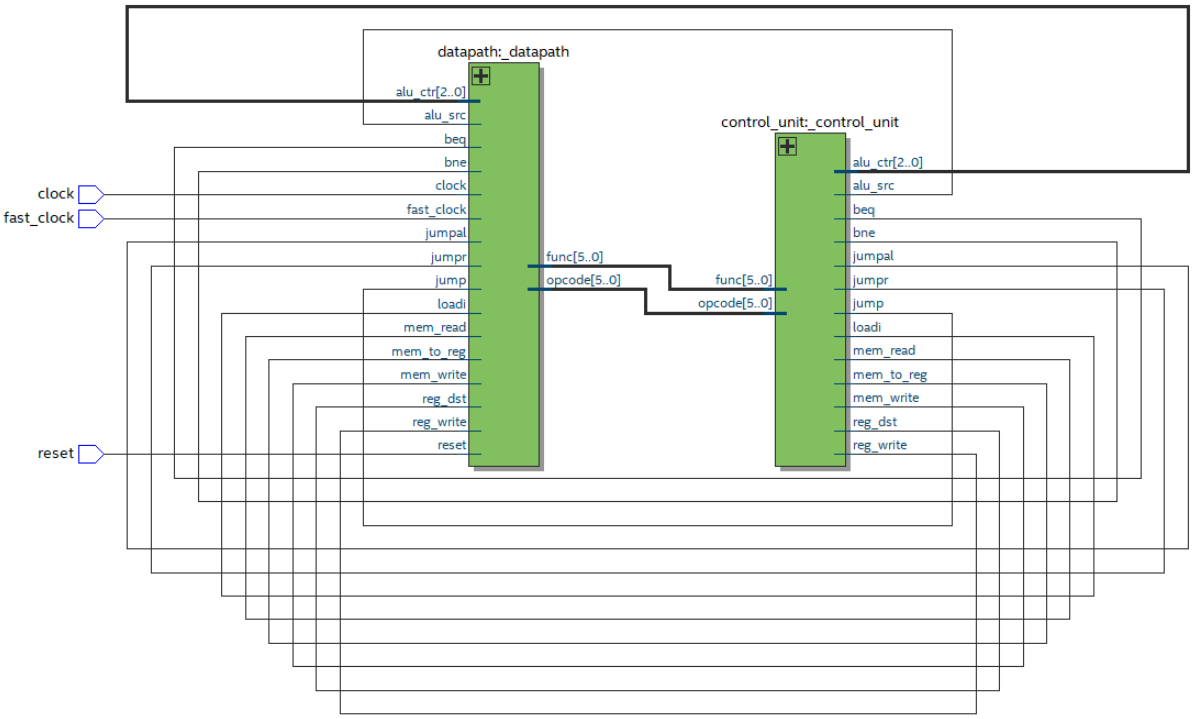
GTU Department of Computer Engineering
CSE 331/503 - Fall 2022
Final Project

Emirkan Burak Yılmaz
1901042659

1. mips_single_cycle_16



The processor consists of control unit and datapath. The control unit produces the required signals that datapath needs. There are general signals like reg_dst reg_write and instructions specific signals like jump, beq. Supported instructions and some instruction examples are given below.

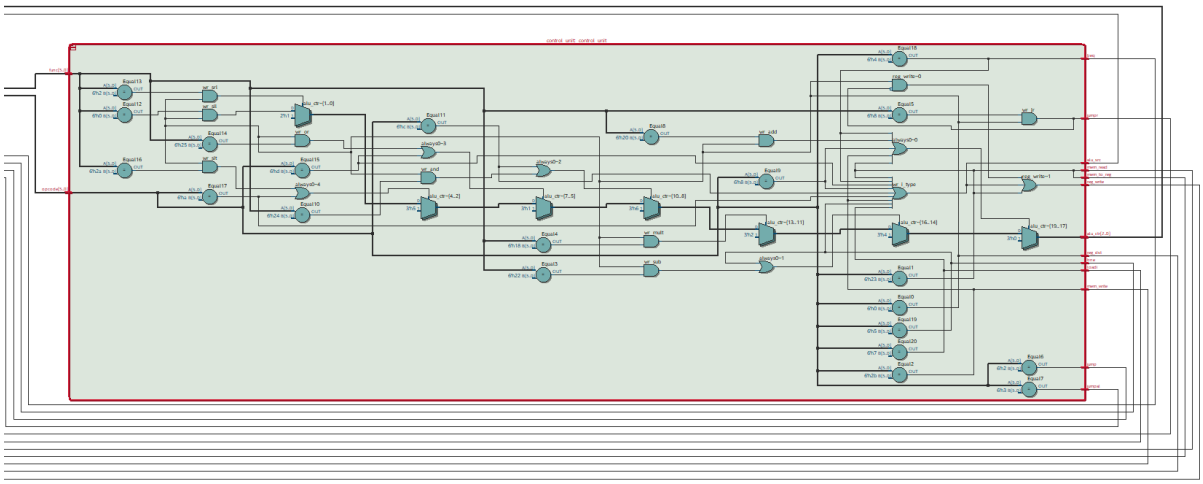
R-type: add, sub, slt, and, or, sll, srl, mult
I-type: addi, lw, sw, li, beq, bne, slti, andi, ori
J-type: j, jr, jal

INSTR R-TYPE	func(hex)	Opcode	rs	rt	rd	shamt	func	--
add $r1, $r2, $r3	20	000000	0010	0011	0001	0000	100000	0000
and $r1, $r4, $r5	24	000000	0100	0101	0001	0000	100100	0000
sub $r1, $r2, $3	22	000000	0010	0011	0001	0000	100010	0000
or $r1, $r4, $r5	25	000000	0100	0101	0001	0000	100101	0000

INSTR I-TYPE	opcode (hex)	opcode	rs	rt	Imm16	--
addi $r1, $r2, 15	08	001000	0010	0001	0000 0000 0000 1111	00
slti $r1, $r6, 26	0a	001010	0110	0001	0000 0000 0001 1010	00
andi $r1, $r5, 1023	0c	001100	0101	0001	0000 0011 1111 1111	00
li $r2, 100	07	000111	0000	0010	0000 0000 0110 0100	00
lw $r1, $r0, 4	23	100011	0000	0001	0000 0000 0000 0100	00
ori $r5, $r1, 0	0d	001101	0001	0101	0000 0000 0000 0000	00

INSTR J-TYPE	opcode (hex)	opcode	addr	--
j 2	02	000010	00 0000 0010	0000 0000 0000 0000

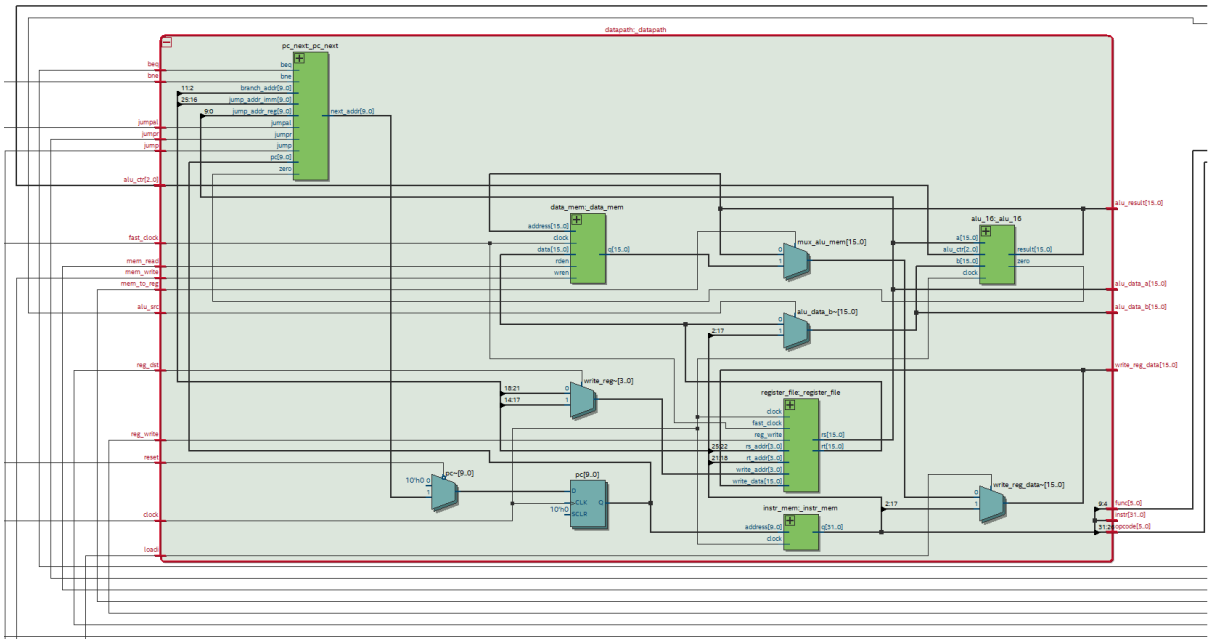
2. control_unit



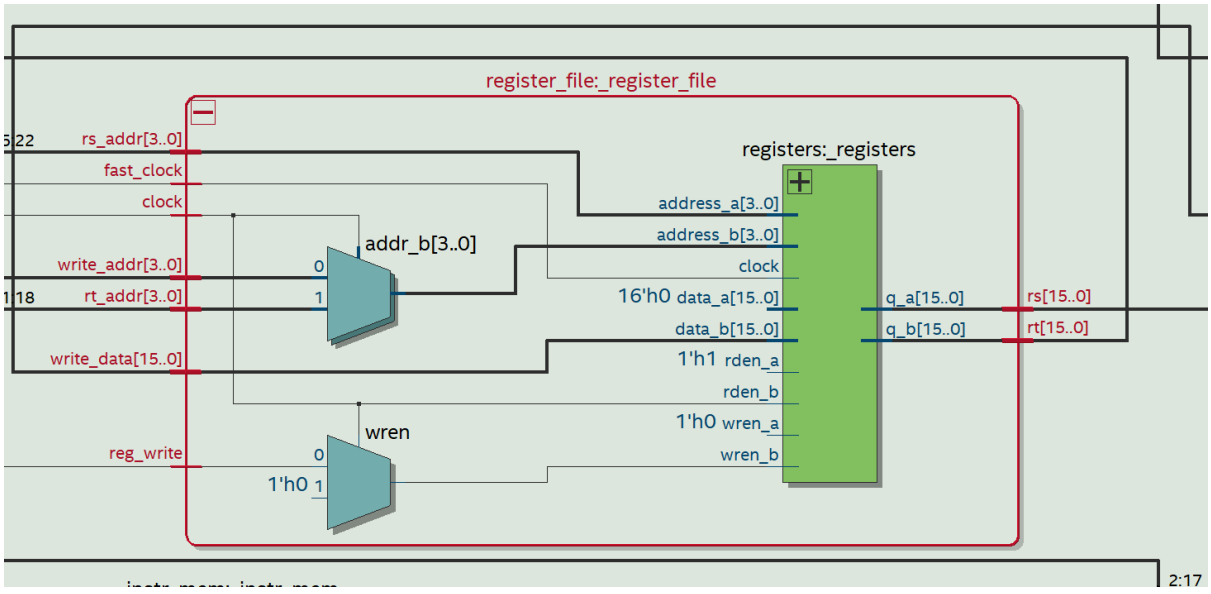
Control unit takes the opcode and the function part of the instruction and produces control signals. The following shows the signals and their meaning.

Signal	Meaning
reg_dst	Selects rt or rd register for write register address
beq	beq instruction
bne	bne instruction
jump	j instruction
jumpr	jr instruction
jumpal	jal instruction
loadi	li instruction
reg_write	Enables register write port
alu_ctr	Selects ALU operation
alu_src	Select immediate or value from rt register
mem_read	Enables data memory read
mem_write	Enables data memory write
mem_to_reg	Select the data comes from memory or ALU result

3. datapath



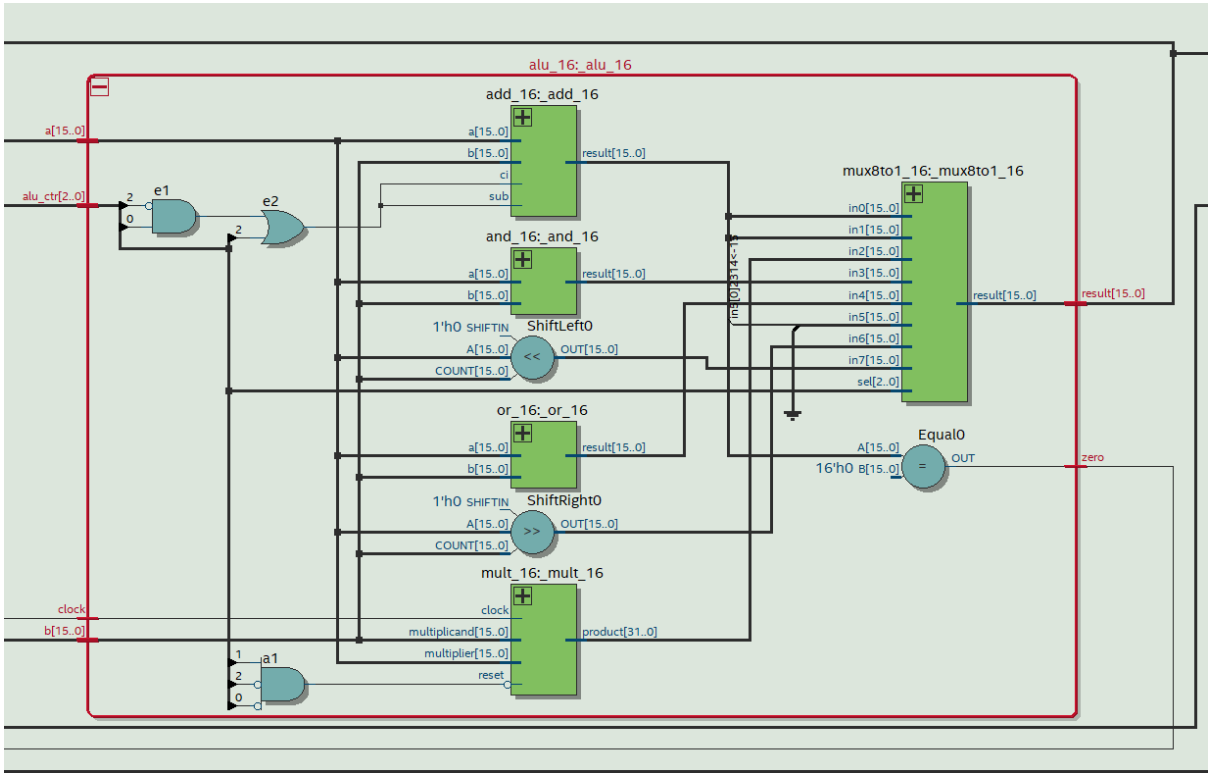
MIPS follows Harvard architecture and uses separate memories for instruction and data. In this design instructions are kept in a ROM which has capacity of 2^{10} instruction (32-bit instruction) and data are kept in a RAM which has capacity of 2^{16} words (16-bit word). For these two types of memory data_mem and instr_mem modules are created with Quartus MegaWizard.



For the registers register_file module created, and it contains registers module which created with Quartus MegaWizard. Register file has 16 registers inside and it has 2 ports. Reading and writing operations are done in half cycle. For accomplish this half cycle execution, fast_clock is used. It has double frequency when we compared to main clock.

Datapath also contains ALU for executing the instructions by doing arithmetic and logical operations.

4. alu_16

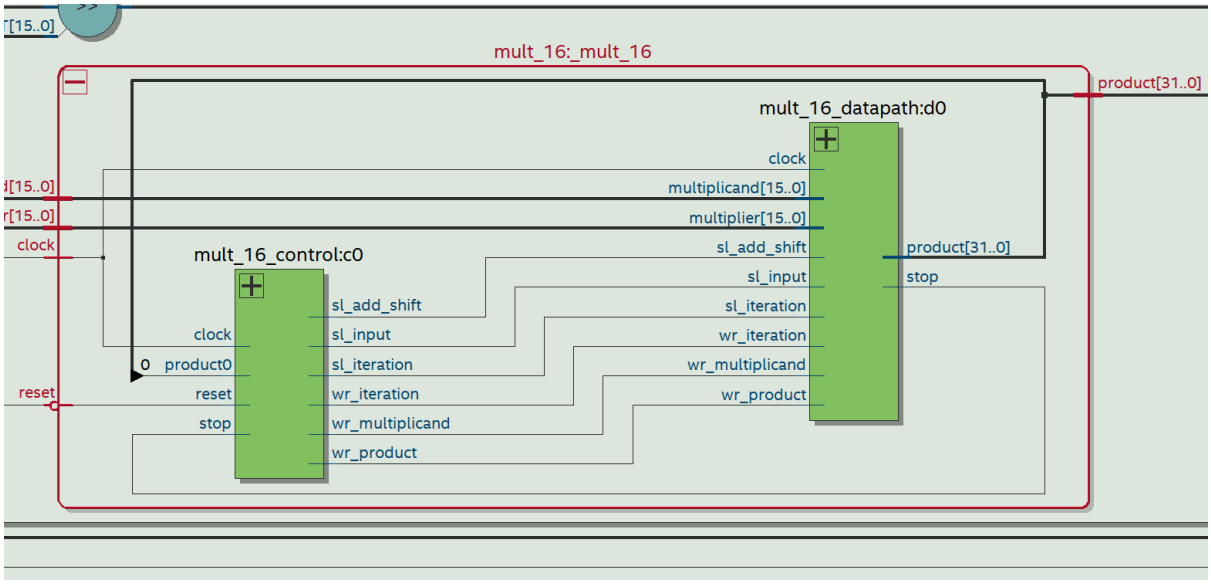


ALU is implemented with structural verilog and it supports 8 different operations. Basically, all the operations are calculated, and the result is selected with control signal alu_ctr and an 8x1 MUX. ALU operations and instruction required operation is given following tables.

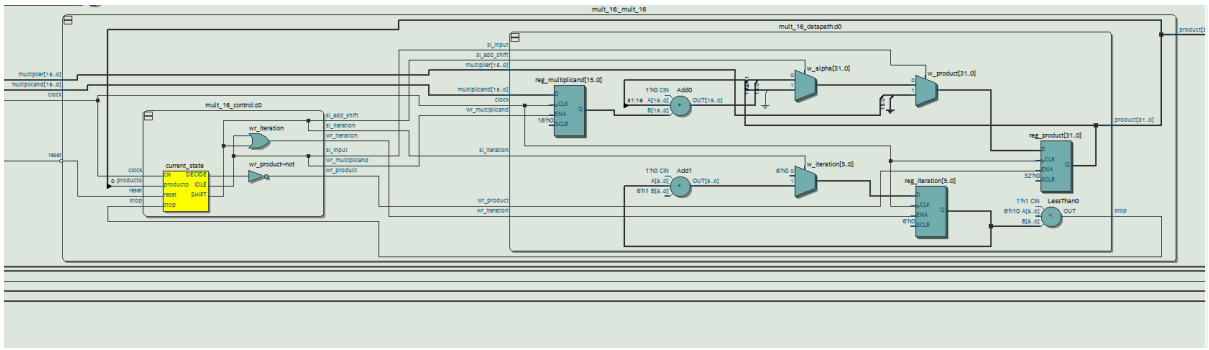
alu_ctr	Operation
000	ADD
001	SUB
010	MULT
011	AND
100	OR
101	SLT
110	SRL
111	SLL

instruction	ALU Operation
add	ADD
sub	SUB
addi	ADD
li	X
lw	ADD
sw	ADD
beq	ADD
bne	SUB
slt	SUB
slti	SUB
j	X
jr	X
jal	X
and	AND
or	OR
andi	AND
ori	OR
sll	SLL
srl	SRL

5. mult_16

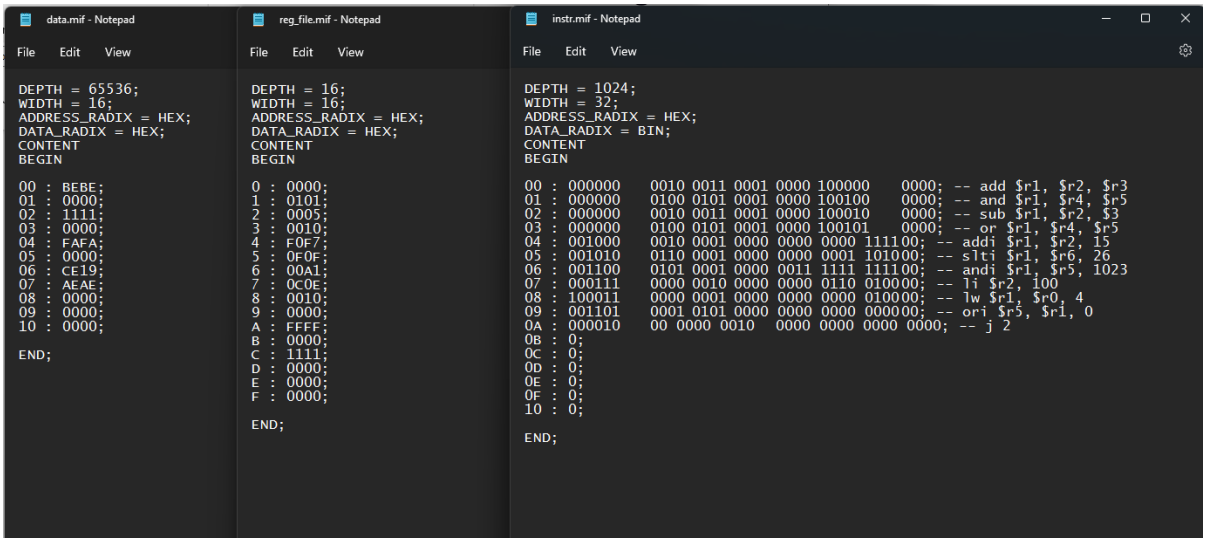


A sequential 16-bit multiplier which consists of control part and datapath part. Control part is basically a FSM and produce signals for the datapath part. According to control signals the multiplication performed and the result become ready after 16 clock cycle.



6. Testbench and Results

mif files for memory initialization



data.mif - Notepad

```
DEPTH = 65536;
WIDTH = 16;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT
BEGIN

00 : BEBE;
01 : 0000;
02 : 1111;
03 : 0000;
04 : FAFA;
05 : 0000;
06 : CE19;
07 : AEAE;
08 : 0000;
09 : 0000;
10 : 0000;

END;
```

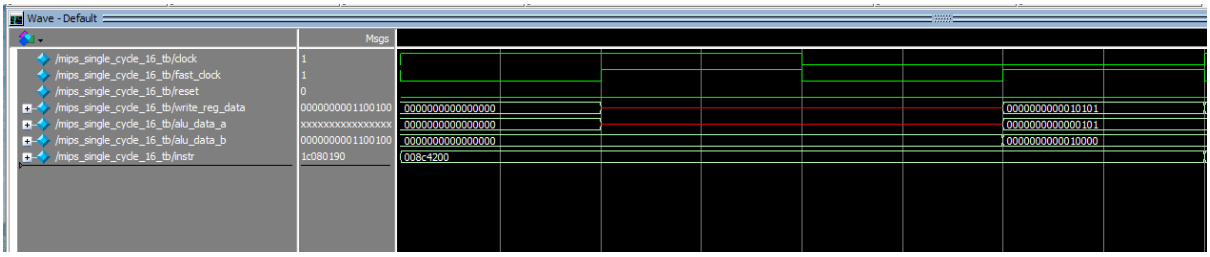
reg_file.mif - Notepad

```
DEPTH = 16;
WIDTH = 16;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT
BEGIN

0 : 0000;
1 : 0101;
2 : 0005;
3 : 0010;
4 : F0F7;
5 : 0F0F;
6 : 00A1;
7 : 0C0E;
8 : 0010;
9 : 0000;
A : FFFF;
B : 0000;
C : 1111;
D : 0000;
E : 0000;
F : 0000;

END;
```

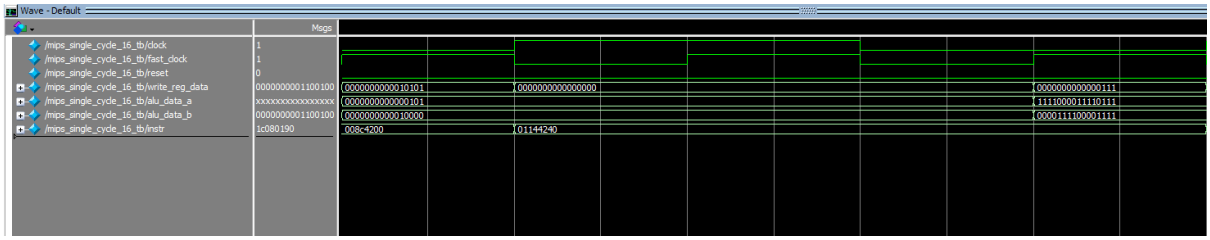
instr.mif - Notepad

```
DEPTH = 1024;
WIDTH = 32;
ADDRESS_RADIX = HEX;
DATA_RADIX = BIN;
CONTENT
BEGIN

00 : 000000    0010 0011 0001 0000 100000    0000; -- add $r1, $r2, $r3
01 : 000000    0100 0101 0001 0000 100100    0000; -- and $r1, $r4, $r5
02 : 000000    0010 0011 0001 0000 100010    0000; -- sub $r1, $r2, $3
03 : 000000    0100 0101 0001 0000 100101    0000; -- or $r1, $r4, $r5
04 : 001000    0010 0001 0000 0000 0000 111100; -- addi $r1, $r2, 15
05 : 001010    0110 0001 0000 0000 0001 101000; -- slti $r1, $r6, 26
06 : 001100    0101 0001 0000 0011 1111 111100; -- andi $r1, $r5, 1023
07 : 000111    0000 0010 0000 0000 0110 010000; -- li $r2, 100
08 : 100011    0000 0001 0000 0000 0000 010000; -- lw $r1, $r0, 4
09 : 001101    0001 0101 0000 0000 0000 000000; -- ori $r5, $r1, 0
0A : 000010    00 0000 0010   0000 0000 0000 0000; -- j 2
0B : 0;
0C : 0;
0D : 0;
0E : 0;
0F : 0;
10 : 0;

END;
```

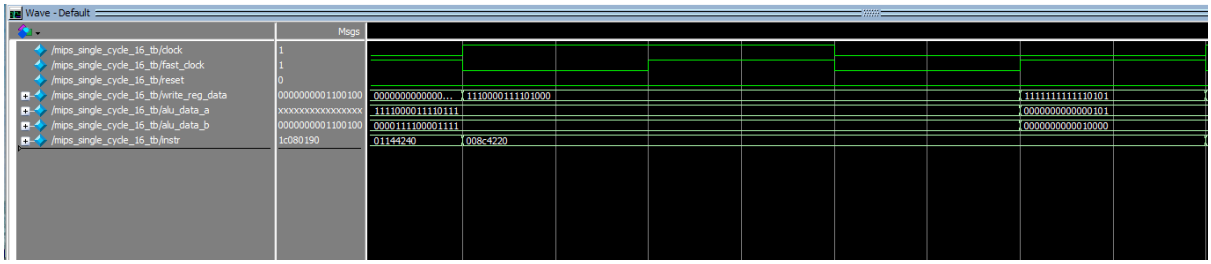
Module: mips_single_cycle_16



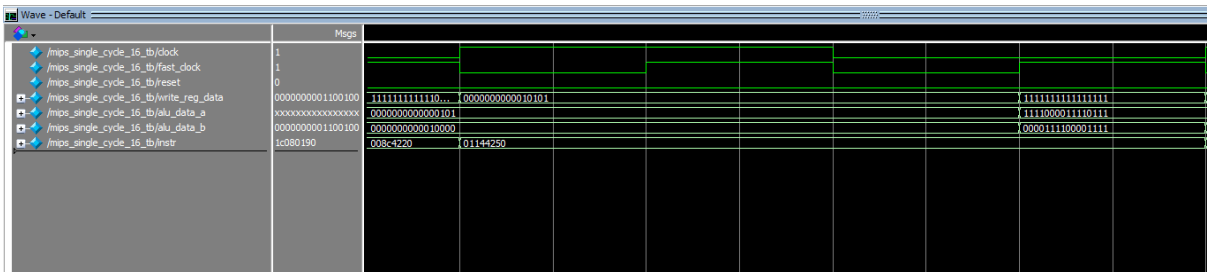
add $r1, $r2, $r3



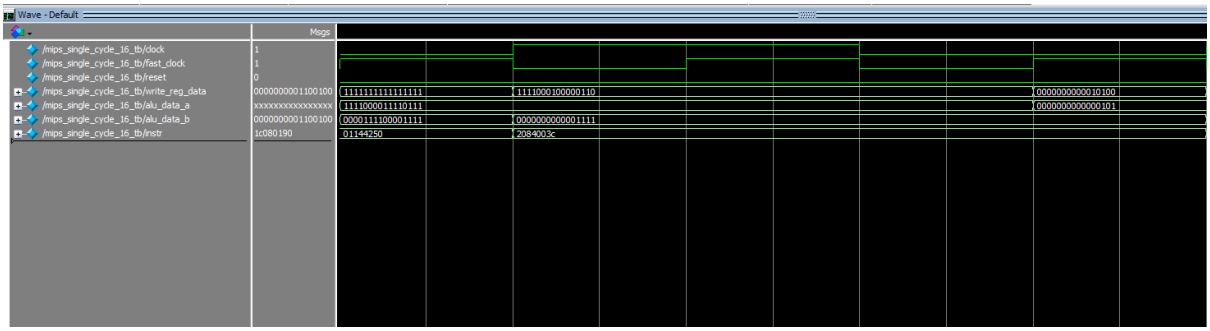
and $r1, $r4, $r5



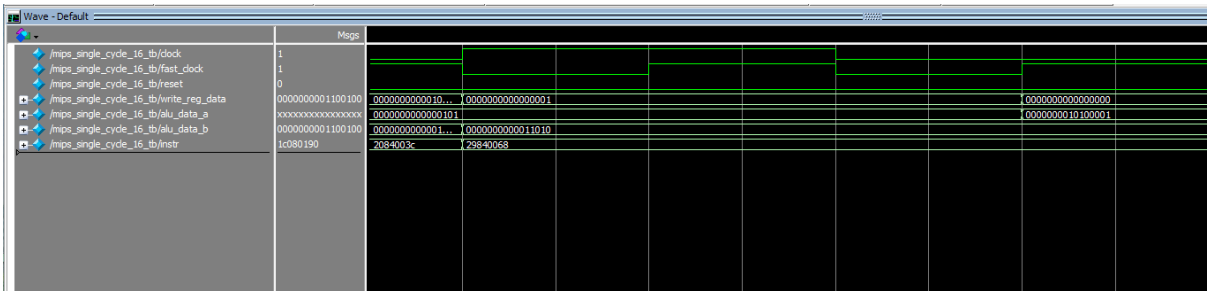
sub $r1, $r2, $3



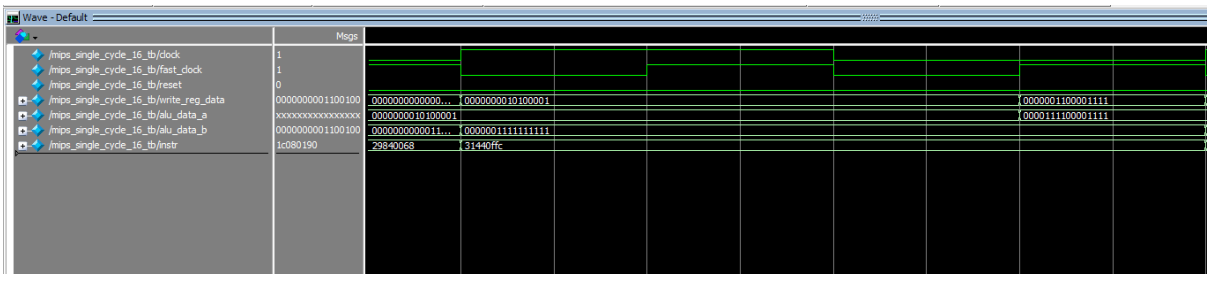
or $r1, $r4, $r5



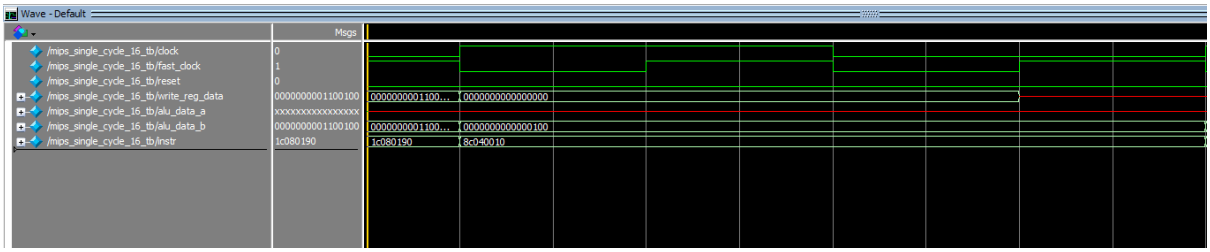
addi $r1, $r2, 15



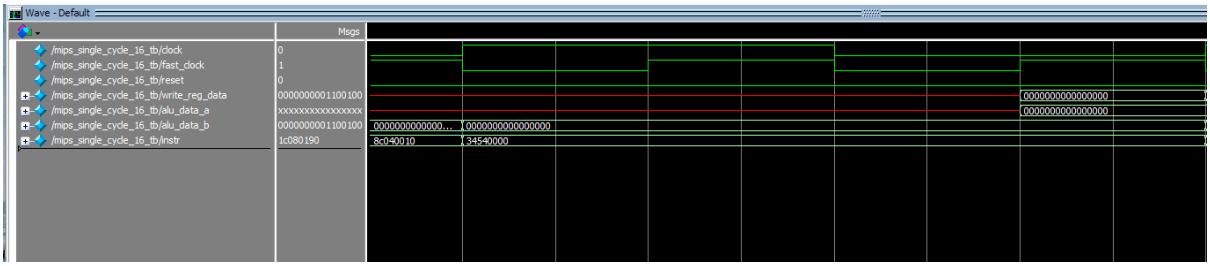
slti $r1, $r6, 26



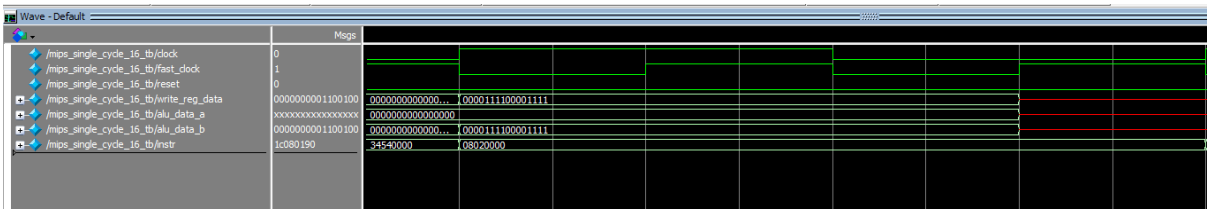
andi $r1, $r5, 1023



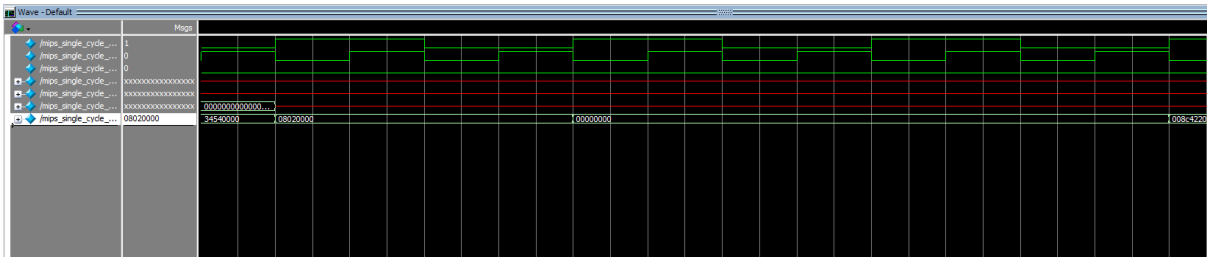
li $r2, 100



lw $r1, $r0, 4 (there is a problem)



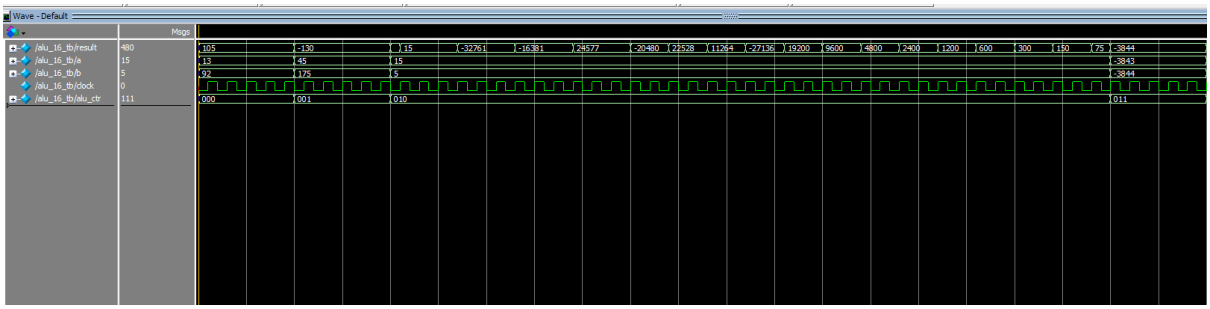
ori $r5, $r1, 0

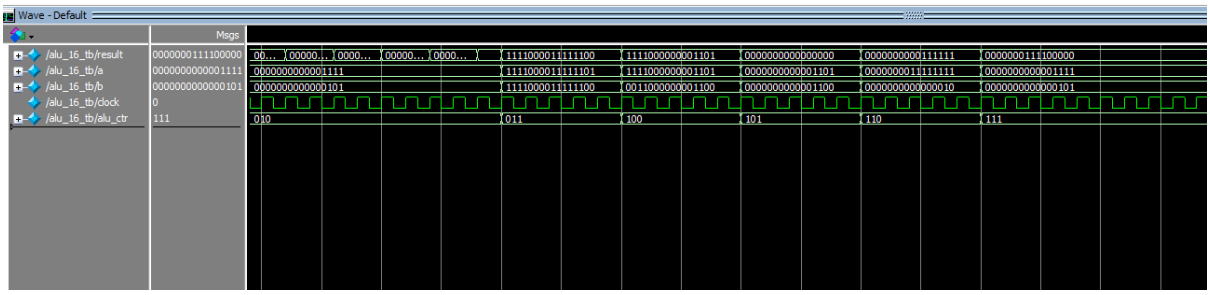


j 2

Module: alu_16

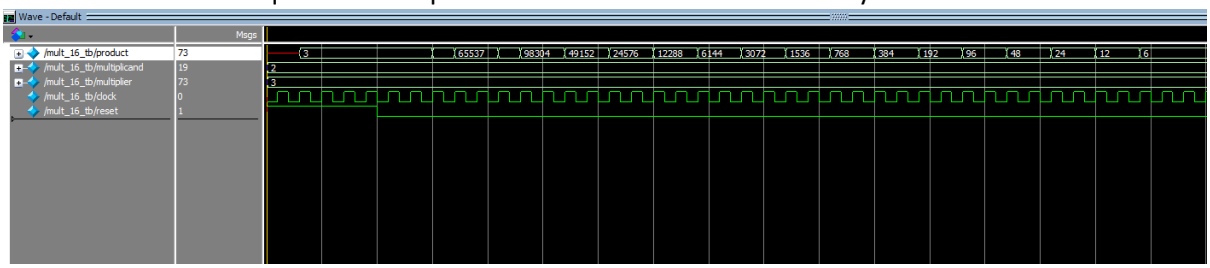
Testbench results are as expected and given below. Result of ADD, SUB and MULT operations are displayed in decimal radix and the remaining operations are displayed in binary radix for better visualization.

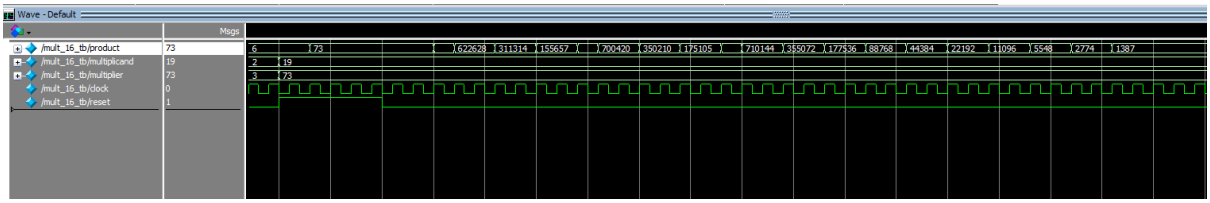




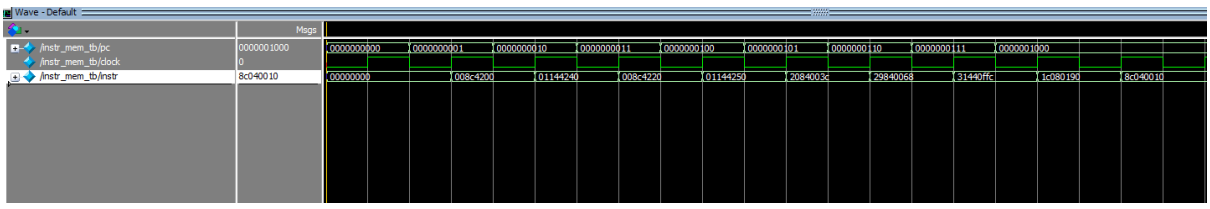
Module: mult_16

Since this is a 16bit sequential multiplier the result found after 16 CPU cycle.





Module: instr_mem



Module: register_file