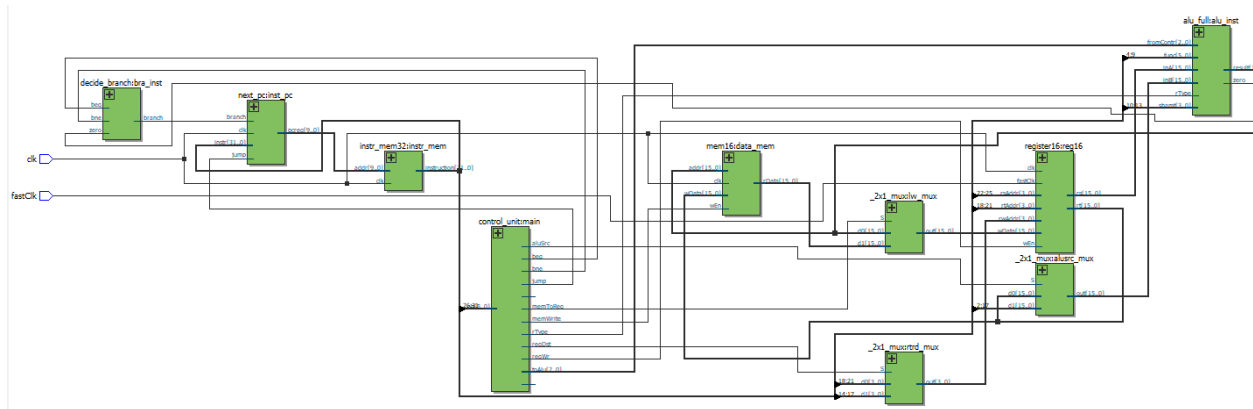
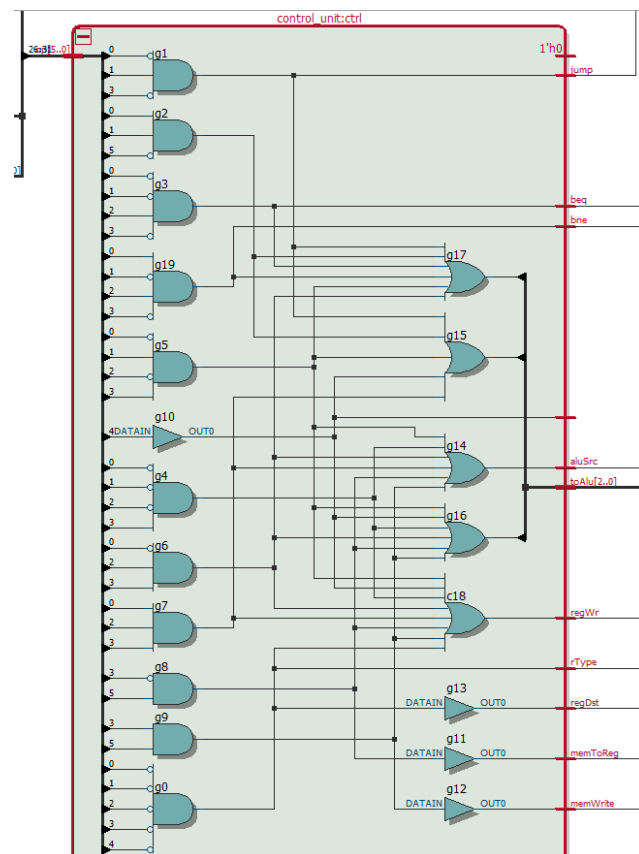


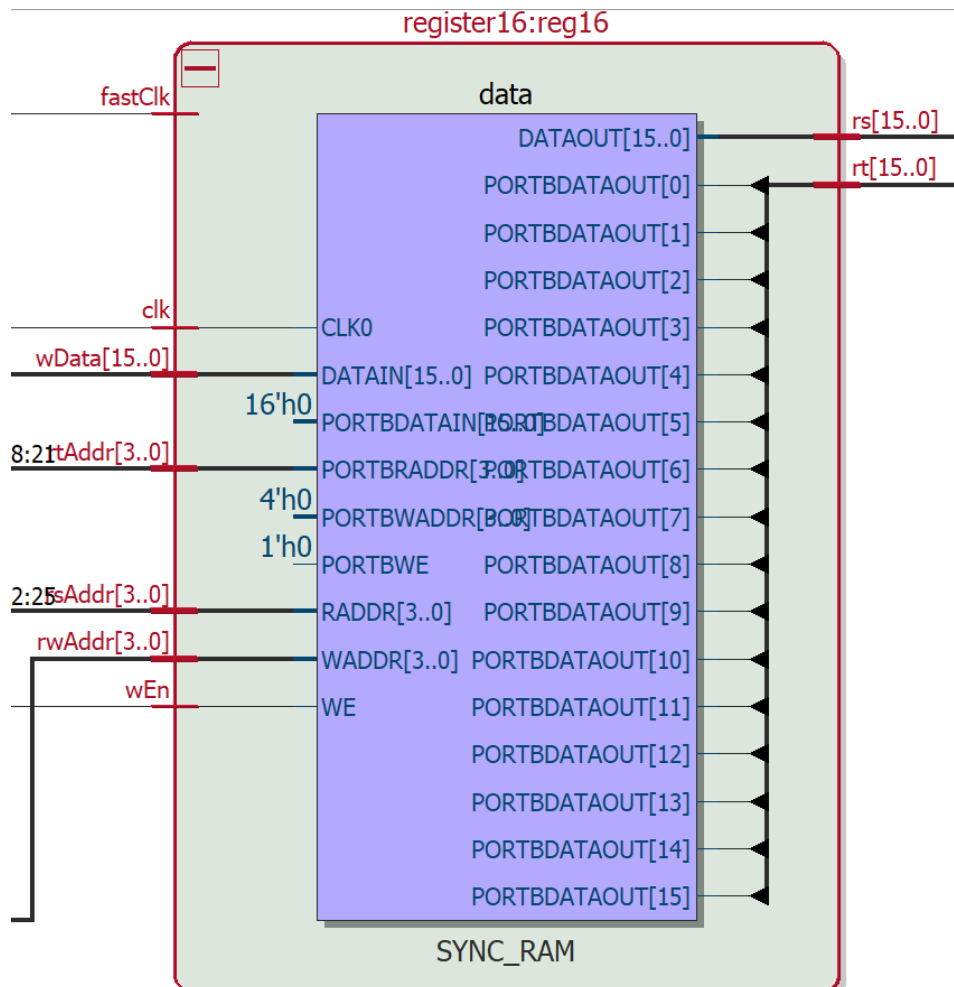
Homework 3 Report



control_unit.v module:

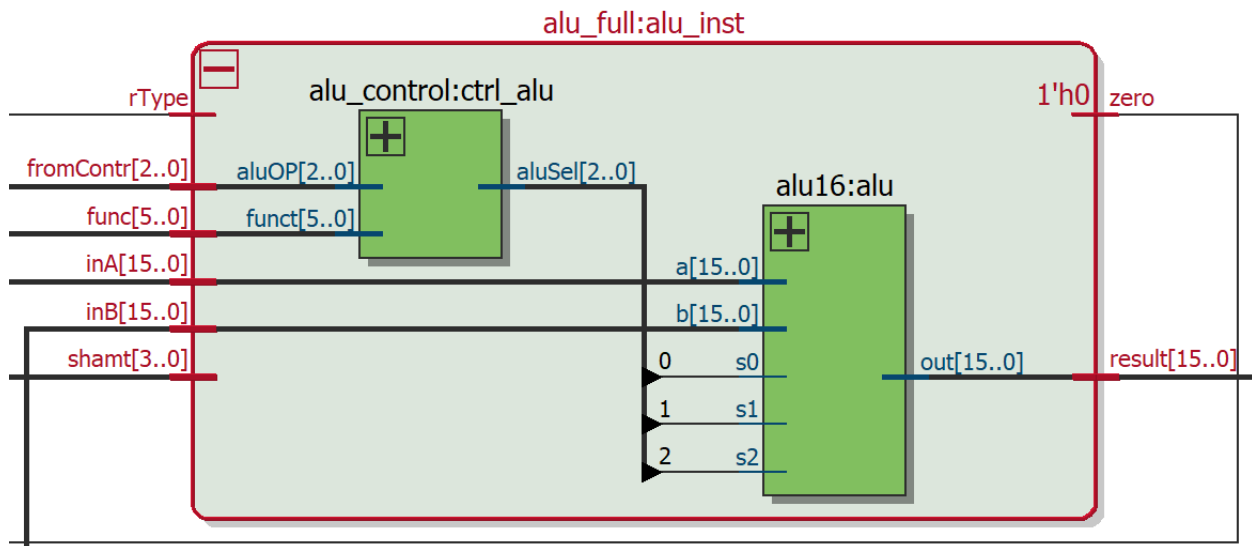


This module consists of a 32-bit adder, subtractor, multiplier, AND, SLT, NOR, OR, XOR modules. These modules are connected to a 32-bit 8x1 mux to give the correct result.



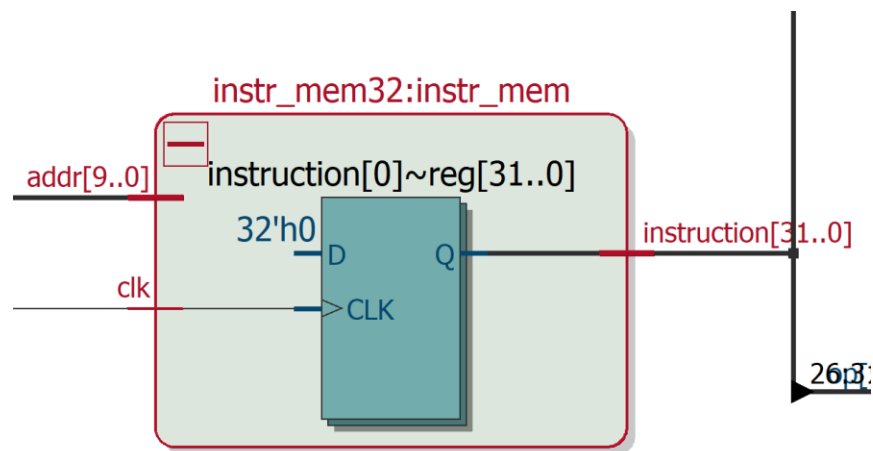
This module consists of a 32-bit adder, subtractor, multiplier, AND, SLT, NOR, OR, XOR modules. These modules are connected to a 32-bit 8x1 mux to give the correct result.

alu_full.v module:



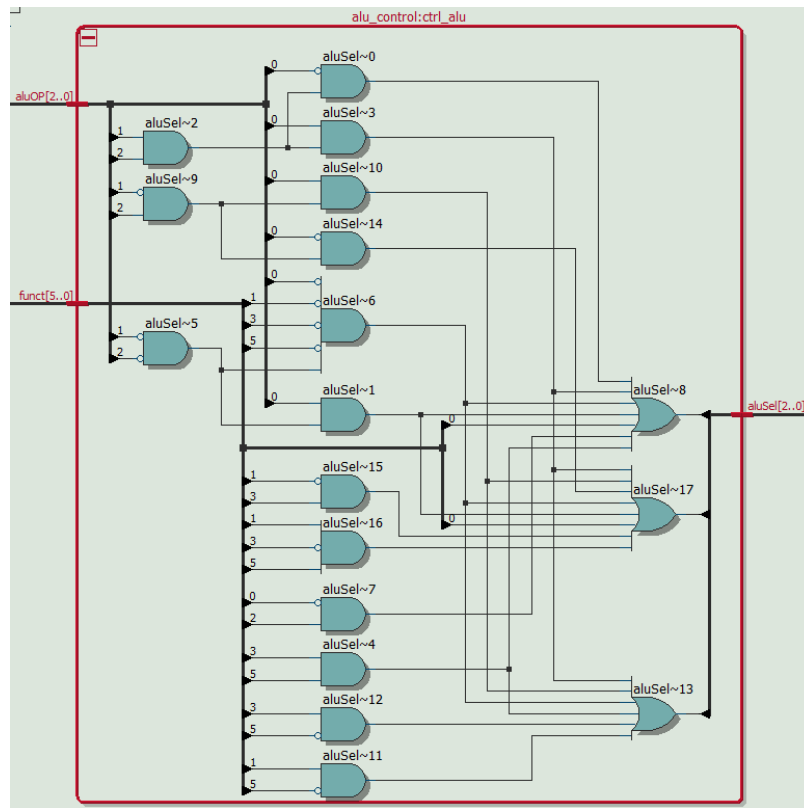
This module consists of a 32-bit adder, subtractor, multiplier, AND, SLT, NOR, OR, XOR modules. These modules are connected to a 32-bit 8x1 mux to give the correct result.

instr_mem32.v module:



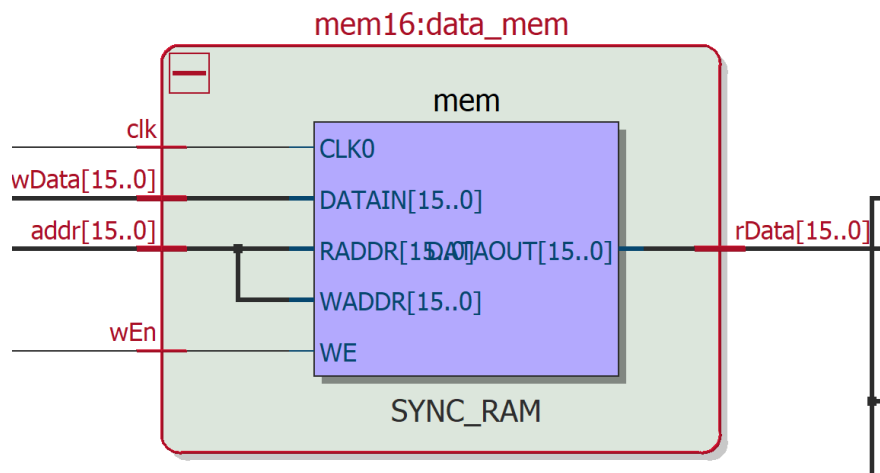
This module takes 10 bit PC address and finds the instruction that corresponds to that address in `instructions.mem` file. It outputs 32 bit instruction.

alu_control.v module:



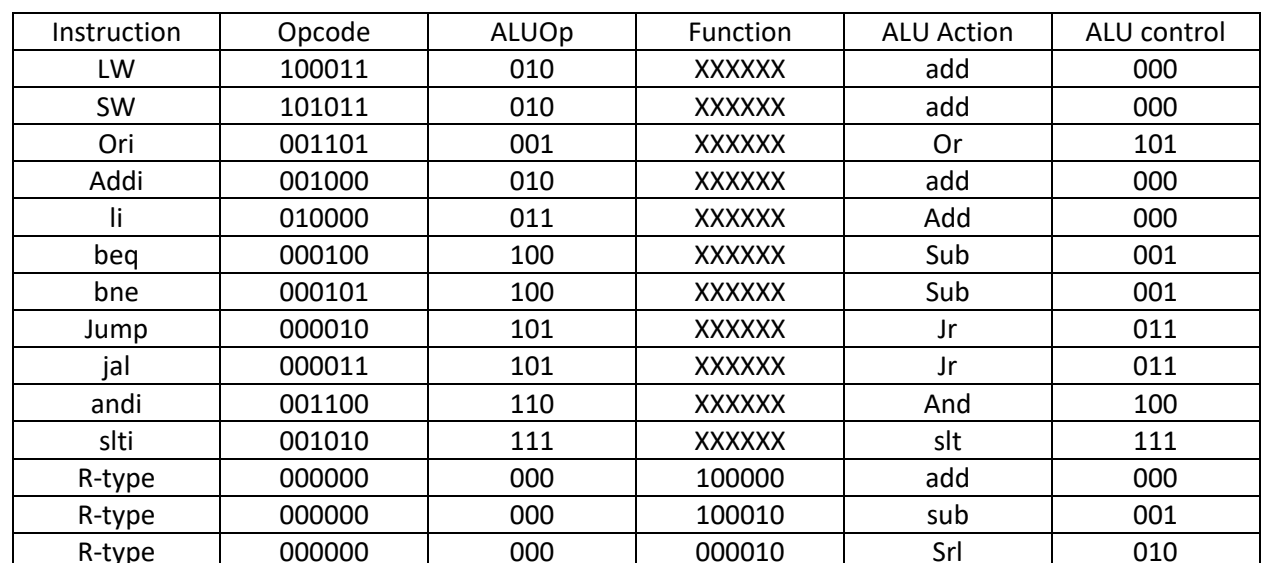
This module consists of a 32-bit adder, subtractor, multiplier, AND, SLT, NOR, OR, XOR modules. These modules are connected to a 32-bit 8x1 mux to give the correct result.

mem16.v module:



This module consists of a 32-bit adder, subtractor, multiplier, AND, SLT, NOR, OR, XOR modules. These modules are connected to a 32-bit 8x1 mux to give the correct result.

This module increases PC by 4 every time except branch or jump. If branch is 1, it will calculate the branchAddr and take the instruction at that address, else if jump is 1, it will take the jump address and jump to it directly.



R-type	000000	000	001000	jr	011
R-type	000000	000	100100	And	100
R-type	000000	000	100101	Or	101
R-type	000000	000	101010	Slt	110
R-type	000000	000	000000	sll	111

The Test Results:

control_unit_testbench results:

```
# time = 0, op=000000,memWrite=0,aluSrc=0,toAlu=000,rType=1,regWr=1,regDst=1,beq=0,bne=0,jump=0,li=0
# time = 20, op=001101,memWrite=0,aluSrc=1,toAlu=001,rType=0,regWr=1,regDst=0,beq=0,bne=0,jump=0,li=0
# time = 40, op=001000,memWrite=0,aluSrc=1,toAlu=010,rType=0,regWr=1,regDst=0,beq=0,bne=0,jump=0,li=0
# time = 60, op=100011,memWrite=0,aluSrc=1,toAlu=010,rType=0,regWr=1,regDst=0,beq=0,bne=0,jump=0,li=0
# time = 80, op=101011,memWrite=1,aluSrc=1,toAlu=010,rType=0,regWr=1,regDst=0,beq=0,bne=0,jump=0,li=0
# time = 100, op=010000,memWrite=0,aluSrc=0,toAlu=011,rType=0,regWr=1,regDst=0,beq=0,bne=0,jump=0,li=1
# time = 120, op=000100,memWrite=0,aluSrc=0,toAlu=100,rType=0,regWr=0,regDst=0,beq=1,bne=0,jump=0,li=0
# time = 140, op=000101,memWrite=0,aluSrc=0,toAlu=100,rType=0,regWr=0,regDst=0,beq=0,bne=1,jump=0,li=0
# time = 160, op=000010,memWrite=0,aluSrc=0,toAlu=101,rType=0,regWr=0,regDst=0,beq=0,bne=0,jump=1,li=0
# time = 180, op=000011,memWrite=0,aluSrc=0,toAlu=101,rType=0,regWr=0,regDst=0,beq=0,bne=0,jump=0,li=0
# time = 200, op=001100,memWrite=0,aluSrc=1,toAlu=110,rType=0,regWr=1,regDst=0,beq=0,bne=0,jump=0,li=0
# time = 220, op=001010,memWrite=0,aluSrc=1,toAlu=111,rType=0,regWr=1,regDst=0,beq=0,bne=0,jump=0,li=0
# Break in Module control_unit_testbench at C:/altera/13.1/mips_single_cycle/control_unit_testbench.v line 65
```

The control unit takes only an OPCode and outputs the correct control signals depending on it. For example 000000 opcode means it is R Type, so the control signals that R Types need will be made 1. Also, it has a toAlu signal, which tells the ALU about the operator that will be made.

Alu_control_testbench results:

```
# Loading work.alu_control_testbench
VSIM 32> step -over -current
# time = 0, aluOP=000, funct=100000, aluSel=000
# time = 20, aluOP=000, funct=100010, aluSel=001
# time = 40, aluOP=000, funct=100100, aluSel=100
# time = 60, aluOP=001, funct=000000, aluSel=101
# time = 80, aluOP=010, funct=000000, aluSel=000
# time = 100, aluOP=011, funct=000000, aluSel=000
# time = 120, aluOP=100, funct=000000, aluSel=001
# time = 140, aluOP=101, funct=000000, aluSel=011
# time = 160, aluOP=110, funct=000000, aluSel=100
# time = 180, aluOP=111, funct=000000, aluSel=111
```

The alu control unit takes funct and aluOP, and arranges the aluSel depending on them. If aluOP is 000, this means that it is R Type and we will check the funct field. Otherwise, we will choose depending on aluOP.

register16_testbench results:

```

V$IM 56> step -over -current
# time = 0, rs =32767, rt= 7
# time = 20, rs = 1, rt= 7
# Break in Module register16_testbench at C:/altera/13.1/mips_single_cycle/register16_testbench.v line 32

```

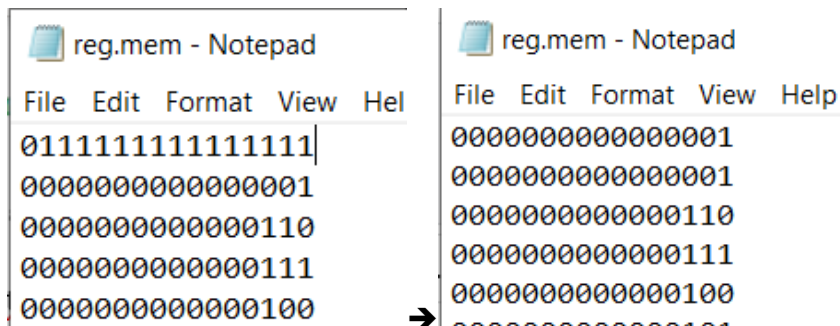
Firstly, I am giving addresses to rs and rd (000 and 001), and read the current data. After that, I enable writing, and I set write data and the write address. In final, I write the result of the test to reg_out.mem file with writememb.

```

rsAddr=3'd0; rtAddr=3'd1;
#`DELAY;
wEn=1'b1; wData=16'b0000000000000001; rwAddr=3'd0;
#`DELAY;
$writememb("C:/altera/13.1/mips_single_cycle/reg_out.mem",test.data);
#`DELAY;
$stop;

```

These are before and after of the register data. (First line)



Alu_full_testbench results:

```


V$IM 71> step -over -current
# Results: time= 0, a= 5bf3, b= 0011, result= 5c04, fromContr= 000, func= 100000, rType= 1, shamt= 0000, zero= 0
# Results: time= 20, a= 0000, b= 5678, result= 5678, fromContr= 000, func= 100000, rType= 1, shamt= 0000, zero= 0
# Results: time= 40, a= 5bf3, b= 0011, result= 5c04, fromContr= 010, func= 000000, rType= 0, shamt= 0000, zero= 0
# Results: time= 60, a= 0000, b= 5678, result= 5678, fromContr= 010, func= 000000, rType= 0, shamt= 0000, zero= 0
# Results: time= 80, a= af23, b= 0023, result= 0001, fromContr= 000, func= 101010, rType= 1, shamt= 0000, zero= 0
# Results: time= 100, a= ffff, b= ffff, result= 0000, fromContr= 000, func= 101010, rType= 1, shamt= 0000, zero= 0
# Results: time= 120, a= af23, b= 0022, result= af01, fromContr= 100, func= 000000, rType= 0, shamt= 0000, zero= 0
# Results: time= 140, a= ffff, b= ffff, result= 0000, fromContr= 100, func= 000000, rType= 0, shamt= 0000, zero= 1
# Results: time= 160, a= 0001, b= 0002, result= 0001, fromContr= 000, func= 001010, rType= 1, shamt= 0001, zero= 0
# Results: time= 180, a= 0002, b= 0007, result= 0001, fromContr= 000, func= 001010, rType= 1, shamt= 0010, zero= 0
# Results: time= 200, a= 5bf3, b= 0011, result= 5c04, fromContr= 000, func= 100000, rType= 1, shamt= 0000, zero= 0
# Results: time= 220, a= 0000, b= 5678, result= 5678, fromContr= 000, func= 100000, rType= 1, shamt= 0000, zero= 0
# Results: time= 240, a= 0000, b= 1234, result= 1234, fromContr= 000, func= 000100, rType= 1, shamt= 0000, zero= 0
# Results: time= 260, a= 2abf, b= 2abf, result= 2abf, fromContr= 000, func= 000100, rType= 1, shamt= 0000, zero= 0
# Results: time= 280, a= 0000, b= 1234, result= 1234, fromContr= 000, func= 000101, rType= 1, shamt= 0000, zero= 0
# Results: time= 300, a= 2abf, b= 2abf, result= 2abf, fromContr= 000, func= 000101, rType= 1, shamt= 0000, zero= 0
# Results: time= 320, a= 3bfc, b= 4212, result= 4212, fromContr= 000, func= 010010, rType= 1, shamt= 0000, zero= 0
# Results: time= 340, a= 000f, b= 0001, result= 0001, fromContr= 000, func= 010010, rType= 1, shamt= 0000, zero= 0
# Results: time= 360, a= 0001, b= 0002, result= 0004, fromContr= 000, func= 000000, rType= 1, shamt= 0001, zero= 0
# Results: time= 380, a= 0002, b= 0007, result= 001c, fromContr= 000, func= 000000, rType= 1, shamt= 0010, zero= 0

```

Alu_full takes 3 bits from control and 6 bits from the instruction (funct field), it also takes rType and shamt for some instructions. If the fromContr is 000, this means it is an RType so, the alu_control will output a aluSel value depending on funct field. If fromContr is anything else, it is either I or J type, so the alu_control will determine the proper aluSel to perform the necessary operation.

Instr_mem32_testbench results:

```
# Loading work.instr_mem32_testbench
# Loading work.instr_mem32
VSIM 83> step -over -current
# time = 0, addr =0000000000, instruction=00000000000001001000001000000000
# time = 40, addr =0000000001, instruction=00000000000001001100001000100000
# time = 80, addr =0000000010, instruction=00000000000000000000000000000000
```

 instructions.mem - Notepad

File Edit Format View Help


```
00000000000001001000001000000000
00000000000001001100001000100000
00000000000000000000000000000000
00000000000000000000000000000000
```

It takes an address input and read the corresponding address from the instructions memory.

Mips_single_cycle_testbench results:


```
# time = 100,instruction=00000000010010001100001000000000, rsAddr=0001, rtAddr=0010, rwAddr=0011
# rs= 1, rt= 6, aluResult= 7 and rw= 7
```

This part is tricky so I commented it, I can show and explain how it works in the demo. The given instruction is 000000_0001_0010_0011_0000_100000_0000. This translates to => add \$3, \$1 \$2.

 reg.mem - Notepad

File Edit Format View I

```
// memory data file
// instance=/mips_s
// format=bin addre
0111111111111111
0000000000000001
0000000000000110
0000000000000000
```

 reg.mem - Notepad

File Edit Format View I

```
// memory data file
// instance=/mips_s
// format=bin addre
0111111111111111
0000000000000001
0000000000000110
→ 0000000000000111
```

It reads the register \$1 and \$2 and add them (op=000000, func=100000), then writes it in \$3.