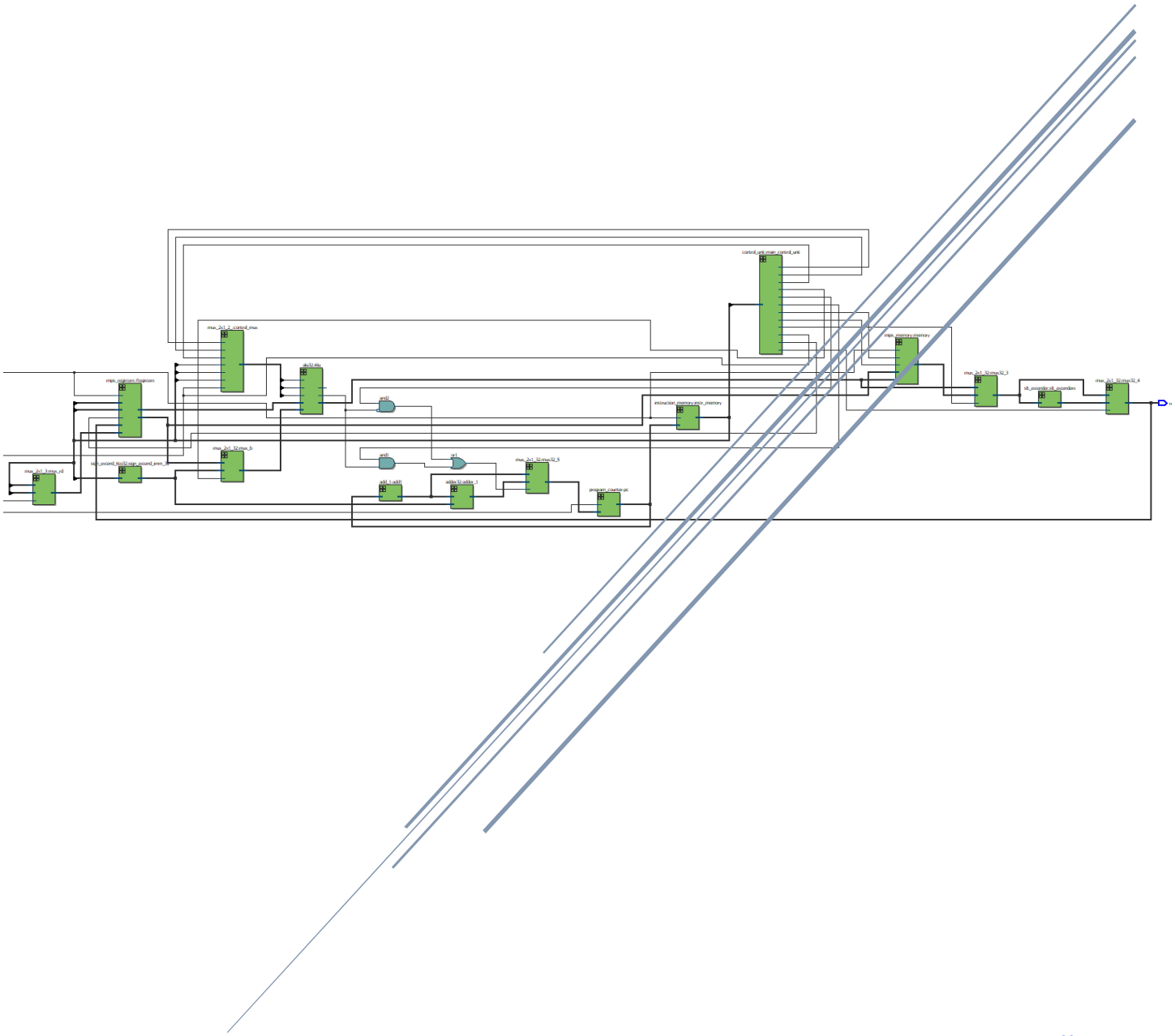
FINAL PROJECT DOCUMENTATION

Single Cycle Processor Design



GTÜ

CSE-331

MiniMIPS_testbench() file

```
# first instruction is r1 = r1 & r3
# instruction = 0000001011001000
# mips.Registers.registers[001] =       01100110010011000011100111100010
#  mips.Registers.registers[011] =      01100000000000000000010001001110
# mips.Register.registers[001] =        01100000000000000000000001000010
#
# second instruction is r6 = r3 & r7
# instruction = 0000011111110000
# mips.Registers.registers[011] =       01100000000000000000010001001110
#  mips.Registers.registers[111] =      11111111111111111111111111111111
# mips.Register.registers[110] =        01100000000000000000010001001110
#
# 3. instruction is r1 = r1 add r3
# instruction = 0000001011001001
# mips.Registers.registers[001] =       01100000000000000000000001000010
#  mips.Registers.registers[011] =      01100000000000000000010001001110
# mips.Register.registers[001] =        11000000000000000000010010010000
#
# 4. instruction is r6 = r3 add r7
# instruction = 0000011111110001
# mips.Registers.registers[011] =       01100000000000000000010001001110
#  mips.Registers.registers[111] =      11111111111111111111111111111111
# mips.Register.registers[110] =        01100000000000000000010001001101
#
# 5. instruction is r1 = r1 sub r3
# instruction = 0000001011001010
# mips.Registers.registers[001] =       11000000000000000000010010010000
#  mips.Registers.registers[011] =      01100000000000000000010001001110
# mips.Register.registers[001] =        01100000000000000000000001000010
#
# 6. instruction is r6 = r3 sub r7
# instruction = 0000011111110010
# mips.Registers.registers[011] =       01100000000000000000010001001110
#  mips.Registers.registers[111] =      11111111111111111111111111111111
# mips.Register.registers[110] =        01100000000000000000010001001111
#
```

```
7. instruction is r1 = r1 xor r3
instruction = 0000001011001011
mips.Registers.registers[001] =       01100000000000000000000001000010
 mips.Registers.registers[011] =      01100000000000000000010001001110
mips.Register.registers[001] =        00000000000000000000010000001100

8. instruction is r6 = r3 xor r7
instruction = 0000011111110011
mips.Registers.registers[011] =       01100000000000000000010001001110
 mips.Registers.registers[111] =      11111111111111111111111111111111
mips.Register.registers[110] =        10011111111111111111101110110001

9. instruction is r1 = r1 nor r3
instruction = 0000001011001100
mips.Registers.registers[001] =       00000000000000000000010000001100
 mips.Registers.registers[011] =      01100000000000000000010001001110
mips.Register.registers[001] =        10011111111111111111101110110001

10. instruction is r6 = r3 nor r7
instruction = 0000011111110100
mips.Registers.registers[011] =       01100000000000000000010001001110
 mips.Registers.registers[111] =      11111111111111111111111111111111
mips.Register.registers[110] =        00000000000000000000000000000000

11. instruction is r1 = r1 or r3
instruction = 0000001011001101
mips.Registers.registers[001] =       10011111111111111111101110110001
 mips.Registers.registers[011] =      01100000000000000000010001001110
mips.Register.registers[001] =        11111111111111111111111111111111

12. instruction is r6 = r3 or r7
instruction = 0000011111110101
mips.Registers.registers[011] =       01100000000000000000010001001110
 mips.Registers.registers[111] =      11111111111111111111111111111111
mips.Register.registers[110] =        11111111111111111111111111111111
```

```
# 13. instruction is r3 = r1 addi 010101
# instruction = 0001001011010101
# mips.Registers.registers[001] =       11111111111111111111111111111111
#  mips.imm_extended =                  00000000000000000000000000010101
# mips.Register.registers[011] =        00000000000000000000000000010100
#
# 14. instruction is r7 = r3 addi 010101
# instruction = 0001011111010101
# mips.Registers.registers[011] =       00000000000000000000000000010100
#  mips.imm_extended =                  00000000000000000000000000010101
# mips.Register.registers[111] =        00000000000000000000000000101001
#
# 15. instruction is r3 = r1 andi 010101
# instruction = 0010001011010101
# mips.Registers.registers[001] =       11111111111111111111111111111111
#  mips.imm_extended =                  00000000000000000000000000010101
# mips.Register.registers[011] =        00000000000000000000000000010101
#
# 16. instruction is r7 = r3 andi 010101
# instruction = 0010011111010101
# mips.Registers.registers[011] =       00000000000000000000000000010101
#  mips.imm_extended =                  00000000000000000000000000010101
# mips.Register.registers[111] =        00000000000000000000000000010101
#
# 17. instruction is r3 = r1 ori 010101
# instruction = 0011001011010101
# mips.Registers.registers[001] =       11111111111111111111111111111111
#  mips.imm_extended =                  00000000000000000000000000010101
# mips.Register.registers[011] =        11111111111111111111111111111111
#
# 18. instruction is r7 = r3 ori 010101
# instruction = 0011011111010101
# mips.Registers.registers[011] =       11111111111111111111111111111111
#  mips.imm_extended =                  00000000000000000000000000010101
# mips.Register.registers[111] =        11111111111111111111111111111111
#
# 19. instruction is r3 = r1 nori 010101
# instruction = 0100001011010101
# mips.Registers.registers[001] =       11111111111111111111111111111111
#  mips.imm_extended =                  00000000000000000000000000010101
# mips.Register.registers[011] =        00000000000000000000000000000000

# 20. instruction is r7 = r3 nori 010101
# instruction = 0100011111010101
# mips.Registers.registers[011] =       00000000000000000000000000000000
#  mips.imm_extended =                  00000000000000000000000000010101
# mips.Register.registers[111] =        11111111111111111111111111101010
#
```

```
# 21. instruction is r3 = r1 < 010101
# instruction = 0111001011010101
# mips.Registers.registers[001] =       11111111111111111111111111111111
#  mips.imm_extended =                  00000000000000000000000000010101
# mips.Register.registers[011] =        00000000000000000000000000000001
#
# 22. instruction is r7 = r3 < 010101
# instruction = 0111011111010101
# mips.Registers.registers[011] =       00000000000000000000000000000001
#  mips.imm_extended =                  00000000000000000000000000010101
# mips.Register.registers[111] =        00000000000000000000000000000001
#
```

```
# 23. instruction is $r3 = M[$r1 + 010101]
# instruction = 1000001011010101
# mips.Registers.registers[001] =       11111111111111111111111111111111
#  mips.imm_extended =                  00000000000000000000000000010101
#  $r1 + 010101 =                       00000000000000000000000000010100
# mips.Register.registers[011] =        00000000000000000000000000011111
#
# 24. instruction is $r7 =  M[$r3 + 010101]
# instruction = 1000011111010101
# mips.Registers.registers[011] =       00000000000000000000000000011111
#  mips.imm_extended =                  00000000000000000000000000010101
#  $r3 + 010101 =                       00000000000000000000000000110100
# mips.Register.registers[111] =        00000000000000000000000000010100
#
```

20 =

52 =

data_memory.txt

```
19  00000000000000000000000000010010
20  00000000000000000000000000010000
21  00000000000000000000000000011111
22  00000000000000000000000000111111
23  00000000000000000000000001111111
```

```
50  00000000000000000000000000010001
51  00000000000000000000000000010010
52  00000000000000000000000000010011
53  00000000000000000000000000010100
54  00000000000000000000000000010101
```

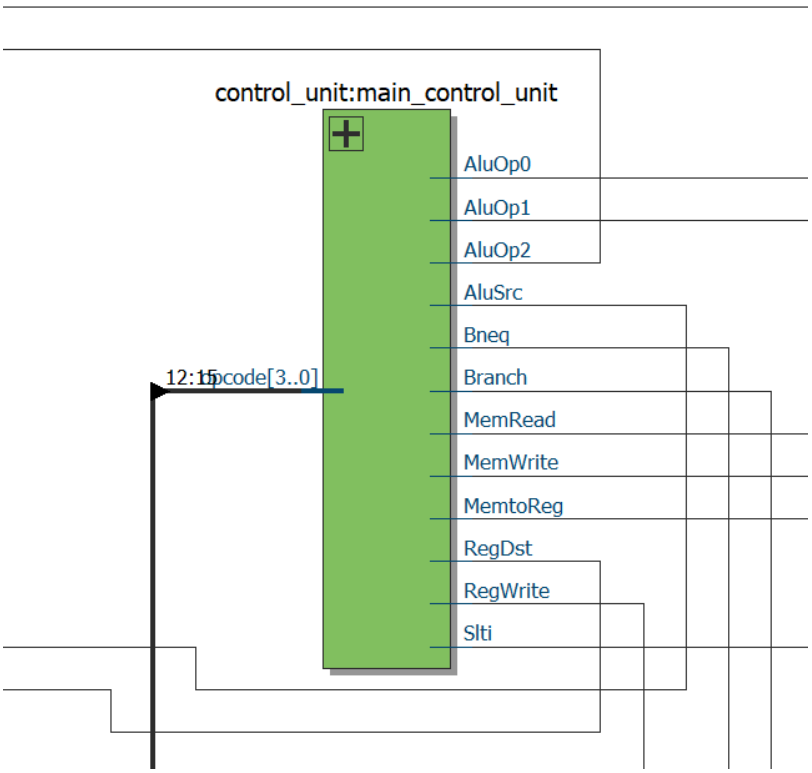
```
27. instruction is  M[$r1 + 010101] = $r3
instruction = 1001001011010101
mips.Register.registers[011] =        00000000000000000000000000011111
 M[$r3 + 010101] =                    00000000000000000000000000011111

28. instruction is M[$r3 + 010101] = $r7
instruction = 1001011111010101
mips.Registers.registers[111] =       00000000000000000000000000010100
 M[$r3 + 010101] =                    00000000000000000000000000010100
```

```
# 31. instruction is if(rl == r3) -> pc = pc + 1 + signextend
# instruction = 1001011111010101
# mips.Registers.registers[001] =       11111111111111111111111111111111
# mips.Registers.registers[011] =       00000000000000000000000000011111
# mips.imm_extended =                   00000000000000000000000000010101
# Before instruction mips.pc.out =      00000000000000000000000000011110
# After instruction mips.pc.out =       00000000000000000000000000100011
#
# 32. instruction is if(r7 == r3) -> pc = pc + 1 + signextend
# instruction = 0110001011000100
# mips.Registers.registers[011] =       00000000000000000000000000011111
# mips.Registers.registers[111] =       00000000000000000000000000010100
# mips.imm_extended =                   00000000000000000000000000000100
# Before instruction mips.pc.out =      00000000000000000000000000100011
# After instruction mips.pc.out =       00000000000000000000000000101000
#
# 33. instruction is if(rl != r3) -> pc = pc + 1 + signextend
# instruction = 0110011111000100
# mips.Registers.registers[001] =       11111111111111111111111111111111
# mips.Registers.registers[011] =       00000000000000000000000000011111
# mips.imm_extended =                   00000000000000000000000000000100
# Before instruction mips.pc.out =      00000000000000000000000000101000
# After instruction mips.pc.out =       00000000000000000000000000101101
#
# 34. instruction is if(r7 != r3) -> pc = pc + 1 + signextend
# instruction = 0110011111000100
# mips.Registers.registers[011] =       00000000000000000000000000011111
# mips.Registers.registers[111] =       00000000000000000000000000010100
# mips.imm_extended =                   00000000000000000000000000000100
# Before instruction mips.pc.out =      00000000000000000000000000101101
# After instruction mips.pc.out =       00000000000000000000000000110010
#
```

In this single cycle processor, there are 7 main component: Instruction memory, control unit, Registers, Data memory, ALU, PC and also other components (adders, muxers, sign_extenders etc.).

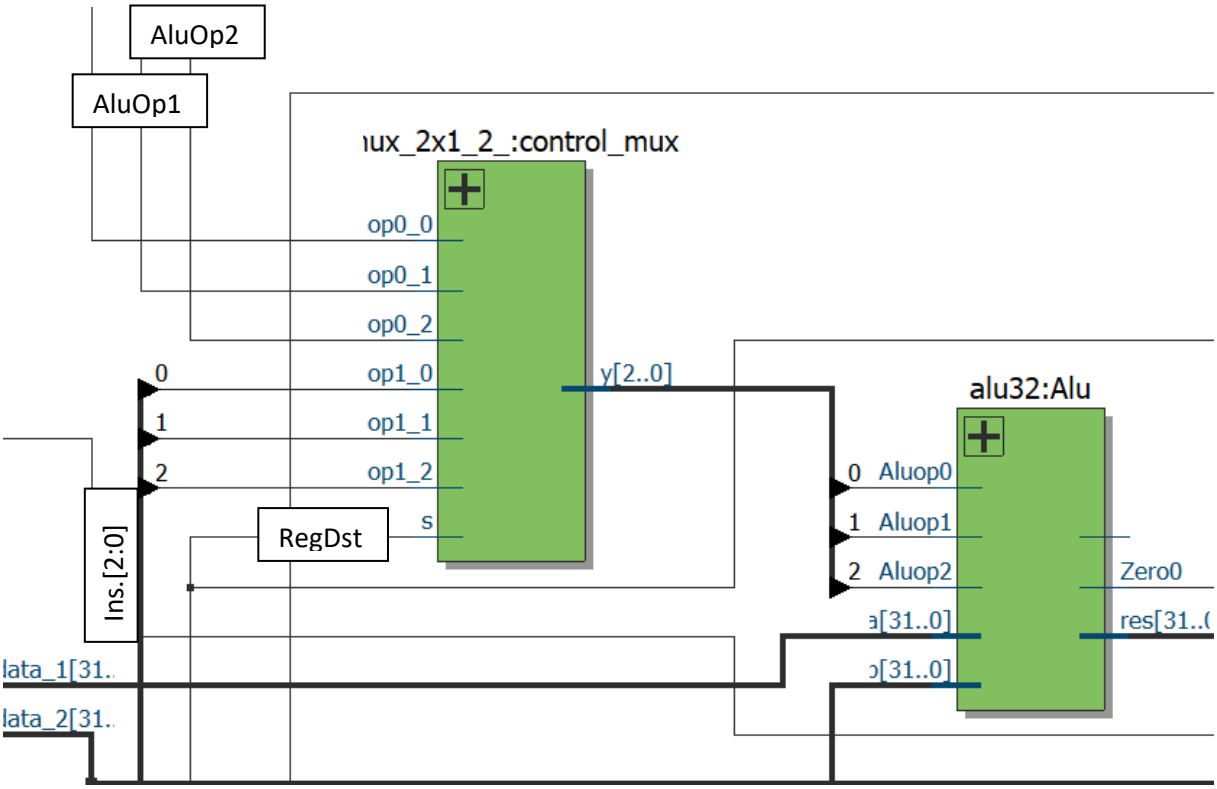
While processing design single cycle processor firstly I draw every component and implement finally wrote testbenches and find problems and then again test for every component.

1- Control unit part

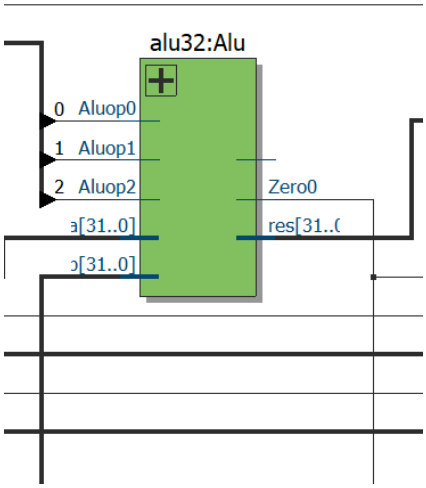



```
k.control_unit_tb
k.control_unit
-current
opcode = 0000 ,RegDst =1, Branch =0, Bneq =0, MemRead =0, MemtoReg =0, AluOp2 =0, AluOp1 =0,  AluOp0 =0, MemWrite=0,  AluSrc =0,  RegWrite =1,  Slti =0,
 opcode = 0001 ,RegDst =0, Branch =0, Bneq =0, MemRead =0, MemtoReg =0, AluOp2 =0, AluOp1 =0,  AluOp0 =1, MemWrite=0,  AluSrc =1,  RegWrite =1,  Slti =0,
 opcode = 0010 ,RegDst =0, Branch =0, Bneq =0, MemRead =0, MemtoReg =0, AluOp2 =0, AluOp1 =0,  AluOp0 =0, MemWrite=0,  AluSrc =1,  RegWrite =1,  Slti =0,
 opcode = 0011 ,RegDst =0, Branch =0, Bneq =0, MemRead =0, MemtoReg =0, AluOp2 =1, AluOp1 =0,  AluOp0 =1, MemWrite=0,  AluSrc =1,  RegWrite =1,  Slti =0,
 opcode = 0100 ,RegDst =0, Branch =0, Bneq =0, MemRead =0, MemtoReg =0, AluOp2 =1, AluOp1 =0,  AluOp0 =0, MemWrite=0,  AluSrc =1,  RegWrite =1,  Slti =0,
 opcode = 0101 ,RegDst =0, Branch =1, Bneq =0, MemRead =0, MemtoReg =0, AluOp2 =0, AluOp1 =1,  AluOp0 =0, MemWrite=0,  AluSrc =0,  RegWrite =0,  Slti =0,
 opcode = 0110 ,RegDst =0, Branch =0, Bneq =1, MemRead =0, MemtoReg =0, AluOp2 =0, AluOp1 =1,  AluOp0 =0, MemWrite=0,  AluSrc =0,  RegWrite =0,  Slti =0,
 opcode = 0111 ,RegDst =0, Branch =0, Bneq =0, MemRead =0, MemtoReg =0, AluOp2 =0, AluOp1 =1,  AluOp0 =0, MemWrite=0,  AluSrc =1,  RegWrite =1,  Slti =1,
 opcode = 1000 ,RegDst =0, Branch =0, Bneq =0, MemRead =1, MemtoReg =1, AluOp2 =0, AluOp1 =0,  AluOp0 =1, MemWrite=0,  AluSrc =1,  RegWrite =1,  Slti =0,
 opcode = 1001 ,RegDst =0, Branch =0, Bneq =0, MemRead =0, MemtoReg =0, AluOp2 =0, AluOp1 =0,  AluOp0 =1, MemWrite=1,  AluSrc =1,  RegWrite =0,  Slti =0,
```

‘‘Ayrıca Hocam alu controlü kullanmaya gerek kalmadan control unitten çıkan AluOp sinyallerini kullanıp ve RegDst sinyaliyle R type olup olmadığını kontol edip, instructionın son 3 biti ile muxlayarak bir tasarım yaptım. Ekstra puan imkanı varsa baya iyi olur 😊.’’

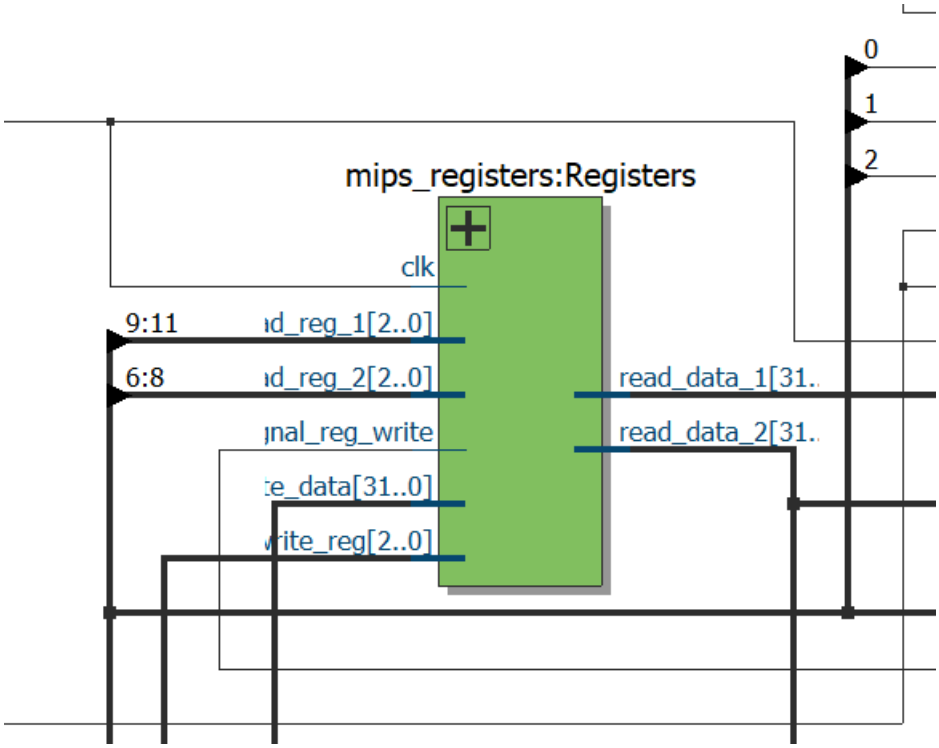


2- Alu



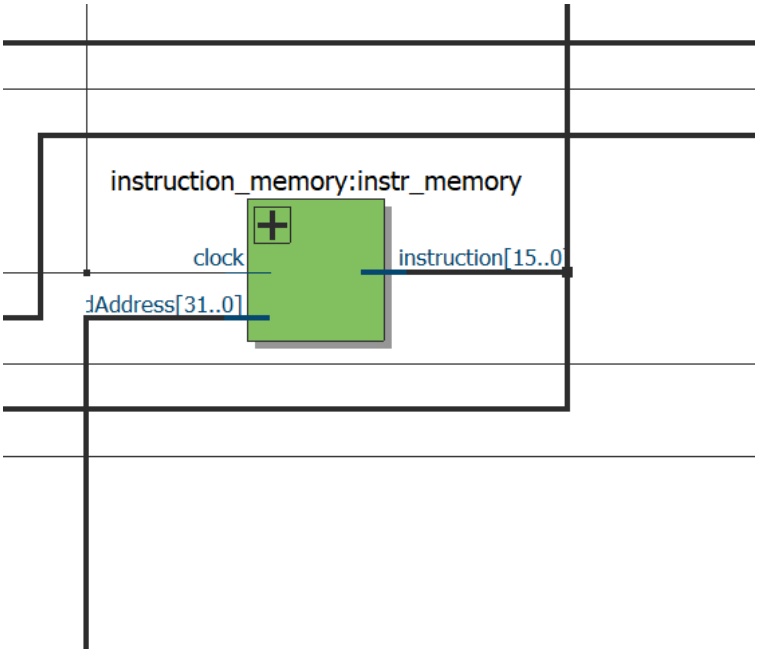

```
A =00000000000000000000000000000000, B=00000000000000000000000000000000, ALUop=000, R=00000000000000000000000000000000
 A =00110011001100110011001100110011, B=00110011001100110011001100110011, ALUop=000, R=00110011001100110011001100110011
 A =11111111111111111111111111111111, B=11111111111111111111111111111111, ALUop=000, R=11111111111111111111111111111111
 A =00110011001100110011001100110011, B=00110011001100110011001100110011, ALUop=000, R=00110011001100110011001100110011
 A =10000001000100100010010100000000, B=00000000111000000000000000000000, ALUop=001, R=10000001111100100010010100000000
 A =10110011001100110011001100110011, B=00110011001100110011001100110011, ALUop=001, R=11100110011001100110011001100110
 A =10111011111111001110100111111011, B=11110001111111111000011110011011, ALUop=001, R=10101101111110001110001100101110
 A =10101111001100011011001000000011, B=00110011001100110011011100110011, ALUop=001, R=11100010011001001110100100110110
 A =01010101010101010101010101010101, B=00011101010101010001110101010101, ALUop=010, R=00111000000000000011100000000000
 A =00000000000000000000001100110011, B=00110011001100110011001100110011, ALUop=010, R=11001100110011001101000000000000
, A =01010101010101010101010101010101, B=01010101010101010101010101010101, ALUop=010, R=00000000000000000000000000000000
, A =00001111001100011011001000000011, B=00110011001100110011011100110011, ALUop=010, R=11011011111111100111101011010000
, A =10000001000100100010010100000000, B=00000000011000000000000000000000, ALUop=011, R=10000001011100100010010100000000
, A =10110011001100110011001100110011, B=00110011001100110011001100110011, ALUop=011, R=10000000000000000000000000000000
, A =10111011111111001110100111111011, B=11110001111111111000011110011011, ALUop=011, R=01001010000001101101110011001000
, A =10101111001100011011001000000011, B=00110011001100110011011100110011, ALUop=011, R=10011100000000101000010100110000
, A =01010101010101010101010101010101, B=00011101010101010001110101010101, ALUop=100, R=10100010101010101010001010101010
, A =00000000000000000000001100110011, B=00110011001100110011001100110011, ALUop=100, R=11001100110011001100110011001100
, A =01010101010101010101010101010101, B=01010101010101010101010101010101, ALUop=100, R=10101010101010101010101010101010
, A =00001111001100011011001000000011, B=00110011001100110011011100110011, ALUop=100, R=11000000110011000100100011001100
, A =10000001000100100010010100000000, B=00000000111000000000000000000000, ALUop=101, R=10000001111100100010010100000000
, A =10110011001100110011001100110011, B=00110011001100110011001100110011, ALUop=101, R=10110011001100110011001100110011
, A =10111011111111001110100111111011, B=11110001111111111000011110011011, ALUop=101, R=11111011111111111111101111111011
, A =10101111001100011011001000000011, B=00110011001100110011011100110011, ALUop=101, R=10111111001100111011011100110011
, A =10000001000100100010010100000000, B=00000000111000000000000000000000, ALUop=110, R=00000000000000000000000000000000
, A =10110011001100110011001100110011, B=00110011001100110011001100110011, ALUop=110, R=00110011001100110011001100110011
, A =10111011111111001110100111111011, B=11110001111111111000011110011011, ALUop=110, R=10110001111111001000000110011011
, A =10101111001100011011001000000011, B=00110011001100110011011100110011, ALUop=110, R=00100011001100010011001000000011
, A =10000001000100100010010100000000, B=00000000111000000000000000000000, ALUop=111, R=00000000000000000000000000000000
, A =10110011001100110011001100110011, B=00110011001100110011001100110011, ALUop=111, R=00110011001100110011001100110011
, A =10111011111111001110100111111011, B=11110001111111111000011110011011, ALUop=111, R=10110001111111001000000110011011
, A =10101111001100011011001000000011, B=00110011001100110011011100110011, ALUop=111, R=00100011001100010011001000000011
```

3- Registers



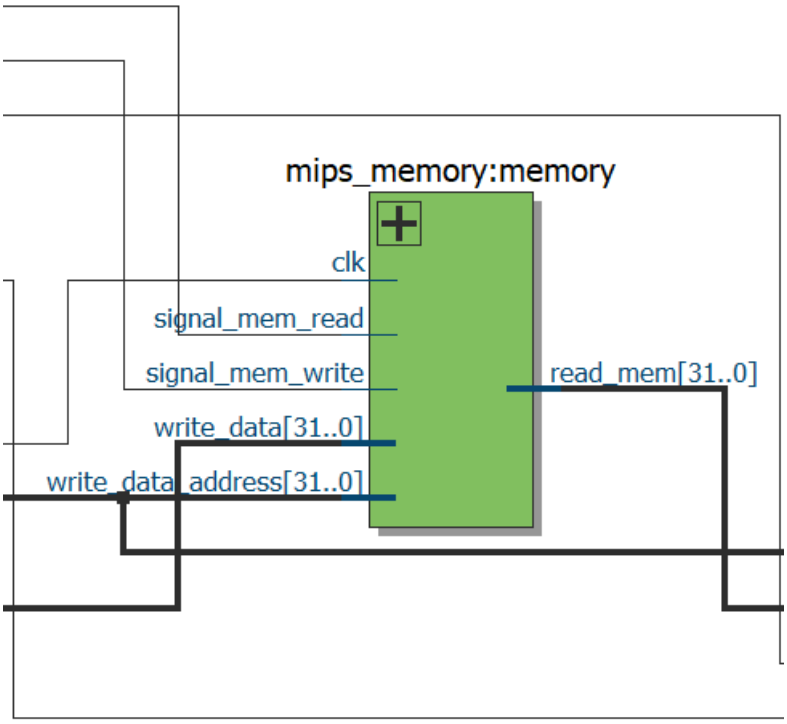

```
VSIM 124> step -current
# read 0 and 6. register with try to writing reg0 for something but if try to write reg0 then register write 00000...0000 to r0
# time = 40, read_data_1 = 00000000000000000000000000000000, read_data_2 = 00000000000000000000000000000110, write_data = 00110011001100110011001100110000, signal_reg_write = 1
# read 1 and 4. register and writing 32'b0000..0000 to 4. register so read_reg_2 == 0000....000
# time = 80, read_data_1 = 00000000000000000000000000000001, read_data_2 = 00000000000000000000000000000000, write_data = 00000000000000000000000000000000, signal_reg_write = 1
# read 4. register only both of read_reg and dont write anything
# time = 120, read_data_1 = 00000000000000000000000000000000, read_data_2 = 00000000000000000000000000000000, write_data = 00110011001100110011001100110011, signal_reg_write = 0
# Break in Module mips_registers_tb at C:/Users/OmerF/Desktop/hw4/mips_registers_tb.v line 64
```

4- Instruction Memory




```
VSIM 286> step -current
# time = 80, read_mem = 1001000000100000
# time = 120, read_mem = 0000001000010001
# time = 160, read_mem = 0000001000010001
# time = 200, read_mem = 0000001000010001
# time = 240, read_mem = 0000001000010001
# Break in Module instruction_memory_tb at C:
```

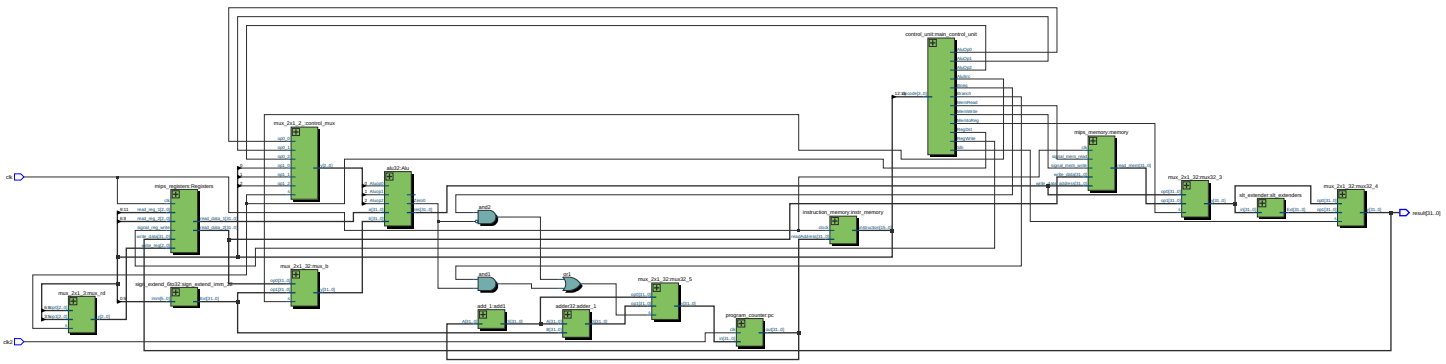
5- Data Memory




```
vsim work.mips_memory_tb
Loading work.mips_memory_tb
Loading work.mips_memory
VM 256> step -current
write to 0. address but signal mem write = 0 so dont write anything but signal_mem_read = 1 so read 1. a
write_data_address = 00000000000000000000000000000001, signal_mem_read 1, signal_mem_write 1
time = 80, read_mem = 00000000000000000000000000010100
write to 0. address 32'b1111....1111 signal mem write = 1 so mem_data[0]=1111...1111
write_data_address = 00000000000000000000000000000000, signal_mem_read 0, signal_mem_write 1
time = 120, read_mem = 00000000000000000000000000010100
write to 0. address but signal mem write = 0 so dont write anything
write_data_address = 00000000000000000000000000000000, signal_mem_read 1, signal_mem_write 0
time = 160, read_mem = 11111111111111111111111111111111
Break in Module mips_memory_tb at C:/Users/OmerF/Desktop/hw4/mips_memory_tb.v line 63
```

Finally minimips single cycle processor works properly but there is a one point, when lw proceedings' to I cant access right result end of 1 clock, I think this is about the clock design, but this problem handled by repeat instruction processing.

Ömer F. Akduman
1801042094


control_unit:main_control_unit
mux_2x1_2:control_mux
alu32:Alu
mips_registers:Registers
mips_memory:memory
instruction_memory:instr_memory
mux_2x1_32:mux32_3
alt_extender:alt_extenders
mux_2x1_32:mux32_4
mux_2x1_32:mux_9
mux_2x1_32:mux32_5
mux_2x1_3:mux_m1
sign_extend_16x32:sign_extend_imm_32
add_1:add1
adder32:adder_1
program_counter:pc
and0
and1
or0
clk
clk2
result[31..0]