# Gebze Technical University
# Computer Engineering

CSE 331 - 2018 Fall

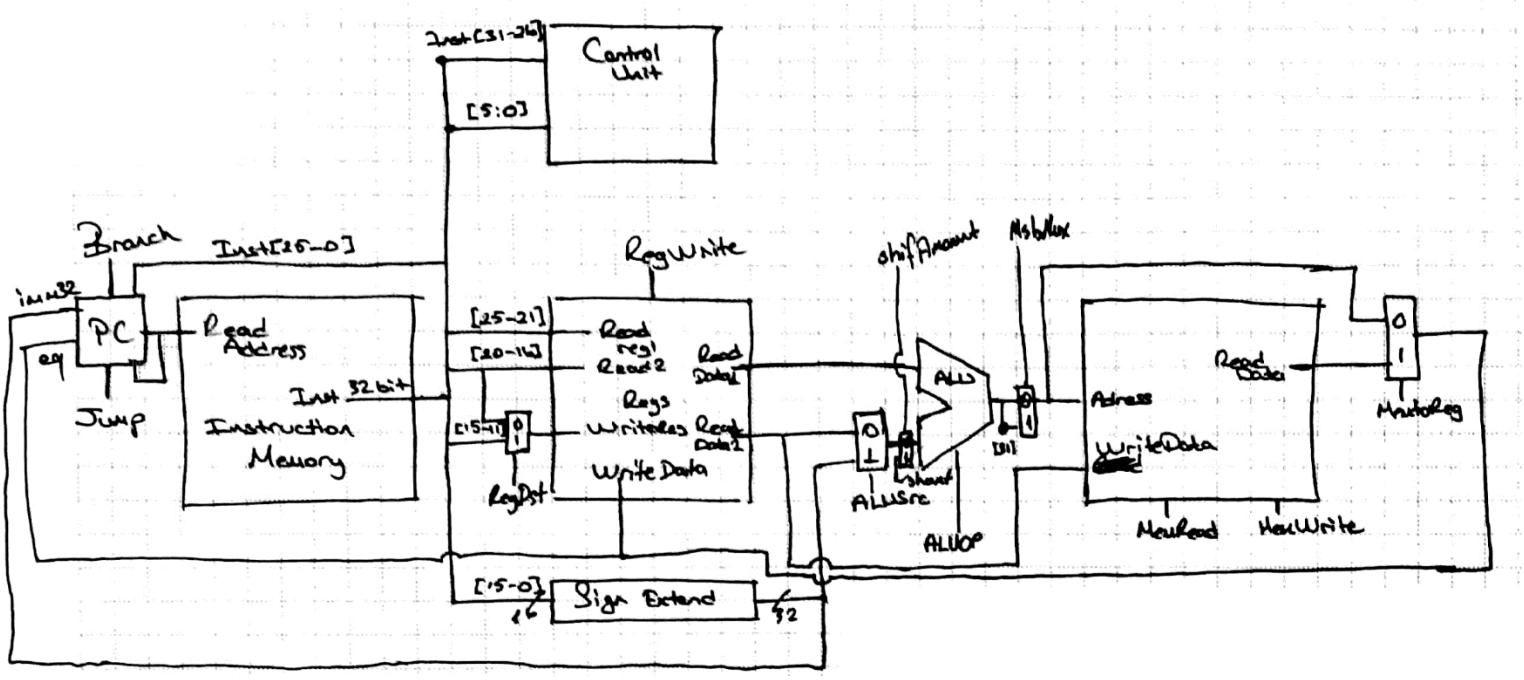
**ASSIGNMENT 4 REPORT**

HALİL ONUR ÇEÇEN

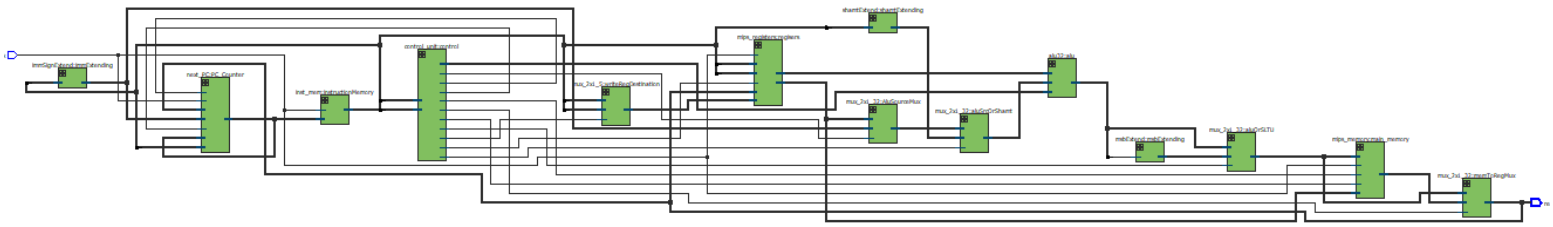
161044057

# Making of Single Cycle Datapath of a mips processor

To make the required datapath I used following schema.




- RTL View:

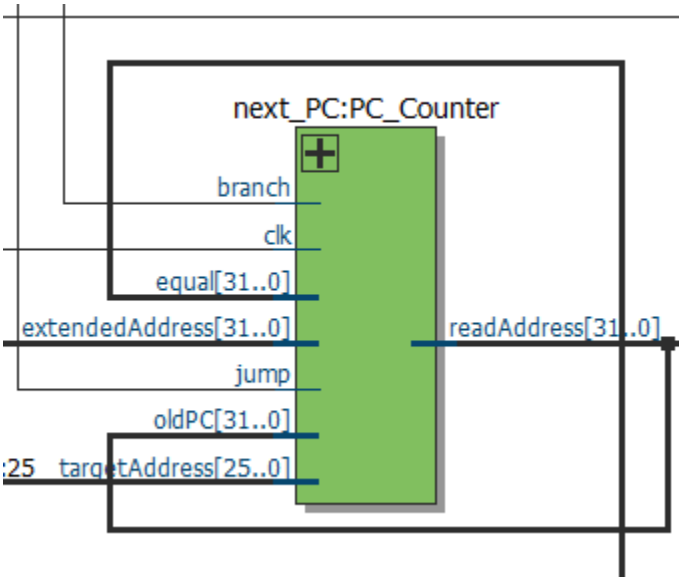


# Main Mips32 Module(Explained)

## 1. PC Counter (next_PC module)

I took jump address, branch address, jump and branch signal, old address and a clock as inputs. In every negative edge of clock first I checked if old address is unknown to determine if it's the first run or not. After that simply I incremented old address by one and assigned it to output. If its jump replaced output with target address. If its branch and equal is zero, I concatenated new address with branch offset.

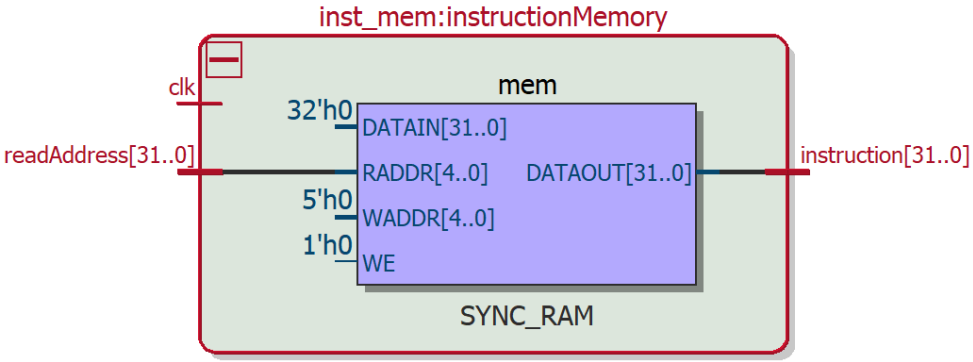
### RTL View:




## 2. PC Counter (next_PC module)

This module looks like register module. Only difference is this doesn't write anything back and whenever readAddress is changed (which happens on negedge of clock), instruction also changes.

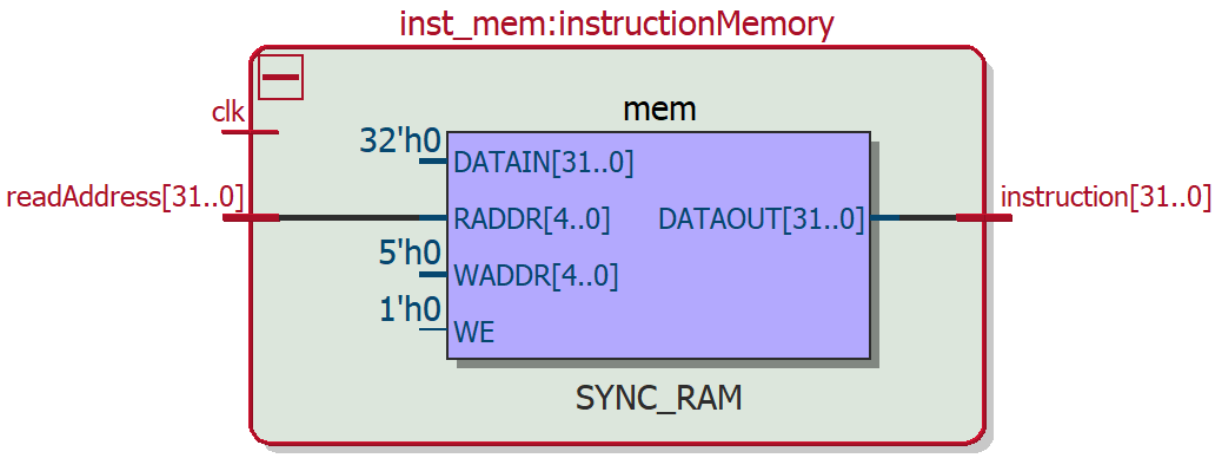
### RTL View:



## 3. Instruction Memory (inst_mem module)

I used inst.mem file to keep instructions. Read from this file like we I did in register block every time read_address is changed.

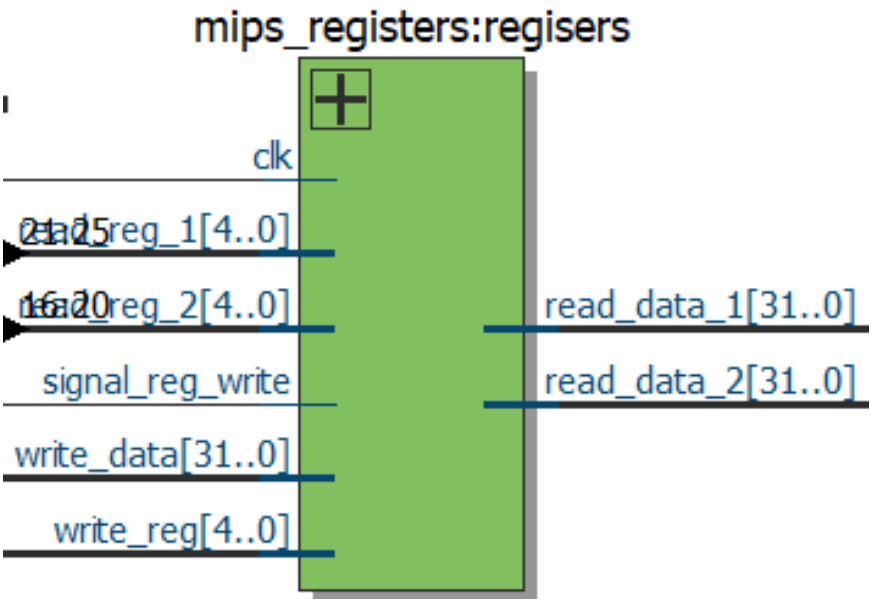
### RTL View:




## 4. Registers (Mips_register module)

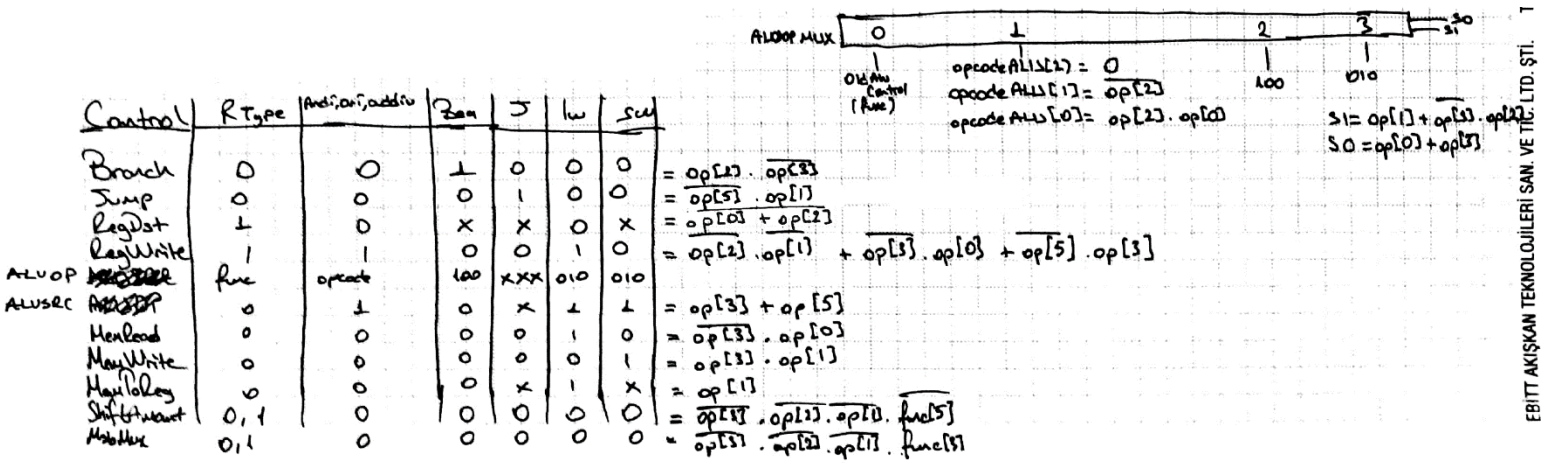
I haven't made changes from assignment3 in this module except for negedge clock. Now I change read data whenever read reg inputs are changed.

### RTL View:



## 5. Control Unit (control_unit module)

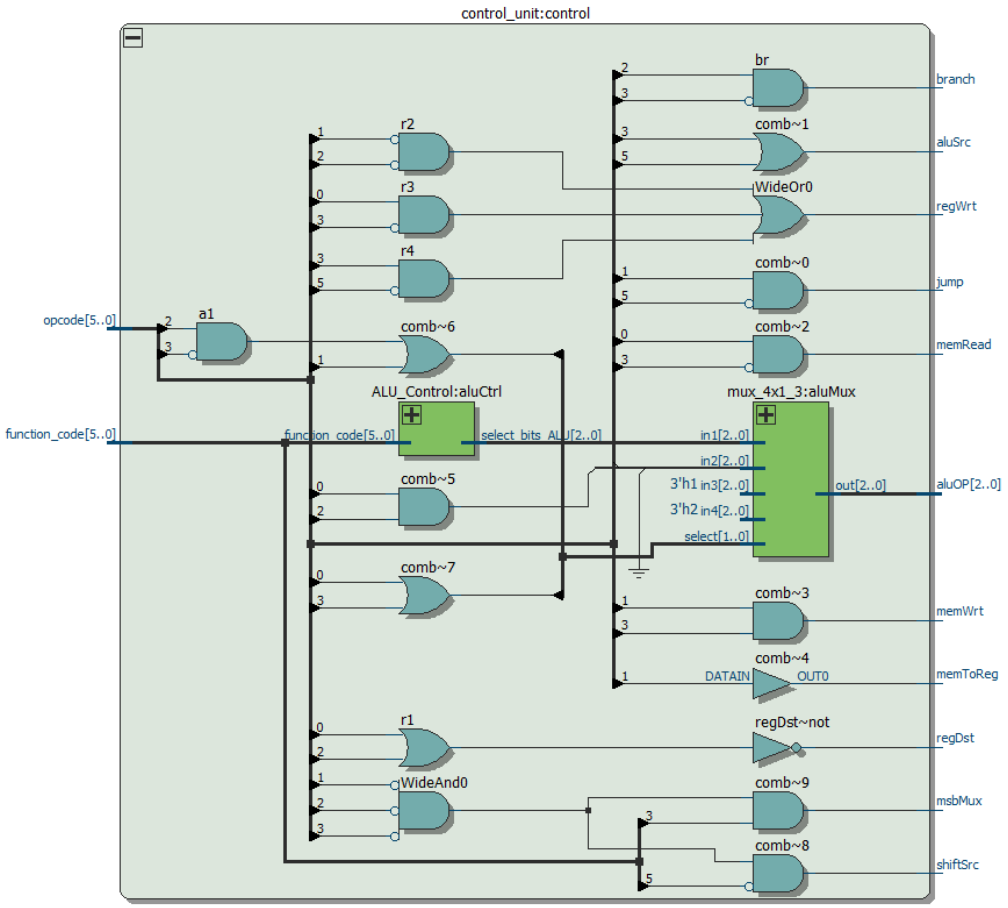
For this unit I used following logical operations: (op is opcode and func is function_code)




In the left table we can see control signals and their changes according to opcodes and function codes of instructions.

In the right table we can see how 3-bit aluOP signal is produced. I used muxes input0 with old control_unit as Alu_Control, used muxes input 1 for andi, ori, addiu instructions, input 2 for beq instruction and input 3 for load and store word instructions.

### RTL View:



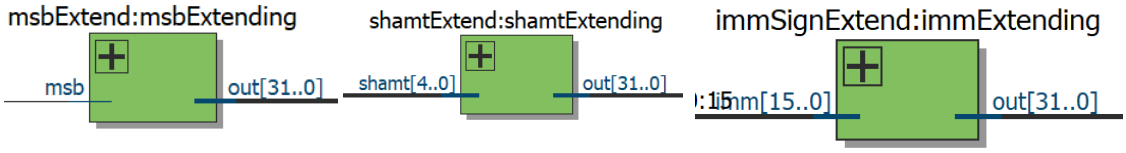
## 6. Alu Control (ALU_Control unit)

I used assignment3's control unit as Alu control for R types.

## 7. Extenders (msbExtend, shamtExtend, immSgnExtend modules)

I used 3 extender modules. msbExtend for getting only msb as 32-bit, shamtExtend for extending shamt as 32-bit, immSignExtend for extending immediate with respect to sign bit.

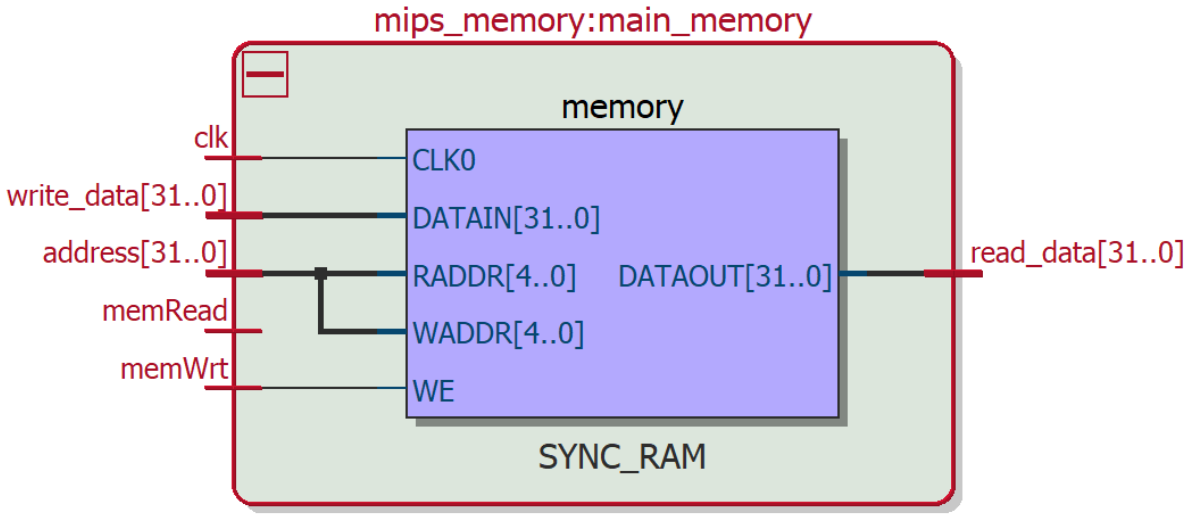
### RTL View:




## 8. Main Memory Block (mips_memory module)

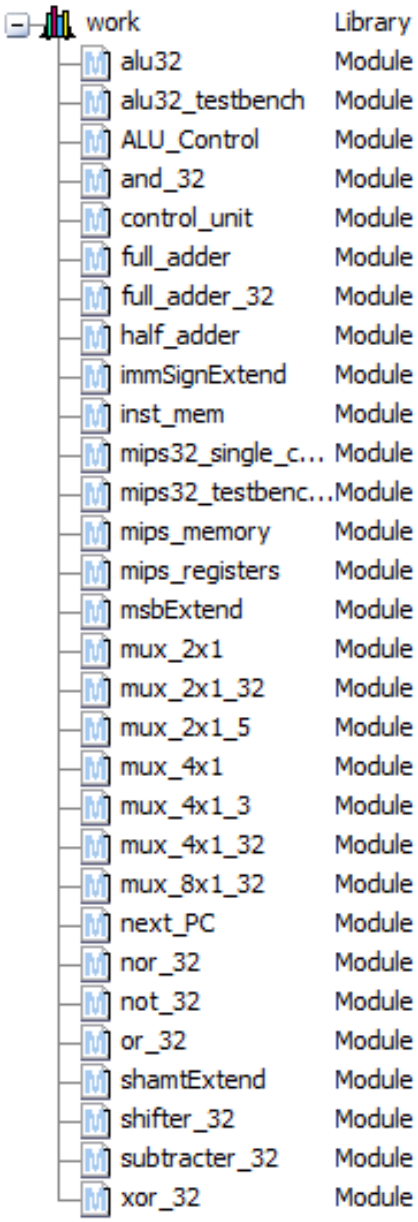
I used main.mem file to keep main memory. Read from and write to this file like we I did in register block with respect to memWrt and memRead signals. If memRead signal is 0, filled read data with 32-bit x.

### RTL View:



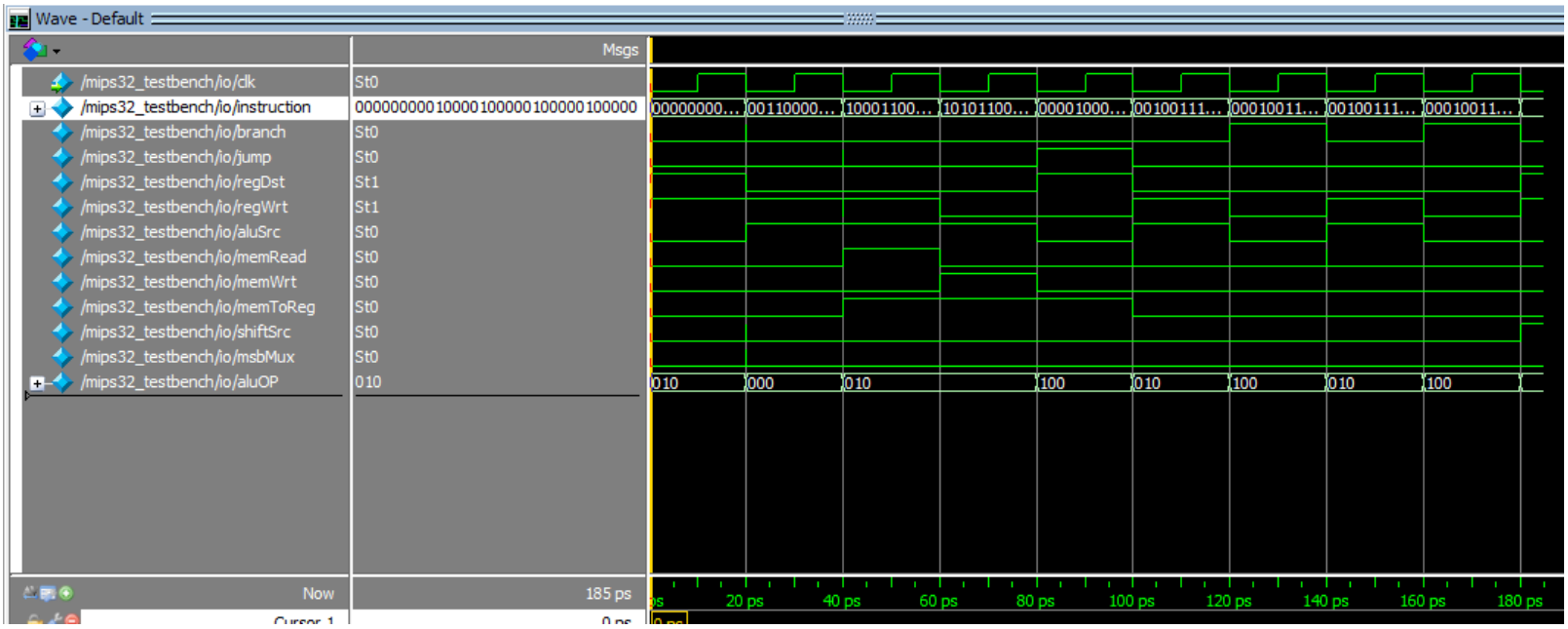
# ModelSim Results

## All modules



## Simulation Results

```
# PC=00000000000000000000000000000000, oldPC = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
# opcode= 000000, rs = 00010, rt = 00010, rd = 00001, shamt = 00000, func = 100000, imm = 0000100000100000, target = 00010000100000100000100000
# PC=00000000000000000000000000000001, oldPC = 00000000000000000000000000000000
# opcode= 001100, rs = 00010, rt = 01101, rd = 00000, shamt = 00000, func = 011000, imm = 0000000000011000, target = 00010011010000000000011000
# PC=00000000000000000000000000000010, oldPC = 00000000000000000000000000000001
# opcode= 100011, rs = 00011, rt = 00100, rd = 00000, shamt = 00000, func = 000010, imm = 0000000000000010, target = 00011001000000000000000010
# PC=00000000000000000000000000000011, oldPC = 00000000000000000000000000000010
# opcode= 101011, rs = 00011, rt = 00100, rd = 00000, shamt = 00000, func = 000000, imm = 0000000000000000, target = 00011001000000000000000000
# PC=00000000000000000000000000000100, oldPC = 00000000000000000000000000000011
# opcode= 000010, rs = 00000, rt = 00000, rd = 00000, shamt = 00000, func = 000110, imm = 0000000000000110, target = 00000000000000000000000110
# PC=00000000000000000000000000000110, oldPC = 00000000000000000000000000000100
# opcode= 001001, rs = 11110, rt = 11110, rd = 00000, shamt = 00000, func = 000001, imm = 0000000000000001, target = 11110111100000000000000001
# PC=00000000000000000000000000000111, oldPC = 00000000000000000000000000000110
# opcode= 000100, rs = 11111, rt = 11110, rd = 11111, shamt = 11111, func = 111110, imm = 1111111111111110, target = 11111111101111111111111110
# PC=00000000000000000000000000000110, oldPC = 00000000000000000000000000000111
# opcode= 001001, rs = 11110, rt = 11110, rd = 00000, shamt = 00000, func = 000001, imm = 0000000000000001, target = 11110111100000000000000001
# PC=00000000000000000000000000000111, oldPC = 00000000000000000000000000000110
# opcode= 000100, rs = 11111, rt = 11110, rd = 11111, shamt = 11111, func = 111110, imm = 1111111111111110, target = 11111111101111111111111110
# PC=00000000000000000000000000001000, oldPC = 00000000000000000000000000000111
# opcode= 000000, rs = 00000, rt = 00000, rd = 00000, shamt = 00000, func = 000000, imm = 0000000000000000, target = 00000000000000000000000000
```




This instruction set tests R type, I type, lw, sw, j and beq. Also control bits can be seen in the Wave view.