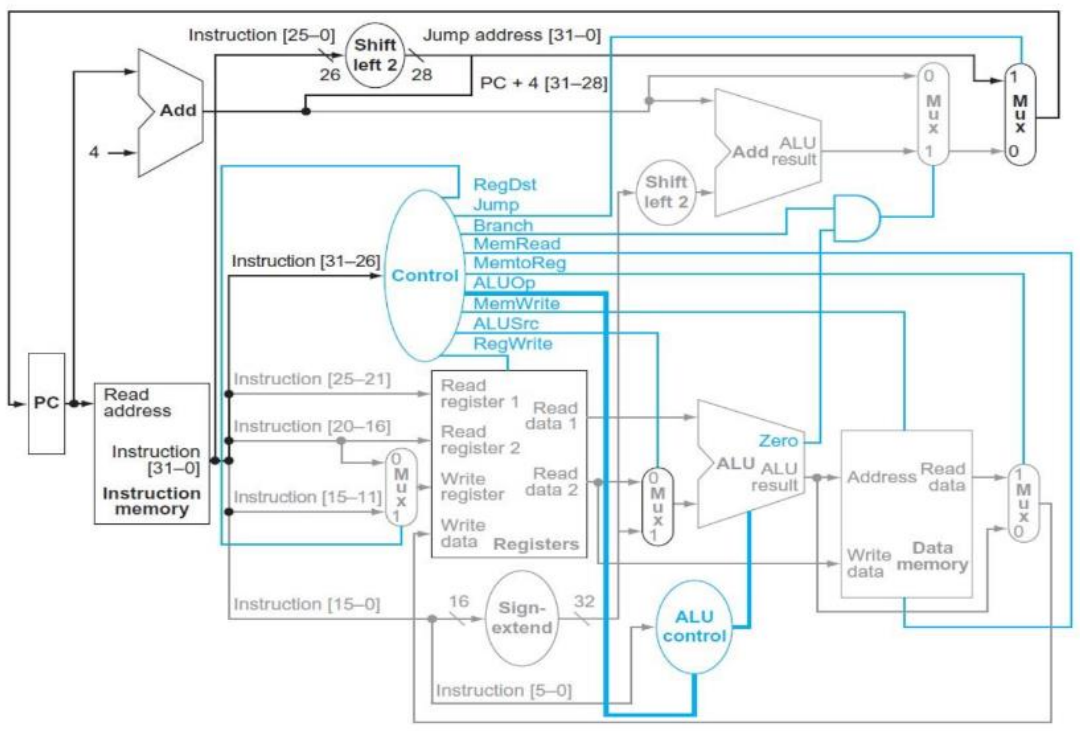
Single cycle processor design



- The above diagram shows the design of a single cycle processor, consisting of different modules that were implemented in Verilog. (Note: The code consists of files that might have various modules combined, slightly different modules or implementation nonetheless, the design models the working of the processor shown in the diagram)
- The various instructions implemented are add, addi, addu, and, andi, or, ori , sub, subu, jump, beq, bne, slt, sltu, slti, lui, lw, sw, jal, j and jr.
- The explanation of different of different files and modules is described below for ease of understanding.

Files and modules

- **ALU.v**
  - This file consists of an ALU module (arithmetic logic unit) which is used for carrying out arithmetic operations on the given inputs, according to the control input specified.

  - The various operations performed by the ALU are add, subtract, and, or, nor, slt etc and all of these operation are specified in the file in comments.

- **d_mem.v**
  - This file consists of the d_mem module, which models the data memory of the processor, where writes and reads take place into the memory. Some of the instructions using data memory are lw and sw. The memory used here has a depth of 256 and width of 32 bits.
  - According to the control signals specified (MemRead and MemWrite), either writes or reads can take place into the data memory. Writes occur at the clock edge whereas reads can occur anytime.

- **register_file.v**
  - This file consists of register_file module, which models the register bank or register file of the processor. Read and write addresses are provided, where data is read or can be written into. MIPS_32 architecture consists of this register bank having 32 registers, each of 32 bits.
  - According to the control signals specified, writes take place into the register file at positive edge of the clock whereas reads can occur anytime.

- **Sign_extension files**
  - These files are used for sign extending the input to 32 bits, the outputs of which act as immediate data or constants for various instructions. Eg: lw, sw etc. The modules present are sign_extender_26_to_32 and sign_extender_16_to_32. (As the name suggests, used for sign extending the bitlength specified to 32 bits)
  - The sign_extender_16_to_32 is also used for calculation of addresses for updation of program counter.

- **Mux files**
  - The mux files are used for selecting one of the inputs to the output, according to the control inputs specified. The control inputs are generated by the main_control module.
  - The location of the different muxes is specified in the diagram of the design.
  - The different mux modules present are mux_5bit and mux_32bit.

- **main_control.v**
  - The main_control.v consists of the main_control module, used for generating different control signals for different instructions carried out by the processor.

- The control signals are as specified in the diagram, with additional control signals implemented for bne and jal instructions.
- This implementation combines the modules Control and ALUControl as specified in the diagram.

- **instr_mem.v**
  - This file consists of instr_mem module, which models the instruction memory of the processor. This module stores the instruction of a program (machine code) which can be retrieved.

- **single_cycle_proc.v**
  - This file consists of the single_cycle_processor module, which models the entire processor. It consists of the instantiations of different modules as specified above and consists of the logic for updating the program counter.

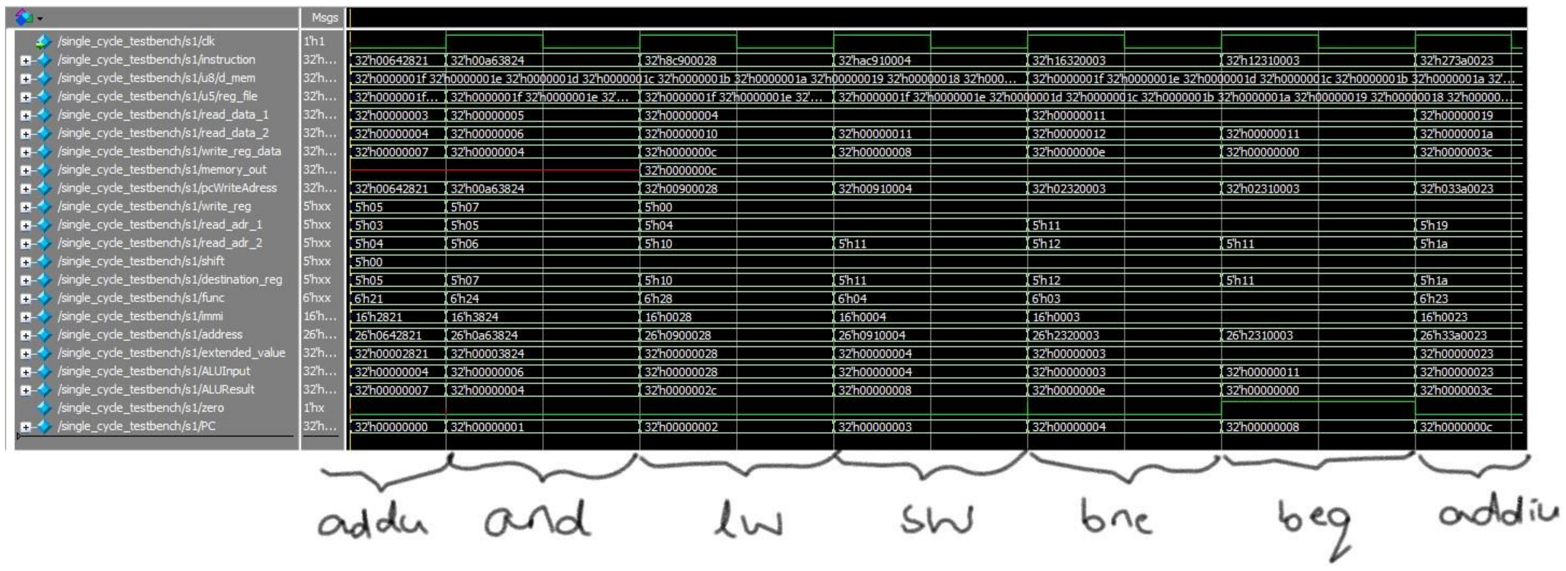
Simulations

- Three files instruction.mem, data.mem and registers.mem are specified for initialization of different memory modules (instruction memory, register file and data memory) and is initialized in the code.
- For testing purposes, the instruction.mem file can be modified to provide the required instructions for simulation.
- The instructions implemented for simulation purposes are:

```
addu $a1, $v1, $a0
and $a3, $a1, $a2
lw $s0, 0x0028, $a0
sw $s1, 0x04, $a0
bne $s1, $s2,0x03
or $t1, $a3, $t0
nor $t3, $t2, $t1
sll $t5, $t4, 0x04
beq $s1, $s1, 0x03
subu $s2, $fp, $s1
slt $s4, $s2, $sp
j 0xd
addiu $k0, $t9, 0x23
```

ori $k1, $k0, 0x19

- The instructions provided above are not exhaustive and instructions can be deleted, added or modified from the instruction.mem file for testing the various instructions implemented.
- While simulating, it is to be made sure that the .mem files are present in the simulation directory.



```
                                            Msgs
/single_cycle_testbench/s1/clk              1'h1
/single_cycle_testbench/s1/instruction      32'h...  32'h00642821 32'h00a63824 32'h8c900028 32'hac910004 32'h16320003 32'h12310003 32'h273a0023
/single_cycle_testbench/s1/u8/d_mem         32'h...  32'h0000001f 32'h0000001e 32'h0000001d 32'h0000001c 32'h0000001b 32'h0000001a 32'h00000019 32'h00000018 32'h000... 32'h0000001f 32'h0000001e 32'h0000001d 32'h0000001c 32'h0000001b 32'h0000001a 32'...
/single_cycle_testbench/s1/u5/reg_file      32'h...  32'h0000001f... 32'h0000001f 32'h0000001e 32'... 32'h0000001f 32'h0000001e 32'... 32'h0000001f 32'h0000001e 32'h0000001d 32'h0000001c 32'h0000001b 32'h0000001a 32'h00000019 32'h00000018 32'h00000...
/single_cycle_testbench/s1/read_data_1      32'h...  32'h00000003 32'h00000005 32'h00000004 32'h00000011 32'h00000019
/single_cycle_testbench/s1/read_data_2      32'h...  32'h00000004 32'h00000006 32'h00000010 32'h00000011 32'h00000012 32'h00000011 32'h0000001a
/single_cycle_testbench/s1/write_reg_data   32'h...  32'h00000007 32'h00000004 32'h0000000c 32'h00000008 32'h0000000e 32'h00000000 32'h0000003c
/single_cycle_testbench/s1/memory_out       32'h...  32'h0000000c
/single_cycle_testbench/s1/pcWriteAdress    32'h...  32'h00642821 32'h00a63824 32'h00900028 32'h00910004 32'h02320003 32'h02310003 32'h033a0023
/single_cycle_testbench/s1/write_reg        5'hxx    5'h05 5'h07 5'h00
/single_cycle_testbench/s1/read_adr_1       5'hxx    5'h03 5'h05 5'h04 5'h11 5'h19
/single_cycle_testbench/s1/read_adr_2       5'hxx    5'h04 5'h06 5'h10 5'h11 5'h12 5'h11 5'h1a
/single_cycle_testbench/s1/shift            5'hxx    5'h00
/single_cycle_testbench/s1/destination_reg  5'hxx    5'h05 5'h07 5'h10 5'h11 5'h12 5'h11 5'h1a
/single_cycle_testbench/s1/func             6'hxx    6'h21 6'h24 6'h28 6'h04 6'h03 6'h23
/single_cycle_testbench/s1/immi             16'h...  16'h2821 16'h3824 16'h0028 16'h0004 16'h0003 16'h0023
/single_cycle_testbench/s1/address          26'h...  26'h0642821 26'h0a63824 26'h0900028 26'h0910004 26'h2320003 26'h2310003 26'h33a0023
/single_cycle_testbench/s1/extended_value   32'h...  32'h00002821 32'h00003824 32'h00000028 32'h00000004 32'h00000003 32'h00000023
/single_cycle_testbench/s1/ALUInput         32'h...  32'h00000004 32'h00000006 32'h00000028 32'h00000004 32'h00000003 32'h00000011 32'h00000023
/single_cycle_testbench/s1/ALUResult        32'h...  32'h00000007 32'h00000004 32'h0000002c 32'h00000008 32'h0000000e 32'h00000000 32'h0000003c
/single_cycle_testbench/s1/zero             1'hx
/single_cycle_testbench/s1/PC               32'h...  32'h00000000 32'h00000001 32'h00000002 32'h00000003 32'h00000004 32'h00000008 32'h0000000c
```

- Because of the effect of bne and beq, many instructions are skipped hence after testing this, these two instructions can be removed for testing the instructions without skipping.
- d_mem and reg_file was checked for updation of values here for checking different instructions.