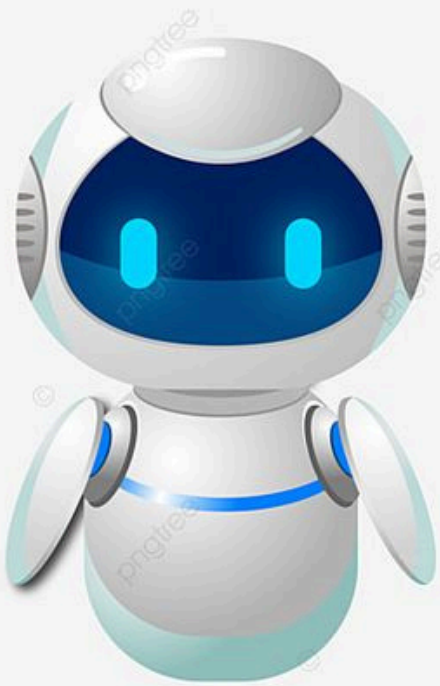


Project Report

Single Cycle Processor

DR/ Howida Abdullatif
Eng/Alaa Gawish
Eng/Ferial Ahmed

Team Members

Menna Allah Essam Ahmed Salem	**20812021200937**
Bassant Tamer Muhammed Abdullatif	**20812021200351**
Salma Osama Mohammed Abdallah	20812021201498

Datapath Components

1. **Program Counter (PC):** A 32-bit register that holds the address of the current instruction.
2. **Instruction Memory:** A memory unit that outputs a 32-bit instruction based on the 32-bit address input.
3. **Register File:** Contains 32 registers, each 32 bits wide, with two read ports and one write port.
    - **Read Ports:** A1 and A2 are 5-bit address inputs for reading; RD1 and RD2 are 32-bit outputs.
    - **Write Port:** A3 is the 5-bit address input, WD is the 32-bit data input, WE3 is the write enable signal, and it operates on the clock's rising edge.
4. **ALU (Arithmetic Logic Unit):** Performs arithmetic and logical operations.
5. **Data Memory:** A memory unit with one read/write port.
    - **Inputs:** A is the address, WD is the data to be written, and WE is the write enable signal.
    - **Output:** RD is the data read from the memory.

Control Unit Signals

The control unit is responsible for generating signals that coordinate the operations of the data path components in the single-cycle MIPS processor. These signals include multiplexer select signals, register enable signals, and memory write signals. Here's a more detailed look at each type:

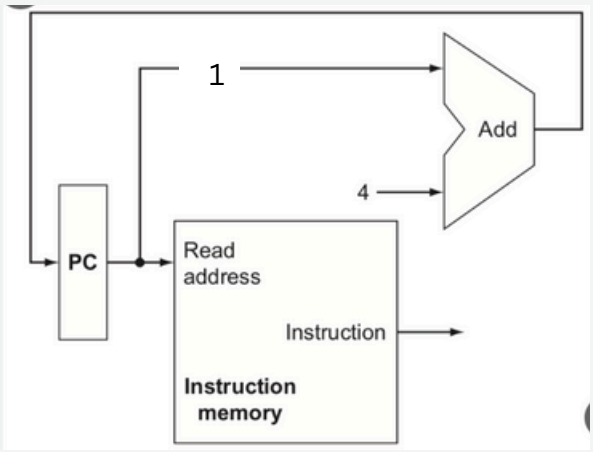
Multiplexer Select Signals
Multiplexer select signals control which data paths are used for operations. Multiplexers are used to select between different inputs based on the control signals provided by the control unit. These signals are essential for directing data flow to the correct destinations within the processor.

Register Enable Signals
Register enable signals control whether a register should be written to. When a register enable signal is active (set to 1), the data provided at the write data input is written into the specified register. This ensures that registers are updated correctly based on the instruction being executed.

Memory Write Signals
Memory write signals control whether data should be written to memory. If the memory write signal is active, data is written to the memory at the specified address. If it is inactive, the memory performs a read operation instead. These signals are crucial for distinguishing between read and write operations in the data memory.

Adder

pc_adder



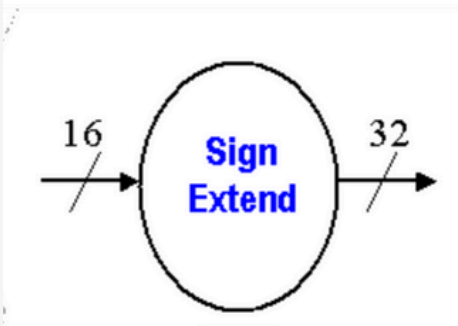

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity pc_adder is
   generic (
      nbit_width : integer := 32
   );
   Port (
      input : in  std_logic_vector(nbit_width-1 downto 0);
      output : out  std_logic_vector(nbit_width-1 downto 0)
   );
end pc_adder;

architecture BEHAV of pc_adder is
begin
   output <= input + X"00000001";
end BEHAV;
```

pc

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity pc_adder is
   generic (
      nbit_width : integer := 32
   );
   Port (
      input : in  std_logic_vector(nbit_width-1 downto 0);
      output : out  std_logic_vector(nbit_width-1 downto 0)
   );
end pc_adder;

architecture BEHAV of pc_adder is
begin
   output <= input + X"00000001";
end BEHAV;
```

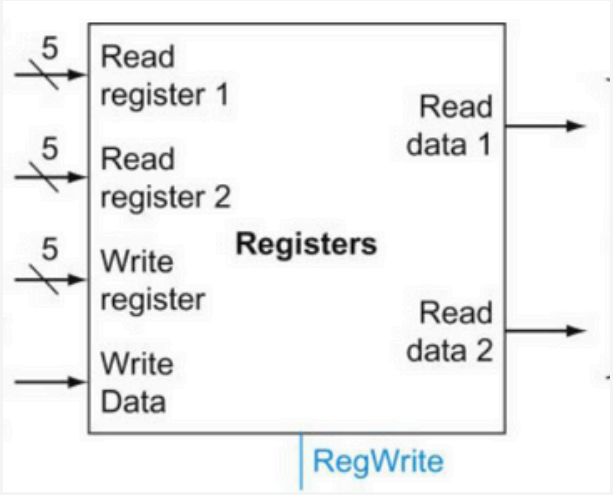
Sign Extension

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```




```
entity sign_extend is
   generic(
         nbit_input_width : integer := 16;
         nbit_output_width : integer := 32
        );
   Port (
        input : in  std_logic_vector (nbit_input_width-1 downto 0);
        output : out  std_logic_vector (nbit_output_width-1 downto 0)
  );
end sign_extend;

architecture BEHAV of sign_extend is
begin
 output<= x"0000"&input when input(nbit_input_width-1) = '0' else
x"FFFF"&input;
end BEHAV;
```

Register File

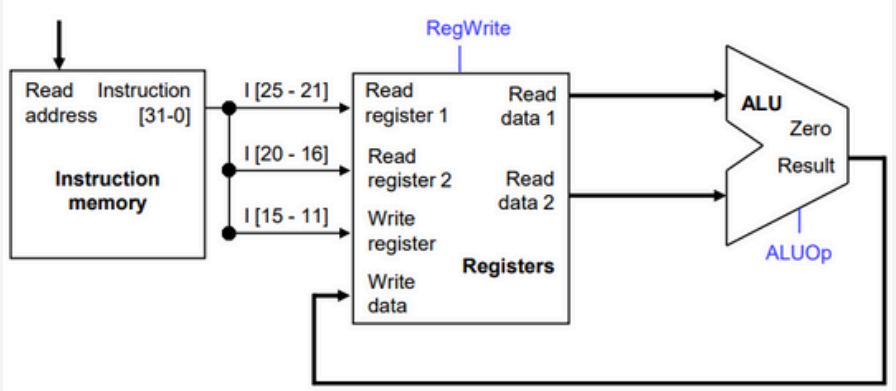



```
library ieee;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

use ieee.std_logic_unsigned.all;

entity register_file is
   generic (
      nbit_width : integer := 32;
      num_of_registers : integer := 32
   );
   port (
 clk : in std_logic;
      read_register1, read_register2 : in std_logic_vector(4 downto 0);
      write_register : in std_logic_vector(4 downto 0);
      write_data : in std_logic_vector(nbit_width-1 downto 0);
      register_write_ctrl : in std_logic;
      read_data1, read_data2 : out std_logic_vector(nbit_width-1 downto 0)
   );
end register_file;

architecture BEHAV of register_file is
   type registerfile_mem is array (0 to num_of_registers-1) of std_logic_vector(nbit_width-1 downto 0);
   signal reg_mem : registerfile_mem := (others => (others => '0'));

begin

 process(clk)
 begin
  if(clk'event and clk='1') then
   if (register_write_ctrl='1') then
    reg_mem(conv_integer(write_register)) <= write_data;
   end if;
  end if;
 end process;

 process(clk, read_register1, read_register2)
  begin
     if (conv_integer(read_register1) = 0) then
     read_data1 <= x"00000000";
    else
           read_data1 <= reg_mem(conv_integer(read_register1));
    end if;

    if (conv_integer(read_register2) = 0) then
       read_data2 <= x"00000000";
     else
         read_data2 <= reg_mem(conv_integer(read_register2));
     end if;
 end process;
end BEHAV;
```

Hint:
32 file register from $0 to $31

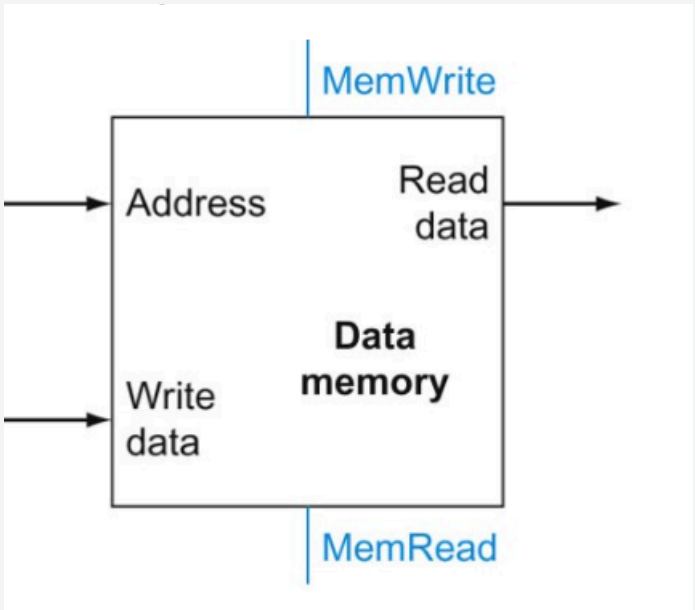
Instruction Memory

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity instruction_memory is
generic (
        nbit_width : integer := 32);

Port (
address : in std_logic_vector(nbit_width-1 downto 0);
instruction : out std_logic_vector(nbit_width-1 downto 0)
);
end instruction_memory;

architecture BEHAV of instruction_memory is

type memory_instr is array(0 to 14) of std_logic_vector(nbit_width-1 downto 0);
--addi,sub,or,and ,addi,lw,sw ,beq
signal sig_memory_instr : memory_instr :=
(
X"8c020002",  --lw $2, 2($0)   ->  $2 = data_mem[2]
X"8CC10000",  --lw $1, 0($6)
X"8CA20000" , --lw $2, 0($5)
X"8c010001",  --lw $1, 1($0)   ->  $1 = data_mem[1]
X"ACA40000",  --sw $4,0($5)
X"20050002",  --addi $5, $0, 2
X"20060001",  -- addi $6, $0, 1
X"20A50001",  --addi $5, $5, 1
X"20C60001",  --addi $6, $6, 1
X"20030014",  --addi $3, $0, 20 (loop count : $3 = 20)
X"00412020",  --R_Type : add $4, $2, $1
X"00253022" , -- R_Type :sub $6,$1,$5
X"00879824",  -- R_Type:and : $4,$7, &19
X"00242825" ,  --R_Type:or:$1,$4,$5
X"114B0001"  --beq $10,$s11,8
--X"1460FFF8"   --bne $3,$0,-8
);

begin
 process(address)

begin
 instruction <= sig_memory_instr(to_integer(unsigned(address)));
 end process;
end BEHAV;
```



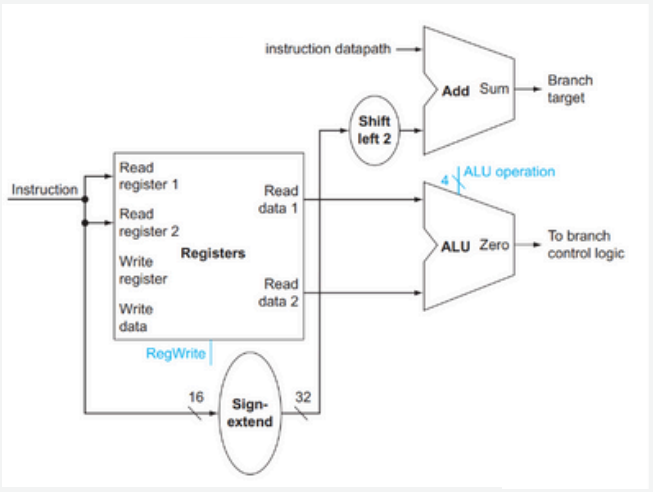
Data Memory

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity data_memory is
   generic (
      nbit_width : integer := 32);
   Port (
 clk : in std_logic;
      address : in std_logic_vector(nbit_width-1 downto 0);
      write_data : in std_logic_vector(nbit_width-1 downto 0);
      memory_write_ctrl , memory_read_ctrl : in std_logic;
      read_data : out std_logic_vector(nbit_width-1 downto 0) );
end data_memory;

architecture BEHAV of data_memory is
   type datamemory_mem is array (0 to 19) of std_logic_vector(nbit_width-1 downto 0);
   signal data_mem : datamemory_mem := (
    x"00000011",
    x"00111000",
    x"00100001",
    x"01000001",
    x"00111100",
    x"01111000",
    x"10000001",
    x"00011000",
    x"00000000",
    x"01000010",
    x"01100000",
    x"00100101",
    x"00000011",
    x"00000000",
    x"00000000",
    x"01111100",
    x"00000000",
    x"00111111",
    x"01000011",
    x"00011100"
    );
begin
   process(clk,address)
   begin
    if (memory_read_ctrl = '1') then
          read_data <= data_mem(to_integer(unsigned(address)));
       end if;
       if (memory_write_ctrl = '1') then
          data_mem(to_integer(unsigned(address))) <= write_data;
       end if;
   end process;
end BEHAV;
```

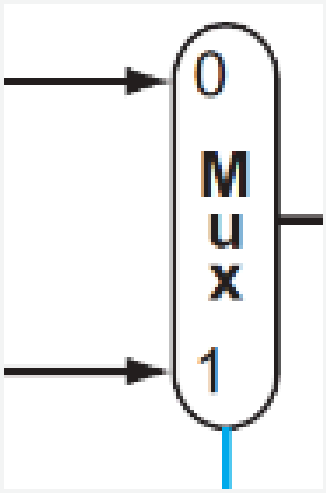


Shift Left

```
library ieee;
use ieee.std_logic_1164.all;


entity shift_left_2 is
   Port (
  input : in  std_logic_vector (31 downto 0);
       output : out  std_logic_vector (31 downto 0)
 );
end shift_left_2;

architecture BEHAV of shift_left_2 is
begin
    output <= input(29 downto 0)&"00";
end BEHAV;
```



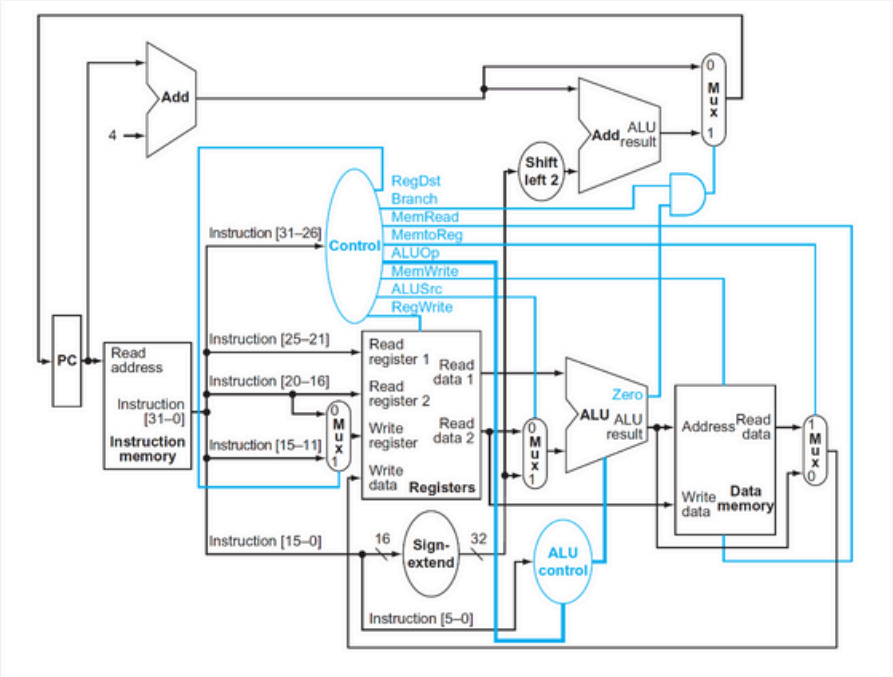
Mux




```
library ieee;
use ieee.std_logic_1164.all;

entity mux is
   generic(
        nbit_width : integer := 32
       );
   Port (
   sel : in  std_logic;
       input0,input1 : in  std_logic_vector (nbit_width-1 downto 0);
       output : out  std_logic_vector (nbit_width-1 downto 0)
 );
end mux;

architecture BEHAV of mux is
begin
 output <= input0 when (sel = '0') else input1;
end BEHAV;
```

Alu

```
lilibrary ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity alu is
   generic (
      nbit_width : integer := 32
   );
   Port (
      opcode : in std_logic_vector(2 downto 0);
      input1, input2 : in std_logic_vector(nbit_width-1 downto 0);
      result : out std_logic_vector(nbit_width-1 downto 0);
      zero : out std_logic
   );
end alu;

architecture BEHAV of alu is
   signal temp_result : std_logic_vector(nbit_width downto 0);
begin
 -- with opcode select
    --   temp_result <= ('0' & input1) + ('0' & input2) when "011",  -- Addition
                --  ('0' & input1) - ('0' & input2) when "100",  -- Subtraction
                 --('0' & input1) and ('0' & input2)when "101",  -- AND
                --('0' & input1) or ('0' & input2) when "110",  -- OR
                 --  ('0' & input1) + ('0' & input2) when "001",  -- calculate address for LW / SW
                  --('0' & input1) + ('0' & input2) when others;  -- addi

  --zero <= '1' when (temp_result = '0' & x"00000000") else '0';

  --result <= temp_result(nbit_width-1 downto 0);  -- ignore the carry

  process(opcode, input1,input2)

  begin

 case opcode is

  --when "011" => -- addition

  --temp_result <= ('0' & input1) + ('0' & input2);

  --zero <= '0';


  when "100" => -- subtraction

  temp_result <= ('0' & input1) - ('0' & input2);

  zero <= '0';
  when "101" => -- and

  temp_result <= ('0' & input1) and  ('0' & input2);

  zero <= '0';
  when "000" => -- Nand

  temp_result <= ('0' & input1) Nand  ('0' & input2);

  zero <= '0';
  when "011" => --slt

  if(input1<input2)then
     temp_result <= (0 => '1' , others => '0');
   zero <= '0';
  else
     temp_result <= ( others => '0');
   zero <= '0';
   end if;

  when "110" => -- or

  temp_result <= ('0' & input1) or ('0' & input2);

  zero <= '0';


  when "001" => -- lw sw

  temp_result <=  ('0' & input1) + ('0' & input2);

  zero <= '0';


  when "111" => --addi

  temp_result <= ('0' & input1) + ('0' & input2);

  zero <= '0';

 when others =>  --beq/bneq   --check with instruction


if(input1=input2)then


  zero <= '1';

  else


  zero <= '0';

  end if;

  end case;

 end process;

  -- ignore the carry
  result <= temp_result(nbit_width-1 downto 0);


end BEHAV;
```



Alu Control

```
lilibrary ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity control_value is
   Port (
  opcode : in  std_logic_vector(5 downto 0);
  RegDst , Branch , MemRead , MemtoReg , MemWrite , AluSrc , RegWrite : out  std_logic;
       ALUOp : out  std_logic_vector (1 downto 0)
 );
end control_value;

architecture BEHAV of control_value is
 signal ctrl_values: std_logic_vector(8 downto 0);
begin

process(opcode)
    begin
 case opcode is
   when "000000" => ctrl_values <= "100000110";  -- R type  D"0"
     when "100011" => ctrl_values <= "001101100";  -- lw  D"35"
     when "101011" => ctrl_values <= "-00-11000";  -- sw  D"43"
     when "000101" => ctrl_values <= "-10-00001";  -- bneq  D"5"
   when "000100" => ctrl_values <= "-10-00001";  -- beq  D"4"
     when "001000" => ctrl_values <= "000001111";  -- addi  D"8"
     when others   => ctrl_values <= "---------";  --others
 end case;
    end process;

 RegDst <= ctrl_values(8);
 Branch <= ctrl_values(7);
 MemRead <= ctrl_values(6);
 MemtoReg <= ctrl_values(5);
 MemWrite <= ctrl_values(4);
 AluSrc <= ctrl_values(3);
 RegWrite <= ctrl_values(2);
 ALUOp <= ctrl_values(1 downto 0);

end BEHAV;
```

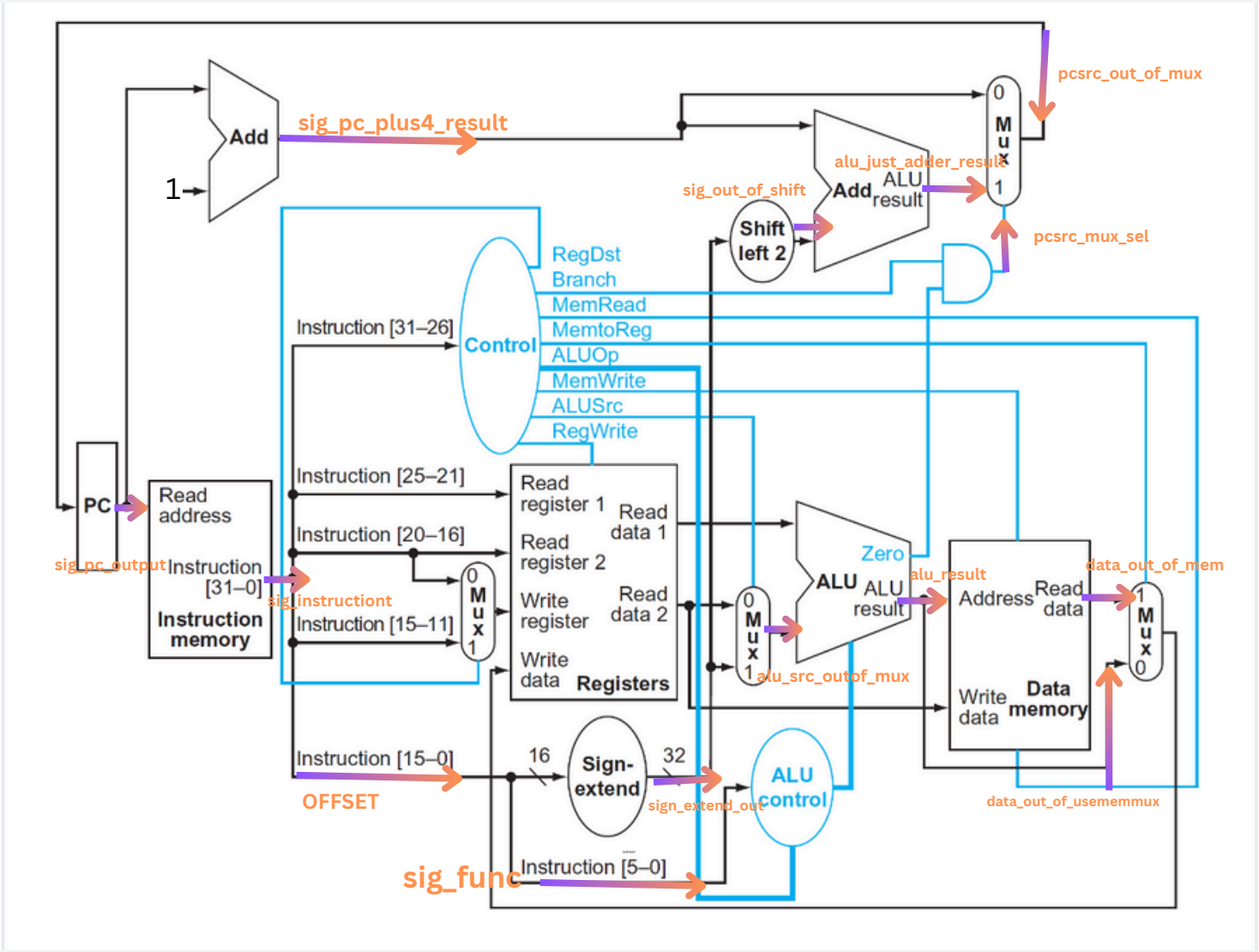
Control Unit

```
library ieee;
use ieee.std_logic_1164.all;

entity control_unit is
   Port (
  opcode , funct : in  std_logic_vector (5 downto 0);
           RegDst , Branch , MemRead , MemtoReg , MemWrite ,
AluSrc , RegWrite : out  std_logic;
       ALUOp : out  std_logic_vector (2 downto 0)
 );
end control_unit;

architecture BEHAV of control_unit is

 COMPONENT control_value
   PORT(
  opcode : in  std_logic_vector(5 downto 0);
    RegDst , Branch , MemRead , MemtoReg , MemWrite , AluSrc ,
RegWrite : out  std_logic;
       ALUOp : out  std_logic_vector (1 downto 0)
 );
 END COMPONENT;

 COMPONENT alu_control
    PORT(
  ALUOp : in  std_logic_vector (1 downto 0);
        funct : in  std_logic_vector (5 downto 0);
   ALU_ctrl : out  std_logic_vector (2 downto 0)
 );
 END COMPONENT;

 signal alu_opcode:std_logic_vector(1 downto 0);

begin

Ucontrol_value : control_value PORT MAP(
    opcode => opcode,
    RegDst => RegDst,
    Branch => Branch,
    MemRead => MemRead,
    MemtoReg => MemtoReg,
    MemWrite => MemWrite,
    AluSrc => AluSrc,
    RegWrite => RegWrite,
    ALUOp => alu_opcode
        );

Ualu_control : alu_control PORT MAP(
       ALUOp => alu_opcode,
       funct => funct,
       ALU_ctrl => ALUOp
      );

end BEHAV;
```

Control Value

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity control_value is
   Port (
  opcode : in  std_logic_vector(5 downto 0);
  RegDst , Branch , MemRead , MemtoReg , MemWrite , AluSrc , RegWrite : out  std_logic;
       ALUOp : out  std_logic_vector (1 downto 0)
 );
end control_value;

architecture BEHAV of control_value is
 signal ctrl_values: std_logic_vector(8 downto 0);
begin

process(opcode)
    begin
 case opcode is
  when "000000" => ctrl_values <= "100000110";  -- R type  D"0"
     when "100011" => ctrl_values <= "001101100";  -- lw  D"35"
     when "101011" => ctrl_values <= "-00-11000";  -- sw  D"43"
     --when "000101" => ctrl_values <= "-10-00001";  -- bneq  D"5"
   when "000100" => ctrl_values <= "010000001";  -- beq  D"4"
     when "001000" => ctrl_values <= "000001111";  -- addi  D"8"
     when others   => ctrl_values <= "---------";
 end case;
    end process;

 RegDst <= ctrl_values(8);
 Branch <= ctrl_values(7);
 MemRead <= ctrl_values(6);
 MemtoReg <= ctrl_values(5);
 MemWrite <= ctrl_values(4);
 AluSrc <= ctrl_values(3);
 RegWrite <= ctrl_values(2);
 ALUOp <= ctrl_values(1 downto 0);

end BEHAV;
```

datapath




```
library ieee;
use ieee.std_logic_1164.all;


entity datapath is
 generic (
          nbit_width : integer := 32
    );
    Port (
     clk : in  std_logic
     );
end datapath;
```

```
architecture BEHAV of datapath is

COMPONENT pc
 Port (
          clk : in  std_logic;
          input : in  std_logic_vector(nbit_width-1 downto 0);
         output : out  std_logic_vector(nbit_width-1 downto 0)
          );
END COMPONENT;

COMPONENT pc_adder
 Port (
          input : in  std_logic_vector(nbit_width-1 downto 0);
              output : out  std_logic_vector(nbit_width-1 downto 0)
       );
END COMPONENT;

COMPONENT instruction_memory
 Port (
          address : in std_logic_vector(nbit_width-1 downto 0);
              instruction : out std_logic_vector(nbit_width-1 downto 0)
         );
END COMPONENT;

COMPONENT control_unit
  Port (
    opcode , funct : in  std_logic_vector (5 downto 0);
         RegDst , Branch , MemRead , MemtoReg , MemWrite , AluSrc , RegWrite : out  std_logic;
           ALUOp : out  std_logic_vector (2 downto 0)
   );
END COMPONENT;

COMPONENT register_File
 Port (
     clk : in std_logic;
        read_register1, read_register2 : in std_logic_vector(4 downto 0);
         write_register : in std_logic_vector(4 downto 0);
         write_data : in std_logic_vector(nbit_width-1 downto 0);
         register_write_ctrl : in std_logic;
          read_data1, read_data2 : out std_logic_vector(nbit_width-1 downto 0)
       );
END COMPONENT;

COMPONENT sign_extend
 Port (
          input : in  std_logic_vector (15 downto 0);
          output : out  std_logic_vector (nbit_width-1 downto 0)
     );
END COMPONENT;
```

```
COMPONENT shift_left_2
Port (
input : in std_logic_vector (31 downto 0);
output : out std_logic_vector (31 downto 0)
);
END COMPONENT;

COMPONENT alu
Port (
opcode : in std_logic_vector(2 downto 0);
input1,input2 : in std_logic_vector(nbit_width-1 downto 0);
result : out std_logic_vector(nbit_width-1 downto 0);
zero : out std_logic
);
END COMPONENT;

COMPONENT data_memory
Port (
clk : in std_logic;
address : in std_logic_vector(nbit_width-1 downto 0);
write_data : in std_logic_vector(nbit_width-1 downto 0);
memory_write_ctrl , memory_read_ctrl : in std_logic;
read_data : out std_logic_vector(nbit_width-1 downto 0)
);
END COMPONENT;

COMPONENT mux generic (
nbit_width : integer := 32
);
Port (
sel : in std_logic;
input0,input1 : in std_logic_vector (nbit_width-1 downto 0);
output : out std_logic_vector (nbit_width-1 downto 0)
);
END COMPONENT;
```

```
--dn
signal
sig_MemtoReg,sig_Branch,sig_AluSrc,sig_RegDst,sig_RegWrite,sig_zero,sig_MemRead,sig_MemW
rite : std_logic;
--dn
signal sig_ALUOp : std_logic_vector(2 downto 0);
--dn
signal sig_pc_output: std_logic_vector(31 downto 0);
--dn
signal sig_instruction: std_logic_vector(31 downto 0);
--dn
signal sig_pc_plus4_result : std_logic_vector(31 downto 0);
--dn
signal sig_write_register : std_logic_vector(4 downto 0);
--dn
signal sig_read_data1,sig_read_data2: std_logic_vector(31 downto 0);
--dn
signal sign_extend_out : std_logic_vector(31 downto 0);
--dn
signal data_out_of_usememmux : std_logic_vector(31 downto 0);
--dn
signal alu_src_outof_mux,alu_result,alu_just_adder_result : std_logic_vector(31 downto 0);
--dn
signal data_out_of_mem : std_logic_vector(31 downto 0);
--dn
signal sig_out_of_shift : std_logic_vector(31 downto 0);
--dn
signal pcsrc_mux_sel : std_logic;
--dn
signal pcsrc_out_of_mux : std_logic_vector(31 downto 0);



-- define each part of instruction with alias
--dn all
alias sig_opcode : std_logic_vector(5 downto 0) is sig_instruction(31 downto 26);
alias sig_funct : std_logic_vector(5 downto 0) is sig_instruction(5 downto 0);
alias rs : std_logic_vector(4 downto 0) is sig_instruction(25 downto 21);
alias rt : std_logic_vector(4 downto 0) is sig_instruction(20 downto 16);
alias rd : std_logic_vector(4 downto 0) is sig_instruction(15 downto 11);
alias shamt : std_logic_vector(4 downto 0) is sig_instruction(10 downto 6);
alias offset : std_logic_vector(15 downto 0) is sig_instruction(15 downto 0);
```

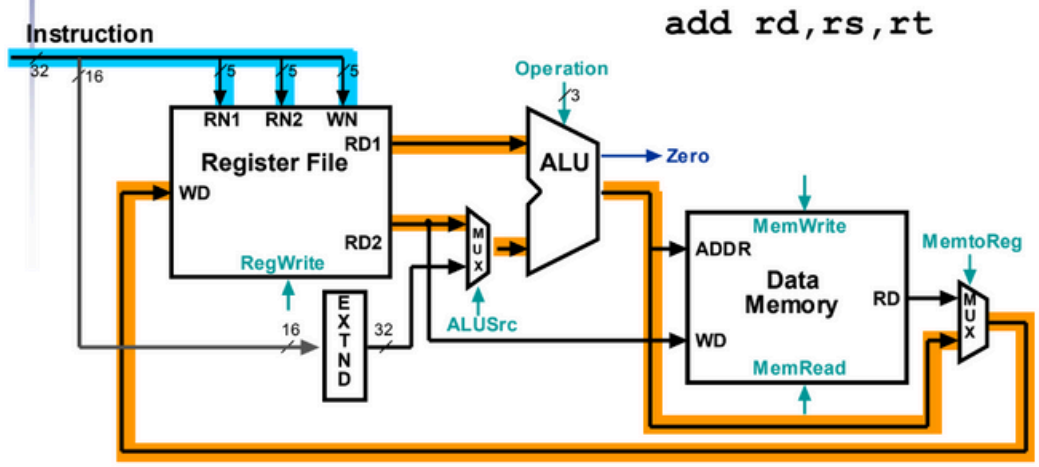
```
begin

 U_control_unit: control_unit PORT MAP(
  opcode => sig_opcode,
  funct => sig_funct,
  RegDst => sig_RegDst,
  Branch => sig_Branch,
  MemRead => sig_MemRead,
  MemtoReg => sig_MemtoReg,
  MemWrite => sig_MemWrite,
  AluSrc => sig_AluSrc,
  RegWrite => sig_RegWrite,
  ALUOp => sig_ALUOp );

 U_PC: pc PORT MAP(
  clk => clk,
  input => pcsrc_out_of_mux,
  output => sig_pc_output );

 U_instruction_memory: instruction_memory PORT MAP(
  address => sig_pc_output,
  instruction => sig_instruction );

 U_pc_adder: pc_adder PORT MAP(
  input => sig_pc_output,
  output => sig_pc_plus4_result );

 U_mux_rtORrd: mux generic map (
     nbit_width => 5
  ) port map (
     sel => sig_RegDst,
      input0 => rt,
      input1 => rd,
     output => sig_write_register );

 U_register_file: register_file PORT MAP(
  clk => clk,
  register_write_ctrl => sig_RegWrite,
  read_register1 => rs,
  read_register2 => rt,
  write_register => sig_write_register,
  read_data1 => sig_read_data1,
  read_data2 => sig_read_data2,
  write_data => data_out_of_usememmux );

 U_sign_extend: sign_extend PORT MAP(
  input => offset,
  output => sign_extend_out  );

 U_shift_left_2 :shift_left_2 PORT MAP (
       input => sign_extend_out,
                output => sig_out_of_shift      );

 U_alu_just_adder: alu PORT MAP(
  opcode => "011",  -- means it should add
  input1 => sig_pc_plus4_result,
       input2 => sig_out_of_shift,
  --input2 => sign_extend_out,
       zero => sig_zero,  -- the result is not zero
  result => alu_just_adder_result );
```

```
U_mux_pcsrc: mux port map (
            sel => pcsrc_mux_sel,
         input0 => sig_pc_plus4_result,
         input1 => alu_just_adder_result,
  output => pcsrc_out_of_mux      );

U_mux_alusrc: mux port map (
            sel => sig_AluSrc,
         input0 => sig_read_data2,
         input1 => sign_extend_out,
  output => alu_src_outof_mux       );

U_alu_main: alu PORT MAP(
 opcode => sig_ALUOp,
 input1 => sig_read_data1,
    input2 => alu_src_outof_mux,
 zero=>sig_zero,
 result => alu_result );
 pcsrc_mux_sel <= sig_Branch and sig_zero;

U_data_memory: data_memory PORT MAP(
 clk => clk,
          address => alu_result,
       write_data => sig_read_data2,
      memory_write_ctrl => sig_MemWrite,
 memory_read_ctrl => sig_MemRead,
      read_data => data_out_of_mem );


 U_mux_usemem: mux PORT MAP (
              sel => sig_MemtoReg,
          input0 => alu_result,
          input1 => data_out_of_mem,
   output => data_out_of_usememmux       );

end BEHAV;
```

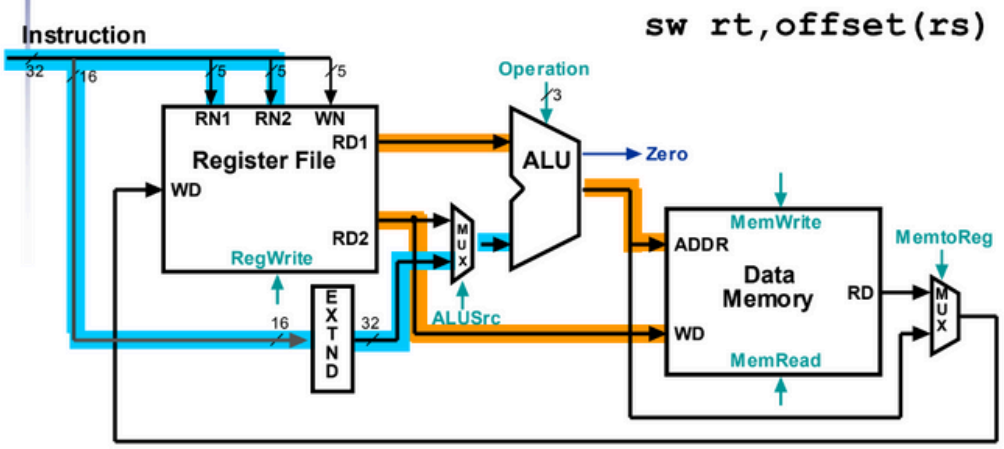
tb_datapath

```
library IEEE;
use IEEE.std_logic_1164.all;

entity tb_datapath is
end tb_datapath;

architecture BEHAV of tb_datapath is

 COMPONENT datapath
  Port (
            clk : in  std_logic
           );
 END COMPONENT;

signal clock : std_logic;

begin

 U_datapath: datapath PORT MAP (
            clk => clock
          );

process
   begin
  for i in 0 to 21 loop  --loop count
       clock <= '1';
  wait for 50 ns;
  clock <= '0';
 wait for 50 ns;
  end loop;
end process;

end BEHAV;

end BEHAV;
```

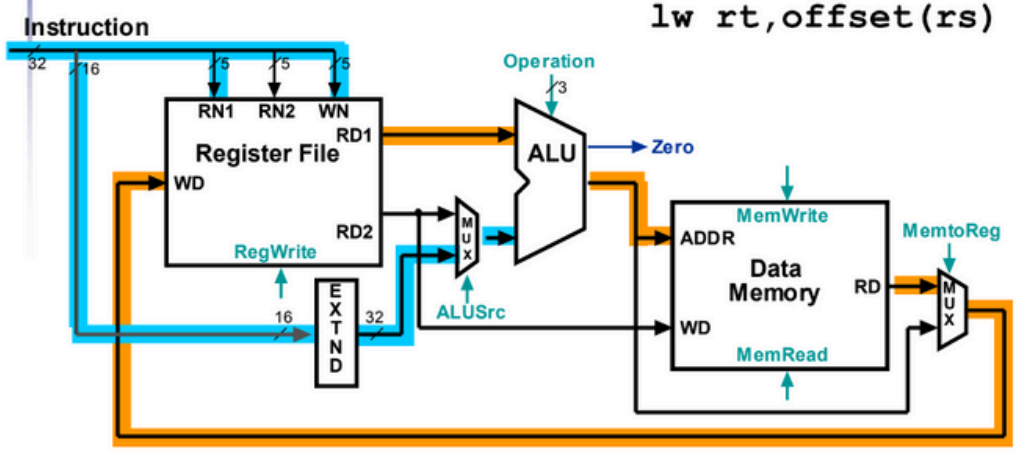
Animating the Datapath: R-type Instruction



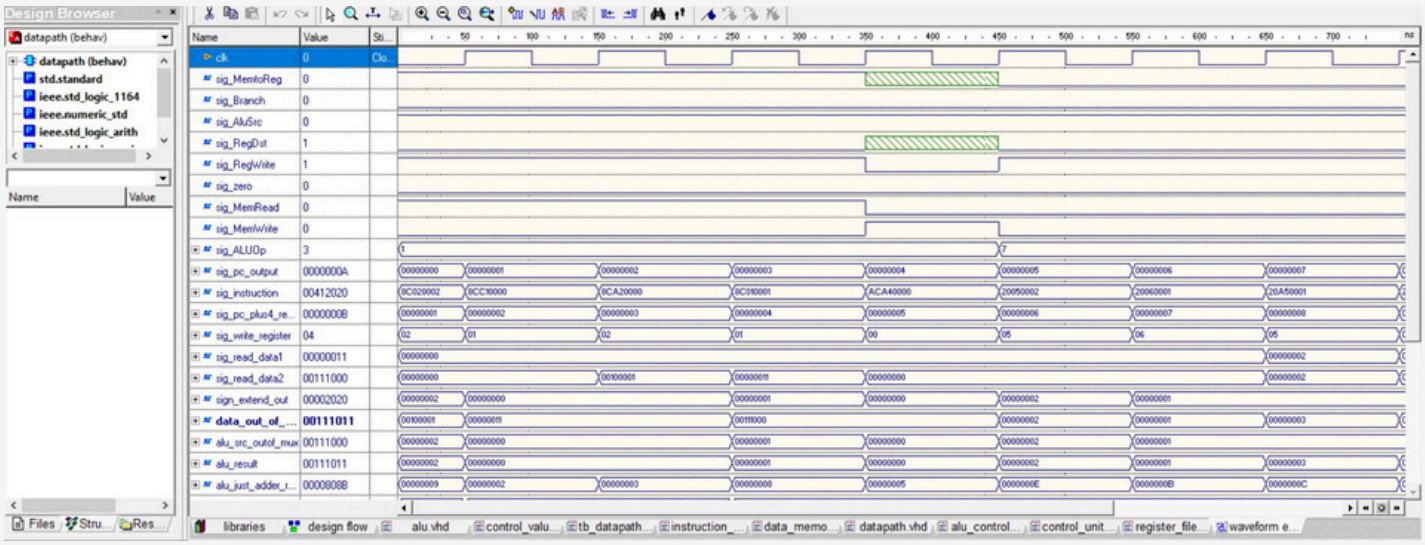
Animating the Datapath: Store Instruction

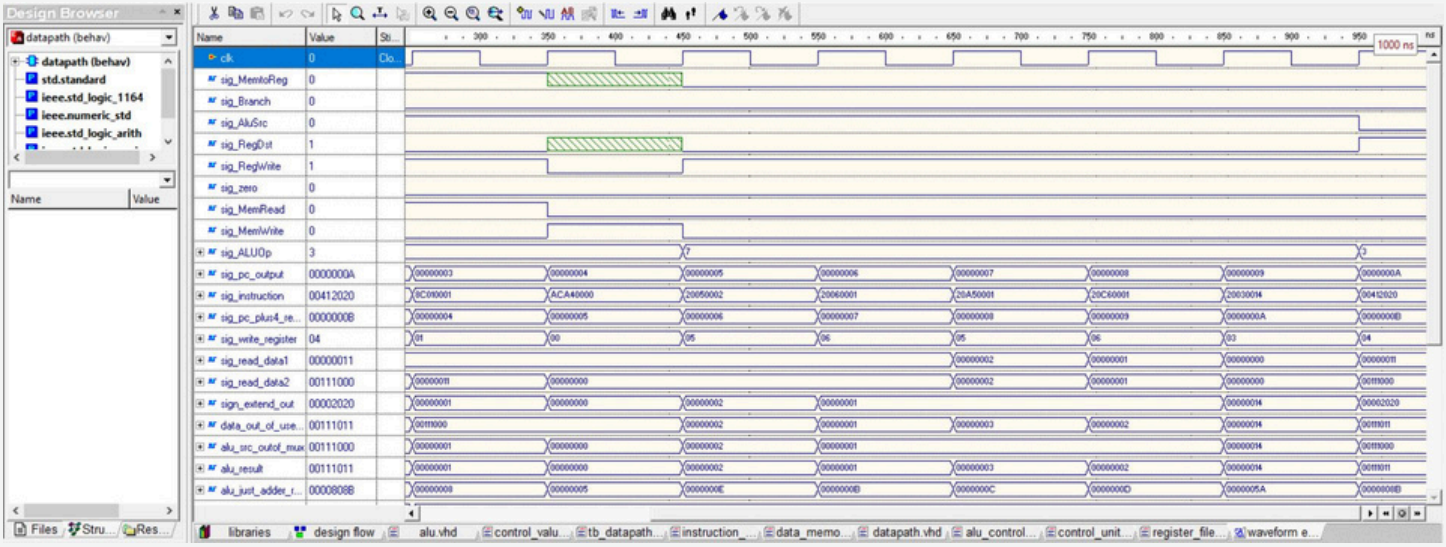


Animating the Datapath: Load Instruction

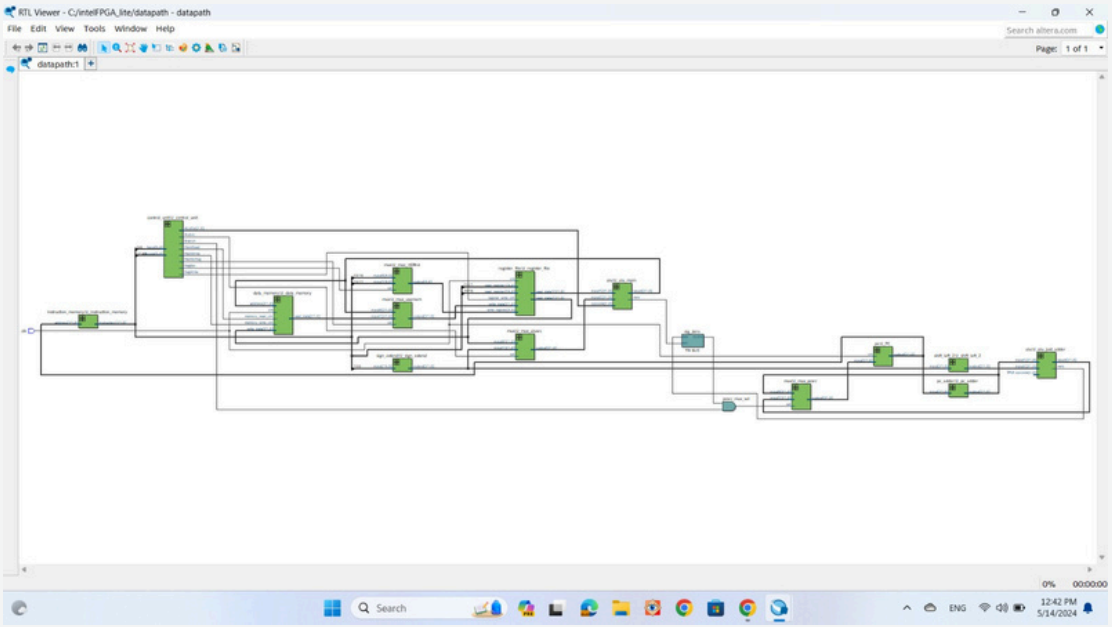


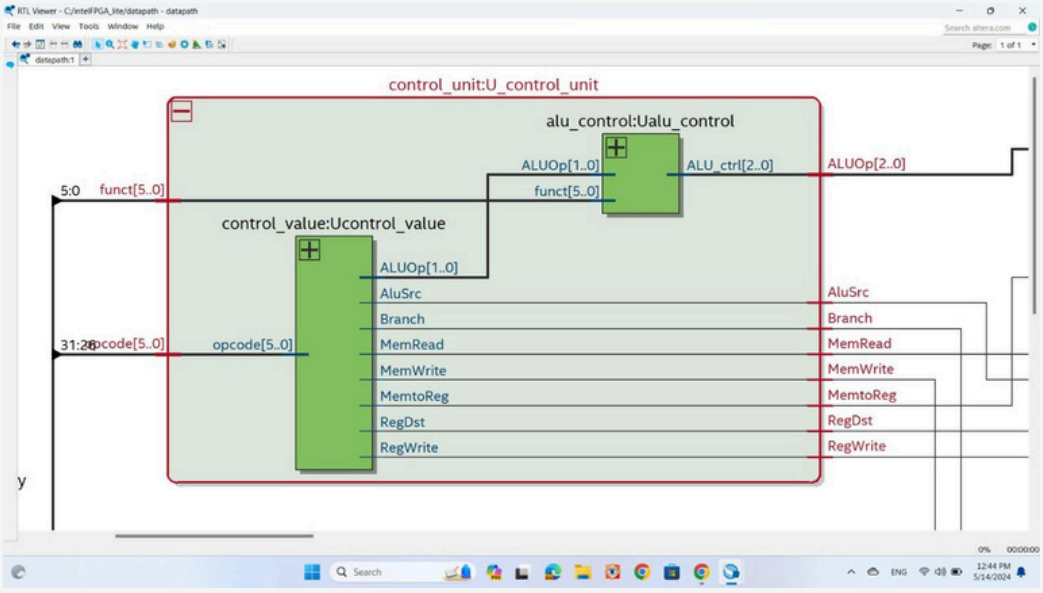
Run in Active Hdl

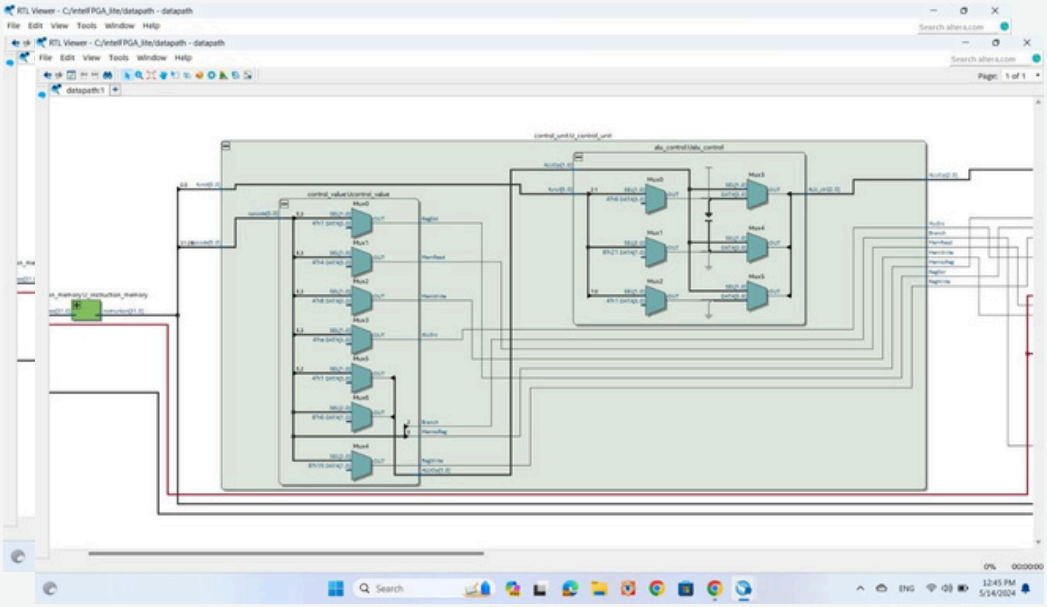


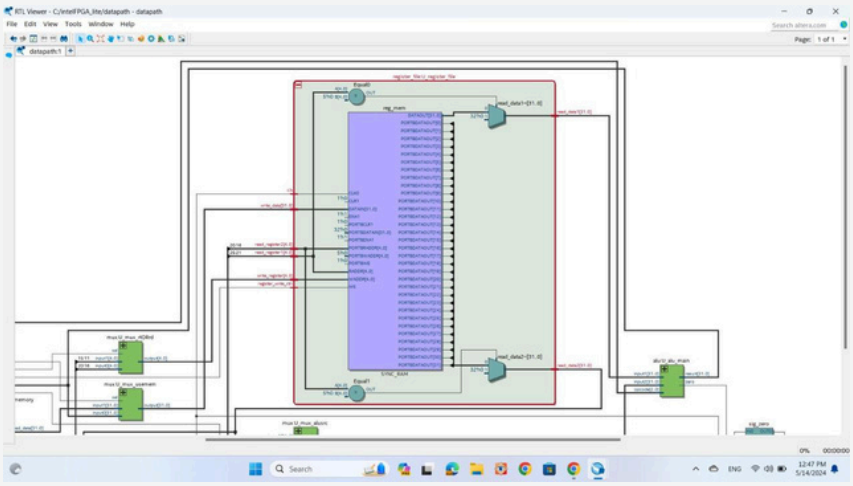


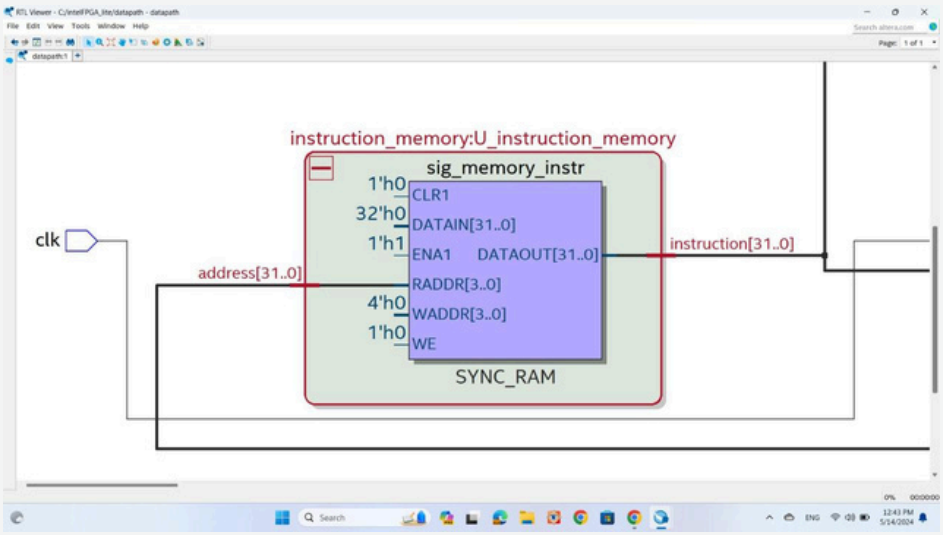
Schematic diagram

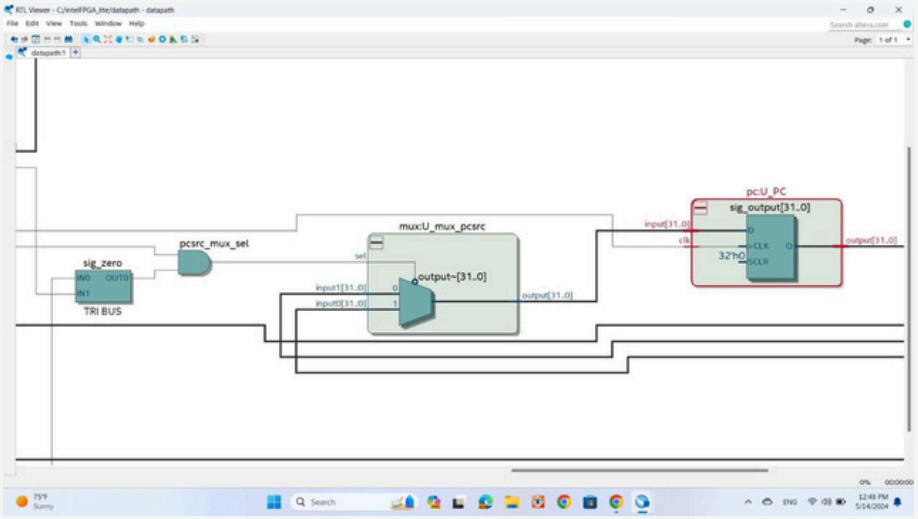


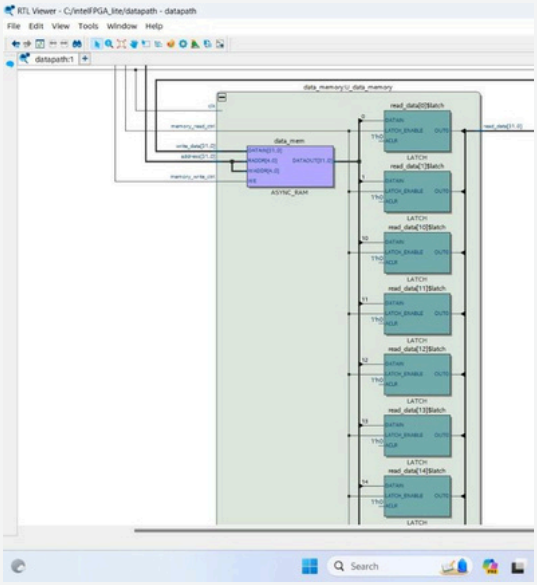


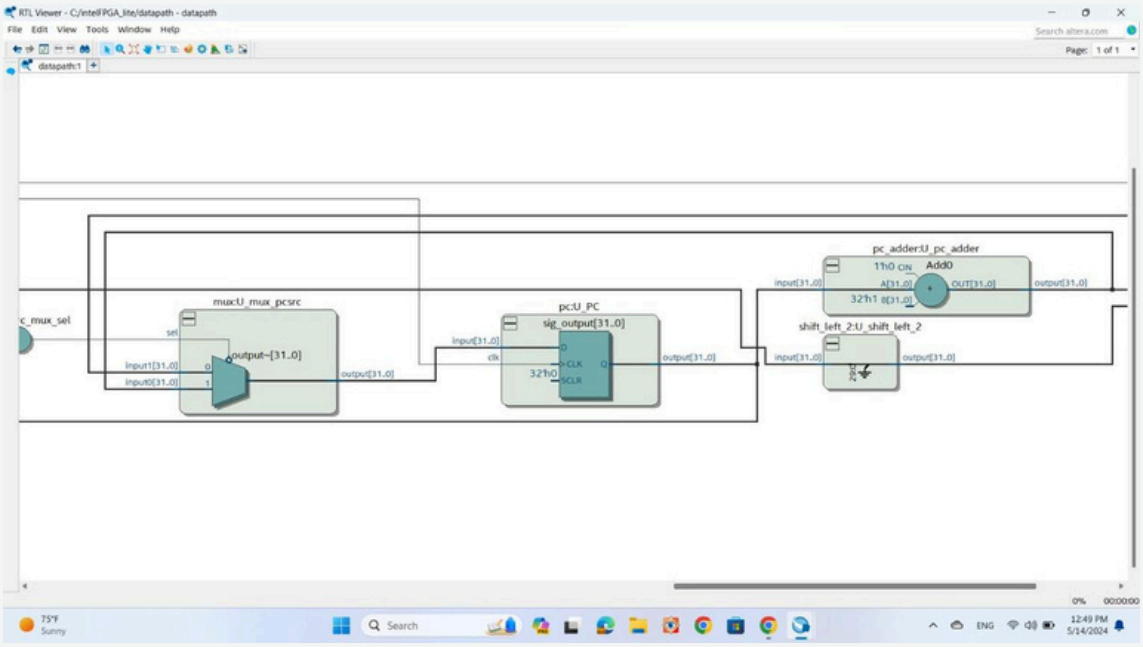


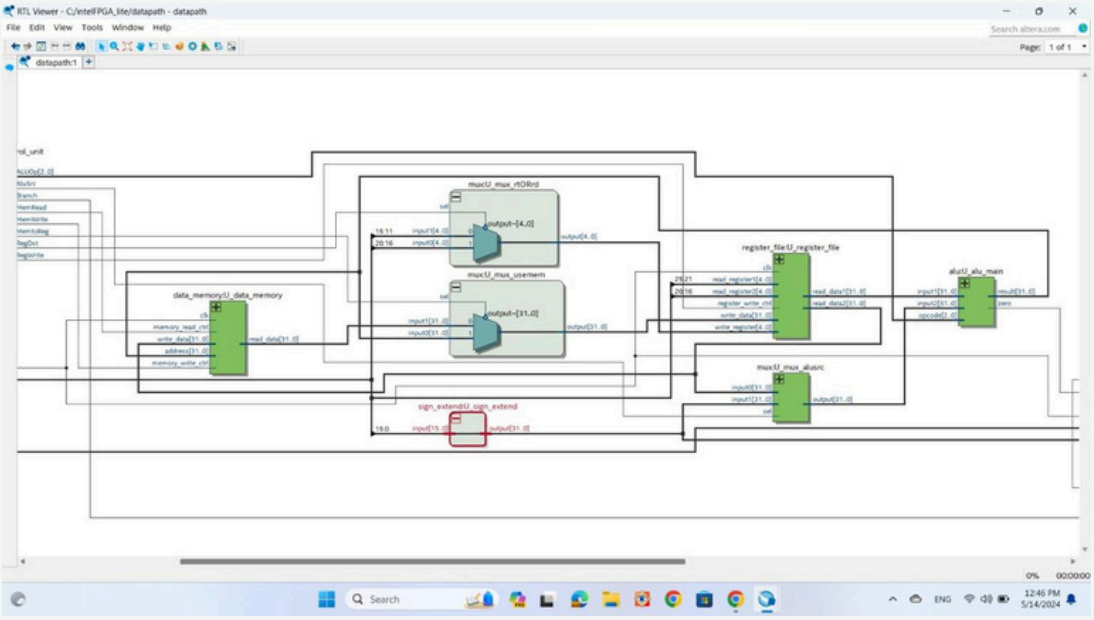


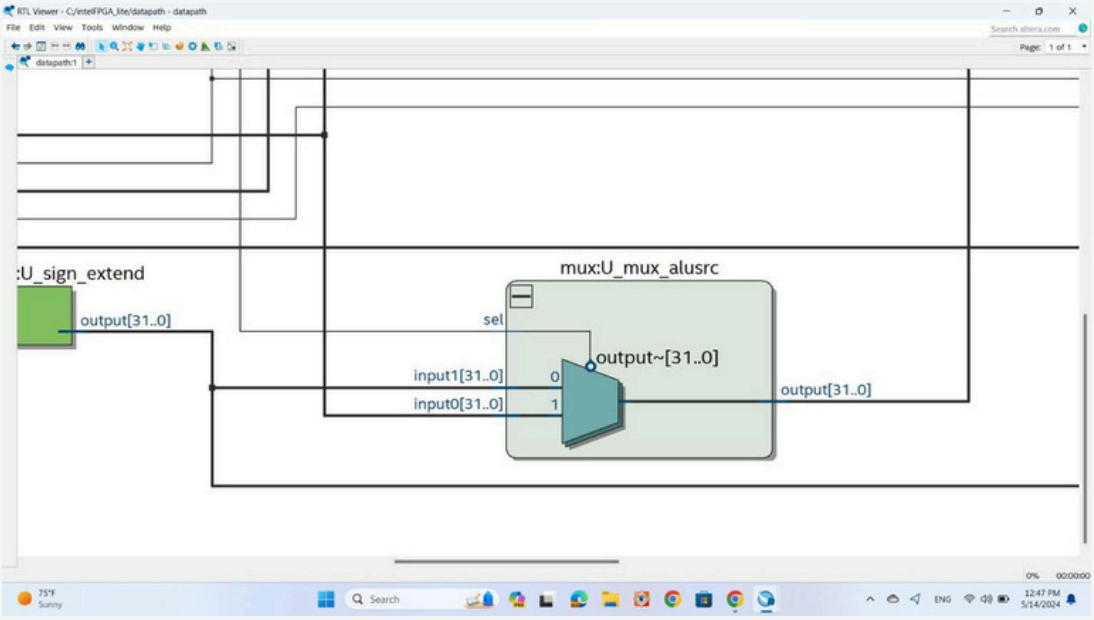

instruction_memory:U_instruction_memory
sig_memory_instr
1'h0
CLR1
32'h0
DATAIN[31..0]
1'h1
ENA1
DATAOUT[31..0]
RADDR[3..0]
4'h0
WADDR[3..0]
1'h0
WE
SYNC_RAM
clk
address[31..0]
instruction[31..0]


pc:U_PC
sig_output[31..0]
32'h0
CLK
SCLR
mux:U_mux_pcsrc
output~[31..0]
pcsrc_mux_sel
sig_zero
TRI BUS



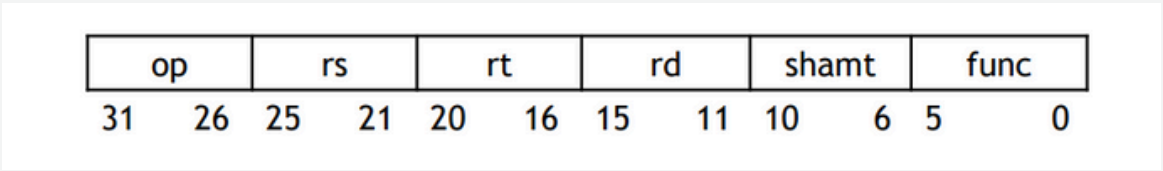

RTL Viewer - C:/intelFPGA_lite/datapath - datapath
File Edit View Tools Window Help
datapath:1
pc_adder:U_pc_adder
Add0
mux:U_mux_pcsrc
output~[31..0]
input1[31..0]
input0[31..0]
pc:U_PC
sig_output[31..0]
shift_left_2:U_shift_left_2
c_mux_sel


RTL Viewer - C:/intelFPGA_lite/datapath - datapath
File Edit View Tools Window Help
datapath:1
data_memory:U_data_memory
register_file:U_register_file
alu:U_alu_main
mux:U_mux_alusrc
sign_extend:U_sign_extend


RTL Viewer - C:/intelFPGA_lite/datapath - datapath
File Edit View Tools Window Help
datapath:1
:U_sign_extend
output[31..0]
mux:U_mux_alusrc
sel
output~[31..0]
input1[31..0]
input0[31..0]
0
1
output[31..0]

Instruction type

R-type



add $4 ,$2,$1

(0) Decimal	$1	$2	$4	shamt	Function (add)
000000	00001	00010	00100	00000	100 000

x “ 00412020 ”

sub $6 ,$1,$5

(0) Decimal	$1	$5	$6	shamt	Function (Sub)
000000 0	0000 1	00101	00110	00000	100010

x “ 00253022 ”

and $4 ,$7,$13

(0) Decimal	**$4**	$7	$13	shamt	Function (and)
000 000	00100	00111	01101	00000	100 100

x “ 00876824”

nand $4 ,$7,$13

op (0) Decimal	**$4**	$7	$13	shamt	Function (nand)
000 000	00100	00111	01101	00000	100 111

x “ 00876827”

or $1 ,$4,$5

op (0) Decimal	**$1**	$4	$5	shamt	Function (or)
000 000	00001	00100	00101	00000	100 101

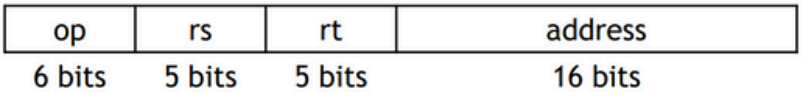
x “ 00242825”

set on less than $4 ,$2,$3

(0) Decimal	**$2**	$3	$4	shamt	Function (slt)
000 000	00010	00011	00100	00000	111 000

x “ 00432038”

I-type



LW $2,2($0)

35 Decimal	$0	$2	“address” 2
100011	00000	00010	0000000000001010

x “ 8c020002 ”

SW $4,0($5)

43 Decimal	$5	$4	“address” 0
101011	00101	00100	0000000000000000

x “ ACA40000 ”

add i $5,$5,1

8 Decimal	$5	$2	“immediant” 1
001000	00101	00101	0000000000000001

x “20A50001”

beq $10,$s11,8

4 Decimal	$10	$11	“address” 1
00100	01010	01011	0000000000000001

x “114B0001”

bneq $3,$0,-8

5 Decimal	$3	$0	“address” 8
00101	00011	00000	0000000011111000

x “1460FFF8”

Address	Instruction	Machine Code
00	LW $2,2($0)	8c020002
01	LW $1,0($6)	8CC10000
02	LW $2,0($5)	8CA20000
03	LW $1,1($0)	8c010001
04	ADDI $5,$0,2	20050002
05	SW $4,0($5)	ACA40000
06	ADDI $6,$0,$1	20060001
07	ADDI $5,$5,$1	20A50001
08	ADDI $6,$6,$1	20C60001
09	ADDI $3,$0,20	20030014
10	ADD $4,$2,$1	00412020
11	SUB $6,$1,$5	00253022
12	AND $13,$4,$7	00876824
13	NAND $13,$4,$7	00876827
14	OR $1,$4,$5	00242825
15	SLT $4,$2,$3	00432038
16	BEQ $10,$s11,1	114B0001
17	BNE $3,$0,-8	1460FFF8

ALU Control

Op Code	Inputs Function (6bit)	inputs Op Code (2 bit)	Operation	Outputs Alu_Ctrl
Load word	xxxxxx	00	add	001
Store word	xxxxxx	00	add	001
bneq	xxxxxx	01	subt	010
beq	xxxxxx	01	subt	010
add i	xxxxxx	11	add	111
R Format	100 000	11	ADD	111
	100 010	10	SUB	100
	100 100	10	AND	101
	100 101	11	OR	110
	100 111	00	NAND	000
	111 000	01	SLT	011

Control Value

Inputs		
Instruction	Op Code (Decimal)	Op Code (6 bits)
R Type	0	000 000
L W	35	100 011
S W	43	101 011
add i	8	001 000
bneq	5	000 101
beq	4	000 100

Out puts				
Reg Des	Branch	MemRead	MemtoReg	MemWrite
1	0	0	0	0
0	0	1	1	0
x	0	0	x	1
0	0	0	0	0
x	1	0	x	0
x	1	0	x	0

Out puts				
Alu src	Rad write	Alu op(2)	Alu op(1)	Alu op(0)
0	1	1	0	1
1	1	0	0	1
1	0	0	0	1
1	1	1	1	1
0	0	0	1	1
0	0	0	1	1

ALU

Inputs			Outputs	
input 1	input 2	Alu_Ctrl	Result	Zero
x	x	001	input1 + input2	0
x	x	111	input1 + input2	0
x	x	100	input1 - input2	0
x	x	101	input1 AND input2	0
x	x	110	input1 or input2	0
x	x	010	input1 - input2	1
x	x	000	input1 NAND input2	0
x	x	011	check if(input1<input2) result = '1' else = '0'	0