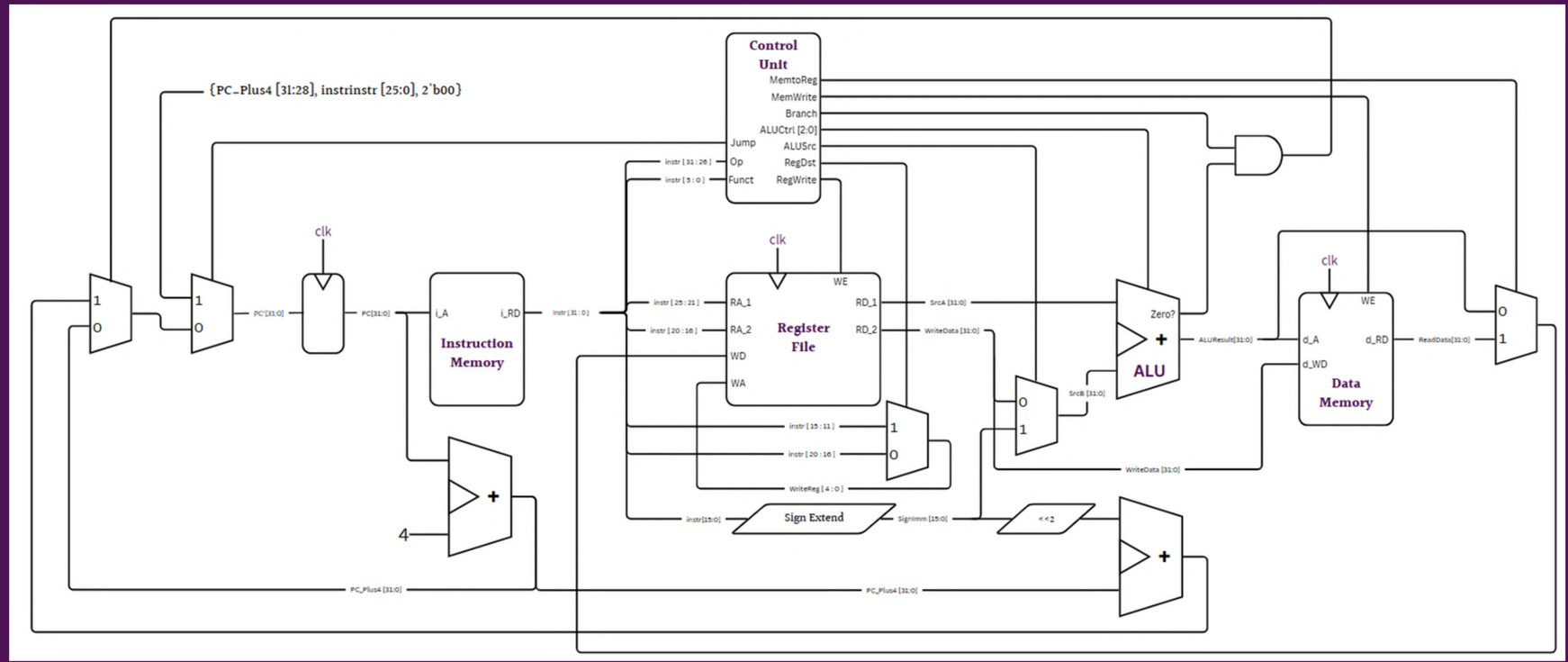
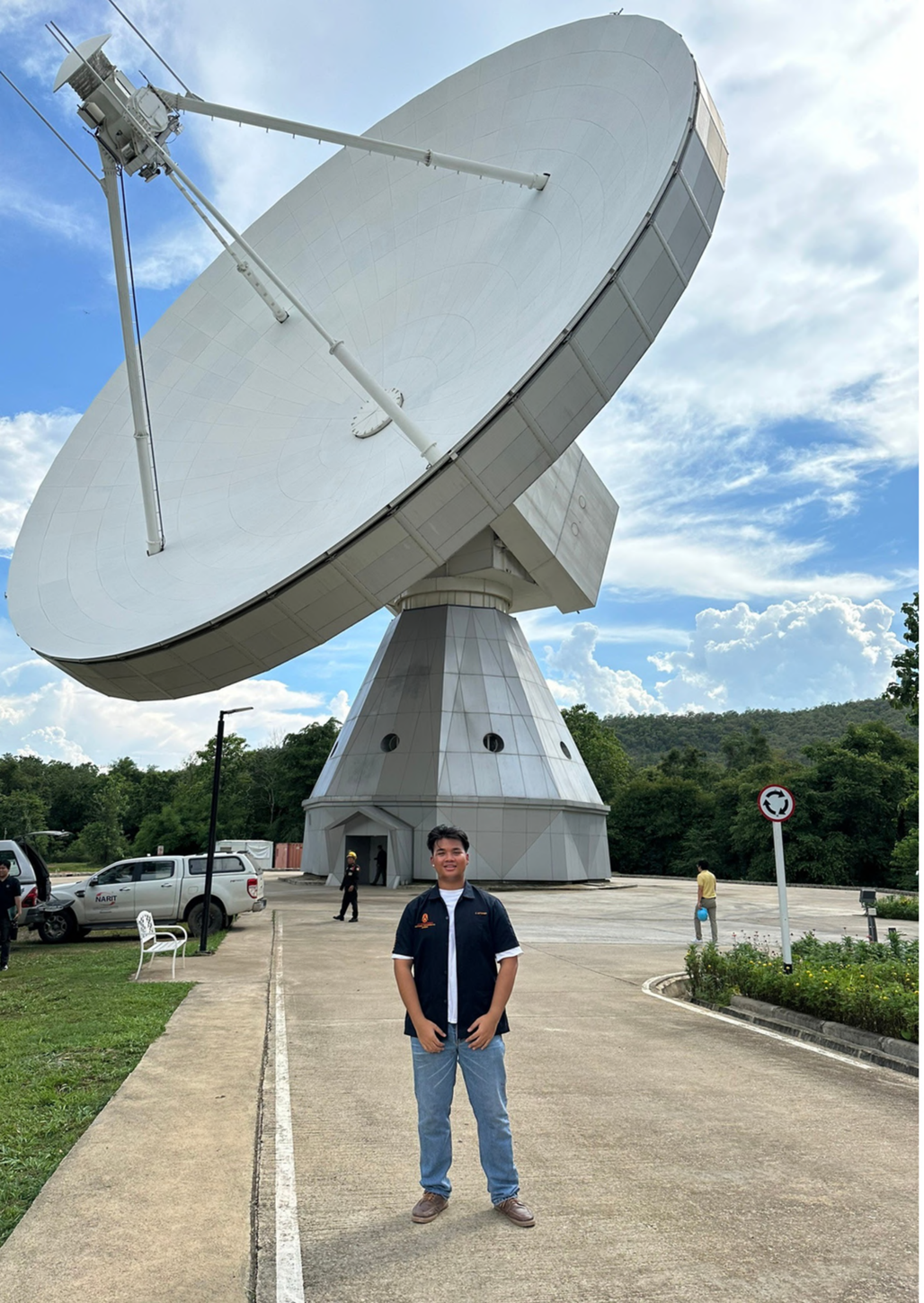


RISC CPU Design of a Single Cycle MIPS Softcore-Processor in Verilog-HDL

Date : 16 September 2025

By Kittiphop Phanthachart
(a 4th-year Engineering student, FPGA/DSP Engineer Intern @Thai Space Consortium, NARIT.)



Personal Background

Kittiphop Phanthachart , (MACKIES)

FPGA/DSP Engineer Intern @Thai Space Consortium, NARIT

Visit My Personal Website – [Click Here]

Education

2022 - Present
Bachelor of Engineering
- King Mongkut's University of Technology Thonburi (KMUTT)
- Major of Electronics and Telecommunication Engineering

2019 - 2022
Vocational Certificate
- Thai - German Pre-Engineering School @KMUTNB
- Major of Pre-Electrical & Electronics Engineering

Training & Seminars

Software Defined Radio with ADI RadioVerse™ Transceivers @Webinar
July 2025

Ansys ASEAN Semiconductor Virtual Summit 2025 @Webinar
May 2025

TI Tech Day Thailand Seminar @Bankok with Texas Instruments
April 2025

Basic Digital IC Desig Tranning @EEI with NECTEC
April 2025

RT - Thread x Renesas Workshop#2 @RMUTT , with TESA
December 2024

Digital Design with FPGA Camp @KKU, with Design Gateway
November 2024

Back-to-Basics Seminar @Bangkok , with Keysight
September 2024

Cabling Contest Training , with INTERLINK
June 2024

About Me

Hi, my name is MACKIES. I am currently a 4th-year Engineering student at KMUTT and an FPGA/DSP Engineer Intern at the Thai Space Consortium, NARIT. This project is a side initiative I developed during my internship to sharpen my RTL design skills and push myself further in Digital Design Engineering. My ultimate goal is to become an IC Design Engineer, driving innovation in advanced hardware systems. I see this project not only as practice but also as a meaningful milestone toward turning my passion for Hardware Electronic Engineering into a professional reality.

PUBLISHED

SALVS-01: Specify Animals Location Via Sound, Novel Mission Ideas for Multiple Nano-satellites The MIC8 Report, International Academy of Astronautics IAA Book Series King

Description

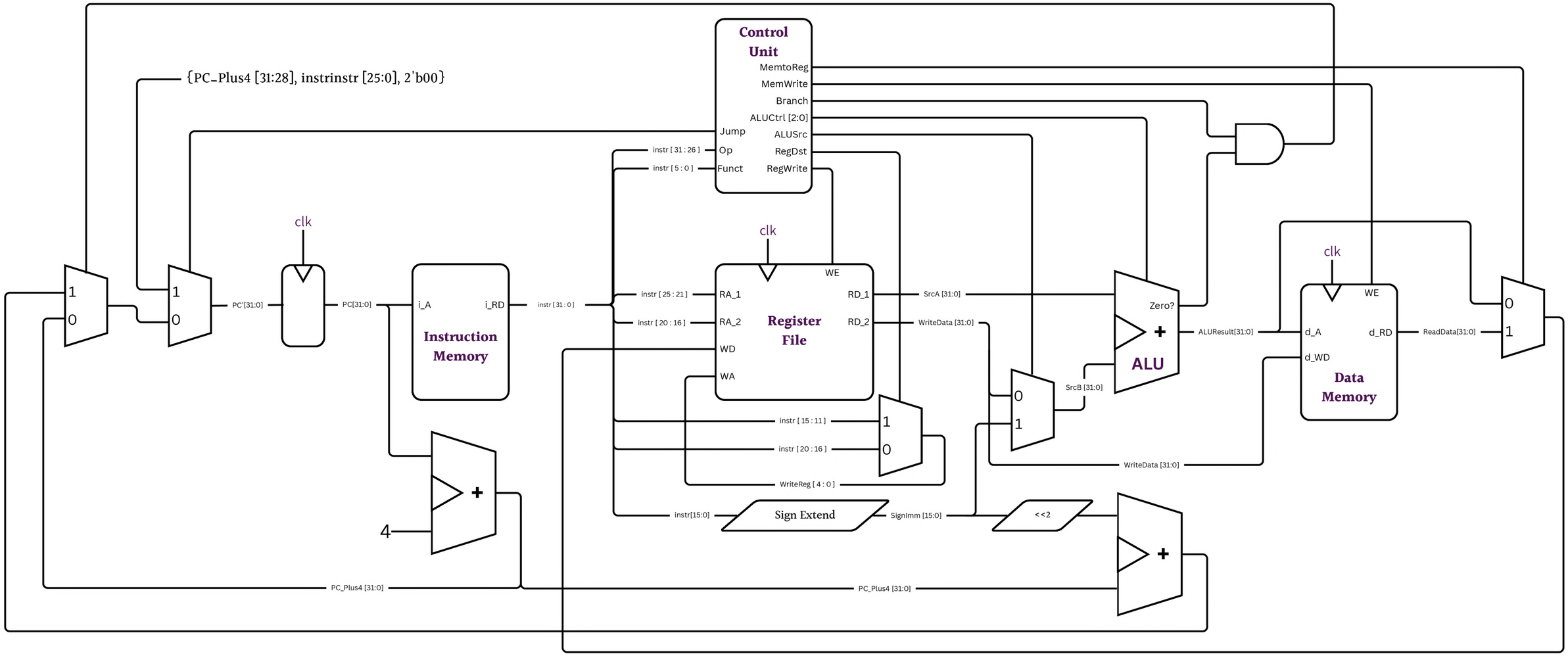
This project involves the design and implementation of a Single-Cycle MIPS Softcore Processor using Verilog HDL. The processor was developed with a RISC CPU architecture and verified through functional simulation using Icarus Verilog (iverilog). The simulation results were visualized with GTKWave, an open-source waveform viewer, to analyze signal transitions and verify processor behavior. The design demonstrates the fundamental principles of instruction execution, datapath, and control unit integration in FPGA-based processor design.

Icarus Verilog ⟶ Verilog Simulation

⟶ Electronic Waveform

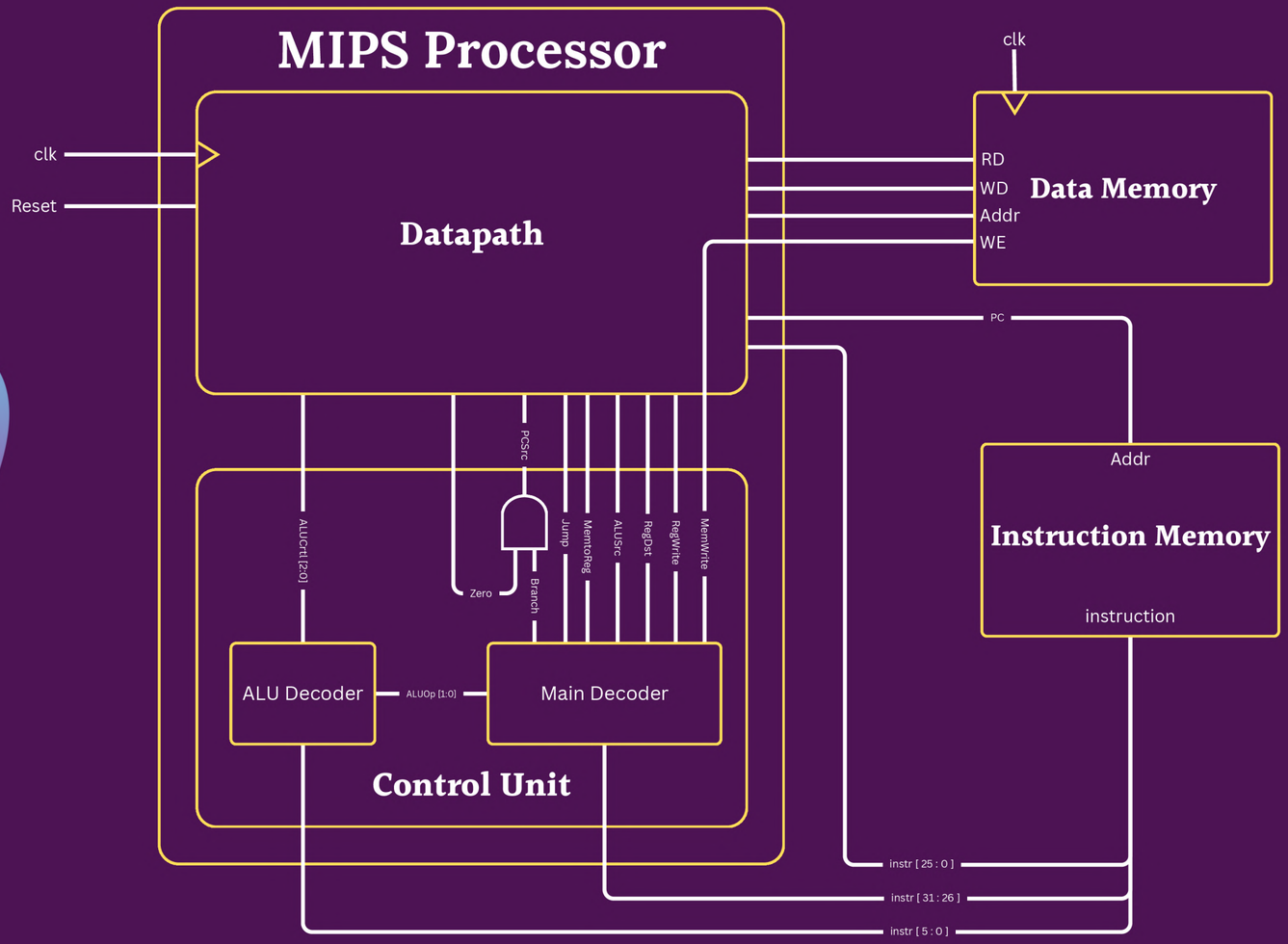
Link toGithub : XACKIES/RISC-CPU-Design-of-a-Single-Cycle-MIPS-Softcore-Processor-in-Verilog-HDL: Softcore MIPS Processor : Single-Cycle RISC Architecture in Verilog HDL

Single Cycle MIPS Processor schematic

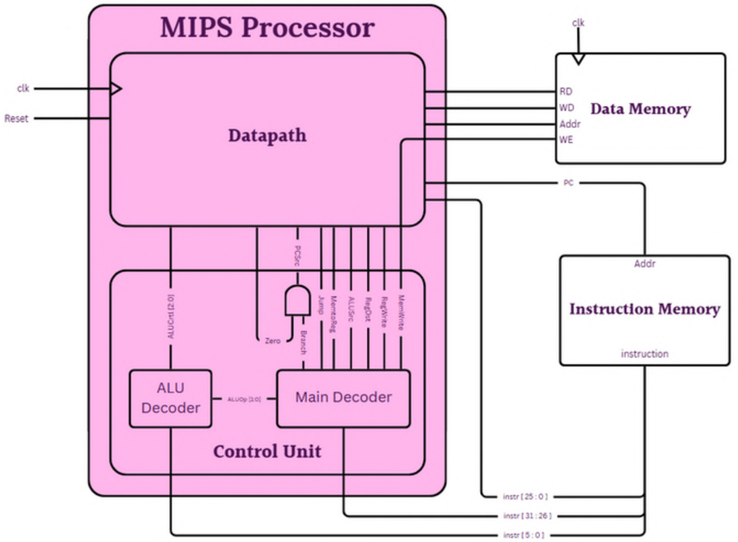


Ref : Digital Design and Computer Architecture , David M. Harris & Sarah L. Harris

Harvard architecture

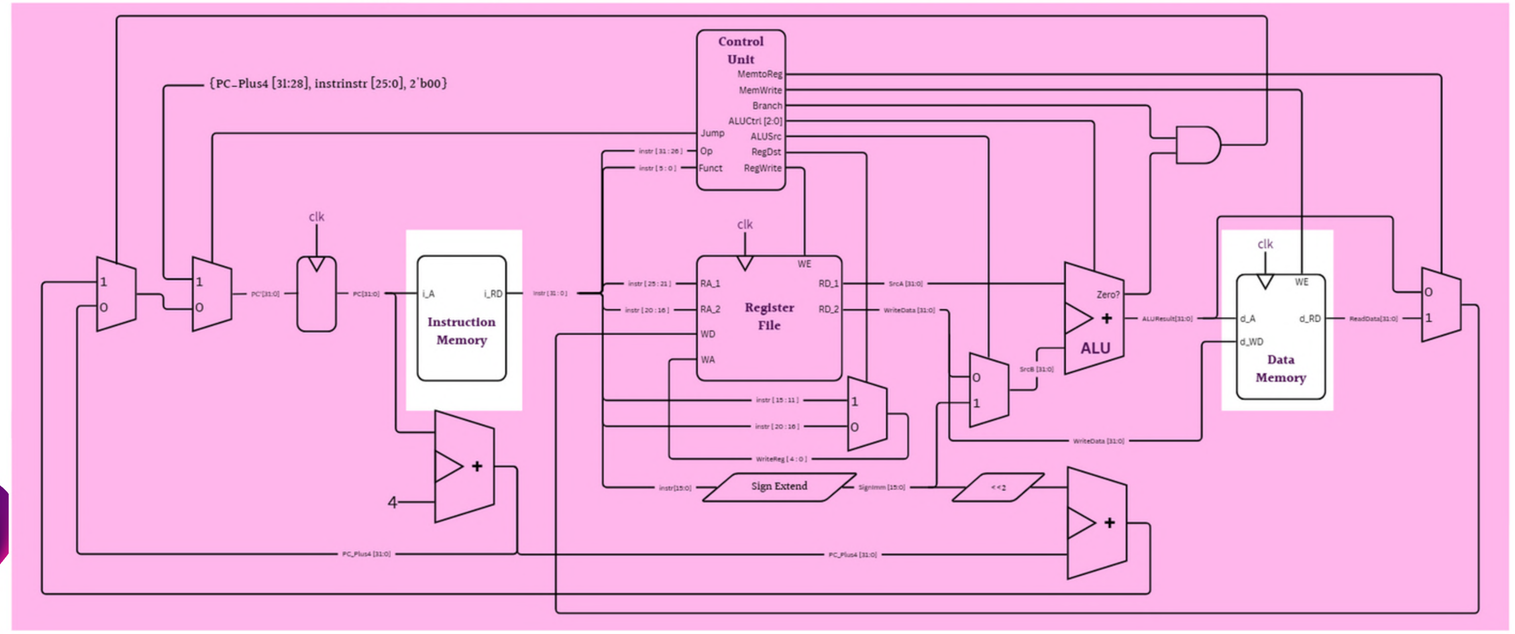


Highlight of relation for RTL Design by Verilog HDL

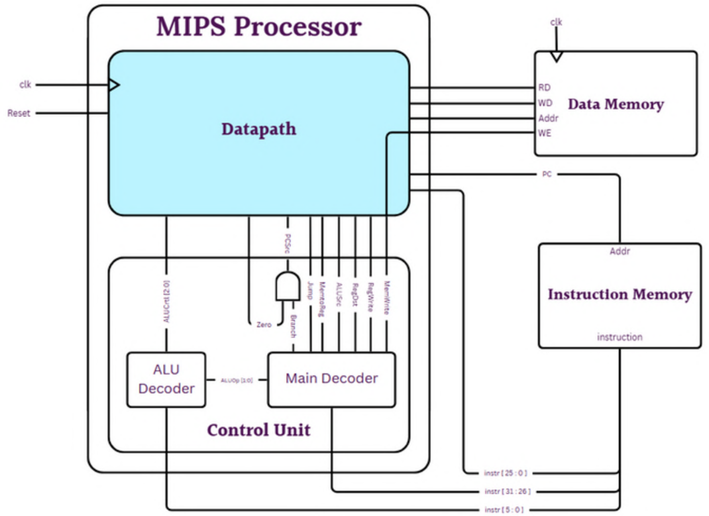


Harvard architecture

Single Cycle MIPS Processor schematic

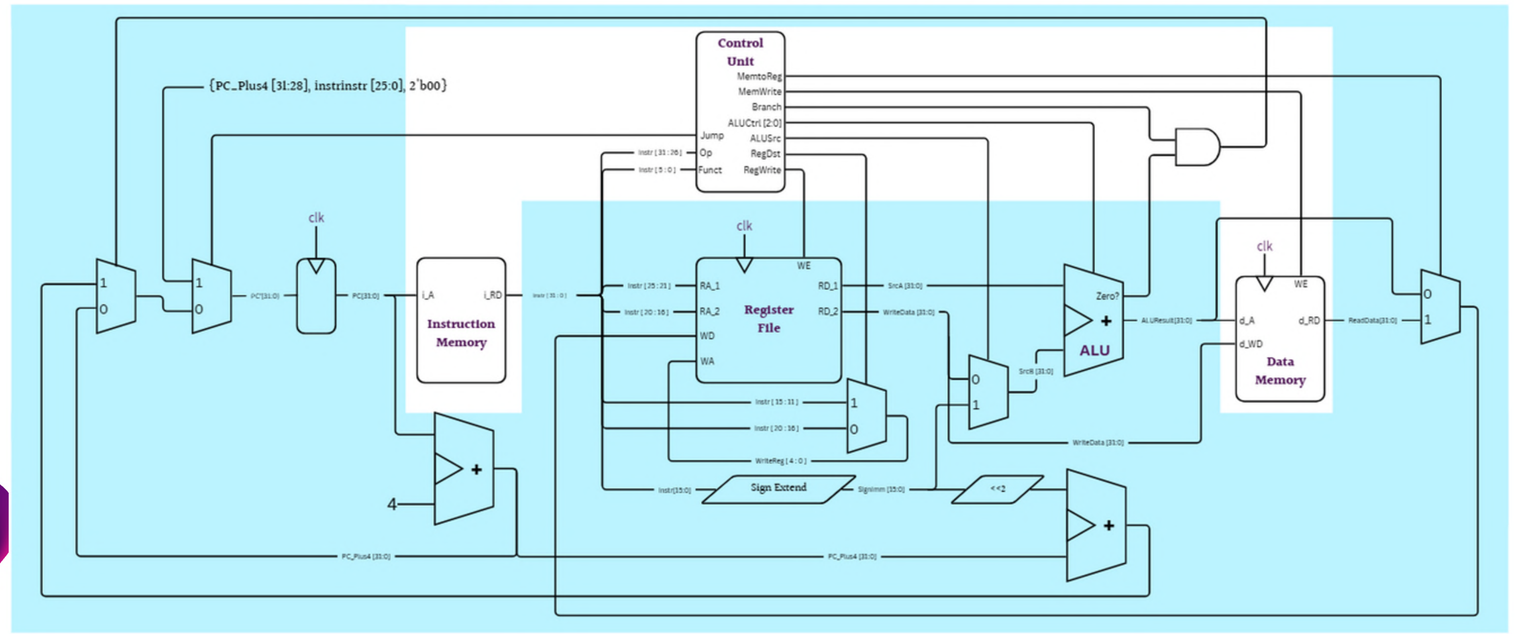


Highlight of relation for RTL Design by Verilog HDL

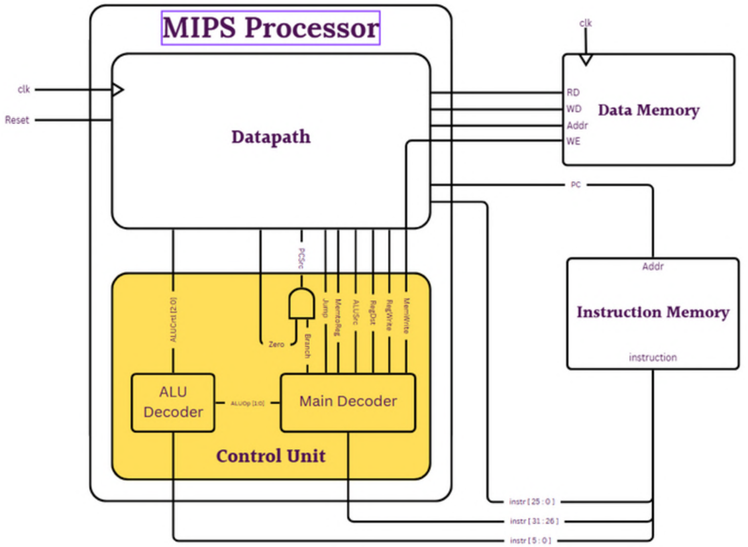


Harvard architecture

Single Cycle MIPS Processor schematic

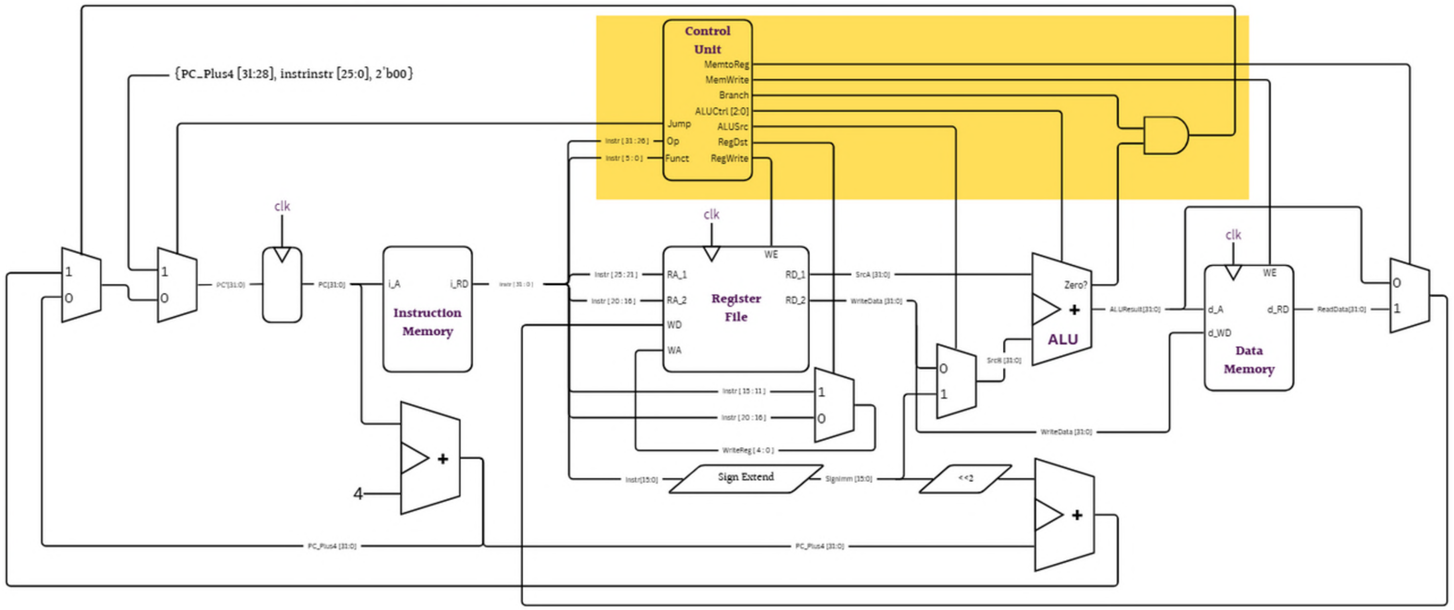


Highlight of relation for RTL Design by Verilog HDL

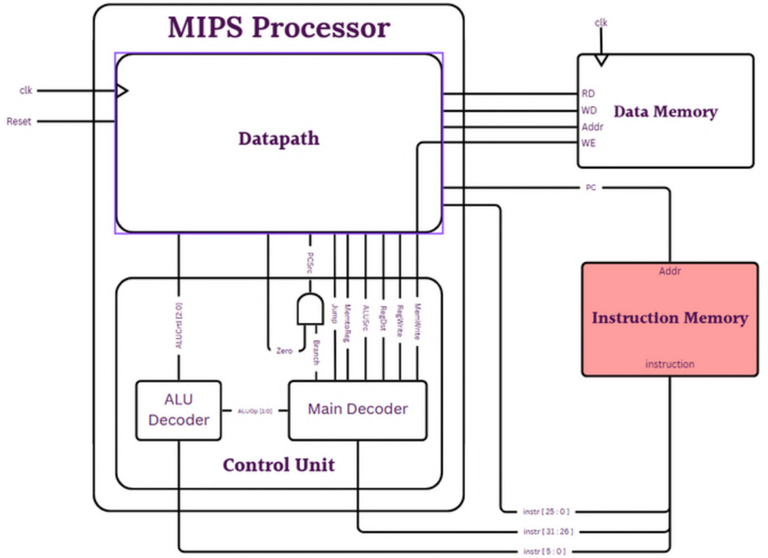


Harvard architecture

Single Cycle MIPS Processor schematic

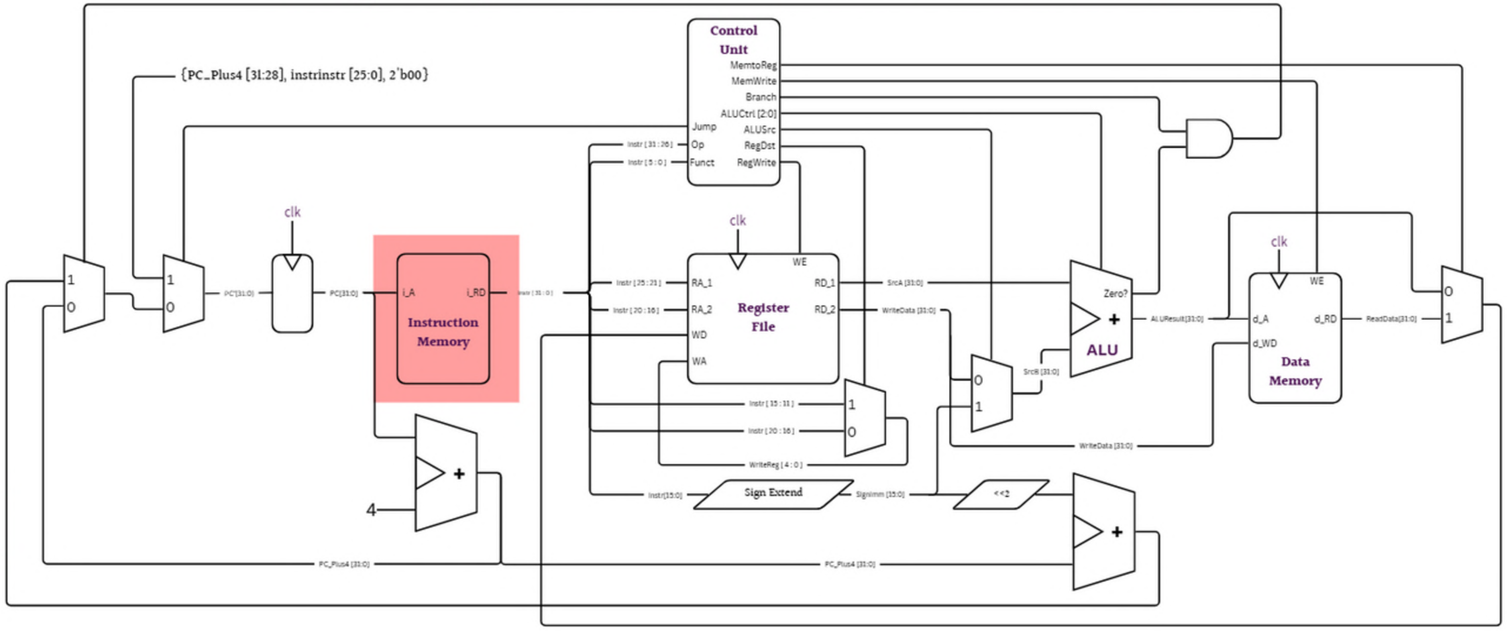


Highlight of relation for RTL Design by Verilog HDL

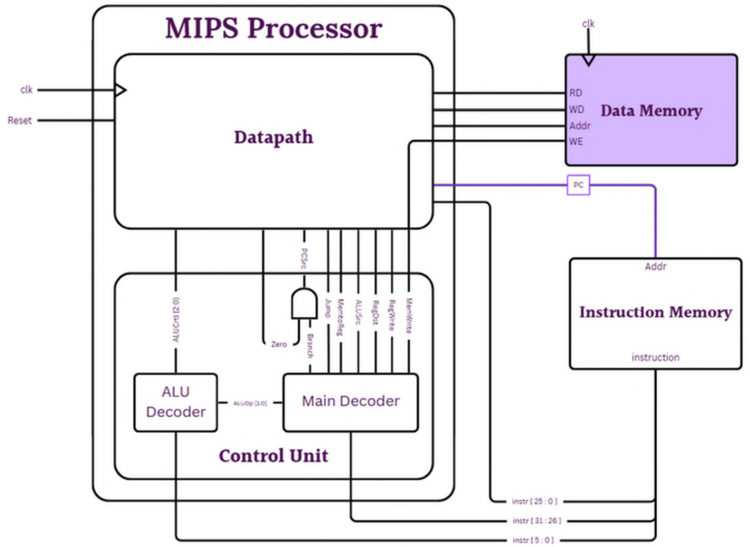


Harvard architecture

Single Cycle MIPS Processor schematic

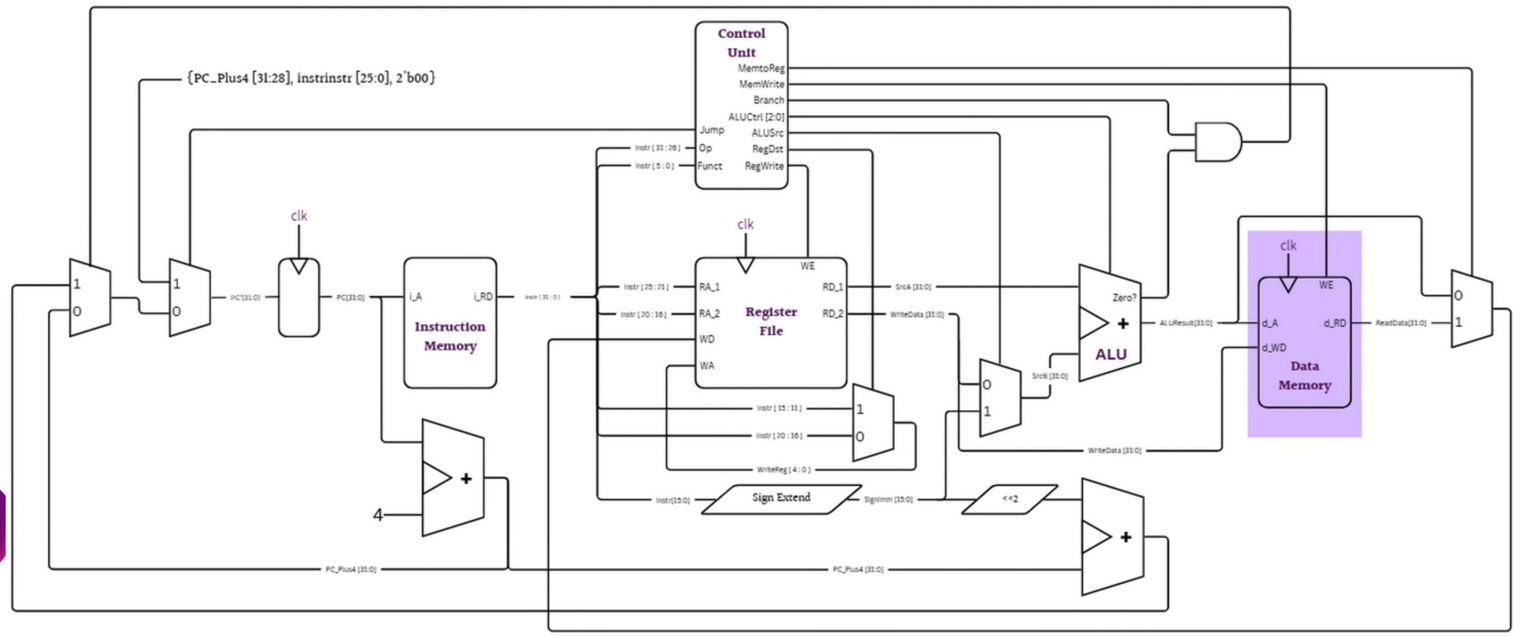


Highlight of relation for RTL Design by Verilog HDL

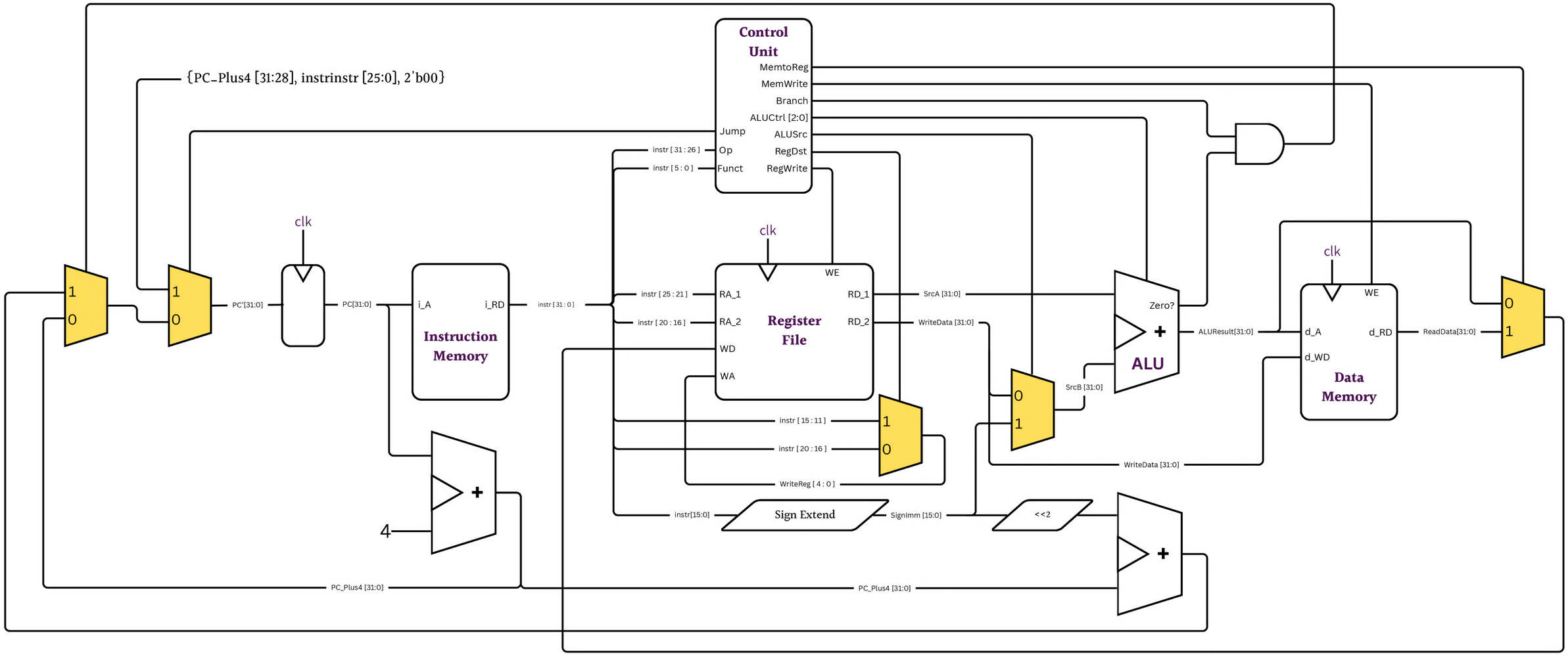


Harvard architecture

Single Cycle MIPS Processor schematic

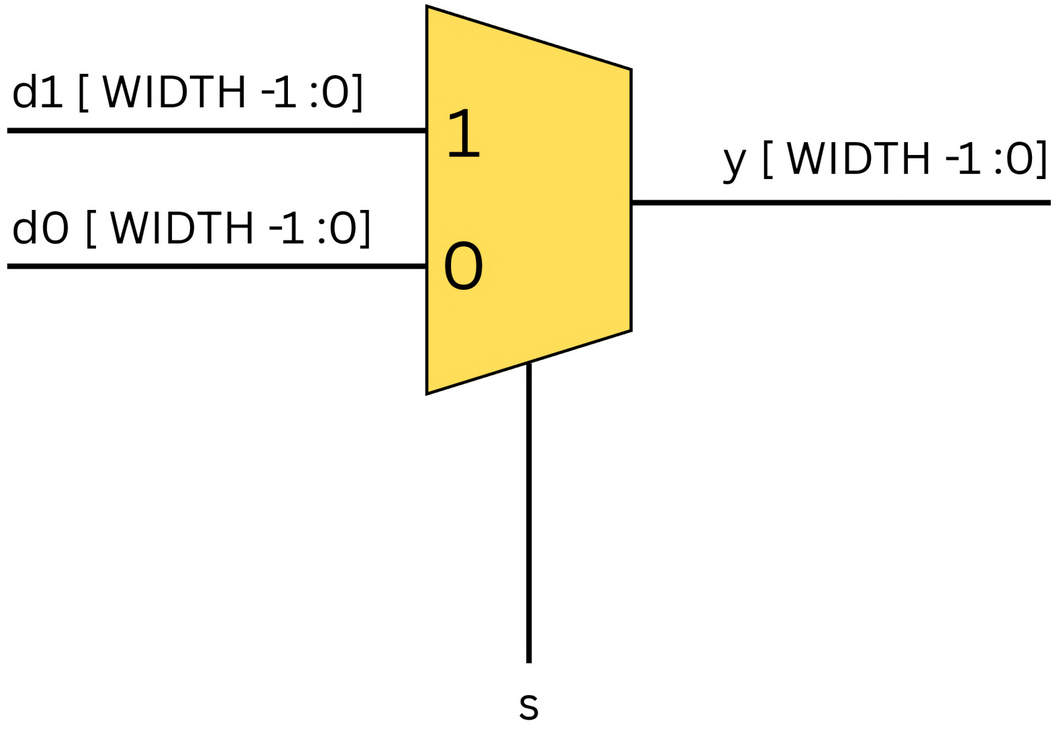


RTL Design by Verilog HDL



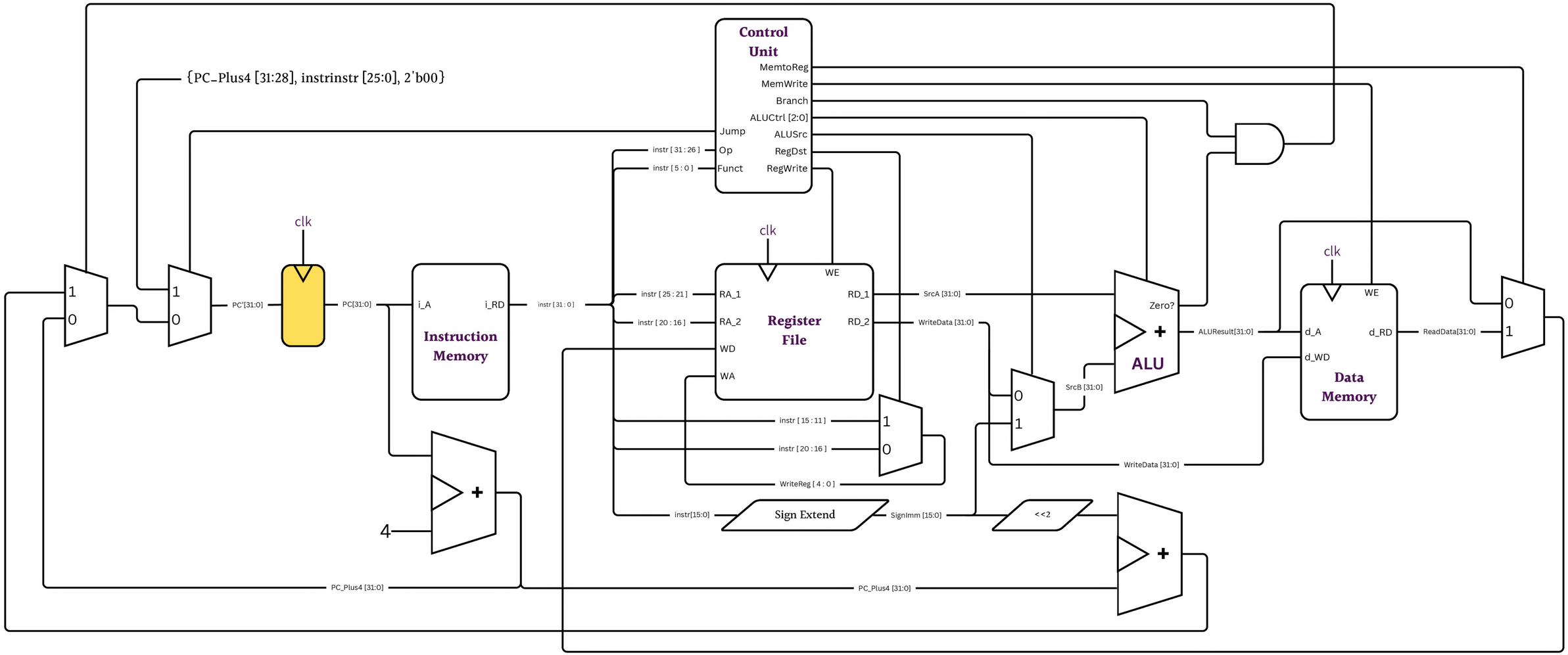
RTL Design by Verilog HDL

MUX2

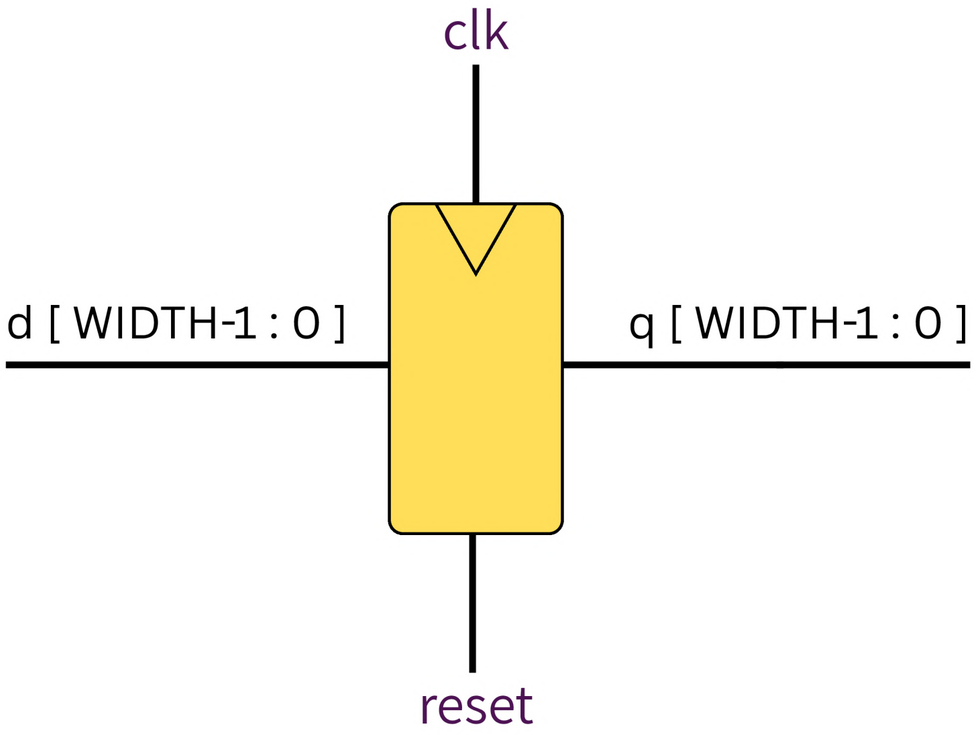



```
1   module MUX2 #(
2       parameter WIDTH = 8
3   ) (
4       input [WIDTH-1:0] d0,
5       d1,
6       input s,
7       output [WIDTH-1:0] y
8   );
9     assign y = s ? d1 : d0;
10  endmodule
```

RTL Design by Verilog HDL

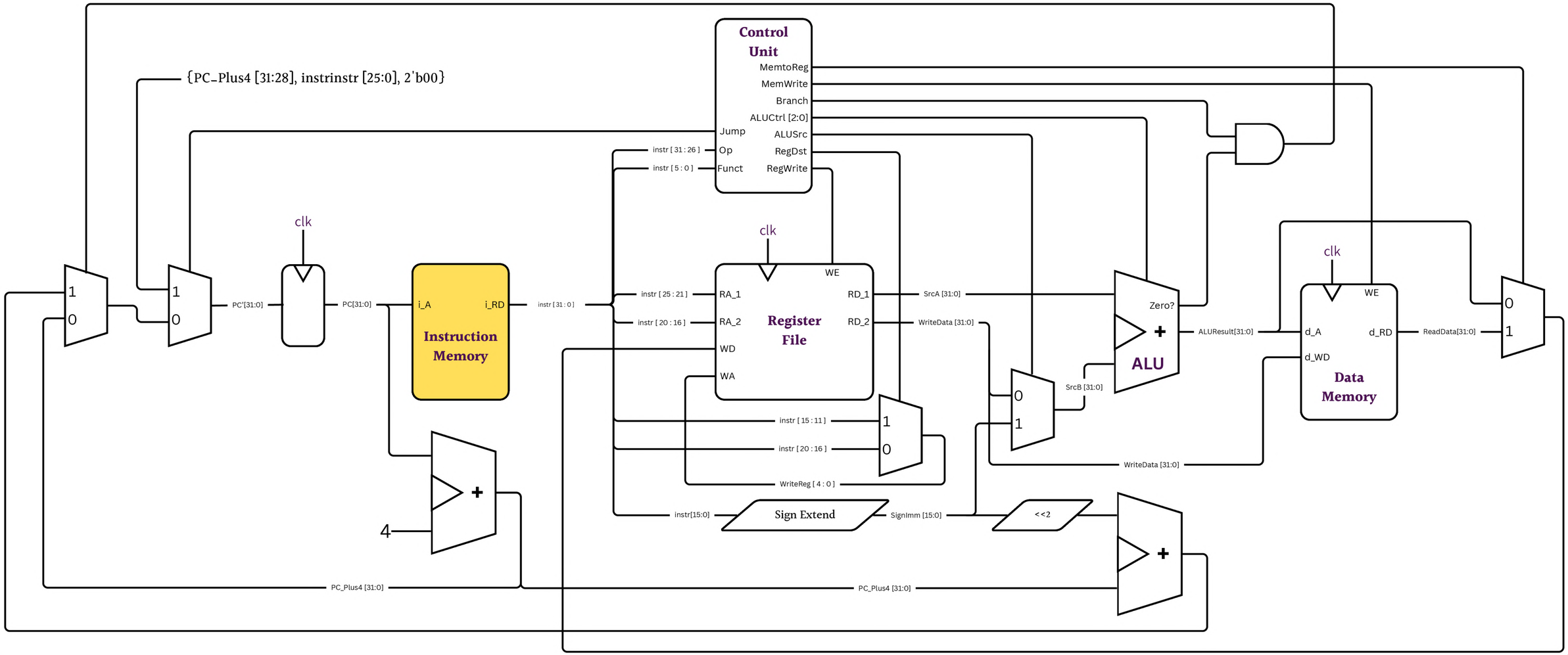


RTL Design by Verilog HDL

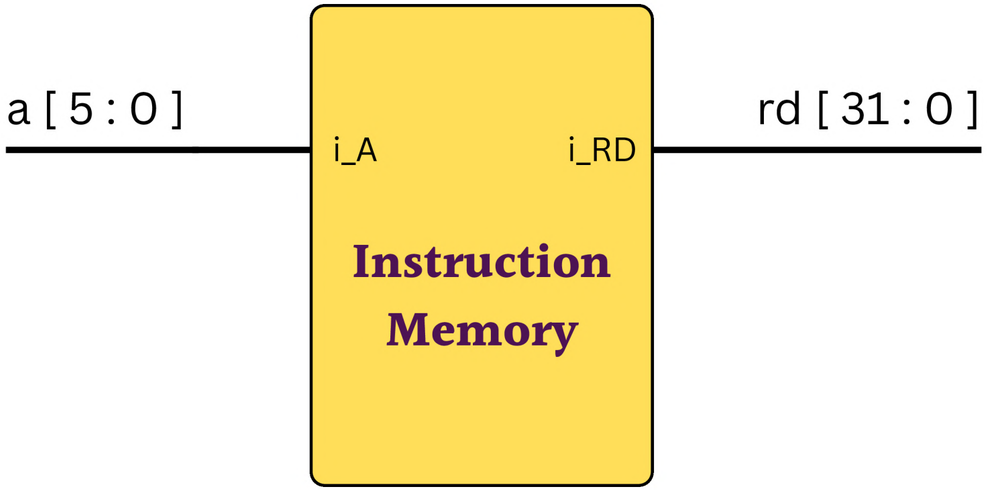



```
1   module DFF #(
2       parameter WIDTH = 8
3   ) (
4       input clk,
5       reset,
6       input [WIDTH-1:0] d,
7       output reg [WIDTH-1:0] q
8   );
9     always @(posedge clk, posedge reset)
10      if (reset) q <= 0;
11      else q <= d;
12  endmodule
```

RTL Design by Verilog HDL

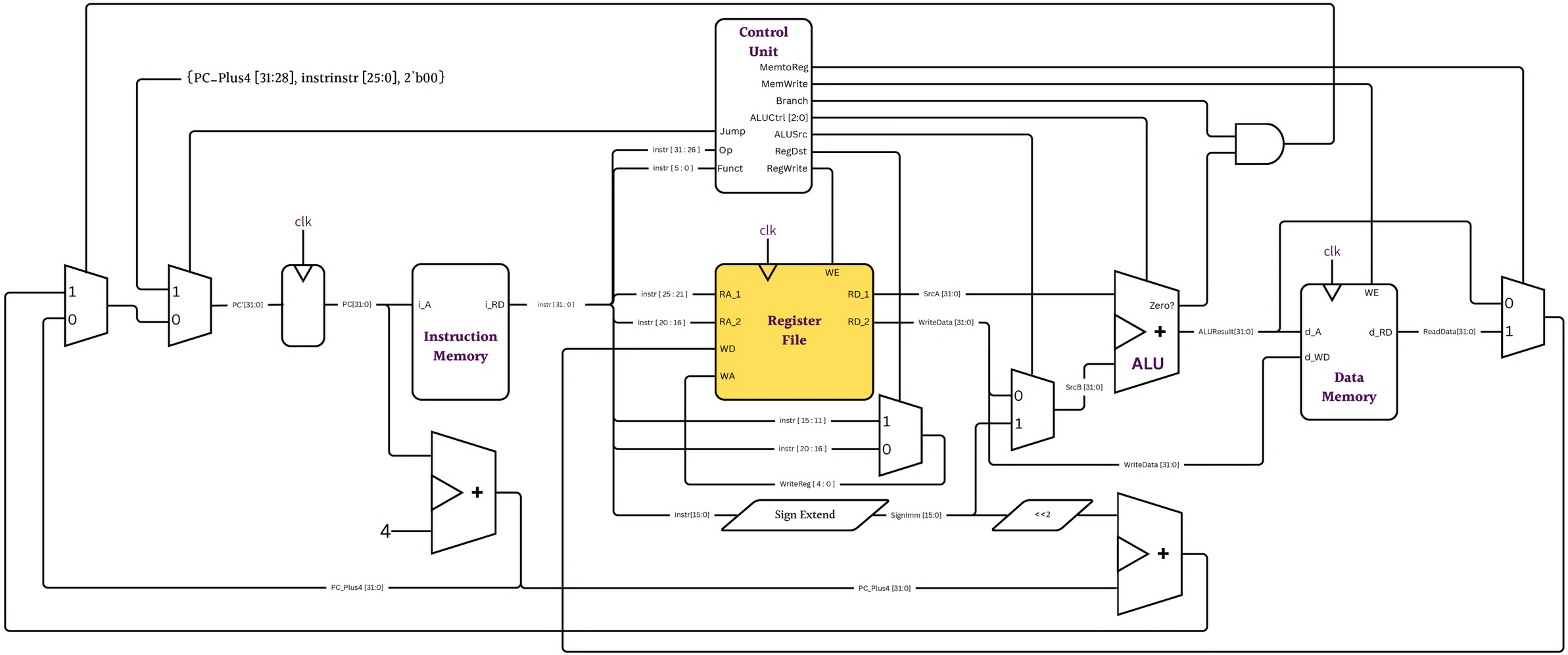


RTL Design by Verilog HDL

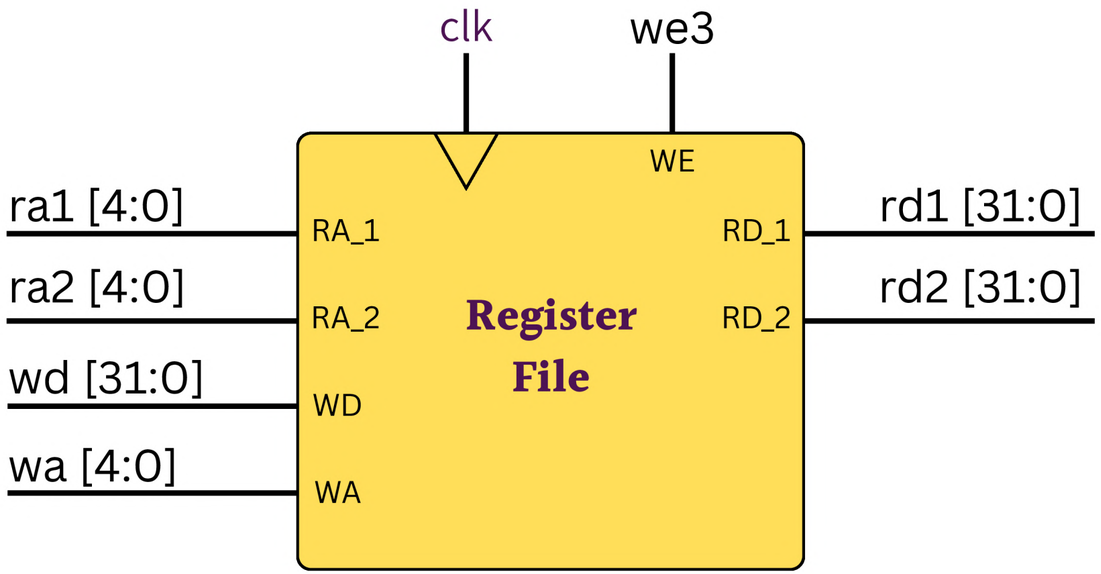



```verilog
module Instruction_Memory (
    input  [ 5:0] a,
    output [31:0] rd
);

  reg [31:0] RAM[63:0];

  initial begin
    $readmemh("Program_Memory.txt", RAM);
  end

  assign rd = RAM[a];
endmodule
```

RTL Design by Verilog HDL

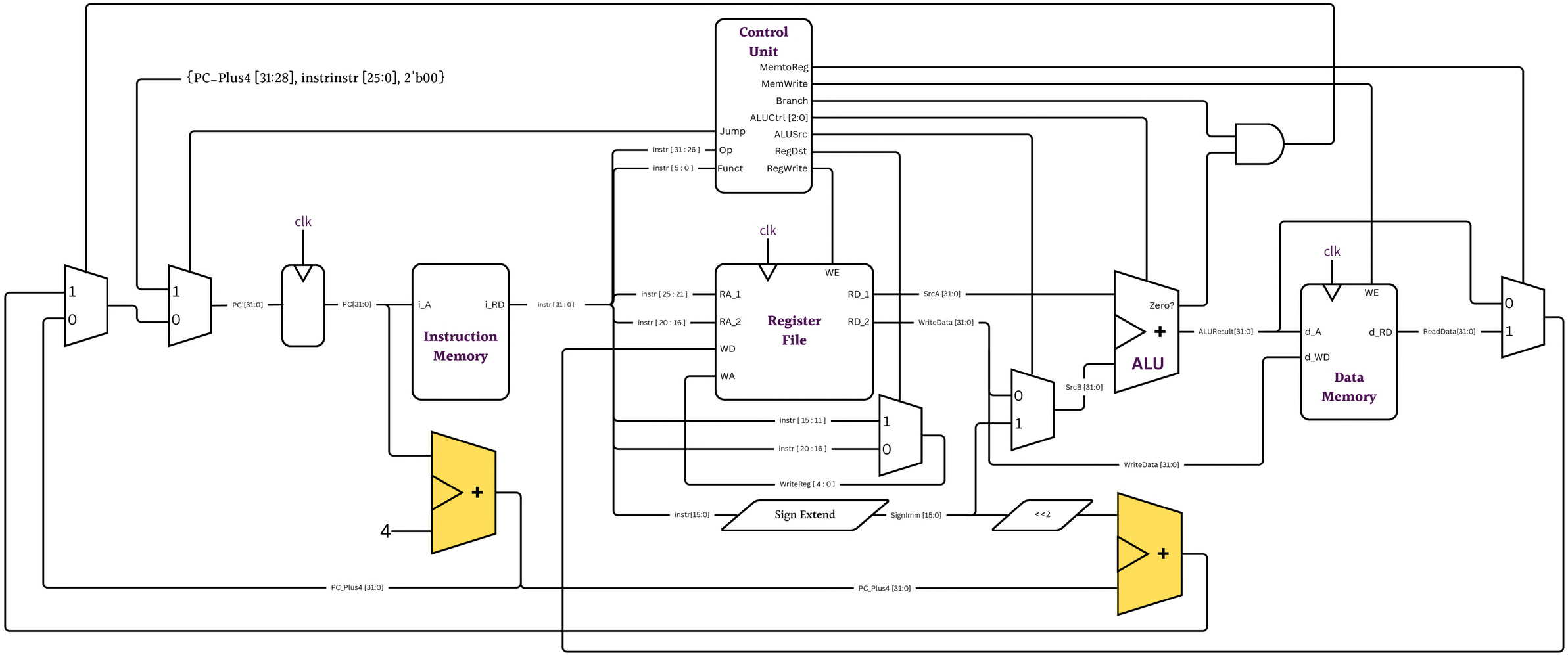


RTL Design by Verilog HDL

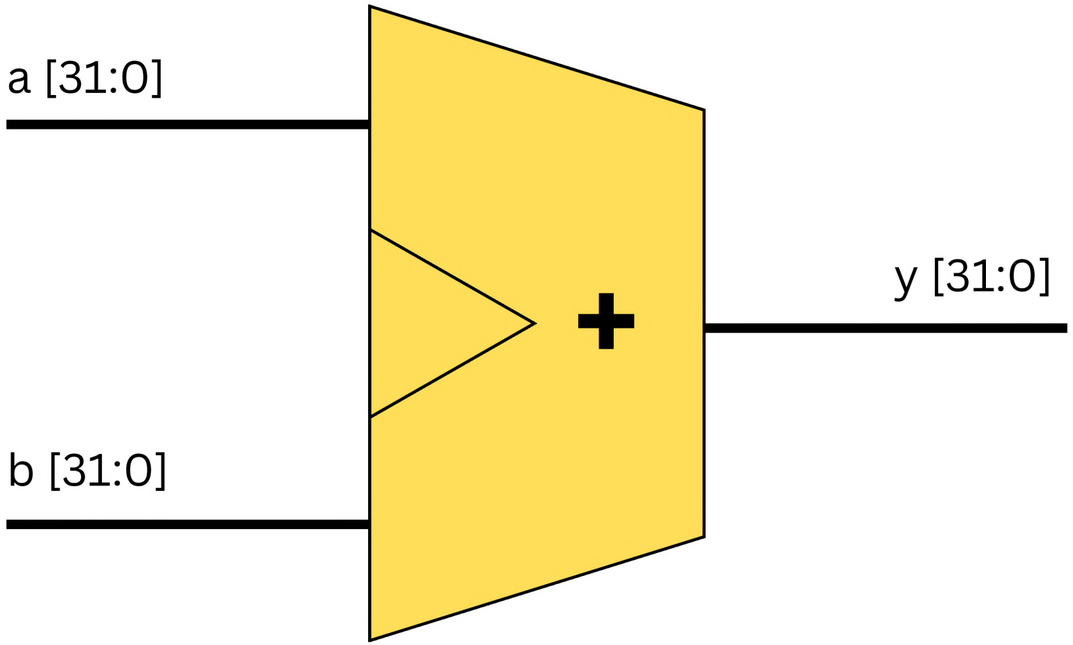



```verilog
module Register_file (
    input          clk,
    input          we3,
    input  [ 4:0] ra1,
    ra2,
    wa3,
    input  [31:0] wd3,
    output [31:0] rd1,
    rd2

);
  reg [31:0] rf[31 : 0];
  always @(posedge clk) if (we3) rf[wa3] <= wd3;
  assign rd1 = (ra1 != 0) ? rf[ra1] : 0;
  assign rd2 = (ra2 != 0) ? rf[ra2] : 0;
endmodule
```

RTL Design by Verilog HDL

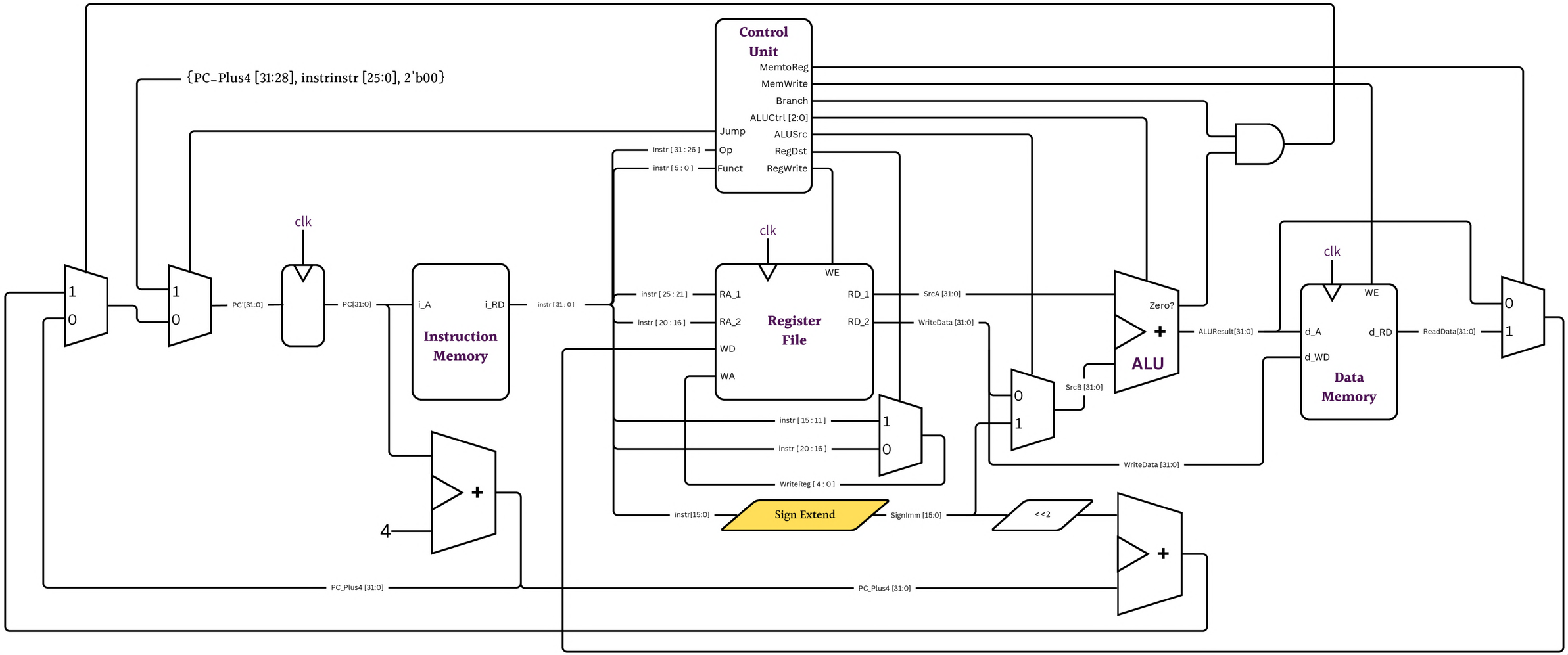


RTL Design by Verilog HDL




```
1   module ADDER (
2       input  [31:0] a,
3       b,
4       output [31:0] y
5   );
6     assign y = a + b;
7   endmodule
```

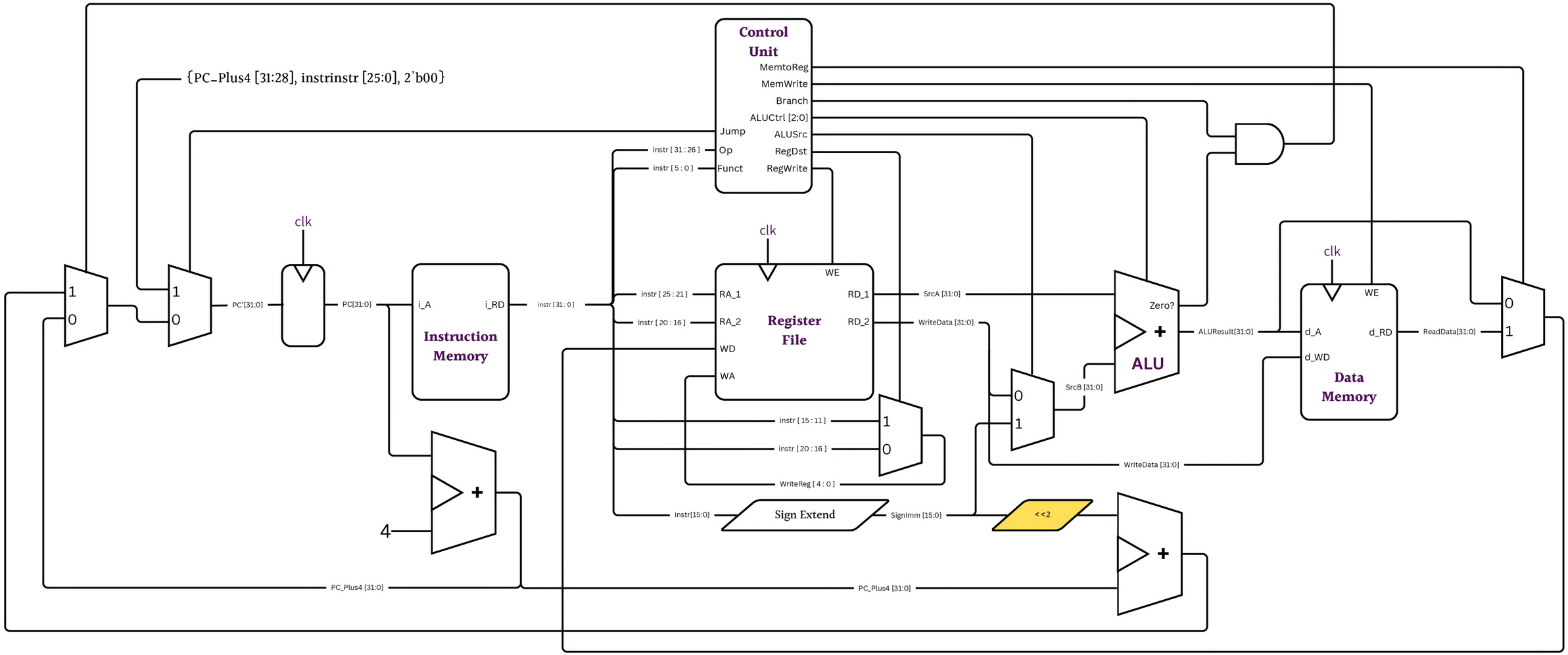
RTL Design by Verilog HDL



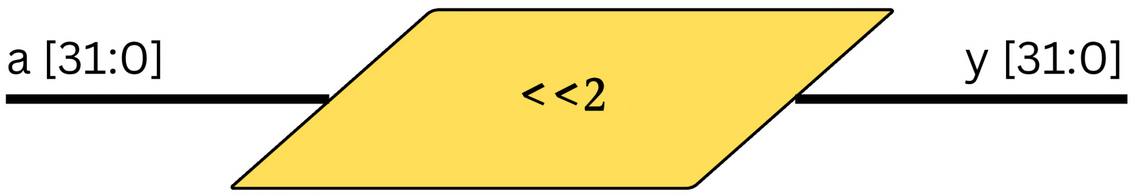
RTL Design by Verilog HDL


```verilog
module Sign_Ext (
    input  [15:0] a,
    output [31:0] y
);
  assign y = {{16{a[15]}}, a};
endmodule
```

RTL Design by Verilog HDL

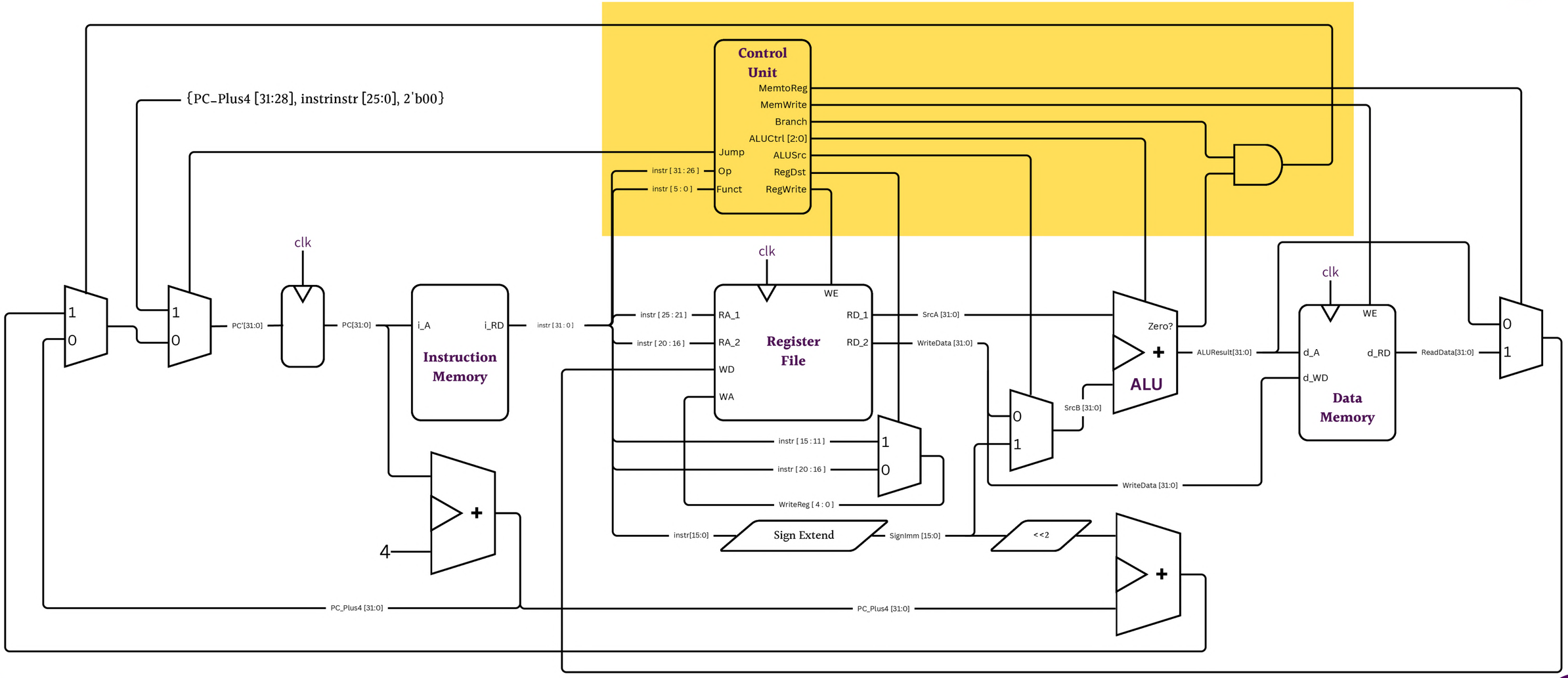


RTL Design by Verilog HDL

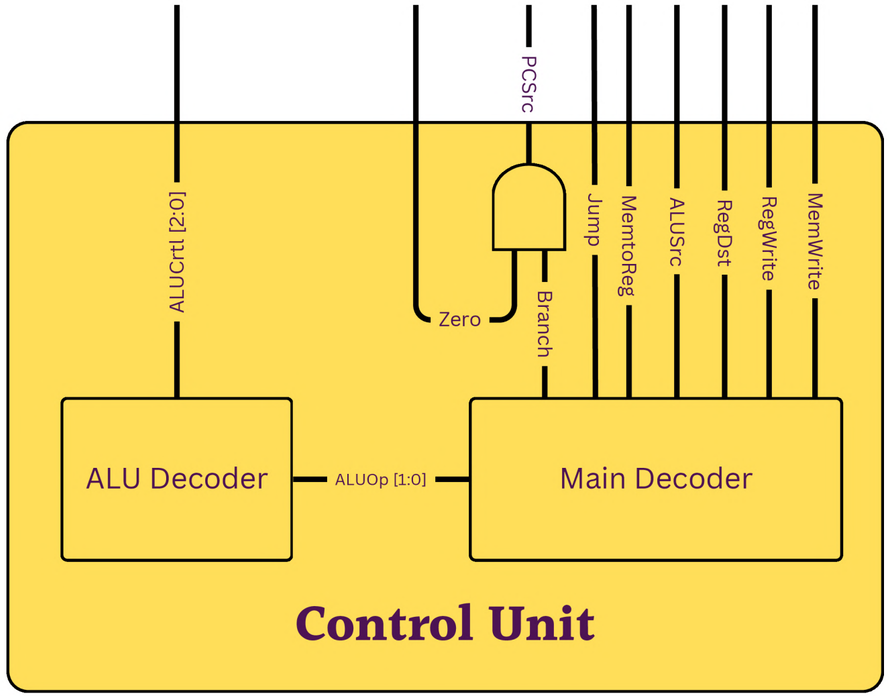



```
1   module Shift_Left (
2       input  [31:0] a,
3       output [31:0] y
4   );
5     assign y = {a[29:00], 2'b00};
6   endmodule
```

RTL Design by Verilog HDL

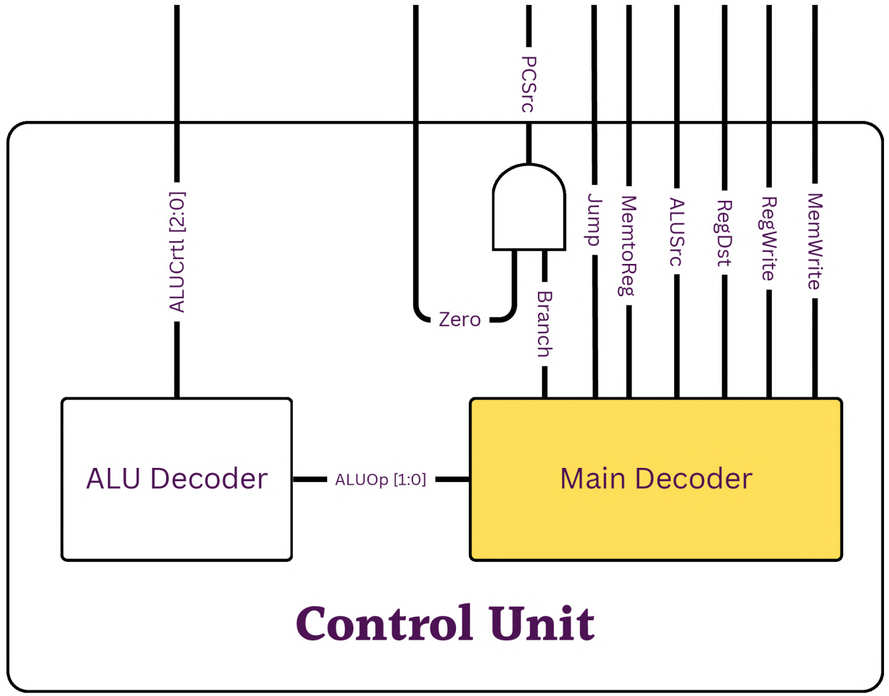


RTL Design by Verilog HDL



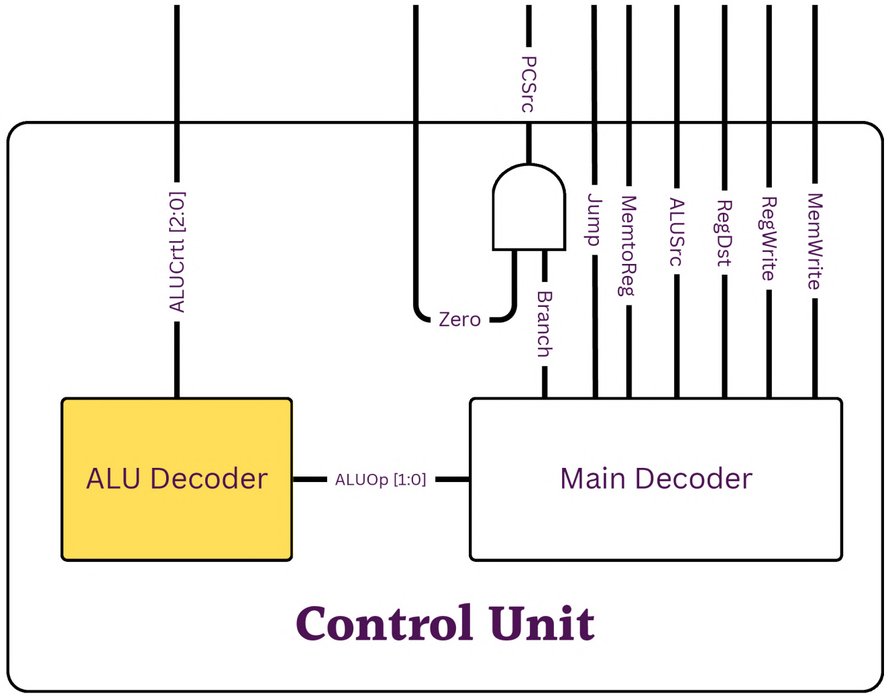

```verilog
module Control_Unit (
    input [5:0] op,
    funct,
    input zero,
    output memtoreg,
    memwrite,
    output pcsrc,
    alusrc,
    output regdst,
    regwrite,
    output jump,
    output [2:0] alucontrol
);
  wire [1:0] aluop;
  wire branch;
  Main_Decoder md (
      op,
      memtoreg,
      memwrite,
      branch,
      alusrc,
      regdst,
      regwrite,
      jump,
      aluop
  );
  ALU_Decoder ad (
      funct,
      aluop,
      alucontrol
  );
  assign pcsrc = branch & zero;
endmodule
```

RTL Design by Verilog HDL



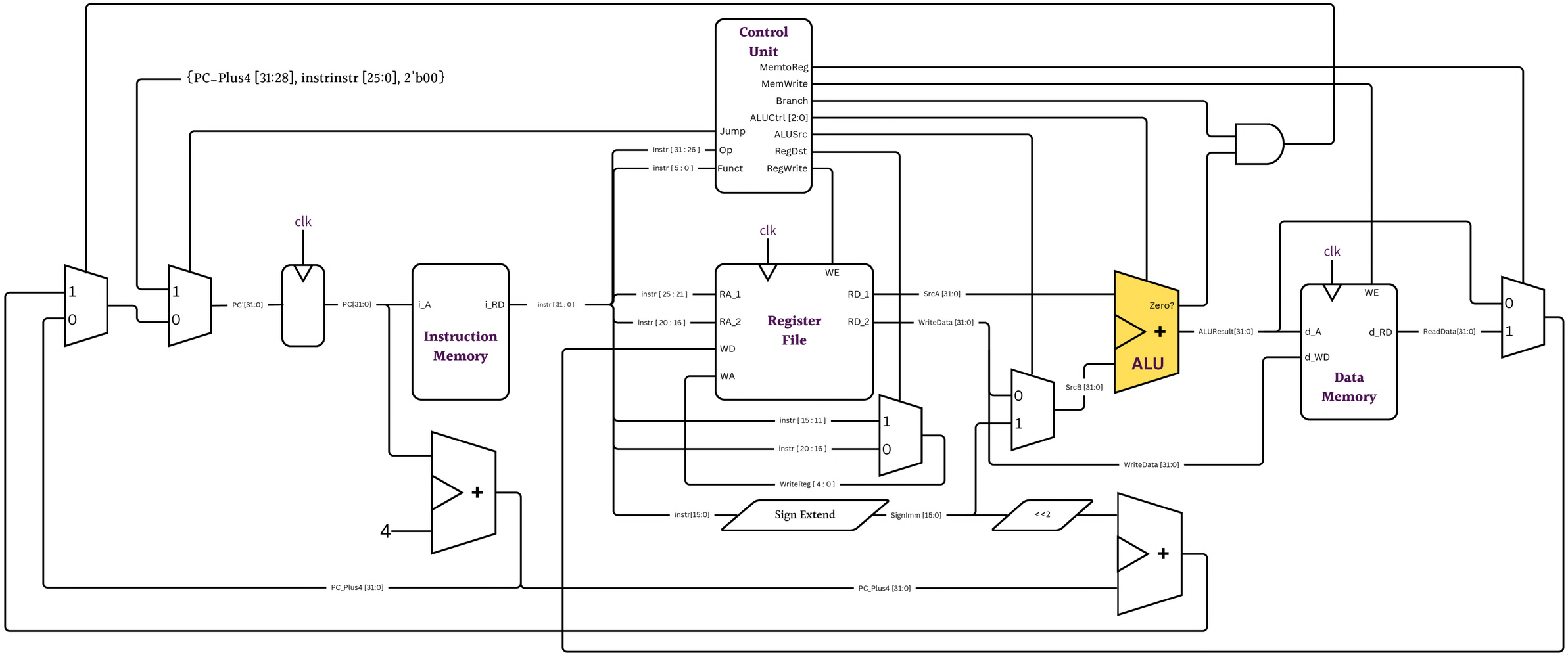

```verilog
module Main_Decoder (
    input [5:0] op,
    output memtoreg,
    memwrite,
    output branch,
    alusrc,
    output regdst,
    regwrite,
    output jump,
    output [1:0] aluop
);
  reg [8:0] controls;
  assign {regwrite, regdst, alusrc, branch, memwrite, memtoreg, jump, aluop} = controls;
  always @(*)
    case (op)
      6'b000000: controls <= 9'b110000010;  //Rtyp
      6'b100011: controls <= 9'b101001000;  //LW
      6'b101011: controls <= 9'b001010000;  //SW
      6'b000100: controls <= 9'b000100001;  //BEQ
      6'b001000: controls <= 9'b101000000;  //ADDI
      6'b000010: controls <= 9'b000000100;  //J
      default:   controls <= 9'bxxxxxxxxx;  //???
    endcase
endmodule
```

RTL Design by Verilog HDL

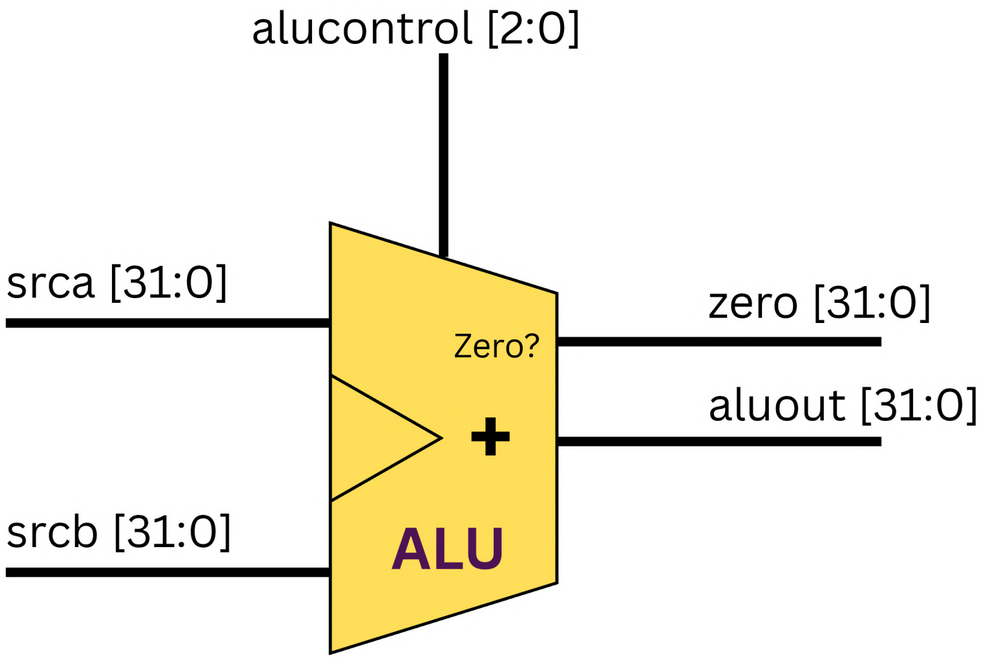



```verilog
module ALU_Decoder (
    input [5:0] funct,
    input [1:0] aluop,
    output reg [2:0] alucontrol
);
  always @(*)
    case (aluop)
      2'b00: alucontrol <= 3'b010;
      2'b01: alucontrol <= 3'b110;
      default:
      case (funct)
        6'b100000: alucontrol <= 3'b010;  // ADD
        6'b100010: alucontrol <= 3'b110;  // SUB
        6'b100100: alucontrol <= 3'b000;  // AND
        6'b100101: alucontrol <= 3'b001;  // OR
        6'b101010: alucontrol <= 3'b111;  // SLT
        default:   alucontrol <= 3'bxxx;  // ???
      endcase
    endcase
endmodule
```

RTL Design by Verilog HDL

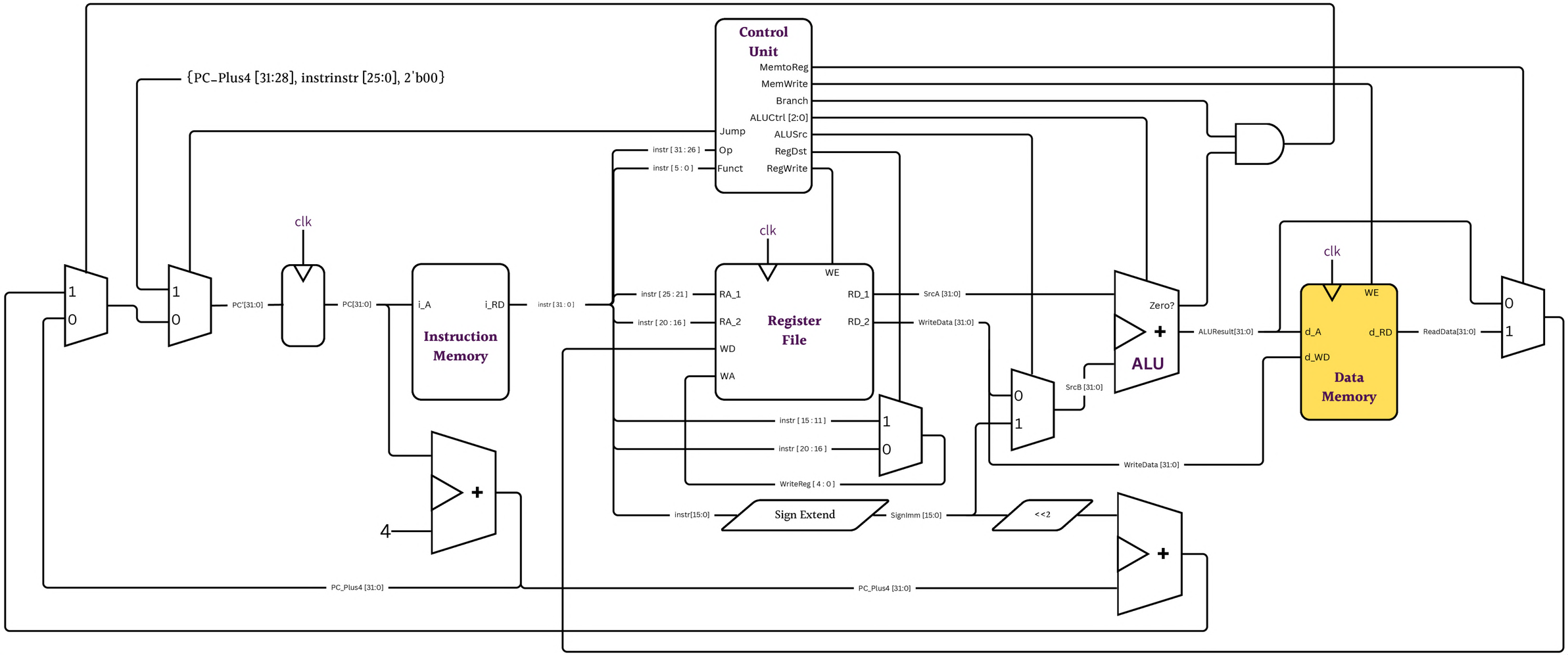


RTL Design by Verilog HDL

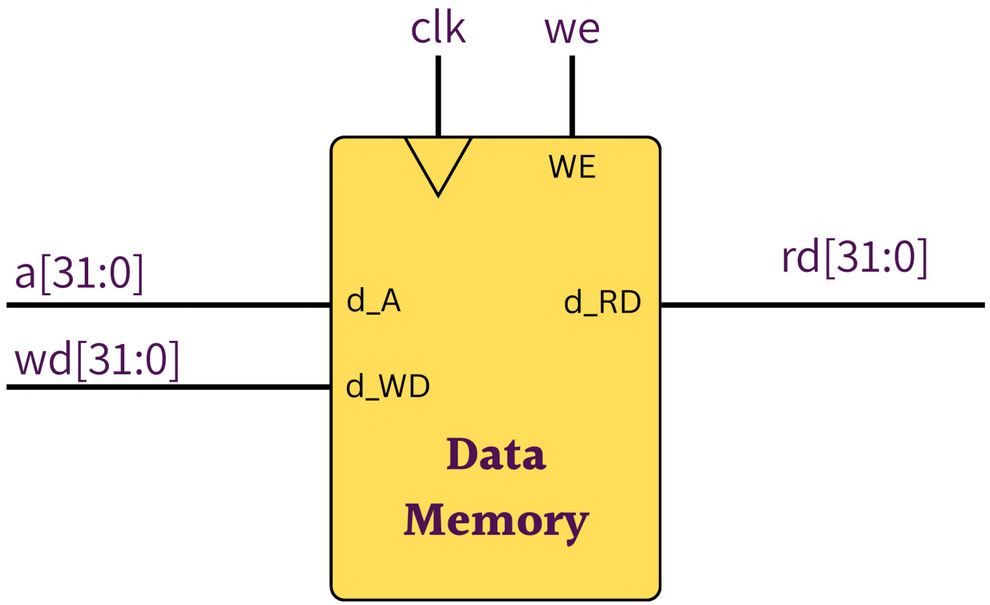



```verilog
module ALU (
    input      [31:0] srca,
    input      [31:0] srcb,
    input      [ 2:0] alucontrol,
    output reg [31:0] aluout,
    output            zero
);

  wire [31:0] condinvb, sum;

  assign condinvb = alucontrol[2] ? ~srcb : srcb;
  assign sum      = srca + condinvb + {31'd0, alucontrol[2]};

  always @(*) begin
    case (alucontrol[1:0])
      2'b00:   aluout = srca & srcb;  // AND
      2'b01:   aluout = srca | srcb;  // OR
      2'b10:   aluout = sum;  // ADD/SUB , 2'Complement Method
      2'b11:   aluout = {31'b0, sum[31]};  // Set Less Than
      default: aluout = 32'hxxxxxxxx;
    endcase
  end

  assign zero = (aluout == 32'b0);

endmodule
```

RTL Design by Verilog HDL

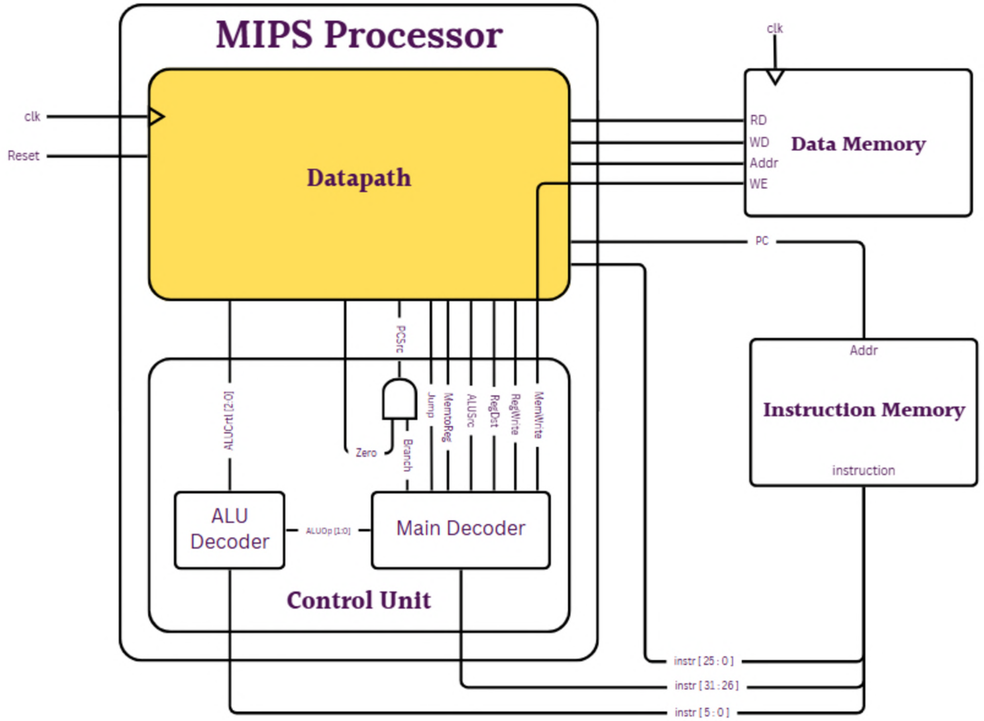


RTL Design by Verilog HDL



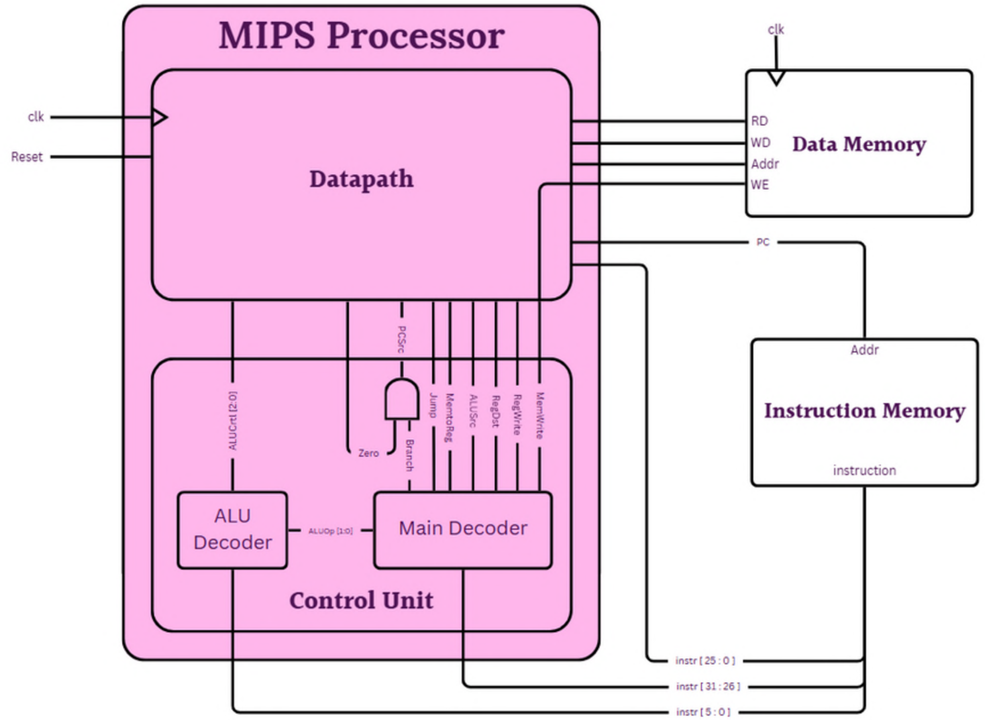

```verilog
module Data_Memory (
    input         clk,
    we,
    input  [31:0] a,
    wd,
    output [31:0] rd
);

  reg [31:0] RAM[63:0];

  assign rd = RAM[a[31:2]];

  always @(posedge clk) if (we) RAM[a[31:2]] <= wd;

endmodule
```

RTL Design by Verilog HDL



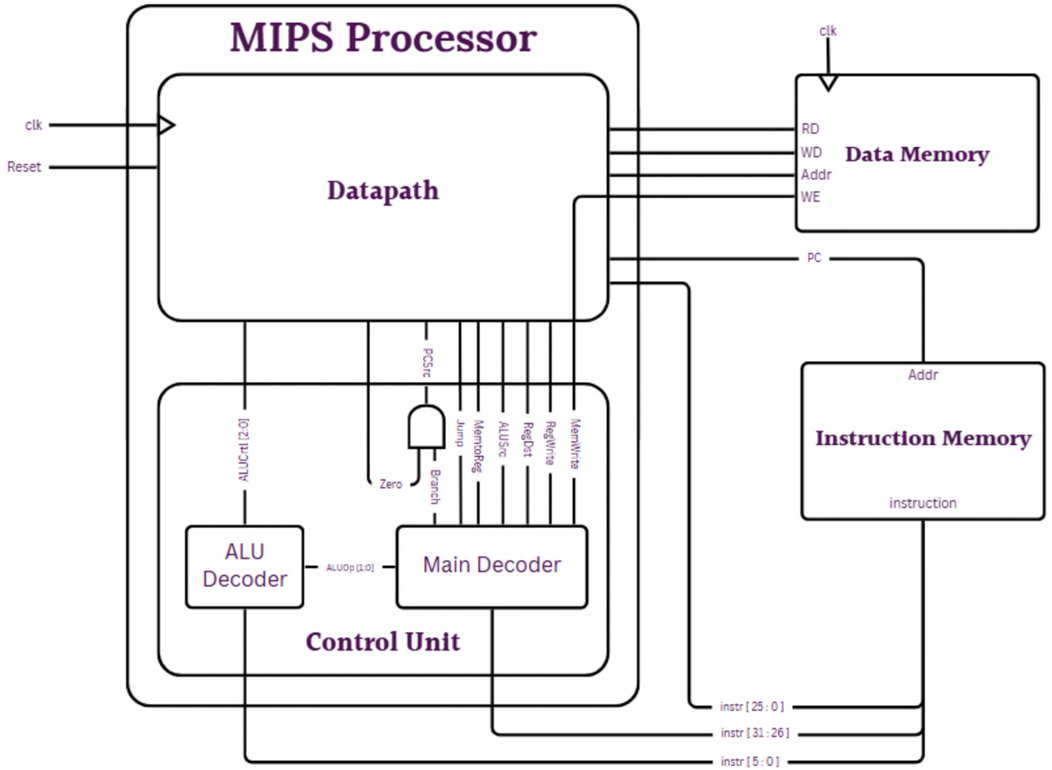

```v
module DataPath (
    input         clk,
    reset,
    input         memtoreg,
    pcsrc,
    input         alusrc,
    regdst,
    input         regwrite,
    jump,
    input  [ 2:0] alucontrol,
    output        zero,
    output [31:0] pc,
    input  [31:0] instr,
    output [31:0] aluout,
    writedata,
    input  [31:0] readdata
);

  wire [4:0] writereg;
  wire [31:0] pcnext, pcnextbr, pcplus4, pcbranch;
  wire [31:0] signimm, signimmsh;
  wire [31:0] srca, srcb;
  wire [31:0] result;

  // next PC logic
  DFF #(32) pcreg (
      clk,
      reset,
      pcnext,
      pc
  );
  ADDER pcadd1 (
      pc,
      32'b100,
      pcplus4
  );
  Shift_Left immsh (
      signimm,
      signimmsh
  );
  ADDER pcadd2 (
      pcplus4,
      signimmsh,
      pcbranch
  );
  MUX2 #(32) pcbrmux (
      pcplus4,
      pcbranch,
      pcsrc,
      pcnextbr
  );
  MUX2 #(32) pcmux (
      pcnextbr,
      {pcplus4[31:28], instr[25:0], 2'b00},
      jump,
      pcnext
  );


  Register_file rf (
      clk,
      regwrite,
      instr[25:21],
      instr[20:16],
      writereg,
      result,
      srca,
      writedata
  );
  MUX2 #(5) wrmux (
      instr[20:16],
      instr[15:11],
      regdst,
      writereg
  );
  MUX2 #(32) resmux (
      aluout,
      readdata,
      memtoreg,
      result
  );
  Sign_Ext se (
      instr[15:0],
      signimm
  );

  // ALU logic
  MUX2 #(32) srcbmux (
      writedata,
      signimm,
      alusrc,
      srcb
  );
  ALU alu (
      .srca(srca),
      .srcb(srcb),
      .alucontrol(alucontrol),
      .aluout(aluout),
      .zero(zero)
  );
endmodule
```

RTL Design by Verilog HDL




```
module MIPS (
    input clk,
    reset,
    output [31:0] pc,
    input [31:0] instr,
    output memwrite,
    output [31:0] aluout,
    writedata,
    input [31:0] readdata
);
  wire memtoreg, branch, alusrc, regdst, regwrite, jump;
  wire [2:0] alucontrol;
  Control_Unit c (
      instr[31:26],
      instr[5:0],
      zero,
      memtoreg,
      memwrite,
      pcsrc,
      alusrc,
      regdst,
      regwrite,
      jump,
      alucontrol
  );
  DataPath dp (
      clk,
      reset,
      memtoreg,
      pcsrc,
      alusrc,
      regdst,
      regwrite,
      jump,
      alucontrol,
      zero,
      pc,
      instr,
      aluout,
      writedata,
      readdata
  );
endmodule
```

RTL Design by Verilog HDL




```verilog
module top (
    input          clk,
    reset,
    output [31:0] writedata,
    dataadr,
    output         memwrite
);

  wire [31:0] pc, instr, readdata;


  MIPS mips (
      clk,
      reset,
      pc,
      instr,
      memwrite,
      dataadr,
      writedata,
      readdata
  );

  Instruction_Memory imem (
      pc[7:2],
      instr
  );
  Data_Memory dmem (
      clk,
      memwrite,
      dataadr,
      writedata,
      readdata
  );

endmodule
```

Testing by Testbench and Icarus Verilog

```
module testbench;

  reg clk;
  reg reset;

  wire [31:0] writedata, dataadr;
  wire memwrite;

  // ========== Device   Under Test ==========
  top dut (
      .clk(clk),
      .reset(reset),
      .writedata(writedata),
      .dataadr(dataadr),
      .memwrite(memwrite)
  );

  // ========== Waveform dump (GTKWave) ==========
  initial begin
    $dumpfile("sim.vcd");
    $dumpvars(0, testbench);
    $dumpvars(0, dut);  // dump DUT internals for easier debug
  end

  // ========== Reset/Clock ==========
  initial begin
    clk   = 1'b0;
    reset = 1'b1;  // assert reset
    #22;
    reset = 1'b0;  // deassert reset
  end

  // Generate a 10 ns clock period (100 MHz)
  always #5 clk = ~clk;
```

```
  // ========== Helper tasks ==========
  // Print Register File (32 registers in 4 rows)
  task automatic print_regfile;
    begin
      $display("==== Register File ====");
      $display("r00=%0h r01=%0h r02=%0h r03=%0h r04=%0h r05=%0h r06=%0h r07=%0h", dut.mips.dp.rf.rf[0],
               dut.mips.dp.rf.rf[1], dut.mips.dp.rf.rf[2], dut.mips.dp.rf.rf[3],
               dut.mips.dp.rf.rf[4], dut.mips.dp.rf.rf[5], dut.mips.dp.rf.rf[6],
               dut.mips.dp.rf.rf[7]);
      $display("r08=%0h r09=%0h r10=%0h r11=%0h r12=%0h r13=%0h r14=%0h r15=%0h", dut.mips.dp.rf.rf[8],
               dut.mips.dp.rf.rf[9], dut.mips.dp.rf.rf[10], dut.mips.dp.rf.rf[11],
               dut.mips.dp.rf.rf[12], dut.mips.dp.rf.rf[13], dut.mips.dp.rf.rf[14],
               dut.mips.dp.rf.rf[15]);
      $display("r16=%0h r17=%0h r18=%0h r19=%0h r20=%0h r21=%0h r22=%0h r23=%0h", dut.mips.dp.rf.rf[16],
               dut.mips.dp.rf.rf[17], dut.mips.dp.rf.rf[18], dut.mips.dp.rf.rf[19],
               dut.mips.dp.rf.rf[20], dut.mips.dp.rf.rf[21], dut.mips.dp.rf.rf[22],
               dut.mips.dp.rf.rf[23]);
      $display("r24=%0h r25=%0h r26=%0h r27=%0h r28=%0h r29=%0h r30=%0h r31=%0h", dut.mips.dp.rf.rf[24],
               dut.mips.dp.rf.rf[25], dut.mips.dp.rf.rf[26], dut.mips.dp.rf.rf[27],
               dut.mips.dp.rf.rf[28], dut.mips.dp.rf.rf[29], dut.mips.dp.rf.rf[30],
               dut.mips.dp.rf.rf[31]);
    end
  endtask

  // Print Data Memory (64 words in 8-word rows)
  task automatic print_dmem_table;
    integer i;
    begin
      $display("==== Data Memory (word-aligned, 64 words) ====");
      for (i = 0; i < 64; i = i + 8) begin
        $display(
            "RAM[%02d]=%h RAM[%02d]=%h RAM[%02d]=%h RAM[%02d]=%h RAM[%02d]=%h RAM[%02d]=%h RAM[%02d]=%h RAM[%02d]=%h",
            i, dut.dmem.RAM[i], i + 1, dut.dmem.RAM[i+1], i + 2, dut.dmem.RAM[i+2], i + 3,
            dut.dmem.RAM[i+3], i + 4, dut.dmem.RAM[i+4], i + 5, dut.dmem.RAM[i+5], i + 6,
            dut.dmem.RAM[i+6], i + 7, dut.dmem.RAM[i+7]);
      end
    end
  endtask
```

```
    // ============ Checker + Timeout ============
    wire [31:0] word_index = dataadr[31:2];  // convert byte addr to word index
    integer cycles;

    initial cycles = 0;
    always @(posedge clk) cycles <= cycles + 1;

    // Timeout to avoid infinite simulation
    always @(posedge clk) begin
      if (cycles > 1000) begin
        $display("    ");
        $display("TIMEOUT: cycles=%0d; ", cycles);
        $display("    ");
        $finish;
      end
    end
```

```
  // Monitor memory writes and check pass/fail
  always @(negedge clk) begin
    if (memwrite) begin
      // Log memory write event
      $display("    ");
      $display(" --------------  Log memory write event --------------");
      $display(
          "T=%0t ns : PC=0x%08h INSTR=0x%08h | memwrite addr=%0d (0x%08h) -> RAM[%0d] = %0d (0x%08h)",
          $time, dut.pc, dut.instr, dataadr, dataadr, word_index, writedata, writedata);
      $display("    ");
      // Check success condition
      if ((dataadr == 32'd84) && (writedata == 32'd7)) begin
        $display(
            "==================== Simulation succeeded : dataadr=84 -> RAM[%0d]=7 ====================",
            word_index);
        @(posedge clk);
        #1;  // small delay to allow memory update
        print_regfile();
        print_dmem_table();
        $finish;
      end  // Fail if unexpected address (except addr=80 which is an intermediate write)
      else if (dataadr !== 32'd80) begin
        $display("Simulation failed at T=%0t ns : unexpected write to addr=%0d data=%0d", $time,
                 dataadr, writedata);
        $finish;
      end
    end
  end
endmodule
```

Command for simulation on terminal

Step 1 : iverilog -g2005 -o sim.vvp -s testbench *.v

Step 2 : vvp sim.vvp

Step 3 : gtkwave sim.vcd

Demonstration (Test_Program)

```
# mipstest.asm
# David_Harris@hmc.edu 9 November 2005
#
# Test the MIPS processor.
# add, sub, and, or, slt, addi, lw, sw, beq, j
# If successful, it should write the value 7 to address 84

#        Assembly                Description                  Address   Machine
main:   addi $2, $0, 5           # initialize $2 = 5            0        20020005
        addi $3, $0, 12          # initialize $3 = 12           4        2003000c
        addi $7, $3, -9          # initialize $7 = 3            8        2067fff7
        or   $4, $7, $2          # $4 <= 3 or 5 = 7             c        00e22025
        and  $5, $3, $4          # $5 <= 12 and 7 = 4           10       00642824
        add  $5, $5, $4          # $5 = 4 + 7 = 11              14       00a42820
        beq  $5, $7, end         # shouldn't be taken           18       10a7000a
        slt  $4, $3, $4          # $4 = 12 < 7 = 0              1c       0064202a
        beq  $4, $0, around      # should be taken              20       10800001
        addi $5, $0, 0           # shouldn't happen             24       20050000
around: slt  $4, $7, $2          # $4 = 3 < 5 = 1               28       00e2202a
        add  $7, $4, $5          # $7 = 1 + 11 = 12             2c       00853820
        sub  $7, $7, $2          # $7 = 12 - 5 = 7              30       00e23822
        sw   $7, 68($3)          # [80] = 7                     34       ac670044
        lw   $2, 80($0)          # $2 = [80] = 7                38       8c020050
        j    end                 # should be taken              3c       08000011
        addi $2, $0, 1           # shouldn't happen             40       20020001
end:    sw   $2, 84($0)          # write adr 84 = 7             44       ac020054
```

Ref : Digital Design and Computer Architecture , David M. Harris & Sarah L. Harris

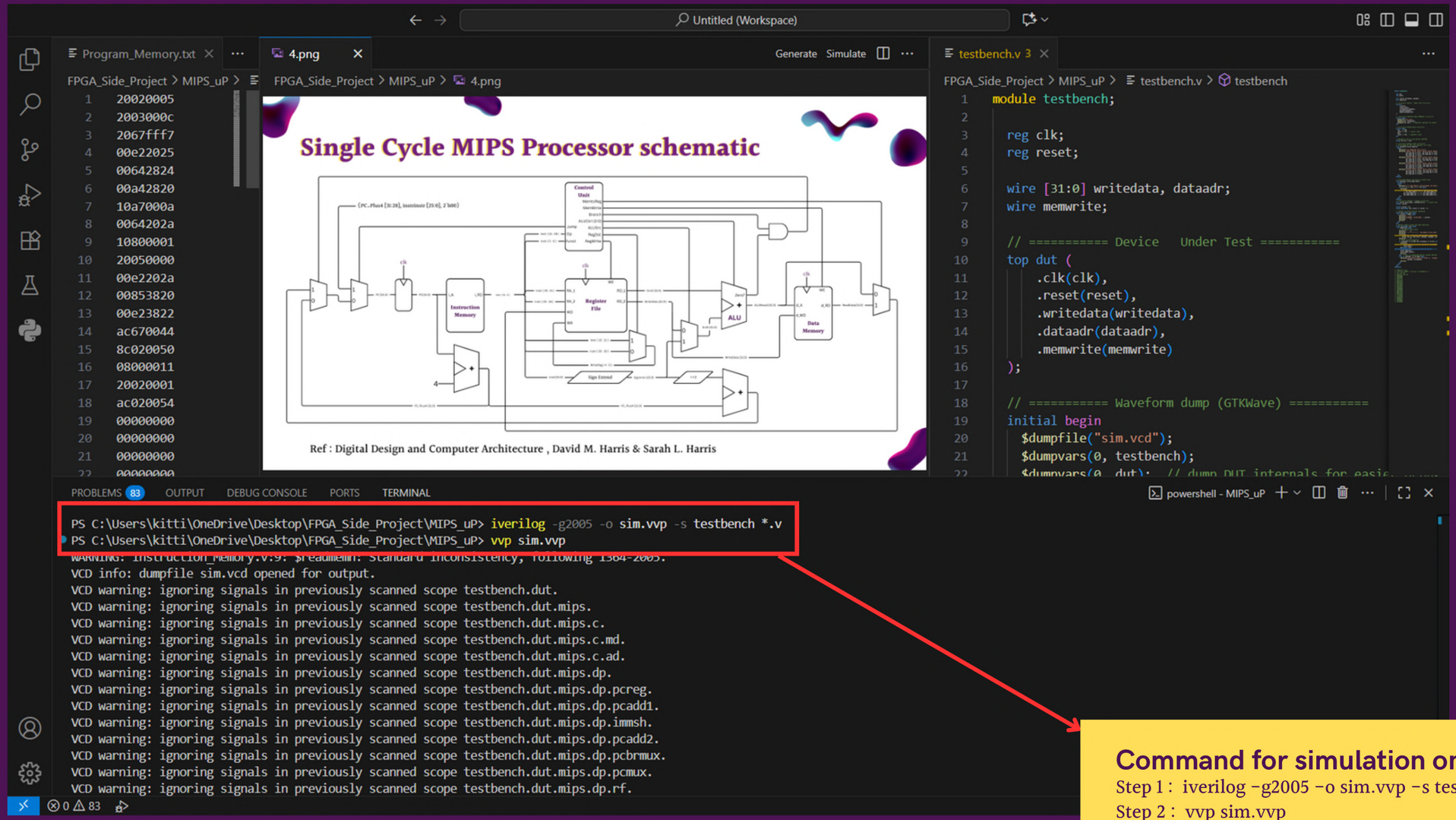
Demonstration (Test_Program)

```
1   module Instruction_Memory (
2       input  [ 5:0] a,
3       output [31:0] rd
4   );
5
6     reg [31:0] RAM[63:0];
7
8     initial begin
9       $readmemh("Program_Memory.txt", RAM);
10    end
11
12    assign rd = RAM[a];
13  endmodule
```

```
1   20020005 33   00000000
2   2003000c 34   00000000
3   2067fff7 35   00000000
4   00e22025 36   00000000
5   00642824 37   00000000
6   00a42820 38   00000000
7   10a7000a 39   00000000
8   0064202a 40   00000000
9   10800001 41   00000000
10  20050000 42   00000000
11  00e2202a 43   00000000
12  00853820 44   00000000
13  00e23822 45   00000000
14  ac670044 46   00000000
15  8c020050 47   00000000
16  08000011 48   00000000
17  20020001 49   00000000
18  ac020054 50   00000000
19  00000000 51   00000000
20  00000000 52   00000000
21  00000000 53   00000000
22  00000000 54   00000000
23  00000000 55   00000000
24  00000000 56   00000000
25  00000000 57   00000000
26  00000000 58   00000000
27  00000000 59   00000000
28  00000000 60   00000000
29  00000000 61   00000000
30  00000000 62   00000000
31  00000000 63   00000000
32  00000000 64   00000000
```

Program_Memory.txt

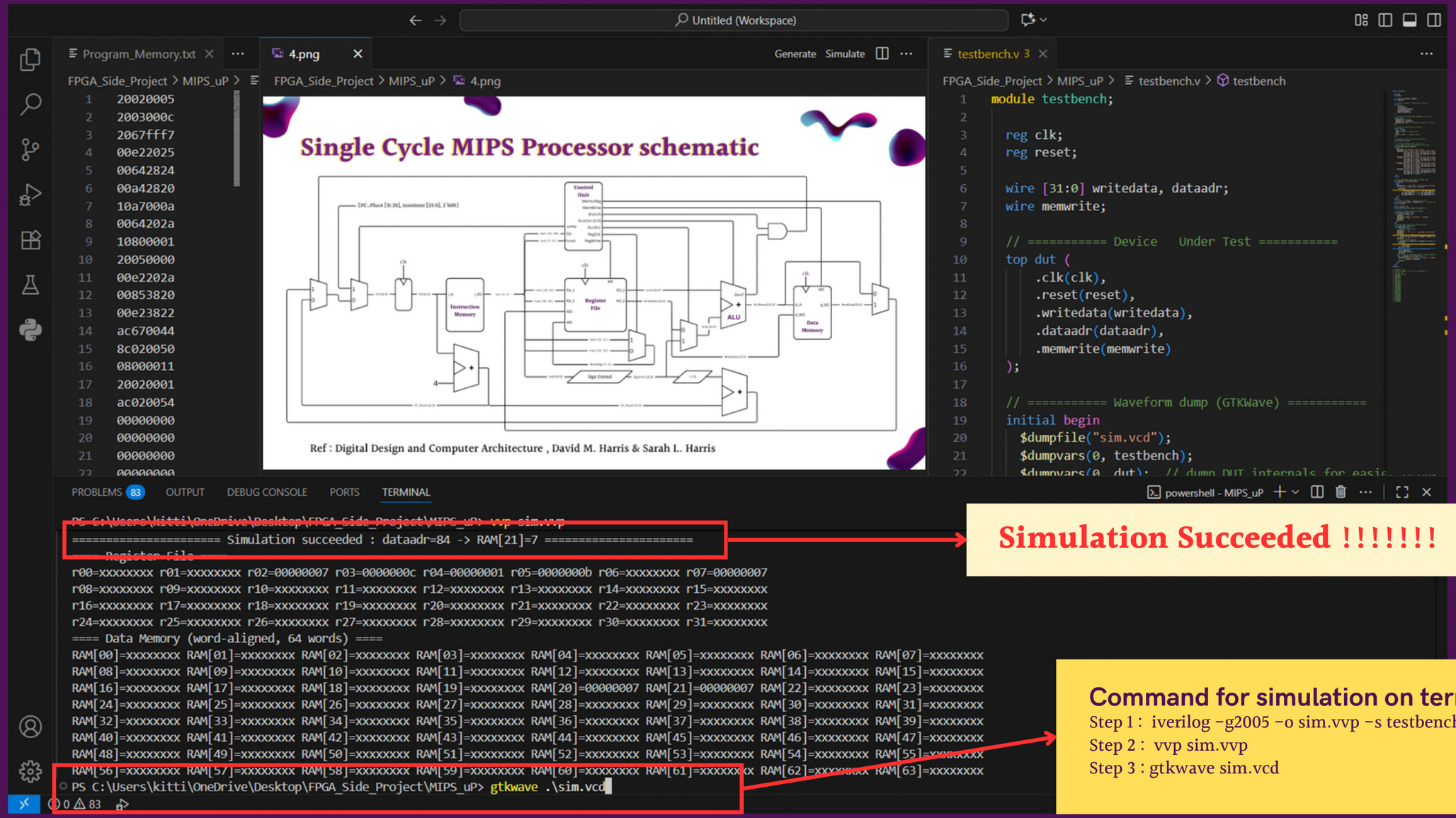
Demonstration (VS-Code)



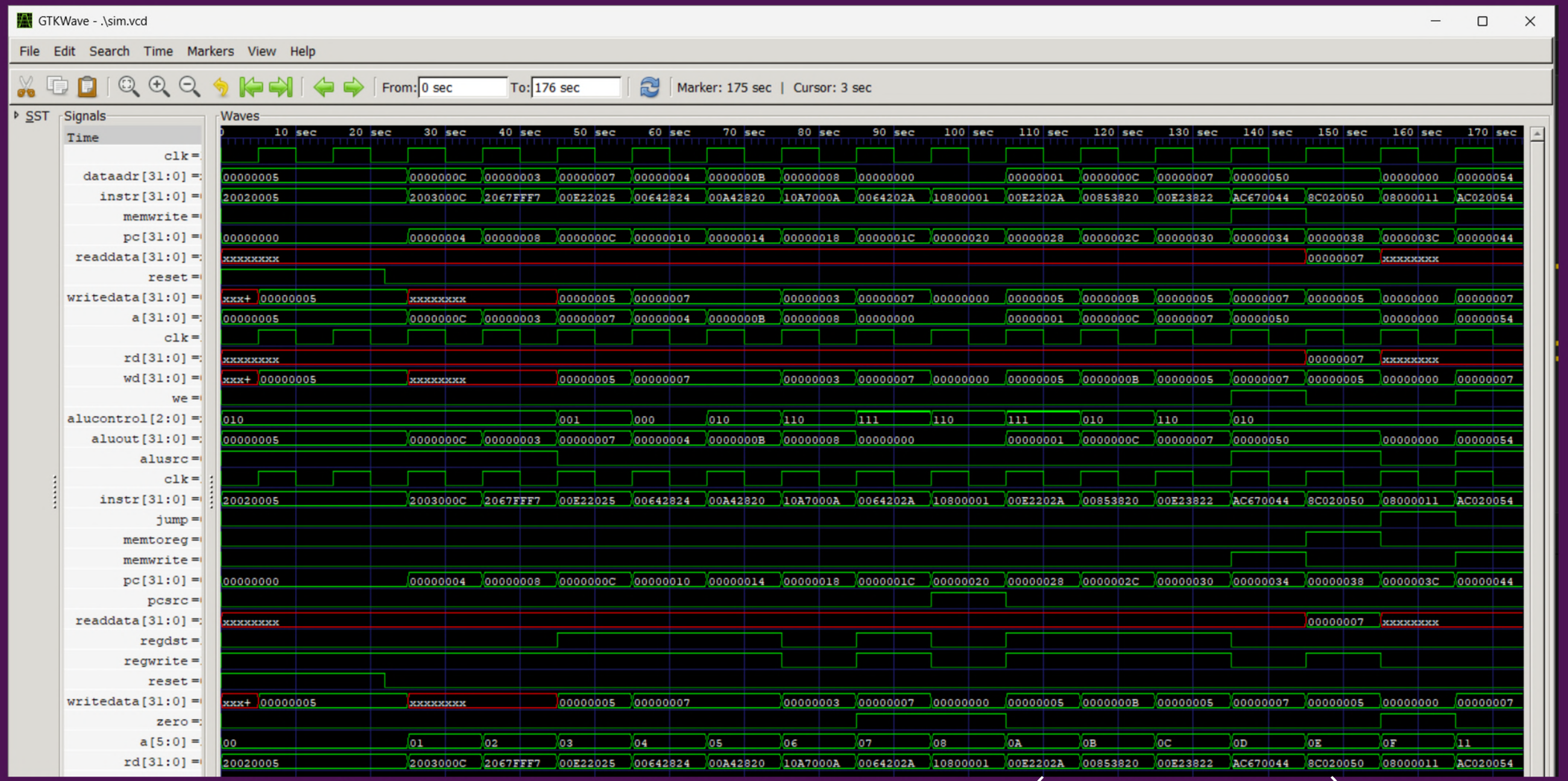
Command for simulation on terminal

Step 1 : iverilog -g2005 -o sim.vvp -s testbench *.v
Step 2 : vvp sim.vvp
Step 3 : gtkwave sim.vcd

Demonstration (VS-Code)



Demonstration (VS-Code)



Signal Operating of MIPS Processor (on GTKWave)

Thank you for your attention. I hope this project inspires you as much as it reflects my passion for Hardware Electronics.

– Kittiphop Phanthachart , MACKIES –

16 September 2025