2023 Computer Organization Project Report

Developer's instructions

Name	SID	Contribution ratio	Work
Ruixiang Jiang	12111611	33%	ALU, IFetch, CPUTop, Led, Switch, Seg, Uart, Report
Yujing Zhang	12111944	33%	Dmemory, MemOrIO, CPUTop, Led, Switch
Yilun Qiu	12013006	33%	Assembly files, Decoder, Controller, Uart

Version modification record

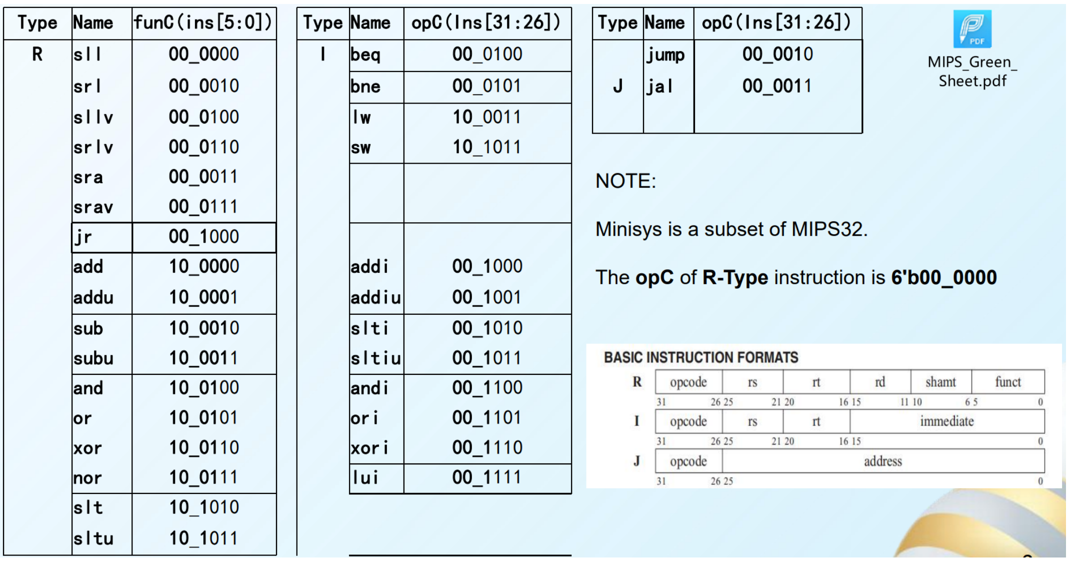
- $\mathrm{v}1.0$ (05-14): Basic modules completed
- $\mathrm{v}1.1$ (05-20): Top module completed
- $\mathrm{v}1.2$ (05-21): Uart completed
- Final Version $\mathrm{v}1.3$(05-23): Assembly part completed

CPU architecture design specification

- **CPU Features**
  - Instruction Set Architecture

    Registers: number = 32, width = 32 bits

    Instruction set: Basic Minisys + mul + mult + div + divu + mfhi + mflo



Type	Name	funC(ins[5:0])
R	sll	00_0000
	srl	00_0010
	sllv	00_0100
	srlv	00_0110
	sra	00_0011
	srav	00_0111
	jr	00_1000
	add	10_0000
	addu	10_0001
	sub	10_0010
	subu	10_0011
	and	10_0100
	or	10_0101
	xor	10_0110
	nor	10_0111
	slt	10_1010
	sltu	10_1011

Type	Name	opC(Ins[31:26])
I	beq	00_0100
	bne	00_0101
	lw	10_0011
	sw	10_1011
	addi	00_1000
	addiu	00_1001
	slti	00_1010
	sltiu	00_1011
	andi	00_1100
	ori	00_1101
	xori	00_1110
	lui	00_1111

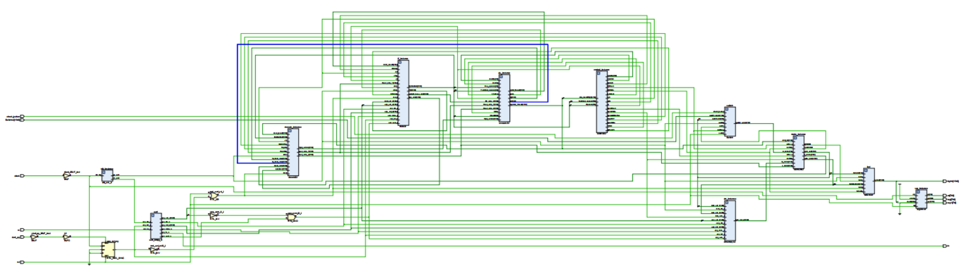
Type	Name	opC(Ins[31:26])
J	jump	00_0010
	jal	00_0011

  - Address Space Design
    - Architecture: Harvard architecture, instruction memory is stored separately from data memory
    - Addressing unit: Byte

        - Instruction space: $0x00000000$ ~ $0xFFFFFFFF$
        - Data space: $0x00000000$ ~ $0xFFFFFC00$
    - External I/O Devices: Accessing I/O using polling

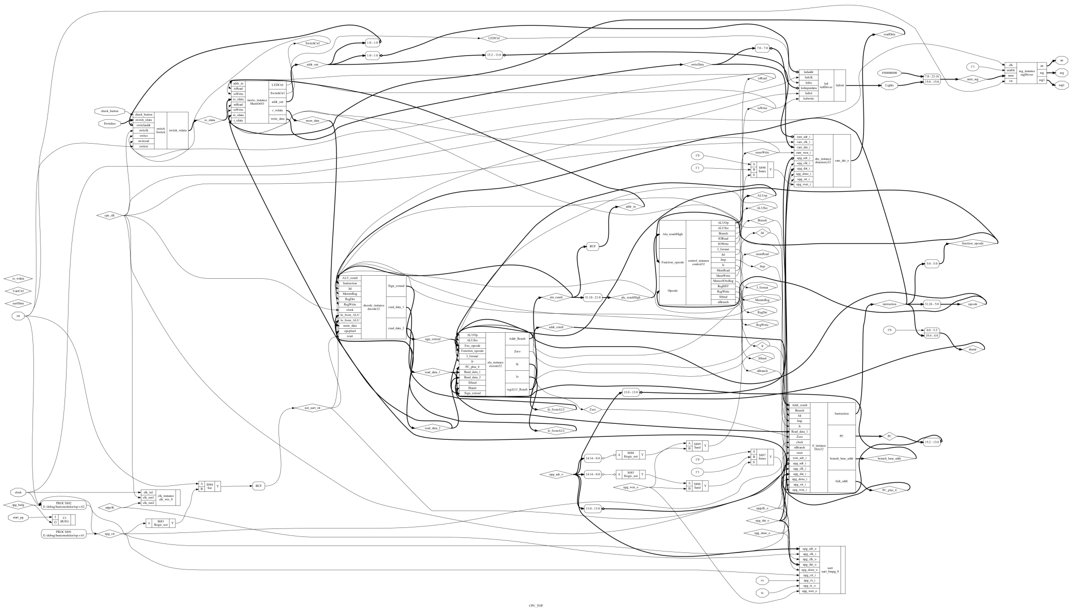
Device	Address
Left LED	$0xFFFFFC62$
Right LED	$0xFFFFFC60$
Left Switch	$0xFFFFFC72$
Right Switch	$0xFFFFFC70$
Enter Button	$0xFFFFFC73$
Segment	$0xFFFFFC80$

    - CPI: single cycle CPU
    - CPU Frequency: 10MHz
- **CPU interface**
    - Clock: Built-in clock interface of EGO1 development board
    - Reset: R1 of EGO1 development board
    - Uart: Use Uart IP core
    - Switch: Left switches for input data, and right switches for input instruction
    - 16-bit-width Led: Output result in a binary number
    - 8-bit-width Seg: Output arithmetic result with max value = 99999999
- **Internal Structure of CPU**
    - Interface Connections among Submodules within CPU



HD Figure: https://ooad-1312953997.cos.ap-guangzhou.myqcloud.com/cpu/overall1.pdf

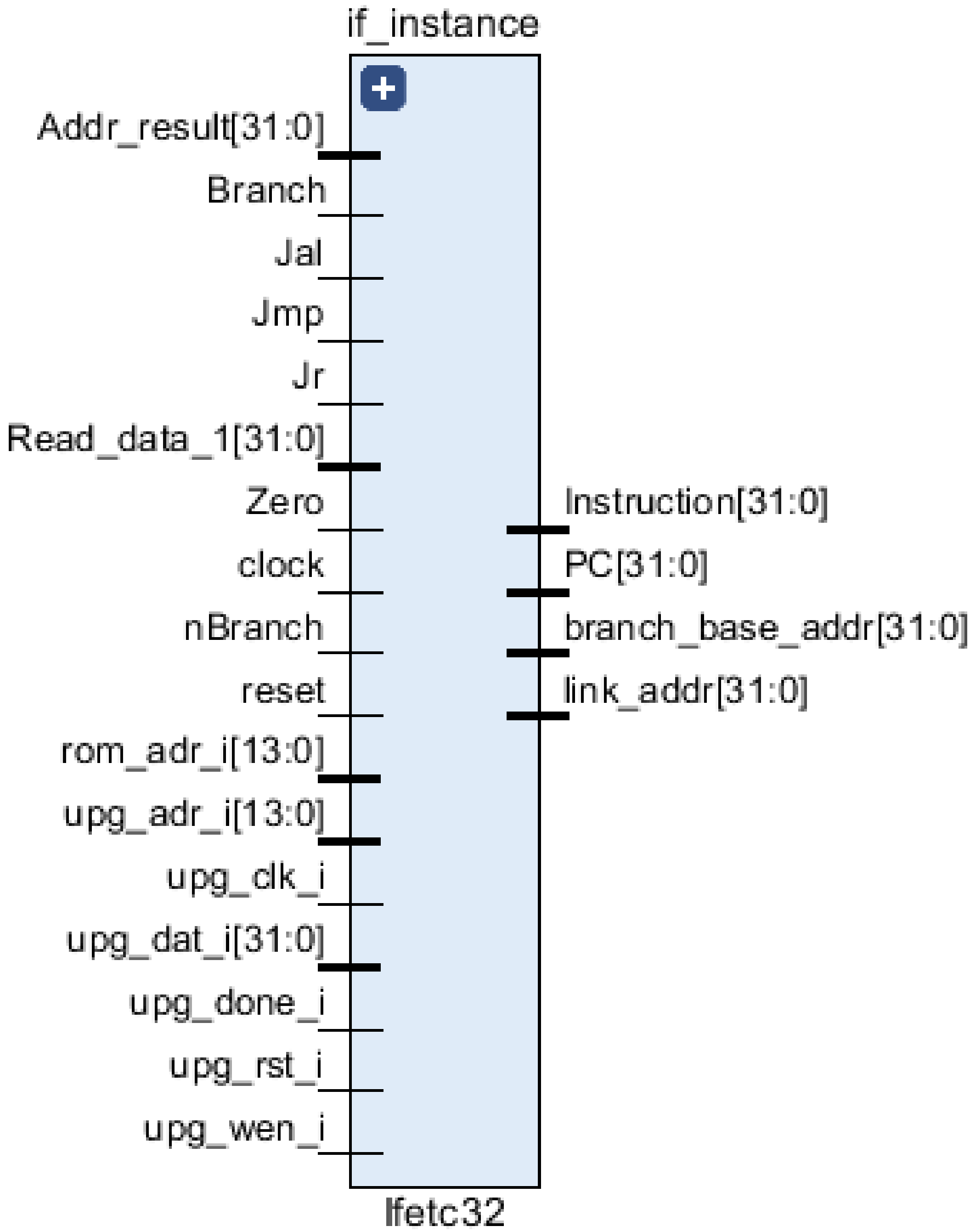
Note: generated from VIVADO RTL analysis.



HD Figure: https://ooad-1312953997.cos.ap-guangzhou.myqcloud.com/cpu/overall2.png

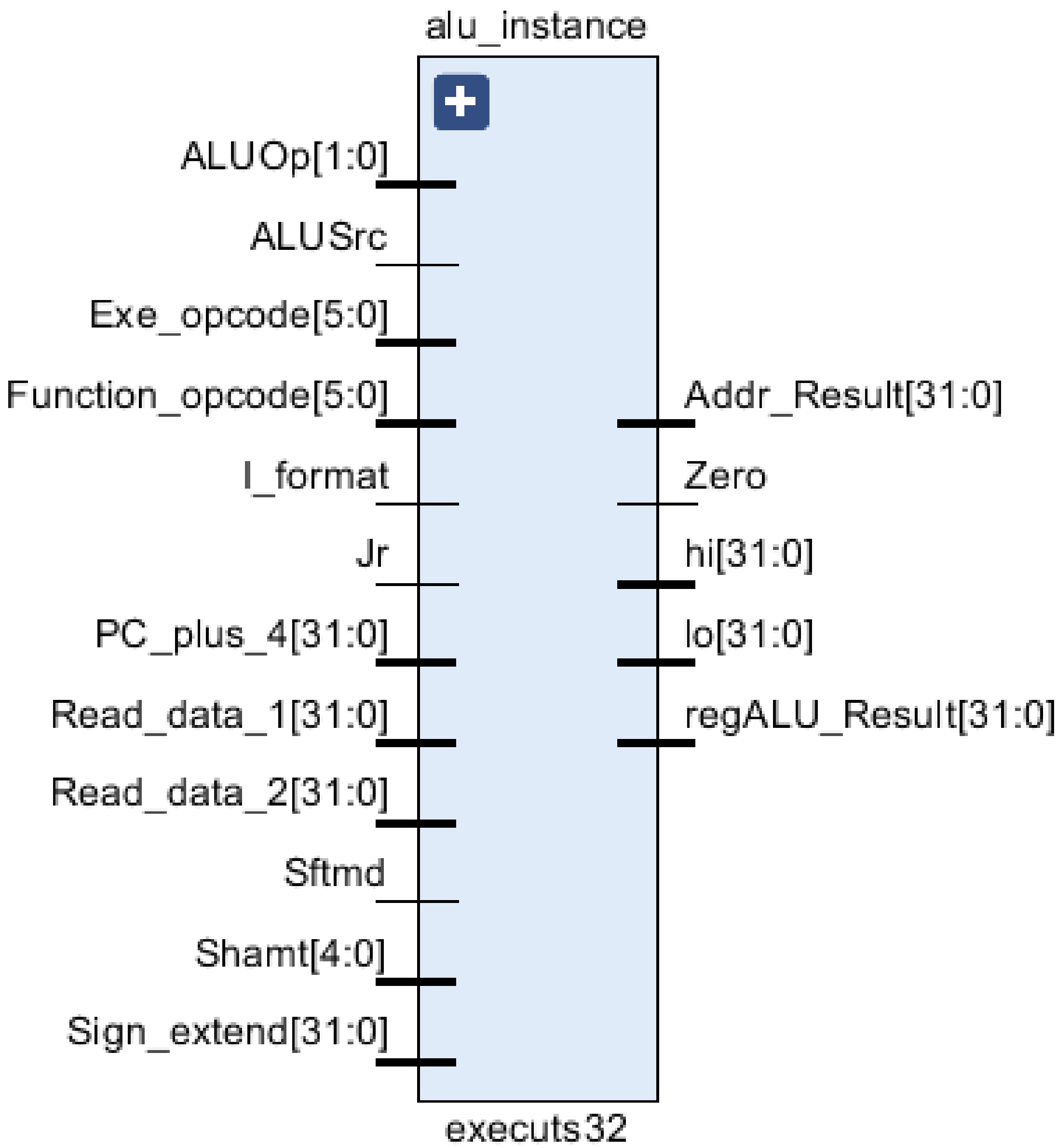
Note: use the tool developed by SE Group.2339, Github URL: https://github.com/sustech-cs304/team-project-2339

- Submodules
- Instruction Fetcher (Ifetch32)
  The instruction fetch stage involves retrieving the instruction from the memory. The program counter (PC) is responsible for providing the memory address of the current instruction. This address is sent to the instruction memory, which fetches the instruction stored at that address and transfers it to the instruction register (IR). Additionally, the program counter is incremented to point to the subsequent instruction in memory.

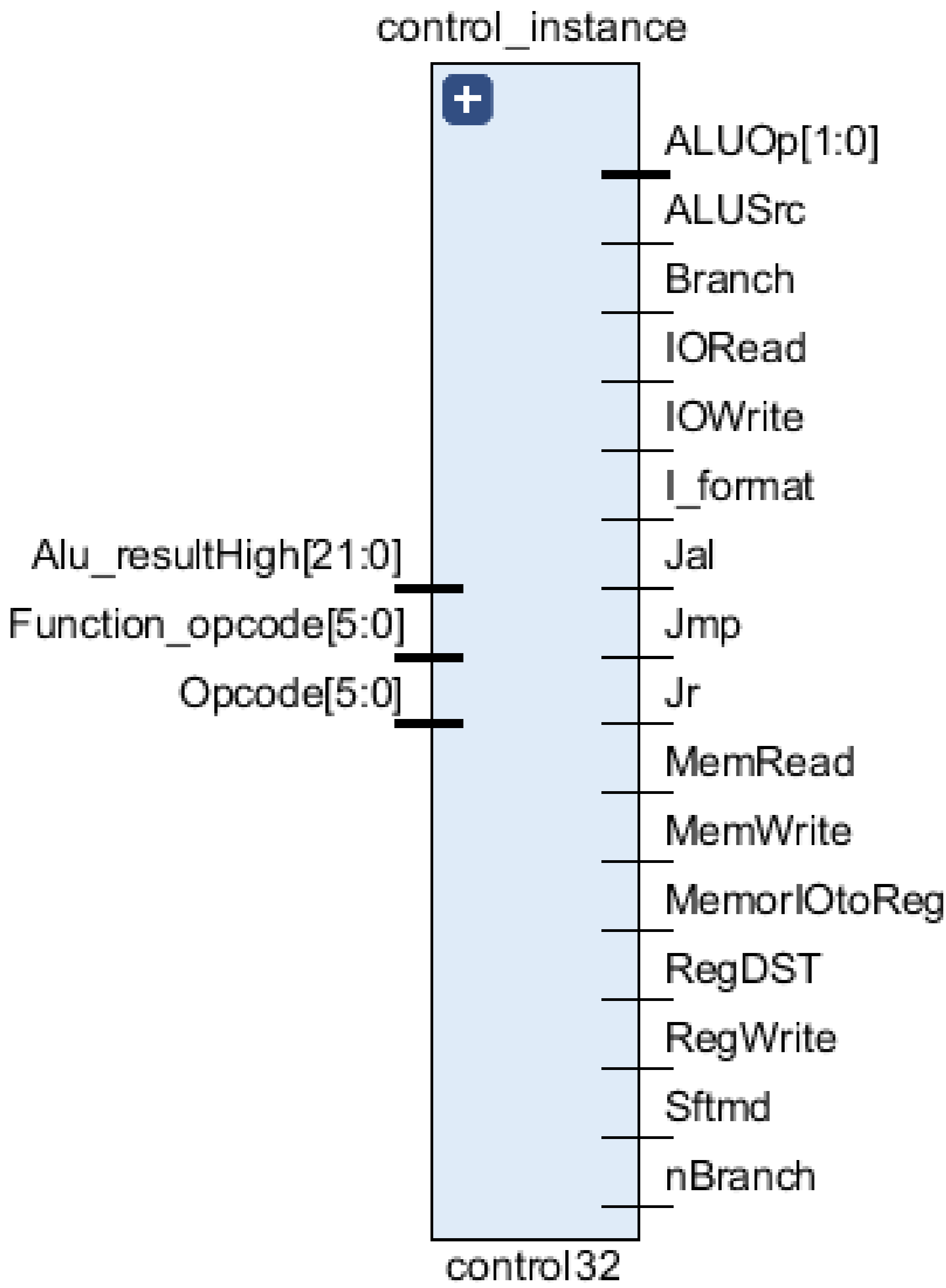


- Decoder (decode32)
  It performs as an instruction decoder stage, the received instruction is analyzed and decoded. The control unit examines the opcode of the instruction and determines the required control signals for subsequent stages. By interpreting the opcode, the control unit configures the CPU components accordingly, enabling the appropriate data paths and control signals necessary for executing the instruction.

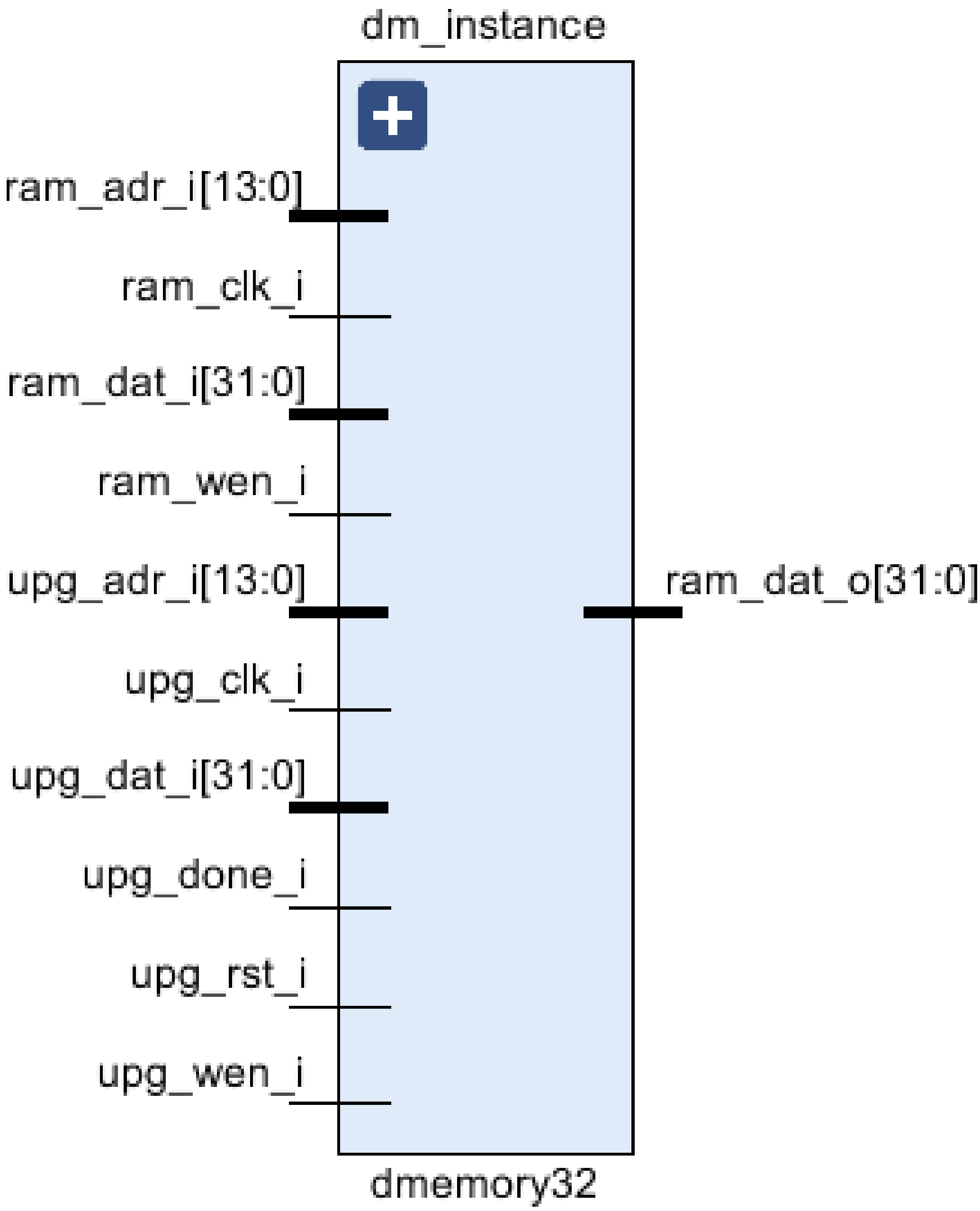
- ALU (executs32)
  The ALU carries out arithmetic and logical operations on the data within the CPU. Depending on the specific instruction, the ALU performs operations such as addition, subtraction, logical AND, logical OR, and other specified computations. The ALU takes inputs from the general-purpose registers and applies the operation indicated by the control signals received from the control unit.



- Controller (control32)
  The controller is responsible for generating the required control signals to coordinate the activities of the CPU components. It examines the decoded instruction and produces control signals to enable or disable specific registers, select the appropriate source and destination registers, activate the ALU, and govern memory operations. The controller plays a crucial role in orchestrating the overall execution of the instruction.



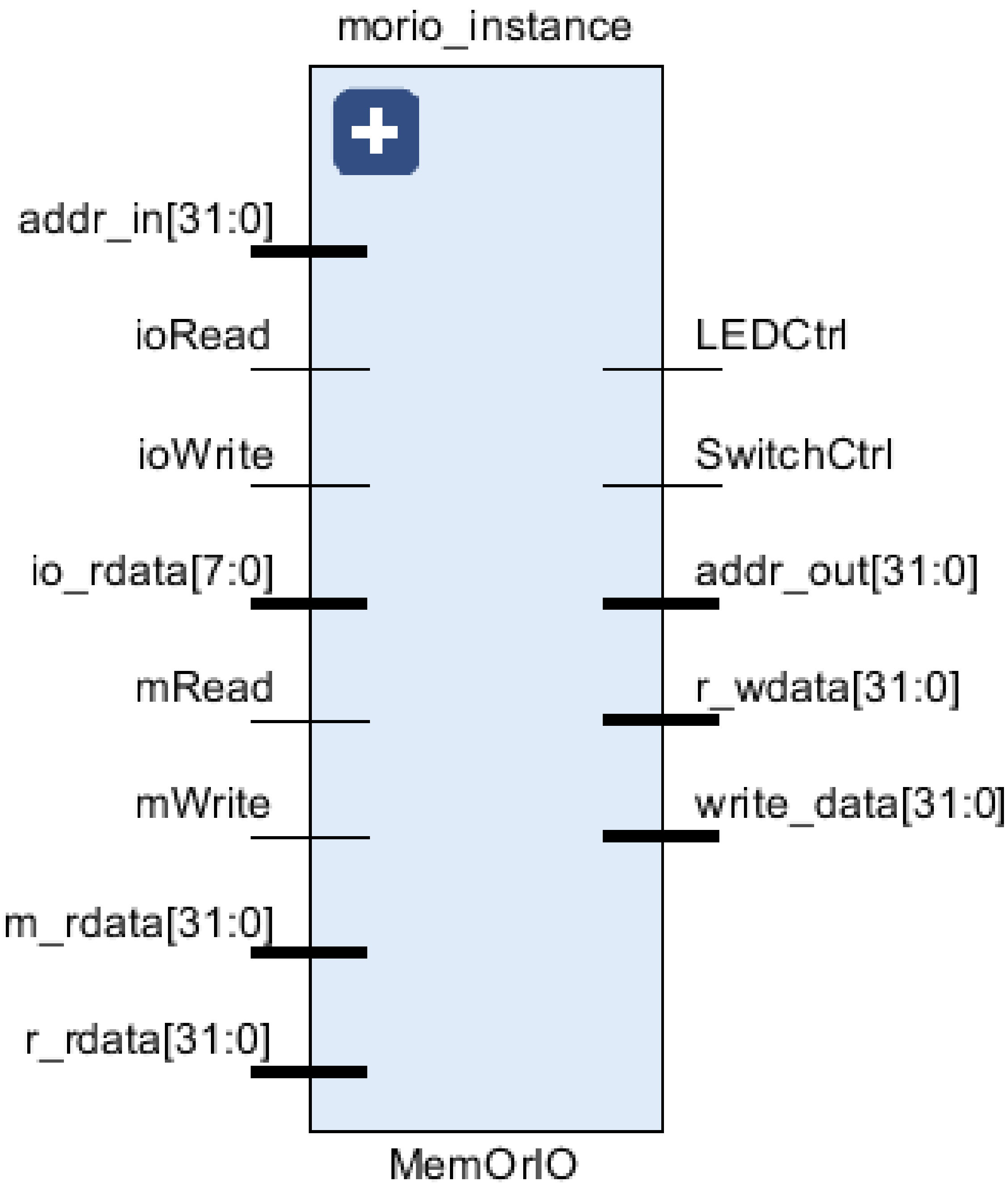
- Data Memory (dmemory32)
  The data memory stage involves accessing the memory unit for reading and writing operations. The data memory unit reads data from the specified address (in the case of a load operation) or writes data to the specified address (in the case of a store operation). Here we use an IP core to simulate RAM.



- MemOrIO (MemOrIO)
  During the MemOrIO stage, the CPU interacts with external memory or I/O devices to either read data from memory or write data to memory or I/O devices. This stage involves transferring data between the CPU and the memory or I/O devices, as well as handling any necessary address calculations or data transfers.
  It contains the following operations:
  - Memory Read: If the instruction requires reading data from memory, the memory address calculated in earlier stages (typically stored in the Memory Address Register, MAR) is used to fetch the data from the memory. The fetched data is then temporarily stored in the Memory Data Register (MDR) within the CPU.
  - Memory Write: If the instruction involves writing data to memory, the memory address and data to be written (typically stored in the MAR and MDR, respectively) are transferred from the CPU to the memory. The memory then stores the data at the specified memory address.
  - I/O Operations: In some cases, the MemOrIO stage can involve input/output operations instead of or in addition to memory operations. This includes communication with

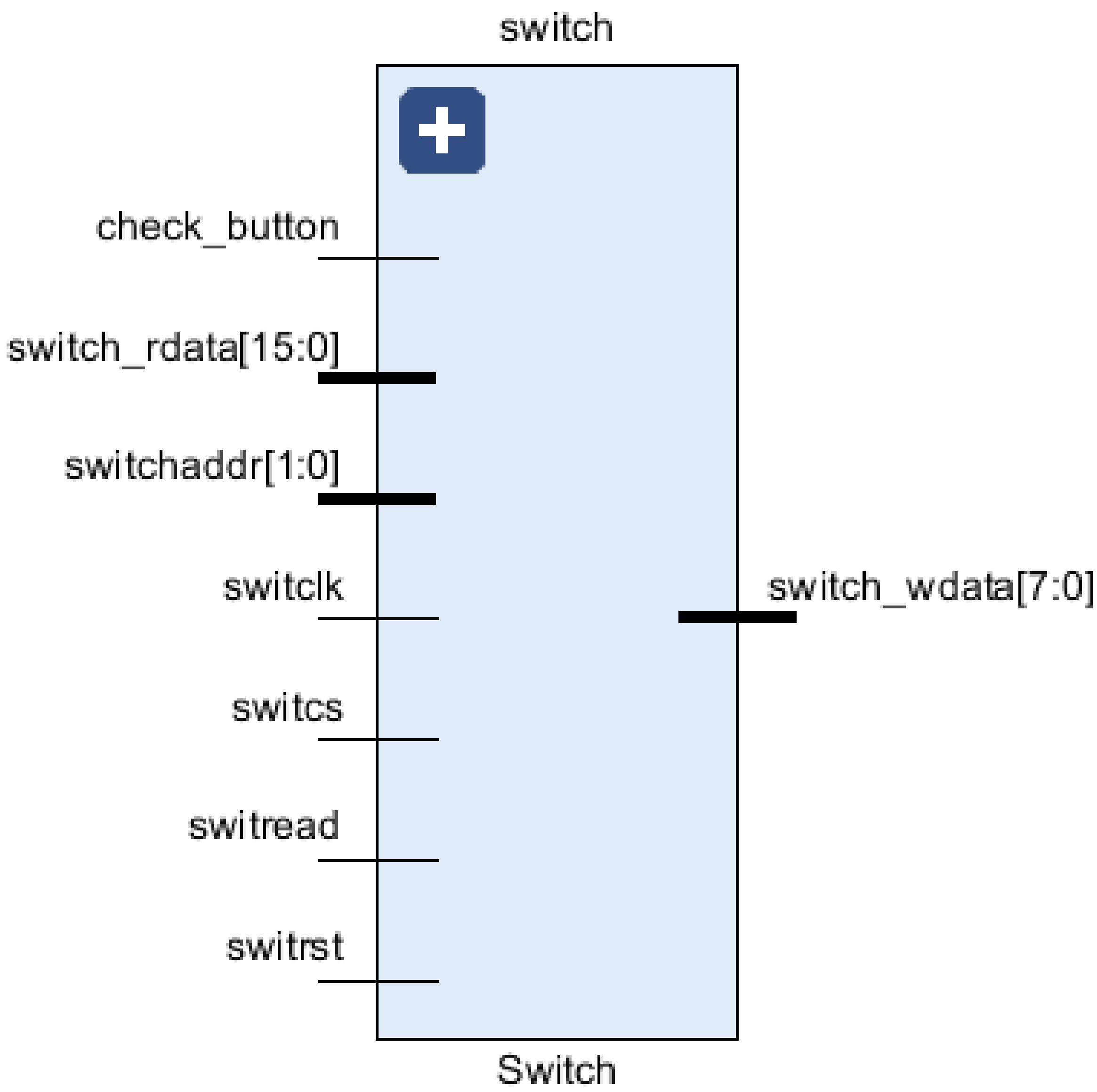
peripheral devices such as keyboards, displays, or storage devices. The CPU may send or receive data to or from these devices during this stage.

The specific operations and data transfers during the MemOrIO stage depend on the instruction being executed and the specific design of the CPU. The MemOrIO stage is an integral part of the single-cycle CPU architecture, ensuring that memory and I/O operations are properly handled within the CPU's instruction execution process.



- Switch Driver (Switch)
  The switches on the EGO1 development board can be toggled on or off to provide input signals to the board. The switch driver module would typically include the necessary circuitry and logic to read the state of the switches and provide the corresponding digital signals to the other components or modules on the board.

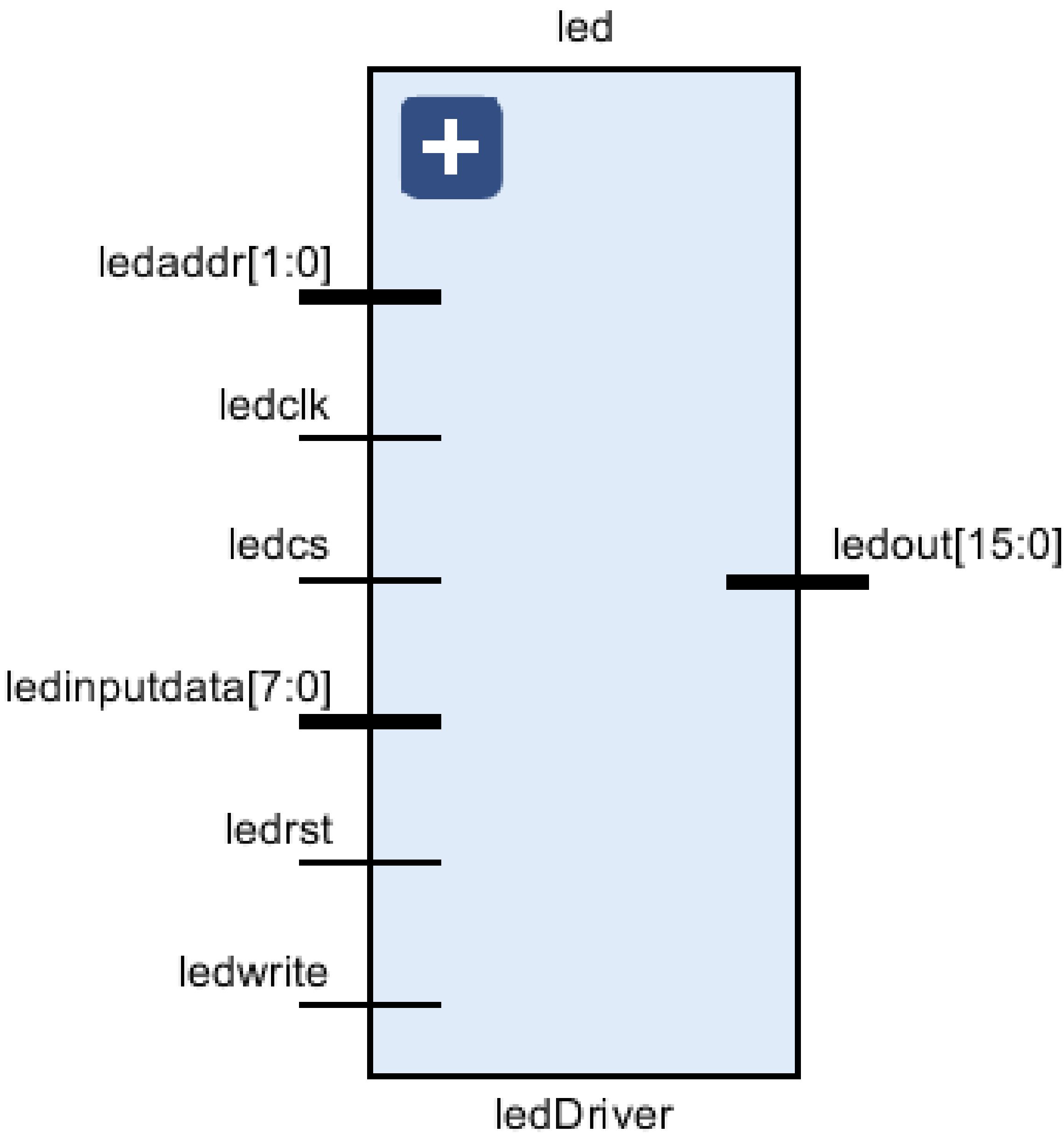
  The switch driver module enables the EGO1 board to read the status of the switches and use that information for various purposes, such as what we need to do in the basic test.



- LED Driver (ledDriver)

  The LED driver is an essential module responsible for controlling the LEDs (Light Emitting Diodes) on the board. Receiving input signals from various sources, such as the microcontroller or other modules on the board, it provides the necessary circuitry and logic to control the illumination of the LEDs based on the desired patterns or states, typically including the LED patterns, sequences, or behaviors.

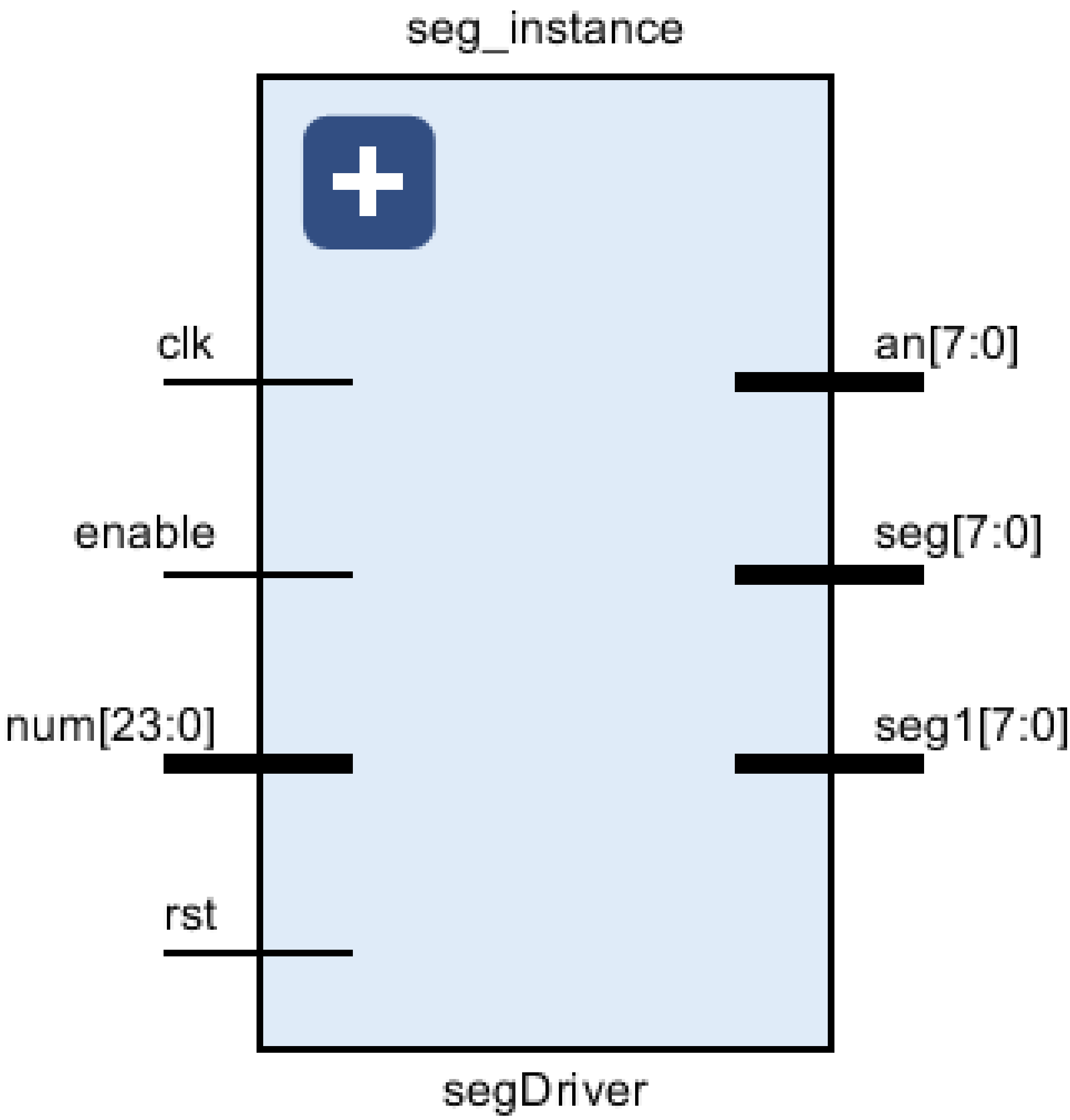
  With the LED driver module, the EGO1 development board can effectively control the illumination and behavior of the LEDs, providing visual feedback or indicators for various purposes, such as status indication, user interaction, and debugging information.



- Segment Driver (segDriver)
  The segment driver is responsible for controlling the seven-segment digital tube used to display numbers. It takes input signals, such as digital data representing numbers to be displayed, and generates the appropriate signals to activate the specific segments required to form the desired pattern.

  By controlling the activation and deactivation of the tubes, the seg driver module enables the EGO1 board to display numbers or characters on the 7-segment display. It can be programmed or configured to update the display in real-time, showing dynamic information, or to show static values based on the input provided.

  Special attention should be paid to the value of the clock cycle for the 7-segment display on the EGO1 development board.



Test instructions

- Test for Verilog

Method	Type	Detail	Result
Simulation	Unit	Test the 5 basic modules on OJ http://172.18.34.109/	Accepted
Synthesis	Module	Test whether the modules are sussessfully mixed	Accepted

- Test for MIPS

Detail	Result
Test waterfall lights	Accepted

Test single R-type instructions	Accepted
Test single I-type instructions	Accepted
Test single J-type instructions	Accepted
Test scene 1	Accepted
Test scene 2	Accepted

- Test Scene 1

Scenario 1. Testcase ID	**Testcase Description**	**Result**
3'b000	Enter the test number **a**, display **a** on the LED light. At the same time, use one LED light to determine **whether a is a power of two** (e.g. 8'h01 and 8'h10 are powers of two, the LED light is on. 8'ha0 and 8'h0a are not powers of two, the LED light is not on)	Accepted
3'b001	Input the test number **a**, display **a** on the output device. At the same time, use one LED light to display **whether a is an odd number**(e.g, 8'h01 and 8'hab are odd numbers, the LED light will be on. 8'ha0 and 8'hbc are not odd numbers, the LED light is not on)	Accepted
3'b010	Execute testcase 3'b111 first, then calculate the bitwise **OR** operation of **a** and **b**, and display the results on the output device	Accepted
3'b011	Execute testcase 3'b111 first, then calculate the bitwise **NOR** operation of **a** and **b**, and display the results on the output device	Accepted
3'b100	Execute test case 3'b111 first, then calculate the bitwise **XOR** operation of **a** and **b**, and display the results on the output device	Accepted
3'b101	First execute test case 3'b111, then execute the **SLT** instruction, **compare a and b as signed numbers**, and use the output device to demonstrate whether the relationship between a and b is valid.(Relationship established, light on, relationship not established, light off)	Accepted

3'b110	First execute test case 3 b111, then execute the **SLTU** instruction, **compare a and b as unsigned numbers**, and use the output device to demonstrate whether the relationship between a and b is valid(Relationship established, light on, relationship not established, light off)	Accepted
3'b111	Input test number a, input test number b, and display the values of a and b on the output device	Accepted

- Test Scene 2

Scenario 2. Testcase ID	Testcase Description	Result
3'b000	Enter the numerical value of **a** (**a is considered a signed number**), **calculate the cumulative sum of 1 to a,** and display the cumulative sum on the output device **(if a is a negative number, give a blinking prompt**)	Accepted
3'b001	Enter the numerical value of **a (a is considered an unsigned number), recursively calculate the sum of 1 to a**, record the number of times the stack was pushed and pushed, and **display the sum of the times the stack was pushed and popped on the output device**	Accepted
3'b010	Enter the numerical value of **a (a is considered an unsigned number**), **recursively calculate the sum of 1 to a,** record the data of stack entry and exit, and **display the parameters which is pushed to the stack on the output device**. **Each parameter of the stack is displayed for 2-3 seconds** (indicating that the output here does not pay attention to the stack entry and exit information of $ra)	Accepted
3'b011	Enter the numerical value of **a (a is considered an unsigned number**), **recursively calculate the sum of 1 to a,** record the data of stack entry and exit, and **display the parameters which is popped from the stack on the output device**. **Each parameter of the stack is displayed for 2-3 seconds** (indicating that the output here does not pay attention to the stack entry and exit information of $ra)	Accepted
3'b100	Input test number **a** and test number **b** to implement the **addition of signed numbers** (**a, b, and the sum of additions are all 8 bits, where the highest bit is considered the sign bit. If the sign bit is 1, it represents the 2's complement of the negative number**), and	Accepted

	determine whether overflow occurs. **Output the operation result and overflow judgment**	
3'b101	Input test number **a** and test number **b** to **subtract signed numbers** (**a, b, and the difference are all 8 bits, where the highest bit is considered as the sign bit. If the sign bit is 1, it represents the 2's complement of the negative number**), and determine whether overflow occurs. Output the operation result and overflow judgment	Accepted
3'b110	Input test number **a** and test number **b** to implement **the multiplication of signed numbers (a and b are both 8 bits, the product is 16 bits, and the highest bit is considered as the sign bit. If the sign bit is 1, it represents the 2's complement of the negative number)**, and **output the product**	Accepted
3'b111	Input test number **a** and test number **b** to achieve **division of signed numbers (a, b, quotient and remainder are both 8 bits, where the highest bit is considered the sign bit. If the sign bit is 1, it represents the complement of the negative number**), and **output quotient and remainder (quotient and remainder are displayed alternately, each lasting for 5 seconds)**	Accepted

- **Summary**: We have passed all tests for both Verilog and MIPS, our CPU and assembly files are correct.

Bonus Part

Function

Implement the Uart Interface

We have completed the uart function based on the lab slides (lab13).
CPU_TOP.v : add inputs and outputs definition as well.

```
// Uart
wire[15:0] uartData;
wire upgclk;
wire upgclk_o;
wire upg_wen_o;
wire upg_done_o; // iFpgaUartFromPC finish
wire[14:0] upg_adr_o; // data to which memory unit of rom/dmemory
wire[31:0] upg_dat_o; // data to rom or Dmemory
wire spg_bufg;
BUFG U1(.I(start_pg), .O(spg_bufg)); // de-twitter
reg upg_rst = 1; // generate uart rst signal
always @(posedge clock) begin
    if (spg_bufg) upg_rst = 0;
    if (rst) upg_rst = 1;
end
wire not_uart_rst = rst | (!upg_rst);
```

```
// CPU works on normal/uart mode when kickOff = 1/0
uart_bmpg_0 uart  (.upg_adr_o(upg_adr_o),
       .upg_clk_i(upgclk),
       .upg_clk_o(upgclk_o),
       .upg_dat_o(upg_dat_o),
       .upg_done_o(upg_done_o),
       .upg_rst_i(upg_rst),
       .upg_rx_i(rx),
       .upg_tx_o(tx),
       .upg_wen_o(upg_wen_o));

clk_wiz_0 clk_instance(
    .clk_in1(clock),
    .clk_out1(cpu_clk),
    .clk_out2(upgclk)
);
```

dmemory.v : add inputs and outputs definition as well.

```
module dmemory32(ram_clk_i, ram_wen_i, ram_adr_i, ram_dat_i, ram_dat_o,
                upg_rst_i, upg_clk_i, upg_wen_i, upg_adr_i, upg_dat_i,
upg_done_i);

    input ram_clk_i;
    input ram_wen_i;
    input [13:0] ram_adr_i;
    input [31:0] ram_dat_i;
    output [31:0] ram_dat_o;
    input upg_rst_i;
    input upg_clk_i;
    input upg_wen_i;
    input [13:0] upg_adr_i;
    input [31:0] upg_dat_i;
    input upg_done_i;

    wire ram_clk = !ram_clk_i;

    wire kickOff = upg_rst_i | (~upg_rst_i & upg_done_i);

    // ram
    RAM ram(
        .clka (kickOff? ram_clk:upg_clk_i),
        .wea (kickOff? ram_wen_i:upg_wen_i),
        .addra (kickOff? ram_adr_i:upg_adr_i),
        .dina (kickOff? ram_dat_i:upg_dat_i),
        .douta (ram_dat_o)
    );

endmodule
```

Extended instruction type

We implement 6 extension instructions: `mult`, `multu`, `div`, `divu`, `mflo`, `mfhi`

Thoughts:

To implement these instructions, we need to use two extra registers `hi` register and `lo` register to store the result of the instructions. In such case, we need to add codes in the ALU module to support multiplication and division. Meanwhile, in the Decoder modulem, we need to initialize `hi` and `lo` registers and consider when and how to write the corresponding registers.

Codes:

- ALU (executs32):
  We need `hi` and `lo` registers as the output of the module.

```
output reg[31:0] hi,
output reg[31:0] lo
```

We add a piece of combinational logic codes to get the results of `hi` and `lo` registers. Such codes can get correct results of the extended instructions.

```
always @(*) begin
    if (Exe_opcode == 6'b000000) begin
        case (Function_opcode)
            6'b01_1000: {hi, lo} = $signed(Ainput) * $signed(Binput); // mult
            6'b01_1001: {hi, lo} = Ainput * Binput; // multu
            6'b01_1010: begin // div
                lo = $signed(Ainput) / $signed(Binput);
                hi = $signed(Ainput) % $signed(Binput);
            end
            6'b01_1011: begin // divu
                lo = Ainput / Binput;
                hi = Ainput % Binput;
            end
            default: {hi, lo} = 64'b0;
        endcase
    end
    else {hi, lo} = 64'b0;
end
```

- Decoder (decode32)
  We need `hi_from_ALU` and `lo_from_ALU` as the input of the module. These registers are passed from the ALU mudole.

```
input [31:0] hi_from_ALU; // the hi register result from ALU
input [31:0] lo_from_ALU; // the lo register result from ALU
```

We need to judge whether to use the result of `hi` register and `lo` register, and we need to judge whether the instruction is `mflo` or `mfhi`.

```
wire hi_lo_calculate = Instruction[31:26] == 6'b000000 &&
    (Instruction[5:0] == 6'b011000 ||
     Instruction[5:0] == 6'b011001 ||
     Instruction[5:0] == 6'b011010 ||
     Instruction[5:0] == 6'b011011); // judge whether the instruction needs
hi/lo
wire mflo = (Instruction[31:26] == 6'b000000 && Instruction[5:0] == 6'b010010)?
1'b1: 1'b0; // judge whether the instruction is mflo
wire mfhi = (Instruction[31:26] == 6'b000000 && Instruction[5:0] == 6'b010000)?
1'b1: 1'b0; // judge whether the instruction is mfhi
```

We add a piece of combinational logic codes in the decoder module. Such codes are required to initialize the `hi` and `lo` registers and move the values of `hi` and `lo` registers to the specific registers.

```
    integer i;
    always @(posedge clock) begin
        if (reset) begin
            for (i = 0;i < 32;i = i + 1) registers[i] <= 32'b0; // initialize
registers
            hi <= 32'b0; // initialize hi
            lo <= 32'b0; // initialize lo
        end
        else begin
            if (RegWrite && writeReg) begin
                if (Jal) begin
                    registers[writeReg] <= opcplus4; // write the address of the
next instruction to the register
                end
                else if (MemtoReg) begin
                    registers[writeReg] <= mem_data; // write the data read from
memory to the register
                end
                else begin
                    registers[writeReg] <= ALU_result; // write the result from
ALU to the register
                end
            end
            if (hi_lo_calculate) begin // calculate hi and lo
                hi <= hi_from_ALU;
                lo <= lo_from_ALU;
            end
            if (mfhi && rd) registers[rd] <= hi; // write hi to the register
            if (mflo && rd) registers[rd] <= lo; // write lo to the register
        end
    end
```

Better user experience

- Boot effect: waterfall lights
  We implemented the flow light effect using `mips` and used the resulting coe file as the initial file for the ip core. After the chip burns the bit stream successfully, the development board

will show the special effects of the water lamp, and then enter the uart mode for subsequent operations according to the user's operation.
- Seven-Segment Digital Tube
  The results of arithmetic operations are displayed in 7 sections of digital tube for showing a clear arithmetic result.
- Enter button
  When it is necessary to enter data through the switch, we designed a enter button to confirm the completion of the input. And the address of this button value is 0xFFFFFC73.

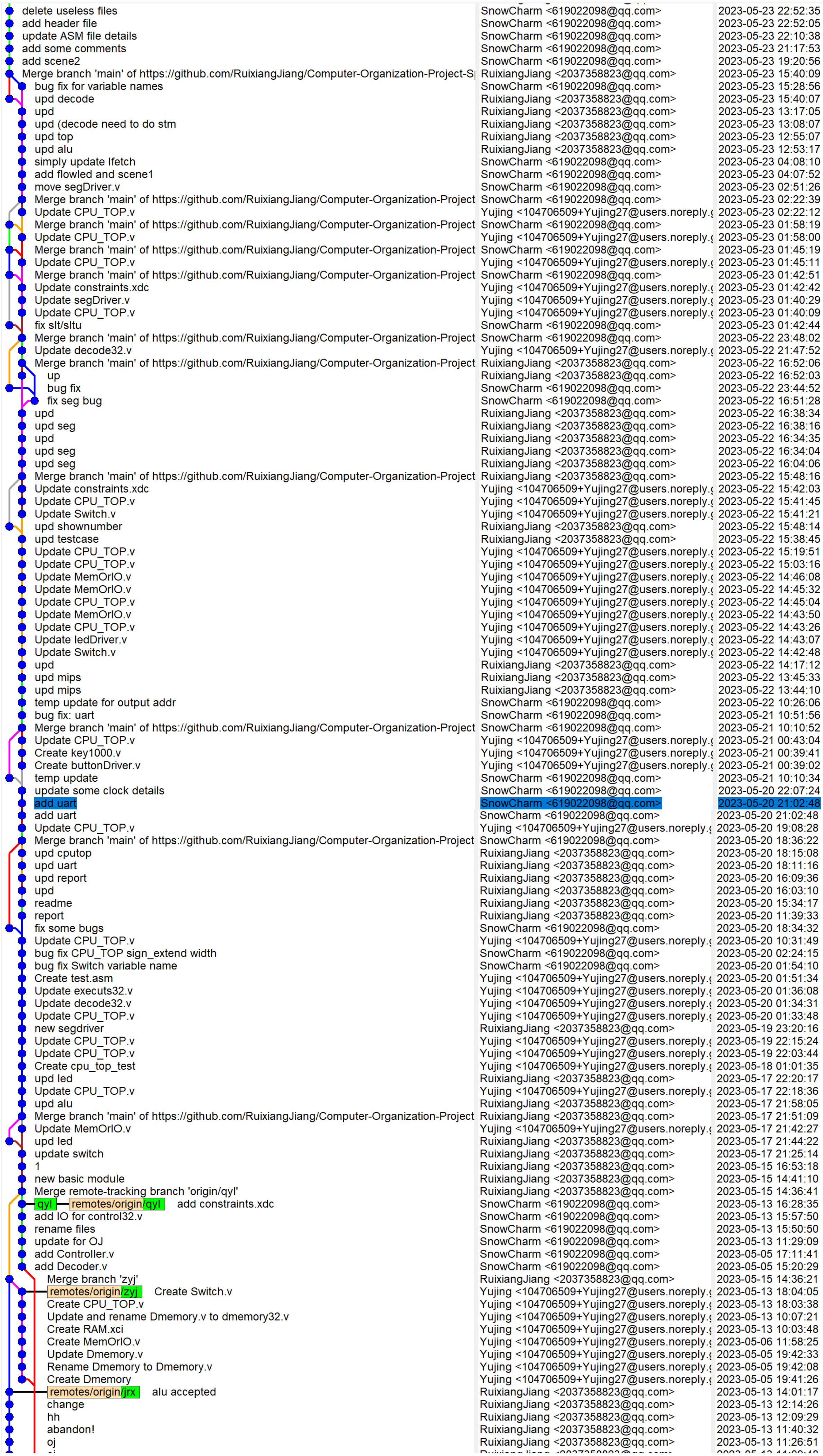
Test instructions

The test process is detailed in the project video.

Detail	Result
Show water lamp	Accepted
Test uart	Accepted
Test `mult`,`multu`	Accepted
Test `div`,`divu`,`mfhi`,`mflo`	Accepted
Show different IO(digital tube,enter button)	Accepted

Issue and Summary for both Basic and Bonus

- Use Github to merge codes written by each one, so we can synchronize the codes in an efficienct way.



delete useless files	SnowCharm <619022098@qq.com>	2023-05-23 22:52:35
add header file	SnowCharm <619022098@qq.com>	2023-05-23 22:52:05
update ASM file details	SnowCharm <619022098@qq.com>	2023-05-23 22:10:38
add some comments	SnowCharm <619022098@qq.com>	2023-05-23 21:17:53
add scene2	SnowCharm <619022098@qq.com>	2023-05-23 19:20:56
Merge branch 'main' of https://github.com/RuixiangJiang/Computer-Organization-Project-S	RuixiangJiang <2037358823@qq.com>	2023-05-23 15:40:09
bug fix for variable names	SnowCharm <619022098@qq.com>	2023-05-23 15:28:56
upd decode	RuixiangJiang <2037358823@qq.com>	2023-05-23 15:40:07
upd	RuixiangJiang <2037358823@qq.com>	2023-05-23 13:17:05
upd (decode need to do stm	RuixiangJiang <2037358823@qq.com>	2023-05-23 13:08:07
upd top	RuixiangJiang <2037358823@qq.com>	2023-05-23 12:55:07
upd alu	RuixiangJiang <2037358823@qq.com>	2023-05-23 12:53:17
simply update Ifetch	SnowCharm <619022098@qq.com>	2023-05-23 04:08:10
add flowled and scene1	SnowCharm <619022098@qq.com>	2023-05-23 04:07:52
move segDriver.v	SnowCharm <619022098@qq.com>	2023-05-23 02:51:26
Merge branch 'main' of https://github.com/RuixiangJiang/Computer-Organization-Project	SnowCharm <619022098@qq.com>	2023-05-23 02:22:39
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-23 02:22:12
Merge branch 'main' of https://github.com/RuixiangJiang/Computer-Organization-Project	SnowCharm <619022098@qq.com>	2023-05-23 01:58:19
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-23 01:58:00
Merge branch 'main' of https://github.com/RuixiangJiang/Computer-Organization-Project	SnowCharm <619022098@qq.com>	2023-05-23 01:45:19
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-23 01:45:11
Merge branch 'main' of https://github.com/RuixiangJiang/Computer-Organization-Project	SnowCharm <619022098@qq.com>	2023-05-23 01:42:51
Update constraints.xdc	Yujing <104706509+Yujing27@users.noreply.(	2023-05-23 01:42:42
Update segDriver.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-23 01:40:29
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-23 01:40:09
fix slt/sltu	SnowCharm <619022098@qq.com>	2023-05-23 01:42:44
Merge branch 'main' of https://github.com/RuixiangJiang/Computer-Organization-Project	SnowCharm <619022098@qq.com>	2023-05-22 23:48:02
Update decode32.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-22 21:47:52
Merge branch 'main' of https://github.com/RuixiangJiang/Computer-Organization-Project	RuixiangJiang <2037358823@qq.com>	2023-05-22 16:52:06
up	RuixiangJiang <2037358823@qq.com>	2023-05-22 16:52:03
bug fix	SnowCharm <619022098@qq.com>	2023-05-22 23:44:52
fix seg bug	SnowCharm <619022098@qq.com>	2023-05-22 16:51:28
upd	RuixiangJiang <2037358823@qq.com>	2023-05-22 16:38:34
upd seg	RuixiangJiang <2037358823@qq.com>	2023-05-22 16:38:16
upd	RuixiangJiang <2037358823@qq.com>	2023-05-22 16:34:35
upd seg	RuixiangJiang <2037358823@qq.com>	2023-05-22 16:34:04
upd seg	RuixiangJiang <2037358823@qq.com>	2023-05-22 16:04:06
Merge branch 'main' of https://github.com/RuixiangJiang/Computer-Organization-Project	RuixiangJiang <2037358823@qq.com>	2023-05-22 15:48:16
Update constraints.xdc	Yujing <104706509+Yujing27@users.noreply.(	2023-05-22 15:42:03
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-22 15:41:45
Update Switch.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-22 15:41:21
upd shownumber	RuixiangJiang <2037358823@qq.com>	2023-05-22 15:48:14
upd testcase	RuixiangJiang <2037358823@qq.com>	2023-05-22 15:38:45
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-22 15:19:51
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-22 15:03:16
Update MemOrIO.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-22 14:46:08
Update MemOrIO.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-22 14:45:32
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-22 14:45:04
Update MemOrIO.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-22 14:43:50
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-22 14:43:26
Update ledDriver.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-22 14:43:07
Update Switch.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-22 14:42:48
upd	RuixiangJiang <2037358823@qq.com>	2023-05-22 14:17:12
upd mips	RuixiangJiang <2037358823@qq.com>	2023-05-22 13:45:33
upd mips	RuixiangJiang <2037358823@qq.com>	2023-05-22 13:44:10
temp update for output addr	SnowCharm <619022098@qq.com>	2023-05-22 10:26:06
bug fix: uart	SnowCharm <619022098@qq.com>	2023-05-21 10:51:56
Merge branch 'main' of https://github.com/RuixiangJiang/Computer-Organization-Project	SnowCharm <619022098@qq.com>	2023-05-21 10:10:52
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-21 00:43:04
Create key1000.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-21 00:39:41
Create buttonDriver.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-21 00:39:02
temp update	SnowCharm <619022098@qq.com>	2023-05-21 10:10:34
update some clock details	SnowCharm <619022098@qq.com>	2023-05-20 22:07:24
add uart	SnowCharm <619022098@qq.com>	2023-05-20 21:02:48
add uart	SnowCharm <619022098@qq.com>	2023-05-20 21:02:48
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-20 19:08:28
Merge branch 'main' of https://github.com/RuixiangJiang/Computer-Organization-Project	SnowCharm <619022098@qq.com>	2023-05-20 18:36:22
upd cputop	RuixiangJiang <2037358823@qq.com>	2023-05-20 18:15:08
upd uart	RuixiangJiang <2037358823@qq.com>	2023-05-20 18:11:16
upd report	RuixiangJiang <2037358823@qq.com>	2023-05-20 16:09:36
upd	RuixiangJiang <2037358823@qq.com>	2023-05-20 16:03:10
readme	RuixiangJiang <2037358823@qq.com>	2023-05-20 15:34:17
report	RuixiangJiang <2037358823@qq.com>	2023-05-20 11:39:33
fix some bugs	SnowCharm <619022098@qq.com>	2023-05-20 18:34:32
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-20 10:31:49
bug fix CPU_TOP sign_extend width	SnowCharm <619022098@qq.com>	2023-05-20 02:24:15
bug fix Switch variable name	SnowCharm <619022098@qq.com>	2023-05-20 01:54:10
Create test.asm	Yujing <104706509+Yujing27@users.noreply.(	2023-05-20 01:51:34
Update executs32.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-20 01:36:08
Update decode32.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-20 01:34:31
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-20 01:33:48
new segdriver	RuixiangJiang <2037358823@qq.com>	2023-05-19 23:20:16
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-19 22:15:24
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-19 22:03:44
Create cpu_top_test	Yujing <104706509+Yujing27@users.noreply.(	2023-05-18 01:01:35
upd led	RuixiangJiang <2037358823@qq.com>	2023-05-17 22:20:17
Update CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-17 22:18:36
upd alu	RuixiangJiang <2037358823@qq.com>	2023-05-17 21:58:05
Merge branch 'main' of https://github.com/RuixiangJiang/Computer-Organization-Project	RuixiangJiang <2037358823@qq.com>	2023-05-17 21:51:09
Update MemOrIO.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-17 21:42:27
upd led	RuixiangJiang <2037358823@qq.com>	2023-05-17 21:44:22
update switch	RuixiangJiang <2037358823@qq.com>	2023-05-17 21:25:14
1	RuixiangJiang <2037358823@qq.com>	2023-05-15 16:53:18
new basic module	RuixiangJiang <2037358823@qq.com>	2023-05-15 14:41:10
Merge remote-tracking branch 'origin/qyl'	RuixiangJiang <2037358823@qq.com>	2023-05-15 14:36:41
qyl remotes/origin/qyl add constraints.xdc	SnowCharm <619022098@qq.com>	2023-05-13 16:28:35
add IO for control32.v	SnowCharm <619022098@qq.com>	2023-05-13 15:57:50
rename files	SnowCharm <619022098@qq.com>	2023-05-13 15:50:50
update for OJ	SnowCharm <619022098@qq.com>	2023-05-13 11:29:09
add Controller.v	SnowCharm <619022098@qq.com>	2023-05-05 17:11:41
add Decoder.v	SnowCharm <619022098@qq.com>	2023-05-05 15:20:29
Merge branch 'zyj'	RuixiangJiang <2037358823@qq.com>	2023-05-15 14:36:21
remotes/origin/zyj Create Switch.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-13 18:04:05
Create CPU_TOP.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-13 18:03:38
Update and rename Dmemory.v to dmemory32.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-13 10:07:21
Create RAM.xci	Yujing <104706509+Yujing27@users.noreply.(	2023-05-13 10:03:48
Create MemOrIO.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-06 11:58:25
Update Dmemory.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-05 19:42:33
Rename Dmemory to Dmemory.v	Yujing <104706509+Yujing27@users.noreply.(	2023-05-05 19:42:08
Create Dmemory	Yujing <104706509+Yujing27@users.noreply.(	2023-05-05 19:41:26
remotes/origin/jrx alu accepted	RuixiangJiang <2037358823@qq.com>	2023-05-13 14:01:17
change	RuixiangJiang <2037358823@qq.com>	2023-05-13 12:14:26
hh	RuixiangJiang <2037358823@qq.com>	2023-05-13 12:09:29
abandon!	RuixiangJiang <2037358823@qq.com>	2023-05-13 11:40:32
oj	RuixiangJiang <2037358823@qq.com>	2023-05-13 11:26:51

- Code Standards and Naming
  The variable naming in the submodule code is not standardized and unified enough, which caused us to have problems with port connection errors and incorrect port widths when connecting top-level modules and submodules during development, resulting in a lot of time debugging.
- Sequential Logic
  Make sure that each operation in sequential logic modules is executed on the correct clock edge, especially when using IP cores.
- Inadequate clock cycle
  When implementing the extension instruction `div`, we encountered the difficulty that the result was not correct. Finally, we found that the clock cycle was not appropriate, so we changed the original 23MHz clock cycle to 10MHz, and finally implemented this instruction.
- New features need to be added with care
  During the development of Uart module, some inappropriate modifications caused a bug in our J-type instruction. We found this bug and modified it. We also understood that the implementation of each new function should be made sure that it does not affect the previous old function.