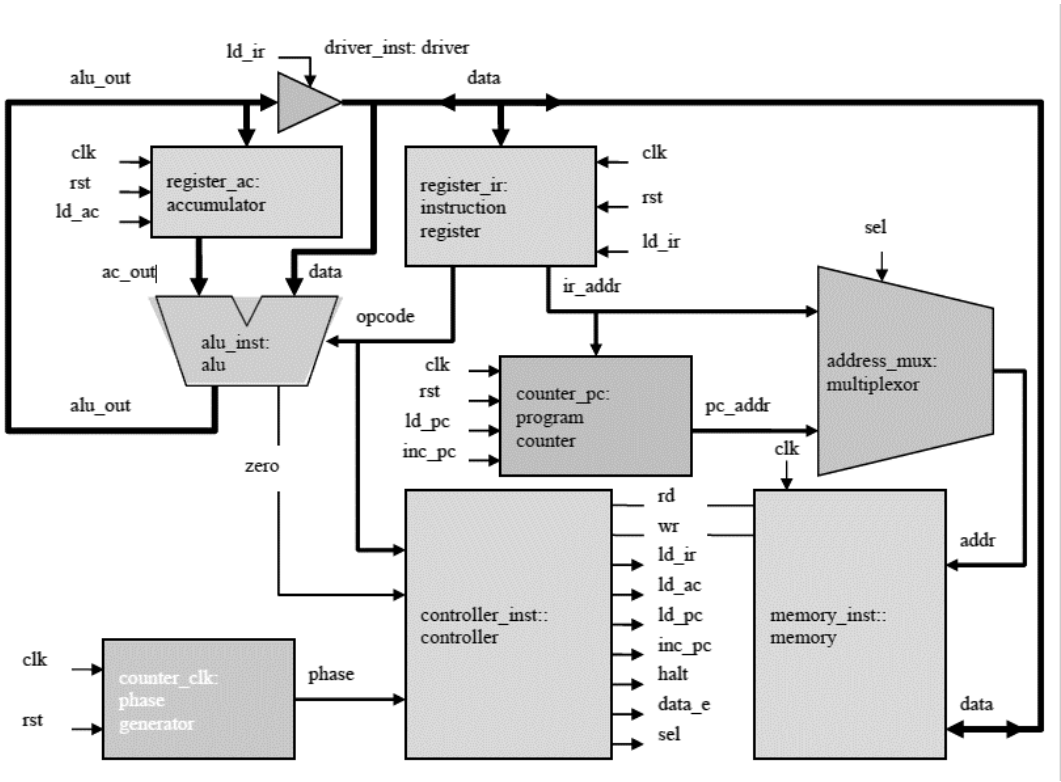
VeriRISC CPU

The VeriRISC model is a very-reduced-instruction-set processor coded in the Verilog HDL. Its instruction consists of a three-bit operation code and a five-bit operand. That restricts its instruction set to eight instructions and its address space to 32 locations. To simplify visualization and debug, it implements the fetch-and-execute cycle in eight phases.

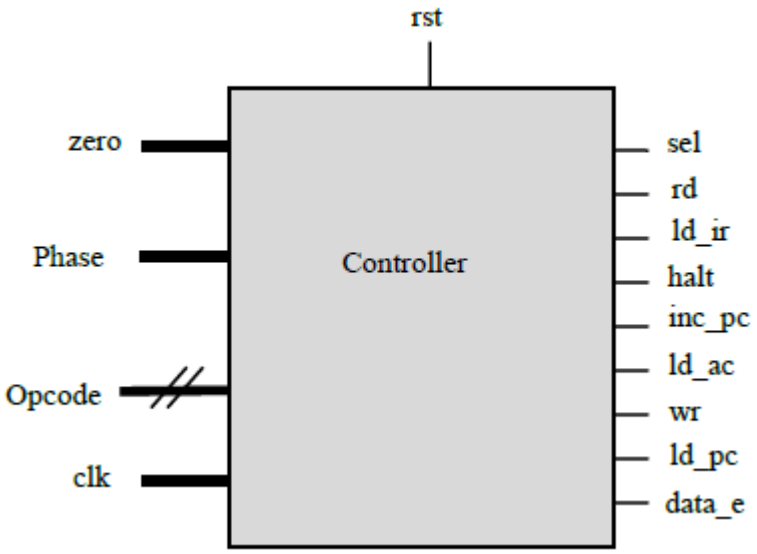


Block diagram

Note that you should use parametrized values

1-Controller

The main sub block is the controller as it generates all control signals for the VeriRISC CPU. The operation code, fetch-and execute phase, and whether the accumulator is zero determine the control signal levels.



Specifications

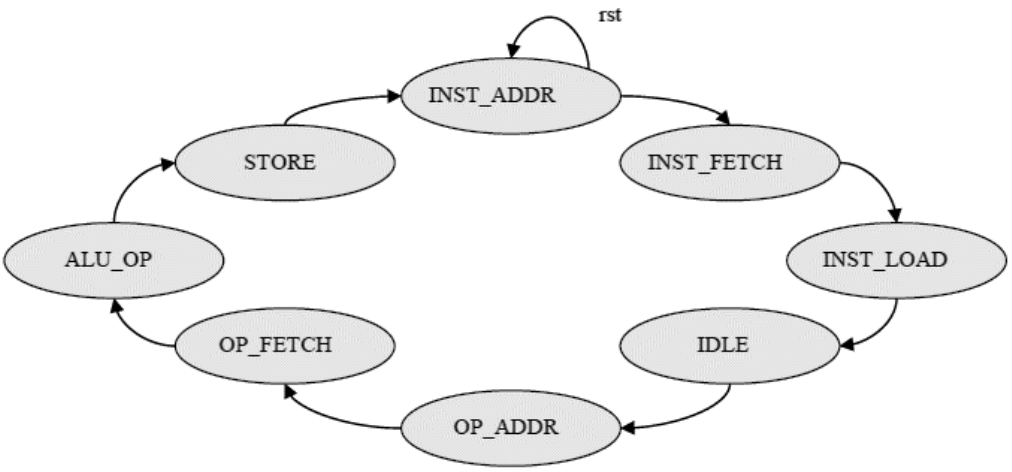
- The controller is clocked on the rising edge of clk.
- rst is synchronous and active high.
- zero is an input which is 1 when the CPU accumulator is zero and 0 otherwise.
- opcode is a 3-bit input for CPU operation, as shown in the following table.

Opcode/ Instruction	Opcode Encoding	Operation	Output
HLT	000	PASS A	in_a => alu_out
SKZ	001	PASS A	in_a => alu_out
ADD	010	ADD	in_a + in_b => alu_out
AND	011	AND	in_a & in_b => alu_out
XOR	100	XOR	in_a ^ in_b => alu_out
LDA	101	PASS B	in_b => alu_out
STO	110	PASS A	in_a => alu_out
JMP	111	PASS A	in_a => alu_out

There are 7 single-bit outputs, as shown in this table.

Output	Function
sel	select
rd	memory read
ld_ir	load instruction register
halt	halt
inc_pc	increment program counter
ld_ac	load accumulator
ld_pc	load program counter
wr	memory write
data_e	data enable

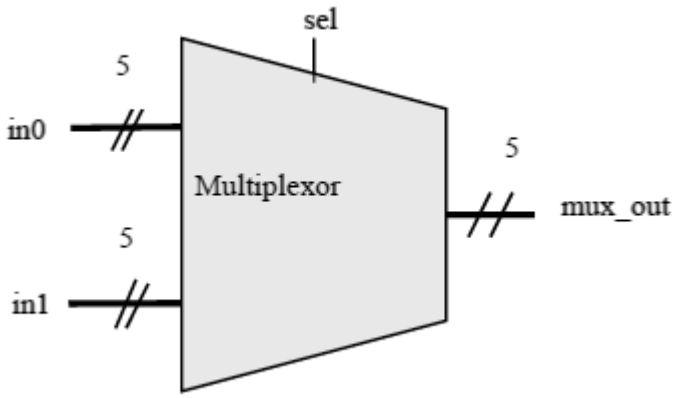
The controller has a single-bit phase input with a total of 8 phases processed. Phase transitions are unconditional, i.e., the controller passes through the same 8-phase sequence, from INST_ADDR to STORE, every 8 clk cycles. The reset state is INST_ADDR.



The controller outputs will be decoded w.r.t phase and opcode, as shown in this table.

Outputs	Phase						Notes		
	INST_ ADDR	INST_ FETCH	INST_ LOAD	IDLE	OP_ ADDR	OP_ FETCH	ALU_ OP	STORE	
sel	1	1	1	1	0	0	0	0	ALU_OP = 1 if opcode is ADD, AND, XOR or LDA
rd	0	1	1	1	0	ALUOP	ALUOP	ALUOP	
ld_ir	0	0	1	1	0	0	0	0	
halt	0	0	0	0	HALT	0	0	0	
inc_pc	0	0	0	0	1	0	SKZ && zero	0	
ld_ac	0	0	0	0	0	0	0	ALUOP	
ld_pc	0	0	0	0	0	0	JMP	JMP	
wr	0	0	0	0	0	0	0	STO	
data_e	0	0	0	0	0	0	STO	STO	

2-Multiplexer

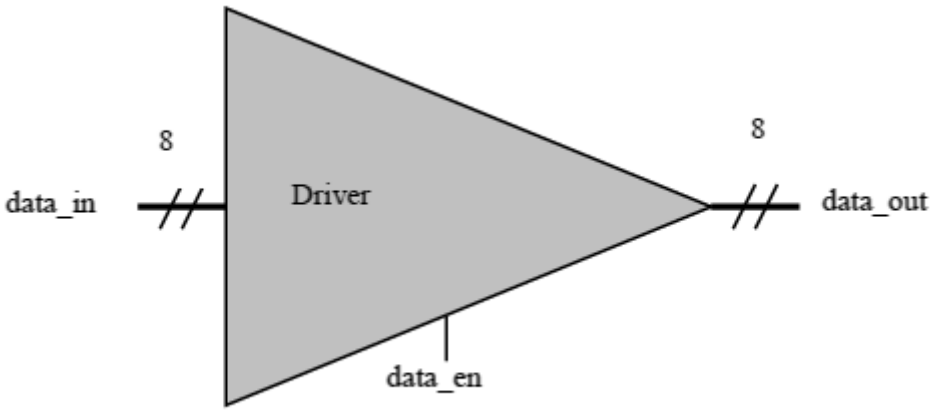


Specifications

- The address multiplexor selects between the instruction address during the instruction
- fetch phase and the operand address during the instruction execution phase.
- MUX width is parameterized with the default value of 5.
- If sel is 1'b0, input in0 is passed to the output mux_out.
- If sel is 1'b1, input in1 is passed to the output mux_out.

3-Driver

The driver output is equal to the input value when enabled (*data_en* is true) and is high-impedance when not enabled (*data_en* is false).

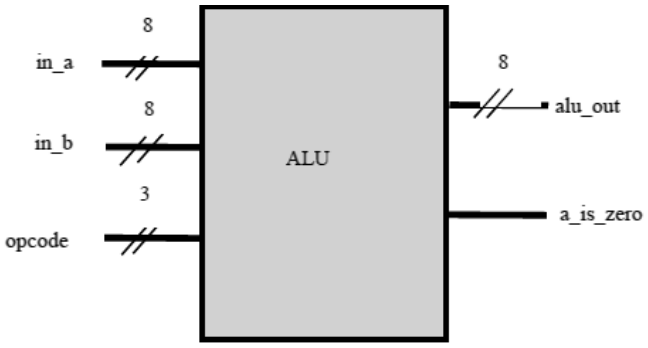


Specifications

- data_in and data_out are both parameterized widths of 8-bit.
- If data_en is high, the input data_in is passed to the output data_out.
- Otherwise, data_out is high impedance.

4-ALU

ALU performs arithmetic operations on numbers depending upon the operation encoded in the instruction. This ALU will perform 8 operations on the 8-bit inputs (see table in the S*pecification*) and generate an 8-bit output and single-bit output.

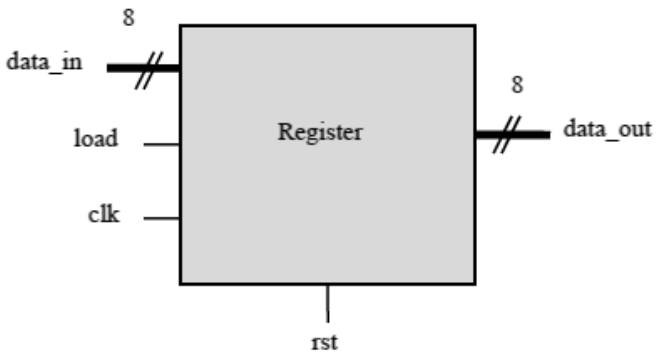


Specifications

- in_a, in_b, and alu_out are all 8-bit long. The opcode is a 3-bit value for the CPU operation code, as defined in the following table.
- a_is_zero is a single bit asynchronous output with a value of 1 when in_a equals 0. Otherwise, a_is_zero is 0.
- The output alu_out value will depend on the opcode value as per the following table.
- To select which of the 8 operations to perform, you will use opcode as the selection lines.
- The following table states the opcode/instruction, opcode encoding, operation, and output.

Opcode/ Instruction	Opcode Encoding	Operation	Output
HLT	000	PASS A	in_a => alu_out
SKZ	001	PASS A	in_a => alu_out
ADD	010	ADD	in_a + in_b => alu_out
AND	011	AND	in_a & in_b => alu_out
XOR	100	XOR	in_a ^ in_b => alu_out
LDA	101	PASS B	in_b => alu_out
STO	110	PASS A	in_a => alu_out
JMP	111	PASS A	in_a => alu_out

5-Register

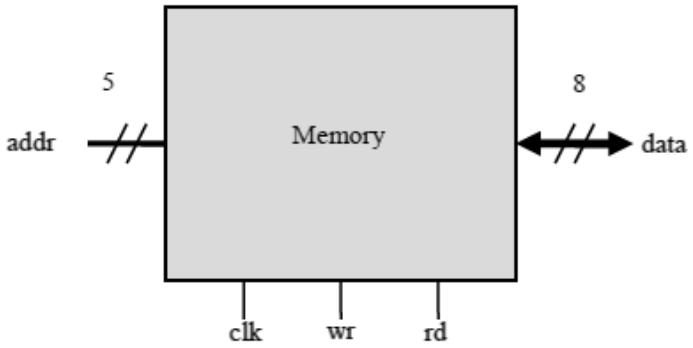


Specifications

- data_in and data_out are both 8-bit signals.
- rst is synchronous and active high.
- The register is clocked on the rising edge of clk.
- If load is high, the input data is passed to the output data_out.
- Otherwise, the current value of data_out is retained in the register.

6- Memory

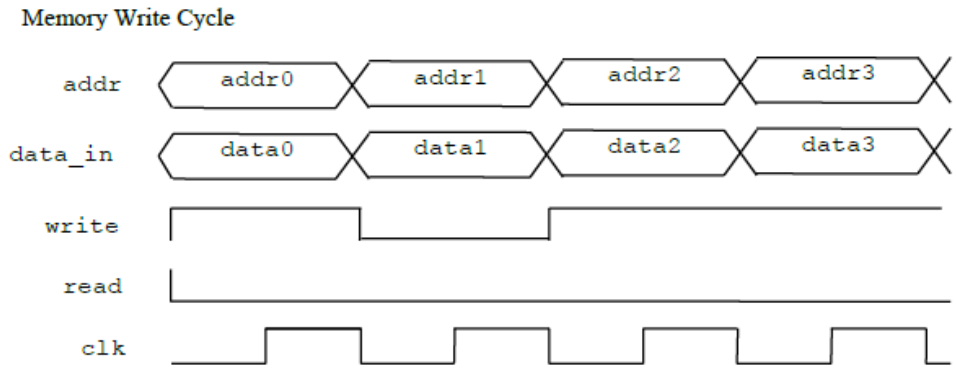
The VeriRISC CPU uses the same memory for instructions and data. The memory has a single bidirectional data port and separate write and read control inputs. It cannot perform simultaneous write and read operations.



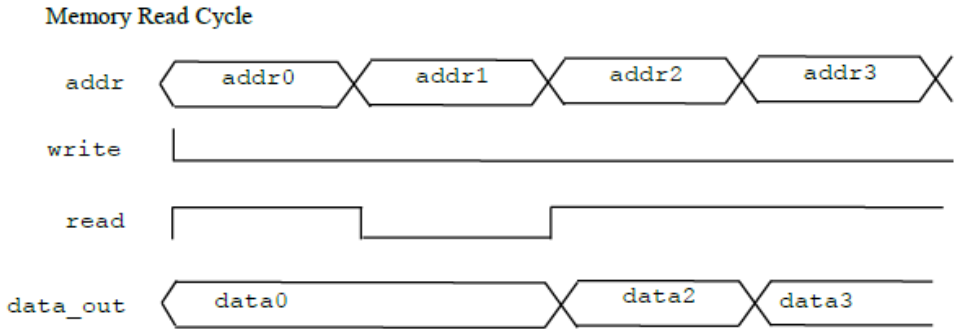
Specifications

- addr is parameterized to 5 and data is parameterized to 8.
- wr (write) and rd (read) are single-bit input signals.
- The memory is clocked on the rising edge of clk.
- Analyze the memory read and write operation timing diagram, as shown in the following graphic.

- **Memory write:** `data_in` is written to memory[addr] on the positive edge of `clk` when `wr` (write) =1.

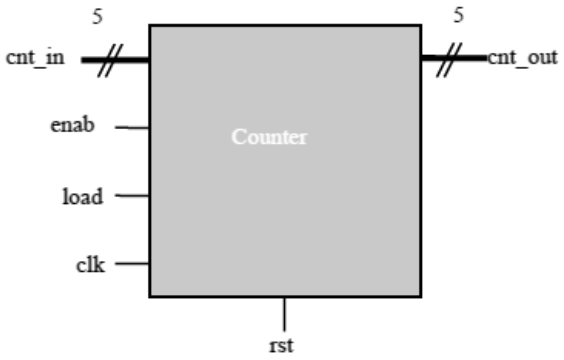


- **Memory read**: `data_out` is assigned from memory[addr] when `rd` (read) =1.



7-counter

The VeriRISC CPU contains a program counter and a phase counter. One generic counter definition can serve both purposes.



Specifications

- The counter is clocked on the rising edge of clk.
- rst is active high.
- cnt_in and cnt_out are both 5-bit signals.
- If rst is high, output will become *zero*.
- If load is high, the counter is loaded from the input cnt_in.
- Otherwise, if enab is high, cnt_out is incremented, and cnt_out is unchanged.

At the *behavioral* level of abstraction, you code behavior with no regard for an actual hardware implementation, so you can utilize any Verilog construct. The behavioral level of abstraction is useful for exploring design architecture and especially useful for developing test benches. As both uses are beyond the scope of this training module, it only briefly introduces testbench concepts.

Specifications

The CPU architecture is as follows:

- The Program Counter (`counter`) provides the program address.
- The MUX (`mux`) selects between the program address or the address field of the instruction.
- The Memory (`memory`) accepts data and provides instructions and data.
- The Instruction Register (`register`) accepts instructions from the memory.
- The Accumulator Register (`register`) accepts data from the ALU.
- The ALU (`alu`) accepts memory and accumulator data, and the opcode field of the instruction, and provides new data to the accumulator and memory.

If all components of the CPU are working properly, it will:
1. Fetch an instruction from the memory.
2. Decode the instruction.
3. Fetch a data operand from memory if required by the instruction.
4. Execute the instruction, processing mathematical operations, if required.
5. Store results back into either the memory or the accumulator. This process is repeated for every instruction in a program until an `HLT` instruction is found.