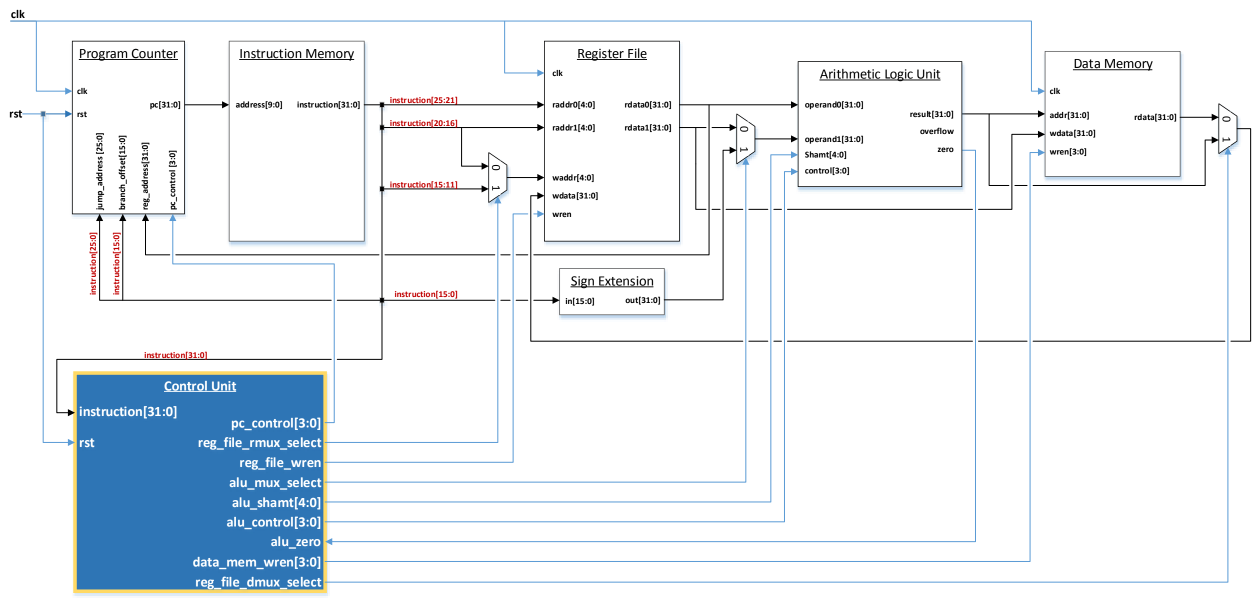

Figure 1) Block diagram of the CPU. You need to create the Control Unit and complete the implementation of the CPU by instantiating all components and wiring them appropriately. Use this diagram as a guide.

Overview

In this assignment you will complete a Verilog description of a MIPS CPU. You will create the Control Unit (CU) and integrate it with components that have been provided. The CPU performs all operations in a single clock cycle. In other words instruction fetch, instruction decode, execute, write back, and program counter update occurs within a single clock cycle.

Refer to Figure 1. Upon assertion of the "rst" signal the program counter is asynchronously cleared to "0". The program counter is presented as an address to the instruction memory. The instruction memory subsequently outputs the 32-bit instruction. The 32-bit instruction is presented to several datapath components and the Control Unit. The Control Unit generates the signals to (1) appropriately control the flow of data throughout the datapath, (2) enable/disable register file update, (3) enable disable data memory write, and (4) control the PC update mode. At the rising edge of "clk" the program counter, register file, and data memory are updated as determined by the Control Unit.

Modules

Most of the required modules have been implemented and provided. You are responsible for implementing the Control Unit and integrating it with the other components to finish the CPU.

Program Counter (PROVIDED) – program_counter.v

The program counter maintains the current 32-bit instruction address and outputs it on the "pc" bus. The program counter is a synchronous unit that is updated at the rising edge of the clock signal "clk". The program counter is asynchronously cleared (zeroed) whenever the active-high reset signal "rst" is asserted. The value of the program counter is updated based on the value of the 4-bit "pc_control" signal. See Table 1 for description of the update behavior. If the program counter is to be updated by a Jump

Immediate instruction then the 26-bit “jump_address” bus contains the value that should be used to compute the new pc value. For Jump Register instructions the 32-bit “reg_address” bus contains the new PC value. If the program counter is to be updated by a Branch instruction then the 16-bit “branch_offset” bus contains the value that should be used to compute the new pc value.

Table 1) pc_control[3:0] encoding and description

pc_control	Updated PC Value	Description
4’b0000	PC = PC + 4	PC is updated to the next sequential instruction address.
4’b0001	PC = {PC[31:28] , (jump_address*4)[27:0]}	When the CPU executes an unconditional Jump instruction the new 32-bit PC is the concatenation of the upper 4-bits of PC and the 28-bit result of the jump address multiplied by 4. See definition of the Jump instruction (opcode 2).
4’b0010	PC = register	PC is updated with the value contained in the register specified in the instruction. “reg_address” holds the value of the specified register.
4’b0011	PC =(PC + 4) + (SignExtend(branch_offset)*4)	PC is updated with the sum of PC+4 and the branch offset multiplied by 4. 16-bit to 32-bit sign extension of the branch offset is necessary.
4’b0100-4’b1111	Undefined	These control values are not currently specified and the behavior of the PC update is left to the implementer (you).

Table 2 Program Counter signal definitions

Signal Name	Direction	Description
clk	Input	Positive edge active clock. All PC updates are synchronous to this clock.
rst	Input	Asynchronous active high reset signal.
pc_control[3:0]	Input	Specifies the update behavior of the PC. See Table 1 for encodings.
jump_address[25:0]	Input	Specifies the value to be used when computing the new PC for Jump instructions
branch_offset[15:0]	Input	Specifies the value to be used when computing the new PC for Branch instructions. Note that this signal must be properly sign extended before use.
reg_address[31:0]	Input	Specifies the value to be used as the new PC for Jump Register instructions.
pc[31:0]	Output	The current value of the program counter.

Instruction Memory (PROVIDED) – instruction_memory.v

The instruction memory is a Read Only Memory (ROM) that holds the program that your CPU will execute. Table 3 lists the Instruction Memory signals.

Table 3 Instruction Memory signal definitions

Signal Name	Direction	Description
address[31:0]	Input	32-bit address.
instruction[31:0]	Output	The 32-bit mips instruction located at the specified address.

Register File (PROVIDED) – register_file.v

The Register File, RF, contains the 32 32-bit MIPS registers. The register file has two read ports and a single write port. The register file is read asynchronously and written synchronously at the rising edge of the clock. The register file supports simultaneous read and writes.

Table 4 Register File signal definitions

Signal Name	Direction	Description
clk	Input	Positive edge active clock. All Register File writes are synchronous to this clock.
raddr0[4:0]	Input	Specifies the port0 register read address
raddr1[4:0]	Input	Specifies the port1 register read address
waddr[4:0]	Input	Specifies the register write address
wdata[31:0]	Input	Specifies the value to be written into the register specified by “waddr”
wren	Input	Enables/Disables the writing of the register file. When asserted, the register specified by “waddr” is updated with the value on “wdata” at the rising edge of the clock, otherwise the register file contents are not modified
rdata0[31:0]	Output	Port0 read data
rdata1[31:0]	Output	Port1 read data

Arithmetic Logic Unit (PROVIDED) – alu.v

The ALU supports **Add, Subtract, AND, OR, XOR, NOR, Set on Less than, Shift Left, Shift Right,** and **Shift Right Arithmetic**. The specific operation that the ALU performs is specified by the 4-bit “alu_control” bus. Table 5 specifies the bit encoding for each operation.

Table 5 ALU Function codes

alu_control	ALU Function
0000	AND
0001	OR
0010	XOR
0011	NOR
0100	ADD
0101	SIGNED ADD
0110	SUBTRACT
0111	SIGNED SUBTRACT
1000	SET ON LESS THAN
1001	SHIFT LEFT
1010	SHIFT RIGHT
1011	SHIFT RIGHT ARITHMETIC

Table 6 ALU signal definitions

Signal Name	Direction	Description
control[3:0]	Input	Specifies the operation that the ALU should perform. See Table 5
shamt[4:0]	Input	Specifies the shift amount for all shift operations.
operand0[31:0]	Input	First operand
operand1[31:0]	Input	Second operand.
result[31:0]	Output	The result of the operation.
overflow	Output	Signifies that an overflow occurred during the computation. Only valid during signed version of operations (add, addi, sub) and is de-asserted otherwise.
zero	Output	Signifies that the current ALU operation resulted in a zero value. Useful for branching.

Data Memory (PROVIDED) – data_memory.v

The data memory is a RAM that provides a store for the CPU to load from and store to. The Data Memory has one read port and one write port. Reads are asynchronous while writes are synchronous to the rising edge of the "clk" signal. The Word width of the data memory is 32-bits to match the datapath width. The data memory contains 65536 entries. The data memory has a 4-bit write enable bus (wren[3:0]) to allow writes from specific bytes of the 32-bit "wdata". The write enable bus is useful only when performing a write to the data memory and is meaningless for reads (reads return all 4 bytes of a word). For example, wren[3:0] == 4'b0001 indicates that only the lower byte of the "wdata" should be used to update the location specified by the address. There are four valid values of wren: 4'b0000 (No write), 4'b0001 (byte 0), 4'b0011 (bytes 0-1), and 4'b1111 (bytes 0-3).

Table 7 Data Memory Signals

Signal Name	Direction	Description
clk	Input	Positive edge active clock. All Data Memory writes are synchronous to this clock.
addr[31:0]	Input	32-bit address.
wdata[31:0]	Input	The 32-bit data to be written at the specified address.
wren[3:0]	Input	4-bit Write Enable that specifies how memory is updated. See Table 8 Explanation of Data Memory Wren encodingsTable 8
rdata[31:0]	Output	The 32-bit data located at the specified address.

Table 8 Explanation of Data Memory Wren encodings

wren[3:0]	Data Memory Write
0000	No Write
0001	The location is updated with the lower byte of the wdata bus. (wdata[7:0])
0011	The location is updated with the lower half word of the wdata bus. (wdata[15:0])
1111	The location is updated with the entire word present on the wdata bus. (wdata[31:0])

Control Unit (Not-Implemented) – control_unit.v

The Control Unit takes the 32-bit instruction and generates the appropriate signal values to control the flow of data throughout the datapath. See Table 9 for the list of instructions that you are expected to support. See Table 10 for a description of each of the control unit's input/output signals.

Table 9 Control Unit supported instructions

Instruction Type	Supported Instructions
Arithmetic	Add, Add Unsigned, Subtract, Subtract Unsigned, Add Immediate, Add Immediate Unsigned
Logical	AND, AND Immediate, OR, OR Immediate, XOR, NOR, set on less than, set on less than immediate
Data Transfer	Load Word, Load Upper Immediate, Store Word, Store Half Word, Store Byte
Shift	Shift Left Logical, Shift Right Logical, Shift Right Arithmetic
Conditional Branch	Branch on equal, Branch on not equal
Unconditional Jump	Jump, Jump Register

Table 10 Control Unit signal definitions

Signal Name	Direction	Description
rst	Input	Active high reset. The control unit should not allow updates to data memory or the register file when the "rst" signal is asserted.
instruction[31:0]	Input	32-bit instruction from instruction memory
data_mem_wren[3:0]	Output	4-bit write enable signal that controls writes to the data memory at byte granularity. See Data Memory description.
reg_file_wren	Output	1-bit write enable signal that controls whether the register file is written at the upcoming rising clock edge.
reg_file_dmux_select	Output	Controls the multiplexer that selects from which source the register file should get its write data: either the ALU result or Data Memory output.
reg_file_rmux_select	Output	Controls the multiplexer that selects from which source the register file should get its write address: either $rd for R-type instructions or $rt for I-type instructions.
alu_mux_select	Output	Controls the multiplexer that selects from which source the ALU gets its second operand: either the sign extended immediate value or the read data from the register file.
alu_control[3:0]	Output	Specifies the operation that the ALU should perform based on the decoded instruction. See Table 5.
alu_shamt[4:0]	Output	Specifies the amount of shift that the ALU should perform.
alu_zero	Input	Signal that indicates that current ALU operation resulted in a zero value. Useful for branching.
pc_control[3:0]	Output	Specifies the manner in which the PC should be updated based on the decoded instruction. See Table 1.

CPU (Not-Implemented) – cpu.v

The CPU module represents the top-level of the design. All components should be instantiated in the CPU module and wired appropriately. Use Figure 1 as a guide of how components should be instantiated and how they should be connected.

Support Modules (PROVIDED) – sign_extension.v & mux_2to1.v

There are two supporting modules that you will need when creating the CPU module. These modules include a parameterized Sign Extension module and a parameterized 2-to-1 Multiplexer. The behavior of these components should be obvious. Review the implementation of these modules to determine the purpose and use of each parameter.

Implementation Details

All module files have been provided with input and output ports already defined. You may not add ports, remove ports, or change the width of ports. However you are allowed to change the type of a port (e.g from wire to reg) or add additional signals (wire or reg) internal to CPU as necessary. Your tasks are as follows: (1) Create the Control Unit from scratch. (2) Instantiate all necessary modules inside of the empty cpu.v module. (3) Connect the components, inputs, and outputs as illustrated in Figure 1. (4) Simulate and verify the operation of CPU.

A testbench has been provided. The Instruction Memory is preloaded with the machine code of a sample program that can be found at "\simulation\program.mips". The corresponding assembly program is located at "\simulation\program.asm". The testbench generates the clock and reset signals but does not perform automatic checking of the output. You should validate your CPU by manually checking the simulated CPU behavior with the behavior expected of the assembly code. The easiest way to do this is to open the provided assembly code in MARS and step-wise verify the hardware simulation with the software simulation. Note that by default MARS puts the text at 0x00400000 and data at 0x10010000. You need to change this to work with the instruction and data memories provided. To make this change go to the Settings->Memory Configuration menu in MARS. Set the Configuration to "Compact, Text at Address 0".

To execute the simulation do the following:

1. Start ModelSim simulator
2. At the ModelSim command prompt, change the "Current Directory" to the "simulation" folder which contains the script file **"startsim.do"**: cd <path to folder>
3. Compile and start the simulation by typing the following at the prompt: do startsim.do
4. Add the necessary signals into the waveform viewer
5. Run the simulation

Deliverables

Please submit only the Verilog files that you create or modify. At a minimum these files should include cpu.v and control_unit.v. DO NOT zip these files when submitting on Angel: upload each Verilog file individually to the Angel dropbox.