Second Partial Project
Delivery 1/3
MIPS R-Type datapath

1 Objectives

To implement the datapath of a custom Microprocessor without Interlocked Pipeline Stage (MIPS) R-Type instruction in SystemVerilog.

2 Second partial grade weight

This delivery constitutes 35% of your second partial final grade.

3 Deadline

23:59 hours on Wednesday October 7th 2020

4 Teamwork policy

This is a group assignment.

5 Pre-requisites

It is assumed that you are familiar with working with ModelSim and Quartus. If you require assistance, you can refer to the first assignment tutorial. It is assumed that you have completed previous assignments for modelling an Arithmetic and Logic Unit (ALU), a Register File (RF), a sign-extender, various Multiplexers (MUXes), a Random Access Memory (RAM) and a Read Only Memory (ROM) modules in SystemVerilog.

6 Background

In R-Type instructions, data stored in registers is used in order to perform arithmetic or logical operations, with the result of the operation being stored back into registers. The basic instruction encoding of R-Type instructions is presented in Figure 1.

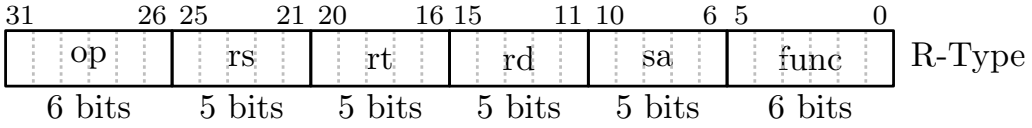

Figure 1: MIPS R-Type instruction encoding.

In R-Type instructions, `op` field has a constant value of `6'b000000`. The actual operation to be performed in R-Type instructions is given by the `func` field. By contrast, for I-Type and J-Type instructions, `op` presents different values and field `func` is not present.

7 Design specifications

The following sections provide an overview of the design specifications for this assignment.

7.1 General specifications

Your custom MIPS design must comply with the following design specifications.

- Instruction set based (but modified) on a standard 16-bit MIPS Microprocessor (μP).
- Single-cycle design.
- Data width is 16-bits.
- Instruction encoding (instruction width) is 32-bits long.
- RF contains 32 registers.
  - `r0` **must** always be 0, *i.e.*, `r0` is not writeable.
  - `r31` is return address.

7.2 R-Type datapath

Figure 2 shows the instruction encoding and the Microarchitecture (μA) diagram required in order to perform an arithmetic/logic custom MIPS R-Type instruction.

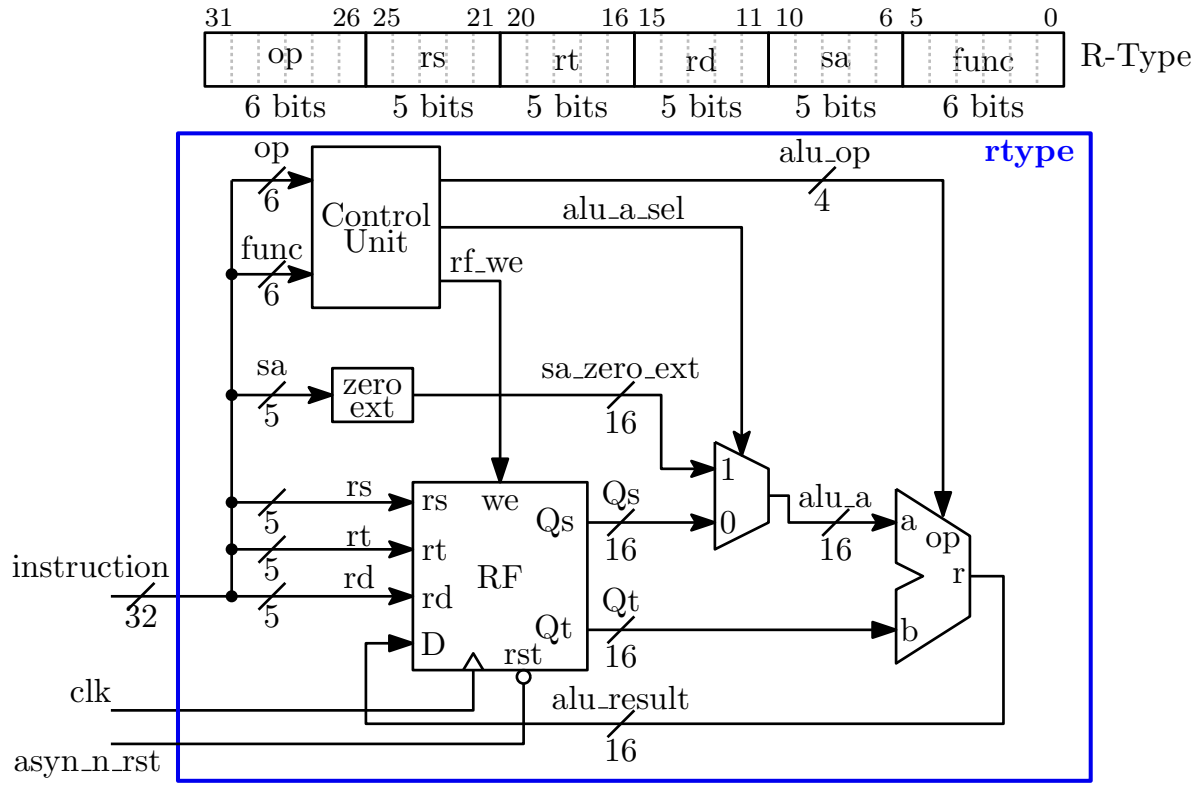

Figure 2: MIPS R-Type arithmetic/logic instruction partial datapath.

Here, the blue box in Figure 2 represents the top-level module, named `rtype`. Table 1 specifies the top-level ports required for this assignment.

Table 1: Top-level ports.

Port name	Direction	Width	Description
`clk`	`input`	1	Clock signal
`asyn_n_rst`	`input`	1	Asynchronous active-low reset
`instruction`	`input`	32	Encoded instruction to be performed

This top-level module consists of only three input ports, namely `clk`, `asyn_n_rst` and `instruction`, which correspond to the clock signal, an asynchronous active-low reset and the R-Type instruction to be performed, respectively. Note that there are no output ports in this design since the result provided by the ALU must be stored back into the RF.

7.3 List of R-Type instructions

Your design must be able to perform the instructions specified in Table 2.

Table 2: R-Type instructions.

Instruction	Syntax	Meaning
ADD	ADD rd, rs, rt	rd ← rs + rt
SUB	SUB rd, rs, rt	rd ← rs - rt
NAND	NAND rd, rs, rt	rd ← ∼(rs & rt)
NOR	NOR rd, rs, rt	rd ← ∼(rs | rt)
XNOR	XNOR rd, rs, rt	rd ← ∼(rs ^ rt)
AND	AND rd, rs, rt	rd ← rs & rt
OR	OR rd, rs, rt	rd ← rs | rt
XOR	XOR rd, rs, rt	rd ← rs ^ rt
SLL	SLL rd, rs, sa	rd ← rt << sa
SRL	SLL rd, rs, sa	rd ← rt >> sa
SLA	SLL rd, rs, sa	rd ← rt <<< sa
SRA	SLL rd, rs, sa	rd ← rt >>> sa

7.4 Description of each submodule

The following sections provide a brief description about each of the submodules presented in Figure 2.

Register File

The RF of Figure 2 stores the contents of the 32 available registers. As indicated in Section 7.1, `r0` must contain a constant value of 0. This implies that `r0` must not be writeable. As a result of this, your RF design must be updated in order to prevent write operations to be performed on `r0`. All other registers, including `r31` which will be used in J-Type instructions, may be written to. Similarly, you may have to update your `rf` module in order to comply with the following input and output ports. The inputs to this module are two source (read) registers `rs` and `rt`, a destination (write) register `rd`, input data `alu_result` provided by the ALU, a control signal `rf_we` provided by the control unit for write operations and a `clk` and an asynchronous active-low reset `asyn_n_rst`. This module provides at each clock cycle outputs `Qs` and `Qt`, which correspond to data stored in registers `rs` and `rt`, respectively. These outputs correspond to operands required by the ALU.

ALU

The ALU of Figure 2 performs the arithmetic and logical operations specified by the `instruction`. More specifically, signal `alu_op` provided by the control unit determines which operation the ALU must perform. Input `a` of the ALU is connected to signal `alu_a`, which corresponds to either output `Qs` from the RF or a zero-extended version of the field `sa` from `instruction`. This is due to the fact that field `sa` is used to determine the shift amount in shift-type operations in the ALU. However, since `sa` is only 5-bits, it must first be zero-extended to 16 bits. The selection of `alu_a` is determined by the MUX of Figure 2 and its control signal `alu_a_sel` provided by the control unit.

Control Unit

The control unit of Figure 2 commands other submodules in the design. This is performed by using several control signals, for example, enable signals. The control unit of the R-Type datapath instructs the RF to write its coming data `alu_result` from the ALU by means of the `rf_we` signal. It also instructs the MUX to provide to operand `a` of the ALU either the zero-extended version of `sa` or the output `Qs` from the RF. This is achieved by means of `alu_a_sel` signal. Finally, the control unit instructs the ALU which of the 15 operations it must perform by means of the signal `alu_op`. These control signal are asserted according the input signals `op` and `func`. More specifically, since R-Type instructions are defined with `op = 6'b000000`, the actual ALU operation will be determined by the value of `func`. A suggested mapping between `func` and `alu_op` is provided in Table 3.

Table 3: Suggested mapping between `func` and `alu_op`.

`func`	ALU operation
`6'b100000`	`ADD`
`6'b100010`	`SUB`
`6'b110100`	`NAND`
`6'b110101`	`NOR`
`6'b110110`	`XNOR`
`6'b100100`	`AND`
`6'b100101`	`OR`
`6'b100110`	`XOR`
`6'b000000`	`SLL`
`6'b000010`	`SRL`
`6'b000001`	`SLA`
`6'b000011`	`SRA`

Note that your ALU design supports operations that are not listed in Table 3. This is due to the fact that operations such as `LUI` and `LLI` are part of I-Type instructions.

As part of the requirements, you **must** use a SystemVerilog user-defined data type for defining all `func` values. The user-defined data type may be declared as in Listing 1.1

```
typedef enum logic [5:0] { // Width and type of data type
  ADD_FUNC: 6'b100000,     // addition
  SUB_FUNC: 6'b100001,     // subtraction
  // all remaining func values
} func_t;                  // Name of data type
```

Listing 1.1: Definition of `func_t` data type.

7.5 SystemVerilog design files

The minimum required SystemVerilog design files are listed below. You may decide to create new SystemVerilog modules and files in order to create different hierarchy levels. For example, you may decide to create a SystemVerilog module specifically for instruction decoding inside the control unit.

- `mips_pkg.sv`. Package file common to all design files. You must declare all the necessary parameters and data types in this file.
- `rtype.sv`. R-Type top-level module. This module instantiates all the submodules such as `rf`, `alu`, `sgn_ext`, `mux_2x1`, and `control_unit`. The connections among the different

submodules may be done using glue logic, *i.e.*, signal assignments for interconnecting all submodules. For example, the input port `instruction` must divided into several signals for different submodules.

- `rf.sv`. RF module description.
- `alu.sv`. ALU module description.
- `sgn_ext.sv`. Zero/sign extender module description.
- `mux_2x1.sv`. 2x1 MUX module description.
- `control_unit.sv`. Control unit module description.

8 Grading criteria

The following grading criteria will be considered.

1. **The correct functionality of your designs.** I will use my own testbenches in order to automatically stress your designs and verify that they perform the tasks according to the specifications. For example, I will try different values for the parameters in your designs and I expect them to still perform according to the specifications. This is why it is paramount that you follow the name convention specified for file names and port names. Moreover, it is important that your designs and testbenches compile in ModelSim without errors. **Your maximum grade for this assignment will automatically drop to 50/100 should ModelSim trigger a compilation or simulation error.**
2. **The quality of your testbenches.** Even though I will use my own testbenches, I expect you to consider a thorough and concious set of test scenarios. In this way, you should be able to spot any mismatches between the expected results and the actual results provided by your designs.
3. **Your designs must be synthesized in Quartus without latches and without errors.** Warnings are tolerated at this point. **Your maximum grade for this assignment will automatically drop to 50/100 should Quartus trigger a synthesis error or generate unwanted latches.** Remember that a design is not useful if it can't be synthesized.

9 Deliverables and Submission instructions

Prepare a single `zip` file containing **at least** the following files. You may have created additional SystemVerilog modules as stated in Section 7.5. If that's the case, please include them in your `zip` file.

1. `mips_pkg.sv`.
2. `rtype.sv`.
3. `tb_rtype.sv`. Testbench for your top-level R-Type design.
4. `rf.sv`.
5. `alu.sv`.
6. `sgn_ext.sv`.

7. `mux_2x1.sv`.
8. `control_unit.sv`.
9. `rtype.do`. ModelSim script for compiling and simulating your design. You may decide to breakdown your ModelSim script into several scripts for separating each design step.
10. A Quartus Register-Transfer Level (RTL) screenshot of your synthesized top-level R-Type design.

Submit your assignment through Canvas **no later** than 23:59 hours on Wednesday October 7th 2020.
Please send any questions to isaac.perez.andrade@tec.mx.