Final Project
MIPS

1 Objectives

- To implement the datapath of a custom Microprocessor without Interlocked Pipeline Stage (MIPS) in SystemVerilog.
- To implement a MIPS assembler program in order to multiply two signed numbers.
- To implement a MIPS assembler program in order to calculate the first n-th Fibonacci numbers.
- To implement a MIPS assembler program in order to calculate the first n-th element in a sum-of-square series.

2 Final grade weight

This delivery constitutes 25% of your final grade.

3 Deadline

23:59 hours on Thursday June 4th 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 the previous assignment for modelling the MIPS R-Type and I-Type instructions in SystemVerilog.

6 Background

The following sections provide some background on the tasks you are required to complete for this assignment.

6.1 Fibonacci sequence

A Fibonacci number F_n is defined as the sum of the two previous Fibonacci numbers as defined in Equation (1.1)

$$F_n = F_{n-1} + F_{n-2} \tag{1.1}$$

with $n >= 2$ and $F_0 = 0$ and $F_1 = 1$.

Moreover, the n-th order Fibonacci sequence is defined as $\{F_0, F_1, F_2, \ldots, F_{n-2}, F_{n-1}, F_n\}$. For example, the 5-th order Fibonacci sequence is $\{F_0, F_1, F_2, F_3, F_4, F_5\} = \{0, 1, 1, 2, 3, 5\}$.

6.2 Sum of squares

The n-th term in a sum of square series is defined as in Equation (1.2)

$$S_n = \sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + \cdots + (n-2)^2 + (n-1)^2 + n^2 \tag{1.2}$$

Moreover, the n-th order sum of square sequence is defined as $\{S_1, S_2, \ldots, S_{n-2}, S_{n-1}, S_n\}$. For example, the 5-th order sum of square sequence is $\{S_1, S_2, S_3, S_4, S_5\} = \{1, 5, 14, 30, 55\}$.

7 Project 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.
- Register File (RF) contains 32 registers.
  - `r0` **must** always be 0, *i.e.*, `r0` is not writeable.
  - `r31` is return address.
- Instruction Memory (IM) contains 256 address.
- Data Memory (DM) contains 256 address.

7.2 MIPS top-level

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

This top-level module consists of only inputs `clk` and `asyn_n_rst`. There are no outputs in this top-level module. You must include the IM and the Program Counter (PC) as part of your MIPS design.

7.3 List of instructions

Table 2 shows the minimum set of instructions that your MIPS should support. You may support additional instructions that you deem necessary.

7.4 MIPS test programs

The following sections describe three MIPS test programs that you must complete for this assignment. All three programs should be designed using assembly language with the instructions you have designed in your custom MIPS. You must follow the mnemonic convention for each instruction in Table 2.

Multiplication

You are required to design an assembly program for multiplying 2 signed numbers. For this part of the assignment you must consider that the two operands A and B are placed in `Mem[0]` and `Mem[1]`. As a result of this, your testbench must initialize your DM with the two operands. The result of the signed multiplication must be stored in `Mem[2]`.

Fibonacci sequence

You are required to design an assembly program for calculating the n-th order Fibonacci sequence. For this part of the assignment, you must consider that the value of n is stored in `Mem[255]`. As a result of this, your testbench must initialize `Mem[255]` with the value of n. The $n + 1$ elements of the Fibonacci sequence must be stored from `Mem[0]` to `Mem[`n`]`. For example, for the 5-th order Fibonacci sequence, the results must be stored in `Mem[0]`, `Mem[1]`, `Mem[2]`, `Mem[3]`, `Mem[4]`, `Mem[5]`.

Sum of squares sequence

You are required to design an assembly program for calculating the n-th order sum of squares sequence. For this part of the assignment, you must consider that the value of n is stored in `Mem[0]`. As a result of this, your testbench must initialize `Mem[0]` with the value of n. The n elements of the sum of squares sequence must be stored from `Mem[1]` to `Mem[`n`]`. For example, for the 5-th order sum of squares sequence, the results must be stored in `Mem[1]`, `Mem[2]`, `Mem[3]`, `Mem[4]`, `Mem[5]`.

Table 2: Required MIPS instructions.

Instruction	Syntax	Meaning
	R-Type	
ADD	ADD rd, rs, rt	Reg[rd] ← Reg[rs] + Reg[rt]
SUB	SUB rd, rs, rt	Reg[rd] ← Reg[rs] - Reg[rt]
NAND	NAND rd, rs, rt	Reg[rd] ← ~(Reg[rs] & Reg[rt])
NOR	NOR rd, rs, rt	Reg[rd] ← ~(Reg[rs] | Reg[rt])
XNOR	XNOR rd, rs, rt	Reg[rd] ← ~(Reg[rs] ^ Reg[rt])
AND	AND rd, rs, rt	Reg[rd] ← Reg[rs] & Reg[rt]
OR	OR rd, rs, rt	Reg[rd] ← Reg[rs] | Reg[rt]
XOR	XOR rd, rs, rt	Reg[rd] ← Reg[rs] ^ Reg[rt]
SLL	SLL rd, rs, sa	Reg[rd] ← Reg[rt] << sa
SRL	SLL rd, rs, sa	Reg[rd] ← Reg[rt] >> sa
SLA	SLL rd, rs, sa	Reg[rd] ← Reg[rt] <<< sa
SRA	SLL rd, rs, sa	Reg[rd] ← Reg[rt] >>> sa
JR	JR rs	PC ← Reg[rs]
	I-Type	
ADDI	ADDI rt, rs, imm	Reg[rt] ← Reg[rs] + imm
SUBI	SUBI rt, rs, imm	Reg[rt] ← Reg[rs] - imm
NANDI	ANDI rt, rs, imm	Reg[rt] ← ~(Reg[rs] & imm)
NORI	ORI rt, rs, imm	Reg[rt] ← ~(Reg[rs] | imm)
XNORI	XORI rt, rs, imm	Reg[rt] ← ~(Reg[rs] ^ imm)
ANDI	ANDI rt, rs, imm	Reg[rt] ← Reg[rs] & imm
ORI	ORI rt, rs, imm	Reg[rt] ← Reg[rs] | imm
XORI	XORI rt, rs, imm	Reg[rt] ← Reg[rs] ^ imm
LUI	LUI rt, imm	Reg[rt] ← {imm[7:0], 8'b0}
LLI	LLI rt, imm	Reg[rt] ← {8'b0, imm[7:0]}
LI	LI rt, imm	Reg[rt] ← imm
LW	LW rt, imm(rs)	Reg[rt] ← Mem[Reg[rs]+imm]
SW	SWR rt, imm(rs)	Mem[Reg[rs]+imm] ← Reg[rt]
BEQ	BEQ rt, rs, imm	if (Reg[rs] == Reg[rt]) then PC ← imm
BNEQ	BNEQ rt, rs, imm	if (Reg[rs] != Reg[rt]) then PC ← imm
BZ	BZ rt, rs, imm	if (Reg[rs] == 0) then PC ← imm
BNEG	BNEG rt, rs, imm	if (Reg[rs] < 0) then PC ← imm
	J-Type	
J	J imm	PC ← imm
JAL	JAL imm	Reg[31] ← PC+1 PC ← imm

7.5 SystemVerilog design and testbench files

The minimum required SystemVerilog design files and testbenches 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.

- Desing units.
  - mips_pkg.sv. Package file common to all design files. You must declare all the necessary parameters and data types in this file.
  - mips.sv. Top-level MIPS module.
  - alu.sv. Arithmetic and Logic Unit (ALU) module description.
  - rf.sv. RF module description.
  - sgn_ext.sv. Zero/sign extender module description.
  - mux_2x1.sv. 2x1 Multiplexer (MUX) module description.
  - mux_4x1.sv. 4x1 MUX module description (if needed).
  - mux_8x1.sv. 8x1 MUX module description (if needed).
  - control_unit.sv. Control unit module description.
  - dm.sv. DM module description.
  - im.sv. IM module description.
  - program_counter.sv. PC module description.
- Testbenches.
  - tb_mips.sv. Top-level MIPS testbench. This testbench tests all possible MIPS instructions.
  - tb_multiplication.sv. Testbench for testing the multiplication program.
  - tb_fibonacci.sv. Testbench for testing the Fibonacci program.
  - tb_sum_squares.sv. Testbench for testing the sum of squares program.
- IM initialization files. These files may be in either binary or hexadecimal format.
  - mips_tests.txt. File for loading IM in tb_mips.sv.
  - multiplication.txt. File for loading IM in tb_multiplication.sv.
  - fibonacci.txt. File for loading IM in tb_fibonacci.sv.
  - sum_squares.txt. File for loading IM in tb_sum_squares.sv.

8 Report

You are required to submit a report, which must include the following sections.

1. **Introduction**. An introduction to the report and to the project.
2. **MIPS Instruction Set Architecture (ISA) design**. An explanation of your MIPS ISA. This section must include:
   - A table showing your instruction encoding. This must include **ALL** your supported instructions.
   - An explanation of your design choices and design flow.
   - A computer-drawn schematic diagram showing your final Microarchitecture (μA) of your MIPS design.

(a) This μA must support all R-Type, I-Type and J-Type MIPS instructions.
(b) All signals must be clearly labelled, including bus widths.

3. **Design synthesis.** You must include a screenshot of the generated Register-Transfer Level (RTL) in Quartus.

4. **Test programs.** For each test program, a table similar to Table 3 explaining your test files (`.txt`) line by line. This explanation must include the binary or hexadecimal value loaded into your IM and the test you are providing.

   Table 3: Example of test file in your report.

Line	Hex code	Meaning
1	`FEDCBA98`	`ADDI r1, r2, -16`
2	`76543210`	`BNE r3, r4, 123`

   For each test program, you must also state the limitation of your program. For example, you must indicate the maximum value of n that your MIPS μA supports for the Fibonacci and sum of squares programs. Similarly, you must indicate the maximum and minimum possible results for your multiplication program. This section must also include screenshots of your ModelSim simulation for each test program.

5. **Improvements and future work.** Mention any potential improvements to you MIPS ISA and/or μA.

6. **Conclusions.** Concluding remarks on what you've learned in this project.

7. **References.** List of citations in IEEE format used for the report.

NOTE. There is no restriction on the number of pages. However, quality is prioritized over quantity. A lengthy but meaningless report will result in a lower score than a shorter but more meaningful report.

9 Grading criteria

The grade of this project is divided as presented in Table 4.

Table 4: Grading criteria.

Rubric	Percentage
J-Type implementation & testbench	5%
Multiplication program	25%
Fibonacci program	25%
Sum of squares program	25%
Report	20%
TOTAL	100%

In addition to the grading criteria specified in Table 4, I will consider the following aspects.

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 warnings and without errors. **Each RTL warning will deduct 5% of your final project grade. 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, without RTL warnings and without errors. Each RTL warning will deduct 5% of your final project grade. Your maximum grade for this assignment will automatically drop to 50/100 should Quartus trigger a synthesis error or generate unwanted latches.**

   The only warnings that will be tolerated are those that are not related to the RTL itself. Examples of such warnings are:

   - `Warning (18236): Number of processors has not been specified which may cause overloading on shared machines. Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate value for best performance.`
   - `Warning (13046): Tri-state node(s) do not directly drive top-level pin(s).`
   - `Warning (292013): Feature LogicLock is only available with a valid subscription license. You can purchase a software subscription to gain full access to this feature.`
   - `Warning (15714): Some pins have incomplete I/O assignments. Refer to the I/O Assignment Warnings report for details.`

   Remember that a design is not useful if it can't be synthesized.

10 Deliverables and Submission instructions

Prepare a single `zip` file containing all your design, testbench and test program files. You may follow the list provided in Section 7.5 as a reference. 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. In addition to the design, testbench and test program files, you must include the following ModelSim scripts.

1. `mips.do`. ModelSim script for compiling and simulating your design. This will be used for testing the J-Type instructions, along with the R-Type and I-Type instructions.

2. `multiplication.do`. ModelSim script for compiling and simulating your design using the multiplication test program.

3. `fibonacci.do`. ModelSim script for compiling and simulating your design using the Fibonacci test program.

4. `sum_squares.do`. ModelSim script for compiling and simulating your design using the sum of squares test program.

Submit your assignment through Blackboard **no later** than 23:59 hours on Thursday June 4th 2020. Only one submission per team is necessary.
Please send any questions to isaac.perez.andrade@tec.mx.