Project Brief

Pipeline MIPS Processor Verilog Design

Group #15

Mohamed Ashraf Elattar
Mohamed Magdy Taha
Ahmed Mahmoud Awad
Mohamed Ahmed Fouad
Ahmed Mohamed Antar

Table of contents:

1.Intro:

1.1. Short Description:

This Project is simply a Design of pipeline **MIPS** processor written in **Verilog (**VHDL) with handling of Data hazards and forwarding process to achieve the maximum performance in units of clock cycles.

Also, this Design Code is written in simple way without any complexities, and every single module is written in a separate file to avoid coding traffic and to manage and distribute tasks on all members in the team; which makes the development process easier and co-operative.

1.2. Attached Files:

With this project brief you will find the design Verilog code which is all with extension ".v" except the program assembly code that you will write as a binary file of extension ".txt".

As we said before in the previous section every single module was written in separate file, so surely you will find file for specified module you want to look at with no effort.

In addition, there are some //comments for some ports' names and the behavior description for other code lines.

1.3. Workflow management:

At the beginning, we made a project plan in addition to some guide lines to make the integration process more easy and quick.

We created a "GitHub" repository to handle task completion and final integrations so every task finished by one of the team members is pulled with request to be added to the Master repository.

After adding the task, the rest of team members review it and ask for explanation in an online meeting to discuss every single detail that may not be clear and from here the FAQ comes. After that, the module is then updated if there is something to be edited for optimization or to get better results at all the test cases.

1.4. Process Phases:

The project process was divided mainly in 2 phases:
Phase One:
The single cycle MIPS processor was implemented and the modules of the single cycle were integrated, after that we tested it with a neat Test-bench and it totally worked after some bugs and issues which were of course solved and cured.

Phase Two:

In phase 2 we continued our work to implement the pipeline MIPS processor by implementing the modules of hazard detection unit and forwarding units in addition to the four Pipeline registers and other modules.

After finishing all the modules and their implementation, we made a test bench for the pipelined processor with different test cases to test data and control hazards handling and it successfully could handle these with good optimization in time.

Note: in the second section we will get a lot deeper in Design and other technicalities.

2. Design:

2.1. Modules:

As we mentioned before that every single module is in a single file attached in the project folder.

In the single cycle the following modules are implemented:

- ALU32Bit.
- ALUControl.
- ControlUnit.
- Adder.
- InstructionMemory.
- PC.
- PCAdder.
- RegisterFile.

- ShiftLeft2.
- SignExtend.
- Mux2x1.
- DataMemory.

All these modules are tasked on all members of the team to be fulfilled, which made the process very easy and quick to be finished.

In **phase 2** we managed to implement the **pipeline MIPS,** so we implemented the necessary modules for hazard detection and handling and forwarding process in addition to cover all probabilities of data and control hazards, so the next modules were also tasked to the team again and finished in a good and professional way after many online meeting for brainstorming and discussion.

Phase 2 Modules for pipelining:

- IF_ID_reg.
- ID_EX_reg.
- EX_MemReg.
- Mem_Wbreg.
- Comparator (Beq).
- HazardDetectionUnit.
- ForwardingUnit.
- Mux3x1.

  - The IF_ID_reg.v file is the register file between the instruction Fitch stage and instruction decoding stage which will pass the stored result of the first stage to the next stage (instruction decoding stage) with the next clock cycle.

- The ID_EX_reg.v file is the register file between the instruction Decoding stage and Execution stage which will pass the stored result of the first stage to the next stage (Execution stage) with the next clock cycle.

- The EX_MemReg.v file is the register file between the Execution stage and Memory stage which will pass the stored result of the first stage to the next stage (Memory stage) with the next clock cycle.

- The Mem_Wbreg.v file is the register file between the Memory stage and Write back stage which will pass the stored result of the first stage to the next stage (Write back stage) with the next clock cycle.

- Comparator is module between the ReadData1 and ReadData2(the outputs of register file) to compare between them to optimize branch instructions and the output signal of this module will be flag of equality between the two Data if equal the flag will equal 1 else the flag will equal 0.

- Hazard detection unit is that module which detect if there is data hazard and handle it, this module covers all data hazard cases (RAW, WAR, RAR).

- Forwarding unit by definition is made to handle forwarding process between different clock cycles and instructions to achieve the maximum optimization of clock cycle.

2.2. Testing and Test Benches:

Testing all these modules was not easy for **phase 1** or **phase 2** but it got harder in **phase 2** after attaching pipeline modules to the design for that we created a unique test bench for every phase.

For **phase 1** we were concerned on testing the more complex code to make sure that the designis working all the way right and efficient with also a maximum optimization, attached for the folder a file "**MIPSTestBenchPhase1.v**" which contain the test bench that we used and also attached file "**Code.txt**" which contain the code we used for testing.

For **Phase2** we made a new separate Test Bench module which was more clear and easier to integrate the Pipeline modules. It's also attached to the folder with the test code used to test the pipeline to achieve optimization goals.

2.3. Test cases and outputs

Regfile initialization:

```
10
11          always @(posedge reset)
12                  begin
13                  memory[0] <= 32'h00000000;//$zero
14                  memory[8] <= 32'h00000001;//$t0
15                  memory[9] <= 32'h00000002;//$t1
16                  memory[10] <= 32'h00000000;//$t2
17                  memory[11] <= 32'h00000000;//$t3
18                  memory[12] <= 32'h00000000;//$t4
19                  memory[13] <= 32'h00000000;//$t5
20                  memory[14] <= 32'h00000000;//$t6
21                  memory[15] <= 32'h00000000;//$t7
22                  memory[16] <= 32'h00000000;//$s0
23                  memory[17] <= 32'h00000000;//$s1
24                  memory[18] <= 32'h00000003;//$s2
25                  memory[19] <= 32'h00000003;//$s3
26                  memory[20] <= 32'h00000004;//$s4
27                  memory[21] <= 32'h00000000;//$s5
28                  memory[22] <= 32'h00000008;//$s6
29                  memory[23] <= 32'h00000000;//$s7
30                  memory[24] <= 32'h00000000;//$t8
31                  memory[25] <= 32'h00000000;//$t9
32                  memory[31] <= 32'h00000000;//$ra
33                  end
```

>Test case 1 :

```
add $t2 $t0 $t1
sw $t2 0($zero)
sub $t3 $t2 $t1
beq $t3 $t0 -4
sw $t2 0($zero)
```

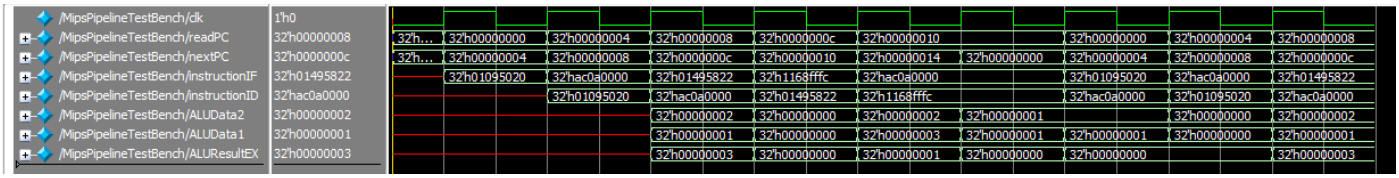
Memory:

```
/MipsPipelineTestBench/dataMemory/memory
   0: 00000003 xxxxxxxx xxxxxxxx xxxxxxxx
   4: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
   8: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
  12: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
  16: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
  20: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
  24: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
  28: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
```

RegisterFile:

```
/MipsPipelineTestBench/regiterFile/memory
   0: 00000000 xxxxxxxx xxxxxxxx xxxxxxxx
   4: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
   8: 00000001 00000002 00000003 00000001
  12: 00000000 00000000 00000000 00000000
  16: 00000000 00000000 00000003 00000003
  20: 00000004 00000000 00000008 00000000
  24: 00000000 00000000 xxxxxxxx xxxxxxxx
  28: xxxxxxxx xxxxxxxx xxxxxxxx 00000000
```

Wave:



>Test case 2:

```
Add $s5 $t0 $t1
Add $s1 $s2 $s4
Beq $s1 $s5 L1
Sub $s3 $s6 $s3
```

```
L1: lw $t2 10($t4)
```

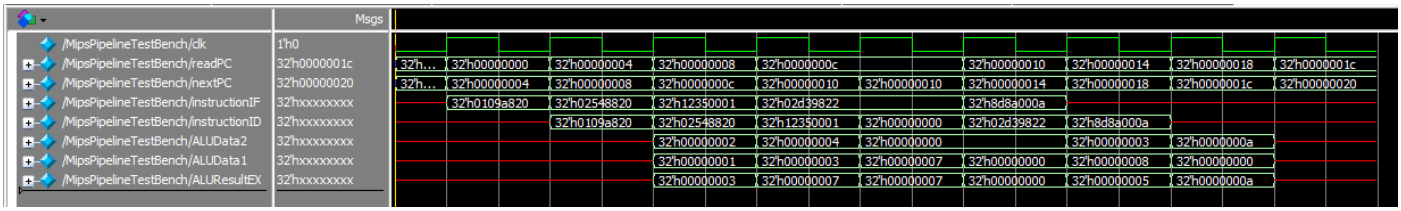
Memory :

```
/MipsPipelineTestBench/dataMemory/memory
   0: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
   4: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
   8: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
  12: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
  16: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
  20: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
  24: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
  28: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
```

Regfile:

```
   0: 00000000 xxxxxxxx xxxxxxxx xxxxxxxx
   4: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
   8: 00000001 00000002 00000000 00000000
  12: 00000000 00000000 00000000 00000000
  16: 00000000 00000007 00000003 00000005
  20: 00000004 00000003 00000008 00000000
  24: 00000000 00000000 xxxxxxxx xxxxxxxx
  28: xxxxxxxx xxxxxxxx xxxxxxxx 00000000
```

Wave:



>test case 3:

```
Add $s1 $t0 $t1
Lw $s2 4($s4)
Beq $s2 $s3 L1
Sub $t2 $t5 $t6
L1: sw $s1 8($t4)
```

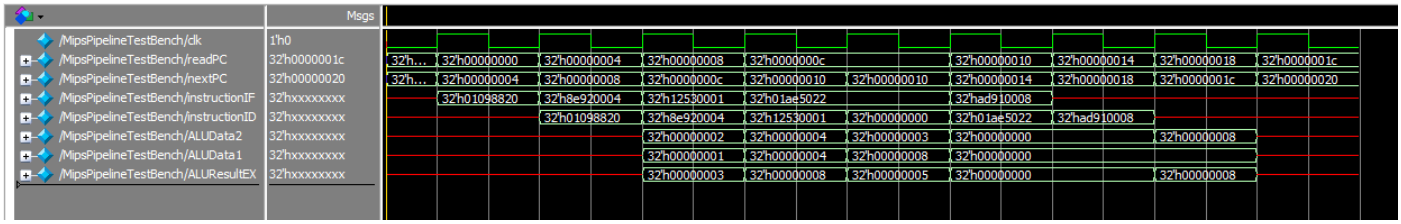
Memory:

```
/MipsPipelineTestBench/dataMemory/memory
   0: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
   4: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
   8: 00000003 xxxxxxxx xxxxxxxx xxxxxxxx
  12: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
  16: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
  20: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
  24: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
  28: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
```

Regfile:

```
/MipsPipelineTestBench/regiterFile/memory
   0: 00000000 xxxxxxxx xxxxxxxx xxxxxxxx
   4: xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
   8: 00000001 00000002 00000000 00000000
  12: 00000000 00000000 00000000 00000000
  16: 00000000 00000003 xxxxxxxx 00000003
  20: 00000004 00000000 00000008 00000000
  24: 00000000 00000000 xxxxxxxx xxxxxxxx
  28: xxxxxxxx xxxxxxxx xxxxxxxx 00000000
```

Wave:



3. Brief description:

3.1. How to Use This Design?

All you have to do is to download the attached file and you should have any **IDE** to run all these codes, this design is open source code you can use it without any license and even edit it.

You could also follow the progress and updates of this project on its repository on GitHub on this link: https://github.com/mohamedel3attar/Mips-Pipeline-Verilog-Design.

3.2. How to Use Your Own Program?

Just write it in the file called “code.txt”

4. Configurations:

4.1. Supported Instruction Types

Our Pipeline processor not only supports R-Type instructions but also supports I-Type instructions.

R-type instructions supported: Add,Sub,And,Or,Nor,Sll,Srl.
I-Type instructions supported: Addi,Andi,Ori,lw,sw.

4.2. Tips for Your Own Code Writing

To write your own piece of code as an input to the processor you will need to write it as an assembly code then, you will have to use any online tool to convert assembly instructions into Hexa code. We recommend using this online tool:

http://www.kurtm.net/mipsasm/

5. Synthesizability

The screenshot in the next page show the synthesizability summary report. The full Xilinix report is attached with the files.