

MIPS Processor Project

G10

Under Supervision of:

Prof\Ashraf Salem

Eng\Ahmed Fathy

Contents:

Sec A: Introduction to our Project

A.1-Description of MIPS project

A.2-Description of Design

A.3-Datapaths & Added Extensions

A.4-Supported Instructions

A.5-Software Description

Sec B: Role of each member

B.1-Data of Members

B.2-Role of Each

Sec C: Testing

C.1-Testing Methodology

C.2-Test Cases

Sec A

A.1

MIPS project is an Implementation of MIPS processor in Verilog hardware description- language. the designed processor implementation supports different operations in Assembly such as: add, and, or, lw, sw, addi, ori, andi, xori, Jr, Jal, J, beq, etc…

Our project is synthesized successfully and without any warnings!

A.2

Our design consists of:

1-RegisterFile: has 32 Registers two of them are special purpose registers: $sp, $ra.

And the others are general purpose registers.

2-Instruction Memory: 32KB memory that has 8192 32-bit instruction spaces, capable of storing 32-bit instructions.

3-Data memory: 32KB memory that has 8192 32-bit word spaces, and is designed as follows: whenever memWirte is enabled it is then accessed and whenever memRead is enabled Data memory reveals the proposed address stored value, otherwise data (memory output is zero).

4-ALU: it's designed to do arithmetic and logic operations such as Add, Xor, And, Or, Sll, Srl/a, etc...

5-Control unit: we separated Alu control and Control unit so to make it easier while designing, we implemented it so that also func segment of the instruction is connected to support rare instructions such as jr.

6-ALU control: it's used to control the alu in it's different modes, its inputs are both Control unit and func segment.

7-Program counter and MUXs: we designed MUXs as modules to make it easy when implementing CPU module to handle jump operations easily, pc counter monitors the current address of instruction and calculates the next pc address with an external ALU (adder).

A.3.

Added modules:

* Jr MUX: to select between Jump MUX output and Jump register address.
* memtoReg MUX: with four inputs instead of two for $ra address and lui instruction.
* ALUInput MUX: to choose from ReadData 2, sign-extended I-field & zero-extended I-field.
* RegDst MUX: with four inputs to choose from rt, rd, $ra register addresses.

Added signals:

* JRMuxControl from ALUControl to control JR MUX.
* Zero-extended I-field for immediate logical operations.
* lui branch.
* Shamt field for sll/srl/sra instructions.
* Branch is now a 2-bit signal to support bne operation.
* AluOp is a 3-bit signal to support all instructions.
* AluControl output is a 4-bit signal to support all R-type instructions.

AluOp outputs:

|  |  |
| --- | --- |
| Output to ALUControl | ALU operation |
| 0 | Add for lw, sw, addi |
| 1 | Subtract for beq, bne |
| 2 | R-type instructions |
| 3 | ori |
| 4 | andi |
| 5 | xori |
| 6 | slti |

AluControl outputs (opcodes in case of R-type instruction):

|  |  |
| --- | --- |
| opcodes | ALU operation |
| 0 | AND |
| 1 | OR |
| 2 | Add |
| 3 | XOR |
| 6 | Subtract |
| 7 | slt |
| 8 | srl |
| 9 | sra |
| 12 | NOR |
| 13 | sll |

A.4

Our MIPS processor supports all operations in assembly of Types I, R, J

I type: lw, sw, addi, andi, ori, xori, slti, lui, beq, bne

R type: add, sub, and, or, xor, nor, slt, sll, sra, srl, jr

J type: j, jal

Also MIPS supports Sign extension and zero extension.

A.5

We used python to write our software That's used for testing also we provided GUI so as to make it user friendly, we used Tkinter as our GUI toolkit.

Sec B

B.1) Our team members:

1-حازم محمد محمود النشار 1600471

2-جون عصام حلمي 1600460

3-عاصم كامل احمد 1600719

4-خالد عاطف عبد العزيز 1600504

5-حازم محمد زكي 1600469

6-طه محمد طه 1600714

7-احمد طه فكري طه 1600108

B.2) Role of each member:

1-Hazem Nashar:

Contributed to assembler program and GUI and also contributed in solving bugs in Verilog code.

2-John Essam:

Contributed to Verilog code implementation by implementing ALU and RegisterFile along with testing and debugging the code

3-Khaled Atef:

Contributed to Verilog design by implementing Memory and ControlUnit along with contribution in synthesis.

4-Assem Kamal:

Contributed to Verilog design by implementing other modules such as: instruction memory, ALU control unit, and tested then (unit testing).

5-Hazem M. Zaki:

Contributed to the overall debugging of the whole project and added some features that solved all bugs present in code, also contributed to synthesis.

6-Taha Mohammed:

Contributed to Verilog design by implementing MUXs and some other features inside mipsCPU module and contributed to unit testing for memory and ALU.

7-Ahmed Taha Fekry:

Contributed to writing test cases, finding out expected outputs of each test case, wrote the report.

Sec C

C.1) Testing Methodology

We started by testing each module individually, we made sure they gave the correct output for all cases. Then we started testing the top module (MIPS\_CPU) by applying each instruction individually and checked the outputs in register file, data memory and control unit. Finally, we prepared 10 testing programs that contain all types of instructions, leaf calls and nested calls, and then we ran them. All cases now run successfully with correct outputs after debugging.

C.1) Test Cases

Test cases with assembly/machine codes & expected register file/data memory outputs are attached in this file:

