# MIPS DESIGN PROJECT

Faculty of Engineering - Ain Shams University - Computer department

SUBMITTED BY:

Group 3:

| | |
|---|---|
| Pierre Nabil | (16E0056) Sec(1) |
| Gerges Michael | (1600446) Sec(1) |
| John Bahaa | (1600459) Sec(1) |
| David John | (1600531) Sec(2) |
| Adham Nour | (1600226) Sec(1) |
| Ziad Tarek | (1600606) Sec(2) |

Computer Organization

## Implementation:

In this project we chose to make a version of the MIPS processor that does all the instructions learned in the previous course, Computer Organization 1.

We used the Datapath given from Chapter 4 from the previous course and tweaked it to be able to execute all instructions from Chapter 2.

We started by redesigning the Datapath and sketching it in a reference sheet of paper. Then, we started delegating blocks to be written in Verilog and tested independently . After that, we gathered to assemble the processor and try some test code to check if everything was working properly. After debugging, we edited some of the code to be able to exit Verilog after execution of the code and print the memory contents and PC contents. Next, we checked for synthesizability using Xilinx. Finally, 1 person was responsible for writing the logic for the assembler, another person was responsible for creating the GUI, and another person was responsible for creating the 10 test code cases used in the automated testing part of the project.

## Non-Standard Assumptions:

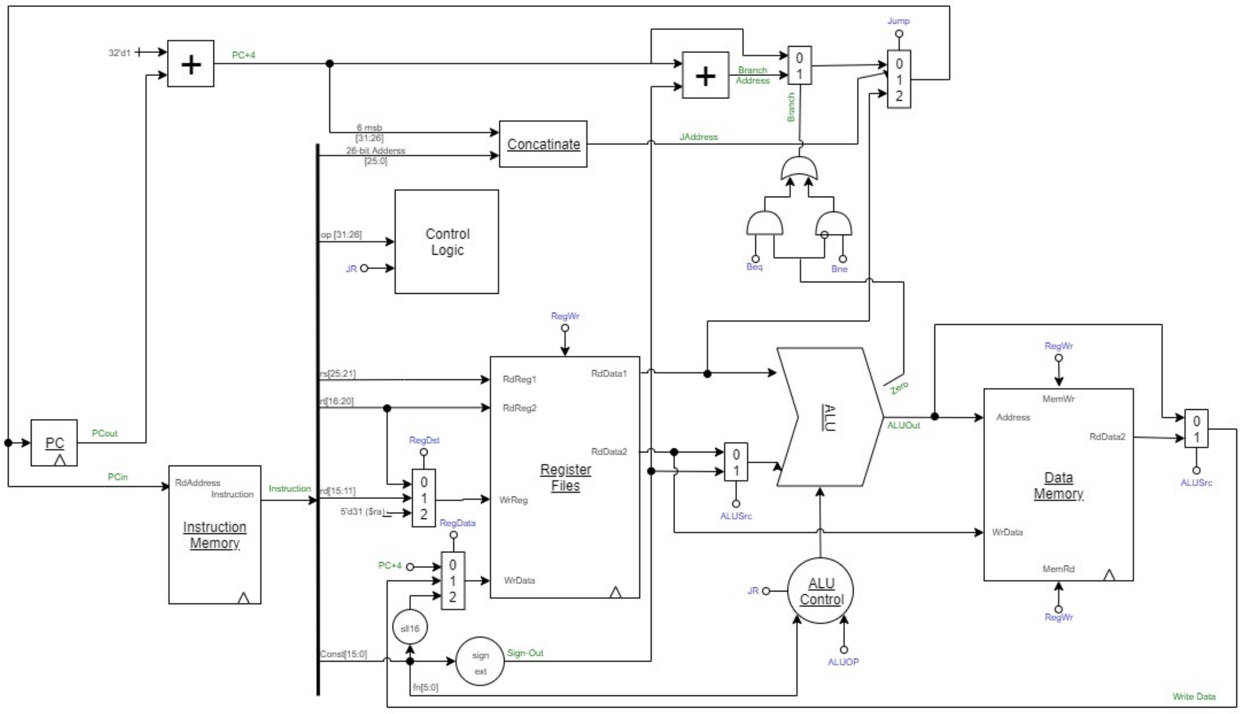
We had to change some things from the standard MIPS Datapath to accommodate for logic errors. Firstly, we connected PCin to the instruction memory instead of PCout. This was to avoid having the PC address take 2 clock cycles to return to the next instruction. This caused the PC to be different from the current instruction being executed. For example, the jump instructions used to execute the next instruction before jumping!

Another thing that we changed was we made the instruction memory and data memory arranged by word instead of by byte. this helped increase the memory capacity while helping in easier implementation in Veriog. However, this changed the standard of increasing PC by 4 for each instruction to be increasing the PC by 1. this changed the Datapath slightly, but not by much…

We designed the ALU to be able to do many more instructions than required. this means that we can increase the instruction set if needed. We also changed the ALU control standard to help accommodate the different instructions added to the MIPS Processor. We also created a JR output signal from the ALU Control Circuit to tell the Main Control Logic Circuit if the instruction was JR or some other Arithmetic or Logical instruction. This is to avoid creating one Large Control Circuit for the entire processor.

The data memory control lines were changed from 1 line each to 2 lines each. This is to differentiate between word, half word and byte instructions.

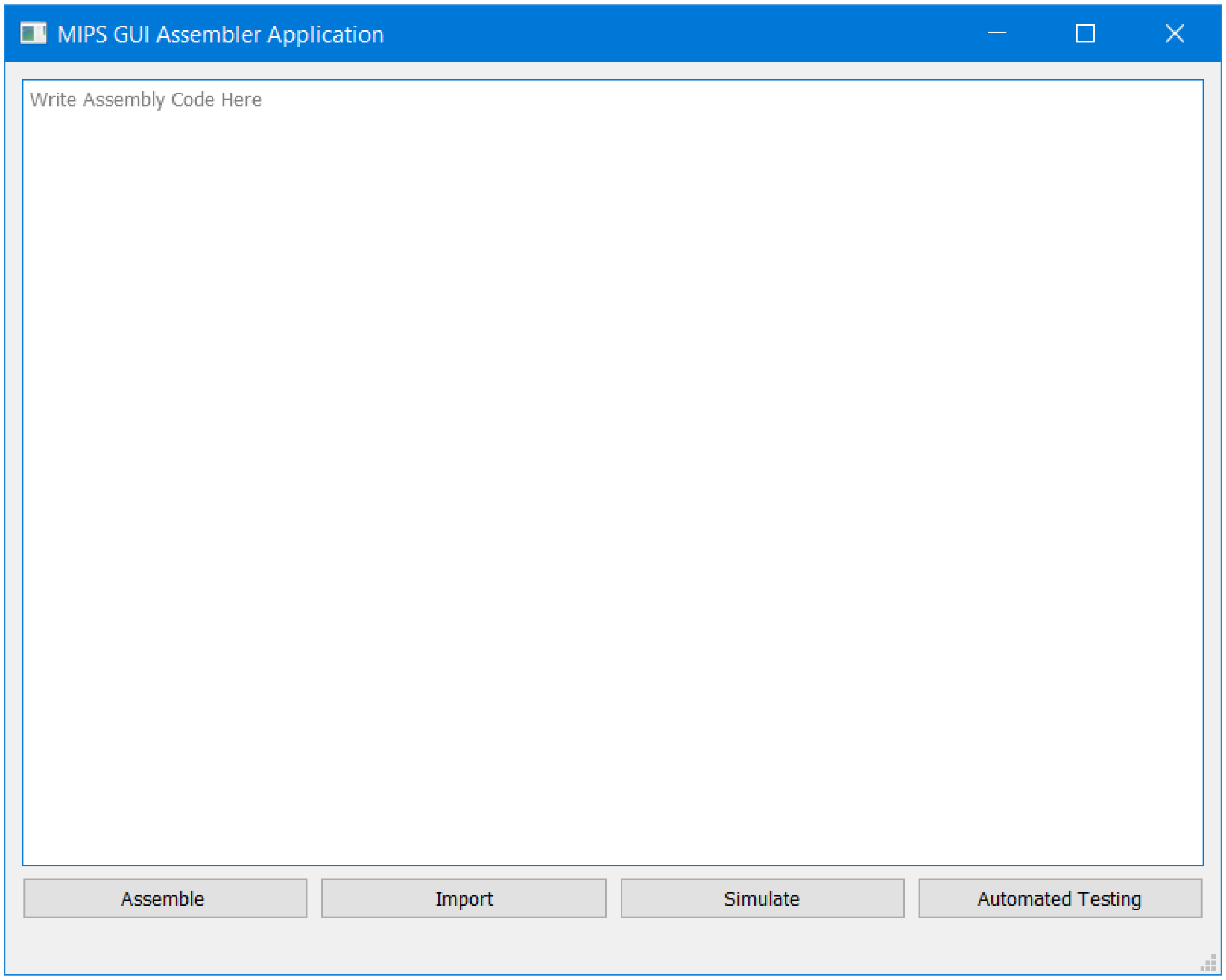
## Datapath Diagram:



## Instruction Set:

| Instructions | | Machine Code | | | | | | ALU Code | JR | Control Lines | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | op [31:26] | rs [25:21] | rt [20:16] | rd [15:11] | sh [10:6] | fn[5:0] | | | RegDst [1:0] | RegData[1:0] | RegWr | ALUSrc | ALUOp[3:0] | MemWr[1:0] | MemRd[1:0] | MemtoReg | Jump | Beq | Bne |
| R-Format | add $t1, $t2, $t3 | 0 | $t2 | $t3 | $t1 | 0 | 32 | 6 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | sub $t1, $t2, $t3 | 0 | $t2 | $t3 | $t1 | 0 | 34 | 7 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | and $t1, $t2, $t3 | 0 | $t2 | $t3 | $t1 | 0 | 36 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | or $t1, $t2, $t3 | 0 | $t2 | $t3 | $t1 | 0 | 37 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | xor $t1, $t2, $t3 | 0 | $t2 | $t3 | $t1 | 0 | 38 | 4 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | nor $t1, $t2, $t3 | 0 | $t2 | $t3 | $t1 | 0 | 39 | 3 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | slt $t1, $t2, $t3 | 0 | $t2 | $t3 | $t1 | 0 | 42 | 12 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | sll $t1, $t2, 5 | 0 | 0 | $t2 | $t1 | 5 | 0 | 13 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | srl $t1, $t2, 5 | 0 | 0 | $t2 | $t1 | 5 | 2 | 14 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | sra $t1, $t2, 5 | 0 | 0 | $t2 | $t1 | 5 | 3 | 15 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | jr $ra | 0 | | $ra | 0 | 0 | 8 | DNE | 1 | x | x | 0 | x | (NOP) 11 | 0 | 0 | 0 | 2 | 0 | 0 |
| I-Format | addi $t1, $t2, 5 | 8 | $t2 | $t1 | 16-bit immideate | | | 6 | 0 | 0 | 1 | 1 | 1 | 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| | andi $t1, $t2, 5 | 12 | $t2 | $t1 | 16-bit immideate | | | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | ori $t1, $t2, 5 | 13 | $t2 | $t1 | 16-bit immideate | | | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | xori $t1, $t2, 5 | 14 | $t2 | $t1 | 16-bit immideate | | | 4 | 0 | 0 | 1 | 1 | 1 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| | slti $t1, $t2, 5 | 10 | $t2 | $t1 | 16-bit immideate | | | 12 | 0 | 0 | 1 | 1 | 1 | 12 | 0 | 0 | 0 | 0 | 0 | 0 |
| | lui $t1, 5 | 15 | 0 | $t1 | 16-bit immideate | | | 6 | 0 | 0 | 2 | 1 | x | (NOP) 11 | 0 | 0 | 0 | 0 | 0 | 0 |
| | lw $t1, 8($t2) | 35 | $t2 | $t1 | 16-bit immideate | | | 6 | 0 | 0 | 1 | 1 | 1 | 6 | 0 | 1 | 1 | 0 | 0 | 0 |
| | lh $t1, 6($t2) | 33 | $t2 | $t1 | 16-bit immideate | | | 6 | 0 | 0 | 1 | 1 | 1 | 6 | 0 | 2 | 1 | 0 | 0 | 0 |
| | lb $t1, 5($t2) | 32 | $t2 | $t1 | 16-bit immideate | | | 6 | 0 | 0 | 1 | 1 | 1 | 6 | 0 | 3 | 1 | 0 | 0 | 0 |
| | sw $t1, 8($t2) | 43 | $t2 | $t1 | 16-bit immideate | | | 6 | 0 | x | x | 0 | 1 | 6 | 1 | 0 | 0 | 0 | 0 | 0 |
| | sh $t1, 6($t2) | 41 | $t2 | $t1 | 16-bit immideate | | | 6 | 0 | x | x | 0 | 1 | 6 | 2 | 0 | 0 | 0 | 0 | 0 |
| | sb $t1, 5($t2) | 40 | $t2 | $t1 | 16-bit immideate | | | 6 | 0 | x | x | 0 | 1 | 6 | 3 | 0 | 0 | 0 | 0 | 0 |
| | beq $t1, $t2, Label | 4 | $t2 | $t1 | 16-bit address | | | 7 | 0 | x | x | 0 | 0 | 7 | 0 | 0 | 0 | 0 | 1 | 0 |
| | bne $t1, $t2, Label | 5 | $t2 | $t1 | 16-bit address | | | 7 | 0 | x | x | 0 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 1 |
| J-Format | j Label | 2 | 26-bit address | | | | | DNE | 0 | x | x | 0 | x | (NOP) 11 | 0 | 0 | 0 | 1 | 0 | 0 |
| | jal Label | 3 | 26-bit address | | | | | DNE | 0 | 2 | 0 | 1 | x | (NOP) 11 | 0 | 0 | 0 | 1 | 0 | 0 |

## GUI & Automated Testing:



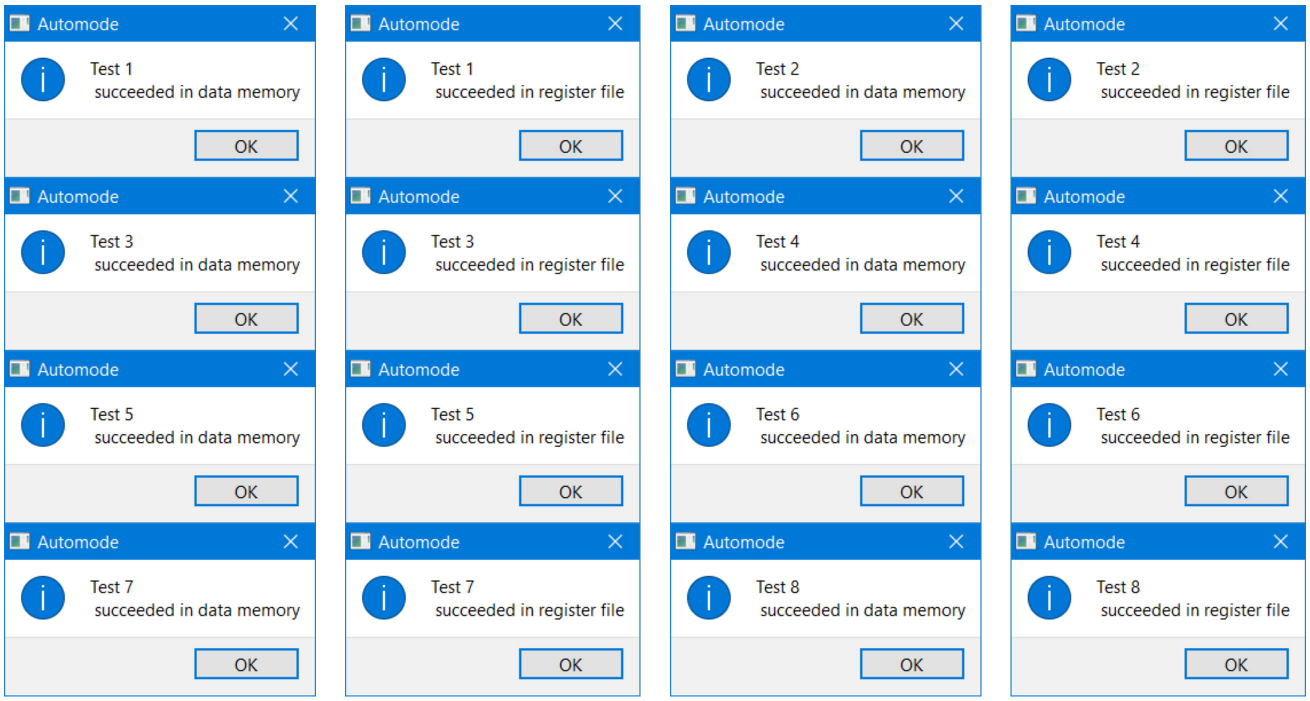
- The “Assemble” button takes the assembly code in the textbox above, assembles it, and stores the Assembly Code and Machine Code in separate files. These will be used later!

  It also shows both Assembly Code and Machine Code in separate Message Boxes.
- The “Import” button takes a File in the computer, assembles it, and stores the Assembly Code and Machine Code in separate files as before.

  It also shows both Assembly Code and Machine Code in separate Message Boxes.
- The “Simulate” button takes the saved file from the Assemble or Import Buttons, opens the command line and gives it an instruction to open ModelSim and run the machine code in the given file. It then stores the PC, Register Files and Data Memory Contents in separate files, and shows them in separate message boxes.

- The “Automated Testing” button takes 10 predefined files containing 10 test cases & 10 predefined folders containing the expected Memory Contents (Register Files and Data Memory), simulates each test code and checks if the Memory Contents match the ones in the answer folders.
  It then shows one message box for each verification done  (Register Files & Data Memory).

Automated Testing Screenshots:



Automode
Test 1
succeeded in data memory
OK

Automode
Test 1
succeeded in register file
OK

Automode
Test 2
succeeded in data memory
OK

Automode
Test 2
succeeded in register file
OK

Automode
Test 3
succeeded in data memory
OK

Automode
Test 3
succeeded in register file
OK

Automode
Test 4
succeeded in data memory
OK

Automode
Test 4
succeeded in register file
OK

Automode
Test 5
succeeded in data memory
OK

Automode
Test 5
succeeded in register file
OK

Automode
Test 6
succeeded in data memory
OK

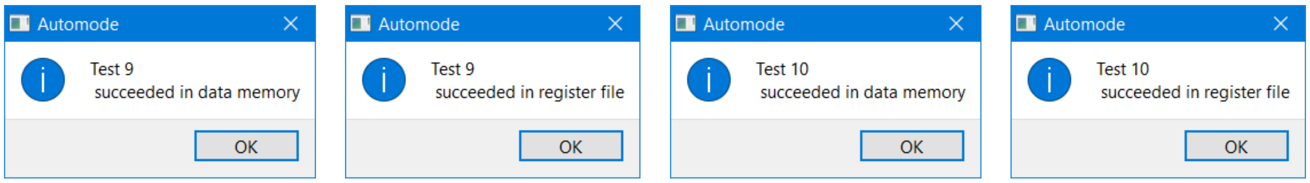
Automode
Test 6
succeeded in register file
OK

Automode
Test 7
succeeded in data memory
OK

Automode
Test 7
succeeded in register file
OK

Automode
Test 8
succeeded in data memory
OK

Automode
Test 8
succeeded in register file
OK



Automode
Test 9
succeeded in data memory
OK

Automode
Test 9
succeeded in register file
OK

Automode
Test 10
succeeded in data memory
OK

Automode
Test 10
succeeded in register file
OK

## Contribution:

Pierre Nabil:

- Team Leader
- Main Design
- Wrote the assembler logic

Gerges Michael:

- Main Design
- Wrote the ALU and ALU Control Blocks in Verilog

John Bahaa:

- Wrote some of the small blocks in the design.
- Created the 10 test Cases

David John:

- Main Design
- Wrote the Data Memory & Register Files Block in Verilog

Adham Nour:

- Wrote the Control Logic Block in Verilog
- Created the GUI.

Ziad Tarek:

- Wrote the Instruction Memory Block in Verilog
- Wrote some of the small blocks in the design.