**Pipelined MIPS processor Project**

Submitted to:

**Eng. Ahmed Fathy**

Submitted by:

* Nabil Negm ElDin
* Mamdouh Reda Mohamed El Nakeeb
* Manar Abdel Rahman
* Menna Ehab
* Hadi Hebat
* Ahmed Khaled Gamil

Contents

* INTRODUCTION ……….……………………… 1
* STEPS OF WORK.....………..……………………….. 2
* IMPLEMENTING A SINGLE CYCLE PROCESSOR ....…………….…………………………
* IMPLEMENTING PIPELINES , HAZARD DETECTION UNIT AND FORWARDING UNIT …
* INTEGRATING ALL OUR WORK ………………… …………………………………
* TESTING THE SYSTEM ……………………………

Introduction:



IMPLEMENTING A SINGLE CYCLE PROCESSOR:

* Instruction Memory:
* Register File:

Register file includes 32 32-bit general purpose registers.

This register file makes possible to simultaneously read from two registers and write into one register as it is appropriate for MIPS processor.

* The module has:
* Inputs:
* RegWrite
* Clk
* ReadR1
* ReadR2
* WriteR
* WriteD
* Output:
* ReadD1
* ReadD2
* ALU:
* The execution stage performs the operations on the data that is passed in from the ID/EX register. It contains the Arithmetic Logic Unit(ALU). The ALU is the calculator of the data path. It performs the actual mathematical operation: adding, subtracting, add ,or and shift left . The result of the ALU operation is stored into the EX/MEM Register along with the control for the MEM and WB stages forwarded from the ID/EX Register.
* Data Memory:
* The module has:
* Inputs:
* Data In
* Address
* Clock
* Memory Read
* Memory Write
* Output:
* Data Out.
* It checks on each positive edge of clock for memory read or memory write.
* The memory is a text file and the module is fetching and saving in this file.

IMPLEMENTING PIPELINES:

we use one pipeline register which takes the values from each stage when they are ready and sends them at the poetive edge of the clock cycle.

our test bench is called processor ...we generated a clock inside it for testing ...it includes all the stages and the pipeline register.

* **Instruction Fetch:**
* Get the next instruction from memory
* Increment Program Counter value by 4
* Simple instruction format means we know which registers we may need before the instruction is fully decoded

details:: it consists of 4 components

• Mux2to1: it chooses between the branch\_PC and the currentPC +1 ….the selector of the mux comes from the ID stage ..

• Adder :it adds the current PC and 4…

• PC : module that chooses between the PC and next PC (if there is branch hazard it chooses the current PC)…

• Instruction memory : its output is the instruction that is in the address (PC)..

This stage takes (the muxselector from ID stage , Pranch PC from ID stage and lw\_hazard from ID stage) and at the clk posedge the stage gets out (the PC and the Instruction to be excute)

* **Instruction Decode**
* Figure out what the instruction says to do
* Get values from the named registers

details:it consists of 3 components

- Register file : takes the addresses of the registers from the instruction and gives their values

- control unit: takes the opcode and gives control signals

- lw hazard :checks if this instruction needs a value from the previous instruction which is lw

this stage takes the instruction and decodes it sending it separated to execution stage

* **Execute Stage:**
* On a memory reference, add up base and offset
* On an arithmetic instruction, do the math

details: : it consists of 7 components

ALU: takes values and does opperations on them depending on the aluctrl

ALUcontrol:takes the op control signal and the function of the instruction then tells the alu what to do

2 values selecting muxes: to choose which value to send to the alu depending on the forwarding hazard unit

1 mux for the second input of the alu to choose to add reg or offset

1 destination mux to choose which reg to write back to

forward hazard unit: to check if the current instruction needs the output of the prev. instruction or the one before it

* **Memory Stage:**
* If load or store, access memory
* If branch, replace PC with destination address
* Otherwise do nothing
* The memory stage’s purpose is to read from and write to the data memory. The control signals passed in from the EX/MEM register: memwrite , memread to determine which of the operations to do. The output of the memory. Is written into the MEM/WB register along with the WB control that is passed from the EX/MEM register.
* **Write Back Stage:**
* The write back stage is responsible for taking the writing the data that we have just computed or loaded from memory out of the EX/MEM register and writing it to a one of the registers in the register file.
* The module has:
* Inputs:
* clk
* WBReg
* ReadDataMem
* AluResult
* ExMemReg
* RegFileAdress
* Output:
* ExMemReg1
* RegFileAdress1
* Result
* **Hazards:**
* **Data Hazard**
* Data hazard detection unit is used for the MIPS branch and load word instructions to stall the pipeline in the event that the next instruction needs its result.

The module has:

* Inputs:
* IDRegRs
* IDRegRt
* EXRegRt
* EXMemRead
* Output:
* PCWrite
* IFIDWrite
* HazMuxCon
* Forwarding unit takes a value from the stage ahead of it and uses it for its source value if the instruction ahead of it is writing to a register that the current instruction uses. This allows the CPU to perform faster because it doesn’t have to wait around for the instruction to finish before it begins.
* Inputs:
* MEMRegRd
* WBRegRd
* EXRegRs
* EXRegRt
* MEM\_RegWrite
* WB\_RegWrite
* Output:
* ForwardA
* ForwardB

They were added to avoid data dependencies within the pipeline.