A REPORT ON

MIPS PIPELINE PROCESSOR

In partial fulfilment of the course: Computer Architecture CS F342

Submitted by

PRATEEK MAHAJAN 2017A3PS0317P

ADITHYA SHANKAR BHATTIPROLU 2017A3PS0205P

Lab Section : 1

Instructors : Abheek Gupta, Kanika Monga

Instructor-in-Charge: Prof. S Gurunarayanan

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

RAJASTHAN, 333031

JULY, 2020

TABLE OF CONTENTS

IF STAGE

The IF (or Instruction Fetch) stage of the MIPS pipeline is essentially responsible for reading instructions from the instruction memory and sending them to the IF/ID pipelined registers. Further, it is responsible for updating the program counter so as to ensure that instructions are read from the correct address of the instruction memory. The IF stage essentially consists of-

a) The Instruction Memory
b) Program Counter Updater

Instruction Memory-

As suggested, the instruction memory contains the instructions that need to be executed by the processor. It takes the program counter as input and gives a 32 bit output (the instruction). Its values are initialised by a text file "Instn_mem.txt" which (in our code's case) has the following input-

00231021

00454022

00c24823

00a62820

00662021

10a60000

The above corresponds to the below instructions- (as given in the problem statement)

sub $2, $1, $3

and $8, $2, $5

or $9, $6, $2

add $5, $5, $6

sub $4, $3, $6

beq $5, $6

Program Counter Updater-

On the other hand, the PC updater essentially consists of a multiplexor that takes register PC_control as input and assigns the new value of PC to either PC+4 or the output address of a branch instruction (depending on the value of PC_control).

Beyond this, on every positive clock edge, the IF/ID registers are filled. Essentially, the 32 bit instruction and 8 bit PC are the only information provided by the IF stage.

ID STAGE

The ID Stage is essentially responsible for decoding the instructions and generating control signals corresponding to the instructions to control the operations of the forthcoming EX, MEM and WB stages.

The ID Stage essentially consists of 2 parts-

a) The Register Field
b) The Control Unit

Register Field-

The register field essentially refers to the field of registers that are used to store data for the processor's operations. In our case, the register field takes an input from bits 25:21 and 20:16 of the 32 bit instruction and sends the data saved at those addresses to the ID/EX register. It also has a facility to have data written into registers at given addresses (discussed in the WB stage). Note that we initialised the registers with the below values-

regfile[0][31:0]<= 0;

regfile[1][31:0]<= 10;

regfile[2][31:0]<= 0;

regfile[3][31:0]<= 5;

regfile[4][31:0]<= 0;

regfile[5][31:0]<= 0;

regfile[6][31:0]<= 3;

regfile[7][31:0]<= 0;

regfile[8][31:0]<= 0;

regfile[9][31:0]<= 0;

(the size of our register field is 10 registers. It can be changed as per requirement)

Control Unit-

The control unit, on the other hand, essentially refers to the part of the ID stage that is responsible for generating control signals to dictate the working of the forthcoming EX, MEM and WB stages of the pipeline. It uses the first 6 bits of the instruction (31:26) and the last 6 bits of the instruction (5:0) to decide which operations need to be activated in the coming stages of the pipeline. In our unit, we essentially assigned the following-

(alucon refers to the 3 control bits of the ALU; the registers in 'Other' refer to other control signals generated)

Instruction	Bits [31:26]	Bits [5:0]	alucon[2:0]	Other
ADD	000000	100000	000	alu_or_mem=1'b1 branch_input=1'b0 dst_control=1'b1
SUB	000000	100001	001	alu_or_mem=1'b1 branch_input=1'b0 dst_control=1'b1
AND	000000	100010	010	alu_or_mem=1'b1 branch_input=1'b0 dst_control=1'b1
OR	000000	100011	011	alu_or_mem=1'b1 branch_input=1'b0 dst_control=1'b1
BEQ	000100	XXXXXX	001	alu_or_mem=1'b0 branch_input=1'b1 dst_control=1'b0
LOAD	000110	XXXXXX	000	alu_or_mem=1'b1 branch_input=1'b0 dst_control=1'b1
STORE	000111	XXXXXX	000	alu_or_mem=1'b1 branch_input=1'b0 dst_control=1'b0

Beyond this, information is sent from the ID stage and IF/ID pipeline to the ID/EX pipeline for further stages. The ID/EX pipeline consists of the following data-

a) 32 bit instruction
b) 8 bit PC
c) 32 bit sign extended value generated from the last 16 bits of the instruction
d) Values of reg[25:21] and reg[20:16] in the register field
e) Control signals (alu_or_mem, branch_input, alucon, dst_control, etc)
f) 5 bit address values of Rs, Rt and Rd, in case needed later

EX STAGE

The inputs to this stage from the ID/EX pipeline register are -

a) PC
b) Rs value
c) Rt value
d) ALU control signals
e) 5 bit Registers Rs,Rt, and Rd
f) Destination control (control signal used to indicate if the destination register is middle 5 bits or lower 5 bits.)
g) Branch control signal
h) Writeback signal
i) Data memory read/write signal
j) 32 bit sign extended and left shifted number of las 16 bits or instruction

The EX stage in the MIPS Pipeline processor comprises-

a) ALU
b) Adder
c) Forwarding unit
d) 3 Multiplexers

The EX stage receives data from the ID/EX pipeline register at the positive edge of the clock and after all computation stores its results in the EX/MEM pipeline register at the next positive clock edge.

ALU-

The ALU as the name suggests is responsible for performing arithmetic (addition,subtraction) and logical operations (and,or) on the 2 32-bit numbers that it receives as input.

In our project the entire design of the ALU is in a file called “MIPSALU.v”.

Adder-

The adder is present for branch target address computation. This adder adds PC to a 32 bit sign extended and left shifted last 16 bits of the instruction. The result of this addition is the branch target address. This value is then stored in the EX/MEM pipeline register.

Forwarding Unit-

The forwarding unit is responsible for detecting the presence of a structural hazard. The possible hazards are -

a) EX/MEM.RegisterRd = ID/EX.RegisterRs

b) EX/MEM.RegisterRd = ID/EX.RegisterRt

c) MEM/WB.RegisterRd = ID/EX.RegisterRs

d) MEM/WB.RegisterRd = ID/EX.RegisterRt

If it's either case ‘a’ or case ‘b’ then the data that is input to the ALU comes from the EX/MEM pipeline register and not from the ID/EX pipeline register.

If it's either case ‘c’ or case ‘d’ then the data that is input to the ALU comes from the MEM/WB pipeline register and not from the ID/EX pipeline register.

If it isn’t any of the above four cases then there is no structural hazard present and the input to the ALU is the value from the ID/EX pipeline register.

The output of the forwarding unit are control signals that are inputs to the multiplexers in front of the ALU (These MUXES Decide the input to the ALU based on the instruction)

ForwardA and ForwardB are control outputs from the Forwarding unit and input to the multiplexers.

Index	Hazards	ForwardA	ForwardB
1.	EX/MEM.RegisterRd = ID/EX.RegisterRs	2’b01	2’b00
2.	EX/MEM.RegisterRd = ID/EX.RegisterRt	2’b00	2’b01
3.	MEM/WB.RegisterRd = ID/EX.RegisterRs	2’b10	2’b00
4.	MEM/WB.RegisterRd = ID/EX.RegisterRt	2’b00	2’b01

MEM STAGE

The stage receives inputs from the EX/MEM pipeline register. The values it gets as inputs are-

a) ALU output
b) ALU zero flag output
c) Branch target output
d) Branch control signal
e) Destination register
f) Data memory read/write signal
g) Writeback signal

The data memory read/Write signal is used for load and store instructions. The value is equal to 1'b0 for read operation and 1'b1 for write operation. The data memory is a txt file named "Data_memory.txt" that can be accessed using $readmh() command in the code. During load and store instructions the data is retrieved from this file (or placed into this file)

The branch control signal is a 1 bit signal that indicates whether the current instruction is a branch or not.

In the case of BEQ instruction 2 additional registers are passed as input in the instruction. Eg- beq $5, $6

In this case the register $5 and $6 are subtracted in the ALU and if the result is zero then the zero flag is set high. This zero flag is given as an input to a logical AND gate along with the branch control signal. The output of this AND gate is PCSrc (control signal that is input to MUX that is responsible for setting the value of PC every positive edge of the clock.

If the value of the Zero flag is not high then the condition is set to have failed and the value of PC is not set as the branch target address.

WB STAGE

The WB stage of the pipeline essentially refers to the stage in which, depending on whether the instruction requires it or not, the registers in the register field are written back into. This stage essentially uses only the register field. It uses the control signal generated in the ID stage (dst_control) and the value of the 5 'Rd' bits in the MEM/WB pipeline to write the output generated by the EX stage (or the MEM stage, depending on the operation) into the register at address Rd of the register field

OUTPUTS

After executing the instructions using the values specified, we get our final output as- (output of each instruction after it crosses the WB stage and exits the pipeline)

5

0

7

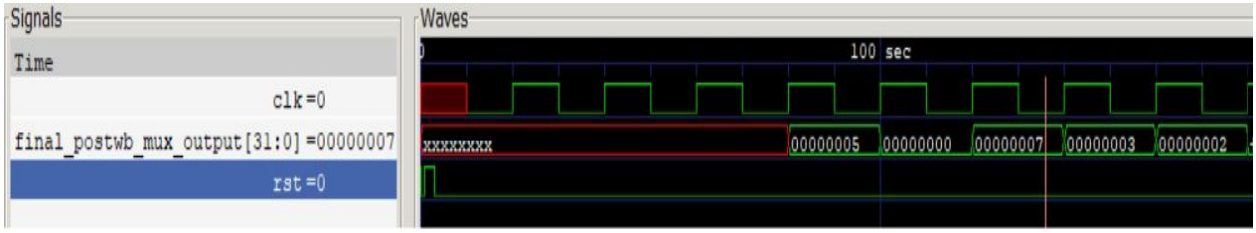
3

2

The terminal output of the processor designed by us can be seen below-

```
VCD info: dumpfile ALU123.vcd opened for output.
                   0 PC = xx IFID_INSTN_VAL = xxxxxxxx ALU_input1 =          x ALU_input2 =          x final_output =          x
File Read
                   1 PC = 00 IFID_INSTN_VAL = xxxxxxxx ALU_input1 =          x ALU_input2 =          x final_output =          x
Initial value of registers:
regfile[           0] =          0
regfile[           1] =         10
regfile[           2] =          0
regfile[           3] =          5
regfile[           4] =          0
regfile[           5] =          0
regfile[           6] =          3
regfile[           7] =          0
regfile[           8] =          0
regfile[           9] =          0
                  20 PC = 01 IFID_INSTN_VAL = xxxxxxxx ALU_input1 =          x ALU_input2 =          x final_output =          x
                  40 PC = 02 IFID_INSTN_VAL = 00231021 ALU_input1 =          x ALU_input2 =          x final_output =          x
                  60 PC = 03 IFID_INSTN_VAL = 00454022 ALU_input1 =         10 ALU_input2 =          5 final_output =          x
                  80 PC = 04 IFID_INSTN_VAL = 00c24823 ALU_input1 =          5 ALU_input2 =          0 final_output =          5
                 100 PC = 05 IFID_INSTN_VAL = 00a62820 ALU_input1 =          3 ALU_input2 =          5 final_output =          0
                 120 PC = 06 IFID_INSTN_VAL = 00662021 ALU_input1 =          0 ALU_input2 =          3 final_output =          7
                 140 PC = 07 IFID_INSTN_VAL = 10a60000 ALU_input1 =          5 ALU_input2 =          3 final_output =          3
                 160 PC = 08 IFID_INSTN_VAL = xxxxxxxx ALU_input1 =          3 ALU_input2 =          3 final_output =          2
                 165 PC control has been updated successfully
                 180 PC = 00 IFID_INSTN_VAL = xxxxxxxx ALU_input1 =          x ALU_input2 =          x final_output =          0
```

The gtkwave output of our processor simulation can found below-



(note that the output of the instructions after the MEM stage are saved in final_postwb_mux_output)