A REPORT ON

Design of pipelined MIPS datapath with Hazards

By

Nihal Singh **2017A3PS1159P**
Vishnu Venkatesh **2017A3PS1154P**

COMPUTER ARCHITECTURE
CS F342

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI (Rajasthan)

(June, 2020)

TASK: To design a 5-stage pipelined MIPS datapath that detects and remedies hazards for the following set of instructions: -

sub $2, $1, $3

and $8, $2, $5

or $9, $6, $2

add $5, $5, $6

sub $4, $3, $6

beq $5, $6

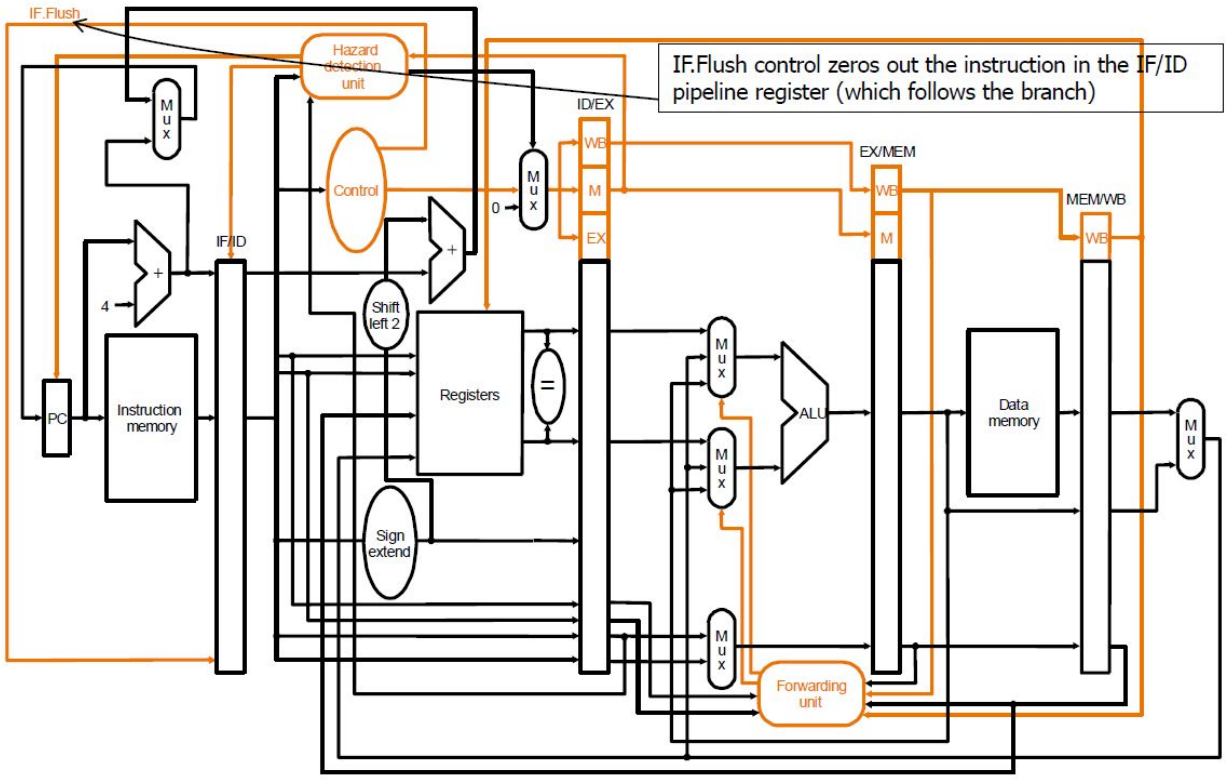
Refer the below excel snippet for the stage-wise breakdown of the above instructions in the pipeline with some additional comments.

sub $2, $1, $3		IF	ID	EX	MEM	WB						forwarding from EX/MEM reg
and $8, $2, $5			IF	ID	EX	MEM	WB					forwarding to ALU ips thru MUX
or $9, $6, $2				IF	ID	EX	MEM	WB				forwarding from MEM/WB reg
add $5, $5, $6					IF	ID	EX	MEM	WB			forwarding to ALU ips thru MUX
sub $4, $3, $6						IF	ID	EX	MEM	WB		
beq $5, $6							IF	IF	ID	EX	MEM	WB	stall one clock cycle to wait for add to WB to register before branch
												forwarding
												stall

These instructions are stored in a text file in hexadecimal (given below)

22
10
23
00
24
40
45
00
25
48
C2
00
20
28
A6

00
22
10
66
00
00
00
A6
10



(Reference image for processor)

The major components of this datapath are:

1. ALU
2. Data memory
3. Instruction Memory
4. PC

5. Register File
6. Sign Extending unit
7. Left shifting unit
8. Adder for calculating the value of PC+4
9. Adder for calculating the value of the branch, relative to PC
10. 6 multiplexors
    a. One give input to PC, either PC+4 or branch address based on branching condition
    b. Two multiplexors at each input of the ALU, as part of the forwarding unit
    c. One mux as the input to one of the muxes giving output to the ALU, which chooses between giving rt and giving the sign-extended immediate value as input to the ALU
    d. One mux to write either data from data memory or from ALU result into register file
    e. One 5-bit mux to choose between rt and rd for destination register (for writing into register file)
11. Main control unit
12. ALU control unit
13. Pipe stage registers

The design of the processor is modular - each component has been designed as an individual module and the top level module only instantiates the modules. The processor datapath is implemented using the data flow approach creating and interlinking the modules. Comments have been added wherever deemed necessary, and smaller modules have been incorporating in the main_pipeline.v itself. All instructions were coded in the standard 32 bits MIPS format in a text file in hex as shown above.

Design of ALU

1. Implemented with behavioural modelling.
2. Operations supported: ADD, SUB, AND, OR
3. Outputs: Result of operation, and a ‘zero’ signal if ALU result is 0.

Design of Data Memory

1. Reading has been implemented with dataflow modelling; writing has been implemented with behavioural modelling.
2. If read_enable = 1, then the contents of the memory (declared as an array of registers in the module) at the given address are returned.
3. If write_enable=1, then data is written into the memory (the same array of registers) at the given address.

Design of Instruction Memory

1. Implemented with dataflow modelling.
2. At reset, this module reads from a text file using *$readmemh* instruction.
3. A 32-bit output returns instruction present at given address input.

Design of PC

1. Implemented with behavioural modelling.
2. Separate module that just behaves as a register; gives whatever input is there as output on the next clock cycle.

Design of register file

1. Implemented with behavioural modelling.
2. At reset, the contents of a text file are read to put content into the registers.
3. At positive edge of the clock, the data to be written is written into the rd register if write_enable=1.
4. The contents of register rs is given at one output.
5. The contents of register rt is given at the other output.

Design of Sign Extending unit

1. Implemented with dataflow modelling.
2. If the leftmost bit is 1, then a hex value of 0xFFFF is appended to the left of the 16-bit immediate value.
3. If the leftmost bit is 0, then a hex value of 0x0000 is appended to the left of the 16-bit immediate value.

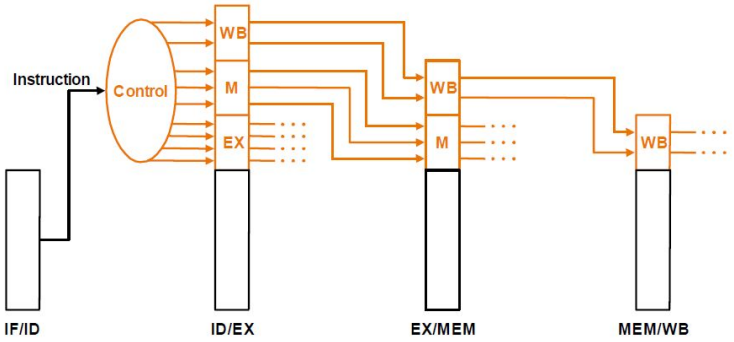
Design of Left shifting unit

1. Implemented with behavioural modelling.

2. Combinational circuit that shifts the 32-bit input by two bits to the left, and gives a 32-bit output.

Design of main control

The control signals for each pipe stage - fixed for a given instruction - were hardcoded in a module (instr_dec), and passed on between pipeline registers in the manner shown below:

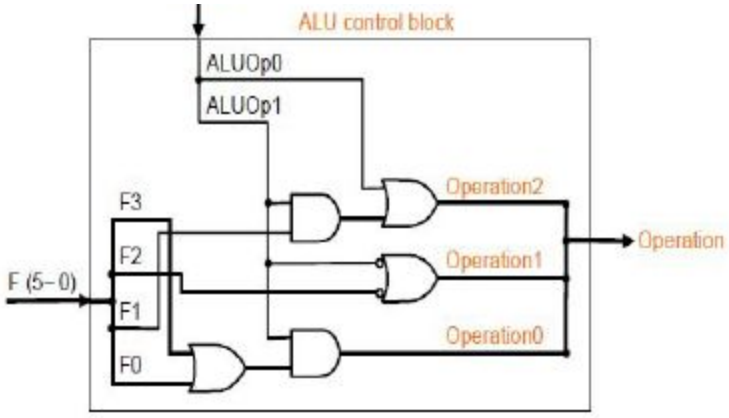


Design of ALU Control

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

The ALU control unit was written according to the truth table on the left.

Ternary operators were used to decide values of the three operation bits individually.



ALU control

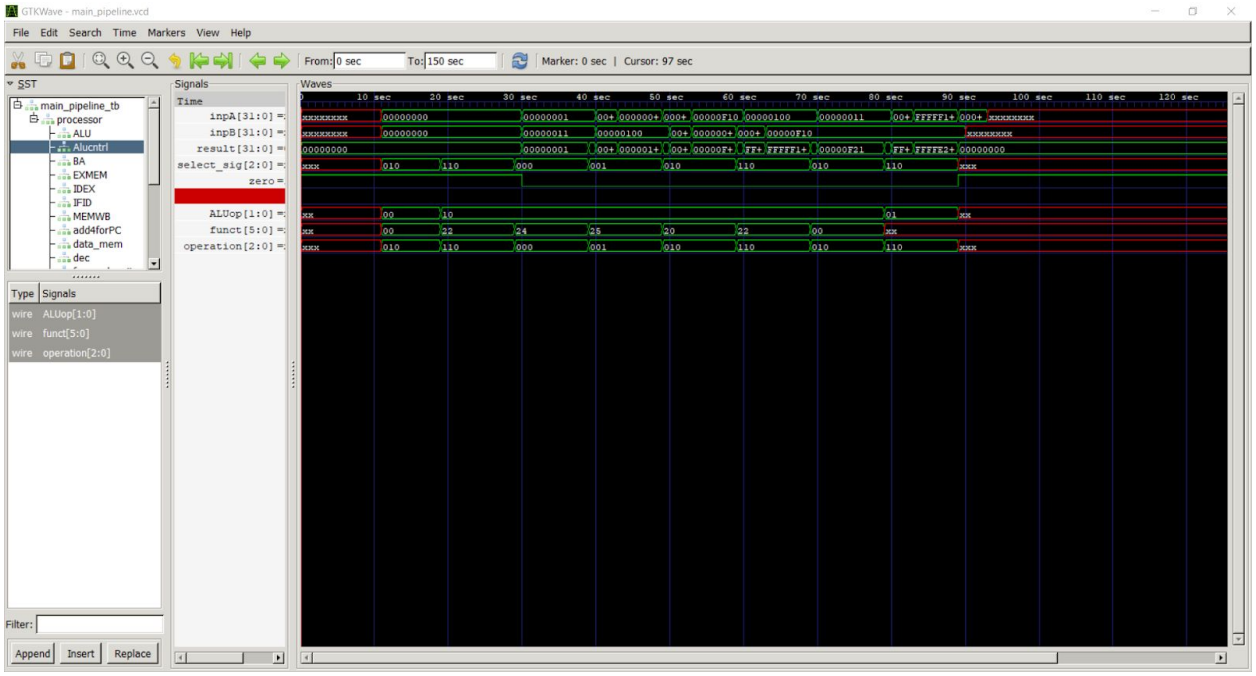
Design of pipe stage registers

Each pipe stage register was just designed to pass on data after a small delay - which is introduced so that the register passes on data on the next cycle only, without affecting the values of the current cycle.

All pipe stage register modules were implemented using behavioural modelling.

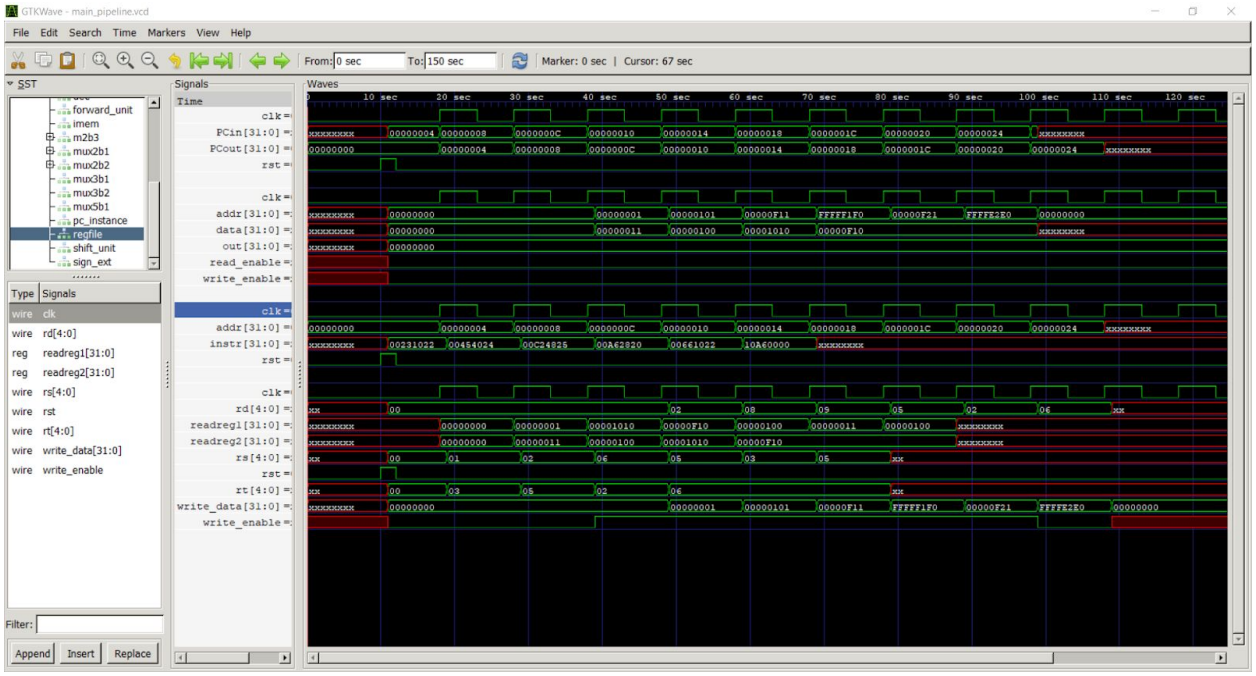
(Additionally, in the main_pipeline.v module, comments have been used to demarcate the difference between the instantiations of different stages in the pipeline.)

Simulation Results



The above screenshot shows simulation results for ALU and ALU Control (separated by a blank signal).

The “operation” signal (output of ALU control unit) and “select_sig” (input of ALU unit) are identical. This signal controls the final operation that the ALU performs. “inpA” and “inpB” are inputs to the ALU, and “result” contains the final value calculated based on the “operation” signal.



The above screenshot shows four different modules separated by blank signals - PC, instruction memory, data memory, and register file.

The input and output of the PC module is the address with which an instruction is read from the instruction memory, delayed by one clock pulse.

The data memory is not used actively for this set of instructions, yet it is included as part of the datapath.

The instruction memory outputs a 32-bit instruction addressed by the PC module's output value. The instructions are identical to the instructions provided near the start of this report, concatenated in little-endian format.

The register file gives outputs "readreg1" and "readreg2" based on the contents of "rs" and "rt" input signals. The single time unit delay between them comes from the delay included in the pipeline registers, from which the inputs "rs" and "rt" are taken. This is done so that the input values do not change at the exact same moment that outputs have to be read, which is the positive clock edge. "write_data" is written into the register numbered "rd" in the register file when write_enable=1.