COL216 Assignment 9

Vishal Bindal
2018CS50425

Chirag Mohapatra
2018CS50403

To Run:

(A) To give instructions in form of MIPS assembly code:

1. Set syntax_input = true in main.cpp (It has been set to true already while submitting)
2. Run '`make`' to create an executable 'main'
3. Enter the required instructions in 'input.txt' (one test case already filled in while submitting). The registers could be like $8, $9, etc or $t1, $t2, etc
4. Run '`./main`'

(B) To give instructions in form of 32-bit instructions directly:

1. Set syntax_input = false in main.cpp
2. Run '`make`' to create an executable 'main'
3. Enter the required instructions in 'binary_input.txt' (one test case already filled in while submitting)
4. Run '`./main`'

Output:

1. The state for each clock cycle is printed in the console. For each clock cycle, following is shown:
    a. Clock cycle no.
    b. Current instruction in each stage of the pipeline (Here instruction nos. refer to the line no in input_clean.txt, read more below)
    c. Details about any read/write from data memory (Memory address and value)
    d. Details about any write to the register file (Register no. and value)
    e. The status of the register file, i.e. values in each register
    f. Details about forwarding of any data (Reg whose data forwarded)
2. At the end, the following 2 statistics are printed:
    a. Total clock cycles
    b. IPC

Documentation

Differences from assignment 8

1. We implemented forwarding to resolve data hazards where possible in order to speed up the processor
2. Forwarding
    a. In the function getData() in Instruction.cpp, we check if the register which we want to read data from (is **rs** or **rt** of current instruction) is equal to the (a) **rd** of an add/sub/sll/srl or (b) **rt** of an lw instruction already in the pipeline. If so, then it is a data hazard.
    b. Then we check if the data is okay to forward (i.e. if the prev instruction has passed the EX stage in case (a), or has passed the MEM stage in case (b)). using the function okay_to_forward()
    c. If yes then we forward the data using forward_data() and there is no stall, else we stall for a cycle.
3. We also print to the console, about any data forwarded during the cycle.

Implementation structure

1. If the input is in the form of assembly code, we first 'clean' the file by removing unnecessary spaces and newlines. We parse instructions into tokens (delimited by space, bracket, comma) and create a 'clean' input file input_clean.txt. **All the instruction nos. in the output refer to corresponding line nos. in input_clean.txt**

2. We then convert each instruction into 32-bit binary instructions, and output them to 'binary_input.txt'. We also take care of labels for jump/branch instructions, by mapping each label to corresponding instruction no first and then encoding the required jump/branch value in the instruction.

3. We have represented each instruction as an object of a class 'Instruction'.
    a. This object stores the instruction string, rd/rs/rt register nos, values read while decoding, result of alu operation, etc. Thus it acts as a proxy for the pipeline registers, while making it easy to organise and print data during the simulation.
    b. This object also has functions: instruction_decode(), execute(), memory_access() and write_back() which are run in the corresponding stage of the pipeline the instruction is in. Each of these functions is implemented in the form of switch-case, with the action depending on the type of instruction.

4. We have implemented stalling by creating a function stall(), which stalls instruction >=i (the ith instruction and instructions after it) for n cycles. This function is called in case of

data hazards and for branch hazards accordingly, during the instruction decode stage. Data hazards is checked using a function ‘is_data_hazard()’, which checks register nos of instructions already in the pipeline.

5. For each clock cycle, a new instruction is fetched (if not stalled), and the instructions in the pipeline run their corresponding functions (depending on the pipeline stage they are in) (if not stalled)

Test cases and comparison with assignment 8 processor:

For all test cases, we have assumed the following initialisation of the register file:
reg_array[0] = 2
reg_array[1] = 1
reg_array[29] = 4095
Others: 0

S. No	Instructions	Details, Final register file	No of clock cycles without forwarding	No of clock cycles With forwarding

1	sw $0 , 1024($1)
	
	add $0,$0,$0
	
	sub $1 , $0, $1
	
	lw $2, 1022($1)
		4 3 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4095 0 0	12	8

2	bgtz $0, second
	
	first:
	
	add $1, $1, $1
	
	add $2, $1, $0
	
	j end
	
	second:
	
	bgtz $0, first
		2 2 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4095 0 0	15	13

	end:			
3	sw $0 , 1024($1) sll $0,$0,2 srl $0,$0,1 sub $1 , $0, $1 lw $2, 1022($1)	4 3 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4095 0 0	15	9

4	sw $0 , 1000($0)
	sw $1 , 1001($0)
	add $2, $1, $0
	sub $3, $1, $0
	sw $2 , 1003($1)
	sw $3 , 1004($1)
	lw $4 , 1000($0)
	lw $5 , 1001($0)
	lw $6 , 1002($0)
	lw $7 , 1003($0)
	add $6, $6, $7
	add $5, $5, $6
	add $4, $5, $4
		2 1 3 -1 5 3 2 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4095 0 0	24	18

5	add $2 , $0 , $1 sub $3 , $2 , $0	4 1 3 1 5 4 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4095 0 0	22	13

	sw $2 , 1024($1) sw $3 , 1024($0) add $0 , $0 , $0 add $4 , $0 , $3 sw $4 , 1024($0) sub $5 , $4 , $3 sll $6 , $5 , 2			
6	sll $4 , $0 , 1 sll $5 , $0 , 2 jal one add $17 , $2 , $16 j end one: sub $29 , $29 , $0 sw $31 , 1($29) sw $4 , 0($29) bgtz $4 , two add $29 , $29 , $0 jr $31 two: sub $4 , $4 , $1 jal one lw $4 , 0($29)	2 1 32 0 4 8 0 0 0 0 0 0 0 0 0 0 0 32 0 0 0 0 0 0 0 0 0 0 4095 0 3	89	79

	lw $31 , 1($29) add $29 , $29 , $0 add $2 , $2 , $5 jr $31 end:			
7	sll $4 , $0 , 3 jal acc add $17 , $2 , $16 j end acc: sub $sp , $sp , $0 sw $ra , 1($sp) sw $4 , 0($sp) bgtz $4 , one add $sp , $sp , $0 jr $ra one: sub $4 , $4 , $1 jal acc lw $4 , 0($sp) lw $ra , 1($sp) add $sp , $sp , $0 add $2 , $2 , $4	2 1 136 0 16 0 0 0 0 0 0 0 0 0 0 0 0 136 0 0 0 0 0 0 0 0 0 0 0 0 4095 0 2	292	258

	jr $ra end:			

As expected no of clock cycles decrease with forwarding.