EE 309: Microprocessors Course Project II
Pipeline RISC-IITB Architecture
Datapath Description

Team Details:
Krishna Reddy (130020118)
Kalpesh Patil (130040019)
Mehul Shah (130020090)
Ankit Patil (13D070018)

Hazard Detection Unit

Data Hazards

1. Source-destination address overlap in consecutive instructions

It is solved by implementing forwarding unit. The hazard is caused when any of the source address of the next instruction is equal to the destination address of the current instruction.

Example 1)

I1: ADD R1, R2, R3 {Here Destination address = R1}

I2: NAND R4, R1, R5 {Source Address = R1,R5}

Remedy: For this type of hazard data from output ALU is forwarded to input of ALU through forward unit.

Example 2)

I1: LW R1, R2, Imm {Destination address is R1}

I2: ADD R3, R1, R2 {Source addresses are R1,R2}

Remedy: For this type of hazard we need to insert stall for one cycle and forward output of Memory to the input of ALU. To avoid fetching of instruction we are keeping a separate bit called 'load_check' in F/D register, which will be used for inserting stall in case of data hazard caused due to load type instruction. PC_en will be cleared to disable PC so that we do not fetch a new instruction. Also NOP will be inserted in the next cycle

2. Data Memory Address Computation Hazard

I1: LW R1, Imm1

I2: LM R1, Imm2

We get the value of R1 only after memory stage; but next instruction requires value of R1 (address for Data Memory) for implementation, which causes hazard.

Remedy: Therefore we forward edb_out (output of data memory) to the T3 register in E/M which provides the address to data memory in next cycle.

Control Hazards

1. Definite Branching instructions (JAL and JLR)

New PC address is calculated after decode stage for JAL instruction and after Register Read stage for JLR instructions. Therefore first interface (F/D) will be flushed for JAL and first two (F/D and D/R) will always be flushed for JLR due to definiteness of the branch

2. Conditional Branch Instruction (BEQ)

Branch is predicted using the prediction unit. The prediction unit is implementing a two state fsm and takes decision based on history bit (1 bit). If prediction is wrong, history bit is updated. The actual decision is taken after execution stage, which will flush the previously fetched instructions if prediction is wrong; otherwise instructions continue to execute.

3. Instructions changing value of R7

If R7 (which also contains PC) is destination register for any instruction, new PC has to be updated using new value of R7. This is done in execution state where R7 value is recomputed. It also requires flushing of few instructions which were fetched using old PC.

4. Carrying PC in Interface Registers

PC is carried forward till execution stage because BEQ instruction requires the PC value of that instruction after execution stage for the computation of new PC value; if prediction is wrong.

Structural Hazards

1. LM, SM instructions

To identify registers to be loaded/stored from/to consecutive addresses in data memory, we are using Priority Encoder. Till the ‘V_bit’ of priority encoder becomes zero (no. of registers to be loaded), we insert stalls (disable interface registers and PC till the end of LM/SM instructions).

2. Updating R7 from PC

We have implemented a Register File with two data_write inputs. First input will correspond to normal register write from write back stage. The other input is used for updating R7 with the PC values from PC_F/D stored in the F/D. This additional data write input was introduced to avoid conflict between data write coming from write_back stage.

Other Datapath Components

1. Interface Registers

a. F/D

PC_F/D (16bit)	IR_F/D (16bit)	Load_check (1bit)	nxt_nop(1bit)

b. D/R

PC_F/D (16bit)	IR_F/D (16bit)

c. R/E

PC_R/E (16bit)	IR_R/E (16bit)	T1 (16bit)	T2 (16 bit)

T1: Output of RF1 T2: Output of RF2 SF_R/E: Output of Shifter

d. E/M

IR_E/M(16bit)	T2_dash(16bit)	T3(16bit)

T2_dash: T2 carried forward T3: ALU output SF_E/M: Output of Shifter carried

e. M/W

IR_M/W (16bit)	MDR (16bit)	T4 (16bit)

MDR: Data output of memory T4: T3 carried forward SF_M/W: Output of Shifter carried forward

2. Memory

a. Instruction memory
b. Data memory

3. PC

4. Register File

Asynchronous read and synchronous write memory is implemented. If reset all the registers are cleared. Two data_write inputs are provided to avoid hazard caused due to updating R7 from PC as explained earlier.

5. Sign Extenders

1)S6->16: It takes 6 bits imm. data from IR and pads it with zeroes in MSB to make the output 16 bits.
2) S_{97}9->16: It takes 9 bits imm. data from IR and shifts them to MSB. It then pads it with zeroes in LSB to make the output 16 bits.
3) S_{79}9->16: It takes 7 bits imm. data from IR and shifts them to MSB. It then pads it with zeroes in LSB to make the output 16 bits.

6. ALU

a. ALU1: Execution stage ALU
b. ALU2: ALU used for computing PC value (for Jumps)
c. incrementer1(+1) : PC = PC +1
d. incrementer2(+1) : T2_dash conditional incrementer used for traversing through memory (used in LM and SM instructions)

7.Priority Encoder:

It is implemented for LM/SM instruction. The output of the priority encoder will be the address of the register to be loaded/stored in that cycle. We will load the immediate 8 bits into 8 to 3 priority encoder. Address of the first high bit will be the output of the priority encoder. We store/load corresponding register. Then we set that bit to zero and recompute the priority encoder output in the next cycle. This will be continued till all bits are zero which will make 'V_bit' low and stop the implementation of LM/SM

8. Muxes

a. **Address Muxes:** Input addresses for register file
i. a1_mux ii)a2_mux iii)a3_mux
b. **ALU_srcA and ALU_srcB:** Inputs for ALU1
c. **PC_ALUsrcA and PC_ALUsrcB:** Inputs for ALU2
(Determining PC offset for jump instructions)
d. **PC_mux:** For computing new PC value
e. **WB_mux:** Choose data to be written back into register file (from memory or ALU output)
f. **Forward_muxA and Forward_muxB:** Forwarding unit muxes
g **T3_mux:**

9. Control Logic Blocks

a. Control_RR
b. Control_M
c. Control_WB
d. Control_D