Hazards

Detection and Removal

# Timing Diagram



# Arithmetic Instructions

## Arithmetic Instructions Not Having R7 as Destination

* Write PC+1 to R7 propagated in registers. Set a bit for this in pipeline registers.

# Logic at Instruction Decode State [Forwarding Logic]

* If [RR\_EX](AR 1/2) = [EX\_MM](AR3) → ALU( 1/2) <= [Ex-MM]{Write Value}
  + Source bits same as WB selection bits. Use them
  + [Ex\_MM]{Write Value} := (Arith\_Instruction -> ALU\_Out)
  + [Ex\_MM]{Write Value} := (LHI -> LS\_PC)
  + [Ex\_MM]{Write Value} := (LLI -> SE)
  + [Ex\_MM]{Write Value} := (JAL,JLR -> PC+1)
* If [RR\_EX](AR 1/2) = [MM\_WB](AR3) → ALU( 1/2) <= [MM-WB] {Write Value}
  + (SM and SW) Invalid AR3.. LM and LW output destination select is Mem\_out. Can be esily neglected.
* If [RR\_EX](AR 1/2) = [WBT](AR3) → ALU( 1/2) <= [WBT] {Write Value}
* If [RR\_EX](AR 1/2) = R7 ALU(1/2) <= [RR\_EX](PC)

# Logic at Write-Back State [Pipeline Flush Logic]

* All previous conditional instructions are assumed to be taken. Conditional Arithmetic too. If condition becomes false at any write-back state and it is in dependency list then flush pipeline. As wrong data may have forwarded for preceding instructions.
* Write PC with (PC+1) in [MRW-WB]
* Dependency List →
  + AR3 of instruction in write-back [MWR-WB] state is same as AR(1/2) of any instruction in [ID–RR, RR–Ex, Ex–MRW].

## Arithmetic Instructions with R7 as Destination

* Clear R7 write bit in pipeline registers.

# Unconditional Instructions

* Update PC in Ex state. i.e. ALU-Out → PC, [Ex-MRW] {Done by hazard detection unit}
* Clear [IF-ID](Controls), [ID-RR](Controls)

# Conditional Instructions

* Treat them as unconditional instructions. (taken)
* If condition becomes false write PC with (PC+1) in [MRW-WB]. No Need to check dependency here. So reset a bit corresponding to this. {Flush Pipeline}

# Load Instructions

# Load With Destination not R7

* To match with hazard logic float one bubble only for immediate dependency. If [ID-RR](Load\_Ins) = 1 and [IF-ID](AR1/2) = [ID-RR](AR3) → Clear [IF-ID](Controls), PC-Ena
* Data taken from MEM-OUT

# Load With Destination R7

* Clear R7 write bit.
* Update PC in MRW step. AR1 → Mem → PC, [MRW-WB].
* Clear [IF-ID](Controls), [ID-RR](Controls), [RR-Ex](Controls)

# Branch and Jump Instructions

## JLR

* RF(AR3) is going to hold PC+1, so to match with Hazard Logic PC+1 → [RR-Ex]
* Jump address is available in RR state. RF-Out → PC
* Clear [IF-ID](Controls)and [ID-RR]

## JAL

* Dedicated adder in ID state.
* Clear [IF-ID](Controls)
* Clear given by decoder and passed through register to clear

## BEQ

* Act as not taken branch
* If condition becomes true in WB state. Flush pipeline.

# LM

* Halt states at Execution state.
* As soon as [RR-Ex](P Encoder In) filled with registers’ numbers (non zero), it will halt state advancement by clearing enable for previous pipeline registers and PC.
* When PE gives invalid next remove lock state.
* Must have bit to get AR3 from PE
* IF R7 is also involved Load Hazard will take care of that
* Invalid next when all are zero. Invalid next Run and clear RR\_EX reg. To make it bubble.
* LM bit activates the priority encoder at RR-Ex stage
* In LM DO1\_en will be connected to the mux before DO1 register in RR\_EX .

### SM

* Again halt at execution state.
* As soon as [RR-Ex](P Encoder In) filled with registers’ numbers (non zero), it will halt state advancement by clearing enable for previous pipeline registers and PC.
* Must have bit to merge two states and by pass register.
* Invalid next when all are zero. Invalid next Run and clear RR\_EX reg. To make it bubble.
* SM bit activates the priority encoder at ID-RR stage.
* Valid should cleared so that the hazard logic doesn’t affect the mux steering inputs between DO2 and the forward logic.
* RR-Ex will have three special clears 1)control clear 2) valid clear 3) register clear

Priority for checking forward Logic:

1) R7

2) EX\_MM

3) MM\_WB

4) WBT

R7 clear when A3 = “111” is done in WB hazard control

IN WB Hazard :

User Flags used for checking condition for conditional arithmetic

ALU Flags are used for checking condition for BEQ

# Hazard Detection Unit Functions