**Pipeline stages design**

**I. Pipeline registers details**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Size** | **Inputs** | **Read** | **Write** |
| **Fetch/Decode** | 48 | Next instruction address (32)  Opcode-Rs-Rd-SHMNT (16) | +ve edge | -ve edge |
| **Decode/Execute** | 72 | Imm value (16)  SHMNT (5)  Source (16)  Destination (16)  Rs (3)  Rd (3)  Control signals (13) | +ve edge | -ve edge |
| **Execute/Memory** | 57 | ALU result (16)  Source (16)  Destination (16)  Rd (3)  Control signals (6) | +ve edge | -ve edge |
| **Memory/Write Back** | 38 | ALU result (16)  Data from memory (16)  Rd (3)  Control signals (3) | +ve edge | -ve edge |

**II. Types of hazards**

**1.Data Hazards**

Handling the data hazard of ALU instructions is in the decode stage to check on the source of the current instruction if it is the destination of the 2 last instructions. Hence, if there is a hazard between 2 ALU instructions it could be solved by giving all this data to the forwarding unit. the output of the FU is control signals to choose the correct input data. (We will design full forwarding Alu-Alu and mem-Alu) also this FU will reduce the number of wasted cycles to one instead of two if the instruction before the instruction before the instruction in decode is a load instruction and there is a data hazard between both. (Using mem-Alu forwarding)

Handling Load use case: There is a HDU for the load use case that will be in the decoding stage "after fetching" to stall on the following conditions:

If(ID/EX.MemRead and

(not ID/EX.pop) and

(ID/EX.RdestAddress == IF/ID.RsrcAddress))

Stall the pipe

**2.Control Hazards**

we assume a not-taken prediction. To handle the control hazards, we calculate the branch result in the decoding stage to reduce the number of wasted cycles to one instead of two by adding additional HW to calculate the PC and flag register and change the pc value if the branch result is true and stall one cycle as there is unwanted instruction fetched.

**III.** **Data Forwarding**

Full forwarding: when the source of the current instruction is a destination at the previous(one or two) instruction

**III.** **Static Branch Prediction**

we assume a not-taken prediction, while we calculate the branch result, we start fetching the instruction only after it. then when it's already calculated at decode stage (We added an additional HW to calculate the PC and flag register) so to not waste an extra cycle for every branch instruction ,then we will flush the instruction we fetched before and change the pc value if the branch result is true by sending a signal to the pc mux in fetch stage.