Hazard Detection

As we have mentioned that we will fetch packet of instructions and execute those using concept of parallelism. The packet consists of only two instructions. These instructions may depend on each other and this dependency will cause a problem called hazards.

Hazards Types

- Structural Hazard
- Control Hazard
- Data Hazard

Structural Hazard

Structural hazard occurs in the following situation: -

- Instruction Fetch is conflicting with data memory access.
    - **Solution** is to use two separated memory (Data and Instruction).
- Register File is accessed by the instruction in the **decode** stage (reading) and at the same time other instruction in the **write-back** stage.
    - **Solution** is to ensure that writing is done in the falling edge and reading is done in the rising edge.
- Packets is executed in parallel and no sufficient resources
    - **Solution** is to duplicate the resources. We will have two ALUs, two MAR, two MDR, Register File has two signals **WB** (WB1 for the first instruction in the packet and WB2 for the second instruction in the packet).

Data Hazards Types

- Data Inner Hazard
- Data Outer Hazard

Data Inner Hazard

This type of hazards occurs when the packet itself is dependent. For example, if I have a load in the first instruction that uses some registers and the second instruction is using the same register to do some logic in the program. Assume code has only three instructions (**Inst1 to Inst3**) and the first, second cause data hazard because of dependency.

Solution is to stall the pipeline until this instruction will be written back in the register file after doing the logic of first instruction then start fetch a new packet and this new packet will contains the following **Inst2**--**Inst3.**

How this solution actually works behind the scenes?!

The Hazard Detection Unit **(HDU)** takes inputs from the previous, current packet. The inputs are the **IR** of the first packet instructions (Two IR) and the **IR** of the second packet instructions (Two IR). **Write after write may also make conflicts in the register.**

check_one = (DEC_Rsrc2 = DEC_Rdst1)
check_waw = (DEC_Rdst1 = DEC_Rdst2)
stall_long = check_one or check_waw

stall_long is responsible for making the first instruction in the packet resume executing (only it) for 3 clock cycles using crawling **stall_long** in the pipeline and take the result of it in the last buffer. How! By disabling the (IR Register) which means that we add to the pipeline **NOP** operations until the desired instruction finished and disabling the **PC** to keep the next instruction but we need after that to decrement it by 1 to fetch the correct packet.

Rst_IR = !(EXE_stall_long OR MEM_stall_long OR WB_stall_long)

enable_PC = !(EXE_stall_long OR MEM_stall_long)

pc = pc – 1 when WB_stall_long.

Then our new fetch will be correct isa.

Outer Hazard

This type of hazards occurs when an instruction (or more) in the packet depend on previous packet (**Load-Use/POP**). For example we've 4 instruction **Inst1 to Inst4**. Assume Inst2 is LDD Rdst, Rsrc and inst3 need to use Rdst. We've 4 cases **(Inst3 and Inst2), (Inst3 and Inst1), (Inst4 and Inst1), (Inst4 and Inst2)**

Solution is to stall the pipeline one cycle (stall bit is added in the control word).

How to stall the pipeline?!

- Disabling The **IR** & **PC** (Don't take the new instruction that the **PC** stored its address and don't increment **PC**).
- Reset The Buffer between the decoder stage and execution stage (act as **NOP**).

Stall Logic

```
stall = EXE_Memory_Read AND
                    ( EXE_Rdst1 = DEC_Rsrc1 )
                 OR ( EXE_Rdst1 = DEC_Rsrc2 )
                 OR ( EXE_Rdst2 = DEC_Rsrc1 )
                 OR ( EXE_Rdst2 = DEC_Rsrc2 )
```

Control Hazard

Control hazard occurs in the following situation:-

- **Branch Taken (Jump)**
  - Static prediction is not to take the branch and continue the program **(NOT-TAKEN).**
  - We know that the instruction is branch in the decode stage and we raise one bit called **branch_taken** if taken or not.
  - In the execution if the branch_taken is raised then we need to flush else continue the programs (no problem).
- **Load Immediate**
  - Some instructions take a load value from the next instruction so I've to take this value and put it in the control signal then flushing the instruction (actually the instruction is a value not an instruction).

Solution is to flush the new instruction or (value).

How to flush the instruction or value?!

- Raise signal **(flush)** (acts as **NOP**), we may need to RST the IR (explained in the branch when occupies the 1st place in the packet).

There are two cases

- The branch instruction is in the first instruction of the packet
- The branch instruction is in the second instruction of the packet

Each case need a different handler. For the case #1 assume we have this code

- JZ Rdst
- Add R1, R2

So in the above example we need to flush the 2nd instruction in the packet and the next packet once we knew that the branch is taken.

For case #2

- Add R1, R2
- JZ Rds

But in the above example we've to complete the 1st instruction in the packet till the end then we back to case #1.

stall_long = (DEC_Inst2 = Branch)

Rst_IR = !(EXE_stall_long OR MEM_stall_long OR WB_stall_long)

enable_PC = !(EXE_stall_long OR MEM_stall_long)

pc = pc – 1 when WB_stall_long.

Flush Logic

Flush = immediate_load OR branch_taken.
PC = effective_address when branch_taken.
Rst_IR = branch_taken (flush second packet)

Immediate Load

There are two cases

- The load instruction is in the first instruction of the packet
- The load instruction is in the second instruction of the packet

We always need the load instruction (**LDM**) to be the first in the packet, so if it occupies in the second instruction in the packet we will make the slight same logic of the **data-inner-hazard**.

Each case need a different handler. For the case #1 assume we have this code

- LDM R5, 10
- 10

In the above example we raise only the signal of flush that will make the mux pass zeros to the alu.

For the case #2 assume we have this code

- Add R1 , R2
- LDM R5, 10

stall_long = (DEC_INST2 = LDM OR DEC_INST2 = SHL OR DEC_INST2 = SHR)

Rst_IR = !(EXE_stall_long OR MEM_stall_long OR WB_stall_long)

enable_PC = !(EXE_stall_long OR MEM_stall_long)

pc = pc – 1 when WB_stall_long.