**Hazard Detection Unit**

Stall\_bit\_1

1-bit Reg

Opcode\_F

Rdst\_D\_code

Rdst\_F\_code

Branch-Hazard

Prediction\_bit

Rsrc1\_D\_code

Opcode\_D

Rsrc1\_E\_code

Opcode\_E

Rsrc1\_M\_code

Opcode\_M

Rdst\_M\_code

Rdst\_E\_code

Long-Fetch-Hazard

clr

Stall\_bit\_2

1-bit Reg

Opcode\_F

Fetch-Hazard

Previous\_Stall

1-bit Reg

Swap-Hazard

Opcode\_D

Load-use-Hazard

Stall\_bit\_3

Opcode\_D

Rsrc1\_D\_code

Opcode\_E

Rsrc2\_D\_code

Stall\_bit\_6

Rdst\_E\_code

Opcode\_D

Opcode\_E

Swap-use-Hazard

Stall\_bit\_7

Opcode\_M

Rsrc2\_D\_code

Rsrc1\_D\_code

Rdst\_M\_code

Rdst\_E\_code

Opcode\_E

Wrong-prediction

Stall\_bit\_4

Prediction-bit

ZF

Wrong-prediction-bit

Stall\_bit\_5

1-bit Reg

RET-RTI-Reset-INT

Opcode\_MW

Opcode\_F

INT\_F

Falling-edge checker

INT\_MW

Reset\_MW

Reset\_F

Load\_ret\_PC

Hazard-Detection

Unit

Stall\_bit\_3

Stall\_bit\_1

not

PC\_Write

Stall\_bit\_6

Stall\_bit\_5

Stall\_bit\_7

Stall\_bit\_2

Stall\_bit\_1

Control\_Unit\_Mux

Stall\_bit\_4

Stall\_bit\_3

Stall\_bit\_6

Stall\_bit\_5

Stall\_bit\_7

**Branch-Hazard unit:**

As we predict in fetch stage so if we have a jz instruction with Prediction\_bit = 1 “predict taken”, a jmp instruction or a call instruction so we need to pass Rdst to the PC.

This will cause hazard in some cases if Rdst is not ready in the register file.

Case 1:

* Add R3,R1,R2
* Jz R3
* Instr. “if taken”

When “jz” is in fetch stage, “add” will be in decode stage so Rdst will not be calculated yet.

We need to stall once then forward Rdst from execute stage to the PC.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Add R3,R1,R2 | F | D | E | M | W |  |  |  |
| Jz R3 |  | F | stall | stall | stall | stall |  |  |
| Jz R3 |  |  | F | D | E | M | W |  |
| Instr. |  |  |  | F | D | E | M | W |

Case 2:

* LDD R3,Imm
* Instr.
* Jz R3
* Instr. “if taken”

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LDD R3,Imm | F | D | E | M | W |  |  |  |  |
| Instr. |  | F | D | E | M | W |  |  |  |
| Jz R3 |  |  | F | stall | stall | stall | stall |  |  |
| Jz R3 |  |  |  | F | D | E | M | W |  |
| Instr. |  |  |  |  | F | D | E | M | W |

Case 3:

* LDD R3,Imm
* Jz R3
* Instr. “if taken”

Same as case 2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LDD R3,Imm | F | D | E | M | W |  |  |  |  |
| Jz R3 |  | F | stall | stall | stall | stall |  |  |  |
| Jz R3 |  |  | F | stall | stall | stall | stall |  |  |
| Jz R3 |  |  |  | F | D | E | M | W |  |
| Instr. |  |  |  |  | F | D | E | M | W |

Case 4:

* Swap R1,R2
* Jmp R1

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Swap R1,R2 | F | stall | stall | stall | stall |  |  |  |  |  |  |
| Swap R1,R2 |  | F | D | E | M | W |  |  |  |  |  |
| Jmp R1 |  |  | F | stall | stall | stall | stall |  |  |  |  |
| Jmp R1 |  |  |  | F | stall | stall | stall | stall |  |  |  |
| Jmp R1 |  |  |  |  | F | stall | stall | stall | stall |  |  |
| Jmp R1 |  |  |  |  |  | F | D | E | M | W |  |
| Instr. |  |  |  |  |  |  | F | D | E | M | W |

In general: if I have jz with taken pred. , jmp, call, I will stall once in these cases

1. The previous instr. is one-op, two-op, LDM, LDD or POP that has the Rdst as me.
2. The instr. before previous one is LDM, LDD or POP that has the Rdst as me.
3. One of the previous 3 instructions is swap with the Rsrc or Rdst as my Rdst

**Stall\_bit\_1**: stall the whole pipe

**Long-Fetch-Hazard unit:**

There’re some instructions that are 32-bit in size so it can’t be fetched once from the memory.

So what we need is to fetch the first half and to stall this half in the decode till we fetch the second part and start decoding and to make sure that the next half will not cause this stalling even if it seems to be a 32-bit instr. cause in fact it’s not, it’s just the rest of the last instruction.

**Stall\_bit\_2**: stall decode of the next cycle only.

**Load-use-Hazard unit:**

Example:

* LDD R3,Imm
* Add R1,R2,R3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LDD R3,Imm | F | D | E | M | W |  |  |  |  |
| Add R1,R2,R3 |  | F | D | stall | stall | stall |  |  |  |
| Add R1,R2,R3 |  |  | F | D | E | M | W |  |  |

Note: no register because I want to stop fetching next instruction once add get into decode stage to fetch the same instr. again.

**Stall\_bit\_3**: stall the whole pipe

**Wrong-prediction unit:**

It outputs 1-bit “Wrong-prediction-bit” (prediction-bit XNOR ZF)

If there’s a “jz” instruction in execute stage, there are two cases:

* If prediction-bit = zero-flag : Wrong-prediction-bit = 0
* If prediction-bit != zero-flag : Wrong-prediction-bit = 1

**Stall\_bit\_4**: stall decode only.

**RET-RTI-Reset-INT unit:**

If you found RET, RTI instruction or interrupt or reset in fetch stall upcoming instructions till this instruction or this interrupt/reset finishes memory stage so that the new PC is ready.

**Stall\_bit\_5**: stall the whole pipe “not the whole pipe but the upcoming in fetch only”

**Swap-Hazard unit:**

If there’s a swap in decode it stalls it and re-fetches it again.

To make sure that the re-fetched swap will not cause another stall, we keep track of the last stall due to swap, if it was ‘1’ so do not stall.

**Stall\_bit\_6**: stall the whole pipe

**Swap-use-Hazard unit:**

Example:

* Swap R1,R2
* Add R1,R2,R3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Swap R1,R2 | F | stall | Stall | stall | stall |  |  |  |  |
| Swap R1,R2 |  | F | D | E | M | W |  |  |  |
| Add R1,R2,R3 |  |  | F | D | stall | stall | stall |  |  |
| Add R1,R2,R3 |  |  |  | F | D | stall | stall | stall |  |
| Add R1,R2,R3 |  |  |  |  | F | D | E | M | W |

Note: no register because I want to stop fetching next instruction once add get into decode stage to fetch the same instr. again.

**Stall\_bit\_7**: stall the whole pipe

**PC Predictor**

If (Wrong\_Prediction\_bit == 0 and Load\_ret\_PC== 0):

if ( (opcode\_F == jz) and (Prediction\_bit == 1) ) or ( (opcode\_F == jmp) or ( (opcode\_F == call):

PC\_predicted = Rdst\_val

PC\_UnPredicted = PC+1

If (opcode\_F == jz) and (Prediction\_bit == 0):

PC\_predicted = PC+1

PC\_UnPredicted = Rdst\_val

Else If (Wrong\_Prediction\_bit == 1):

PC\_predicted = Unpredictted\_PC\_E

PC\_UnPredicted = PC+1

Else If (Load\_ret\_PC== 1):

PC\_predicted = PC\_Mem

PC\_UnPredicted = PC+1

PC\_Write

PC\_Predicted

PC

PC\_UnPredicted

PC\_Predicted

Load\_ret\_PC

Wrong-prediction-bit

Unpredicted\_PC\_E

PC predictor

PC\_Mem

PC

Prediction\_bit

Rdst\_val

Opcode\_F