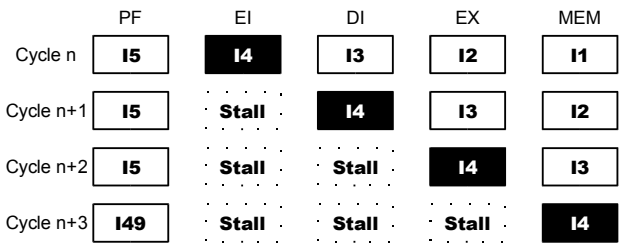
Branch Prediction

This document is the reference one about the branch prediction module included in the miniMIPS core. It contains every details needed to better understand the sources of the module and of course to understand the functionning way of the core. The reason of this module is also given.

The need of branch prediction

In a processor core, as the miniMIPS, based on a pipeline, the apparition of branch or jump instruction launch branch hazards. Those are due to the ignorance of the branch address calculation result.

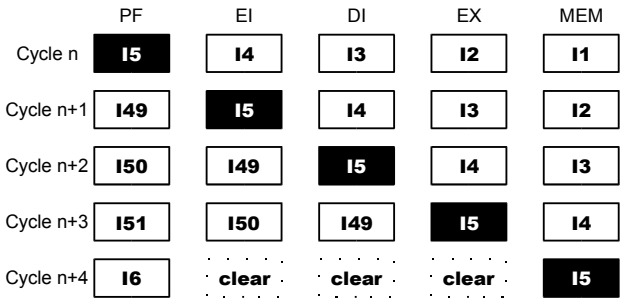
In order to continue the progress of the pipeline, the result must be known immediatly. Different solutions are possible. First, the first pipeline stage can be locked and wait for the result of the branch instruction. That solution means that at each branch instruction, clock cycles are lost. In our case, it is 3.



The state are seen at the pipeline stage output

Illustration 1 : Branch hazard resolution with method 1

The next solution is to anticipate the result of the branch address calculation. That means to predict the result. If the prediction is true then no cycle is lost. Of course there is a cost if the prediction is wrong. In that case, 3 cycles are also lost. This solution is supposing that a branch instruction result is



The state are seen at the pipeline stage output

Illustration 2 : Bad prediction to resolve branch hazard

frequently the same. As in many programs there are loops that may be a good solution. Of course that implies more hardware to implement such a solution than for the first solution.

Thus the choice to predict becomes a necessity as we need a performant execution for our core. The saving time may be very interesting in many programs.

The principle of the prediction

As said just before, branch predictions are a mean to reduce the cost of a branch hazards by anticipating the result of a branch. This section gives the principle of that method.

The idea is to save the last branch instructions results and for that a table of prediction must be designed and maintained during the execution. The contents of that table should be at least :

- the branch address
- the result of the branch.

Three main actions must be done to achieve predictions and maintain the table :

- check if the current instruction address match with one in the prediction table. If there is a match, then branch automatically to the result of the branch.
- check if a new branch instruction has appeared in the pipeline. In that case, store in the table the new instruction and as the result of the branch, the next instruction must be taken in account.
- check if the prediction is ok. If the response is affirmative, nothing happens. In the other case, the pipeline must be cleared to delete the wrong instructions.

The implementation in the miniMIPS

The entity predict is the module which does the branch prediction. The table of predictions contains records that are defined with the signals as follow :

- *is_affected* : a bit signal. When asserted, that means the record contains valid datas.
- *last_bra* : a bit signal. When asserted, that means that the last execution of the instruction had a confirmed test result.
- *code_bra* : a 32 bits large signal. Contains the address of the branch instruction.
- *bra_adr* : a 32 bits large signal. Contains the branch address result of the last execution.

Three pipeline stage are directly concerned by the branch prediction : PF, DI and EX.

First of all, the predict entity try to find a match to the PC actually in the *PF_pc* register. If a match is asserted then the signal *PR_bra_cmd* is asserted to perform a new PC and *PR_bra_adr* gives that PC which is determined thanks to the *bra_adr* signal from the matched prediction record. If a data hazard has appeared just an instruction before then the hazard resolution is executed first.

Then, the predict entity try to find a match to the current PC in the *DI_adr* register. If the match is ok then nothing happens as the branch instruction is already registered. In the other case, if the current PC is the address of a branch instruction (defined by the *DI_bra* signal), a new record is affected. An older record may be erased. The *next_out* signal defined the next record to be cleared. For now, the arbitration of the *next_out* signal is simply the oldest one.

The last step of the prediction is to check if the predictions are ok. That can be

done with the result from the EX pipeline stage. In that stage the branch address result and the test result are achieved. The only things to do is to compare the *last_bra* signal with the *EX_bra_confirm* signal which asserts if the test result is ok and to compare the branch address result (*EX_adresse*) with *bra_adr* signal. If the comparison are true then all gets its own way. If one of the comparison is wrong then the pipeline must be cleared (EI, DI and EX stages only) by setting *PR_clear*, the PC must be changed to the right address by setting *PR_bra_bad* and *PR_bra_adr* and the table of prediction must be updated with the new results.

The illustration following present the mapping of the predict component in the pipeline.

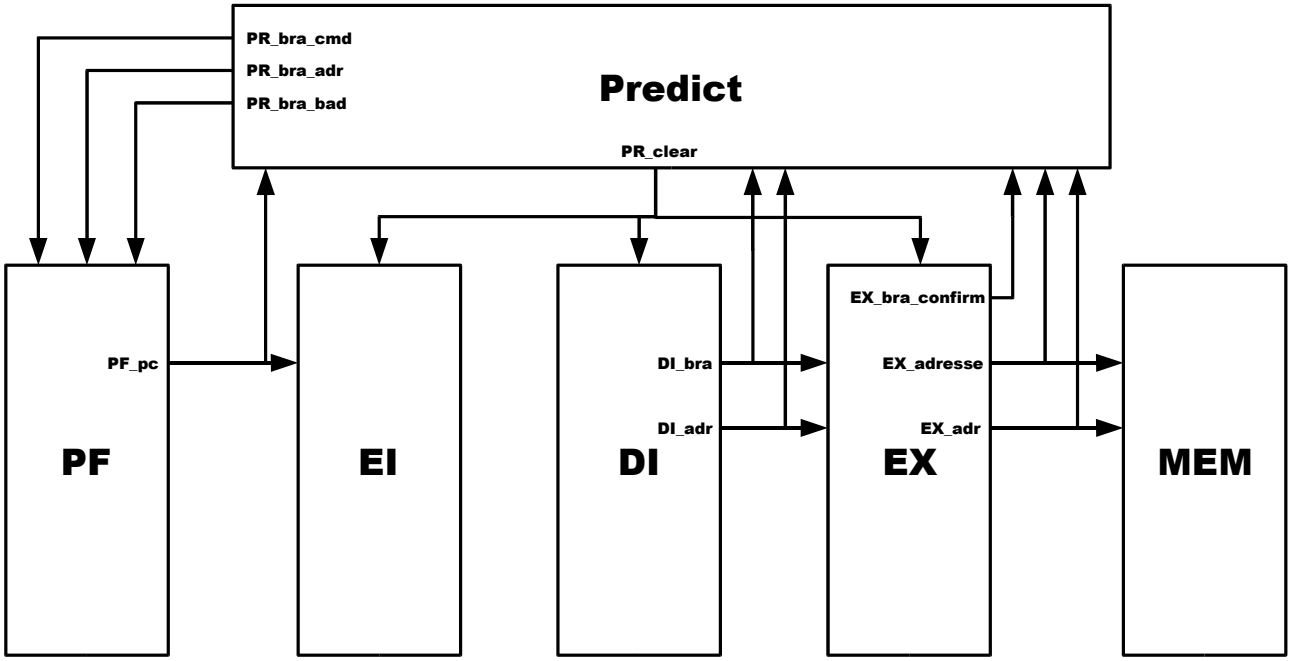


Illustration 3 : Mapping of the predict component