ECE411 Advanced Feature

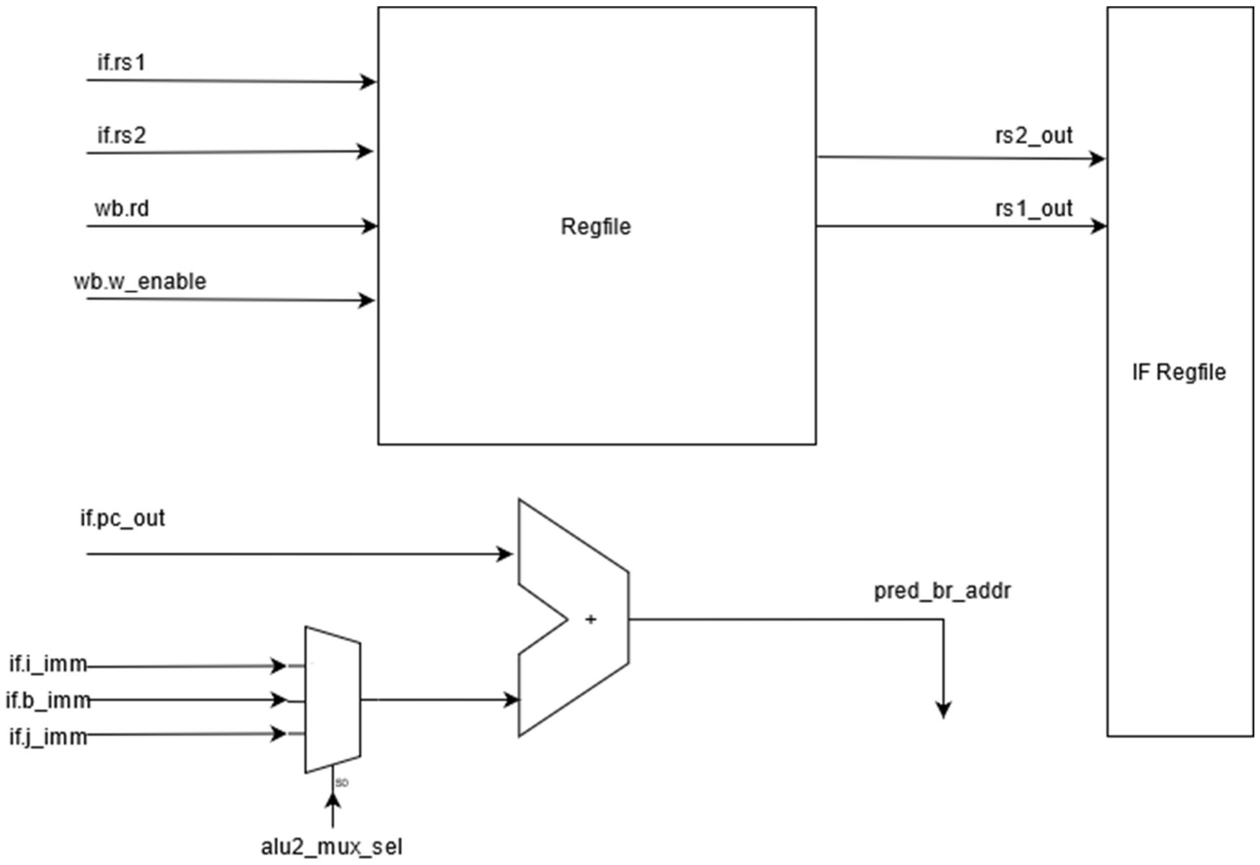
TAGE Branch Predictor

Author: Andrew Gacek (andrewg3)

We implemented a TAGE branch predictor in order to replace the static branch predictor used in CP2. In order to do this, the only change needed was to change the black box that produces the predictions and to calculate the address in ID. The rest of the control logic was already present in CP2.

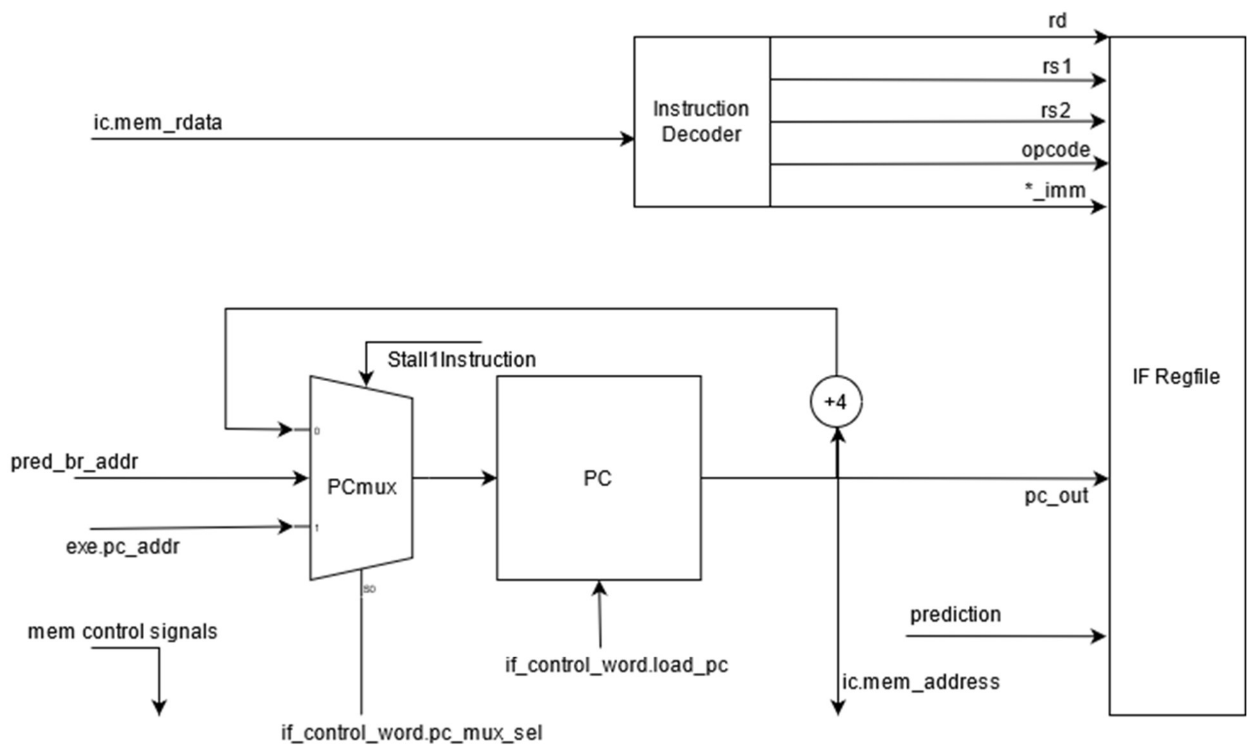
The previous black box for generating predictions always returned a 0. The new black box uses and updates a TAGE predictor. The arithmetic added to ID is very simple and shown below.

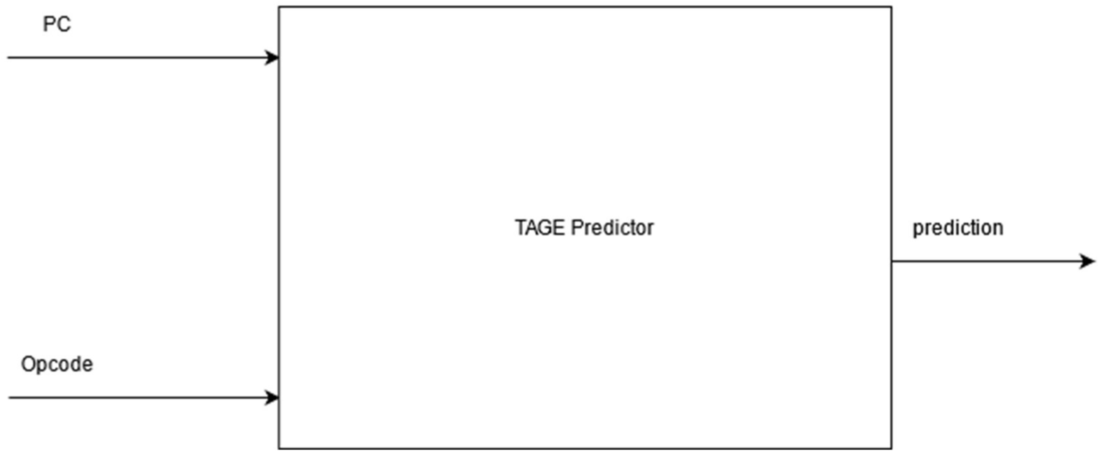
ID Stage



The ID stage now calculates the predicted next address on a branch and sends the value ‘pred_br_addr’ back to IF stage. This address is taken if the predicted direction is ‘1’. Note that this only works for BR and JAL instructions. Without a BTB, we cannot support JALR.

IF Stage





The IF stage is mostly unchanged from before. A new load signal for PC was added. This allows PC to stall exactly one instruction on a hit. Since we need to calculate the new address (we are not using a BTB), we need to wait one cycle to get the new address.

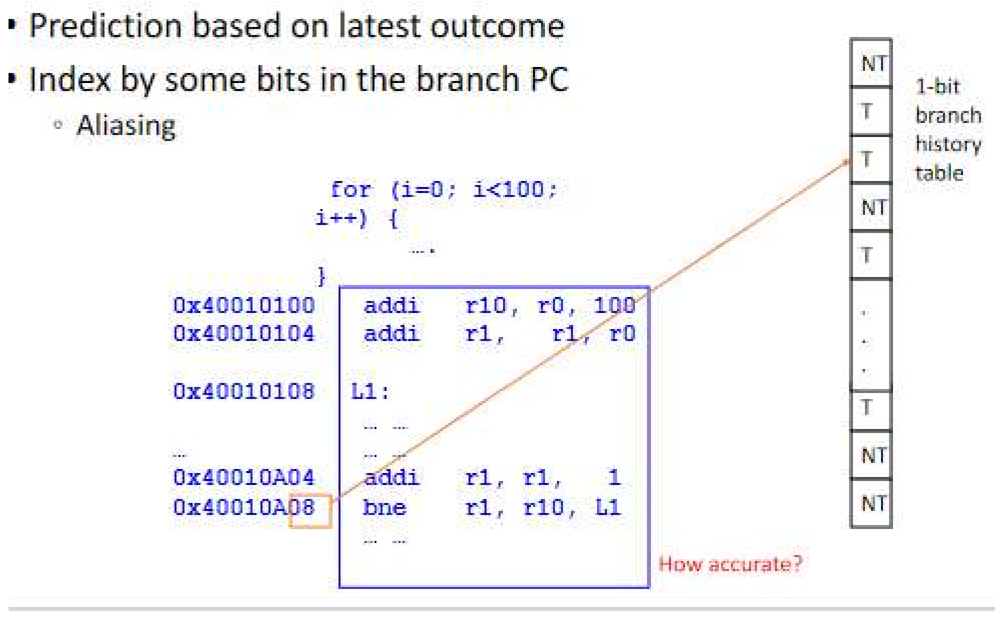
The pc_mux_select signal was expanded to allow a load of the input predicted address from ID.

The prediction still goes through a black box, which is discussed below.

The lack of a BTB does cause a 1 cycle stall, but allows

TAGE Predictor

The TAGE predictor is a type of tournament predictor that uses both local and correlated global predictors. The simple local predictor and the global predictor are unmodified from the lecture notes and are described in some detail here.

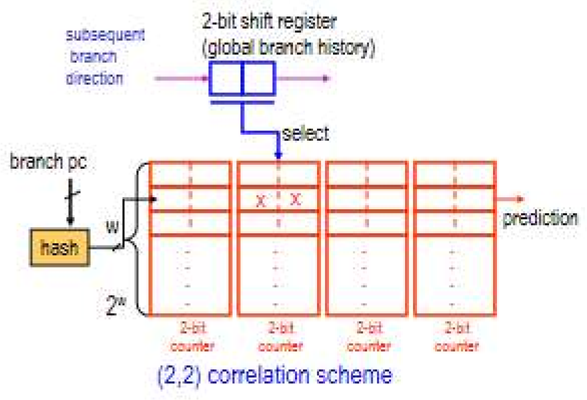


- Prediction based on latest outcome
- Index by some bits in the branch PC
  - Aliasing

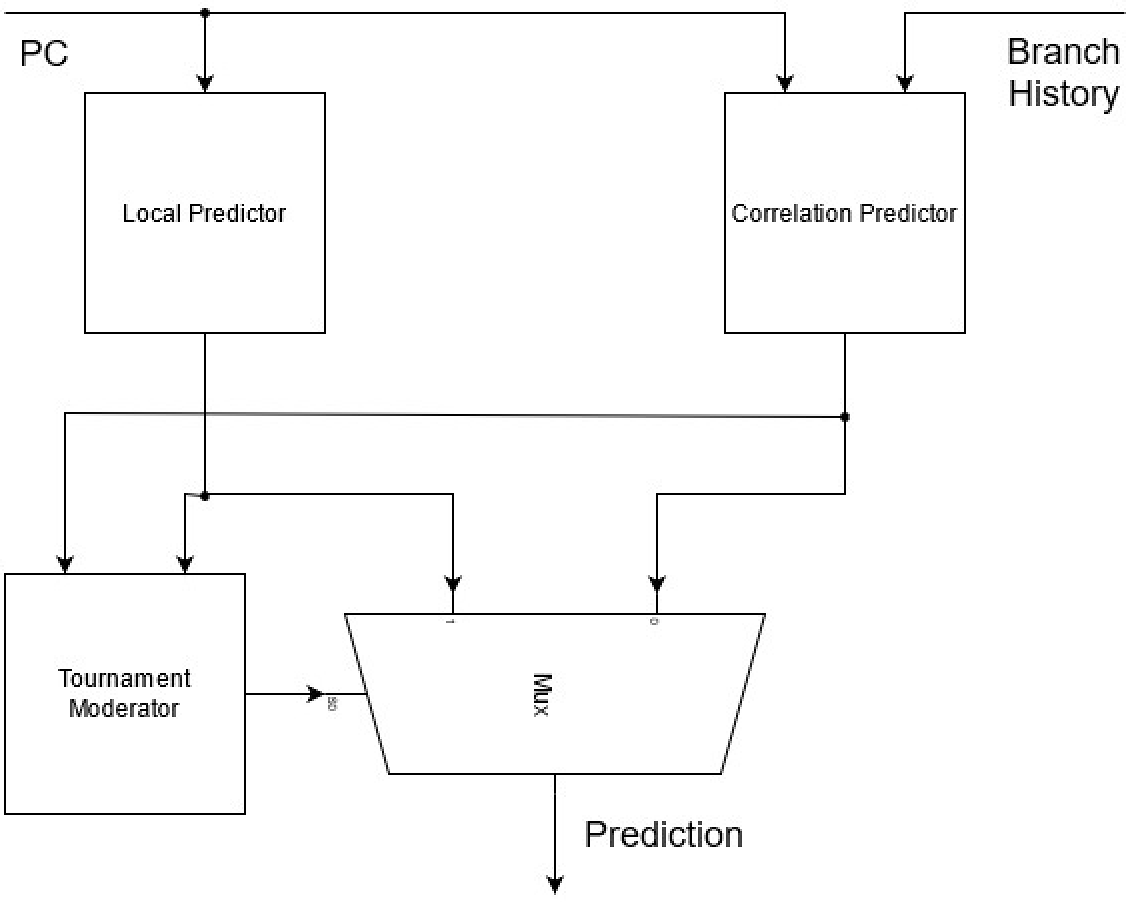
```
                for (i=0; i<100;
              i++) {
                        ….
              }
0x40010100   addi   r10, r0, 100
0x40010104   addi   r1,    r1, r0

0x40010108  L1:
              … …
…             … …
0x40010A04   addi   r1, r1,   1
0x40010A08   bne    r1, r10, L1
              … …
```

Image taken from the lecture notes. The simple predictor will follow this model. The size of the history table, as well as the hashing function have not yet been decided.



The above figure is taken from the notes on correlation prediction schemes.



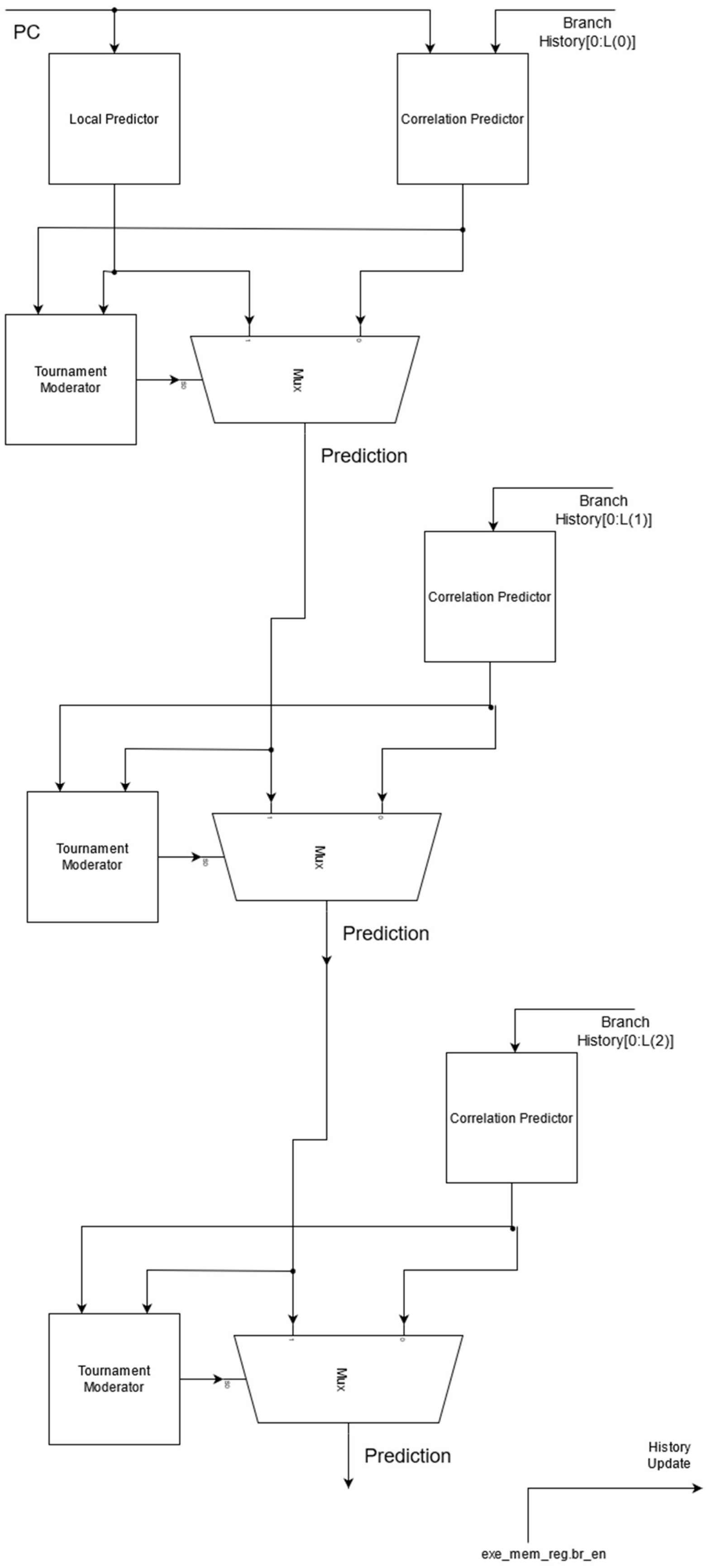
Short diagram showing the basic implementation of a tournament predictor. The final prediction is chosen from the two predictor results.

The local predictor and correlation predictor both take PC as input. The correlation predictors also take the branch history as an input, which is stored in a shift register. This branch history is variable and is updated whenever the opcode is BR.

The tournament moderator chooses the output prediction based on the input confidence. The confidence is based on the value of the input 2 bit predictions from the predictors. The rules for the moderation are simple.

- Pass through the two bit prediction value with the highest confidence
- A prediction of 11 or 00 have equal confidence and a higher confidence than 10 or 01, which are also equal.
- In the event of a tie, choose the output from the right side (correlation predictor).

The TAGE predictor takes the tournament scheme a bit further.


PC
Branch History[0:L(0)]
Local Predictor
Correlation Predictor
Tournament Moderator
Mux
1
0
s0
Prediction
Branch History[0:L(1)]
Correlation Predictor
Tournament Moderator
Mux
Prediction
Branch History[0:L(2)]
Correlation Predictor
Tournament Moderator
Mux
Prediction
History Update
exe_mem_reg.br_en

TAGE uses a series of correlation predictors that each use a different amount of history bits. In a similar way as the tournament predictor, the highest confidence prediction is passed through.

In order to make the TAGE predictor, we needed to create a parameterized correlation predictor. This means we need to have a variable number of registers to hold each of the prediction states, as well as an indexing scheme.

Below is a diagram of the TAGE predictor from the original paper by Andre Seznec. This does a better job of illustrating the TAGE and is included for clarity's sake. The TAGE I implemented is four components with equal spacing between each component of the L vector. No hash functions were used.

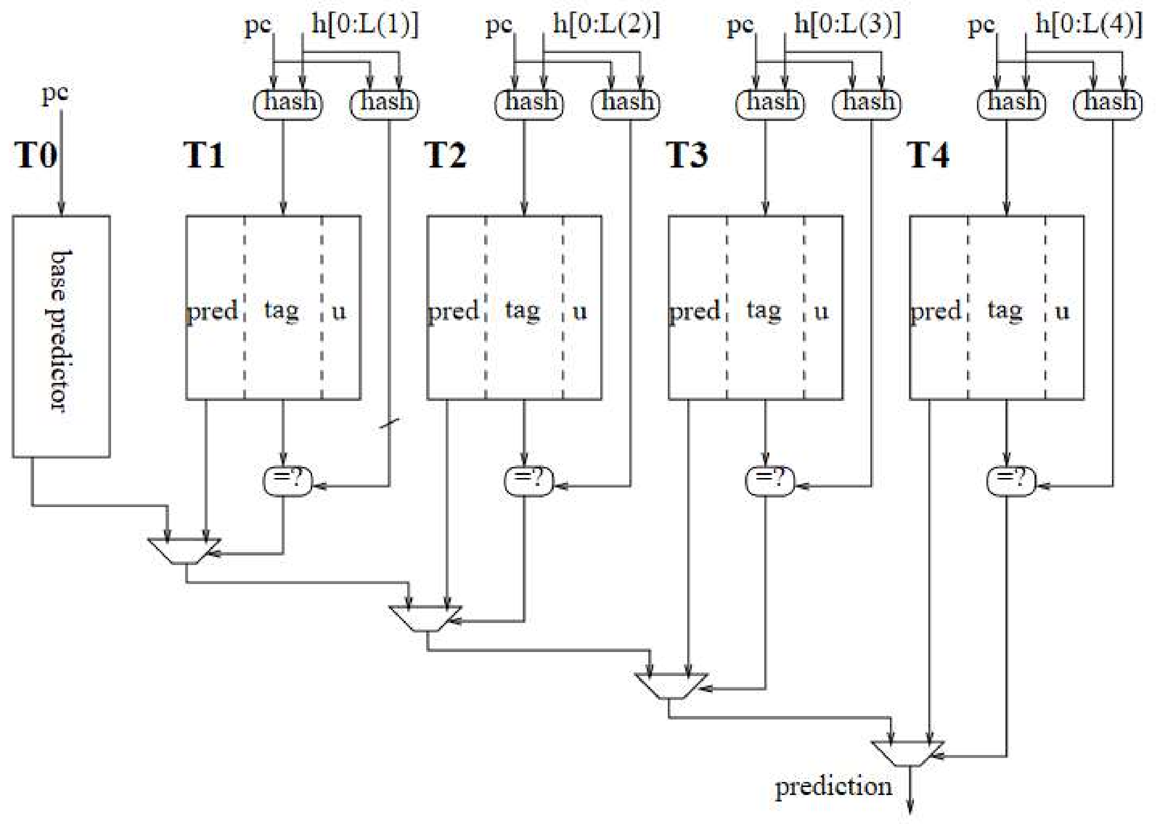


Figure 1: A 5-component TAGE predictor logical synopsis: a base predictor is backed with several tagged predictor components indexed with increasing history lengths. On an effective implementation, predictor selection would be performed through a tree of multiplexors.

Seznec , Andre. The L-TAGE Branch Predictor. In *Journal of Instruction Level Parallelism 9 (2007) 1-13*. 2007.

The L-TAGE branch predictor is very similar to a TAGE, except that it also includes a Loop predictor that stores the number of times a loop branches. We chose to not include this loop predictor as quite frankly, it does not seem to be worth the effort for the small performance boost we might experience.

TESTING

The tournament branch predictor was tested on the different provided test codes. The performance of the predictor does not have to be rigorously tested, as a failure in the prediction process only costs cycles. Regardless, we tested the predictor using different initial state values and tested the transition between states. The mp3-cp2.s code provided the best results, as the branches are almost always taken. The static branch predictor from CP2 served as the baseline.

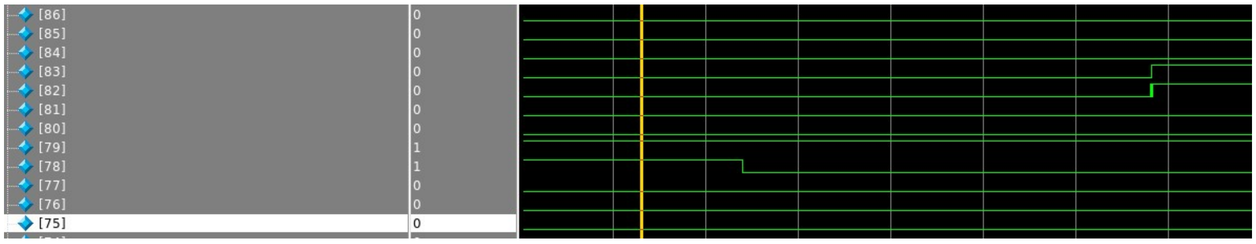
Predictor	Hits	Misses
Static-0	6	581
TAGE Config 1	137	450
TAGE Config 2	581	6

Running on test code with an initial state of strongly confident in one direction leads to some pretty bad results.

Component	Predict 1	Predict 0
Local	587	0
Correlated 1	587	0
Correlated 2	587	0
Correlated 3	587	0

As we can see, this particular example is not running for long enough. Or the code is completely uncorrelated. Inspecting the correlation and local predictors show that the state is changing, but not fast enough to have much effect.

This would make me suspect that the states are not changing. However, this quick snip proves that the predictors are changing state. The initial guess is strongly 1, and most branches are taken, so there is never a chance to leave state 1 in this predictor.



If necessary, I can make some adjustments to the code in order to demo this changing behavior and its effect on the predictions when the initial state is not strongly confident in one guess.