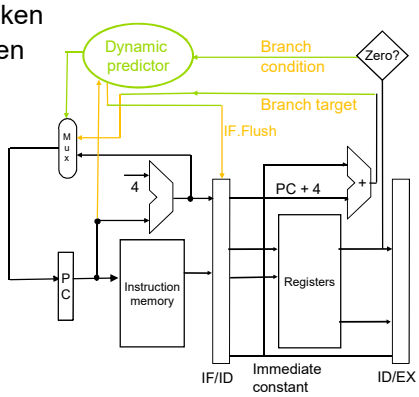
Branch prediction (§3.3)

- Static branch prediction (built into the architecture)
  - ➢The default is to assume that branches are not taken
  - ➢May have a design which predicts that branches are taken
- It is reasonable to assume that
  - ➢forward branches are often not taken
  - ➢backward branches are often taken
- May come up with more static predictors based on branch directions.
- Profiling is the standard technique for predicting the probability of branching.
- Dynamic predictors rely on the history to predict the future branch direction

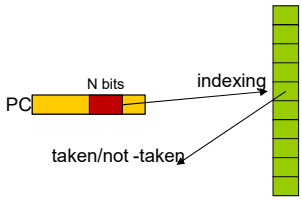


(12)

Dynamic Branch Prediction

- Different than attaching a prediction with each branch
- Performance depends on the accuracy of prediction and the cost of miss-prediction.
- **Branch prediction buffer (Branch history table - BHT)**:
  - 1-bit table (cache) indexed by some bits of the address of the branch instructions (can be accessed in decode stage) → hashing
  - Record whether or not the branch was taken last time
  - May have collision.
  - Will cause two miss-predictions in a loop (at start and end of loop).

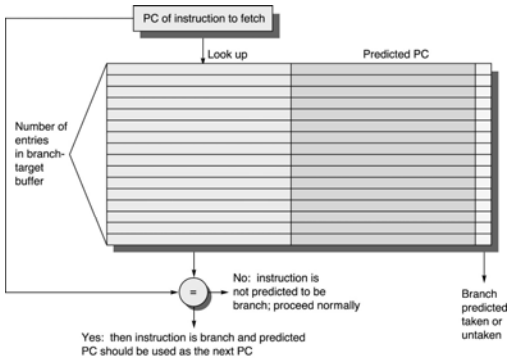
```
L1:  .......
L2:  .......
     .......
     beqz  R2, L2
     .......
     beqz  R1, L1
```



(13)

Branch target buffers(from §3.9)

- Store the address of the branch's target, in addition to the prediction.

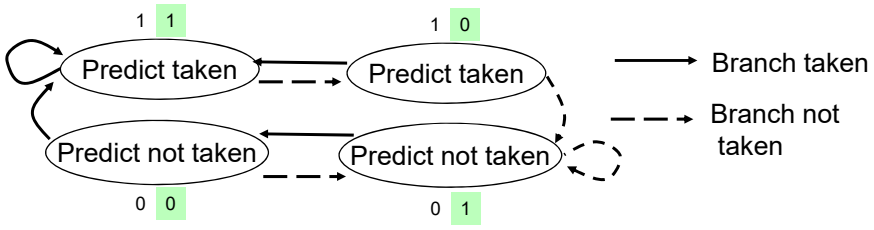


- Can determine the target address while fetching the branch instruction
  - how do you even know that the instruction is a branch?
  - can't afford to use wrong branch address due to collision -- why?

(14)

Two bits branch predictors

- change your prediction only if you miss-predict twice
- helps if a branch changes directions occasionally (ex. Nested loops)



- In general, *n*-bit predictors are called *Local Predictors*.
  - Use a saturated counter (++ on correct prediction, -- on wrong prediction)
  - n-bit prediction is not much better than 2-bit prediction (n > 2).
  - A BHT with 4K entries is as good as an infinite size BHT
  - Miss-predict when gets the wrong branch in BHT or a wrong prediction.

(15)

Correlating Branch predictors (global predictors)

- Hypothesis: recent branches are correlated (behavior of recently executed branches affects prediction of current branch).

- Example 1:

```
if (a == 2)
  a = 0 ;
if (b == 2)
  b = 0;
if (a != b) ...
```

```
        addi   R3,R1, -2
        bnez   R3, L1 …      <= B1
        add    R1, R0, R0
L1:     addi   R3, R2, -2
        bnez   R3, L2        <= B2
        add    R2, R0, R0
L2:     sub    R3, R1, R2
        beqz   R3, L3        <= B3
```

If B1 is not taken and B2 is not taken, then B3 will be taken
If B1 and B2 are taken, then B3 will probably not be taken,

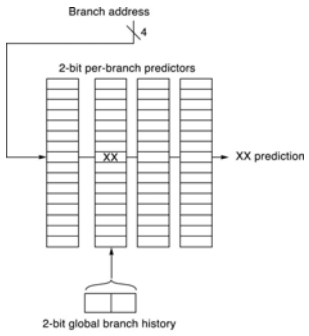
- Example 2:

```
If (d == 0) d = 1 ;
if (d == 1) .....
```

(16)

Correlating Branch predictors

- Keep history of the *m* most recently executed branches in an *m*-bit shift register.
- Record the prediction for each branch instruction, and each of the 2^m combinations.
- In general, (*m,n*) predictor means record last *m* branches to select between 2^m history tables each with *n*-bit predictor.
- Simple access scheme (double indexing).
- A (*0,n*) predictor is a local *n*-bit predictor.
- Size of table is $N \; n \; 2^m$, where *N* is the number of table entries.
- There is a tradeoff between *N* (determines collision), *n* (accuracy of local predicion) and *m* (determines history).

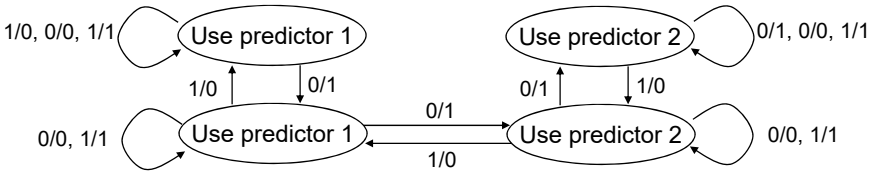


A (2,2) predictor

(17)

Tournament predictor (hybrid local-global predictors)

- Combines a global predictor and a local predictor with a strategy for selecting the appropriate predictor (multilevel predictors).

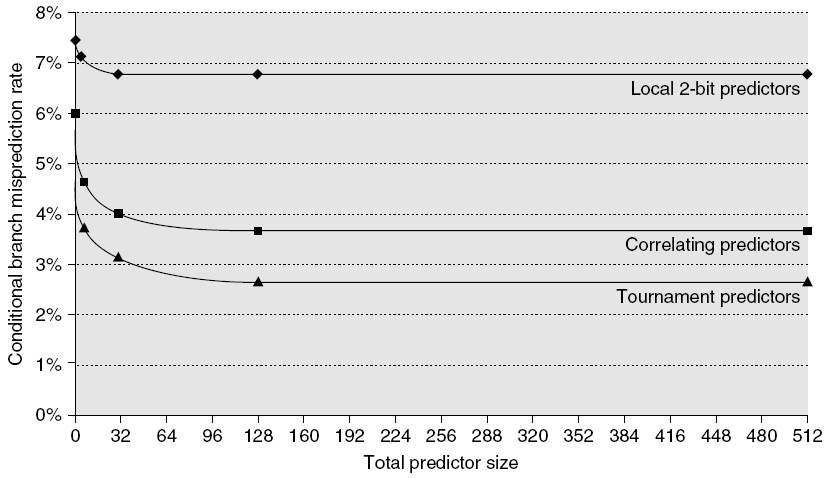


p_1/p_2 == predictor 1 is correct/predictor 2 is correct

- The Alpha 21264 selects between
  - a (12,2) global predictor with 4K entries
  - a local predictor which selects a prediction based on the outcome of the last 10 executions of any given branch.

(18)

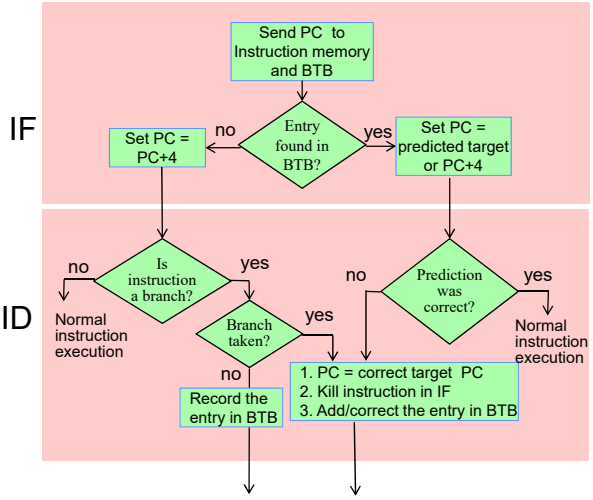
Performance of Branch predictors

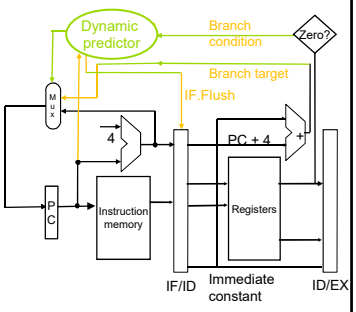


(19)

Branch prediction & pipelining

Assuming that branch condition and target are resolved in ID stage





A similar chart may be drawn if branch condition/target are resolved in EX

(20)

Evaluation example:

- Assume
  - access branch target buffer in IF stage
  - branch condition determined in ID
  - branch address determined in EX stage
- What is the branch penalty if:
  - penalty for correct prediction = 0 cycle
  - penalty for wrong prediction = 1 (or 2) cycles for non-taken (or taken) branch (assuming that target is not stored in BTB if “predict not taken”)
  - penalty if cannot predict and the branch is taken = 2 cycles
  - branch taken frequency = 60%
  - BTB hit rate = 80% (assume not taken in case of inability to predict)
  - BTB prediction accuracy = 90%
  - The correct instruction is fetched 0.8*0.9+0.2*0.6 = 84% of the time
- May store the target instruction and not only the address - useful when access of table needs more than one cycle.

(21)

Dynamically scheduled pipelines (§3.4 - 3.5)

Key idea: allow instructions behind stall to proceed

```
fdiv    F0, F2, F4
fadd    F10, F0, F8
fsub    F12, F8, F14
```

Stall

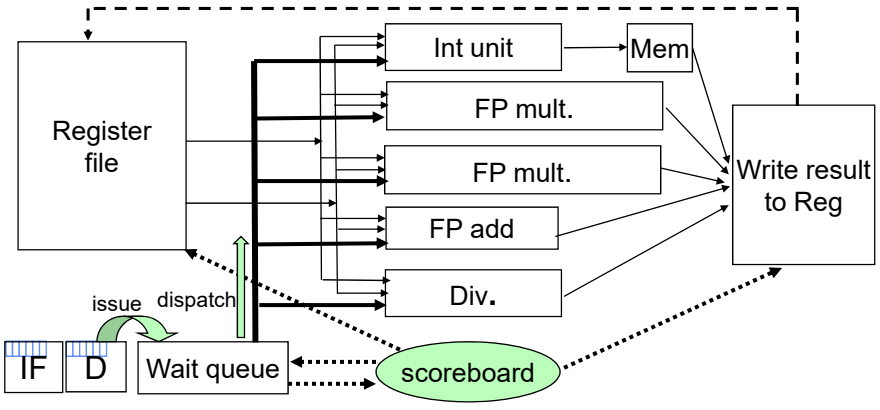
- Enables out-of-order execution,
- Can lead to out-of-order completion.

Using Scoreboards (see Appendix C.7):

- Dates to the first supercomputer, the CDC 6600 in 1963
- Split the ID stage into
  - Issue - decode and check for structural hazards,
  - Read operands → wait until no data hazards, then read operands.
- Instructions wait in a queue and may move to the EX stage (dispatched) out of order.

(22)

A scoreboard architecture



- The scoreboard is responsible for instruction issue and execution, including hazard detection. It is also controlling the writing of the results.
- The "scoreboard" consists of 3 tables to keep track of execution progress and the associated intelligence to determine when to dispatch instructions
- One entry (buffer) in the "wait queue" is associated with each functional unit.

(23)