Technical University of Cluj-Napoca
Computer Science Department

Computer Architecture

Lecturer: Mihai Negru

2nd Year, Computer Science

Lecture 10: Advanced Pipelining 2

http://users.utcluj.ro/~negrum/

More ILP → Speculation

- Speculation vs. Prediction
  - Prediction
    - Refers to Instruction Fetch → Branch prediction → issue rate
    - Must be de-coupled from the decision to execute the fetched instructions
    - Prediction can increase the issue rate but not the completion rate of the instructions
    - The completion rate has to be increased to keep up with the issue rate
  - Speculation
    - Refers to the execution of predicted instructions before it is known if it is safe to do so
    - Helps enhance the executable ILP
    - Relies on branch prediction
    - Key idea: separate instruction execution from instruction commitment
    - Compute on a temporary basis until speculation (prediction) outcome is determined

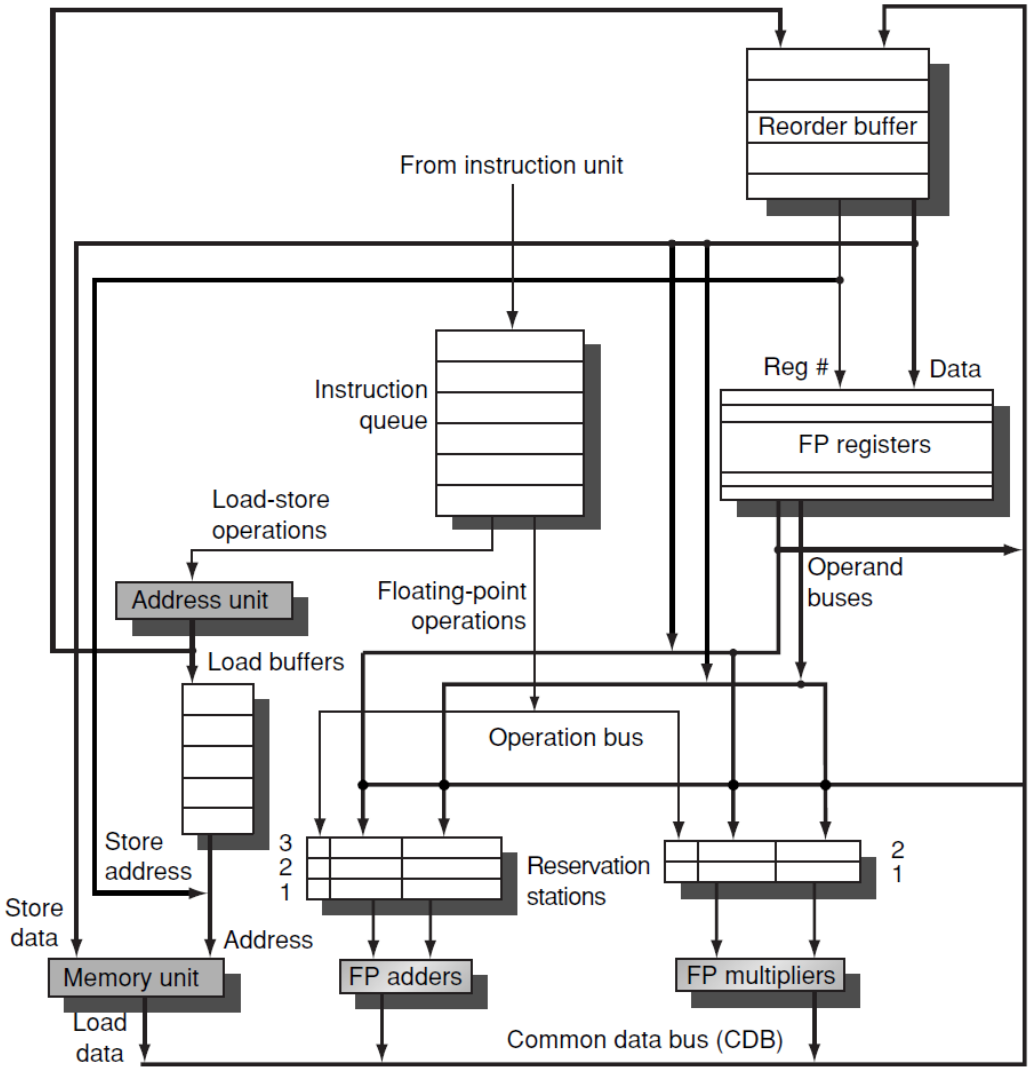
Hardware Based Speculation

- 3 components:
  - Dynamic scheduling
    - out-of-order data flow execution model
    - operations execute as soon as their operands are available
  - Dynamic branch prediction
    - fetch instructions to be speculatively executed
  - Speculation
    - execution of instructions before control dependences are resolved and
    - the ability to undo the effects of incorrectly speculated sequence

Speculative Tomasulo

- Some processors that implement speculative execution based on Tomasulo's algorithm:
  - PowerPC 603/604/G3/G4, MIPS R10000/R12000, Intel Pentium II / III / IV, Alpha 21264, and AMD K5/K6/Athlon
- Additional step – instruction commit
  - When an instruction is no longer speculative, allow it to update the Register File or Memory **in the order of the program**
- Requires an additional set of buffers:
  - To hold the results of instructions that have finished execution but have not committed
  - To reorder the instructions that complete out-of-order
  - This **reorder buffer (ROB)** is also used to pass results among instructions that may be speculated

Speculative Tomasulo



Changes in the data-path:

- ROB (Reorder Buffer)
- Elimination of the store buffers
  - Their function is integrated into the ROB.

- This mechanism can be extended to multiple issues (superscalar) by making the common data bus (CDB) wider to allow for multiple completions per clock.

[1]

MIPS FP Unit using Tomasulo's algorithm extended to handle speculation

Speculative Tomasulo

- Load / Store execution:
  - Loads and stores require a two-step execution as in the non-speculative Tomasulo
  - First step computes the effective address, when the base register is available, and the effective address is then placed in the load or store buffer
  - Loads in the load buffer execute as soon as the memory unit is available
  - Stores in the store buffer wait for the value to be stored before being sent to the memory unit
  - Stores still execute in two steps, but the second step is performed by instruction commit

- ROB completely replaces the store buffers

Reorder Buffer (ROB)

- Reorder Buffer – Possible Entries
  - Instruction type
    - A branch (has no destination result),
    - A store (has a memory address destination)
    - Register operation (ALU operation or load, which has register destinations)
  - Destination
    - Register number (for loads and ALU operations)
    - Memory address (for stores) where the instruction result should be written
  - Value
    - Value of instruction result until the instruction commits
  - Busy
    - Indicates that instruction has completed execution, and the value is ready
  - Supplementary fields to handle speculation and exceptions
    - The speculative field has 3 values: speculative, confirm and not confirmed
    - Should a branch become confirmed it will turn the “speculative” bits of the corresponding speculative instructions to “confirm”.
    - If it is not confirmed, status is set to “not confirmed”

Reorder Buffer (ROB)

- In non-speculative Tomasulo's algorithm, the instructions write their results in the RF.
- With speculation, the RF is not updated until the instruction commits (we know that the instruction should execute):
- The ROB supplies operands in the interval between completion of instruction execution and instruction commit
  - ROB extends architecture registers like Reservation Stations
- Concept of Reorder Buffer (ROB):
  - Holds instructions in FIFO order, exactly as they were issued
  - When instructions complete, results placed into ROB
    - Supplies operands to other instructions between execution complete and commit → more registers like RS
    - Tag results with ROB buffer number instead of reservation station
  - Instructions commit → values at head of ROB are placed in RF
  - As a result, speculated instructions on miss-predicted branches or on exceptions are easily canceled.

Speculative Tomasulo Steps

- Issue
  - Get an instruction from the instruction queue
    - Issue the instruction if there is an empty RS (reservation station at required FU) and an empty slot in the ROB
    - If either all RS are full or the ROB is full, then instruction issue is stalled until both have available entries
  - Allocate a RS and ROB entry
  - Rename the source and destination registers
  - Dispatch the decoded instruction and renamed registers to the RS and ROB
  - Send the operands to the RS if they are available in either the Registers or the ROB
  - Update the control entries to indicate the buffers are in use.
  - The number of the ROB allocated for the result is also sent to the RS, so that the number can be used to tag the result when it is placed on the CDB.

Speculative Tomasulo Steps

- Execute
  - If both operands are available at a RS, execute the operation
  - If one or more of the operands is not yet available, monitor the CDB for them
  - This step checks for **RAW hazards**
  - Instructions may take multiple clock cycles in this stage, and loads still require two steps in this stage.
  - Stores need only have the base register available at this step, since execution for a store at this point is only effective address calculation.

Speculative Tomasulo Steps

- Write Result (finish execution)
  - Condition: At a given FU, some instruction finishes execution
  - When the result is available, write it on the CDB (with the ROB tag sent when the instruction issued) and from the CDB into the ROB, as well as to any RS waiting for this result.
  - Mark the RS as available (de-allocate)
  - Special actions are required for store instructions.
    - If the value to be stored is available, it is written into the Value field of the ROB entry for the store.
    - If the value to be stored is not available yet, the CDB must be monitored until that value is broadcast, at which time the Value field of the ROB entry of the store is updated.

Speculative Tomasulo Steps

- Commit
  - IF: ROB is not empty and ROB head instruction has finished execution
    - Commit valid instructions at the head of the ROB
    - **Commit instructions in-order!**
  - There types of actions depending on the committing instruction:
    - A branch with incorrect prediction / store / any other instruction (normal commit).
    - The normal commit case occurs when an instruction reaches the head of the ROB and its result is present in the buffer; at this point, the processor updates the register with the result and removes the instruction from the ROB.
    - Committing a store is similar: the Memory is updated, not the Register File.
    - When a branch with incorrect prediction reaches the head of the ROB, it indicates that the speculation was wrong.
      - The ROB is flushed and execution is restarted at the correct successor of the branch.
    - If the branch was correctly predicted, the branch is finished.
    - Once an instruction commits, its entry in the ROB is reclaimed and the register or memory destination is updated, eliminating the need for the ROB entry.
  - If the ROB fills, stop issuing instructions until an entry is made free.

Speculative Tomasulo

- Memory Disambiguation
  - Difference for store between speculative and classical Tomasulo's algorithm
  - In Tomasulo's algorithm, a store can update memory when it reaches Write Results and the data value to be stored is available
  - In a speculative processor, a store updates memory only when it reaches the head of the ROB
    - This difference ensures that memory is not updated until an instruction is no longer speculative
  - We must avoid hazards through memory
  - WAW and WAR hazards through memory are eliminated with speculation
    - Memory updating occurs in order, when a store is at the head of the ROB
    - Hence, no earlier loads or stores can still be pending
  - RAW hazards through memory are handled by two restrictions
    - Not allowing a load to initiate the second step of its execution if any active ROB entry occupied by a store has destination address of the load
    - Maintaining the program order for the computation of an effective address of a load with respect to all earlier stores.
    - Together, these two restrictions ensure that any load that accesses a memory location written to by an earlier store cannot perform the memory access until the store has written the data.

Speculative Tomasulo

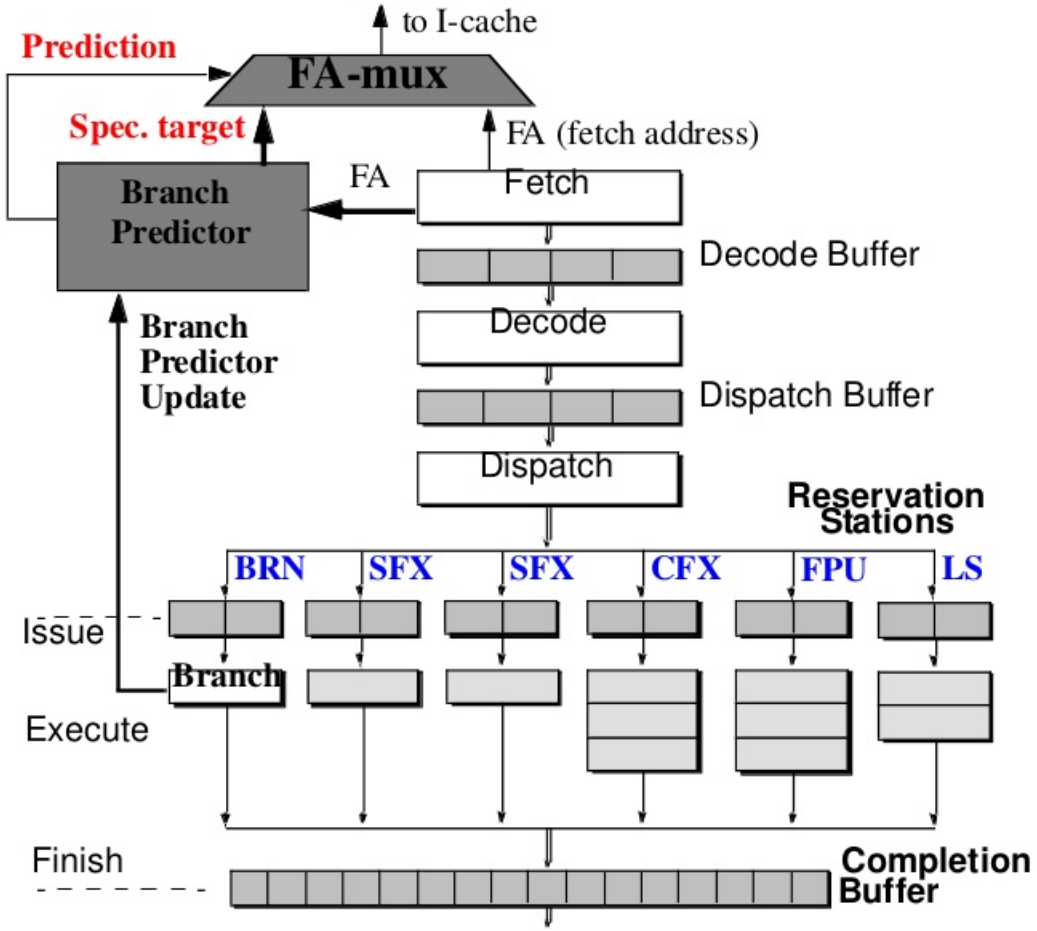
- How much to speculate?
  - One of the significant advantages of speculation is its ability to cover events that would otherwise stall the pipeline early, such as cache misses.
  - A potential disadvantage – the processor may speculate that some costly exceptional event occurs and begins processing the event, when in fact, the speculation was incorrect.
  - To maintain some of the advantage, while minimizing the disadvantages, most pipelines with speculation will allow only low-cost exceptional events (such as a first-level cache miss) to be handled in speculative mode.
  - If an expensive exceptional event occurs, such as a second-level cache miss or a TLB miss, the processor will wait until the instruction causing the event is no longer speculative before handling the event.
  - This may slightly degrade the performance of some programs, but avoids significant performance losses in others (high frequency of such events coupled with less than excellent branch prediction)

Speculative Tomasulo

- Speculation trough multiple branches
  - 3 situations in which speculating on multiple branches is advantageous:
    - A very high branch frequency
    - Significant clustering of branches
    - Long delays in functional units
  - In the first two cases, achieving high performance may mean that multiple branches are speculated
  - Likewise, long delays in functional units can raise the importance of speculating on multiple branches as a way to avoid stalls from the longer pipeline delays
  - Speculating on multiple branches slightly complicates the process of speculation recovery
  - How much parallelism is available depends on the running program
  - The implementation techniques cannot take advantage of more parallelism than the one provided by the application

Branch Prediction

- Branch predictor – determines if a conditional branch is likely to be taken or not!



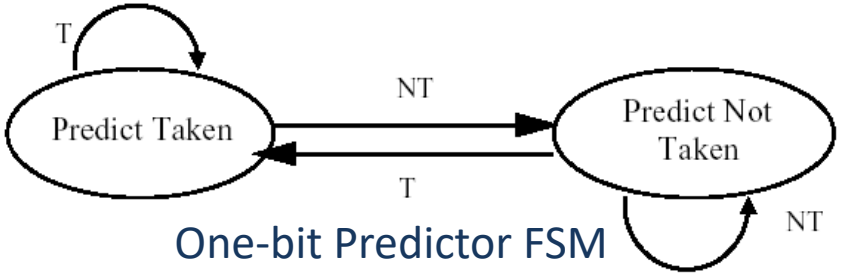
General Pipeline CPU Architecture with Branch Predictor

Branch Prediction

- Static branch prediction
  - Compilers decide the direction
  - Direction of the branch – sign bit of the offset; predict:
    - backward branches taken
    - forward branches not taken
    - Algorithm: Backward Taken Forward Not taken (BTFN) ~ 65% correct (SPECint98)
  - Other:
    - Special branch instructions that encode the compiler's prediction – 1-bit
    - Profiling or feedback-directed compilation
- Dynamic branch prediction
  - Hardware decides the direction using history information
  - Dynamic algorithms take into account run-time information
  - The processor learns from its mistakes and changes its predictions to match the behavior of each particular branch
  - A dynamic algorithm keeps a record of previous branch behavior, allowing it to improve its predictions over time

Dynamic Branch Prediction

- 1-bit predictor
  - A simple scheme, maintains a single history bit for each branch
  - When a branch is encountered, it is predicted to go the same way it did the previous time
  - This technique can push accuracy to 80%.

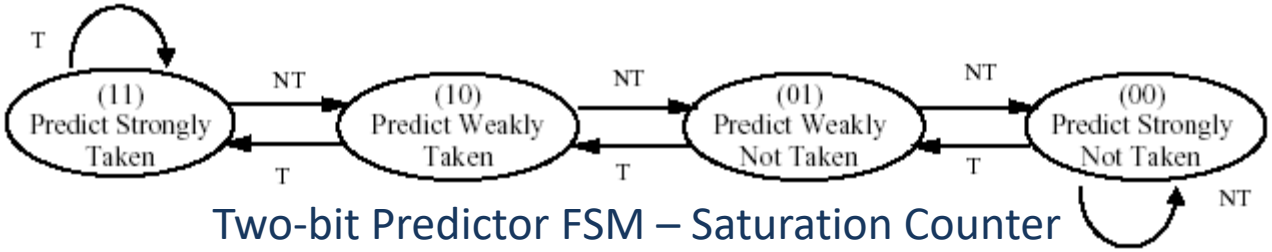


One-bit Predictor FSM

  - A one-bit predictor correctly predicts a branch at the end of loop iteration, as long as the loop does not exit.
  - In nested loops, a one-bit prediction scheme will cause two miss-predictions for the inner loop:
    - One at the end of the loop, when the iteration exits the loop and
    - One at the first loop iteration, when it predicts exit instead of looping
    - 2-bit predictor – avoids the double miss-prediction in nested loops

Dynamic Branch Prediction

- 2-bit predictor

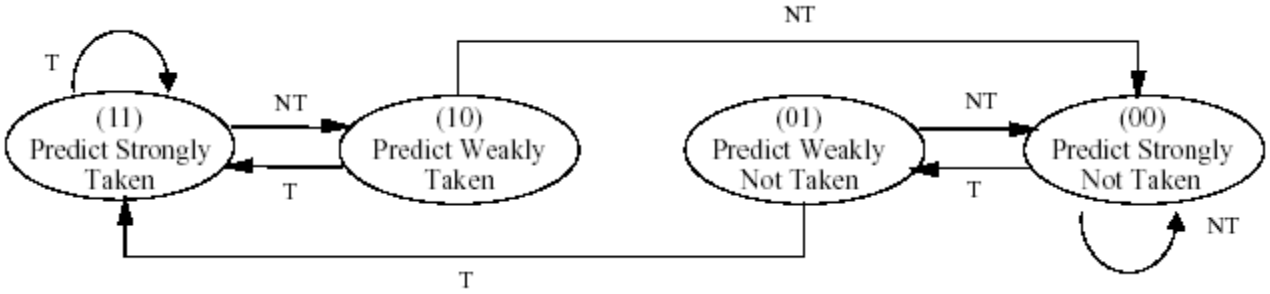


Two-bit Predictor FSM – Saturation Counter

  - States 01, 10 – one wrong prediction changes direction
  - States 00, 11 – two wrong predictions change direction
  - The counter is inc./dec. when the branch is taken/not-taken
  - The most-significant bit is used to predict future occurrences
  - Saturated counter
  - By using a 2-bit counter, the predictor can tolerate a branch going in an unusual direction one time and keep predicting the usual branch direction.
  - This hysteresis effect can boost prediction accuracy to 85% on SPECint92, depending on the size and type of history table that is used.
  - Studies showed that a two-bit prediction scheme does almost as well as an *n*>2-bit scheme

Dynamic Branch Prediction

- 2-bit predictor – UltraSPARC

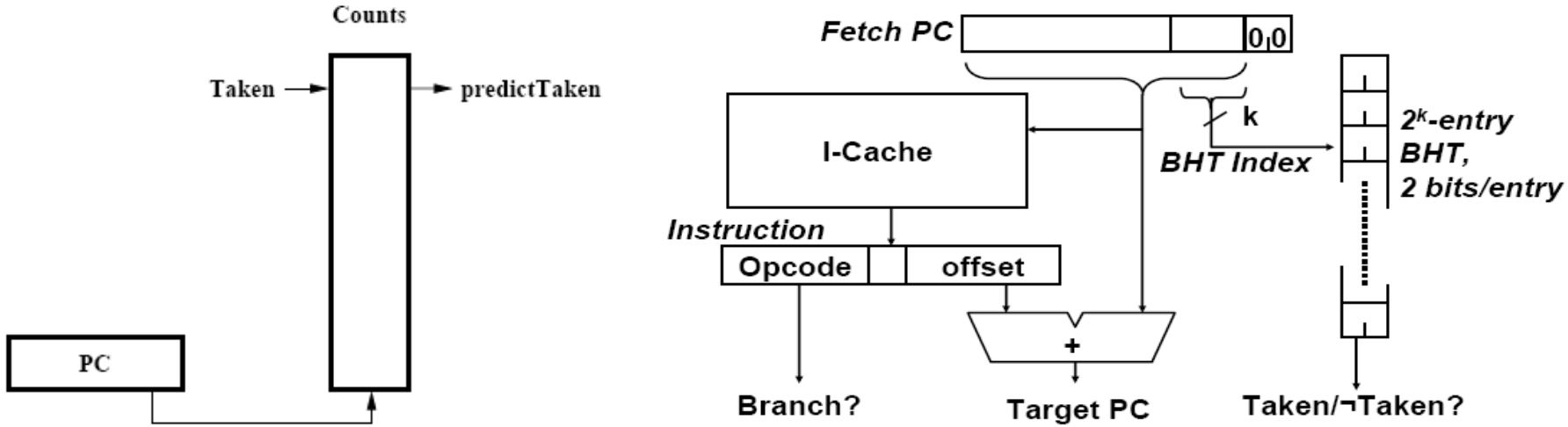


FSM with hysteresis

  - More hysteresis, two wrong predictions needed to change the direction
  - No direct connections between Weakly Taken and Weakly Not Taken states

Dynamic Branch Prediction

- Bimodal Branch Prediction
  - Branches are either taken or not taken.
  - Bimodal branch prediction takes advantage of behavior

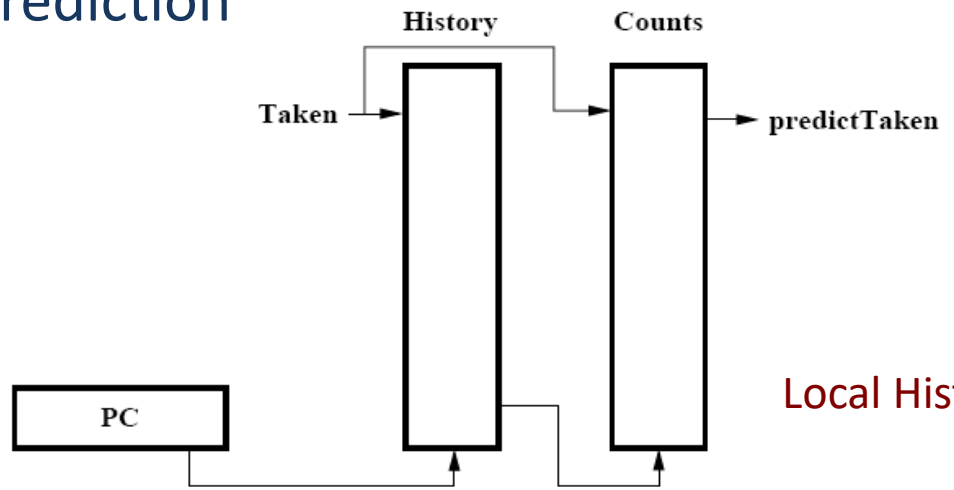


[4] Bimodal Branch Prediction

  - Counts: Table of 2-bit counters indexed by the low order address bits of PC
    - BHT – Branch History Table
  - For each taken/not-taken branch, the appropriate counter is inc. / dec.
  - Prediction is based on the current state (not updated) of the selected counter

Dynamic Branch Prediction

- Local Branch Prediction

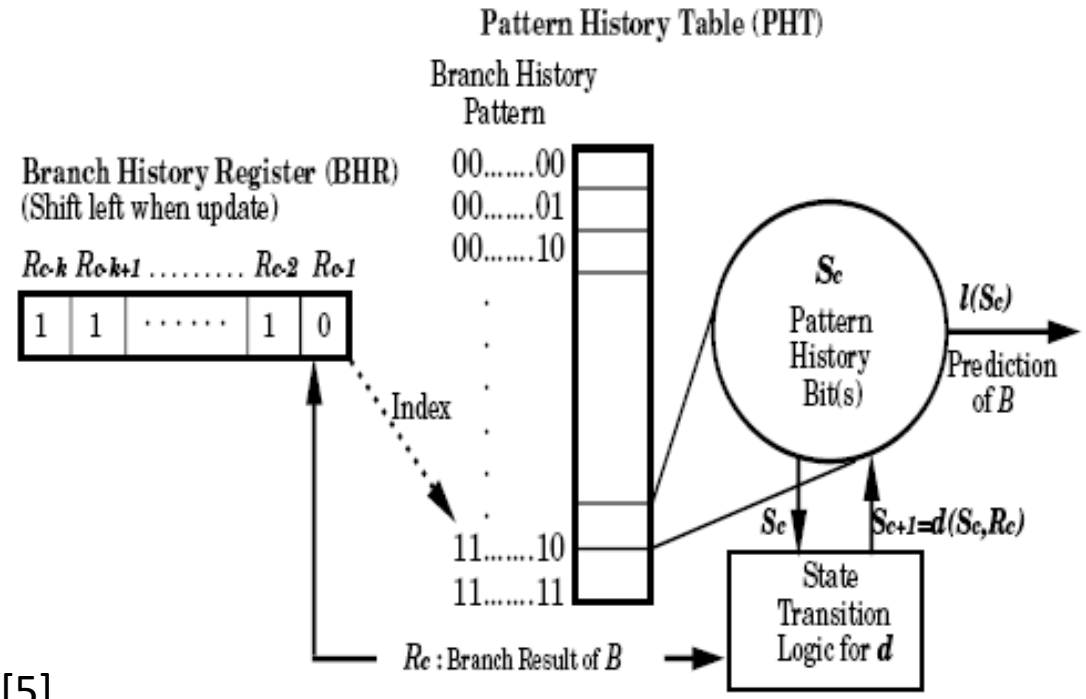


Local History Predictor [4]

  - The first table records the history of recent branches: an array indexed by the low-order bits of the branch address.
  - Performance decreases when multiple branches point to the same entry
  - The history table records the direction of the last n branches whose addresses map to the same entry; Ex: n=6 100100 1=taken, 0=not taken
  - The second table (Counts) is an array of 2-bit counters identical to those used for bimodal branch prediction (indexed with history – reduces aliases)
  - Referred to as local branch prediction (Scott McFarling)

Dynamic Branch Prediction

- Global Branch Prediction
  - 2-level Adaptive Prediction [Yeh & Patt]
  - Branch predictors that use the behavior of other branches to make a prediction are called correlating predictors or two-level predictors



- First level: History of last k branches (context)
- Second level: branch behavior in the given context

- Uses a branch history register (BHR) in conjunction with a Pattern History Table (PHT).

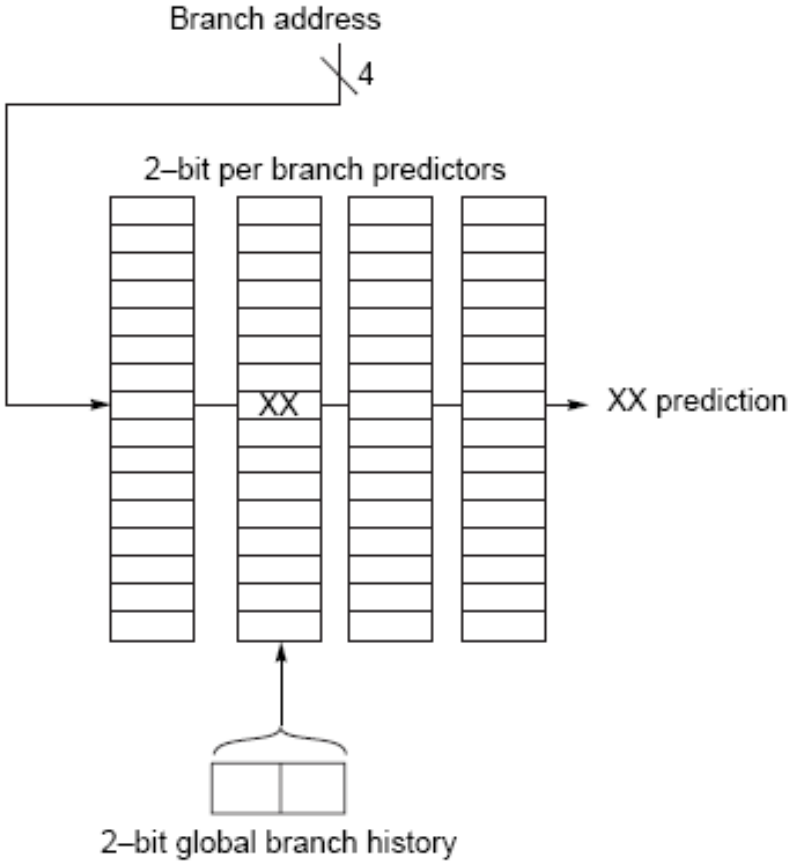
- Effectiveness: Averaging 97% accuracy for SPEC
- Used in the Intel P6 and AMD K6

[5]

Two-level adaptive branch prediction (Global BHR and global PHT)

Dynamic Branch Prediction

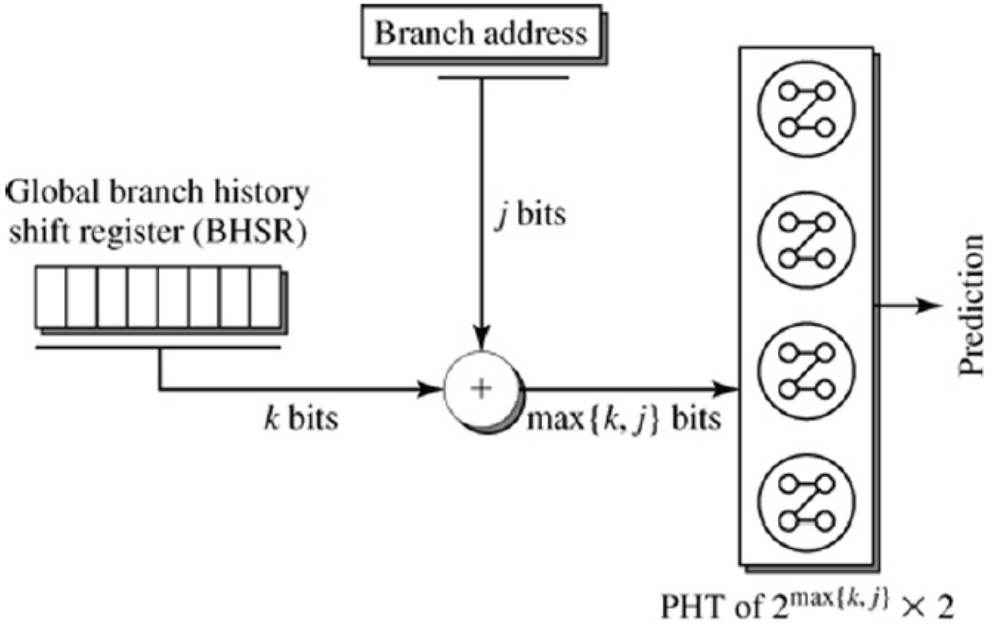
- 2-level adaptive branch prediction



- 2-bit global history selects 1 of 4 predictors for each branch address
- Each predictor is in turn a two-bit predictor for that particular branch
- 4x16=64 entries; the branch address is used to choose four of these entries and the global history is used to choose one of the four
- Global history is a shift register that shifts in the behavior of the branch
- ARM Cortex A57

Dynamic Branch Prediction

- Gshare Branch Prediction [McFarling]

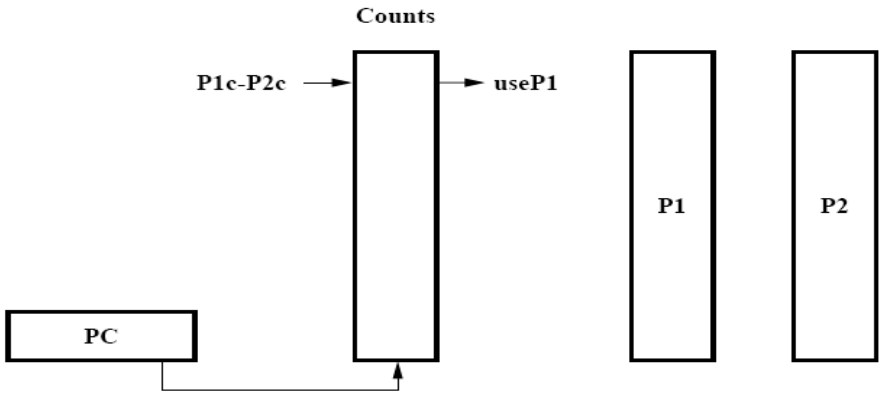


[4]

  - An address maps to many locations
  - Combine branch address with history to reduce aliasing and capture context
  - Ex: AMD Athlon, MIPS R12000, Intel Atom (Silvermont MicroArchitecture), ARM Cortex A53

Dynamic Branch Prediction

- Combining Branch Predictors
  - Two predictors P1 and P2 that could be one of the predictors discussed
  - An additional counter array serves to select the best predictor to use
  - 2-bit up/down saturating counters are used
  - Each counter keeps track of which predictor is more accurate for the branches that share that counter
  - Notation: P1c and P2c denote correct predictions for P1 and P2
  - The counter is incremented or decremented by P1c-P2c as shown
  - One combination of branch predictors that is useful is bimodal/Gshare

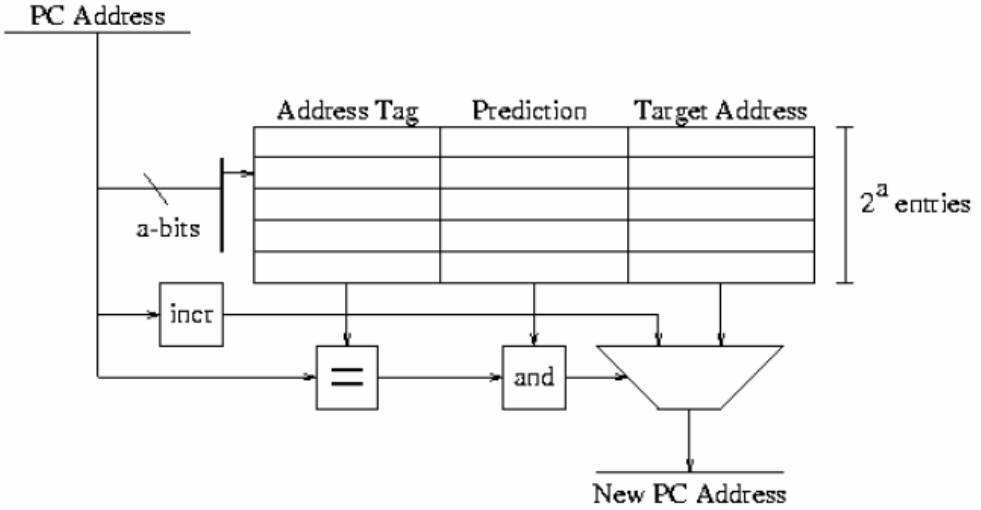


Combined Predictor Structure [4]

P1c	P2c	P1c-P2c	Next counter value
0	0	0	No change
0	1	-1	Decrement counter
1	0	1	Increment counter
1	1	0	No change

Dynamic Branch Prediction

- Branch Target Buffer – BTB
  - A branch-prediction cache that stores the Target Addresses for the taken branches is called a branch-target buffer or branch-target cache

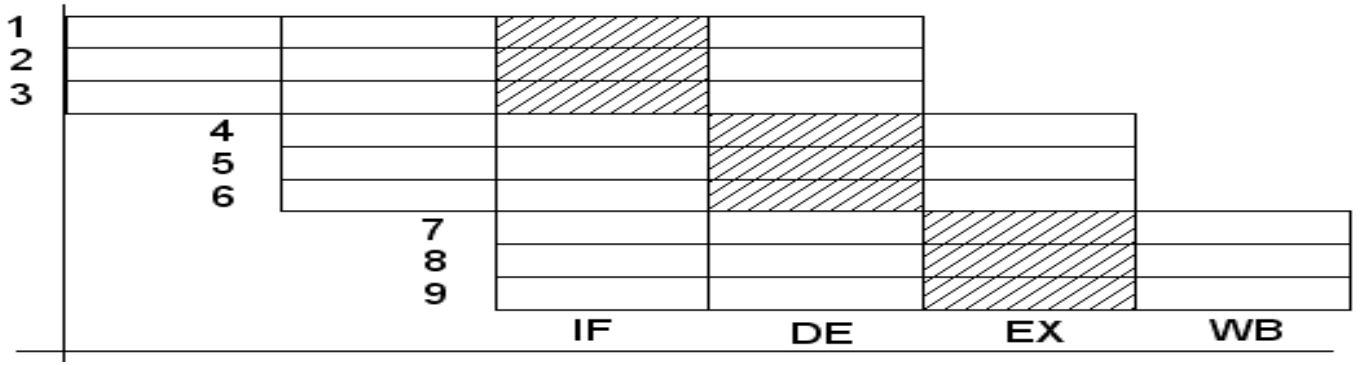


Branch Target Buffer – BTB

  - BTB is used in the instruction fetch stage to determine the new PC.
  - If the PC matches one of the BTB entries, the new PC Address is selected according to Prediction; Fetching begins immediately at that address.
  - One variation on the branch-target buffer is to store one or more *target instructions* in addition to the predicted *target address*
  - Intel Pentium, Intel Pentium MMX

More Advanced Pipelining Concepts

- Superscalar architecture – more ILP
  - Execute more than one instruction in one clock cycle by simultaneously dispatching multiple instructions
  - Exploit ILP
  - A superscalar is dynamically scheduled in hardware



  - Branch prediction – more critical as pipelines get longer and wider
    - Predict both the branch condition and the target
  - Use more registers (reservation stations), load/store buffers, re-order buffer
- In order issue – out of order execution – in order commit

More Advanced Pipelining Concepts

- Multi-Core CPUs – single-chip multiprocessors
- Thread Level Parallelism (TLP) – Hardware Multithreading
  - Fine-grained Multithreading
    - Switching between threads after every instruction
  - Coarse-grained Multithreading
    - Switching between threads only after significant events (Ex: cache miss)
  - Simultaneous Multithreading (SMT)
    - Combines hardware multithreading with superscalar processors to allow multiple threads to issue instructions in each clock cycle
    - Speculation is necessary for SMT performance
    - Intel Hyper-threading = SMT
    - With Hyper-Threading the computer can have one physical processor installed in the motherboard, but the OS will see two logical processors, and treat the system as if there were actually two processors.

References

1. D. A. Patterson, J. L. Hennessy, “Computer Organization and Design: The Hardware/Software Interface”, 5th edition, ed. Morgan–Kaufmann, 2013.
2. R. Iris Bahar, Advanced Computer Architecture, Lecture 9: Branch Prediction, EN292-S10 October 3, 2006
3. Shen & Lipasti, Modern Processor Design: Fundamentals of Superscalar processors, *McGraw Hill*, 2005
4. Combining Branch Predictors; *Scott McFarling* WRL Technical Note TN-36, 1993
5. A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History, Tse-Yu Yeh and Yale N. Patt, Department of Electrical Engineering and Computer Science, University of Michigan.