RISC-V Pipelined Architecture IV

Muhammad Tahir

Lecture 13

Electrical Engineering Department
University of Engineering and Technology Lahore

Contents

Control Hazards

- Control Hazard Sources: `Unconditional Jumps`, `Conditional Branches` as well as `Interrupts/Exceptions`
- Branches and Jumps alter the normal execution flow
- Attributes associated with execution flow control
  - **Taken** or **Not Taken** (control transfer)
  - **Target Address** (When is it known?)

Control Hazards Cont'd

- Unconditional Jumps:
  - JAL: Jump to pc + Immediate
  - JALR: Jump to rs1 + Immediate
- Conditional Branches:
  - B<condition>: Jump to pc + Immediate

Table 1: Control Transfer Behavior

Instruction	Is **Taken** known?	Is **Target Address** known?
JAL	Yes, after Decode	Yes, after Execute
JALR	Yes, after Decode	Yes, after Execute
B<condition>	Yes, after Execute	Yes, after Execute

Control Hazards Cont'd

- Performance can be improved by:
  - Determining the branch taken or not as soon as possible
  - Try to compute branch taken address sooner
- A possible solution:
  - Move the branch test to Decode stage
  - Include an adder in Decode stage to perform pc + Immediate
  - Branch delay is reduced to 1 clock cycle

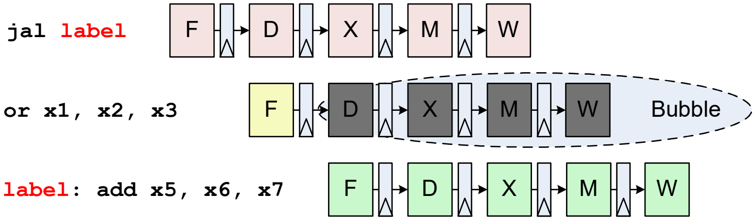
Control Hazards Cont'd

Table 2: Control transfer behavior with suggested performance improvement.

Instruction	Is **Taken** known?	Is **Target Address** known?
JAL	Yes, after Decode	Yes, after Decode
JALR	Yes, after Decode	Yes, after Reg File Read
B<condition>	Yes, after Decode	Yes, after Decode

Control Hazard: JAL Instruction

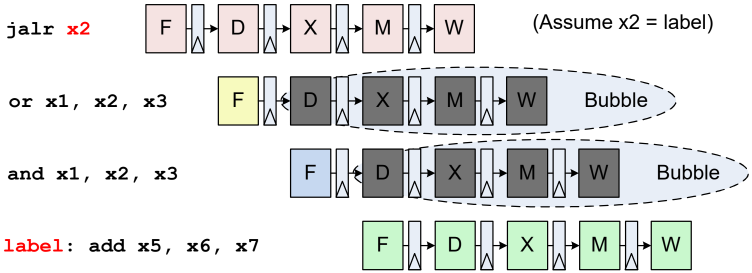
- Jump with PC-relative address, in a simple 5 stage pipeline, costs **one dead** cycle



- A jump or branch instruction kills (not stalls) the following instruction(s)

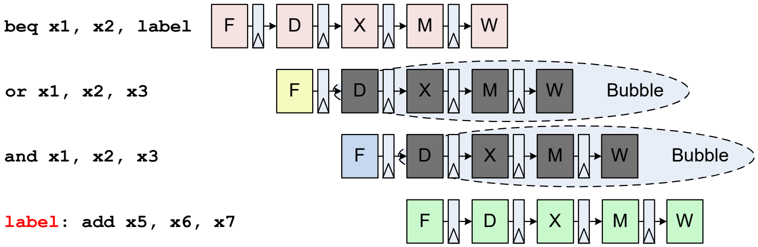
Control Hazard: JALR Instruction

- Jump with register (other then PC) offset costs **two dead** cycles



Control Hazard: Branch Instruction

- Each taken branch costs two dead cycles (one dead cycle with suggested improvement)



Control Flow Penalty

- Control transfer instructions (branches and jumps) roughly account for 15% of the program instructions
- Of the total control transfer instructions, approximately 80% are conditional branches
- Ratio of **taken** and **not taken** branches
  - Loop-closing branches are taken for the first *n*-1 iterations for a loop executing *n* times
  - Remaining branches are taken or not taken with almost equal probabilities
  - Roughly 75 -80% branches are **taken**

Control Flow Penalty Cont'd

- Approximately 12% of the program instructions are conditional branches
- Control transfer penalties are significant for modern processors with deeper pipelines
- Advanced processors may have more than 10 pipeline stages between next PC calculation and branch resolution

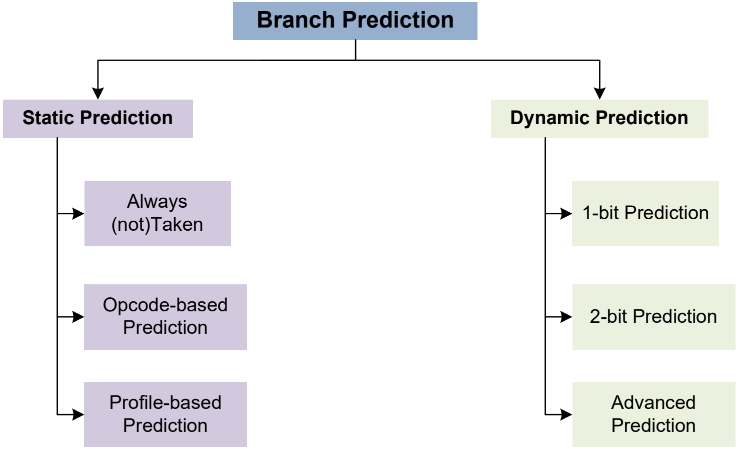
How to Reduce Control Flow Penalty

- Software solutions
    - Reduce the number of branch instructions - loop unrolling (increases the program size)
    - Static branch prediction
- Hardware solutions
    - Stall the pipeline until we know the next fetch address
    - Speculate: Dynamic branch prediction
    - Delayed branching (branch delay slots)
    - Multi-way branching

Branch Prediction

- Used to reduce performance degradation due to branch penalties
- Branch predictors can achieve high prediction accuracy
- Price to be paid:
    - Implementation of prediction structures, branch history table (BHT), branch target buffers (BTB)
    - Recovery mechanism for miss-prediction, kill instructions following branch

Branch Prediction Approaches



Static Branch Prediction

- Always `Not-Taken (NT)`: Simple implementation, no direction prediction, low accuracy
- Always `Taken (T)`: No direction prediction, relatively better accuracy (branches in loops are usually taken)
- Backward Taken forward Not-Taken
- Profile based direction prediction

Accuracy of Branch Prediction

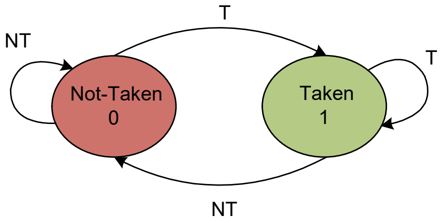
- Needs a performance measure for comparison of different techniques
- Revised CPI with branch miss-prediction:

$$\text{CPI} = 1 + \frac{\text{miss-predictions}}{\text{instruction}} \times \frac{\text{penalty}}{\text{miss-predictions}}$$

Penalty is defined in number of cycles

Dynamic Branch Prediction: 1-bit Prediction

- Simplest history based dynamic branch prediction
- Requires single bit per branch for prediction
- If Taken (T) set the bit (in branch history table)
- State transition for 1-bit prediction



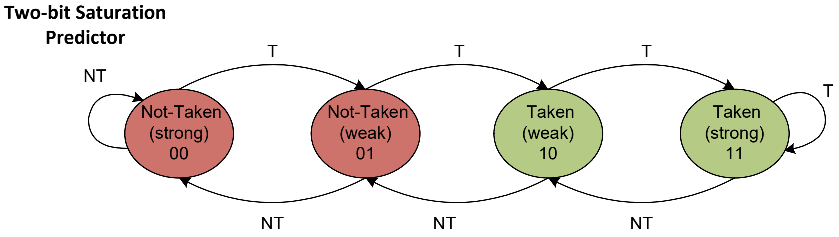
Dynamic Branch Prediction: 1-bit Prediction Limitation

- Reasonable performance for loops with large iterations
- A example illustrating the limitation of 1-bit prediction:

Branch T/NT (Actual)	T	NT	T	NT	T	NT	T	NT
1-bit State	1	0	1	0	1	0	1	0
Prediction	✓	✗	✗	✗	✗	✗	✗	✗

Dynamic Branch Prediction: 2-bit Prediction

- Overcomes the limitation of 1-bit prediction
- 2-bit saturating counter
- Counts up (down) and saturates when Taken (Not-Taken)



Dynamic Branch Prediction: 1-bit vs 2-bit Prediction

Branch T/NT (Actual)	T	NT	T	NT	T	NT	T	NT
1-bit State	1	0	1	0	1	0	1	0
Prediction	✓	✗	✗	✗	✗	✗	✗	✗
2-bit State	11	10	11	10	11	10	11	10
Prediction	✓	✗	✓	✗	✓	✗	✓	✗

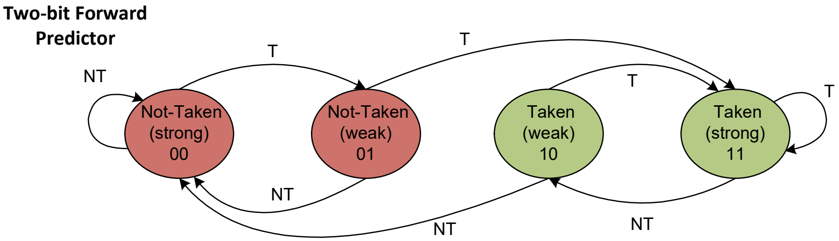
- What if the starting state for 2-bit prediction is 01

Dynamic Branch Prediction: 1-bit vs 2-bit Prediction

Branch T/NT (Actual)	T	T	T	NT	T	T	NT	NT	NT
1-bit State	1	1	1	0	1	1	0	0	0
Prediction	✓	✓	✓	✗	✗	✓	✗	✓	✓
2-bit State	11	11	11	10	11	11	10	01	00
Prediction	✓	✓	✓	✗	✓	✓	✗	✗	✓

Dynamic Branch Prediction: 2-bit Prediction

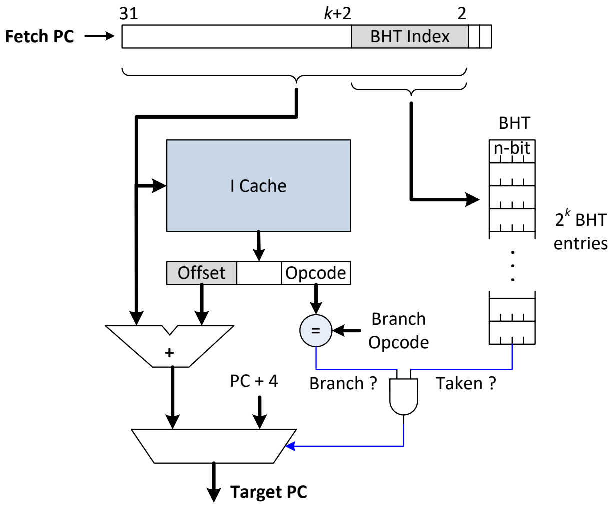
- Another variant for 2-bit prediction



Dynamic Branch Prediction: BHT

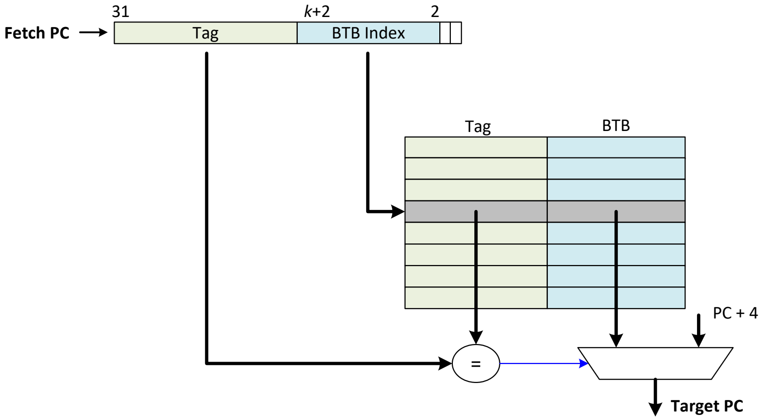
- Branch history table (BHT) keeps record of last $N = 2^k$ branches that are executed by the processor
- Limitations:
  - Only predicts the **branch direction**
  - Does not provide any information about **branch target address**
  - Predicted direction can not redirect instruction fetching until branch target address is calculated

Dynamic Branch Prediction: BHT Cont'd



Dynamic Branch Prediction: BTB

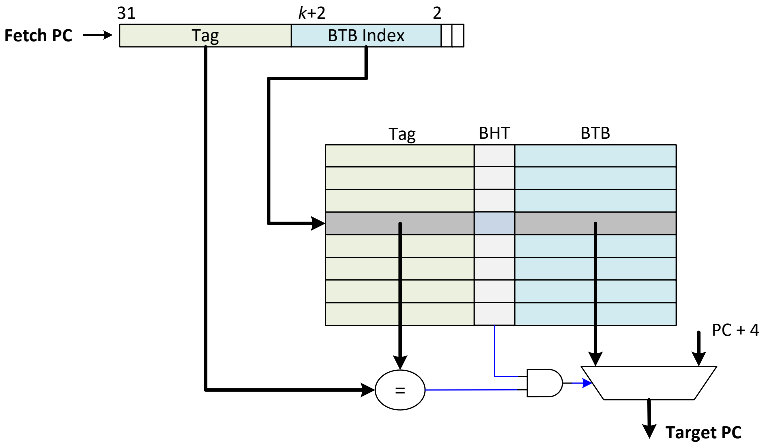
- Branch target buffer (BTB) maintains information about **branch target address**



Dynamic Branch Prediction: BHT & BTB

- Predicted direction, using BHT, is used to redirect instruction fetching
- Branch target address is retrieved from BTB simultaneously
- Both direction prediction as well as target address are fetched (from BHT & BTB, respectively) using current value of PC
- Pipeline is flushed only in case of misprediction

Dynamic Branch Prediction: BHT & BTB



Suggested Reading

- Read relevant sections of Chapter 5 of [Patterson and Hennessy, 2017].

Acknowledgment

- Preparation of this material was partly supported by Lampro Mellon Pakistan.

References

Patterson, D. and Hennessy, J. (2017).
Computer Organization and Design RISC-V Edition: The Hardware Software Interface.
Morgan Kaufmann.