PIPELINED PROCESSOR (III)

Lecturer: Hui Annie Guo
h.guo@unsw.edu.au
K17-501F

Lecture overview

- Topics
  - Design solutions to control hazards
    - Static prediction
    - Dynamic prediction
- Suggested reading
  - H&P Chapter 4.8

Recall: Hazards

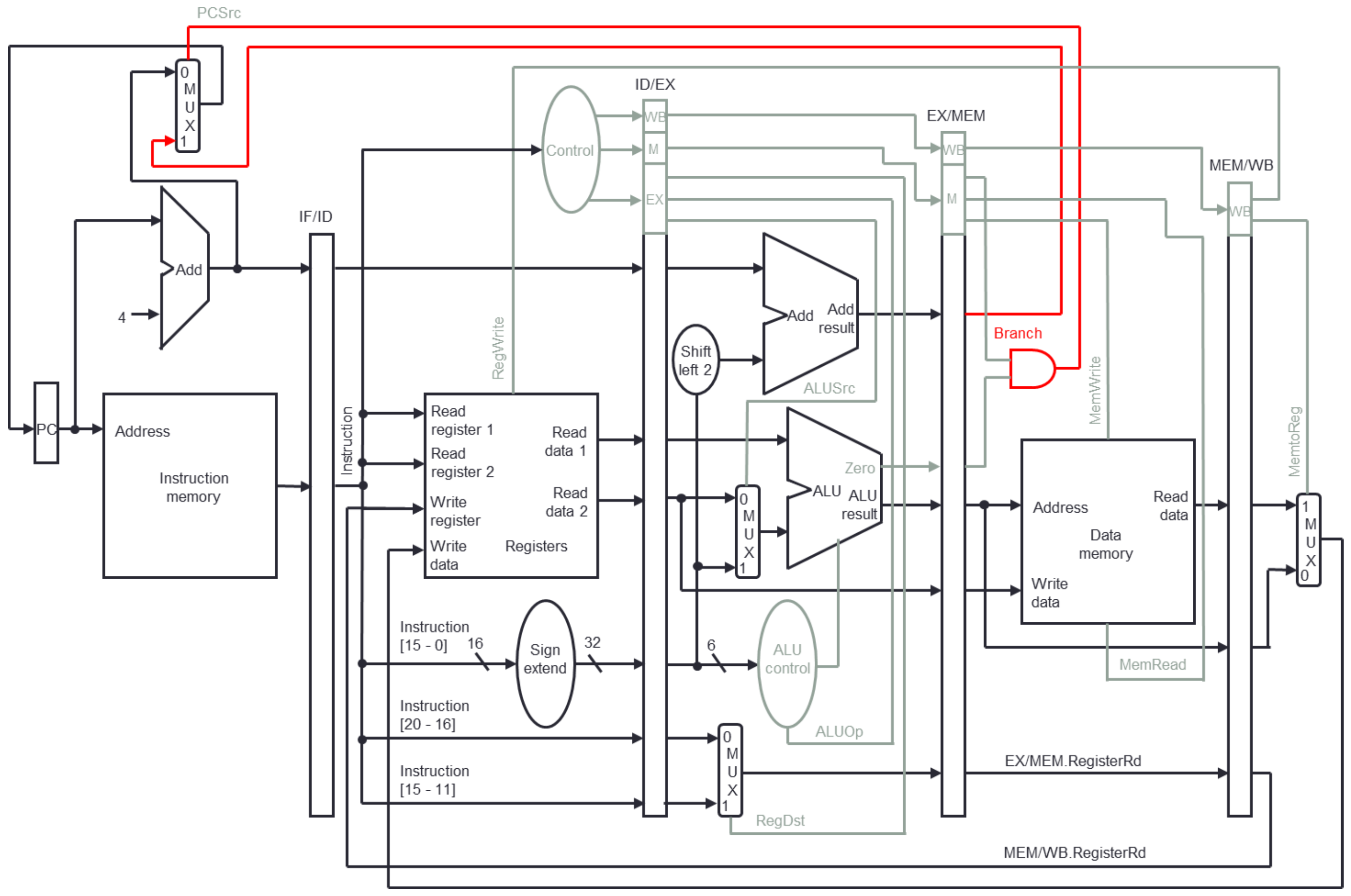
- **Pipeline hazards are the situations where instruction execution cannot proceed in the pipeline.**
  - **Structure hazard**
    - **A required resource is busy.**
  - **Data hazard**
    - **Need to wait for previous instruction to update its data.**
  - **Control hazard**
    - **The control decision cannot be made till the condition check by the previous instruction is completed.**

Control hazard

- **Delay in determining the proper instruction to fetch**
- **Solutions**
  - **Stall**
    - **Wait until the branch decision is clear**
      - A fixed number of clock cycles will be wasted for each branch
  - **Branch prediction**
    - **static**
    - **dynamic**
  - **Branch delay**
    - **Software based**

In-class exercise

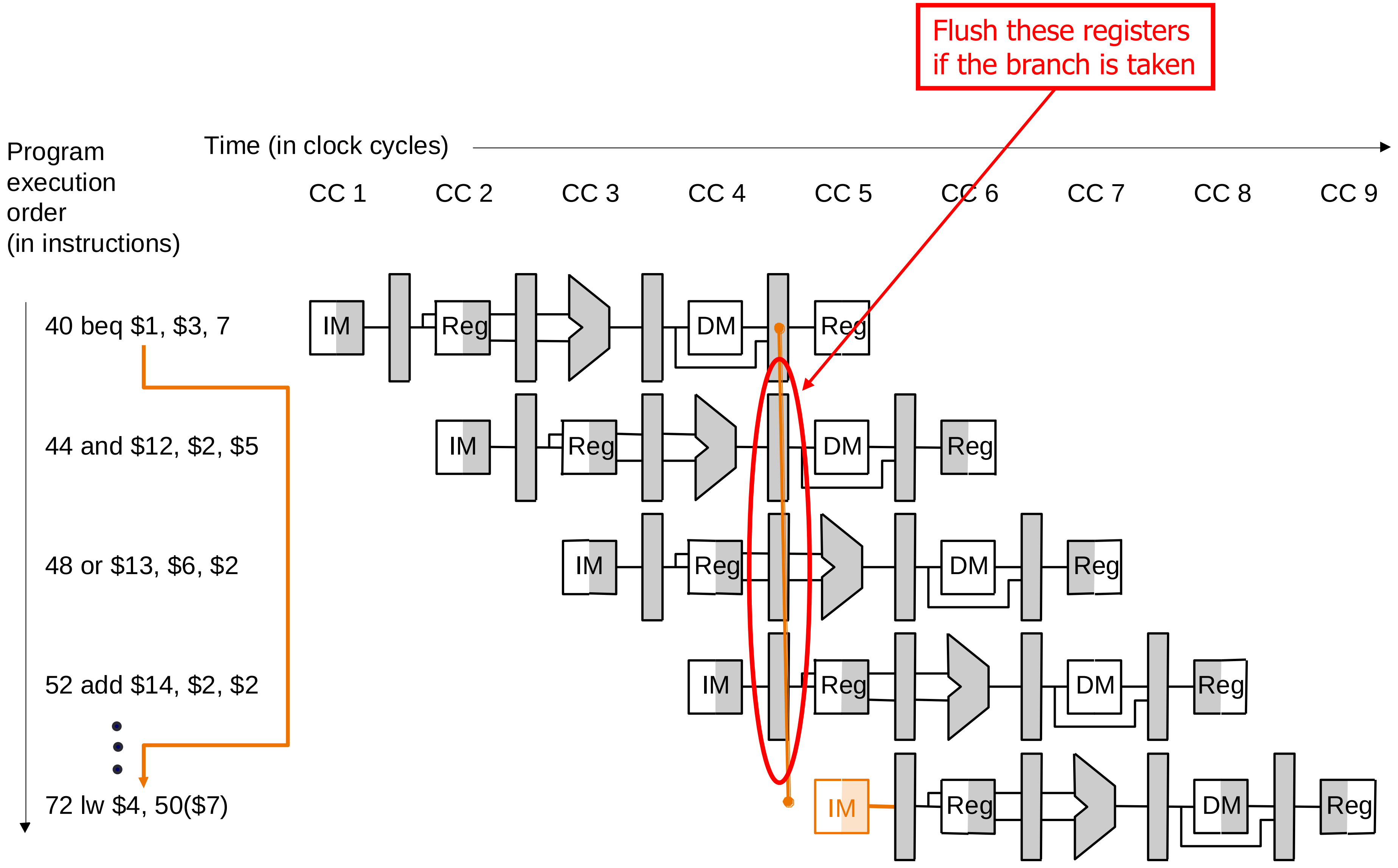
- **Given the pipelined processor we developed so far, as shown below, if stall is used to handle the control hazard, how many clock cycles will be wasted for a branch instruction?**



Static prediction

- **An improved solution to control hazards**
- **It predicts that all branch instructions have the same behavior**
  - **Never taken, or**
  - **Always taken**
- **Two cases**
  - **When the prediction is correct,**
    - **the pipeline continues processing**
  - **When the prediction is wrong,**
    - **the instructions being fetched, decoded, and executed after the branch instruction, must be discarded (flushed)**

Static prediction – branch Not Taken

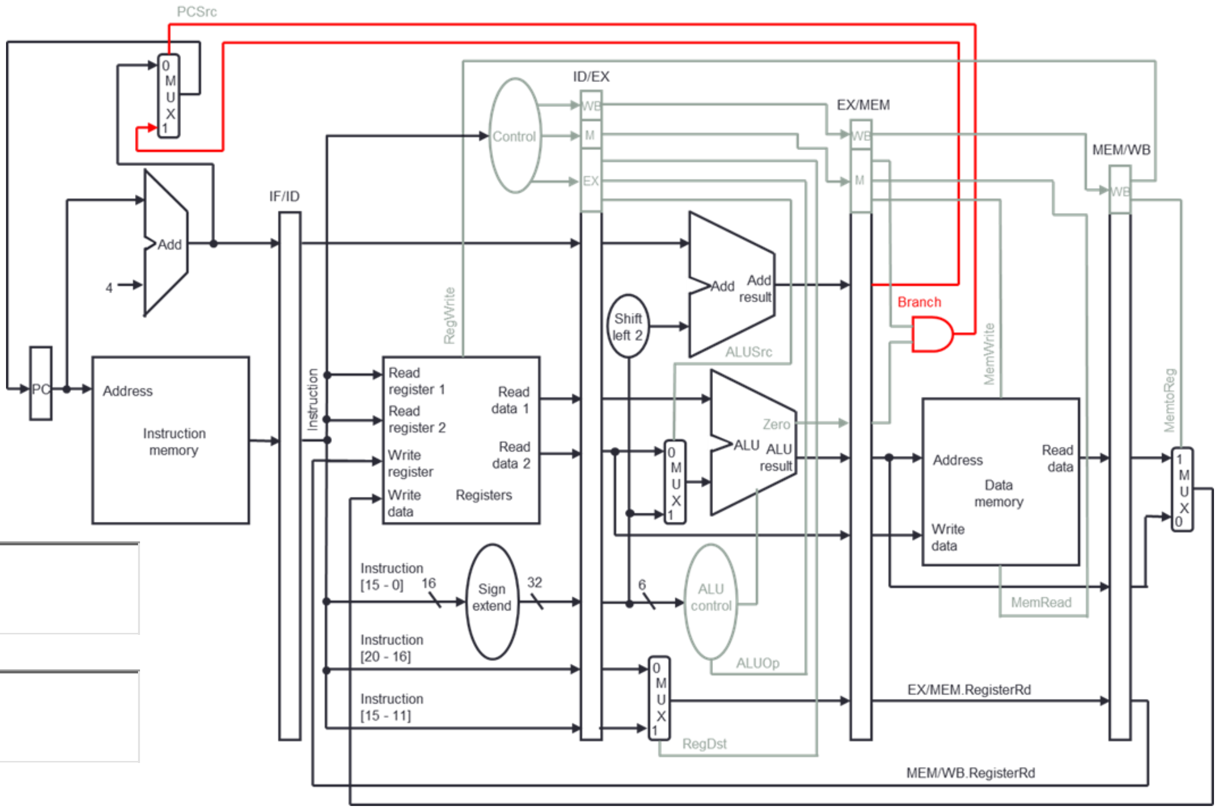


In-class exercise

- **If the accuracy of the static prediction (branch not taken) is x, what is the speedup for branch instructions due to the static prediction?**

$$xN * 1 + (1 - x)N * 4$$

$$\frac{4}{4 - 3x}$$



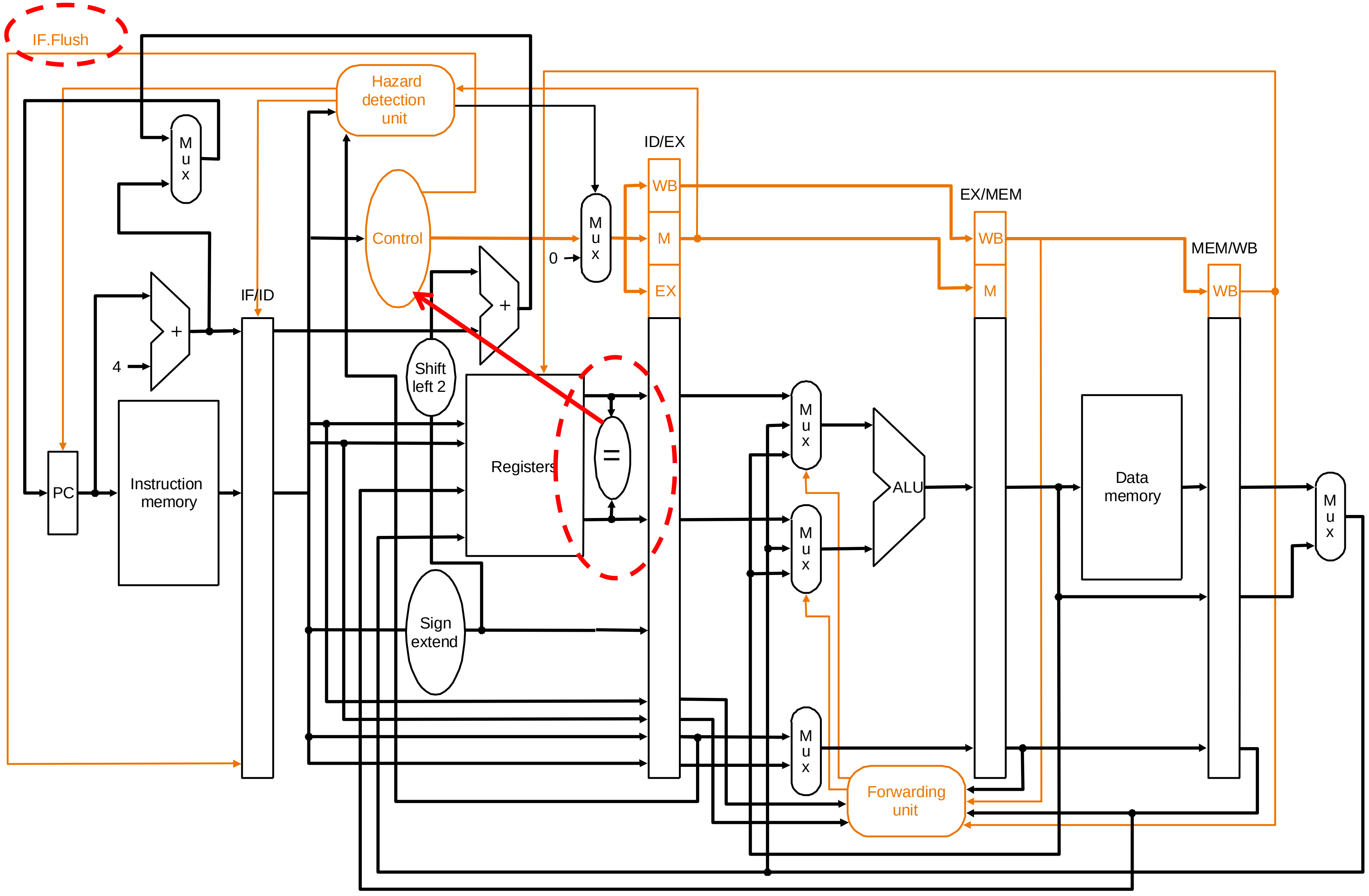
Pipeline flush

- Misprediction leads to pipeline flush
  - 3 instructions needs to be flushed (How?)
    - A heavy penalty!
- Improvement can be made to reduce the flush penalty
  - See next slide

Improvement

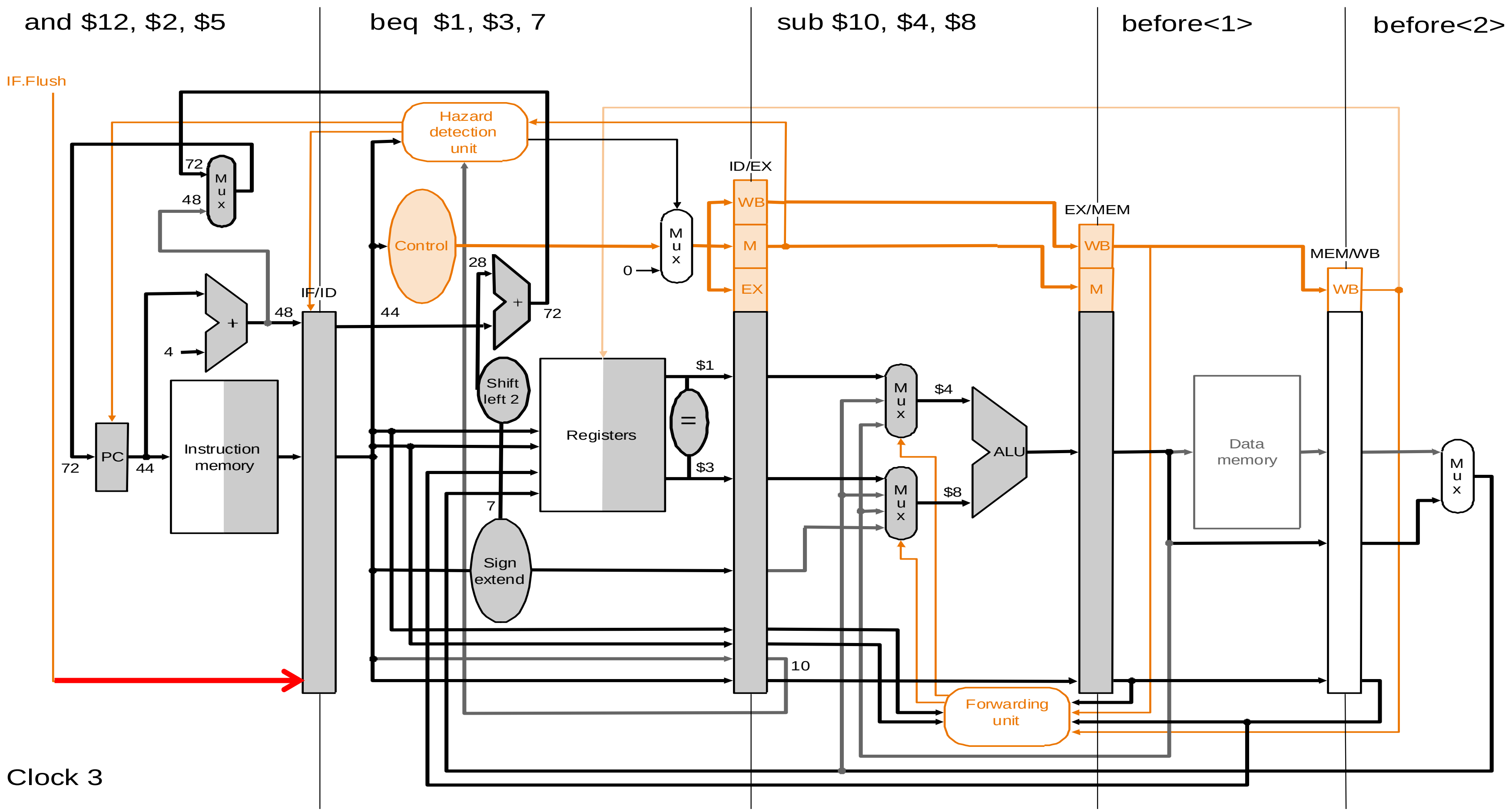
- Make the decision for branch earlier
  - Calculate the target address in ID stage
  - Compare register contents in ID stage
    - use bitwise XOR (FAST, since no carry logic required)
  - Require forwarding to ID stage and hazard detection
    - if the branch is dependent upon the result of a R-type or LOAD instruction that is still in the pipeline
- Only one instruction needs to be flushed
  - The control signal *IF.Flush* is used to flush the instruction in the IF/ID register
    - Sets the instruction field of IF/ID register to 0 (nop)
- See next slide

Static prediction – improved design



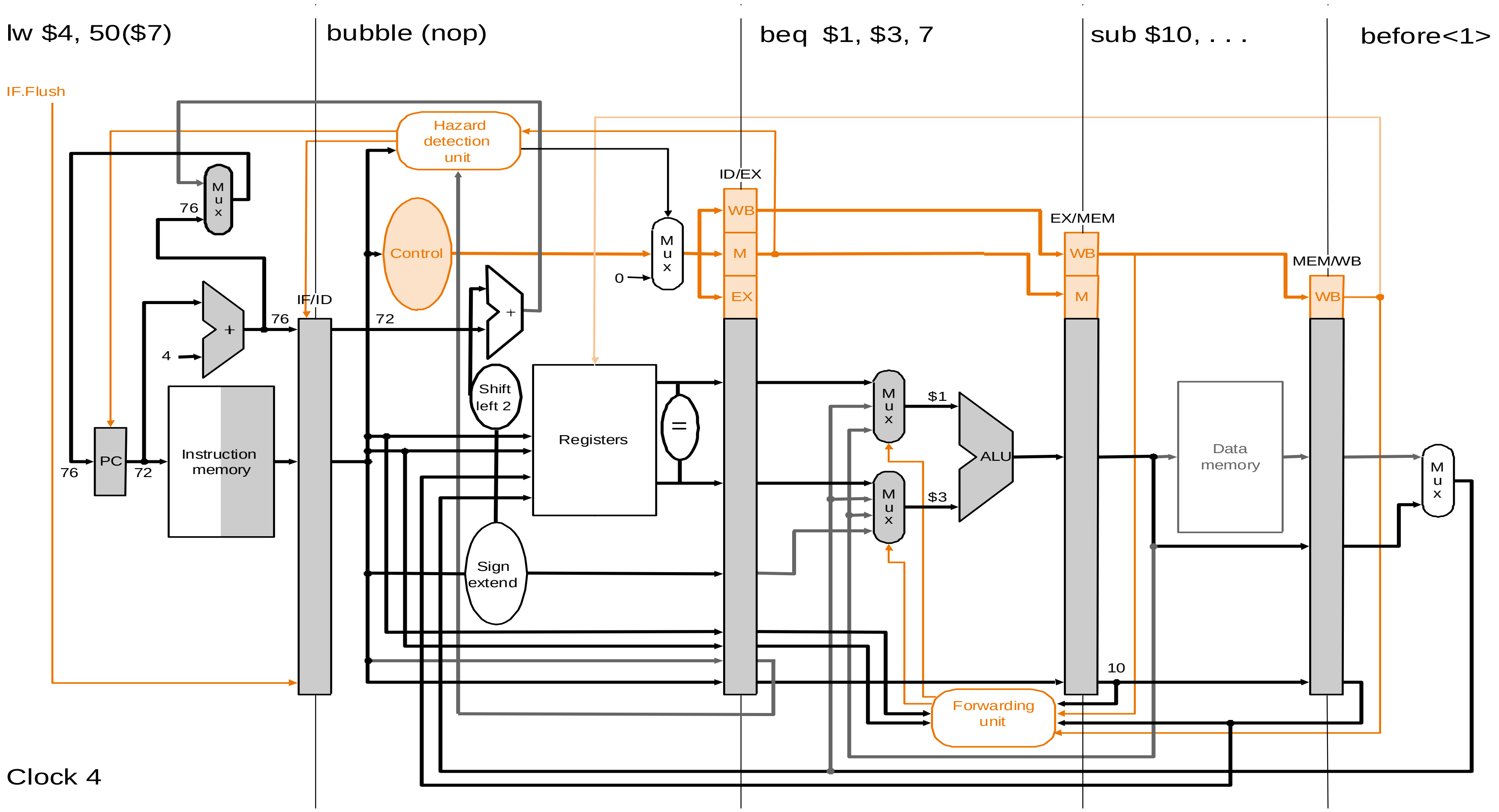
Execution example – static prediction

- **Always not taken**



Execution example – static prediction (cont.)

- **When prediction is wrong**



Clock 4

Dynamic prediction

- **The prediction is made on the fly, depending on the history of the branch behavior**
- **The history is stored in a table or buffer, called *branch history table* or *branch prediction table***
- **The table consists of a portion of the branch instruction address and a branch history bit field**

Branch history table (BPT)

- **Each entry in the table consists of**
  - **Branch address**
  - **Branch history**

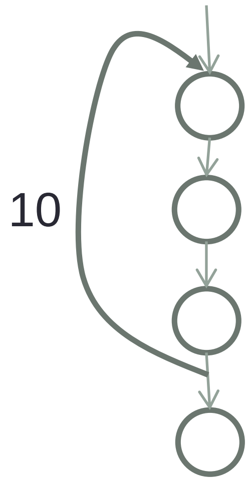
branch addr	branch history

1-bit prediction scheme

- The history bit field contains only 1 bit
  - Representing the last branch behavior
    - 1 for Taken
    - 0 for Not Taken
- Prediction just follows the history
  - If history bit is 1, the prediction is Taken; If history bit is 0, the prediction is Not Taken.
- If the prediction is wrong, the history bit will be changed.

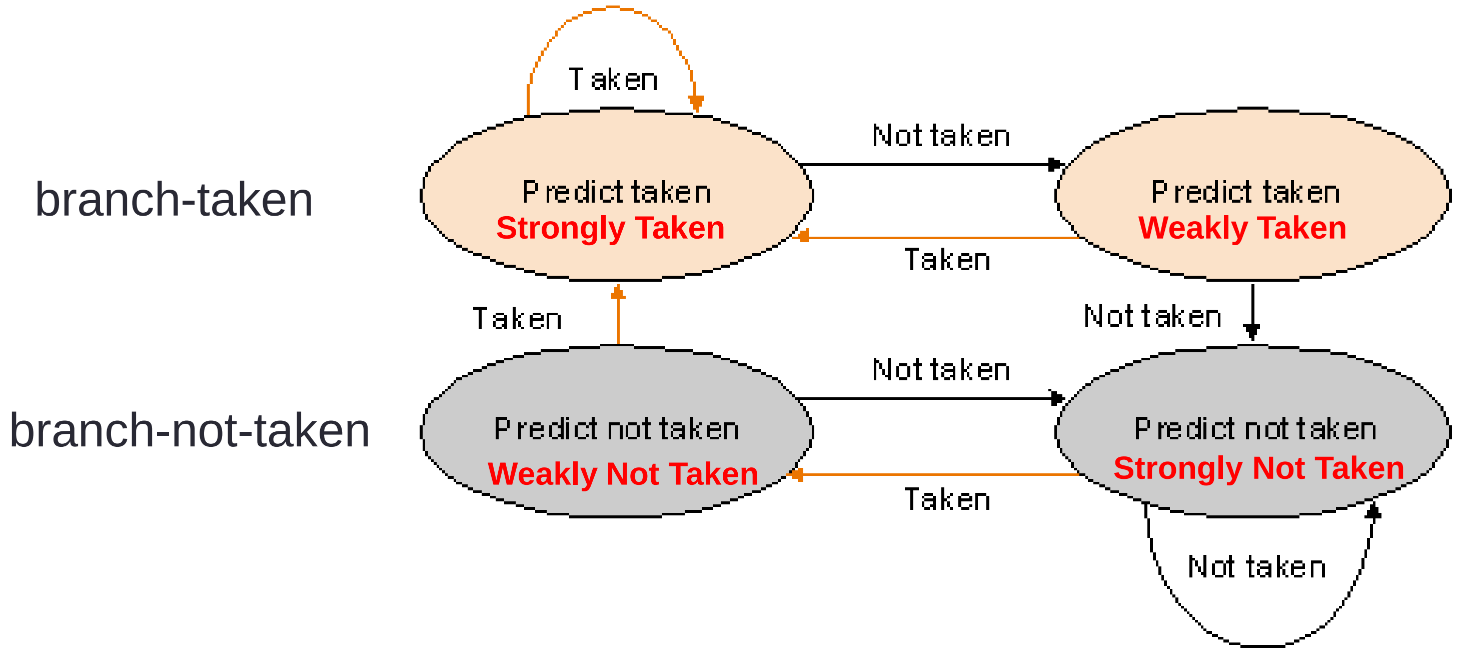
1-bit prediction scheme (cont.)

- **Advantage**
  - **Simple**
- **Disadvantage**
  - **Limited prediction accuracy**
  - **For example:** double mis-prediction.
    - **Consider a nested loop.**
      - With the 1-bit scheme, we always predict incorrectly when we exit the loop. But when we execute loop again, we mispredict on the first iteration since the history bit was set to “not taken” at the end of the last execution



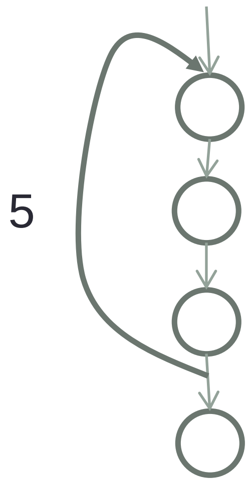
2-bit prediction scheme

- **Overcome the double misprediction problem**
- **The history field has two bits**
- **A prediction must be wrong twice before it is changed**
- **Only mispredict once per loop execution**



In-class exercise

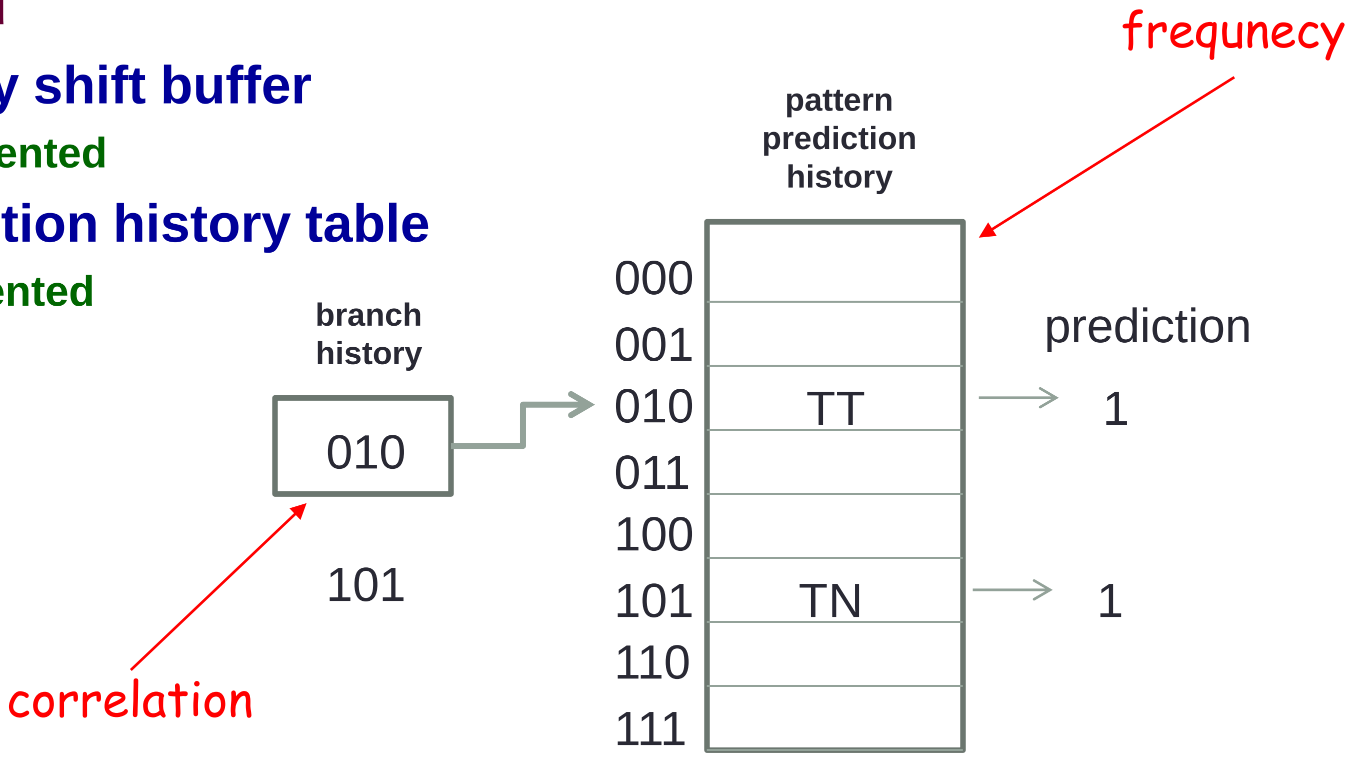
- **For the execution below, what predictions will the 2-bit branch prediction scheme generate? What are the possible values in the branch history table for the execution? Assume, the history table is initially set to**
  1) **Not-Taken**
  2) **Taken**



n-bit prediction scheme?*

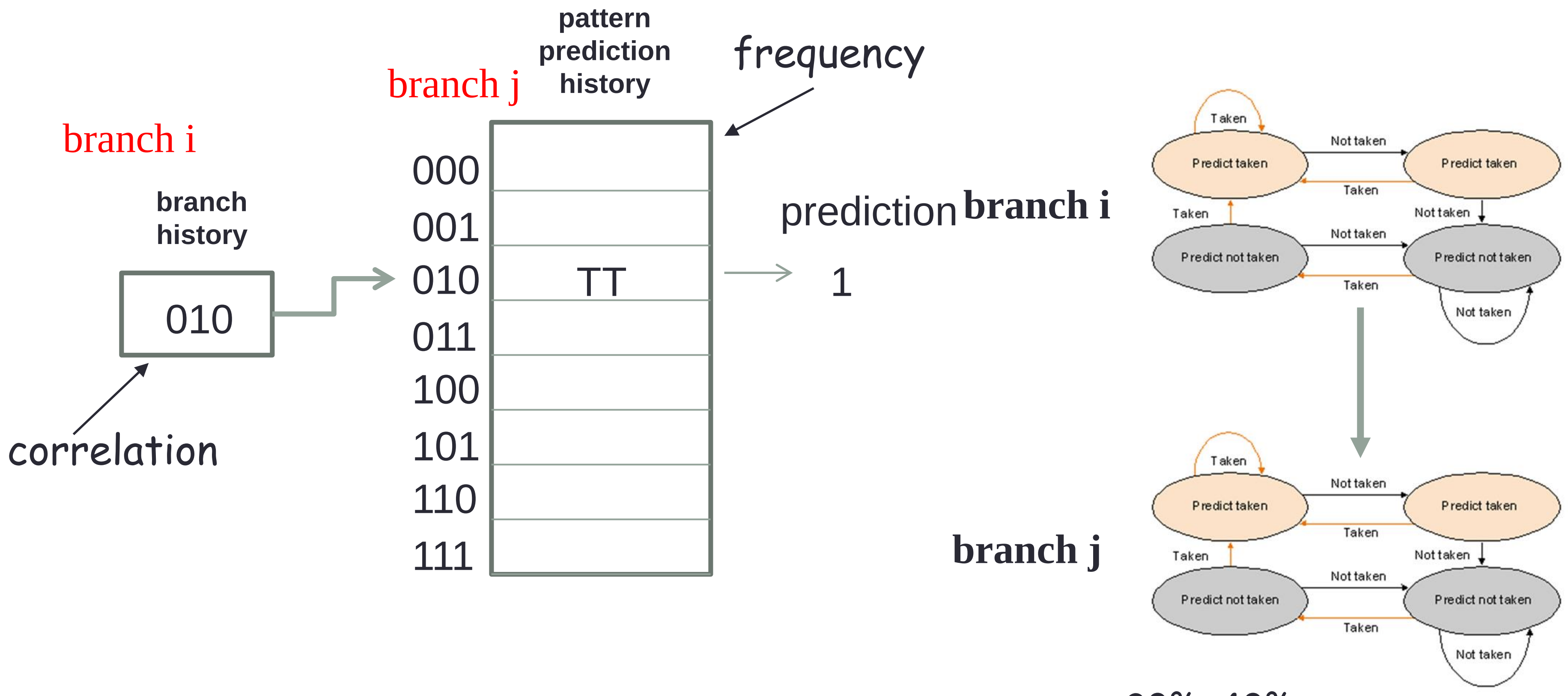
- **Frequency based**
  - **No correlations are taken into account**

    f(N)>0.5→N
    f(T)>0.5→T
- **Pattern based**
  - **Branch history shift buffer**
    - **Correlation oriented**
  - **Pattern prediction history table**
    - **Frequency oriented**



More on correlating prediction*

- **Branch prediction of one branch is correlated to the branch history of another branch**

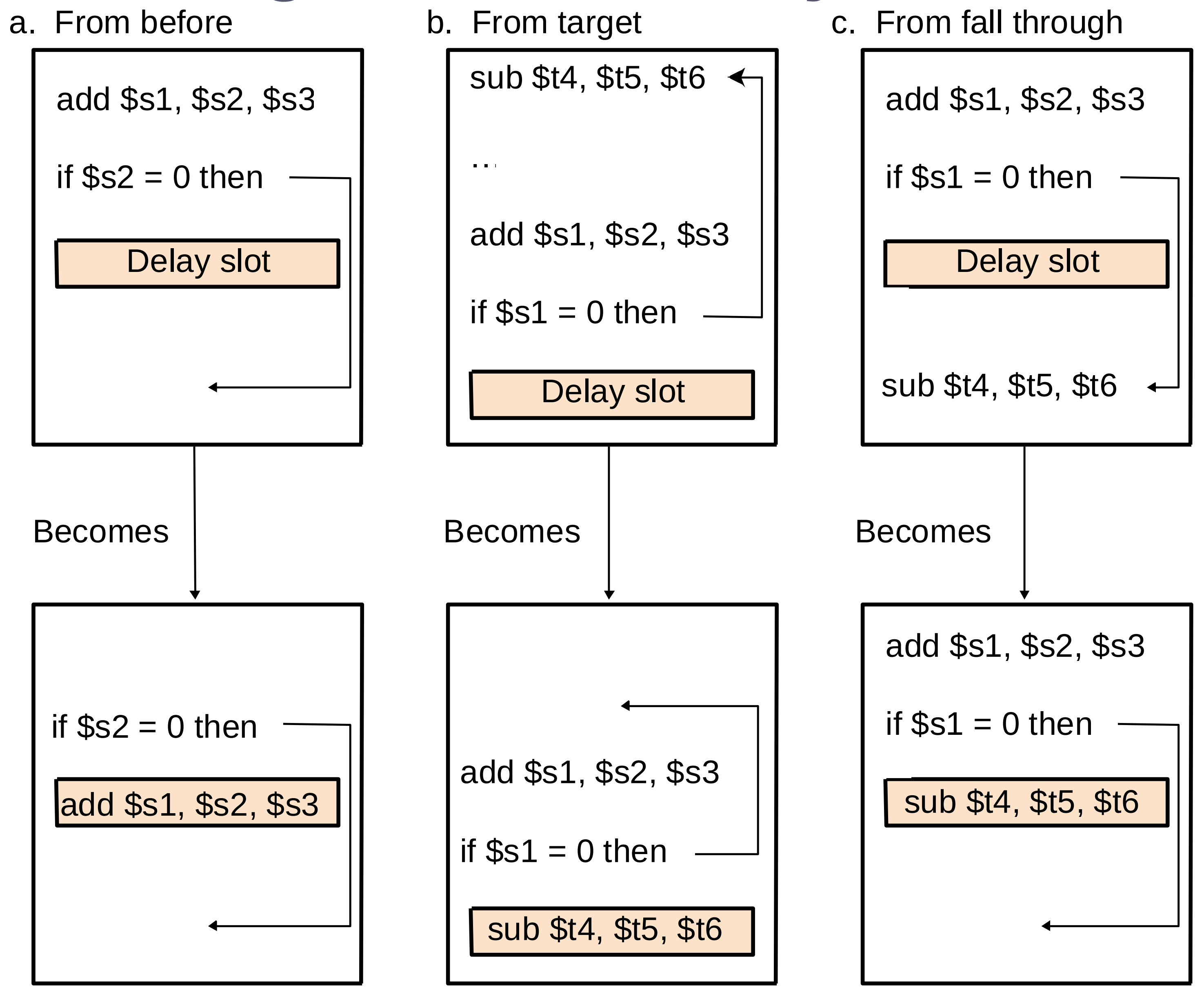


accuracy 20%, 40%, ….

Delay branch*

- A compiler-oriented approach
- Avoid the uncertainty of a branch decision by inserting an independent instruction (immediately after the branch instruction) that can always be executed irrespective of the branch decision (does not need to be flushed)
- The idea is demonstrated in the next slide.

Scheduling a branch delay slot*



Pipelining summary

- What is pipelining?
  - Allows for multiple instructions to be executed in a processor datapath simultaneously
- Pipelined processor vs single cycle processor
  - Similarities
    - CPI = 1 (ideal pipelining case)
    - Control signals are generated with a combination logic
  - Differences
    - Clock cycle time of the pipelined processor is smaller
    - Control signals of the pipelined processor need to be pipelined

Pipelining summary (cont.)

- **Pipelined processor vs multi-cycle processor**
  - **Similarities**
    - **Same (similar) clock cycle time if the same datapath partitioning is used**
    - **Multiple cycles per instruction**
  - **Differences**
    - **Multiple instructions can be concurrently executed in the pipelined datapath while only a single instruction executed in the multi-cycle datapath**
    - **CPI (MC) > CPI (pipe)**

Pipelining summary (cont.)

- Problems with pipelining
  - Resource competition
    - Structural hazards
  - Dependencies between instructions
    - Data hazards
    - Control hazards
- HW solutions
  - Stall
    - A basic way
  - For structural hazards
    - Resource replication
      - E.g. IMEM and DMEM
    - Structural design to avoid potential resource competition
      - E.g. Bypass used in MEM stage

Pipelining summary (cont.)

- **HW solutions**
  - **For data hazards**
    - **Forwarding, e.g**
      - from EX/MEM
      - from MEM/WB
    - **Stall + forwarding, e.g.**
      - Load-use: stall read instruction one clock cycle then forwarding data from MEM/WB
  - **For control hazards**
    - **Evaluate and calculate target address earlier**
      - e.g. in ID stage
    - **Prediction**
      - Branch taken/not taken (static and dynamic)
      - Flush if prediction wrong

About lab …