Quiz 1 Handout

Figure 1 shows the pipeline of an out-of-order machine that uses **Data-in-ROB design** to perform out-of-order execution and in-order commit. Flip flops and queues represent stage boundaries.

The processor consists of the following stages:

1. Fetch: The instruction at PC is fetched from the instruction cache.
    - In parallel, the PC is also fed into a branch target buffer (BTB). On a hit in the BTB, the next PC to be fetched is updated to the target PC indicated in the BTB. Otherwise, the next PC is PC+4. The BTB is a 1-cycle "tight loop".
2. Decode: The fetched instruction is decoded.
    - If the decoded instruction was a conditional branch, its direction is predicted by a branch predictor. The branch predictor is described in the next page.
      Note: Direct jumps (J/JAL) are always taken, so no prediction is needed.
    - For direct jumps and branches (BEQ/BNE/J/JAL), the target is calculated and a prediction on branch direction is made. If the branch is predicted taken, the next PC to be fetched is updated unless the BTB made a correct prediction in the Fetch stage.
3. Pre-Allocation: The reorder buffer (ROB) is checked for an available slot.
4. Register Read & Allocate: If the needed space is available, the instruction is inserted into the ROB. The physical index of the ROB entry is the instruction's "tag". To obtain any required operands, the rename table and register file are read simultaneously. If the rename table has a valid tag for an operand, then the corresponding ROB entry must be checked for that operand. Otherwise, the value in the register file can be used. If the instruction writes a register, its tag is written to the destination register entry in the rename table. *The ROB source fields store either the tag of the data-producing ROB entry, or the actual data when it becomes available***.**
5. Issue: On each cycle, the oldest ready instruction in the ROB is issued, reading its operands from its ROB entry.
6. Execute: Functional units or the memory system may take one or more cycles to execute the instruction.
7. Writeback: The output from the functional units, or memory access, if any, are written back to the `data` field in the ROB and the `pd` bit is set. Additionally, any dependent instructions in the ROB will receive the value and have the corresponding present bit set (`p1` for the first operand, `p2` for the second operand).
8. Commit: On each cycle, if the oldest instruction in the ROB that has finished execution is committed. If the instruction writes a register, the result is written to the register file, and if the tag in the rename table for this register matches the tag of the result, the rename table valid bit is cleared.

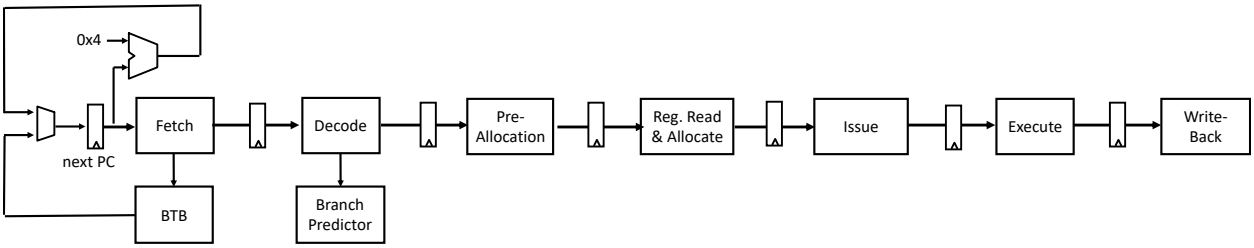

Figure 1: Simplified out-of-order pipeline schematic. Several important structures are not shown, such as commit, bypassing, and some sources of next-PC value

Local History Branch Predictor:

The branch predictor for this processor is a *local history predictor* that consists of a local history table and eight 2-bit prediction counters. The local history table consists of eight shift registers, each of which contains an 8-bit history of a particular branch. To make a prediction, we take the bottom three bits of the PC (excluding the last 2 bits which are always 00 for aligned instructions) to index into the local history table. The bottom 3-bits of each entry in the local history table are then used to index into the prediction counters.

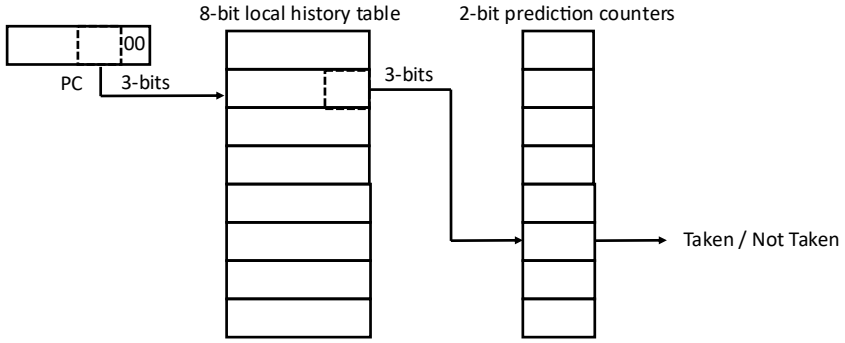

Figure 2: local history branch predictor

In each local history table entry, 1 represents **Taken** and 0 represents **Not-Taken**. The 2-bit counters in this design follow the state-diagram shown in Figure 3. In state **1X**, we will guess **Taken**; in state **0X**, we will guess **Not-Taken**.

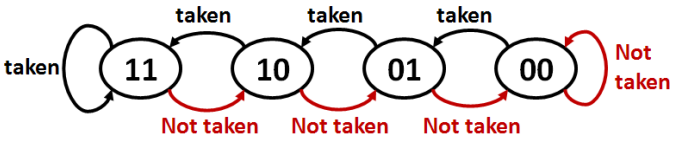

Figure 3: State Diagram of 2-bit counters

Upon predicting a branch direction, the local history table entry is immediately updated in the same cycle by shifting in the prediction from the right. The prediction counter is updated when the corresponding branch commits. The predictor also has a structure (not shown in figure) that tracks which local history table entry each branch updated. Upon detecting a misprediction later down the pipeline, the local history table is recovered by right-shifting the table entries that the mis-predicted branch and all later branches in program order have updated. After recovery, the correct direction is shifted in.

Processor State

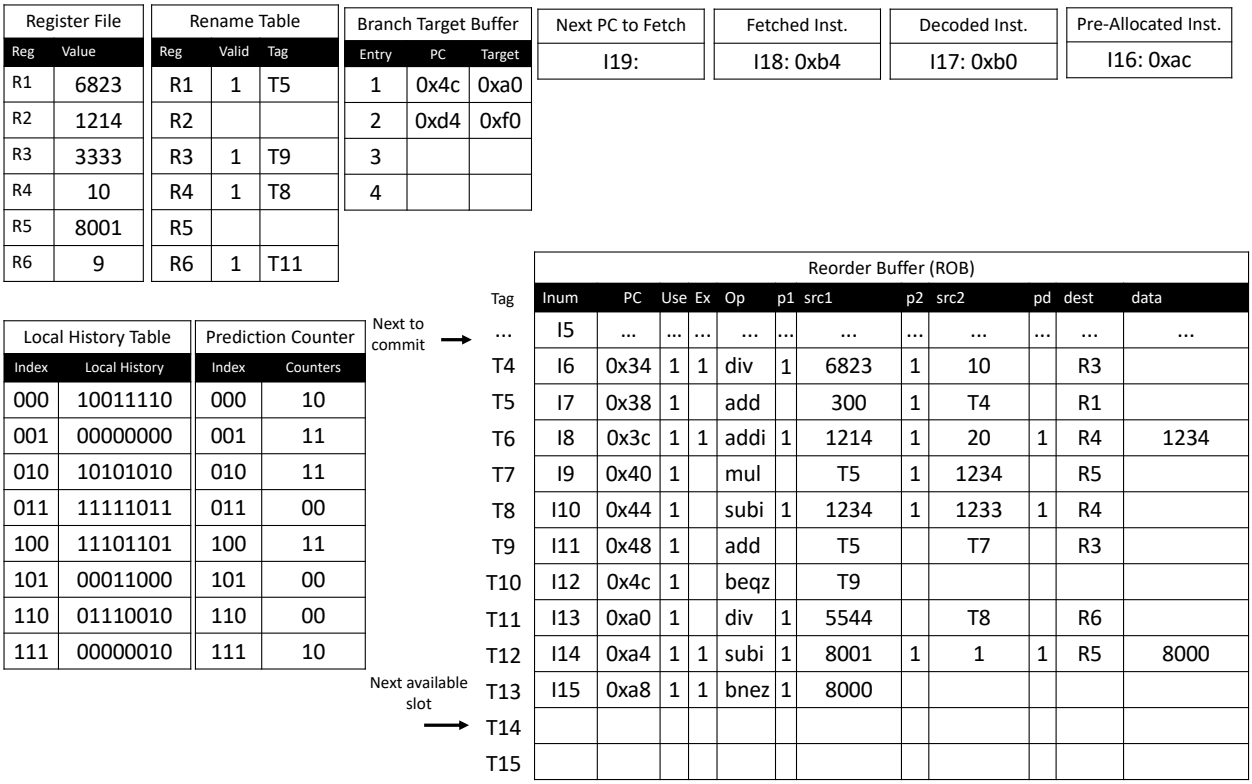

Figure 4: Processor State. There are 26 additional rename table entries and registers, which are not shown.

A snapshot of the processor state is shown in Figure 4. **For simplicity, we do not show interactions between the processor and the memory subsystem.** The processor state consists of the following components:

- **Pre-Allocated Instruction**: Pipeline register holding the next instruction to be allocated to the ROB.
- **Decoded Instruction**: Pipeline register holding a decoded instruction.
- **Fetched Instruction**: Pipeline register holding a raw binary instruction.
- **Next PC to be fetched**: This is the PC register in Figure 1.
- **Branch Target Buffer (BTB):** Holds map of source PC to target PC. If a fetched instruction PC hits in the BTB, the next PC to fetch is the corresponding target PC.
- **Local History Table:** Holds local histories for different branches
- **Prediction Counter:** Provides prediction for a given branch history
- **Branch Global History**: 8-bit global branch history.
- **Register File**: Holds the committed data values of architectural registers. A snapshot is taken every time a branch is allocated into the ROB.
- **Rename Table:** A map from architectural register name to ROB tag (if valid).

- **Reorder Buffer (ROB)**: Contains the bookkeeping information for managing the out-of-order execution and register renaming, and operand data values when they become available.

We provide a list of actions below. Study them carefully and relate them to the concepts covered in the lectures. You will be required to associate events in the processor to one of these actions, and, if required, one of the choices for the blank.

Label List:

A. Satisfy a dependence on _____ by stalling
B. Satisfy a dependence on _____ by bypassing a speculative value
C. Satisfy a dependence on _____ by bypassing a committed value
D. Satisfy a dependence on _____ by speculation using a static prediction
E. Satisfy a dependence on _____ by speculation using a dynamic prediction
F. Write a speculative value using lazy data management
G. Write a speculative value using greedy data management
H. Speculatively update a prediction on _____ using lazy value management
I. Speculatively update a prediction on _____ using greedy value management
J. Non-speculatively update a prediction on _____
K. Check the correctness of a speculation on _____ and find a correct speculation
L. Check the correctness of a speculation on _____ and find an incorrect speculation
M. Abort speculative action and cleanup lazily managed values
N. Abort speculative action and cleanup greedily managed values
O. Commit correctly speculated instruction, where there was no value management
P. Commit correctly speculated instruction, and mark lazily updated values as non-speculative
Q. Commit correctly speculated instruction, and free log associated with greedily updated values
R. Illegal or broken action

Blank choices:

i. Register value
ii. PC value
iii. Branch direction
iv. Memory address
v. Memory value
vi. Latency of operation
vii. Functional unit