Quiz 2 Handout

Figure 1 shows the pipeline of an out-of-order machine that uses **reservation stations** 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.
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 updates the next PC to be fetched unless the branch predictor predicts not-taken.
3. Pre-Allocation: Several structures are checked for space that will be needed by the instruction:
    - Load and store instructions check for an available slot in the memory reservation station. All other instructions check for an available slot in the arithmetic reservation station.
    - For store instructions, an entry in the store buffer.
    - For load instructions, an entry in the load buffer
4. Register Read & Allocate: If all the needed space is available, the instruction is inserted into the appropriate reservation station. Loads and stores grab an entry in the load buffer and store buffer, respectively. The physical index of the instruction's reservation station entry is the instruction's "tag" (e.g., the first entry in the memory reservation station has tag M1). 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 all reservation station entries 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 source fields of reservation stations store either the tag of the instruction producing the data (i.e., the producer instruction's reservation station entry), or the actual data when it becomes available***.**
5. Issue: On each cycle, the oldest ready instruction in each reservation station is issued, reading its operands either from the reservation station or the register file.
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 reservation station and the `pd` bit is set. Additionally, any dependent instructions in the reservation stations will receive the value and has the corresponding present bit set (`p1` for the first operand, `p2` for the second operand).

8. Commit: On each cycle, if the oldest instruction among all reservation stations has finished execution, it 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.

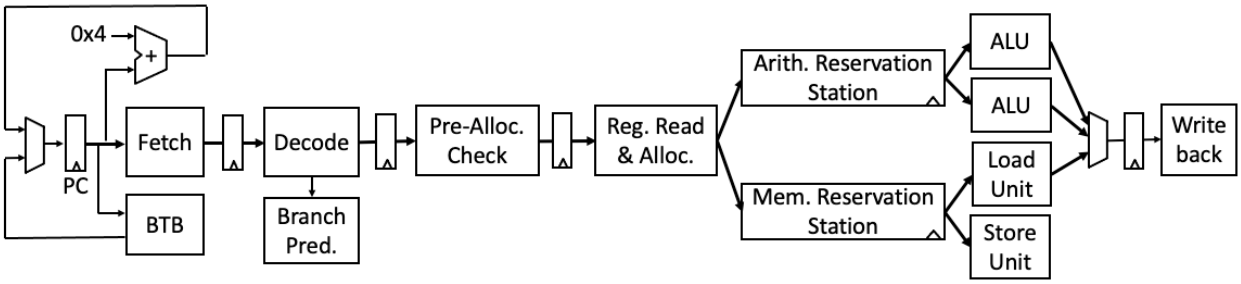

Figure 1: Simplified out-of-order pipeline schematic. Several important structures are not shown, such as commit, bypassing, and some sources of next-PC value

***gshare* Branch Predictor:**

The Branch Predictor used in this processor is called ***gshare***, which uses **exclusive OR (XOR)** to combine the global history and the PC. The gshare branch predictor takes the lower three bits from the global history and the lower three bits from the PC (excluding the last 2 bits which are always 00 for aligned instructions), and calculates an index into an array of eight two-bit counters by exclusive OR-ing them (Figure 2).

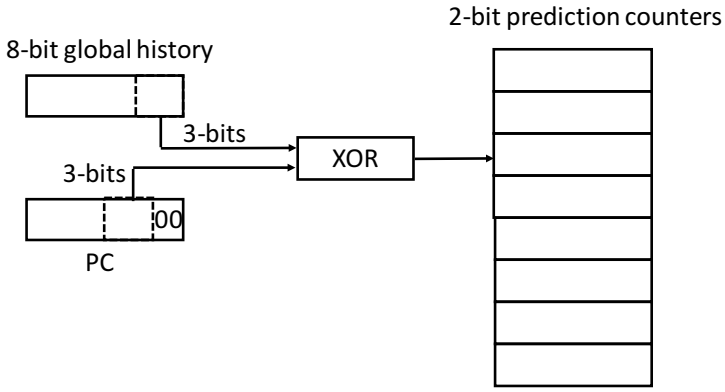

Figure 2: gshare branch predictor

In the global history, 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**.

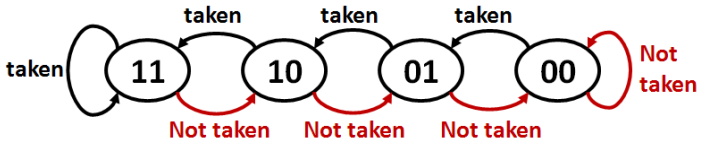

Figure 3: State Diagram of 2-bit counters

Processor State

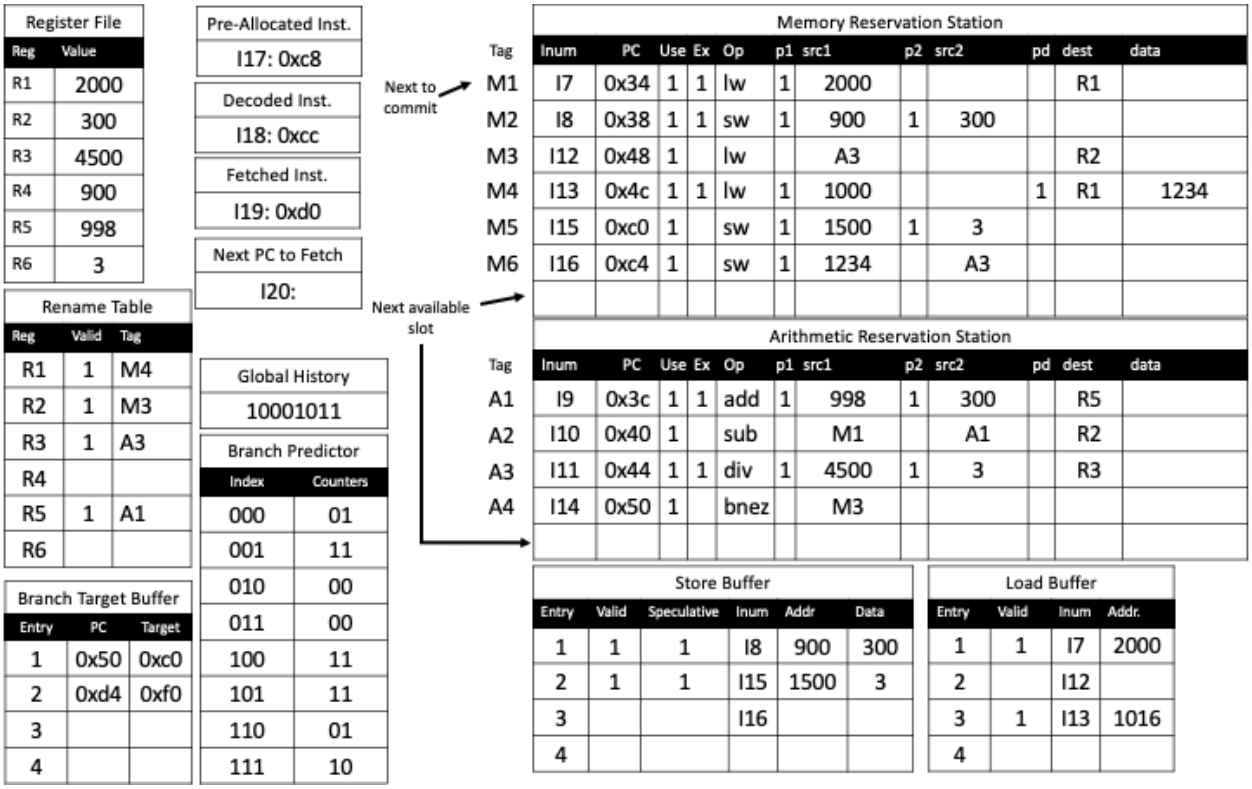

Figure 4: Processor State. There are 26 additional rename table entries and registers, which are not shown.

A snapshot of the processor state is shown in Figure 4. It consists of the following components:

- **Pre-Allocated Instruction**: Pipeline register holding the next instruction to be allocated to the reservation stations.
- **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.
- **Branch Predictor (BHT)**: 2-bit counters for branch prediction.
- **Branch Global History**: 8-bit global branch history.
- **Register File**: Holds the committed data values of architectural registers.
- **Rename Table:** A map from architectural register name to reservation station tag (if valid).
- **Memory Reservation station**: Contains the bookkeeping information of all load and store instructions for managing the out-of-order execution and register renaming, and operand data values when they become available.
- **Arithmetic Reservation Station**: Contains the bookkeeping information of all instructions except load and store instructions. Information is used for managing the out-of-order execution and register renaming, and operand data values when they become available.

- **Store Buffer**: The address and data from an executed SW instruction are temporarily kept here, and then moved to the cache after the instruction commits or cleared if the instruction is aborted.
- **Load Buffer**: The address from an executed LW instruction is temporarily kept here, and cleared after the instruction commits or is aborted.

For `SW` instructions, assume the first operand (`src1`) provides the base register for the store address, and the second operand (`src2`) provides the data source for the store.

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