Quiz 2 Handout

Figure 1 shows the pipeline of an out-of-order machine. Flip flops represent stage boundaries. Blocks in parallel to each other represent parallel operations occurring within the same stage. *This machine uses a Data-in-ROB design.*

The processor consists of the following stages:

1. **Instruction Fetch**: The instruction at PC is fetched from the instruction cache/memory.
    - The PC is also fed into a branch target buffer (BTB), which stores mappings from source PC to target PC. On a hit in the BTB, the next PC to be fetched is updated as the target PC indicated in the BTB.
2. **Instruction Decode**: The 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 branches (BEQZ/BNE/J/JAL), the branch target is calculated by a branch target calculator and updates the next PC to be fetched according to the prediction, if required.
3. **Pre-Dispatch Check**:
    - The reorder buffer (ROB) is checked for an available slot.
    - For store instructions, the store buffer is checked for an available slot.
    - For load instructions, the load buffer is checked for an available slot.

   The ROB slot index is this 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.
4. **Dispatch**: The instruction is inserted into the ROB only if *all* the checks in the previous cycle (Pre-Dispatch Check) pass. *The ROB source fields store either the tag of the data-producing ROB entry, or the actual data when it becomes available***.**
5. **Execute**: The ROB issues an instruction whose operands are all present.
6. **Write-Back**: 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 reorder buffer will receive the value.
7. **Commit**: Instructions are committed in-order. 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.

Note that not all sources, and not all control logic for next PC are shown in Figure 1 for simplicity.

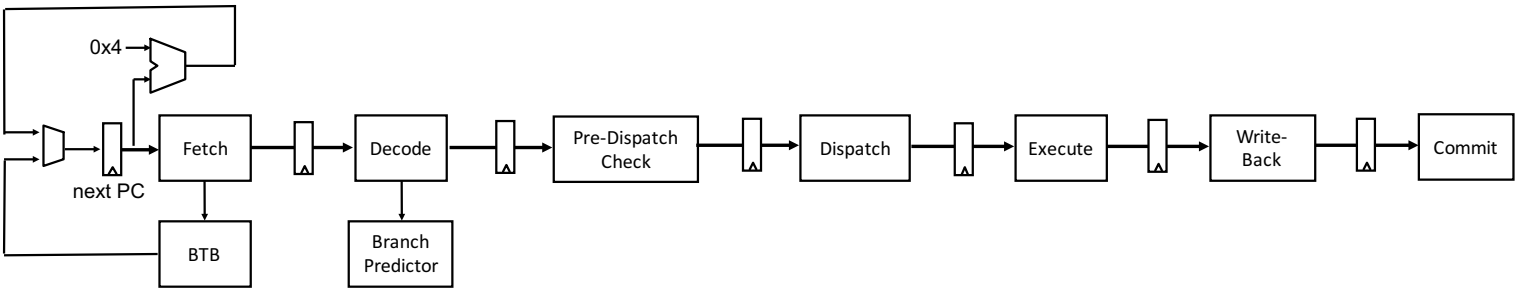

Figure 1: Out-of-order pipeline

***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 the two-bit counters by exclusive OR-ing them (Figure 2).

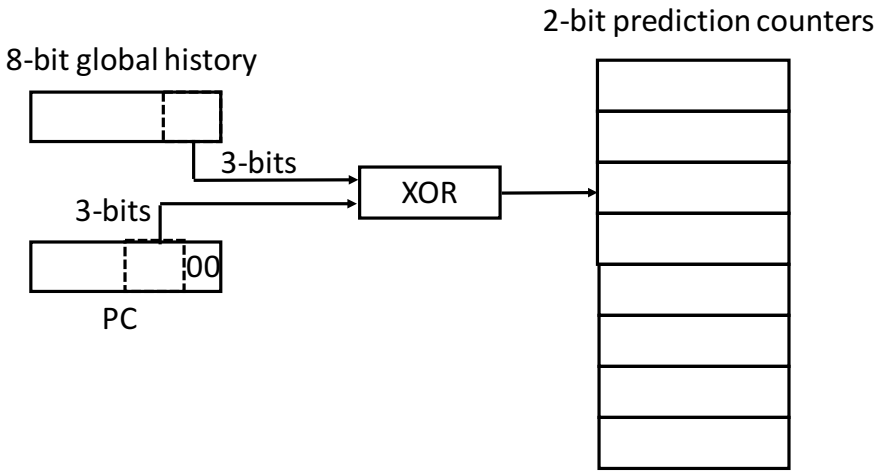

Figure 2: gshare branch predictor

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**.

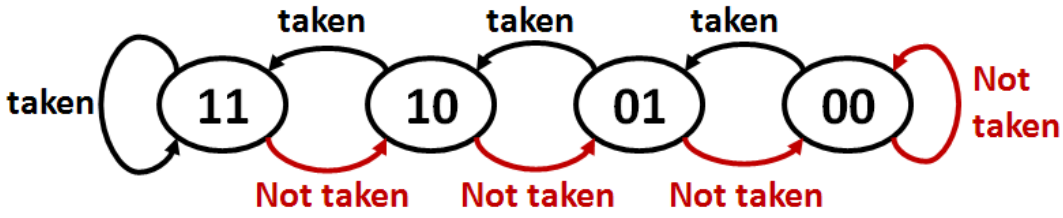

Figure 3: State Diagram of 2-bit counters

Processor State

Prediction Counter	
Index	Value
000	01
001	11
010	00
011	00
100	00
101	11
110	01
111	10

Global History
00011010

Branch Target Buffer		
Entry	PC	Target
1	0x2c	0xa0
2		
3		
4		

Decoded Inst. Queue
I17: 0xb8

Fetched Inst. Queue
I18: 0xbc

Next PC to Fetch
I19:

Store Buffer					
Entry	Valid	Speculative	Inum	Addr	Data
1	1	0	I2	3800	129
2	1	1	I6	4004	692
3	1	1	I9	4168	4000
4			I12		

Load Buffer			
Entry	Valid	Inum	Addr.
1	1	I1	2008
2	1	I5	1004
3		I11	
4	1	I15	4100

Register File	
Reg	Value
R1	25
R2	1234
R3	3980
R4	1000
R5	
R6	

Reorder Buffer												
Tag	Inum	PC	Use	Ex	Op	p1	src1	p2	src2	pd	dest	data
…	…	…	…	…	…	…	…	…	…	…	…	…
T6	I6	…	…	…	…	…	…	…	…	…	…	…
T7	I7	0x20	1	1	addi	1	3980			1	R2	4000
T8	I8	0x24	1	1	div	1	1000	1	25		R4	
T9	I9	0x28	1	1	sw	1	4000	1	4000			
T10	I10	0x2c	1		beqz		T8					
T11	I11	0xa0	1		lw	1	4000				R1	
T12	I12	0xa4	1		sw		T11	1	4000			
T13	I13	0xa8	1		div	1	4000		T11		R5	
T14	I14	0xac	1		beqz		T13					
T15	I15	0xb0	1	1	lw	1	3980				R1	
T16	I16	0xb4	1	1	sub	1	4000	1	3980	1	R5	20

Next to commit

Next available

Rename Table		
Reg	Valid	Tag
R1	1	T15
R2	1	T7
R3		
R4	1	T8
R5	1	T16
R6		

Figure 4: Processor State

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

- **Fetched Instruction Queue**: Holds the fetched instructions.
- **Decoded Instruction Queue**: Holds the decoded instructions.
- **Next PC to be fetched**: See 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.
- **Prediction Counter**: 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 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.

- **Store Buffer**: The address and data from an executed SW instruction are temporarily kept in a store buffer, 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 in the load buffer, 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 using 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 replace old values with lazily updated values
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