Quiz 2 Handout

Figure 1 shows the pipeline of an out-of-order machine which uses a **split issue queue and commit queue**. 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. Rename & Allocate: The rename table is used to locate any source operands. In parallel, several structures are checked for space that will be needed by the instruction:
    - For any instruction, an entry in the issue queue and in the commit queue (ROB).
    - For an instruction that writes a register, a physical register (managed by free list).
    - For a store instruction, an entry in the store buffer.
    - For a load instruction, an entry in the load buffer.
4. Dispatch: If all the needed space is available, then the space is allocated, the instruction is inserted into the issue queue and commit queue, and the rename table is updated with the new destination register, if any.
5. Issue & Register Read: On each cycle, the oldest ready instruction is **removed from the issue queue**, source operands are read from the register file if needed, so that the instruction may be issued to the appropriate functional unit or load/store unit.
6. Execute: Functional units or the memory system may take one or more cycles to execute the instruction.
7. Register Write: The output of the instruction, if any, is written to the register file, and the issue queue is notified.
8. Commit: On each cycle, if the oldest instruction in the commit queue has finished execution, it is committed.

This design uses a unified register file for committed and speculative data. The ROB stores only references to physical registers, and does not store data.

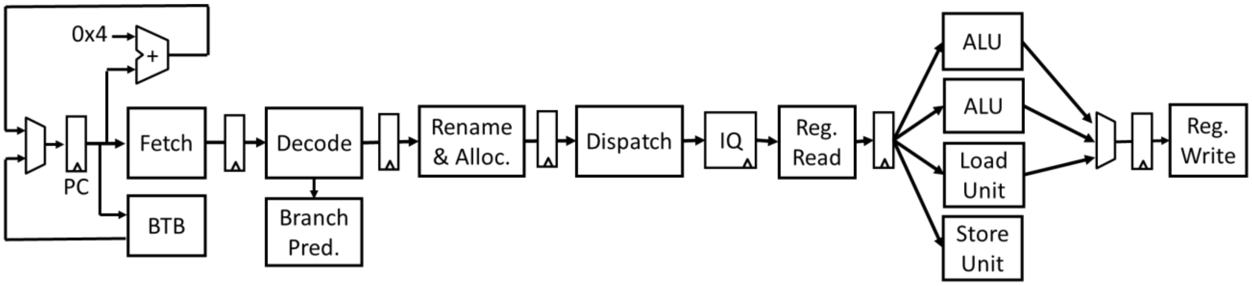


Figure 1: Simplified out-of-order pipeline schematic. Several important structures are not shown, such as commit, bypassing, and some sources of next-PC values

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).

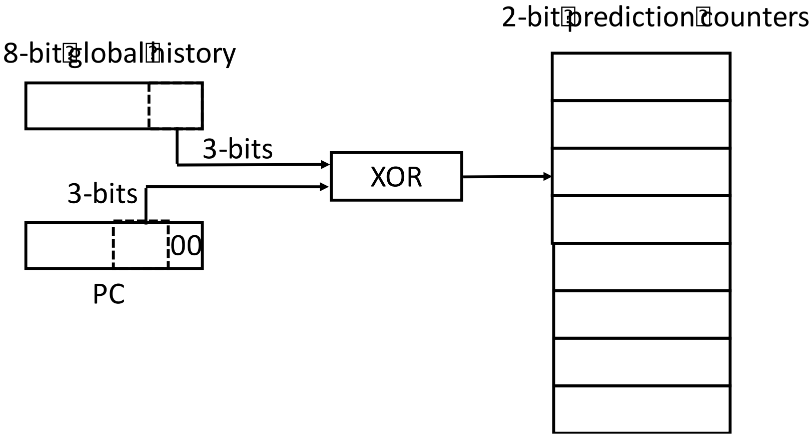


Figure 2: gshare branch predictor

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**.

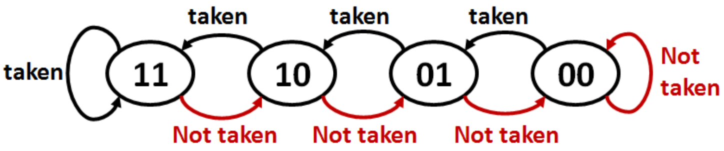
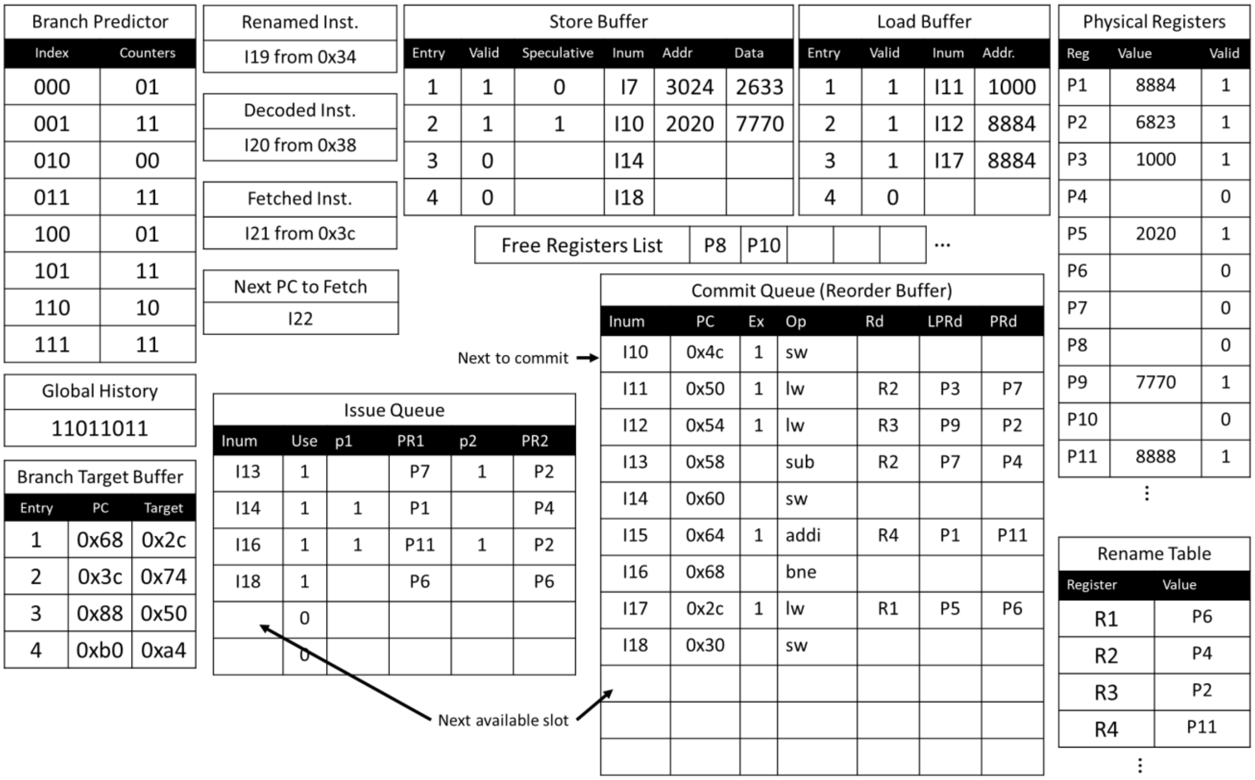


Figure 3: State Diagram of 2-bit counters

Processor State



Branch Predictor

Index	Counters
000	01
001	11
010	00
011	11
100	01
101	11
110	10
111	11

Global History
11011011

Branch Target Buffer

Entry	PC	Target
1	0x68	0x2c
2	0x3c	0x74
3	0x88	0x50
4	0xb0	0xa4

Renamed Inst.
I19 from 0x34

Decoded Inst.
I20 from 0x38

Fetched Inst.
I21 from 0x3c

Next PC to Fetch
I22

Store Buffer

Entry	Valid	Speculative	Inum	Addr	Data
1	1	0	I7	3024	2633
2	1	1	I10	2020	7770
3	0		I14		
4	0		I18		

Load Buffer

Entry	Valid	Inum	Addr.
1	1	I11	1000
2	1	I12	8884
3	1	I17	8884
4	0		

Free Registers List	P8	P10				…

Issue Queue

Inum	Use	p1	PR1	p2	PR2
I13	1		P7	1	P2
I14	1	1	P1		P4
I16	1	1	P11	1	P2
I18	1		P6		P6
	0				
	0				

Commit Queue (Reorder Buffer)

Inum	PC	Ex	Op	Rd	LPRd	PRd
I10	0x4c	1	sw			
I11	0x50	1	lw	R2	P3	P7
I12	0x54	1	lw	R3	P9	P2
I13	0x58		sub	R2	P7	P4
I14	0x60		sw			
I15	0x64	1	addi	R4	P1	P11
I16	0x68		bne			
I17	0x2c	1	lw	R1	P5	P6
I18	0x30		sw			

Physical Registers

Reg	Value	Valid
P1	8884	1
P2	6823	1
P3	1000	1
P4		0
P5	2020	1
P6		0
P7		0
P8		0
P9	7770	1
P10		0
P11	8888	1
⋮		

Rename Table

Register	Value
R1	P6
R2	P4
R3	P2
R4	P11
⋮	

Figure 4: Processor State. There are 27 additional rename table entries and physical registers holding committed architectural register values, which are not shown.

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

- **Renamed Instruction**: Pipeline register holding the next instruction to be dispatched to the issue queue. The instruction's source operands have undergone register renaming.
- **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.
- **Prediction Counters (BHT)**: 2-bit counters for branch prediction.
- **Branch Global History**: 8-bit global branch history.
- **Physical Registers**: The processor holds all data in a **unified physical register file.**
- **Free List**: Tracks which physical registers are available for use.
- **Rename Table:** A map from the 31 writeable architectural registers to physical registers.
- **Issue queue**: Contains instructions waiting to execute (but not any register values).
- **Commit queue (ROB)**: Contains bookkeeping information for managing the out-of-order execution and register renaming (but does not contain any register data values).

- **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 (PR1) provides the base register for the store address, and the second operand (PR2) 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