Quiz 1 Handout

Figure 1 shows the pipeline of an out-of-order machine. Registers denote stage boundaries. Blocks in parallel to each other represent parallel operations occurring within the same stage.

Note that the processor *supports only RISC-V integer arithmetic, floating-point arithmetic, and control-flow instructions*. This means we have no loads or stores, and no instructions that operate on both integer and floating-point registers.

The processor consists of the following stages:

1. 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. Decode: The fetched instruction is decoded.
    - If the decoded instruction is a conditional branch, its direction is predicted by a branch predictor. The branch predictor is described in the next page.
      Note: Jumps (JAL/JALR) are always taken, so no prediction is needed.
    - For branches and direct jumps (BEQ/BNE/BLT/BGE/JAL), the branch target is calculated by a branch target calculator.
    - Decode redirects the next PC according to the prediction and target, if required.
    - The integer/floating-point reservation stations and the commit queue are checked for available entries.
    - The free list is checked for free integer/floating-point registers.
3. Rename & Allocate: The instruction is inserted into the corresponding reservation station and the commit queue only if *all* the checks in the previous cycle (Decode) pass. **This design uses Physical Register Files for integer and floating-point registers. The reservation stations and commit queue store only register ids, and do not store data**.
4. Issue & Register Read: On each cycle, the oldest ready instruction in each reservation station is issued, reading its operands from the physical register file. The instruction's used bit is cleared in the reservation station, allowing a new instruction to be allocated.
5. Execute: ALUs and floating-point units may take one or more cycles to execute the instruction.
6. Register Write: The output from the functional unit is written to the physical register file and the register's present bit is set. Dependent instructions in the reservation stations have the corresponding present bit set (`p1` for the first operand, `p2` for the second operand).
7. Commit: The oldest instruction in the commit queue that has finished execution commits. The processor can only commit one instruction per cycle.

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

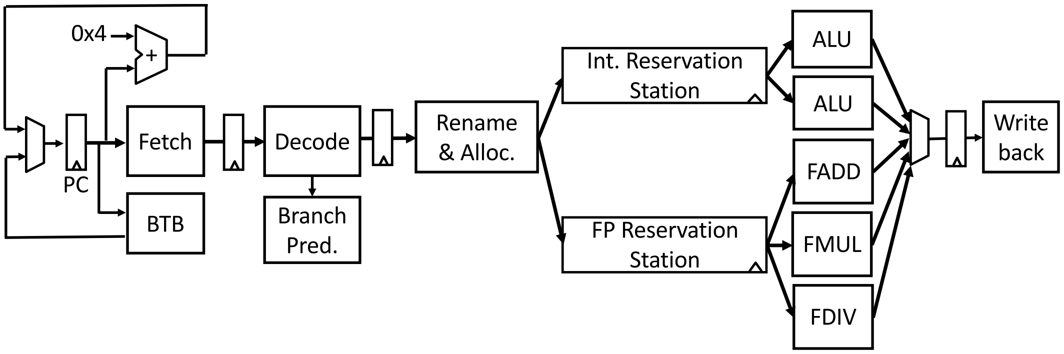


Figure 1: Out-of-order pipeline.

***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 two bits from the global history and the PC (excluding the last 2 for the PC), and calculates an index into an array of the two-bit counters by exclusive OR-ing them, as shown in Figure 2.

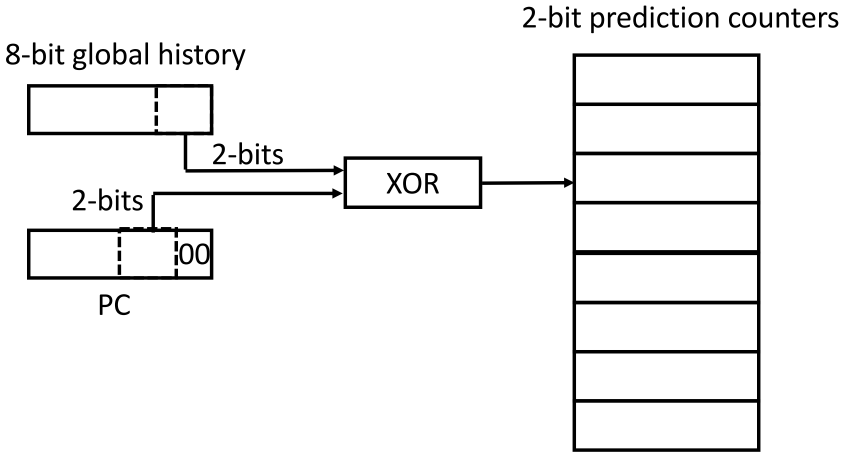


Figure 2: gshare branch predictor

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**.

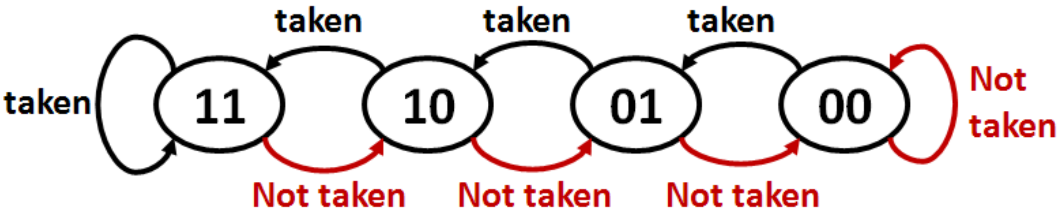
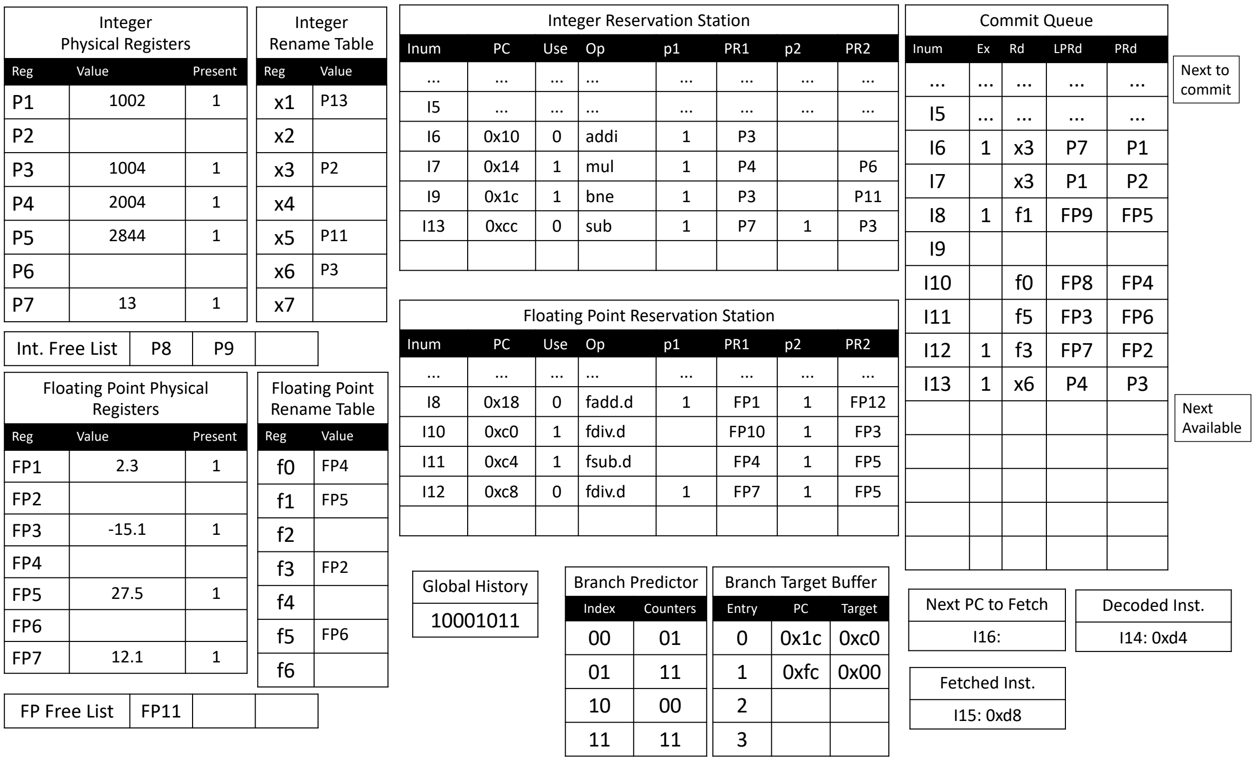


Figure 3: State Diagram of 2-bit counters

Processor State



Integer Physical Registers		
Reg	Value	Present
P1	1002	1
P2		
P3	1004	1
P4	2004	1
P5	2844	1
P6		
P7	13	1

Integer Rename Table	
Reg	Value
x1	P13
x2	
x3	P2
x4	
x5	P11
x6	P3
x7	

Int. Free List	P8	P9	

Integer Reservation Station							
Inum	PC	Use	Op	p1	PR1	p2	PR2
...	...	...	...	...	...	...	...
I5	...	...	...	...	...	...	...
I6	0x10	0	addi	1	P3		
I7	0x14	1	mul	1	P4		P6
I9	0x1c	1	bne	1	P3		P11
I13	0xcc	0	sub	1	P7	1	P3

Commit Queue					
Inum	Ex	Rd	LPRd	PRd	
...	...	...	...	...	Next to commit
I5	...	...	...	...	
I6	1	x3	P7	P1	
I7		x3	P1	P2	
I8	1	f1	FP9	FP5	
I9					
I10		f0	FP8	FP4	
I11		f5	FP3	FP6	
I12	1	f3	FP7	FP2	
I13	1	x6	P4	P3	
					Next Available

Floating Point Physical Registers		
Reg	Value	Present
FP1	2.3	1
FP2		
FP3	-15.1	1
FP4		
FP5	27.5	1
FP6		
FP7	12.1	1

Floating Point Rename Table	
Reg	Value
f0	FP4
f1	FP5
f2	
f3	FP2
f4	
f5	FP6
f6	

FP Free List	FP11		

Floating Point Reservation Station							
Inum	PC	Use	Op	p1	PR1	p2	PR2
...	...	...	...	...	...	...	...
I8	0x18	0	fadd.d	1	FP1	1	FP12
I10	0xc0	1	fdiv.d		FP10	1	FP3
I11	0xc4	1	fsub.d		FP4	1	FP5
I12	0xc8	0	fdiv.d	1	FP7	1	FP5

Global History
10001011

Branch Predictor	
Index	Counters
00	01
01	11
10	00
11	11

Branch Target Buffer		
Entry	PC	Target
0	0x1c	0xc0
1	0xfc	0x00
2		
3		

Next PC to Fetch
I16:

Decoded Inst.
I14: 0xd4

Fetched Inst.
I15: 0xd8

Figure 4: Processor State

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

- **Next PC to fetch**: This is the PC register in Figure 1.
- **Fetched Instruction**: Pipeline Register holding a raw binary instruction.
- **Decoded Instruction**: Pipeline Register holding a decoded instruction.
- **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.
- **Integer/Floating-Point Physical Registers**: The processor holds all integer/floating-point data values in respective physical register files**.**
- **Integer/Floating-Point Free List**: Tracks which integer/floating-point physical registers are available for use.
- **Integer/Floating-Point Rename Table:** A map from integer/floating-point architectural to physical register names.
- **Integer/Floating-Point Reservation Station:** Contains the bookkeeping information for managing out-of-order issue, such as the instruction type and the source operand physical registers. The used bit is cleared when the instruction is issued, freeing the entry for a new instruction.

- **Commit queue:** Contains the bookkeeping information for managing in-order commit of all instructions, such as the destination physical register and whether the instruction has finished execution.

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