Lecture 21: BPred, OOO, Memory Hierarchy

- Today's topics:
  - Branch Predictors
  - Out-of-order execution
  - Cache intro

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

XOR – out is 1 if A or B is 1 but not if A=B=1

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

(sum)

Control Hazards

① Do nothing CPI = 1.2

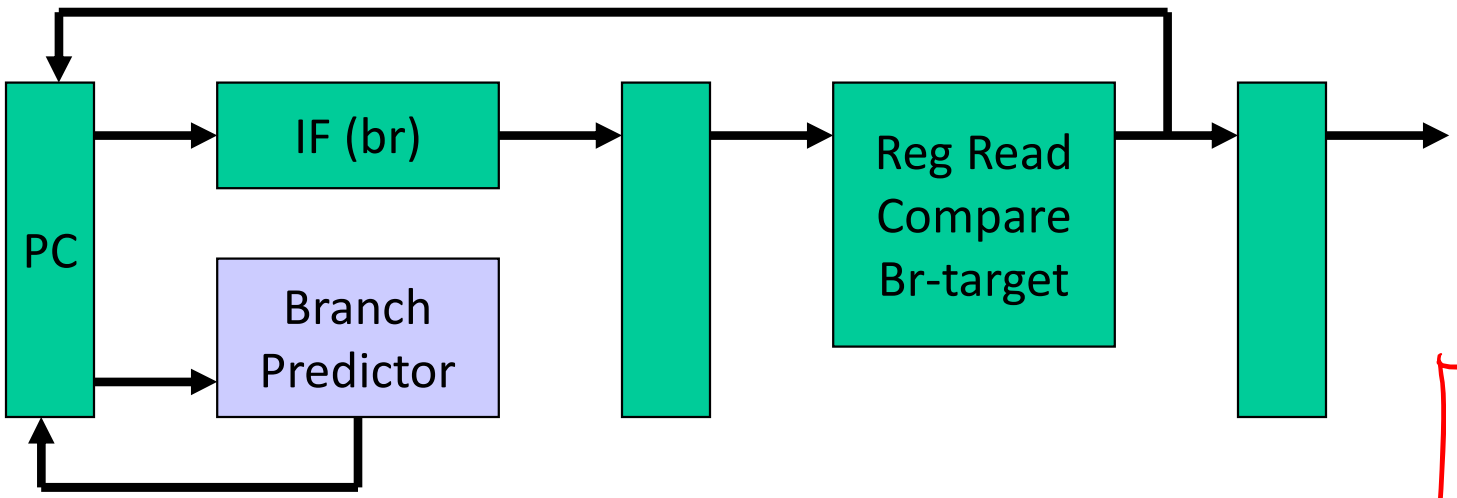
② Predict PC+4 CPI = 1.06

③ Br-delay slots
- Keep proc simple
- Onus is on Compiler

CPI = 1.0 to 1.2

⇒ ④ Hw branch predictors 96%. CPI = 1.008

Pipeline with Branch Predictor

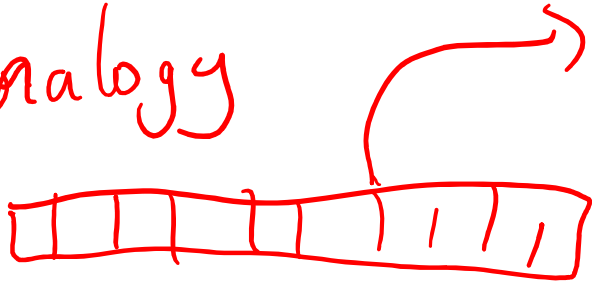


code → BEQ

input vector signature describes what I've seen.

pre-trained model

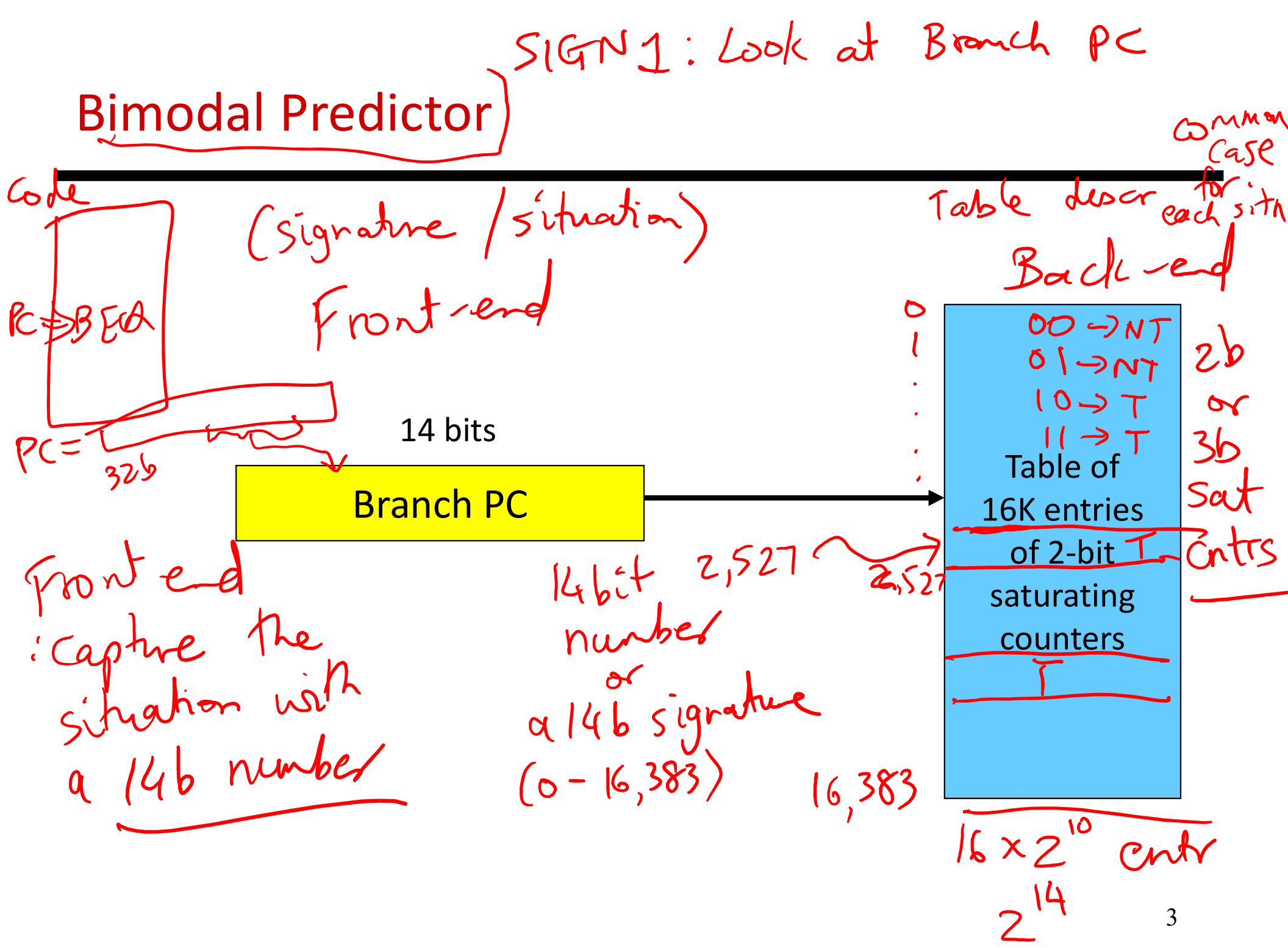
LCM analogy



feature vector (input) describes what you've seen so far

Bimodal Predictor

SIGN 1: Look at Branch PC



(signature / situation)

Front-end

Front end: capture the situation with a 14b number

14bit number or a 14b signature (0 - 16,383)

Back-end

Table descr for each sitn

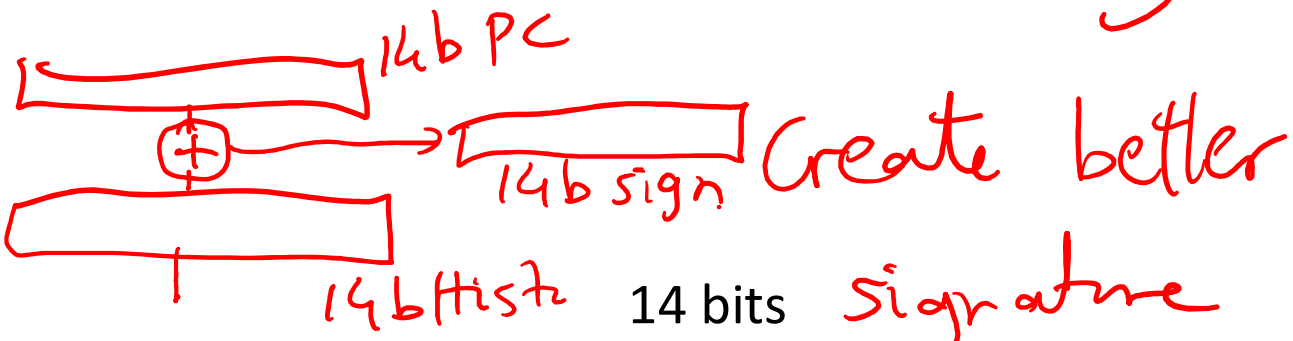
common case

Local, Global, Tournament

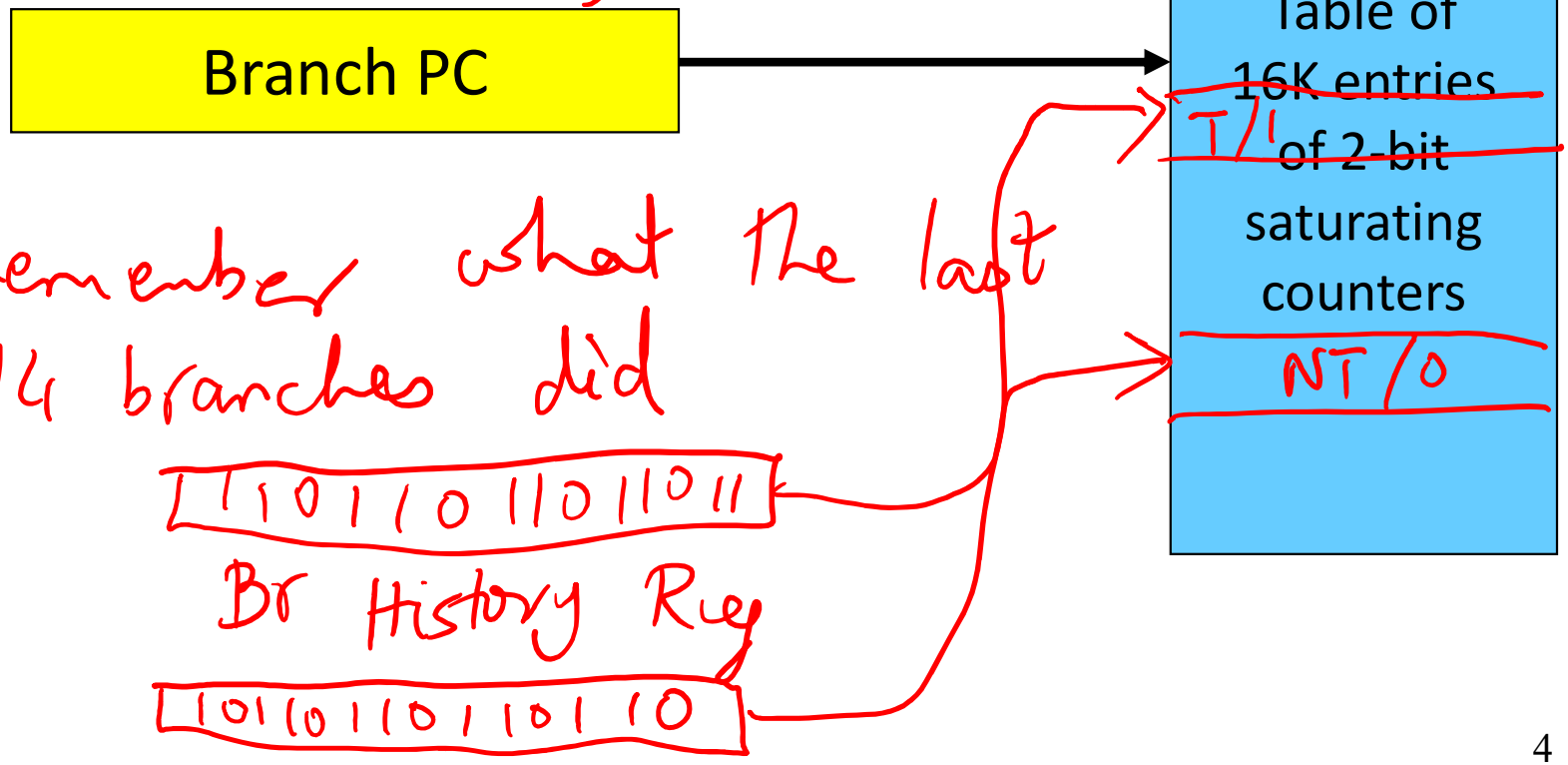
~~Bimodal Predictor~~

14b hist } 14b sign
14b PC

OPT2: combo of PC & Histy Concat, ~~AND~~ XOR



Create better



OPT0: PC

OPT1: Remember what the last 14 branches did

2-Bit Prediction

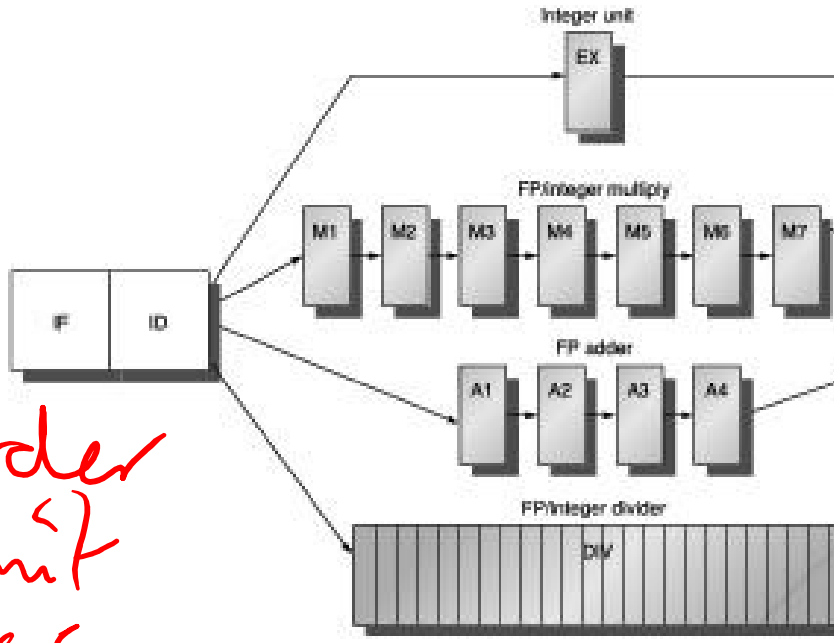
- For each branch, maintain a 2-bit saturating counter:
  if the branch is taken: counter = min(3,counter+1)
  if the branch is not taken: counter = max(0,counter-1)
  … sound familiar?

- If (counter >= 2), predict taken, else predict not taken

- The counter attempts to capture the common case for each branch

Indexing functions
Multiple branch predictors
History, trade-offs

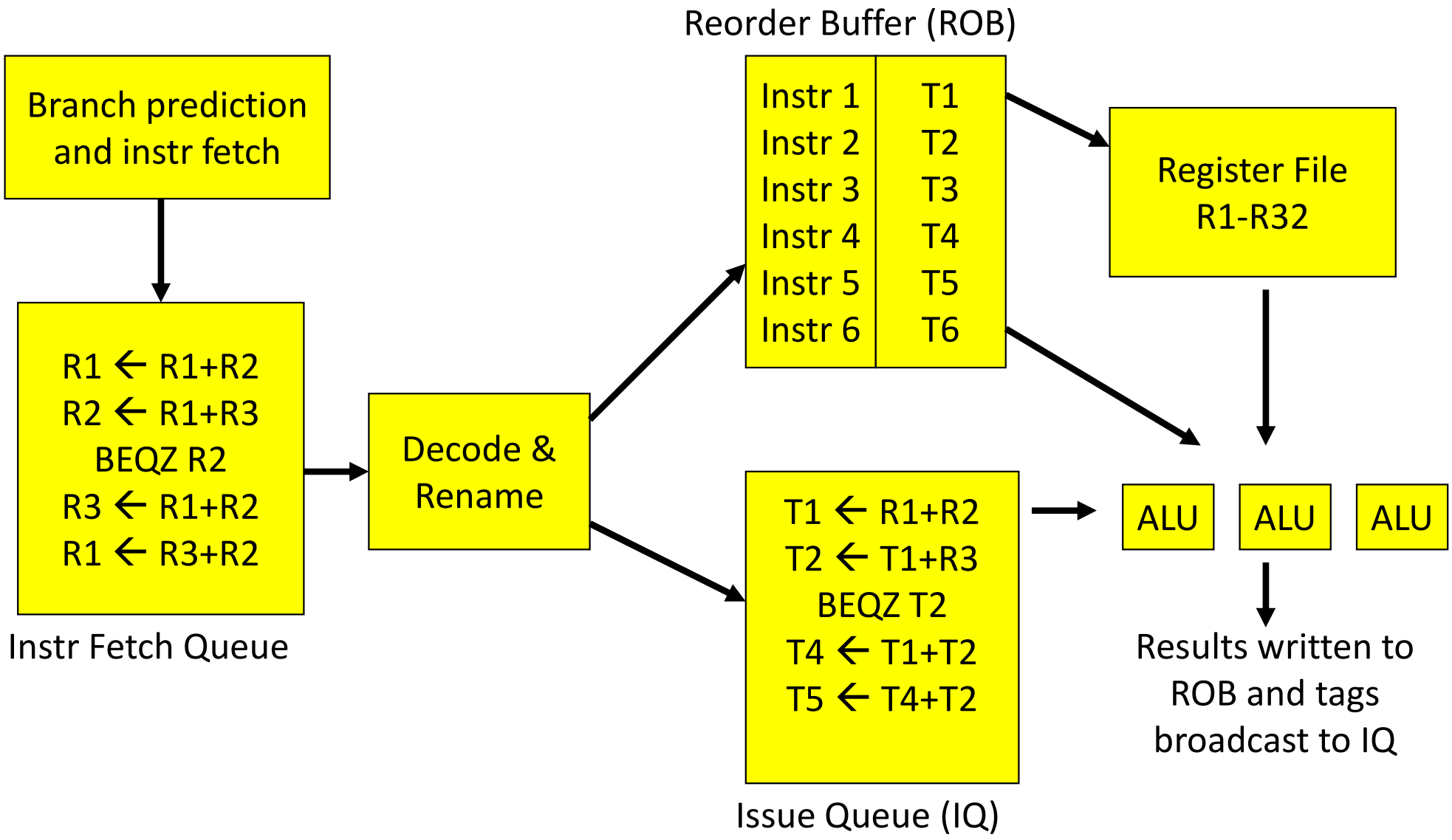
Multicycle Instructions



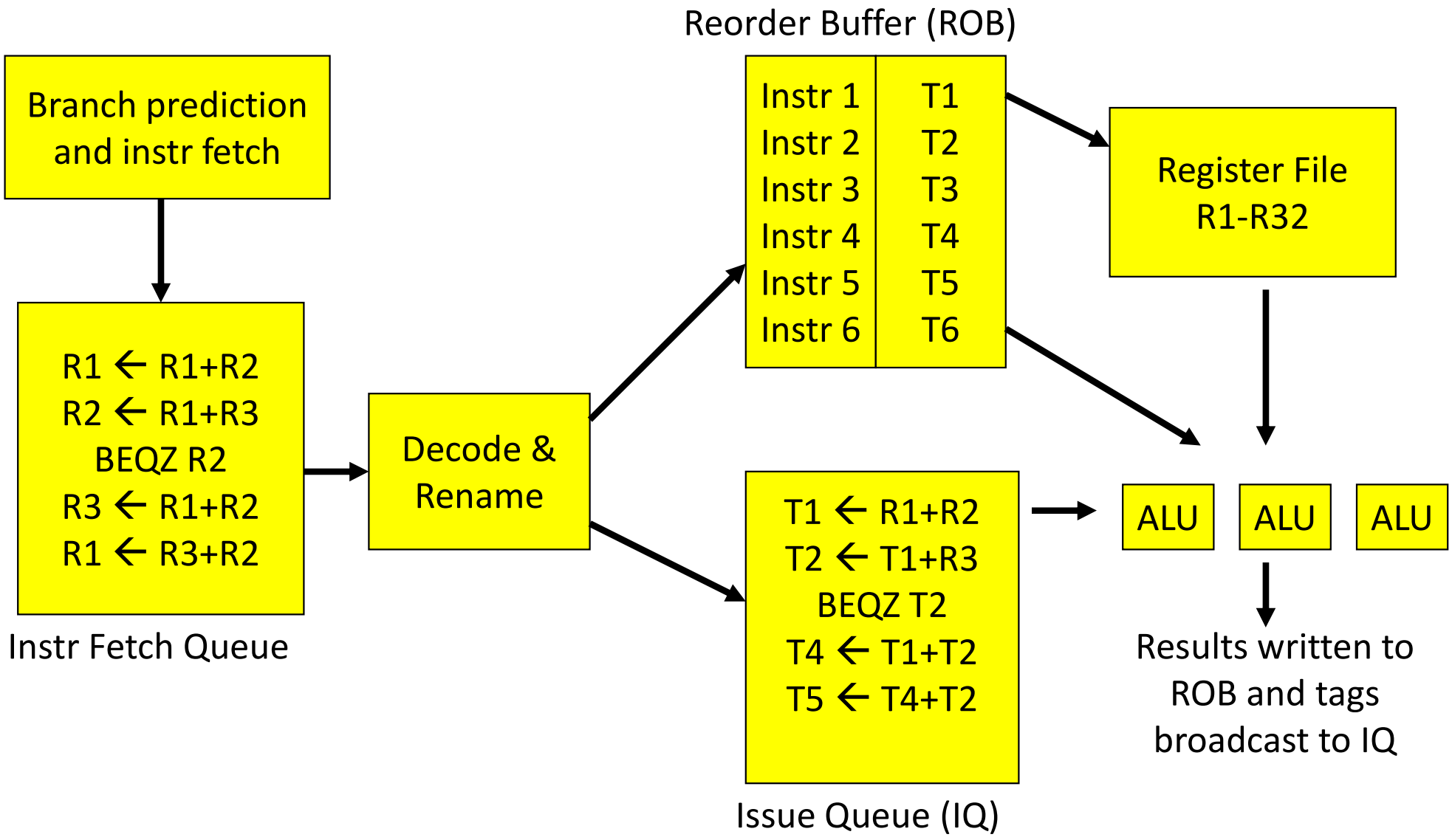
- Multiple parallel pipelines – each pipeline can have a different number of stages

- Instructions can now complete out of order – must make sure that writes to a register happen in the correct order

An Out-of-Order Processor Implementation



An Out-of-Order Processor Implementation



Example Code

Completion times	with in-order	with ooo
ADD R1, R2, R3	5	5
ADD R4, R1, R2	6	6
LW R5, 8(R4)	7	7
ADD R7, R6, R5	9	9
ADD R8, R7, R5	10	10
LW R9, 16(R4)	11	7
ADD R10, R6, R9	13	9
ADD R11, R10, R9	14	10

Cache Hierarchies

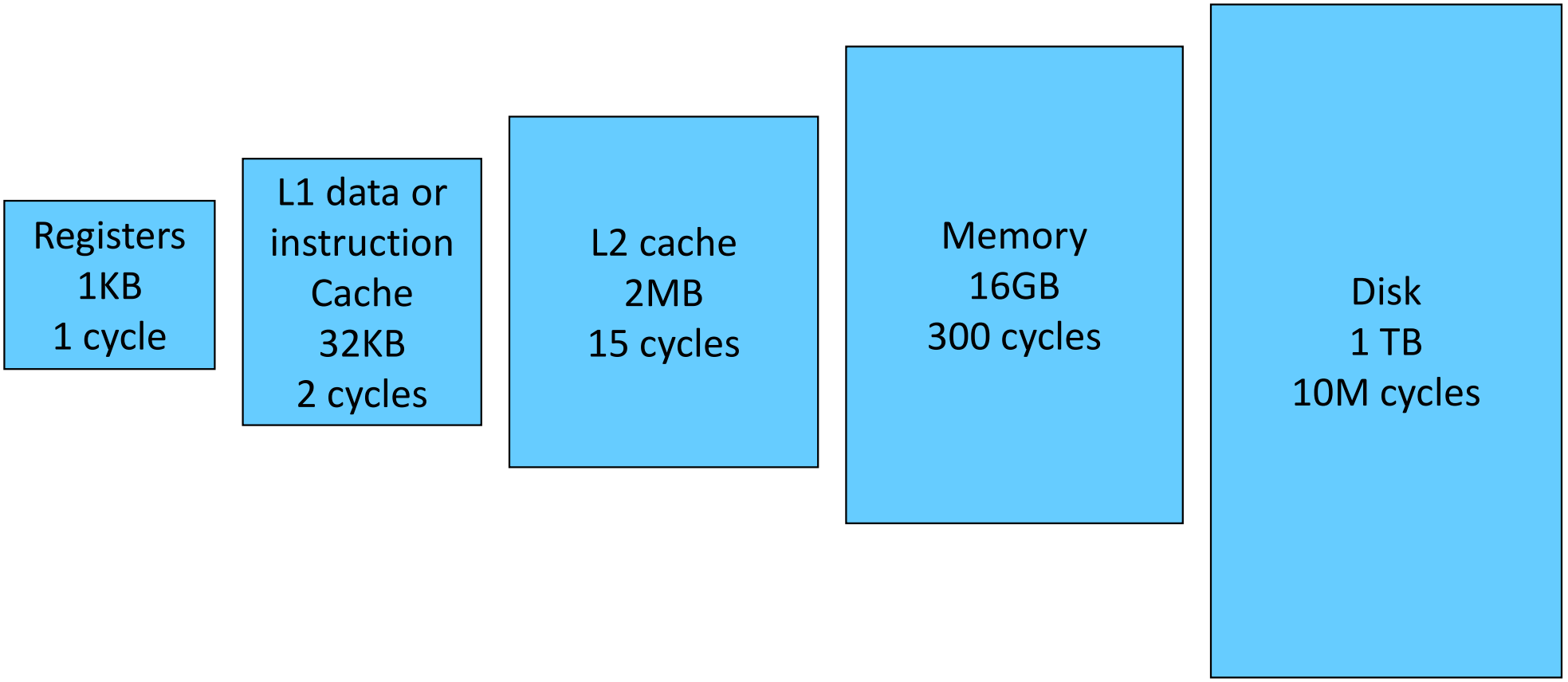
- Data and instructions are stored on DRAM chips – DRAM is a technology that has high bit density, but relatively poor latency – an access to data in memory can take as many as 300 cycles today!

- Hence, some data is stored on the processor in a structure called the cache – caches employ SRAM technology, which is faster, but has lower bit density

- Internet browsers also cache web pages – same concept

Memory Hierarchy

- As you go further, capacity and latency increase



Locality

- Why do caches work?
  - Temporal locality: if you used some data recently, you will likely use it again
  - Spatial locality: if you used some data recently, you will likely access its neighbors

- No hierarchy: average access time for data = 300 cycles

- 32KB 1-cycle L1 cache that has a hit rate of 95%:

$$\text{average access time} = 0.95 \times 1 + 0.05 \times (301)$$
$$= 16 \text{ cycles}$$