# Lecture 20: OOO, Memory Hierarchy

- Today's topics:
  - Branch prediction wrap-up
  - Out-of-order execution
  - Cache basics

#### **Bimodal Predictor**



- 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

- Perfect pipelining with no hazards → an instruction completes every cycle (total cycles ~ num instructions)
  → speedup = increase in clock speed = num pipeline stages
- With hazards and stalls, some cycles (= stall time) go by during which no instruction completes, and then the stalled instruction completes
- Total cycles = number of instructions + stall cycles

# **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



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

- 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

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%: average access time = 0.95 x 1 + 0.05 x (301) = 16 cycles

## Accessing the Cache



# The Tag Array



#### **Example Access Pattern**



## **Increasing Line Size**



# Associativity



# Associativity





- 32 KB 4-way set-associative data cache array with 32 byte line sizes
- How many sets?
- How many index bits, offset bits, tag bits?
- How large is the tag array?



#### Bullet