

Chapter 5

Large and Fast: Exploiting Memory Hierarchy

Fall 2018

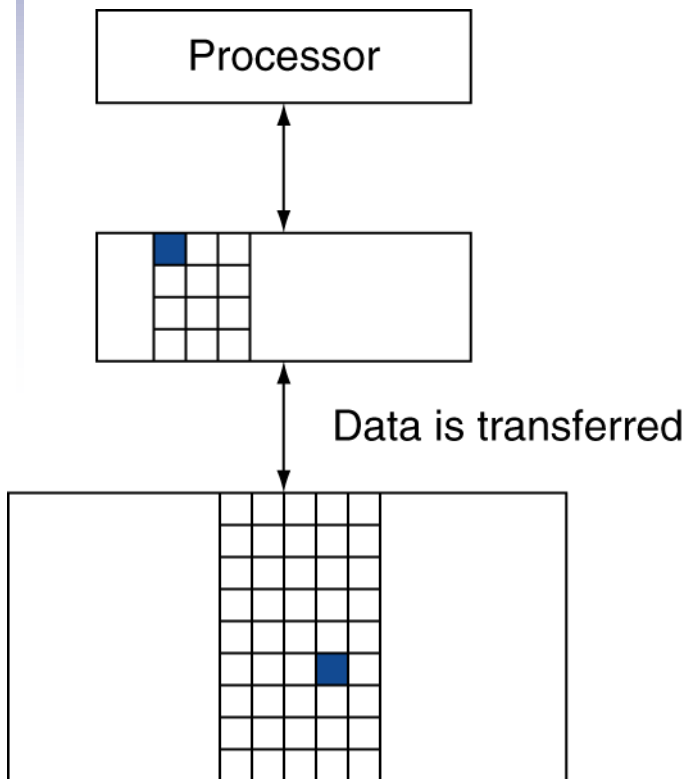
Soontae Kim
School of Computing, KAIST

Homework #2

- Pipelining questions
 - Due on. Nov. 20 (Tues.)

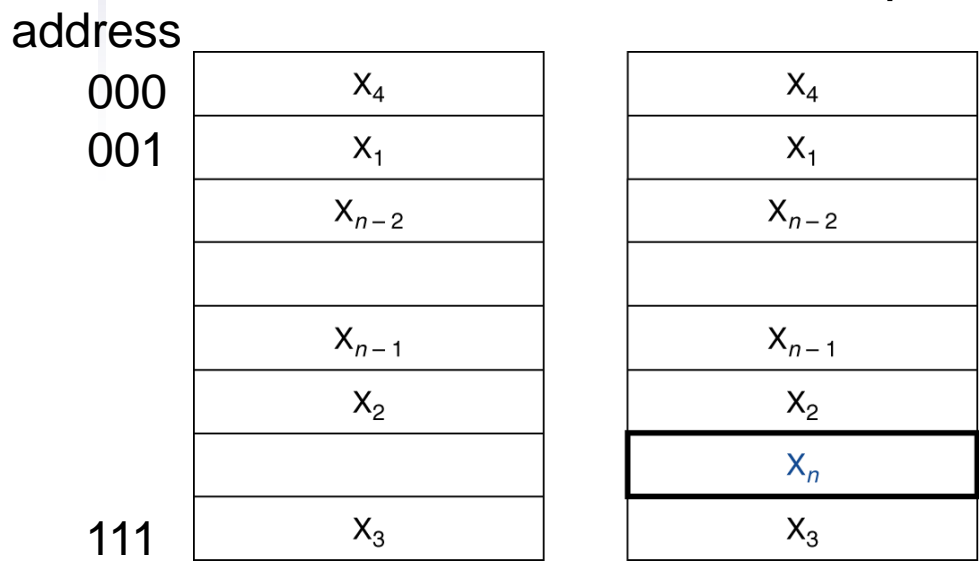
Memory Hierarchy Levels

- Block (aka line): unit of copying
 - May be multiple words
- If accessed data is present in upper level
 - Hit: access satisfied by upper level
 - Hit ratio: hits/accesses
- If accessed data is absent
 - Miss: block copied from lower level
 - Time taken: miss penalty
 - Miss ratio: misses/accesses
= $1 - \text{hit ratio}$
 - Then accessed data supplied from upper level



Cache Memory

- Cache memory
 - The level of the memory hierarchy closest to the CPU, very small compared to main memory
- Given accesses X_1, \dots, X_{n-1}, X_n



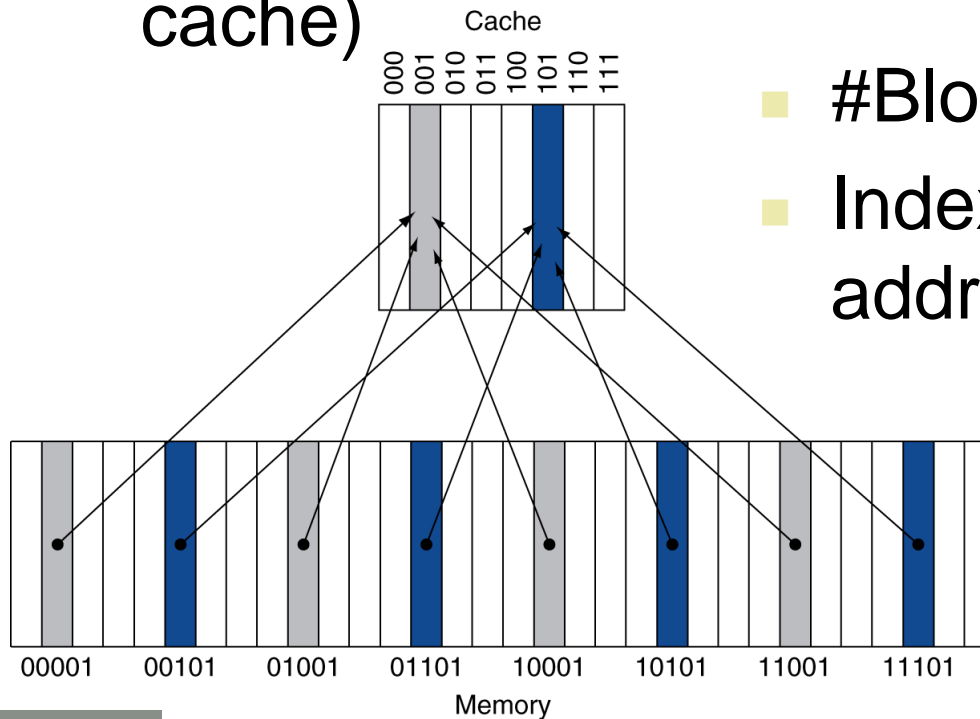
a. Before the reference to X_n

b. After the reference to X_n

- How do we know if the data is present?
- Where do we look?

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
 - Index = (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Index use low-order address bits

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
 - Store block address as well as the data
 - Actually, only need the high-order bits called the tag (2 bits in previous slide)
- What if there is no data in a location?
 - Valid bit: 1 = present, 0 = not present
 - Initially 0

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

| Index | V | Tag | Data |
|-------|---|-----|------|
| 000 | N | | |
| 001 | N | | |
| 010 | N | | |
| 011 | N | | |
| 100 | N | | |
| 101 | N | | |
| 110 | N | | |
| 111 | N | | |

Cache Example

| Word addr | Binary addr | Hit/miss | Cache block |
|-----------|-------------|----------|-------------|
| 22 | 10 110 | Miss | 110 |

| Index | V | Tag | Data |
|-------|---|-----|------------|
| 000 | N | | |
| 001 | N | | |
| 010 | N | | |
| 011 | N | | |
| 100 | N | | |
| 101 | N | | |
| 110 | Y | 10 | Mem[10110] |
| 111 | N | | |

Cache Example

| Word addr | Binary addr | Hit/miss | Cache block |
|-----------|-------------|----------|-------------|
| 26 | 11 010 | Miss | 010 |

| Index | V | Tag | Data |
|-------|---|-----|------------|
| 000 | N | | |
| 001 | N | | |
| 010 | Y | 11 | Mem[11010] |
| 011 | N | | |
| 100 | N | | |
| 101 | N | | |
| 110 | Y | 10 | Mem[10110] |
| 111 | N | | |

Cache Example

| Word addr | Binary addr | Hit/miss | Cache block |
|-----------|-------------|----------|-------------|
| 22 | 10 110 | Hit | 110 |
| 26 | 11 010 | Hit | 010 |

| Index | V | Tag | Data |
|-------|---|-----|------------|
| 000 | N | | |
| 001 | N | | |
| 010 | Y | 11 | Mem[11010] |
| 011 | N | | |
| 100 | N | | |
| 101 | N | | |
| 110 | Y | 10 | Mem[10110] |
| 111 | N | | |

Cache Example

| Word addr | Binary addr | Hit/miss | Cache block |
|-----------|-------------|----------|-------------|
| 16 | 10 000 | Miss | 000 |
| 3 | 00 011 | Miss | 011 |
| 16 | 10 000 | Hit | 000 |

| Index | V | Tag | Data |
|-------|---|-----|------------|
| 000 | Y | 10 | Mem[10000] |
| 001 | N | | |
| 010 | Y | 11 | Mem[11010] |
| 011 | Y | 00 | Mem[00011] |
| 100 | N | | |
| 101 | N | | |
| 110 | Y | 10 | Mem[10110] |
| 111 | N | | |

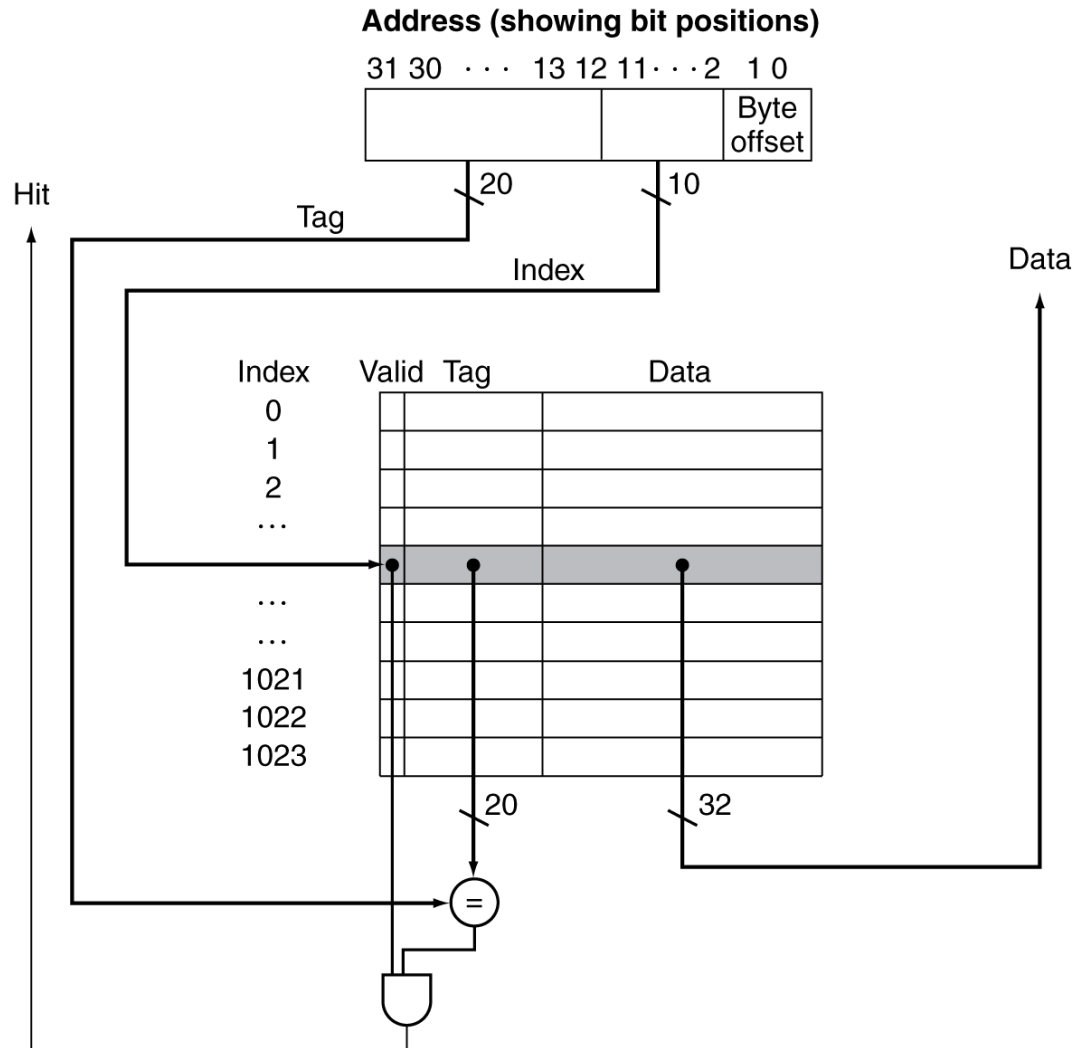
Cache Example

| Word addr | Binary addr | Hit/miss | Cache block |
|-----------|-------------|----------|-------------|
| 18 | 10 010 | Miss | 010 |

11010 replaced by **10010** ← same index

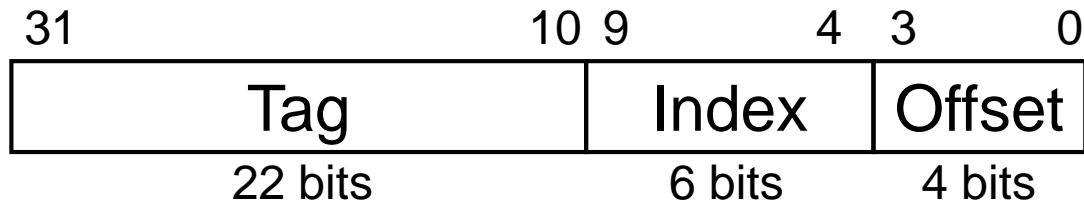
| Index | V | Tag | Data |
|------------|----------|-----------|-------------------|
| 000 | Y | 10 | Mem[10000] |
| 001 | N | | |
| 010 | Y | 10 | Mem[10010] |
| 011 | Y | 00 | Mem[00011] |
| 100 | N | | |
| 101 | N | | |
| 110 | Y | 10 | Mem[10110] |
| 111 | N | | |

Address Subdivision



Example: Larger Block Size

- 64 blocks, 16 bytes/block
 - To what block number does address 1200 map?
- Block address = $\lfloor 1200/16 \rfloor = 75$
- Cache block number = $75 \text{ modulo } 64 = 11$



Block Size Considerations

- Larger blocks should reduce miss rate
 - Due to spatial locality
- But in a fixed-sized cache
 - Larger blocks \Rightarrow fewer of them
 - More competition \Rightarrow increased miss rate
 - Larger blocks \Rightarrow pollution (unnecessary data)
- Larger miss penalty
 - Can override benefit of reduced miss rate
 - Early restart and critical-word-first can help

Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
 - Stall the CPU pipeline
 - Fetch block from next level of hierarchy
 - Instruction cache miss
 - Restart instruction fetch
 - Data cache miss
 - Complete data access

Write-Through

- On data-write hit, could just update the block in cache
 - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
 - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
 - Effective CPI = $1 + 0.1 \times 100 = 11$
- Solution: write buffer
 - Holds data waiting to be written to memory
 - CPU continues immediately
 - Only stalls on write if write buffer is already full

Write-Back

- Alternative: On data-write hit, just update the block in cache
 - Keep track of whether each block is dirty
- When a dirty block is replaced
 - Write it back to memory
 - Can use a write buffer to allow replacing block to be read first

Write Allocation

- What should happen on a write miss?
- Alternatives for write-through
 - Allocate on miss: fetch the block
 - Write around: don't fetch the block
 - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
 - Usually fetch the block