CMPE 110: Computer Architecture

Week 7
Cache

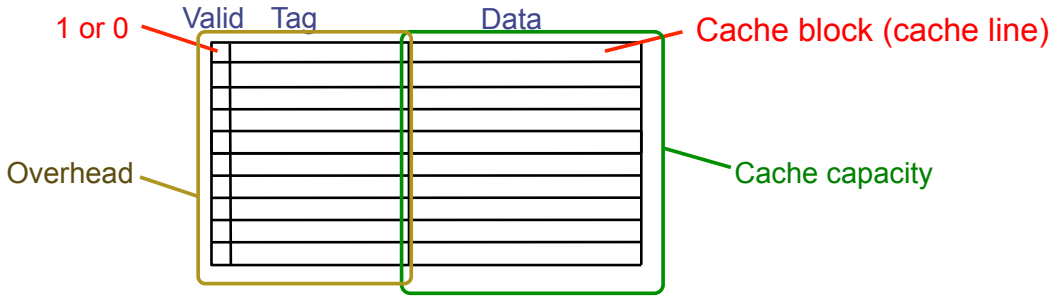
Jishen Zhao (http://users.soe.ucsc.edu/~jzhao/)

[Adapted in part from Jose Renau, Mary Jane Irwin, Joe Devietti, Onur Mutlu, and others]

Reminder

- Homework 2 is due today midnight

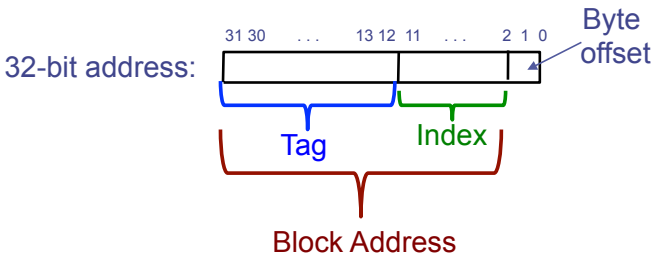
Review: Cache basis



- When data referenced
  - HIT: If in cache, use cached data instead of accessing memory
  - MISS: If not in cache, bring block into cache (invalid → miss)
    - Go to the next level of cache to bring this data up
    - Have to kick something else out to do it, if it is full

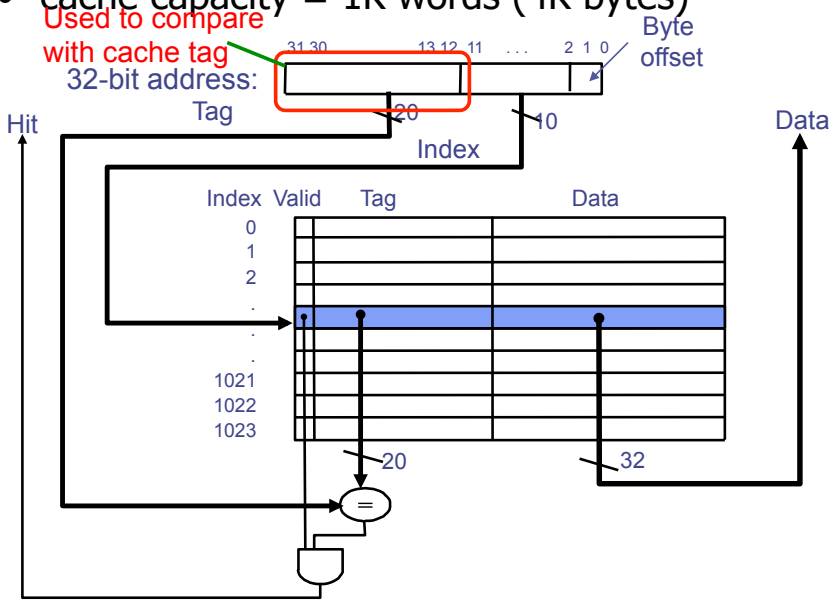
Review: three regions in memory address

- Address is a unique pointer to a data block in main memory



Review: put it all together

- Cache block = 1 word (4 bytes),
- cache capacity = 1K words (4K bytes)

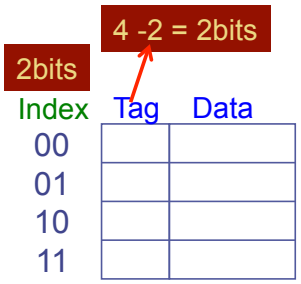


Today

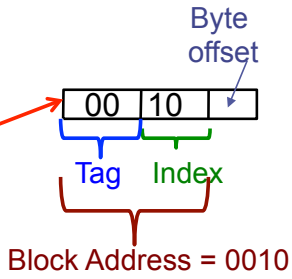
- Example on cache access
- Cache tag overhead calculation
- Cache and pipelining, handling cache hit and miss
- Cache performance, and AMAT
- Types of caches

Example: how is cache accessed

- Start with an empty cache that has a capacity of 4 cache blocks
- A program performs the following *lw* (to make it simple, the load addresses are 4-bit block addresses):




```
lw r2, block address 0
lw r3, block address 1
lw r4, block address 2
lw r5, block address 3
lw r6, block address 4
lw r7, block address 3
lw r8, block address 4
lw r9, block address 15
```



- How does the cache look like after each `lw`?

Example: how is cache accessed

Lw r2, addr(0)

Block Addr: **0=** 0000 miss (tag, Index)

Index	tag	data
00	00	Addr(0)
01		
10		
11		

Lw r3, addr(1)

1=0001 miss

00	Addr(0)
00	Addr(1)

Lw r4, addr(2)

2= 0010 miss

00	Addr (0)
00	Addr (1)
00	Addr (2)

Lw r5, addr(3)

3= 0011 miss

00	Addr (0)
00	Addr (1)
00	Addr (2)
00	Addr (3)

Lw r6, addr(4)

4= 0100 miss

Index	tag	data
00	~~00~~ 01	Addr (0) ~~4~~
01	00	Addr (1)
10	00	Addr (2)
11	00	Addr (3)

Lw r7, addr(3)

3= 0011 hit

01	Addr (4)
00	Addr (1)
00	Addr (2)
00	Addr (3)

Lw r8, addr(4)

4= 0100 hit

01	Addr (4)
00	Addr (1)
00	Addr (2)
00	Addr (3)

Lw r9, addr(15)

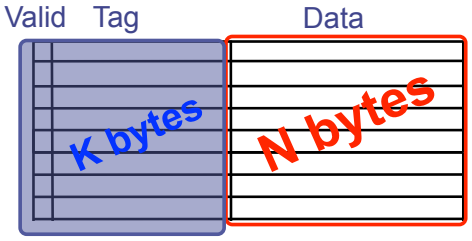
15= 1111 miss

01	Addr (4)
00	Addr (1)
00	Addr (2)
~~00~~ 11	Addr (3) ~~15~~

Calculate tag and valid bit overhead

How to calculate tag overhead

- Tag and valid bit overhead = (K / N) * 100%



Example: Calculating Tag Overhead

- A "4KB cache" -- the cache holds 4KB of data in total
  - Called **capacity**
  - Tag and valid bit are considered as storage overhead
  - Assume 32-bit memory address

- **Question:** Calculate Tag and valid bit overhead of 4KB cache with 1024 of 4B **cache blocks**
- **Solution:**
  - 4B cache block → 2-bit byte offset
  - 1024 cache blocks → 10-bit index
  - 32-bit address – 2-bit offset – 10-bit index = 20-bit tag
  - Overhead = (20-bit tag + 1-bit valid) * 1024 cache blocks = 21Kb = 2.6KB → how much percent?

Exercise: Calculating overhead

- "32KB cache", i.e., cache holds 32KB of data
  - Called **capacity**
  - Tag and valid bit storage is considered as overhead

- Tag overhead of 32KB cache with 1024 32B cache blocks
  - 32B block → 5-bit offset
  - 1024 blocks → 10-bit index
  - 32-bit address – 5-bit offset – 10-bit index = 17-bit tag
  - Overhead = (17-bit tag + 1-bit valid) * 1024 blocks = 18Kb = 2.2KB
  - ~6% overhead
- What about 64-bit addresses?
  - Tag increases to 49 bits, ~20% overhead (worst case)

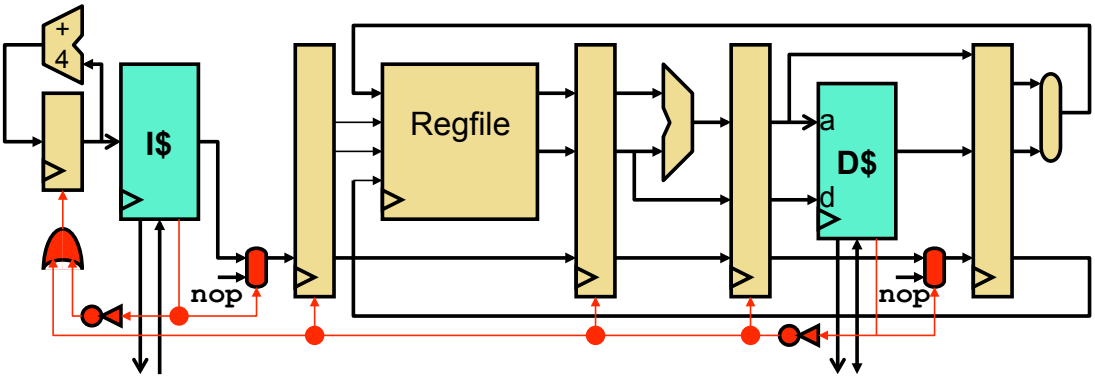
What we learned

- How to access cache with a memory address?
- How to calculate tag overhead?

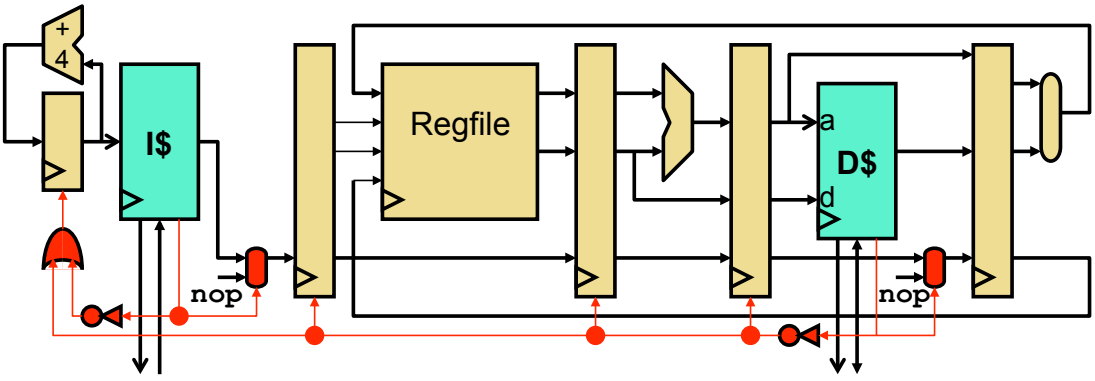
Cache and the pipeline

Review: When is cache accessed?

- Cache is accessed during F and M stage
  - F stage: read instructions
  - M stage: read and write data → `lw` or `sw` instructions only



Cache Misses and Pipeline Stalls



- I$ and D$ misses stall pipeline just like data hazards
  - Stall logic driven by miss signal
    - Cache "logically" re-evaluates hit/miss every cycle
    - Block is filled → miss signal de-asserts → pipeline restarts

Handling cache hit and miss

- L1 cache hit (read / write): no pipeline stall
- Cache miss (read / write): pipeline stall in M stage