CSED311 Lab5: Cache

Okkyun Woo

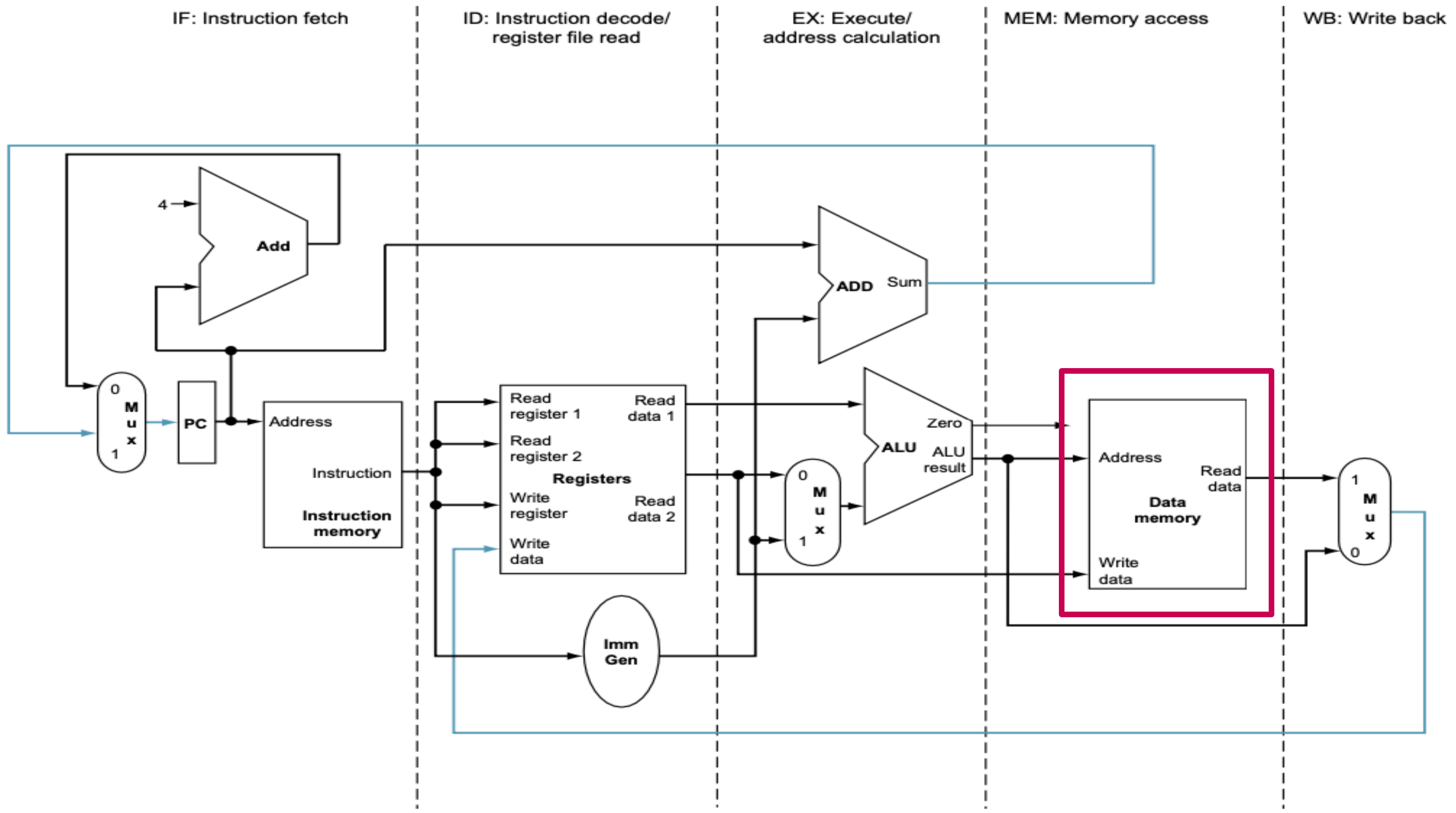
okkyun.w@postech.ac.kr

Contact the TAs at csed311-ta@postech.ac.kr

POSTECH

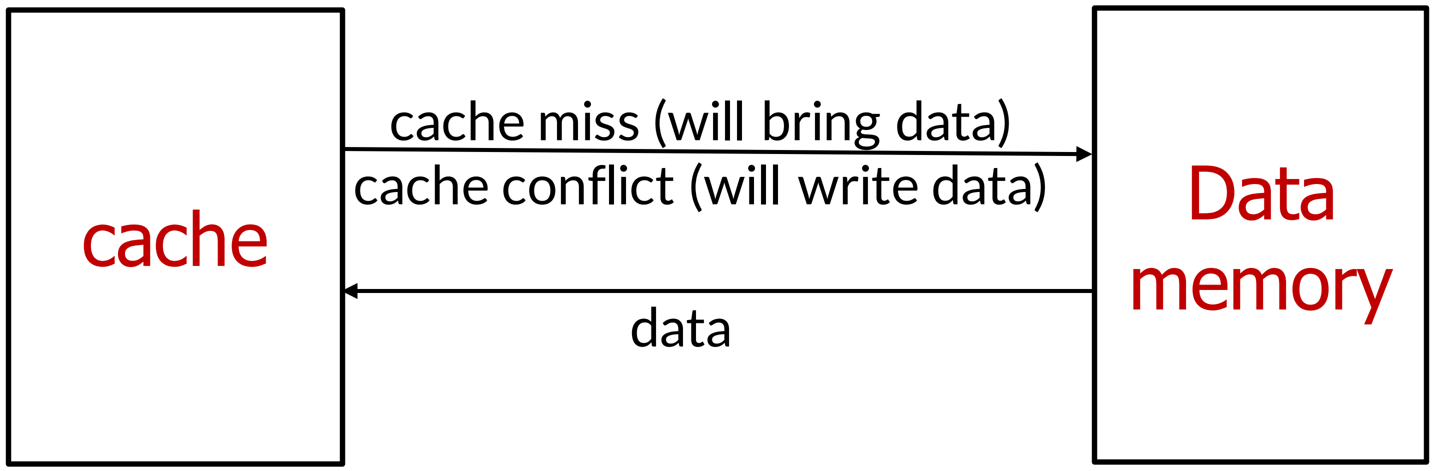
Data cache in Pipelined CPU

- Uses a blocking data cache instead of a “magic memory”

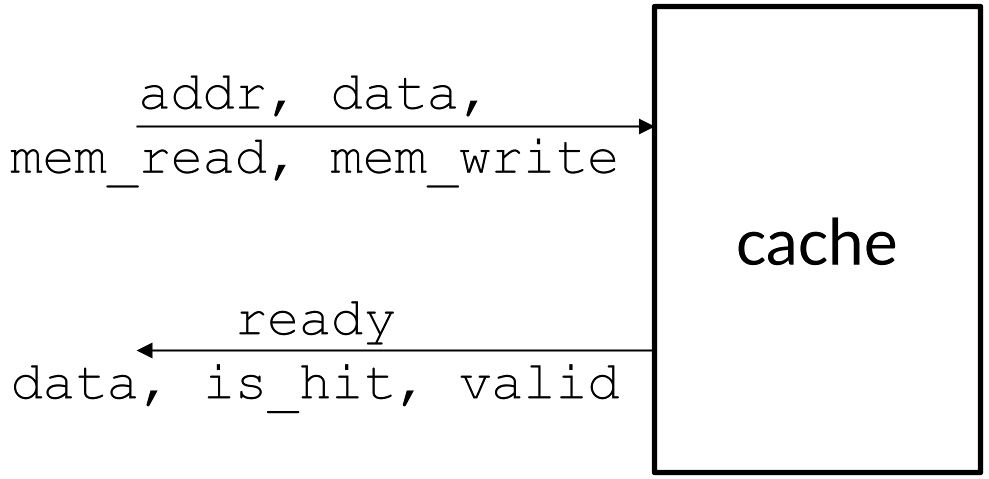


Data flow

- Cache fetches data from the memory

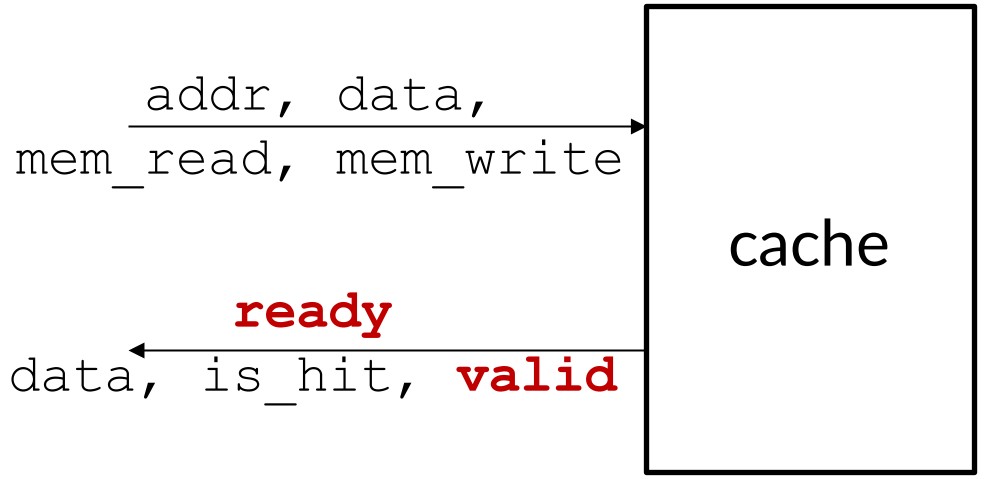

cache
cache miss (will bring data)
cache conflict (will write data)
data
Data memory

Signals to / from the cache



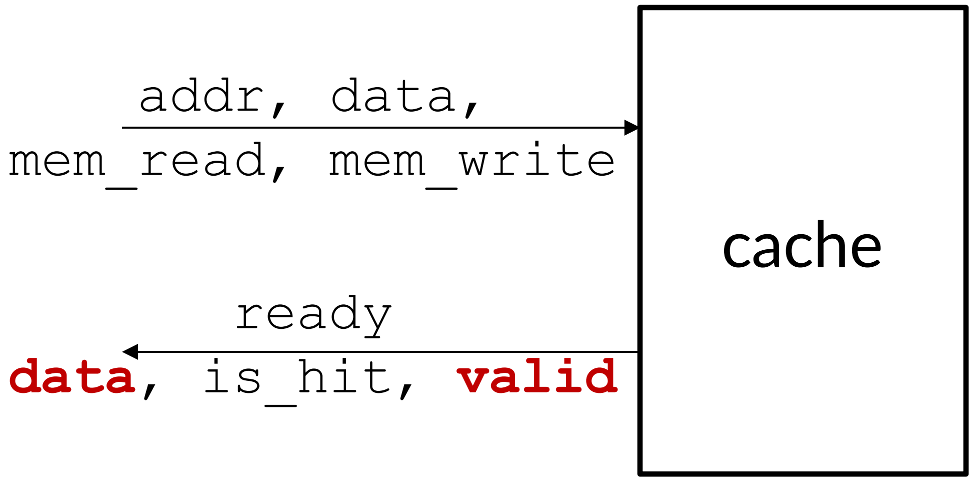
Signals from the data cache

- `ready` represents the status of the cache
  - If `ready` is true => cache is ready to accept request
  - If `ready` is false => cache is busy bringing data from the memory
    - `valid` is false
    - Cache cannot accept a request currently (try later)
    - Need to stall the pipeline



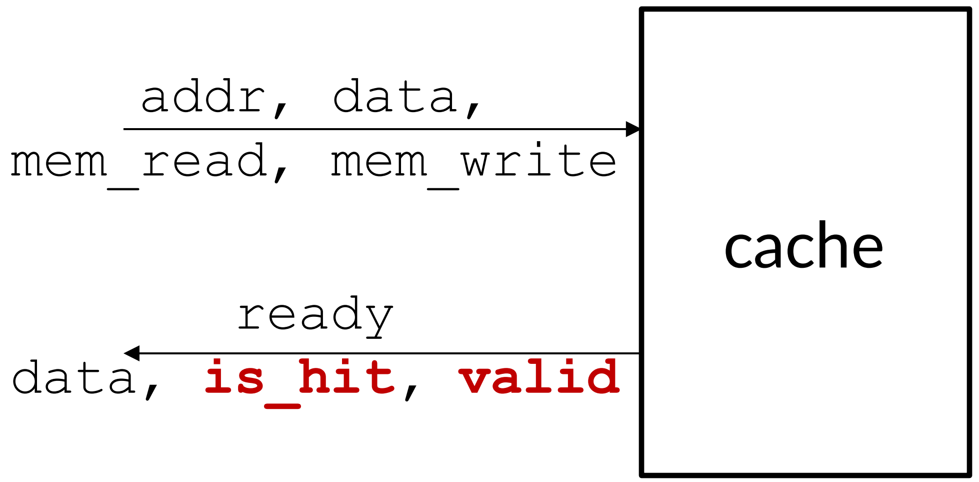
Signals from the data cache

- `data` represents data
  - Ignore `data` when `valid` is false



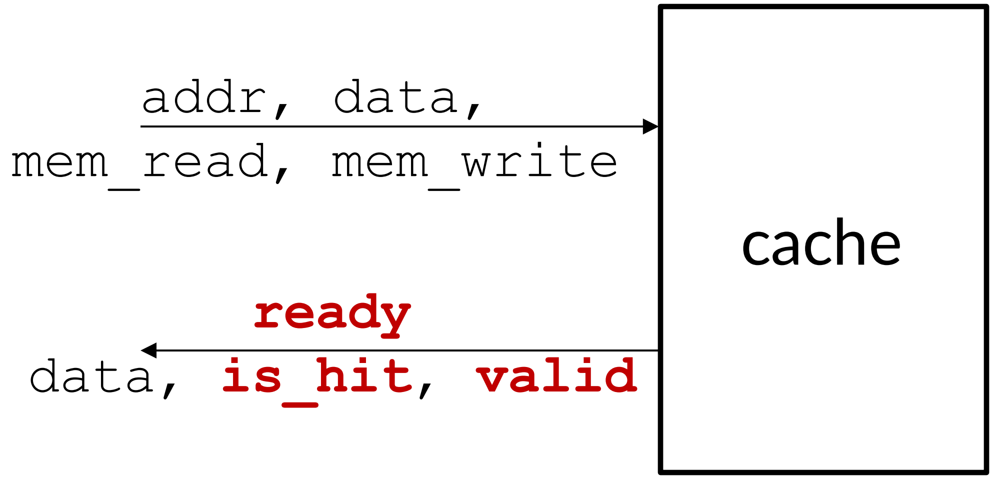
Signals from the data cache

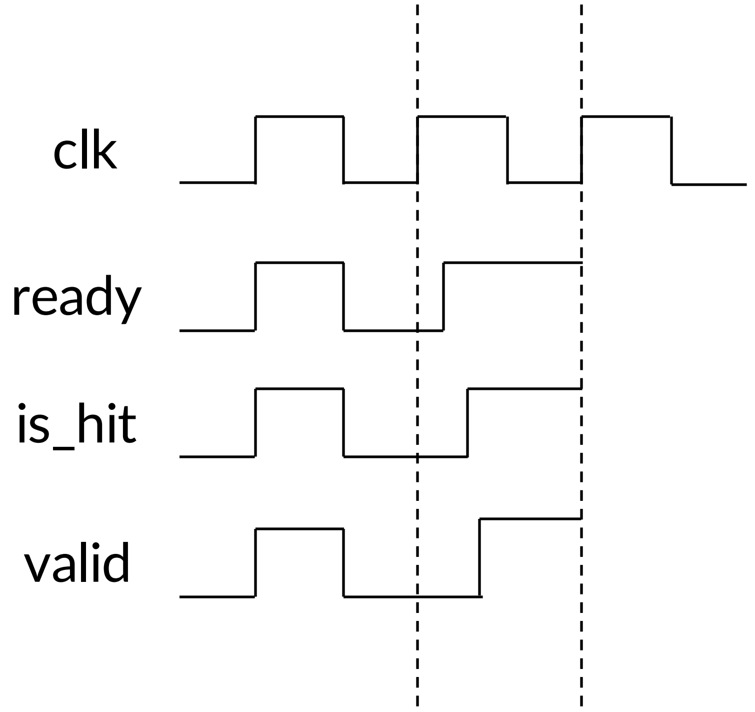
- `is_hit` represents whether a cache hit occurs
  - If a cache miss occurs, stall the pipeline
    - Even if `ready` is true



Signals from the data cache

- Pipeline is not stalled only when:
  - ready, is_hit and valid are true





Data cache design

- The size of a cache line (block) is 16 bytes

32 bits

addr	…	…	…	…	..	4	3	2	1	0
	tags			sets			block offset		4B offset	

# sets	**# ways**	**Block offset 0**	**Block offset 1**	**Block offset 2**	**Block offset 4**
Set 0	Way 0	4B	4B	4B	4B
	Way 1				
	…				

Data cache design

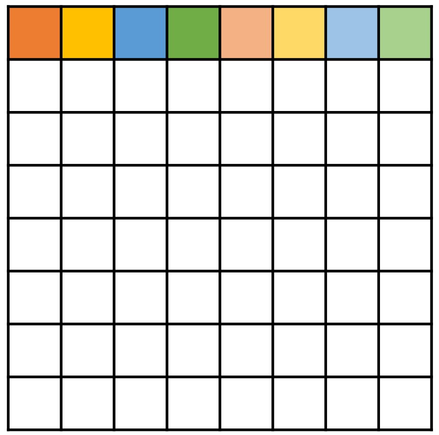
- Asynchronous read:
  - `valid`, `data`, `is_hit`
- Synchronous write
  - Writes to the cache line (from both CPU and memory) should be synchronous
- Write-back, write-allocate
  - Read data from the memory if a write miss occurs

Data cache design

- Replacement policy
  - Choose any way except for MRU way
- Structure
  - Choose between direct-mapped or set-associative (extra point) but not fully-associative
  - Size: 256 Bytes (data bank)
  - You are free to define # of ways and sets
- Each cache line should have:
  - Valid bit
  - Dirty bit
  - Bits for replacement
  - …

Matrix data layout

- Memory layout of the matrix (row-major order)
  - Assume each element of the matrix is 4 B
  - Assume cache line size is 16 B


matrix

address	data
0x00	🟧
0x04	🟨
0x08	🟦
0x0c	🟩
0x10	🟧
0x14	🟨
0x18	🟦
0x1c	🟩

memory

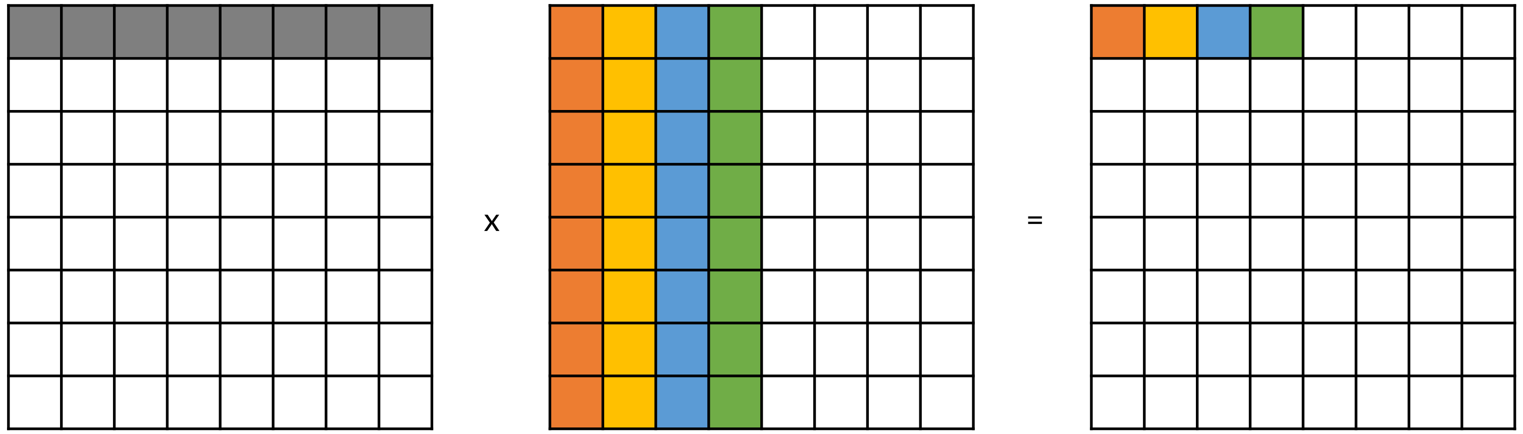
Set 0	🟩	🟦	🟨	🟧
Set 1	🟩	🟦	🟨	🟧

cache

Matrix multiplication

- Naïve implementation
  - Is this cache-friendly? No. Why?

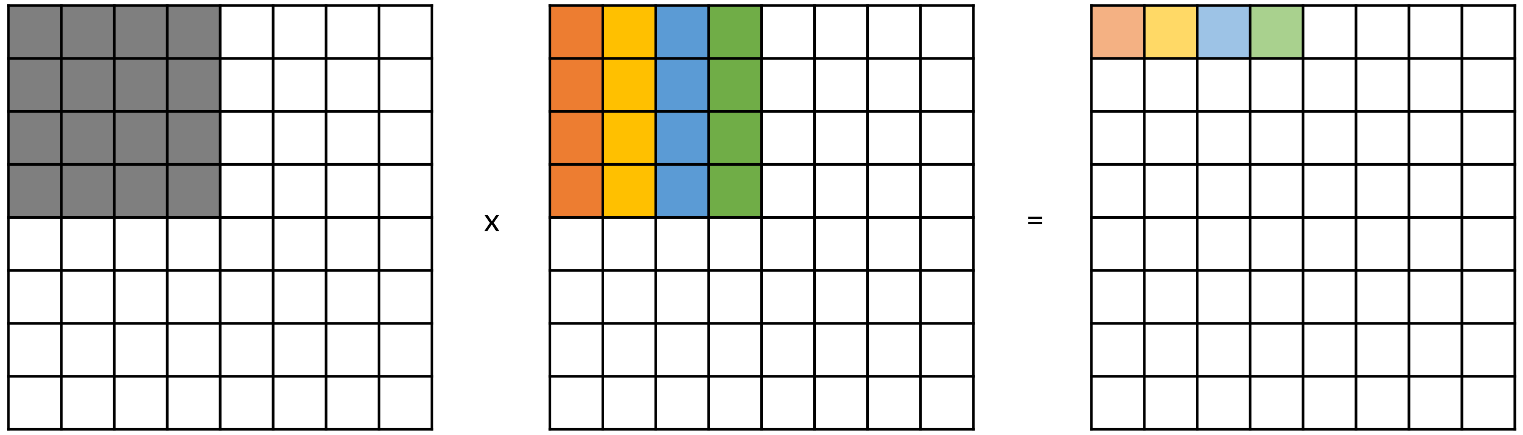
```
// matrix op
for (int m = 0; m < M; ++m) {
  for (int n = 0; n < N; ++n) {
    for (int k = 0; k < K; ++k) {
      c[m][n] += a[m][k] + b[k][n];
    }
  }
}
```



Matrix multiplication

- Tiled implementation
  - Is this cache-friendly? If yes, why?

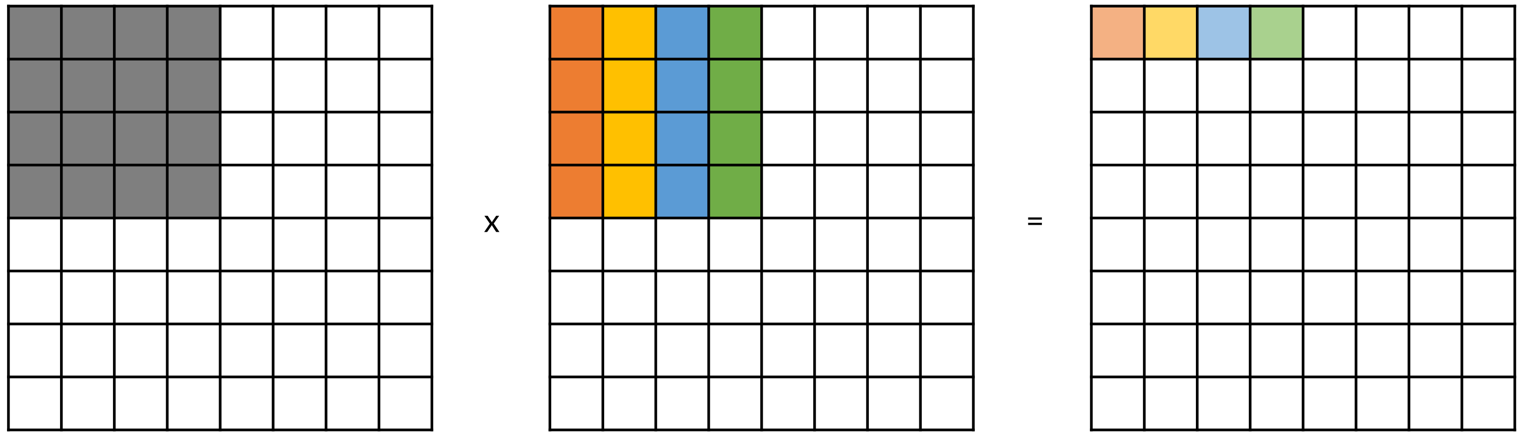
```
// matrix op
for (int tile_m = 0; tile_m < M; tile_m += TILE) {
  for (int tile_n = 0; tile_n < N; tile_n += TILE) {
    for (int tile_k = 0; tile_k < K; tile_k += TILE) {
      for (int m = tile_m; m < tile_m + TILE; ++m) {
        for (int n = tile_n; n < tile_n + TILE; ++n) {
          for (int k = tile_k; k < tile_k + TILE; ++k) {
            c[m][n] += a[m][k] + b[k][n];
          }
        }
      }
    }
  }
}
```



Matrix multiplication

- Tiled implementation
  - Is this cache-friendly? If yes, why?
  - Reuse data (in the cache) as much as possible within each tile
    - The tile size is set to the cache line size

```
// matrix op
for (int tile_m = 0; tile_m < M; tile_m += TILE) {
  for (int tile_n = 0; tile_n < N; tile_n += TILE) {
    for (int tile_k = 0; tile_k < K; tile_k += TILE) {
      for (int m = tile_m; m < tile_m + TILE; ++m) {
        for (int n = tile_n; n < tile_n + TILE; ++n) {
          for (int k = tile_k; k < tile_k + TILE; ++k) {
            c[m][n] += a[m][k] + b[k][n];
          }
        }
      }
    }
  }
}
```



Submission

- Implementation (Deadline: 9:00am, 5/28)
  - Blocking data cache
    - Direct-mapped (no extra credit)
    - N-way associative cache (Full extra credit + 3)
  - You need to follow the rules described in lab_guide.pdf
- Report (Deadline: 23:59, 5/28)
  - The design of the cache
    - Direct-mapped or associative cache
  - Analyze cache hit ratio
    - If you implement associative cache, compare it with direct-mapped cache
    - Explain your replacement policy
    - Naïve matmul vs optimized matmul
    - Why is the cache hit ratio different between two matmul algorithms?
    - What happens to the cache hit ratio if you change the # of sets and # of ways?

Submission

- Implementation file format
  - .zip file name: Lab5_{team_num}_{student1_id}_{student2_id}.zip
  - Contents of the zip file (only *.v):
    - cpu.v
    - …
    - Do not include top.v, InstMemory.v, DataMemory, RegisterFile.v, and CLOG2.v
- Report file format
  - Lab5_{team_num}_{student1_id}_{student2_id}.pdf