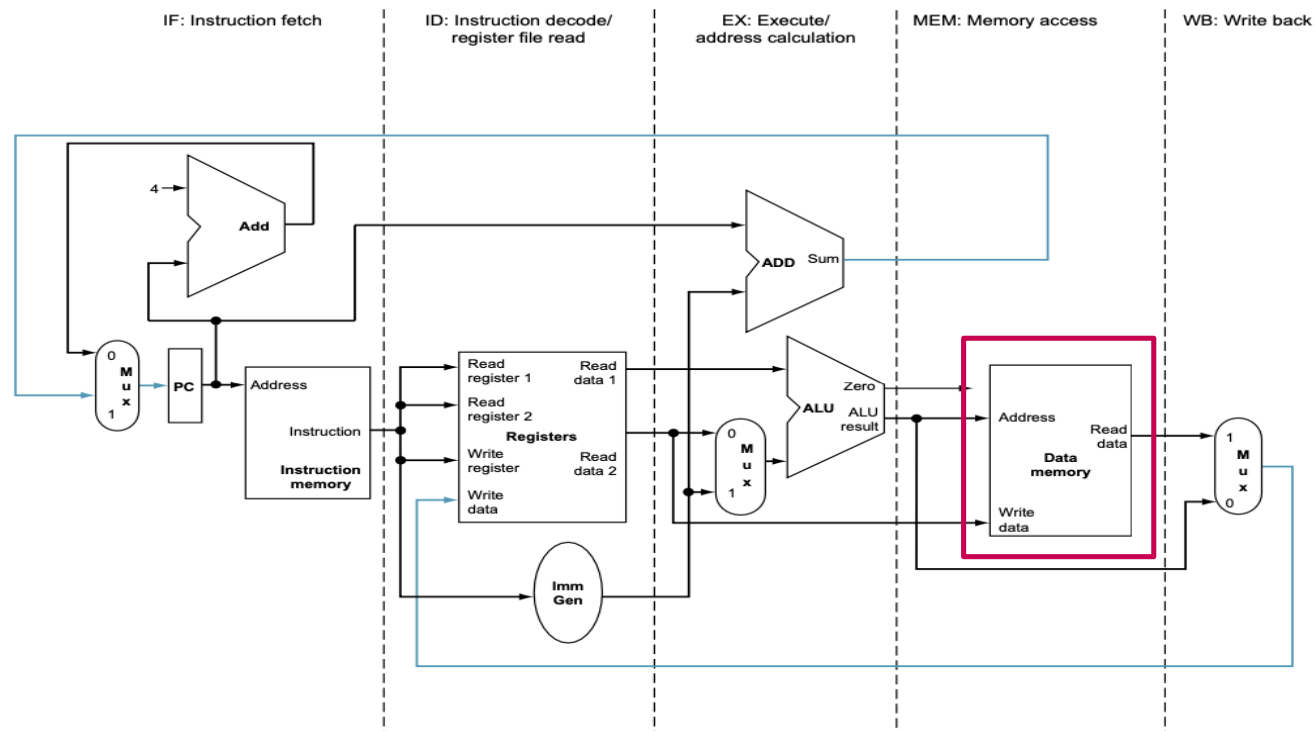# CSED311 Lab5: Cache

Seung-taek Woo & Seonggon Kim

wst9909@postech.ac.kr, sungonuni@postech.ac.kr

Contact the TAs at cs311-2025ta@postech.ac.kr
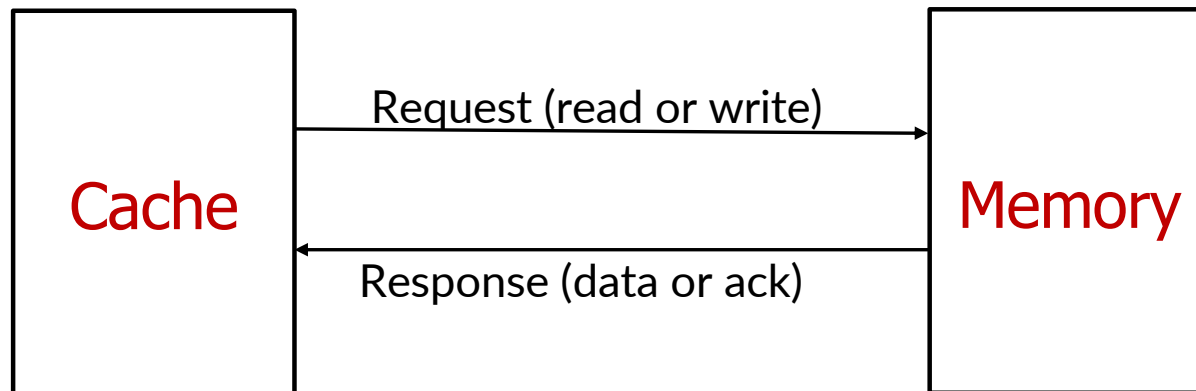
POSTECH

Data cache in Pipelined CPU

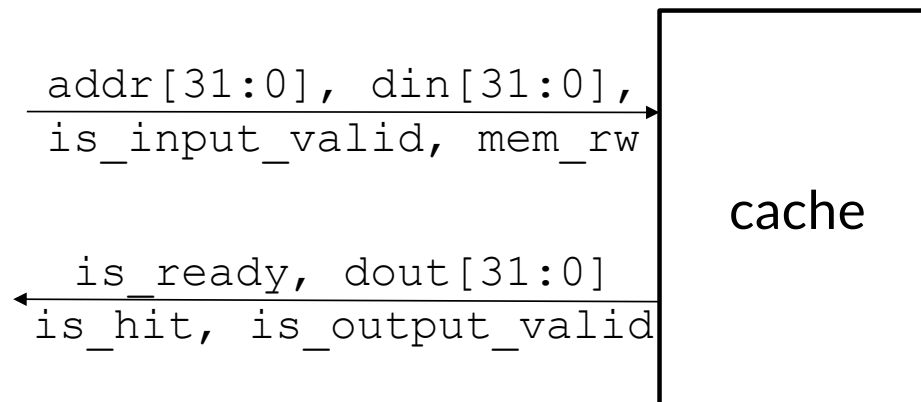- Uses a blocking data cache instead of a "magic memory"

POSTECH

Data flow

- When there is a cache miss, the cache requests data from memory

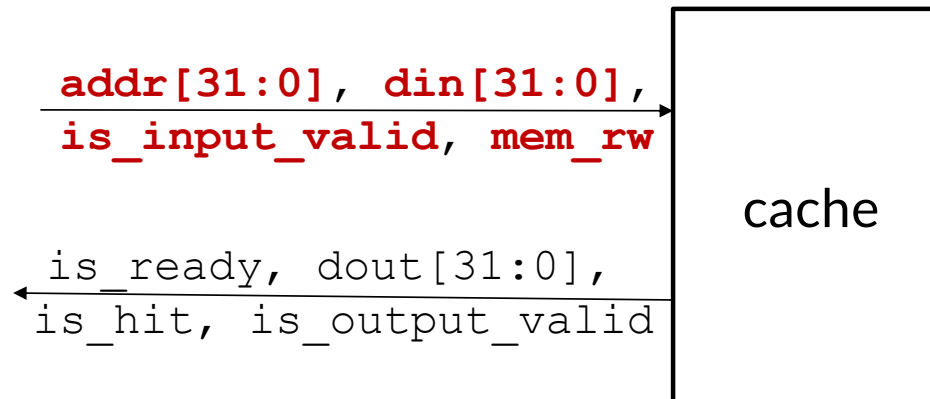- When a dirty cache line is evicted, it should be written back to memory

```
┌──────────┐   Request (read or write)   ┌──────────┐
│          │ ─────────────────────────►  │          │
│  Cache   │                             │  Memory  │
│          │ ◄─────────────────────────  │          │
└──────────┘    Response (data or ack)   └──────────┘
```

*acknowledgement from the memory is implicit in our lab

POSTECH

Signals to / from the cache

addr[31:0], din[31:0],
is_input_valid, mem_rw

cache

is_ready, dout[31:0]
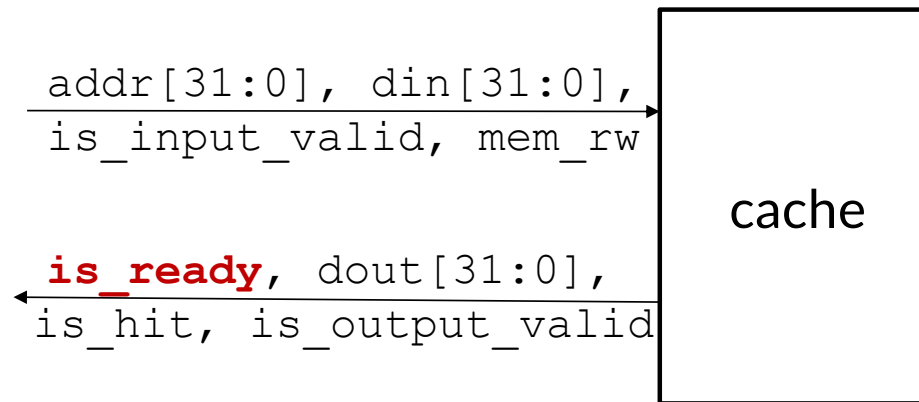is_hit, is_output_valid

POSTECH

Input signals to the data cache

- `addr`: memory address that CPU wants to read or write

- `mem_rw`: access type (0: read, 1: write)

- `din`: data that CPU writes to cache for stores

  - `addr` and `din` should be used only when `is_input_valid` is true

  - `din` should be used only when `mem_rw` is 1

**addr[31:0], din[31:0],
is_input_valid, mem_rw**

is_ready, dout[31:0],
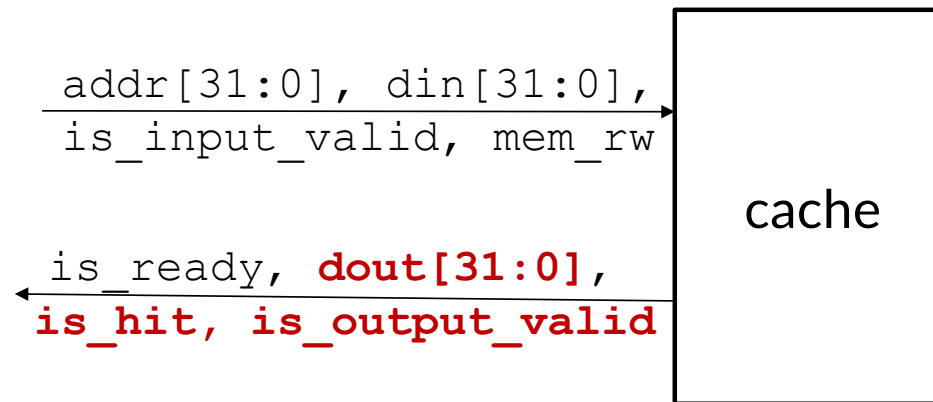is_hit, is_output_valid

cache

POSTECH

Output signals from the data cache

- `is_ready` indicates the status of the cache
  - True if cache is ready to accept a request
  - False if cache is busy serving a prior request
    - Cache cannot accept a request currently. LD/ST would be stalled
- In the 5-stage pipeline, `is_ready` must be true when LD/ST goes into MEM stage

```
addr[31:0], din[31:0],
is_input_valid, mem_rw   ───▶

is_ready, dout[31:0],
◀───  is_hit, is_output_valid
```
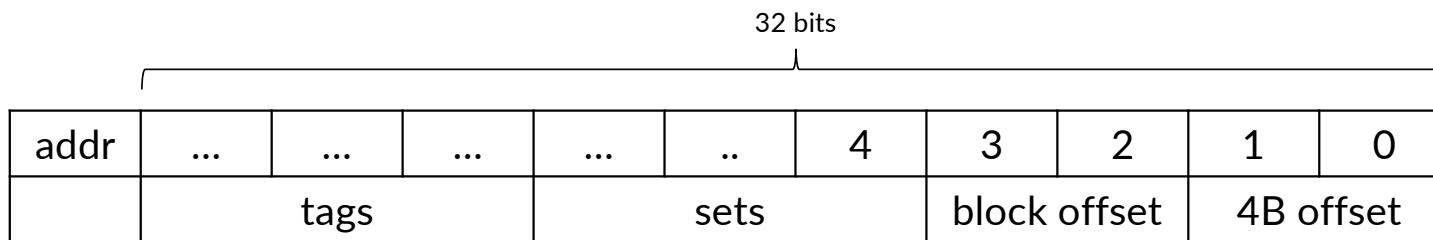
cache

POSTECH

Output signals from the data cache

- `is_output_valid` indicates whether `dout` and `is_hit` are valid(doesn't mean True) or not.

- `dout`: data accessed from cache (for read)

- `is_hit` indicates whether a cache hit occurred

- When the output from cache are valid, LD/ST instructions can use them and continue execution

```
addr[31:0], din[31:0],
is_input_valid, mem_rw
```

```
is_ready, dout[31:0],
is_hit, is_output_valid
```

cache

POSTECH

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 3
	Way 0	4B	4B	4B	4B
Set 0	Way 1				
	…				

POSTECH

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

POSTECH

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
  - …

POSTECH

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
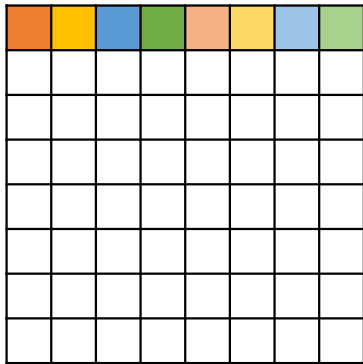
matrix

address	data
0x00	
0x04	
0x08	
0x0c	
0x10	
0x14	
0x18	
0x1c	

memory

Set 0				
Set 1				
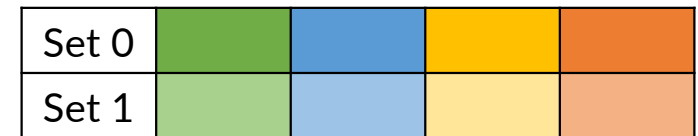
cache

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];
    }
  }
}
```
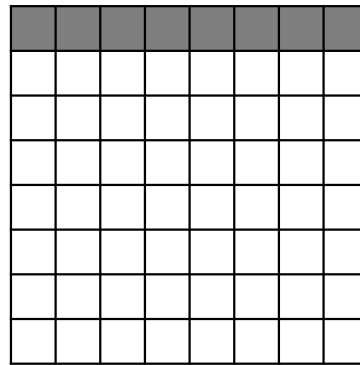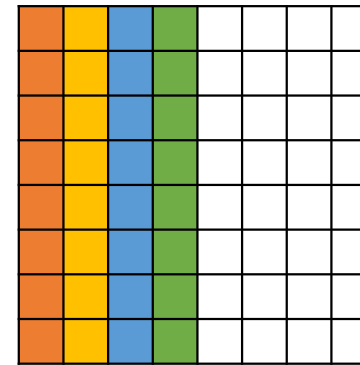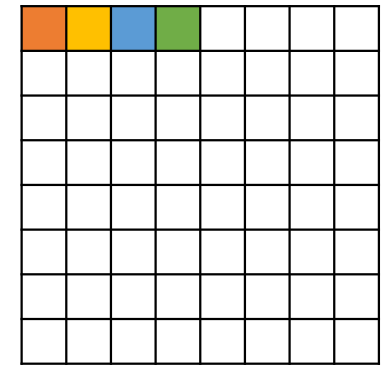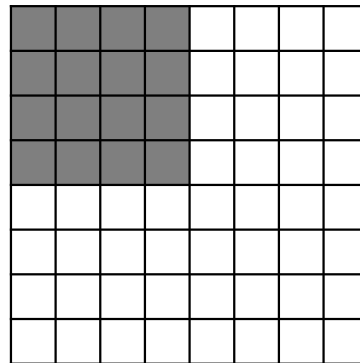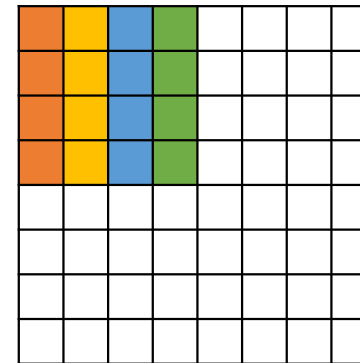
Matrix multiplication

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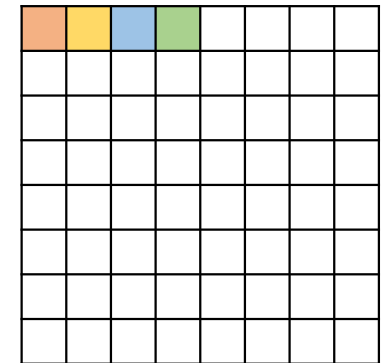
- Is this cache-friendly? If yes, why?

```
// 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

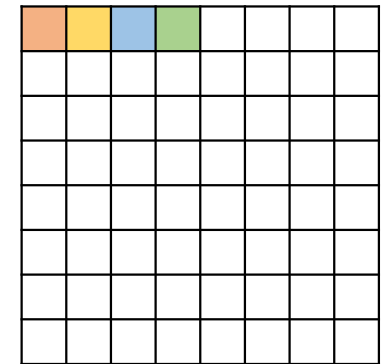   - The tile size is set to the cache line size

```
// 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];
          }
        }
      }
    }
  }
}
```
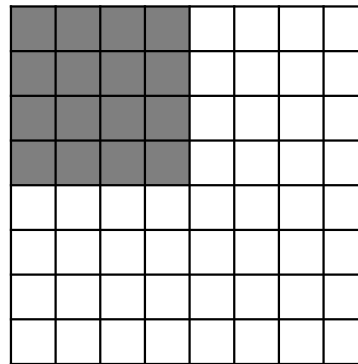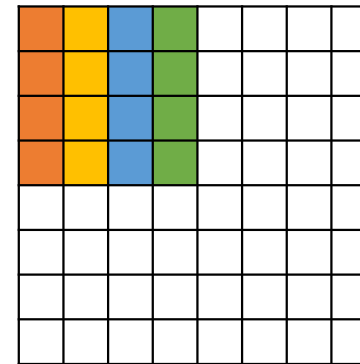
Submission

- Implementation (Deadline: 9:00 am, 5/27)
  - 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: 18:00, 5/27)
  - 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?

POSTECH

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