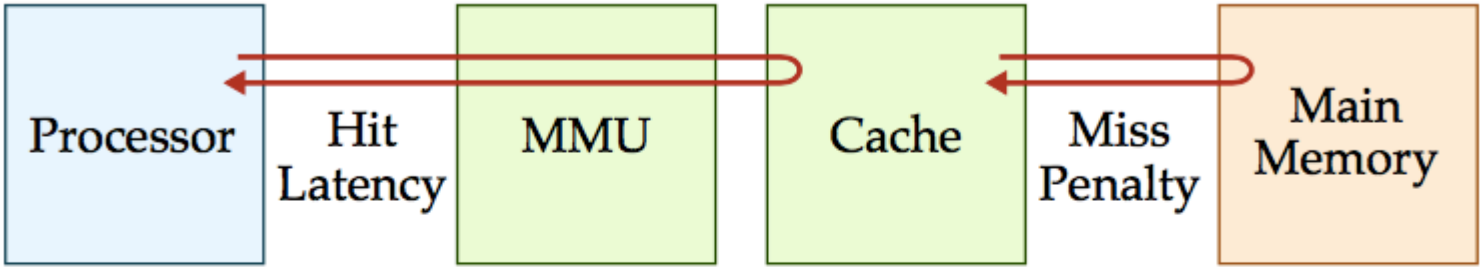
Lecture 15
Process and Memory Integration

Xuan ‘Silvia’ Zhang
Washington University in St. Louis

http://classes.engineering.wustl.edu/ese566/

Analyze Memory Performance

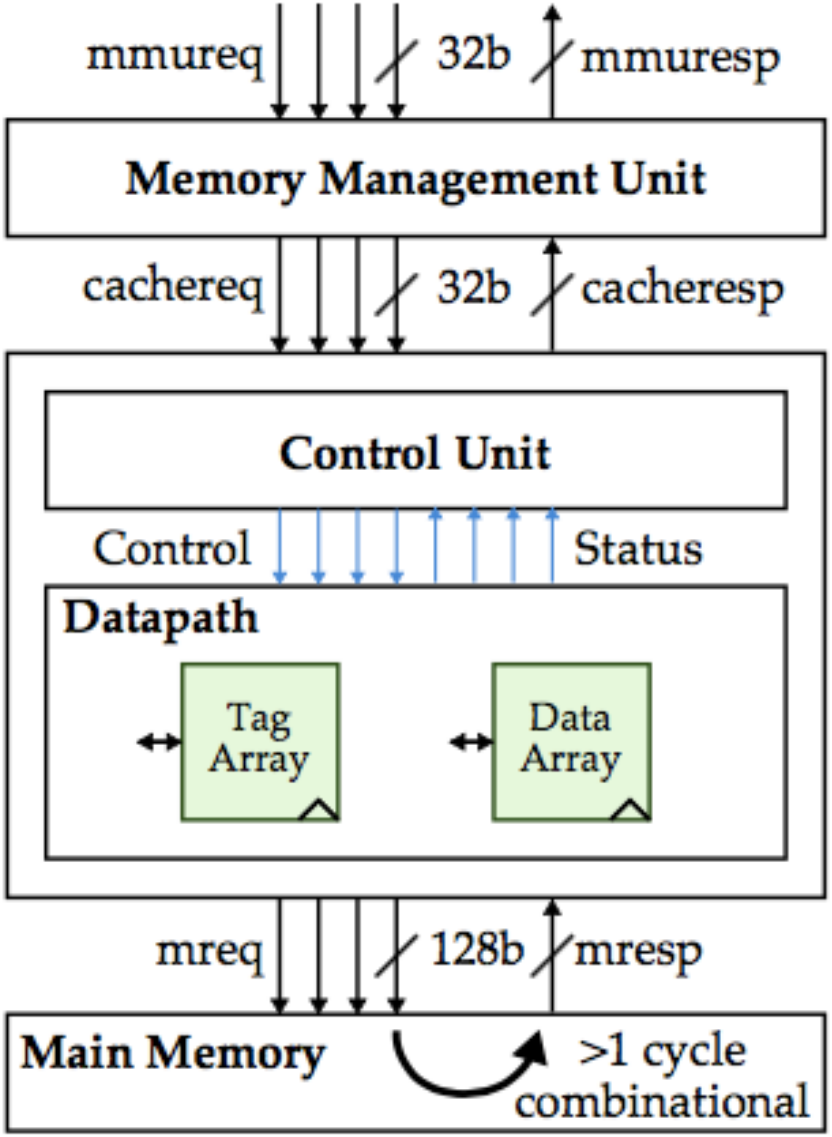


Microarchitecture	Hit Latency	Extra Accesses for Translation
FSM Cache	>1	1+
Pipelined Cache	≈1	1+
Pipelined Cache + TLB	≈1	≈0

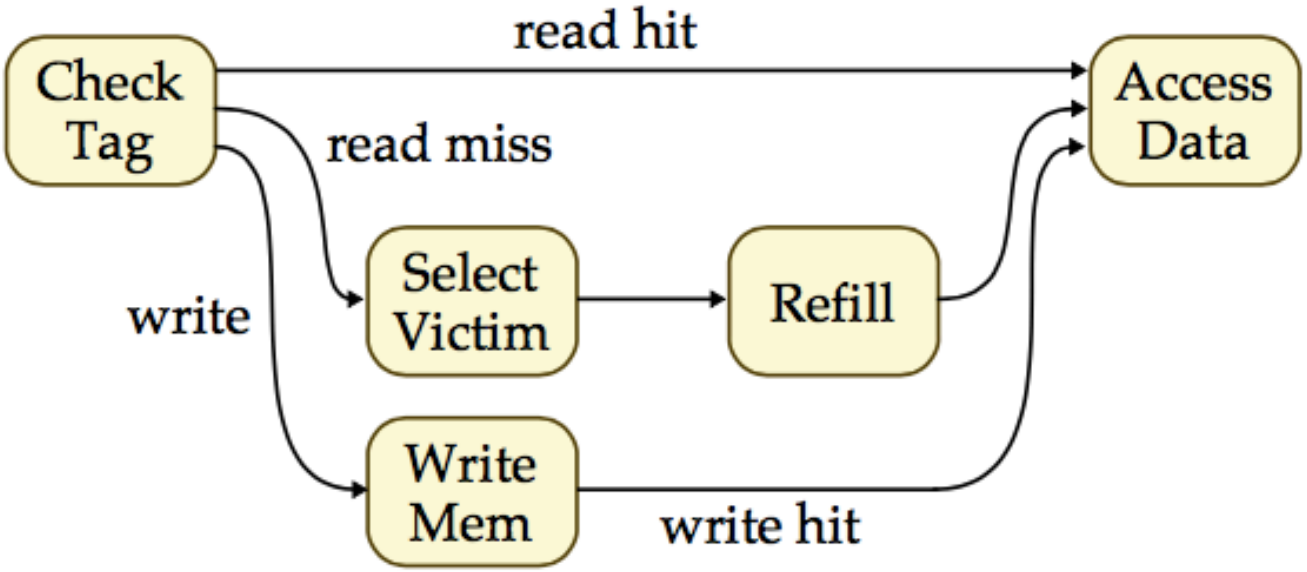
Transactions and Steps, Now for Memory

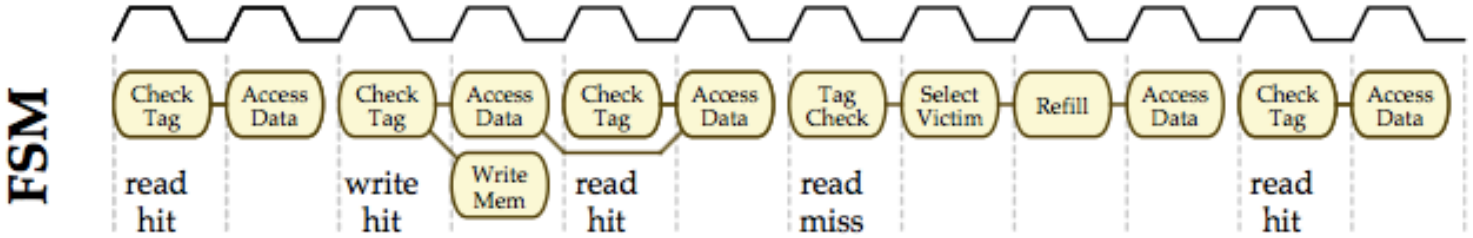
- Executing a memory access involves a sequence of steps
  - check tag: check one or more tags in cache
  - select victim: select victim line from cache using replacement policy
  - evict victim: evict victim line from cache and write victim to memory
  - refill: refill requested line by reading line from memory
  - write mem: write requested word to memory
  - access data: read or write requested word in cache

Memory Microarchitecture Overview

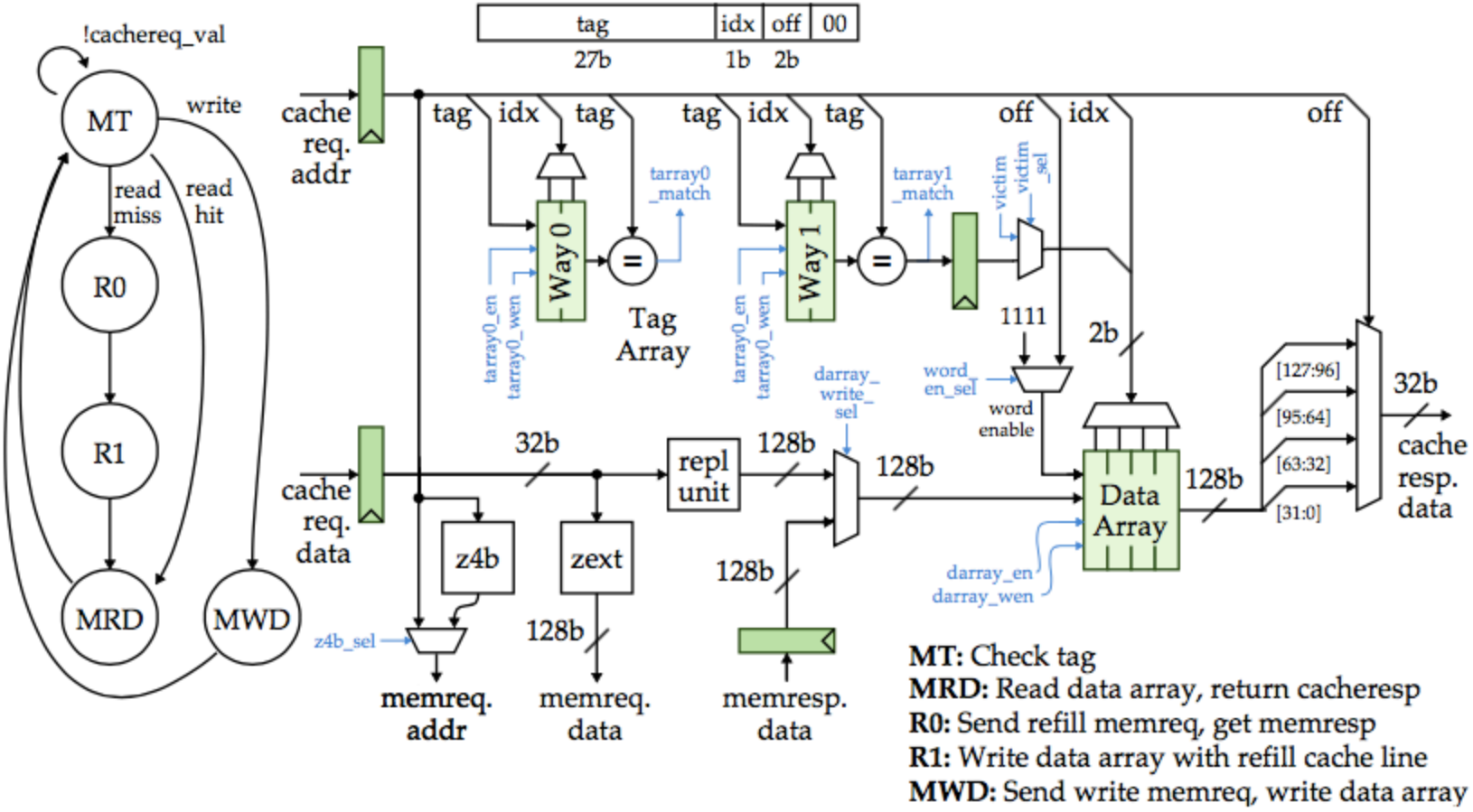


High-level Idea for FSM Cache

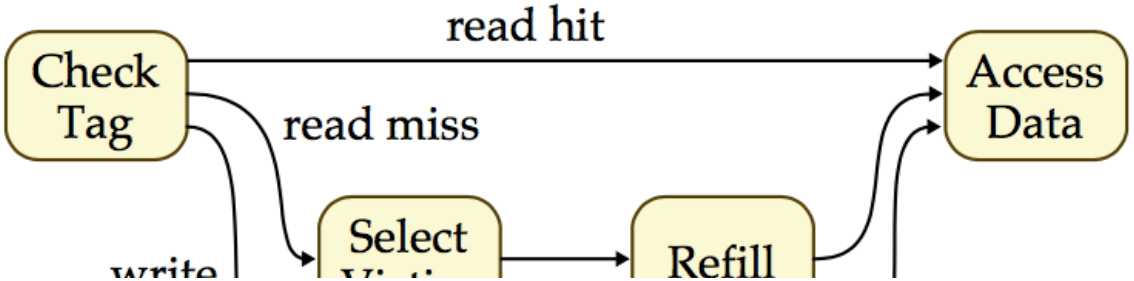


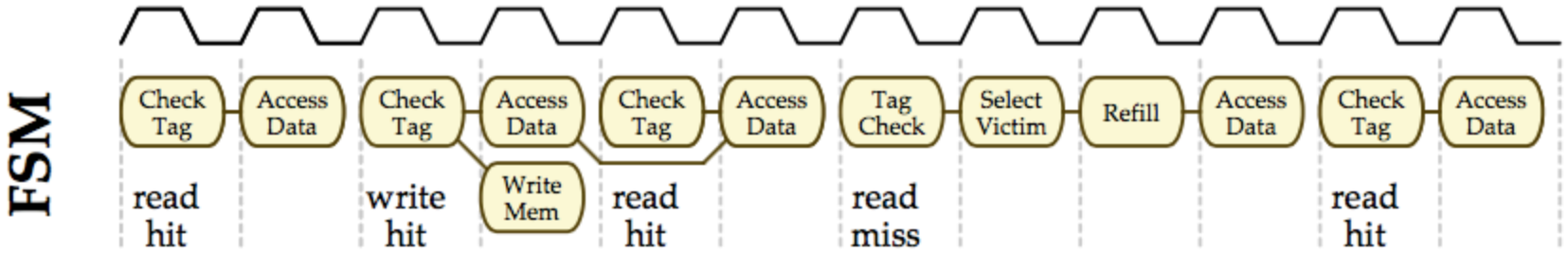


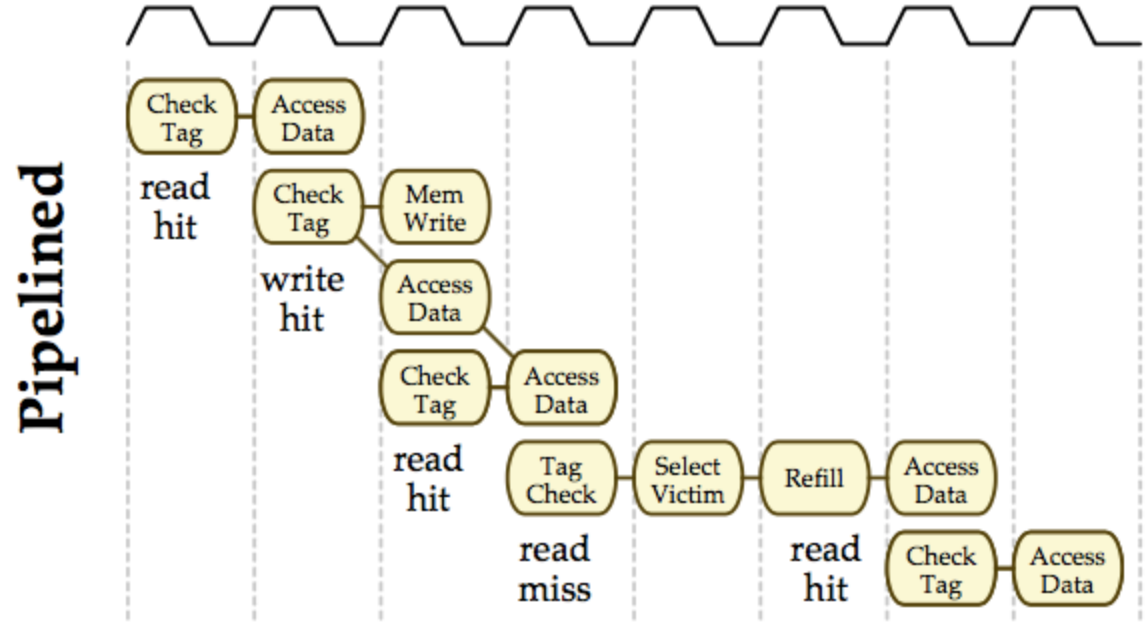
FSM Cache Datapath



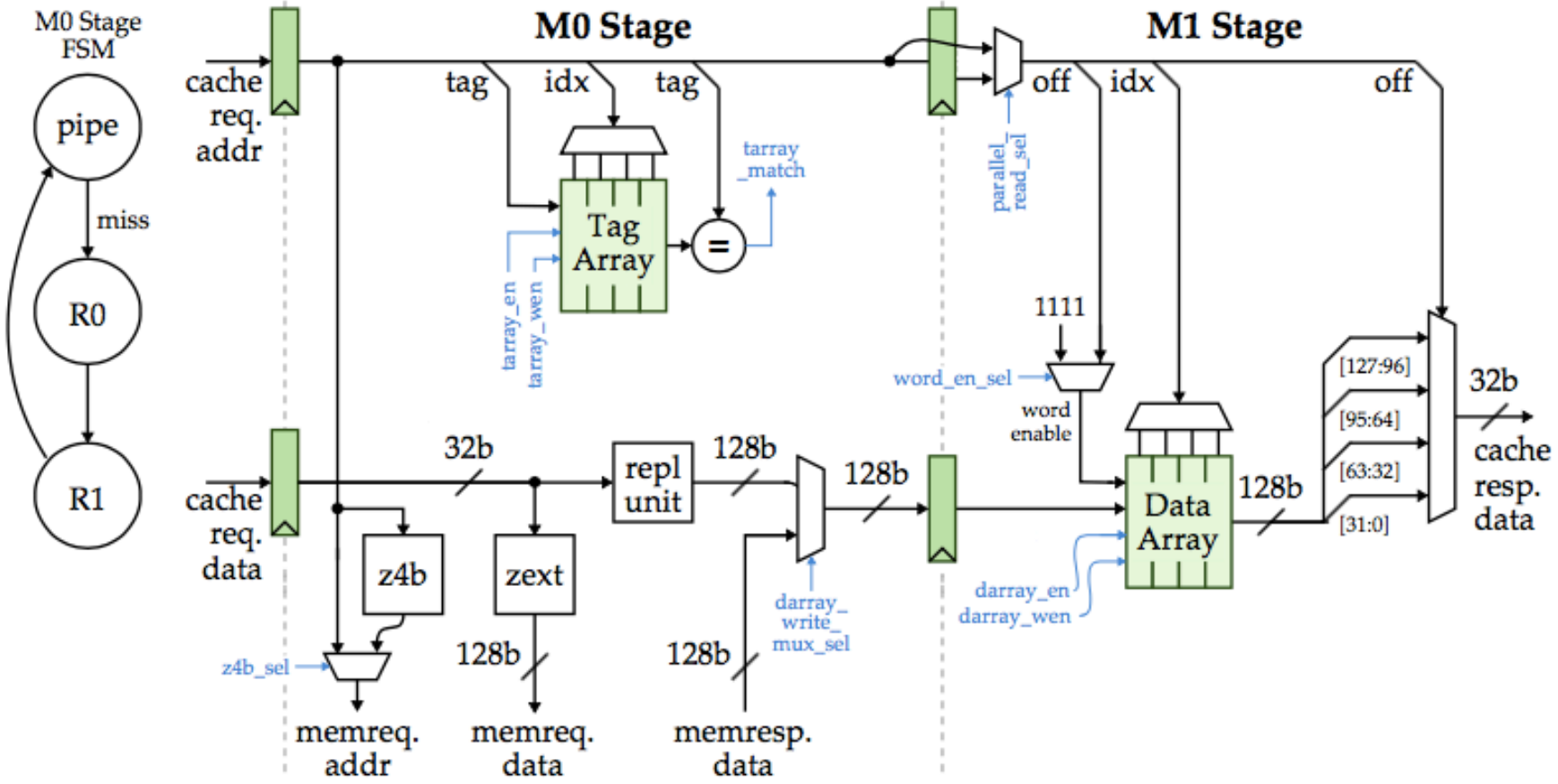
High-level Idea for Pipelined Cache



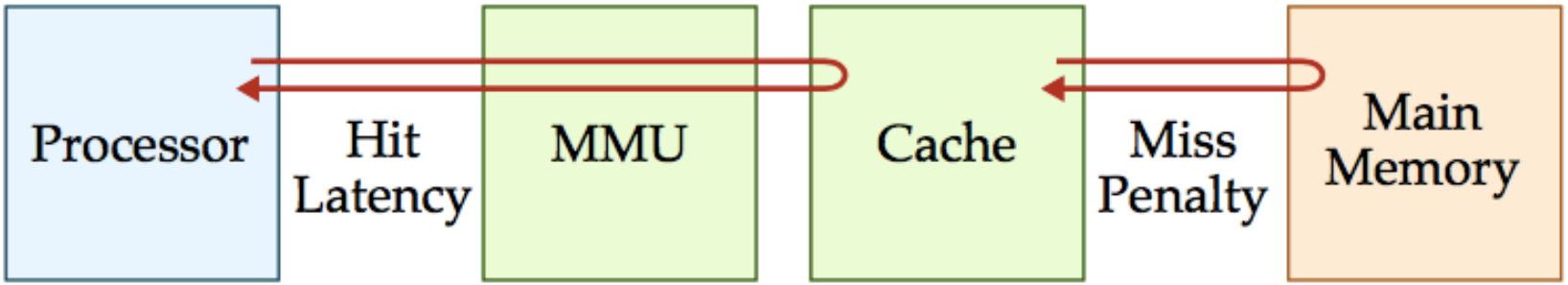




Pipeline Cache Datapath



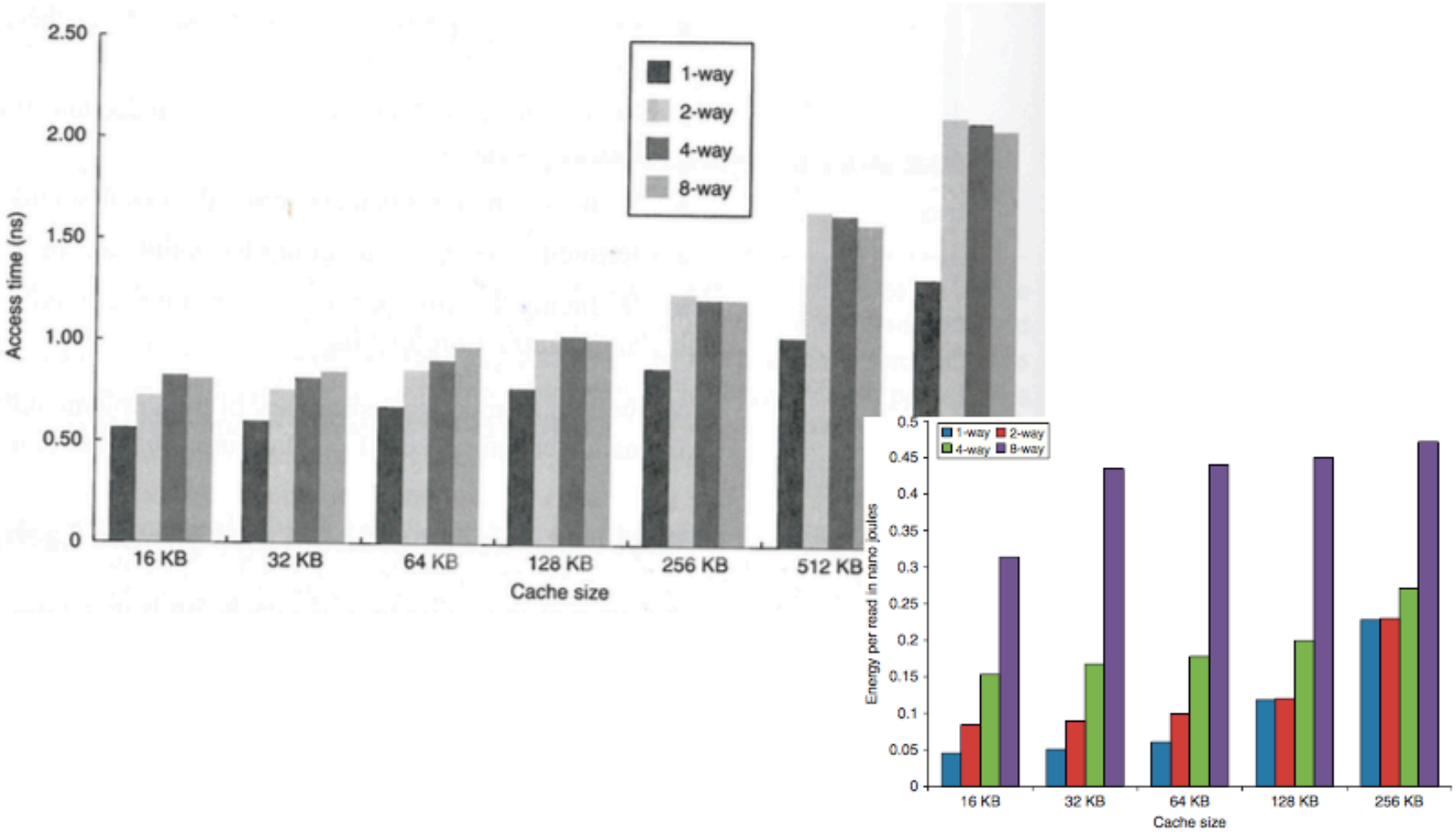
Cache Microarchitecture Optimizations



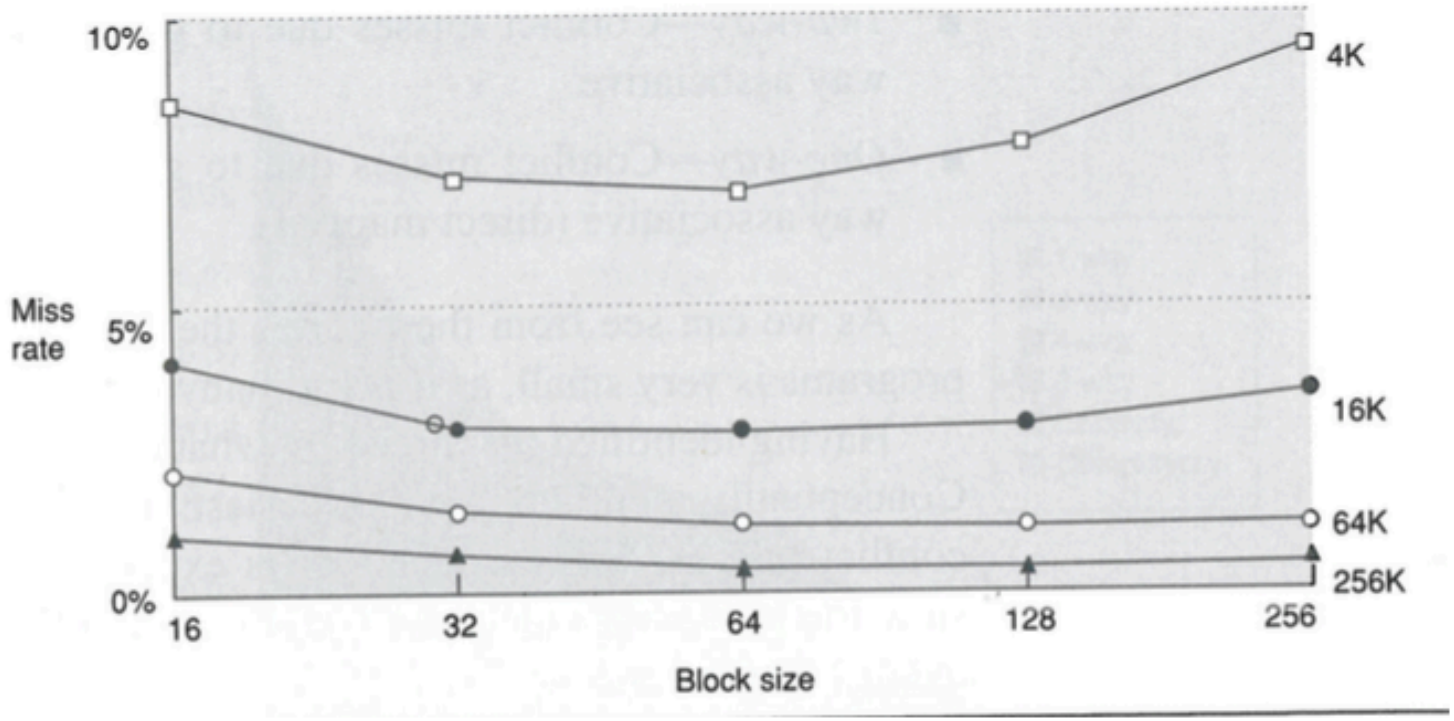
$$\text{AMAL} = \text{Hit Latency} + (\text{Miss Rate} \times \text{Miss Penalty})$$

- Reduce hit time
  - Small and simple caches
- Reduce miss penalty
  - Multi-level cache hierarchy
  - Prioritize reads
- Reduce miss rate
  - Large block size
  - Large cache size
  - High associativity
  - Hardware prefetching
  - Compiler optimizations

Reduce Hit Latency: Small & Simple Caches

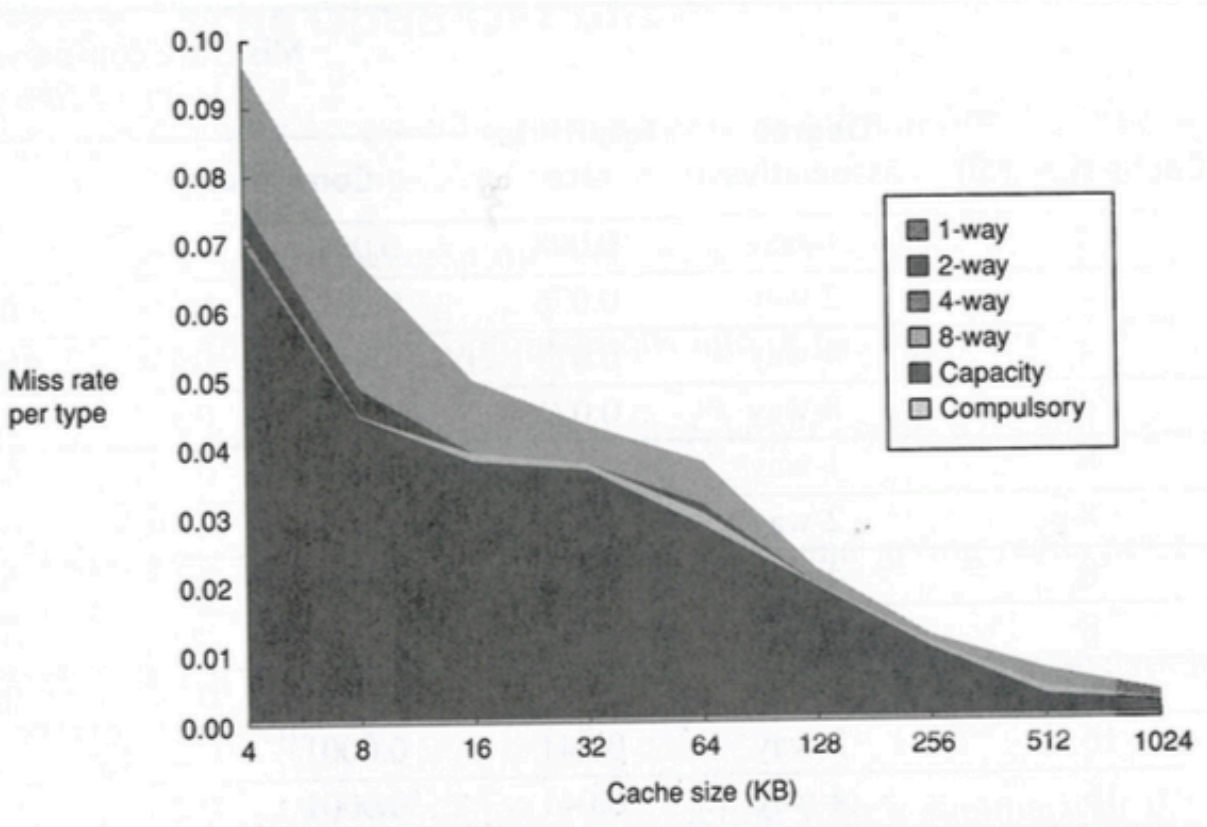


Reduce Miss Rate: Large Block Size



- Less tag overhead
- Exploit fast burst transfers from DRAM and over wide on-chip busses
- Can waste bandwidth if data is not used
- Fewer blocks → more conflicts

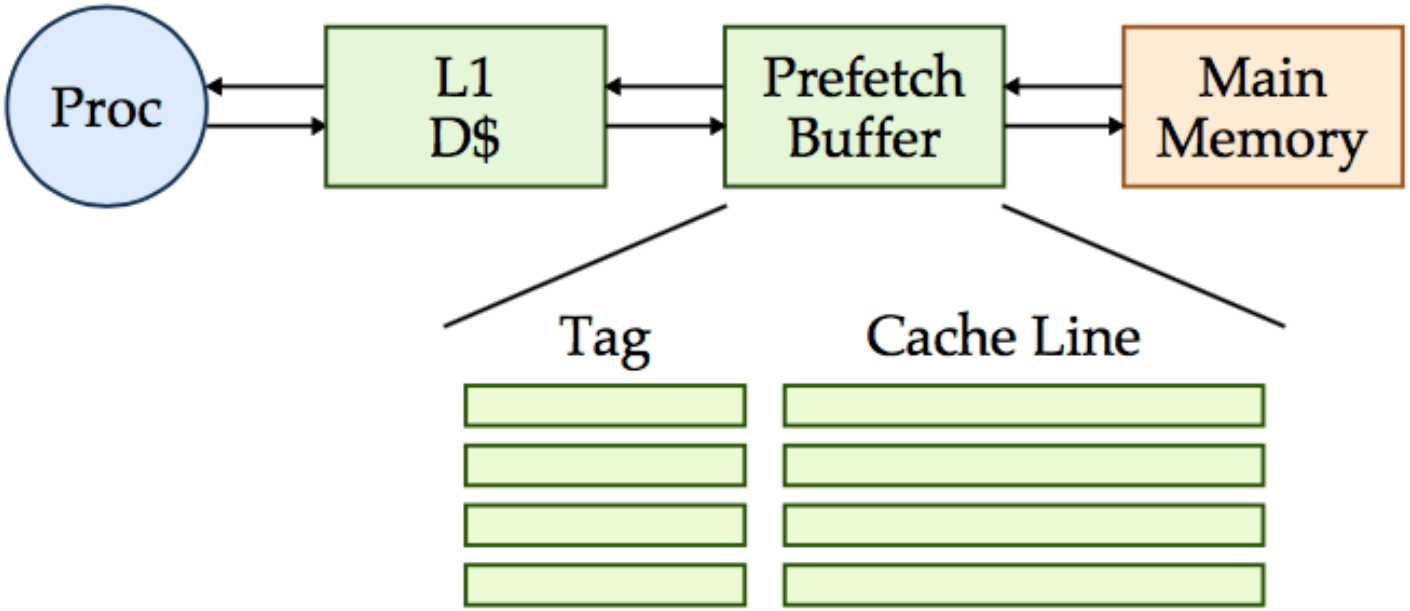
Reduce Miss Rate: Large Cache Size or High Associativity



If cache size is doubled, miss rate usually drops by about $\sqrt{2}$

Direct-mapped cache of size N has about the same miss rate as a two-way set-associative cache of size N/2

Reduce Miss Rate: Hardware Prefetching



- Previous techniques only help capacity and conflict misses
- Hardware prefetcher looks for patterns in miss address stream
- Attempts to predict what the next miss might be
- Prefetches this next miss into a pfetch buffer
- Very effective in reducing compulsory misses for streaming accesses

Reduce Miss Rate: Compiler Optimization

- Restructuring code affects the data block access sequence
  - Group data accesses together to improve spatial locality
  - Re-order data accesses to improve temporal locality
- Prevent data from entering the cache
  - Useful for variales that will only be accessed once before eviction
  - Needs mechanism for software to tell hardware not to cache data ("no-allocate" instruction hits or page table bits)
- Kill data that will never be used again
  - Streaming data exploits spatial locality but not temporal locality
  - Replace into dead-cache locations

Loop Interchange and Fusion

```
for(j=0; j < N; j++) {
    for(i=0; i < M; i++) {
       x[i][j] = 2 * x[i][j];
    }
}
```

```
for(i=0; i < M; i++) {
    for(j=0; j < N; j++) {
       x[i][j] = 2 * x[i][j];
    }
}
```

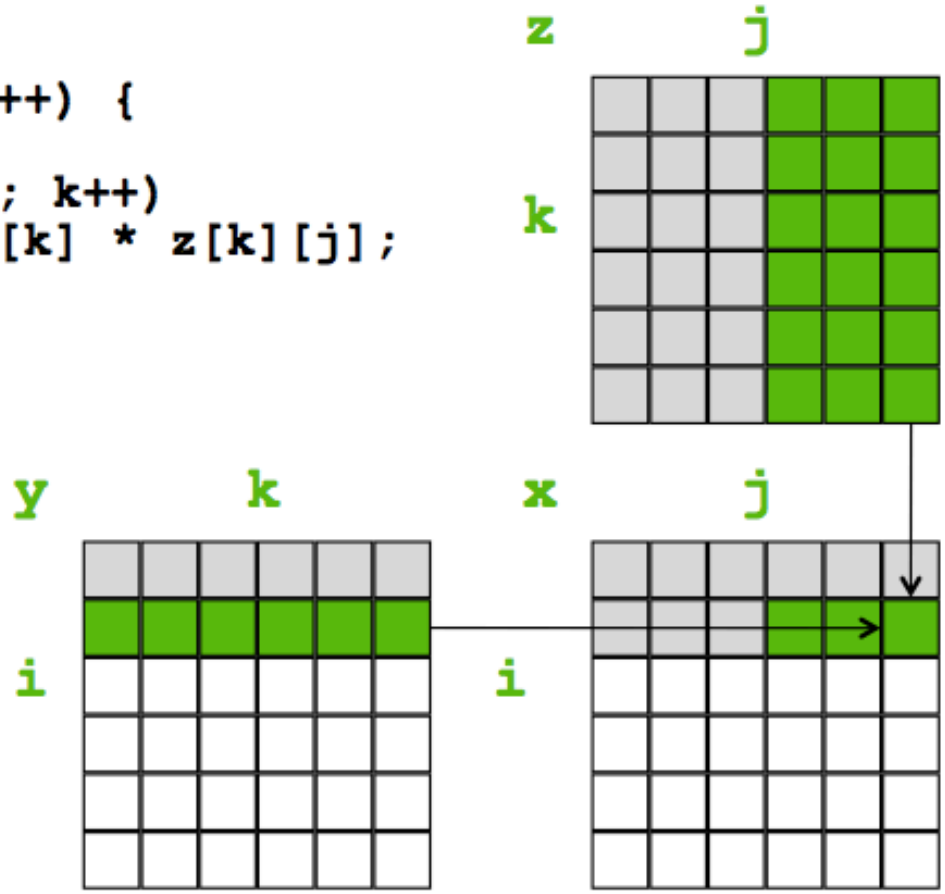
```
for(i=0; i < N; i++)
      a[i] = b[i] * c[i];

for(i=0; i < N; i++)
      d[i] = a[i] * c[i];
```

```
for(i=0; i < N; i++)
{
       a[i] = b[i] * c[i];
       d[i] = a[i] * c[i];
}
```

Matrix Multiply with Naïve Code

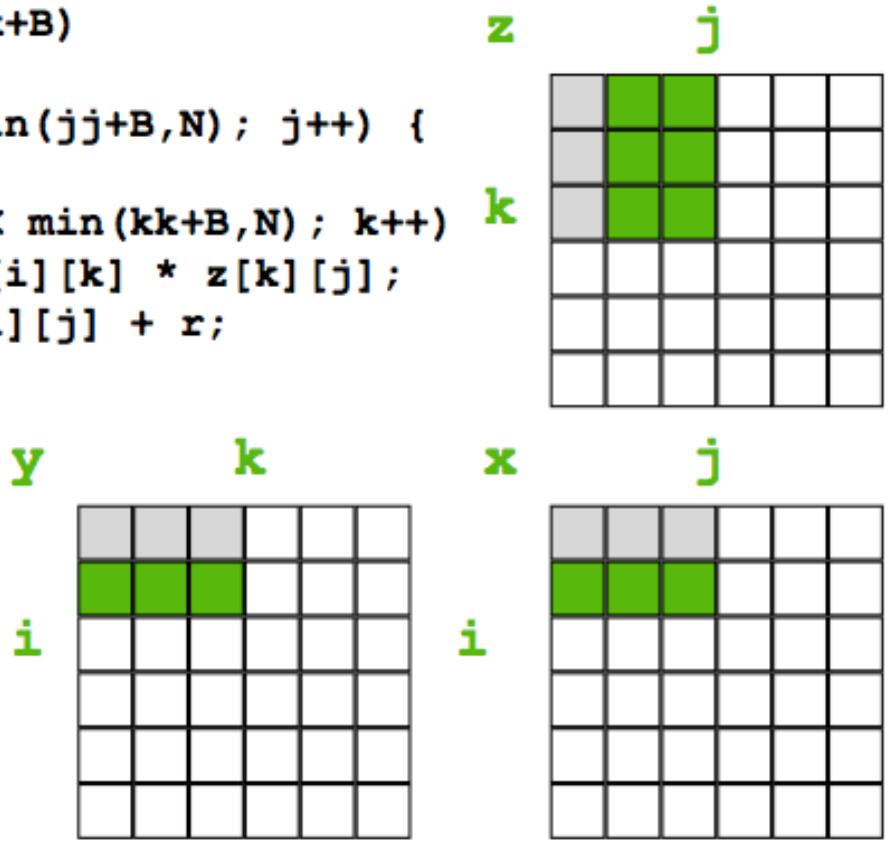
```
for(i=0; i < N; i++)
    for(j=0; j < N; j++) {
        r = 0;
        for(k=0; k < N; k++)
          r = r + y[i][k] * z[k][j];
        x[i][j] = r;
    }
```



Not touched Old access New access

Matrix Multiply with Cache Tiling

```
for(jj=0; jj < N; jj=jj+B)
     for(kk=0; kk < N; kk=kk+B)
        for(i=0; i < N; i++)
            for(j=jj; j < min(jj+B,N); j++) {
               r = 0;
               for(k=kk; k < min(kk+B,N); k++)
                  r = r + y[i][k] * z[k][j];
               x[i][j] = x[i][j] + r;
            }
```



Reduce Miss Penalty: Multi-level Caches

$$\text{AMAL}_{L1} = \text{Hit Latency of L1} + (\text{Miss Rate of L1} \times \text{AMAL}_{L2})$$

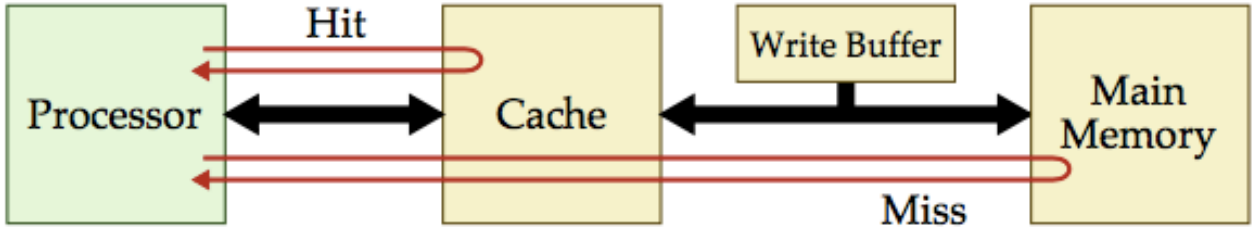
$$\text{AMAL}_{L2} = \text{Hit Latency of L2} + (\text{Miss Rate of L2} \times \text{Miss Penalty of L2})$$

- Local miss rate = misses in cache / accesses to cache
- Global miss rate = misses in cache / processor memory accesses
- Misses per instruction = misses in cache / number of instructions

Reduce Miss Penalty: Multi-level Caches

- Use smaller L1 is there is also a L2
  - Trade increased L1 miss rate for reduced L1 hit time & L1 miss penalty
  - Reduces average access energy
- Use simpler write-through L1 with on-chip L2
  - Write-back L2 cahce absorbs write traffic, doesn't go off-chip
  - Simplifies processor pipeline
  - Simplifies on-chip coherence issues
- Inclusive Multilevel Cache
  - Inner cache holds copy of data in outer cache
  - External coherence is simpler
- Exclusive Multilevel Cache
  - Inner cache may hold data in outer cache
  - Swap lines between inner/outer cache on miss

Reduce Miss Penalty: Prioritize Reads

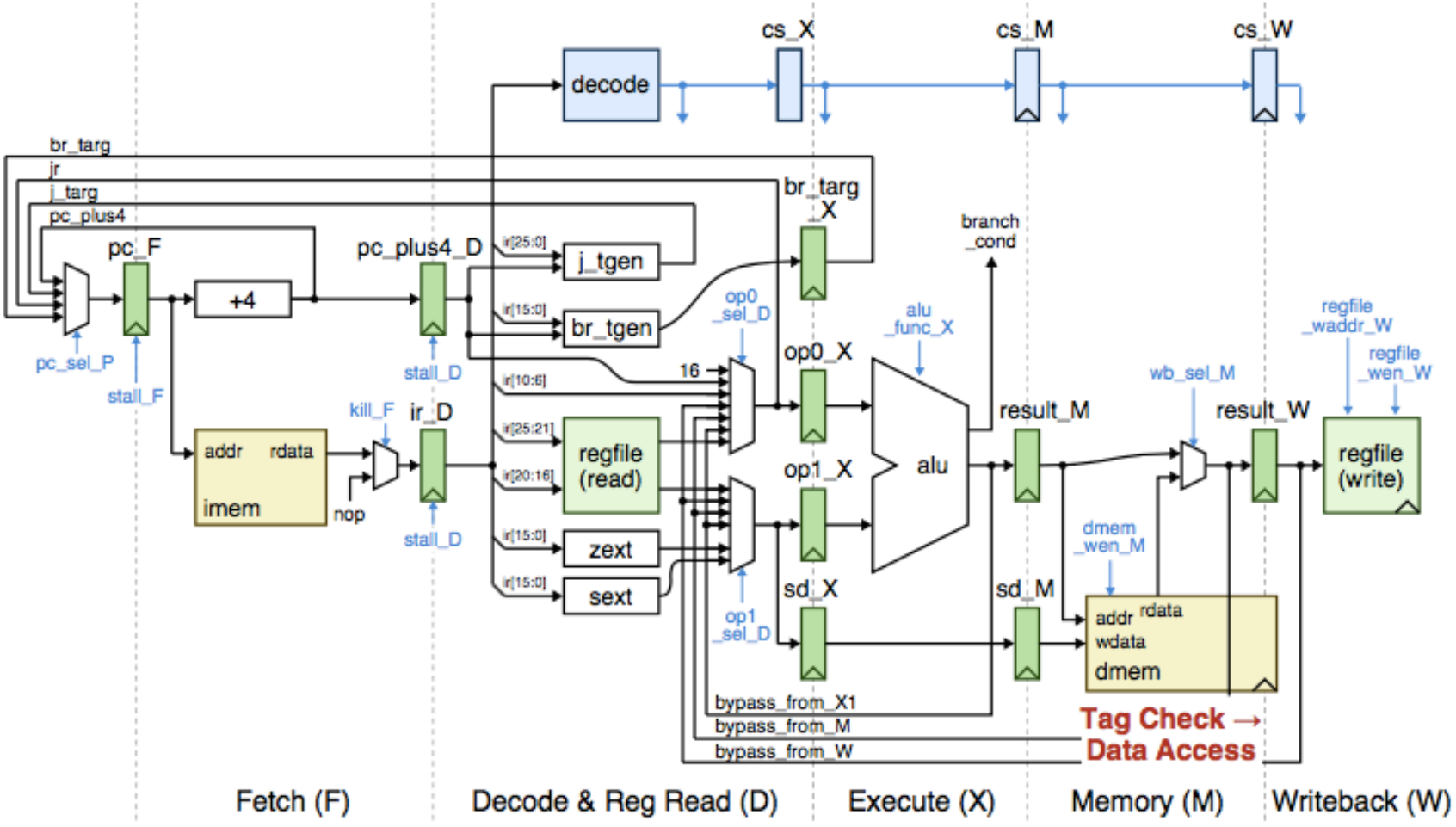


- Processor not stalled on writes, and read misses can go ahead of writes to main memory
- Write buffer may hold updated value of location needed by read miss
  - On read miss, wait for write buffer to be empty
  - Check write buffer addresses and bypass

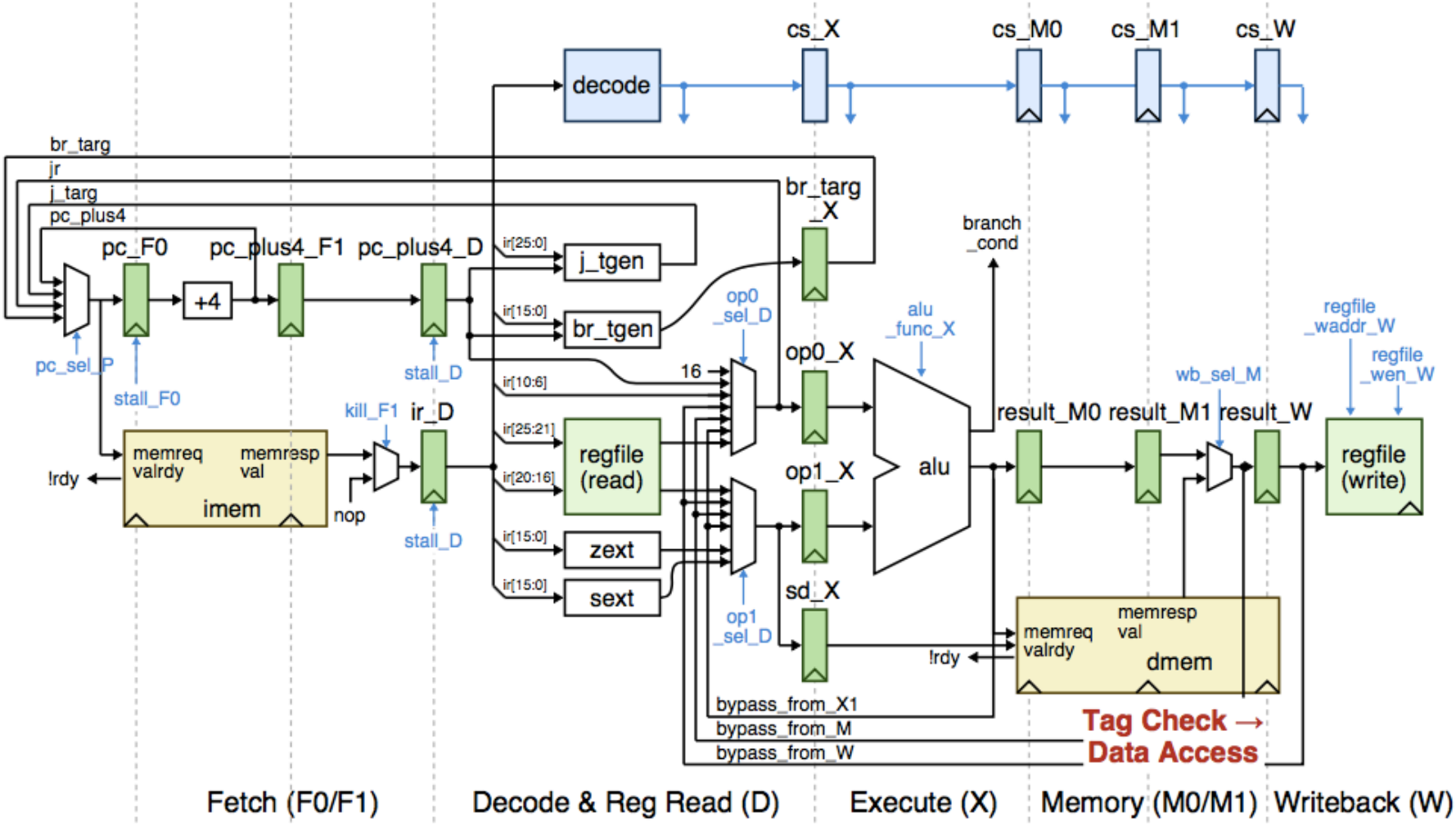
Cache Optimizations Impact on AMAL

Technique	Hit Lat	Miss Rate	Miss Penalty	BW	HW
Smaller caches	–	+			0
Avoid TLB before indexing	–				1
Large block size		–	+		0
Large cache size	+	–			1
High associativity	+	–			1
Hardware prefetching		–			2
Compiler optimizations		–			0
Multi-level cache			–		2
Prioritize reads			–		1
Pipelining	+			+	1

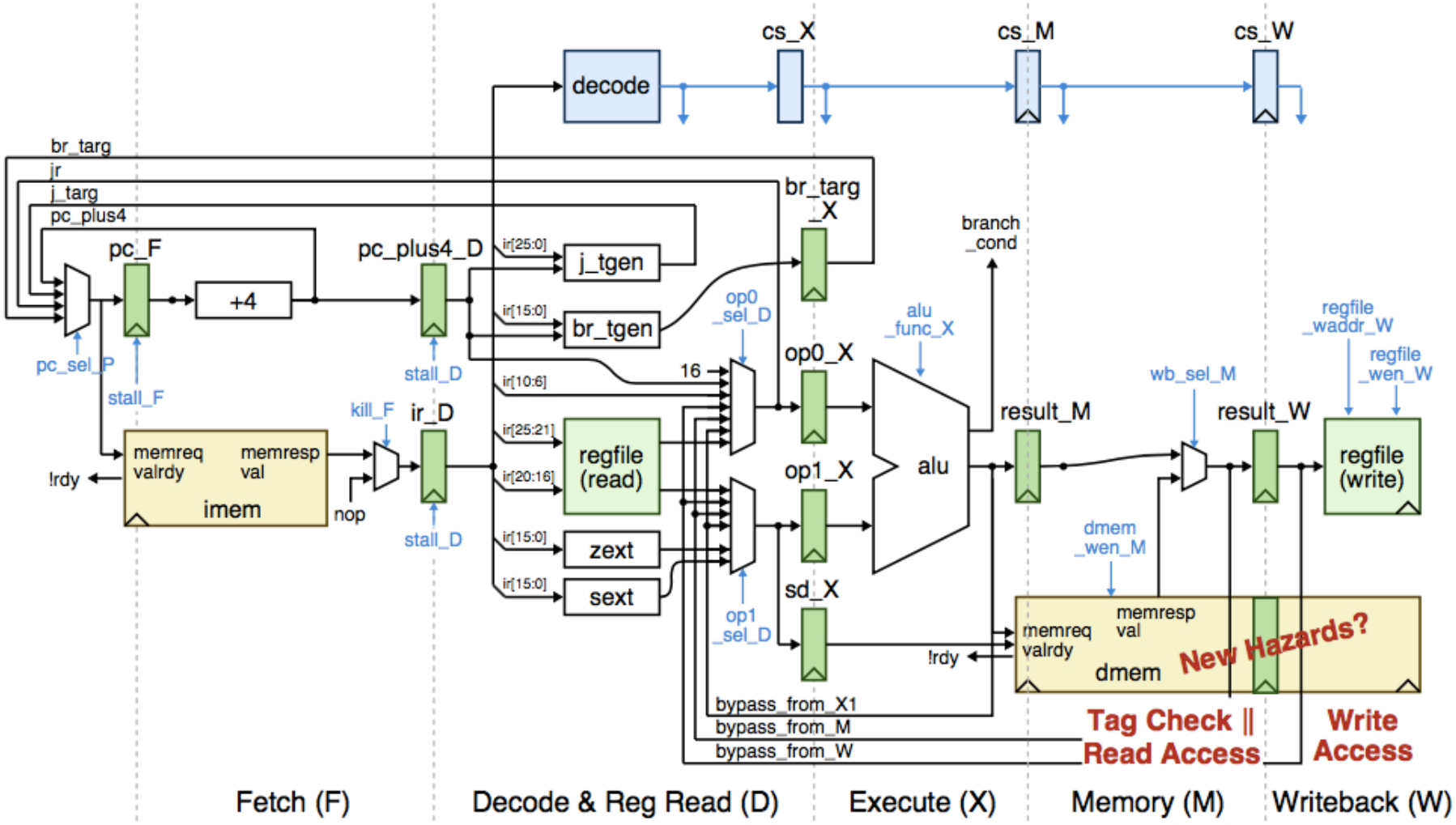
Processor and L1 Cache Interface: Zero-cycle Hit Latency with Tightly Coupled Interface



Processor and L1 Cache Interface: Two-cycle Hit Latency with Val/Rdy Interface



Processor and L1 Cache Interface: Parallel Read, Pipelined Write Hit Path



Questions?

Comments?

Discussion?

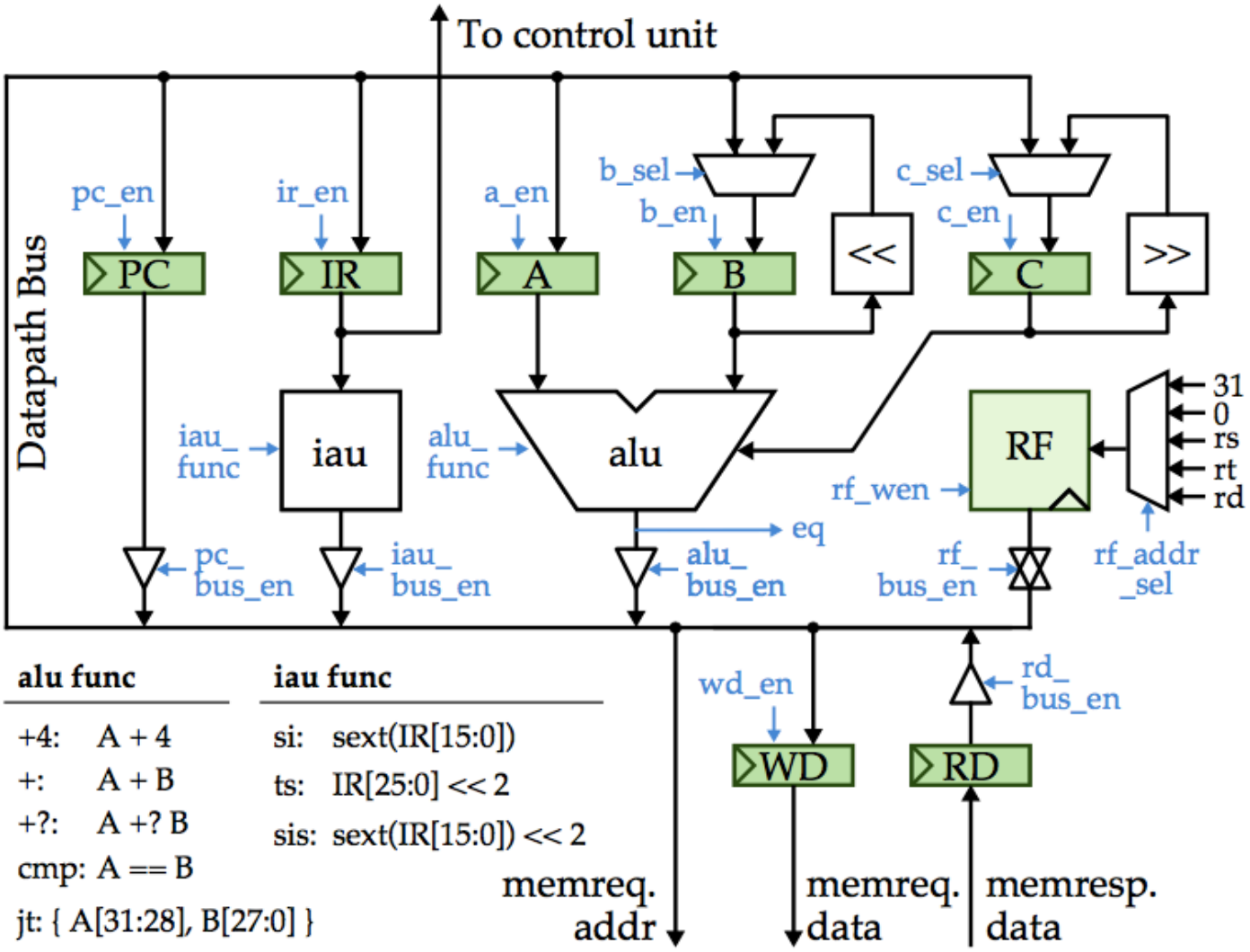
Acknowledgement

Cornell University, ECE 4750

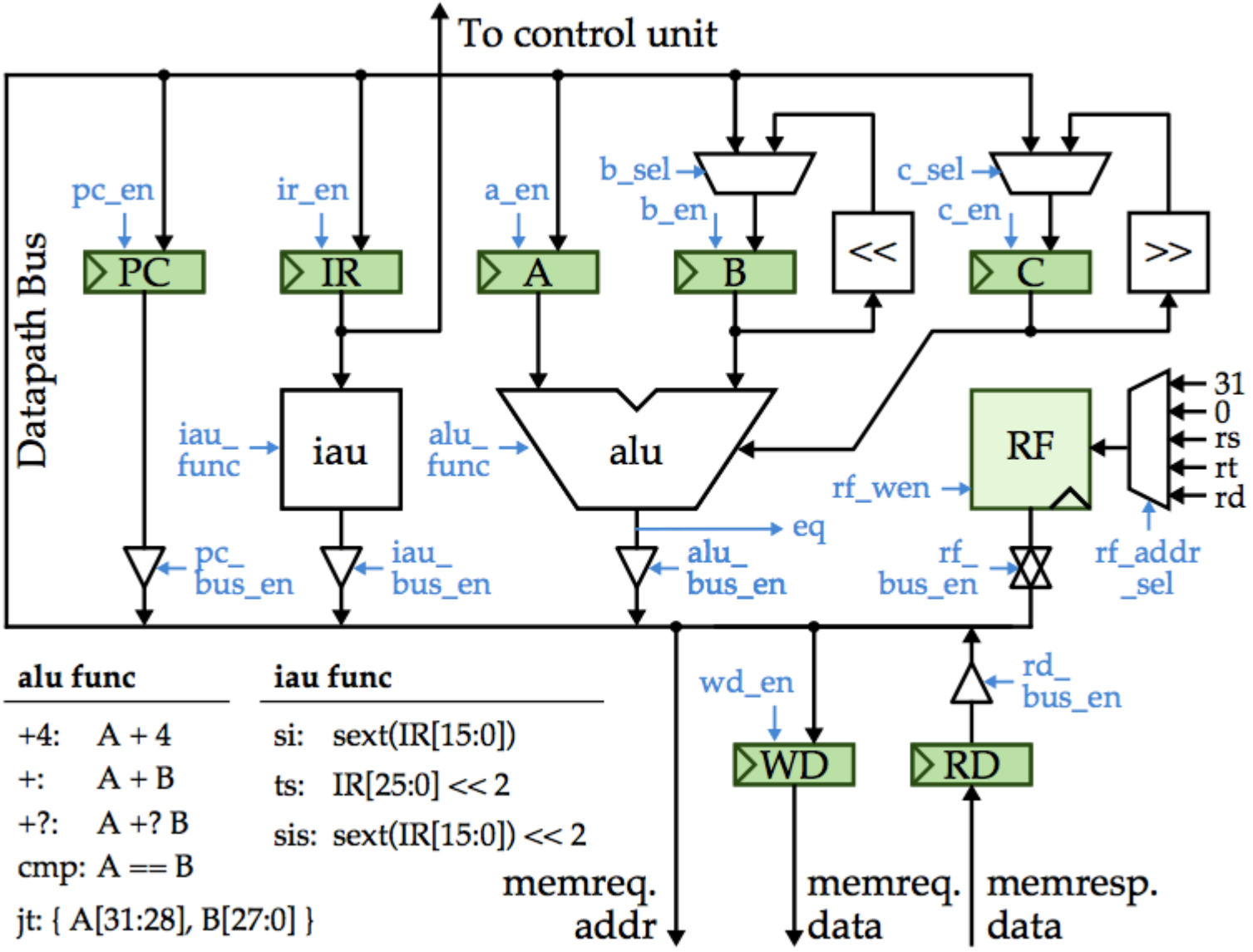
Quiz: Adding a New Auto-Incrementing Load Instruction

`lw.ai rt, offset(rs)`

$$R[\texttt{rt}] \leftarrow M[\, R[\texttt{rs}] + \text{sext}(\texttt{offset})\,];\ R[\texttt{rs}] \leftarrow R[\texttt{rs}] + 4$$



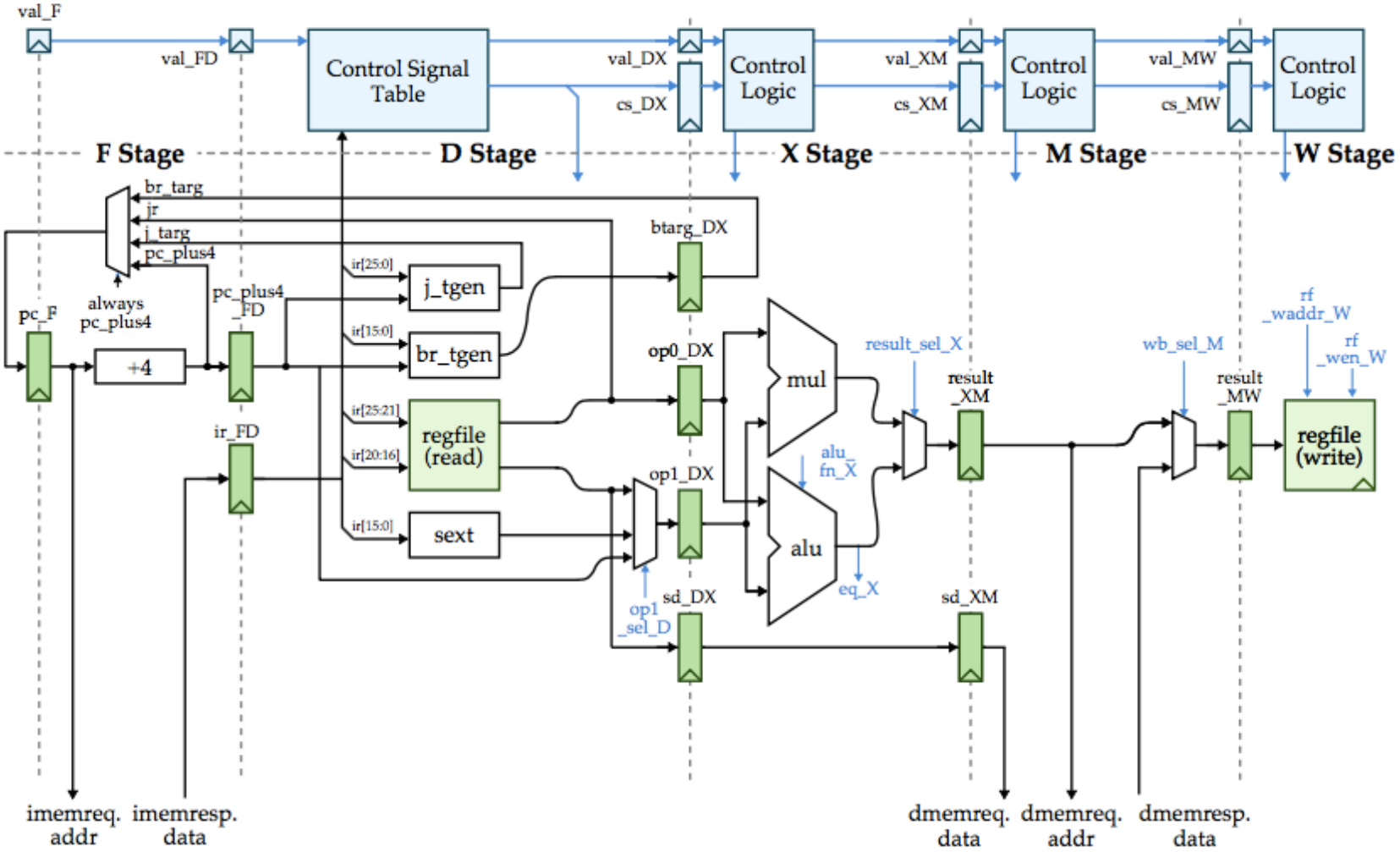
Quiz: Estimating Cycle Time



Quiz: Adding a New Auto-Incrementing Load Instruction

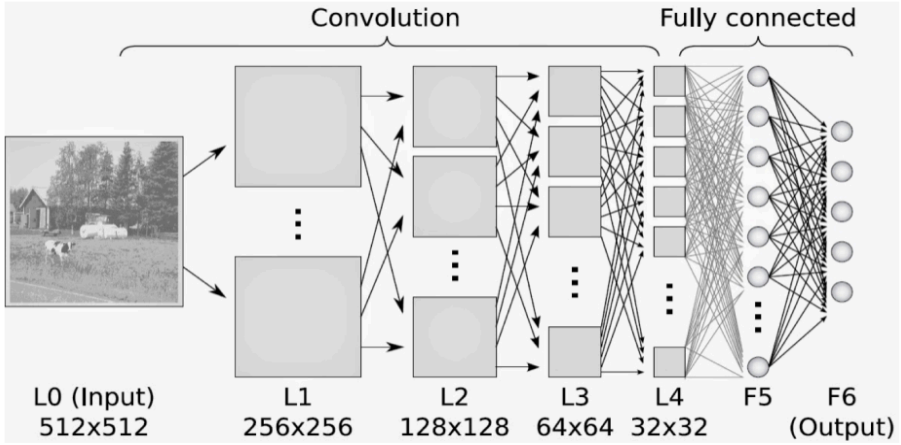
```
lw.ai rt, imm(rs)
```

$$R[rt] \leftarrow M[\, R[rs] + sext(imm)\,];\; R[rs] \leftarrow R[rs] + 4$$



Class Project Introduction

- Convolutional neural network (CNN)
  - an advanced artificial neural network algorithm
  - highly successful in image recognition applications



- Design a CNN hardware accelerator
  - latency and throughput
  - power and area

Project Timeline

- 3/22: Brief introduction; start forming teams
- 3/27: Release description; team finalized
- Week 11-12: Review related research papers
- 4/10: Submit initial project proposal/plan with block diagrams and interfaces
- Week 13-14: No lecture. Project team meetings
- 4/24 and 4/26: Project presentation
- 5/8: Final project report