Computer Organization

Lecture 21 - Memory Hierarchy 1

Reading: 5.1-5.3

Roadmap for the Term: Major Topics

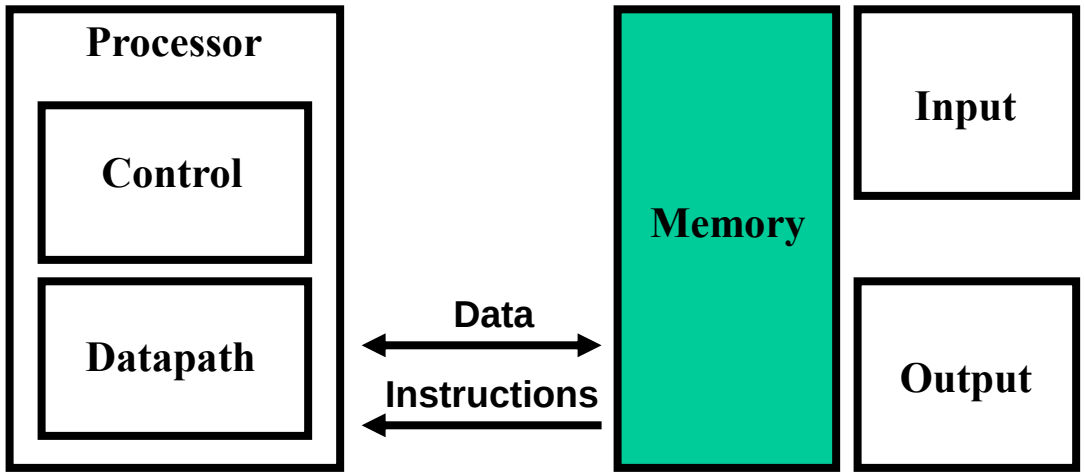
- Computer Systems Overview
- Performance
- Technology Trends
- Instruction Sets (and Software)
- Logic and Arithmetic
- Processor Implementation
- Memory Systems
- Input/Output
- Multiprocessor/Multicore

Outline - Memory Systems

- Overview
  - Motivation
  - General Structure and Terminology
- Memory Technology
  - Static RAM
  - Dynamic RAM
  - Disks
- Cache Memory
- Virtual Memory

Memory Systems - the Big Picture

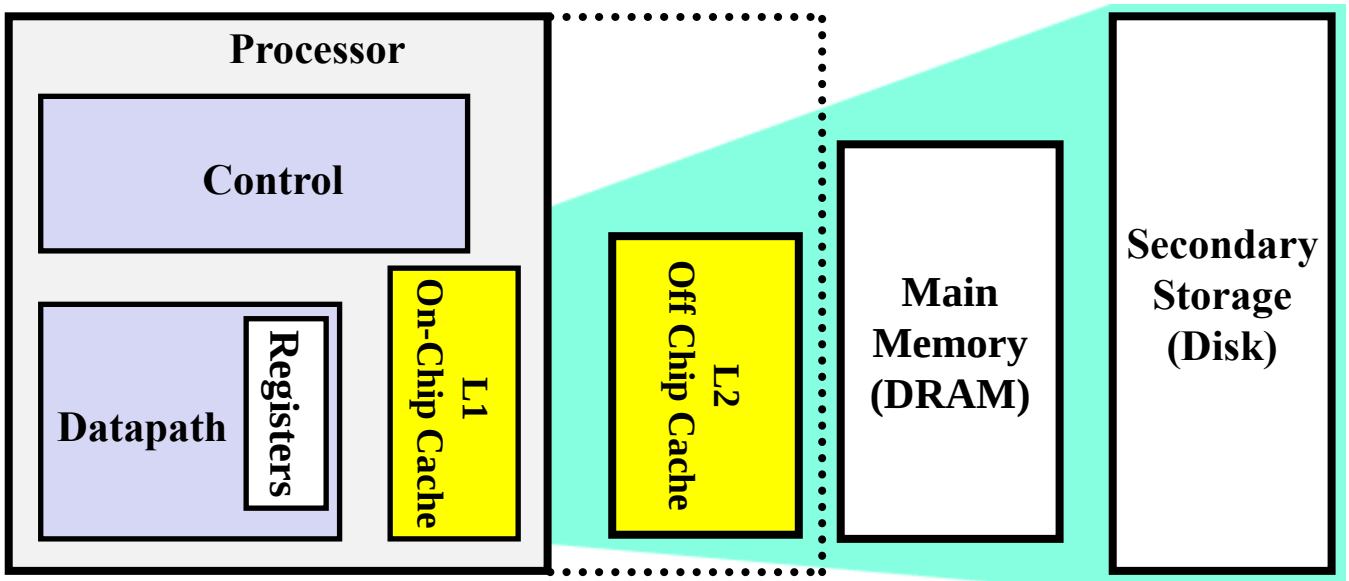
- Memory provides processor with
  - Instructions
  - Data
- Problem: memory is too slow and too small



"Five Classics Components" Picture

Memory Hierarchy - the Big Picture

- Problem: memory is too slow and too small
- Solution: memory hierarchy

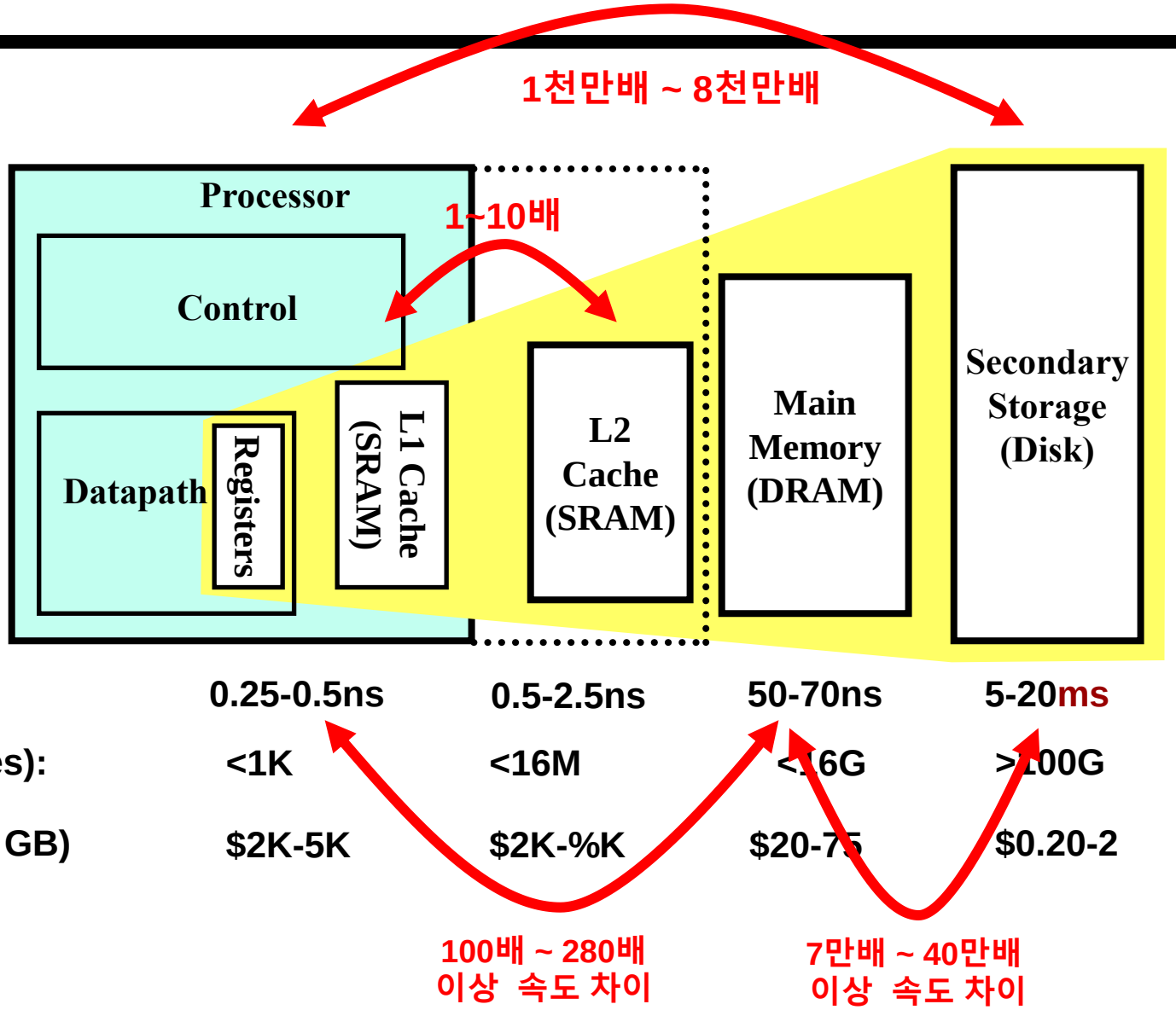


Why Hierarchy Works

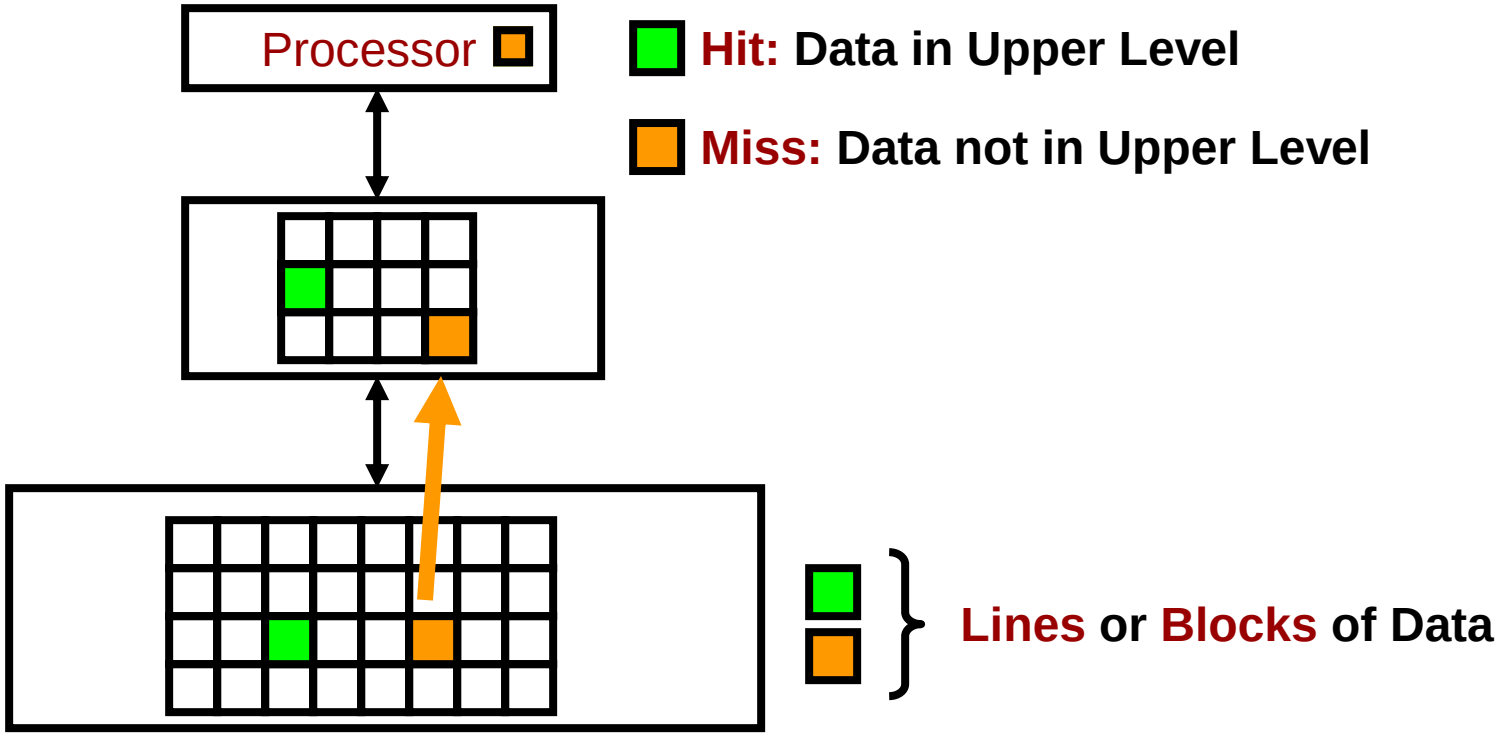
- The principle of locality
  - Programs access a relatively small portion of the address space at any instant of time.

  - Temporal locality: recently accessed data is likely to be used again
  - Spatial locality: data near recently accessed data is likely to be used soon
- Result: the illusion (착각) of large, fast memory

Memory Hierarchy - Speed vs. Size



Memory Hierarchy - Terminology



Memory Hierarchy Terminology (cont'd)

- **Hit**: data appears in some block in the upper level (green block)
  - **Hit Rate**: the fraction of memory accesses that "hit"
  - **Hit Time**: time to access the upper level (time to determine hit/miss + access time)
- **Miss**: data must be retrieved from block in lower level (orange block)
  - **Miss Rate** = 1 - (Hit Rate)
  - **Miss Penalty**: Time to replace block in upper level + Time to deliver data to the processor
- Hit Time << Miss Penalty and Hit Rate >> Miss Rate

Typical Memory Hierarchy - Details

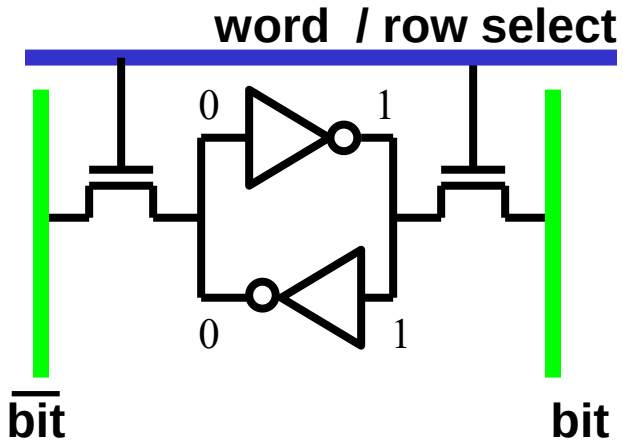
- **Registers** - Small, fastest on-chip storage
  - Managed by compiler and run-time system
- **Cache** - Small, fast on-chip storage
  - Associative lookup - managed by hardware
- **Memory** - Slower, Larger off-chip storage
  - Limited size <4GB – managed by hardware, OS (32bit)
- **Disk** - Slowest, Largest off-chip storage
  - **Virtual memory** - simulate a large memory using disk, hardware, and operating system
  - File storage - store data files using operating system

Outline - Memory Systems

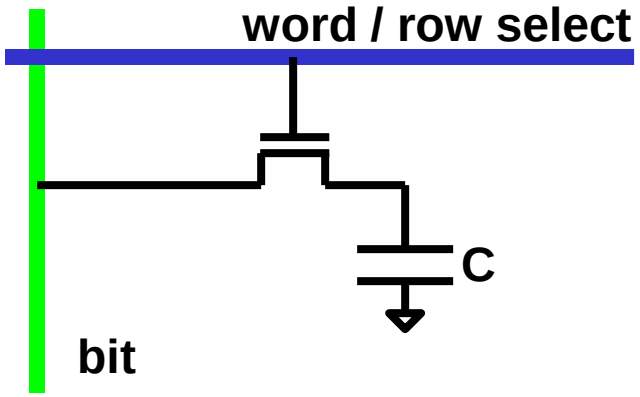
- Overview
  - Motivation
  - General Structure and Terminology
- Memory Technology
  - Static RAM
  - Dynamic RAM
- Cache Memory
- Virtual Memory

Memory Types

- Static RAM
  - Storage using latch circuits
  - Values saved while power on



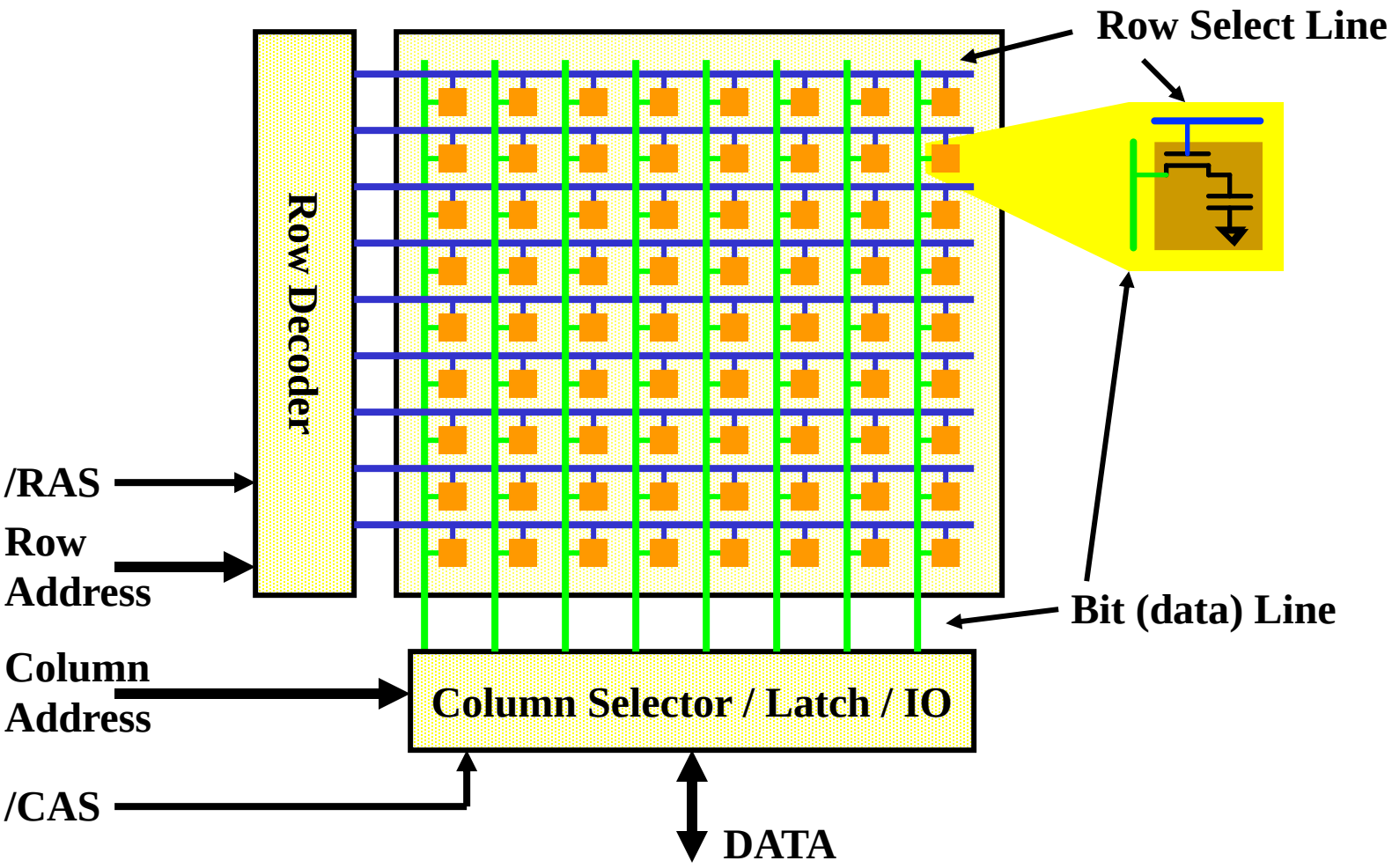
- Dynamic RAM
  - Storage using capacitors
  - Values must be refreshed



Tradeoffs - Static vs. Dynamic RAM

- **Static RAM (SRAM) - used for Regs, L1, L2 cache**
  - **Fast - 0.5-25ns access time (less for on-chip)**
  - **Larger, More Expensive**
  - **Higher power consumption**
- **Dynamic RAM (DRAM) - used for PC main memory**
  - **Slower - 50-70ns access time**
  - **Refresh required**
  - **Smaller, Cheaper**
  - **Lower power consumption**

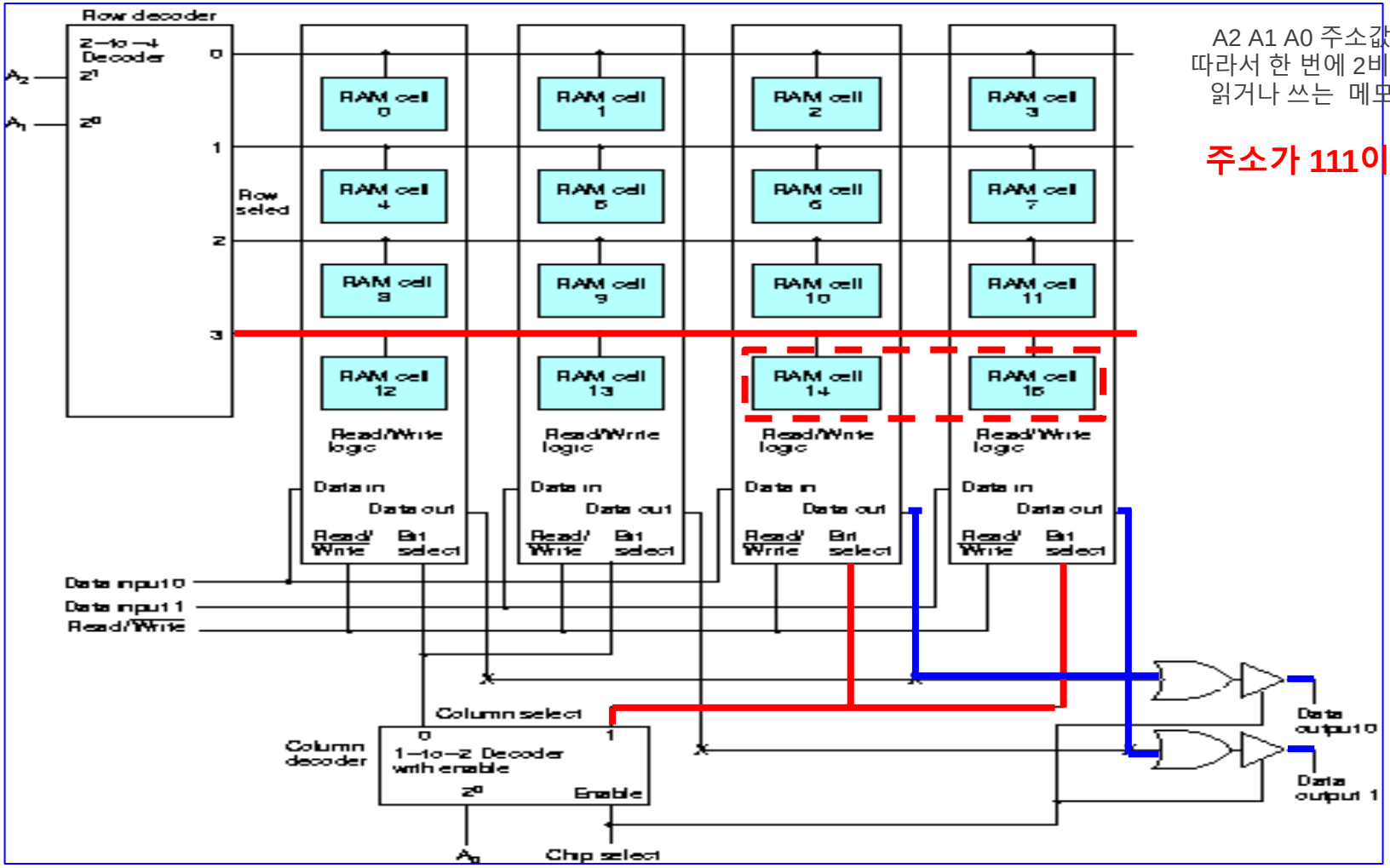
DRAM Organization



RAM Organization (8 x 2 RAM)

1

1

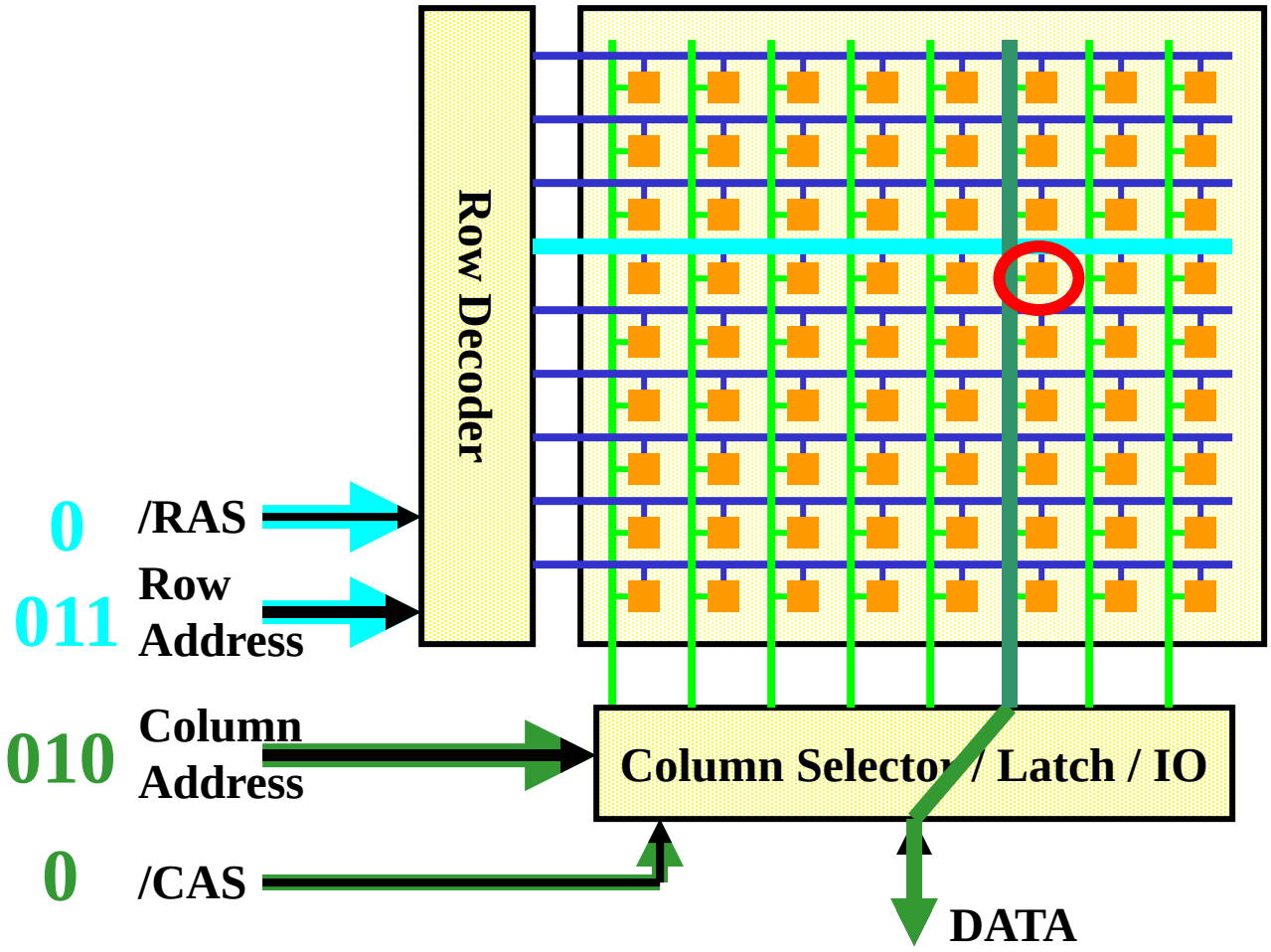


A2 A1 A0 주소값에 따라서 한 번에 2비트를 읽거나 쓰는 메모리

주소가 111이면?

1

DRAM Read Operation (64 x 1 DRAM)



DRAM Trends

- **Size: 4X every 3 years (about 60% increase per year)**
- **Latency: 2X every 10 years**

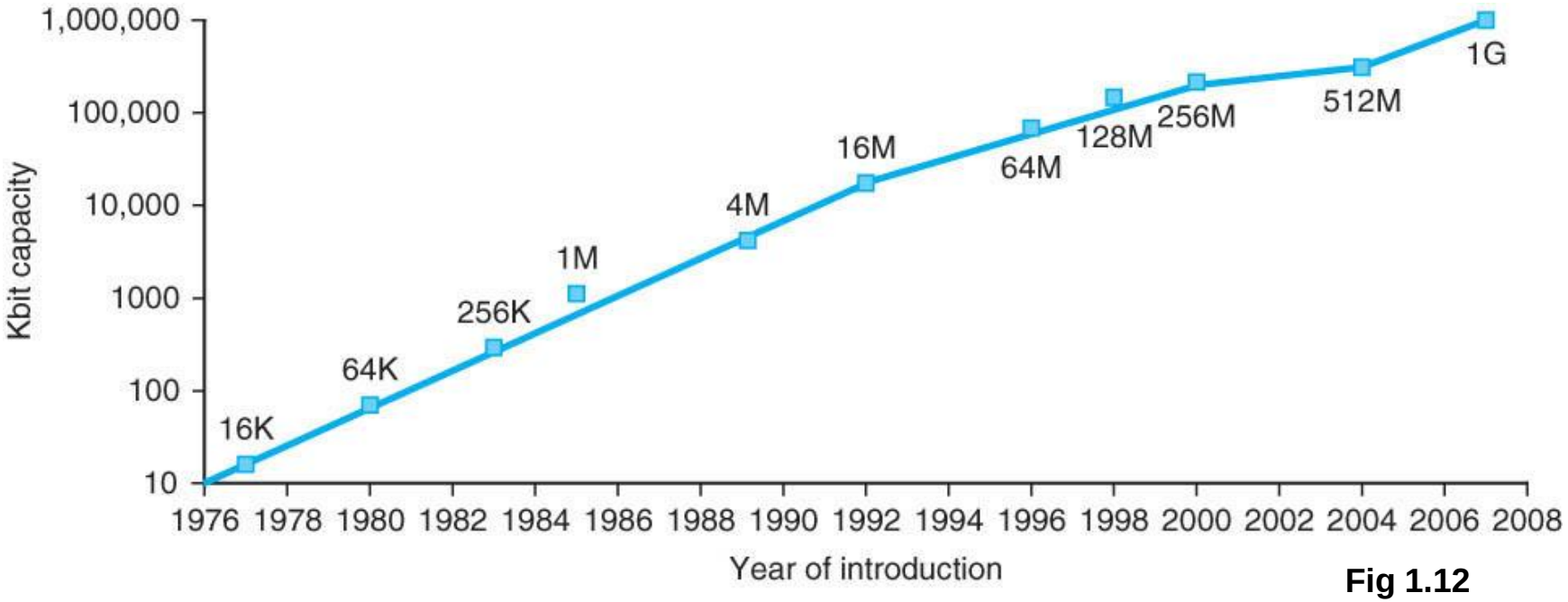
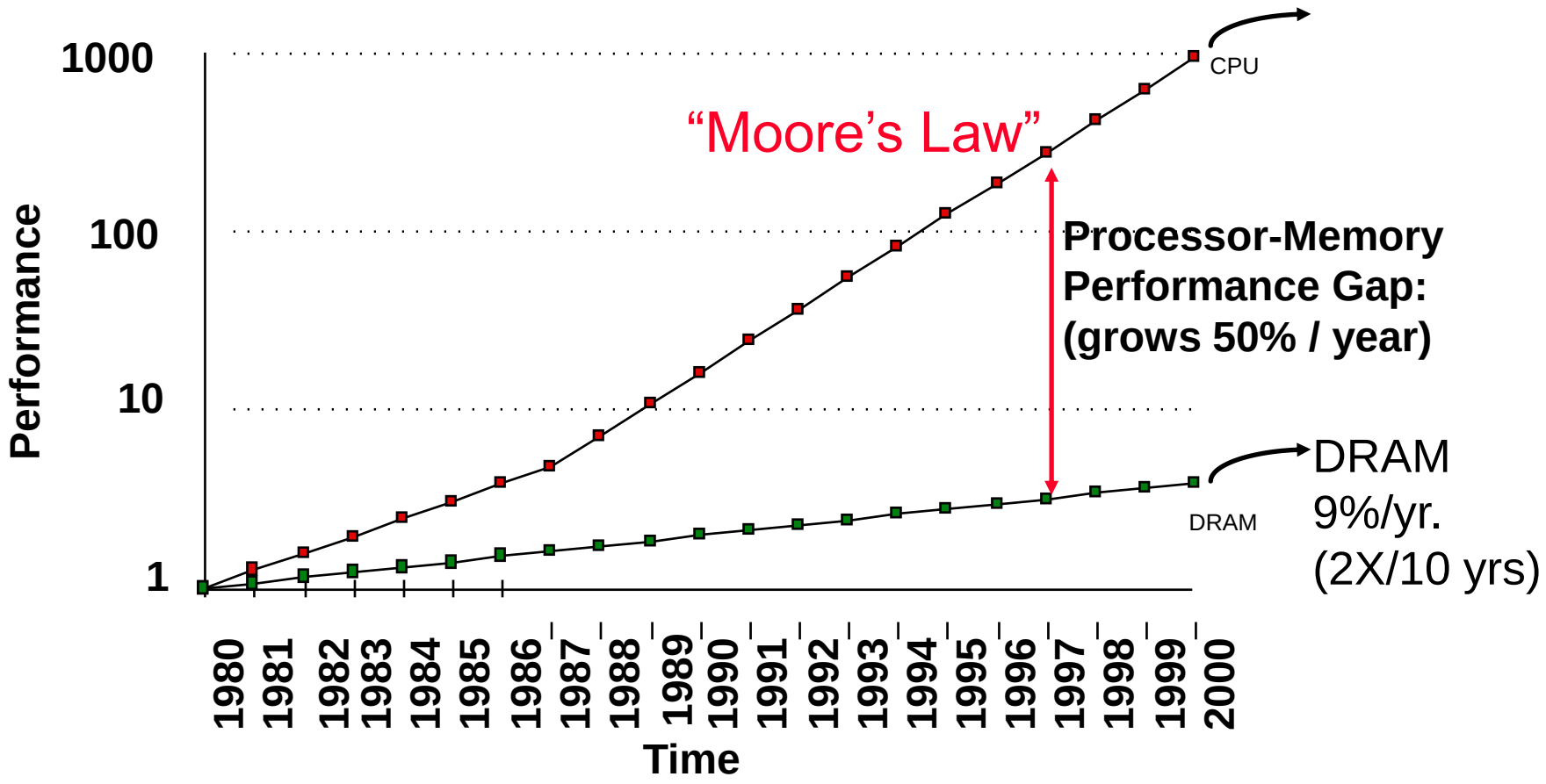


Fig 1.12

The Processor/Memory Speed Gap



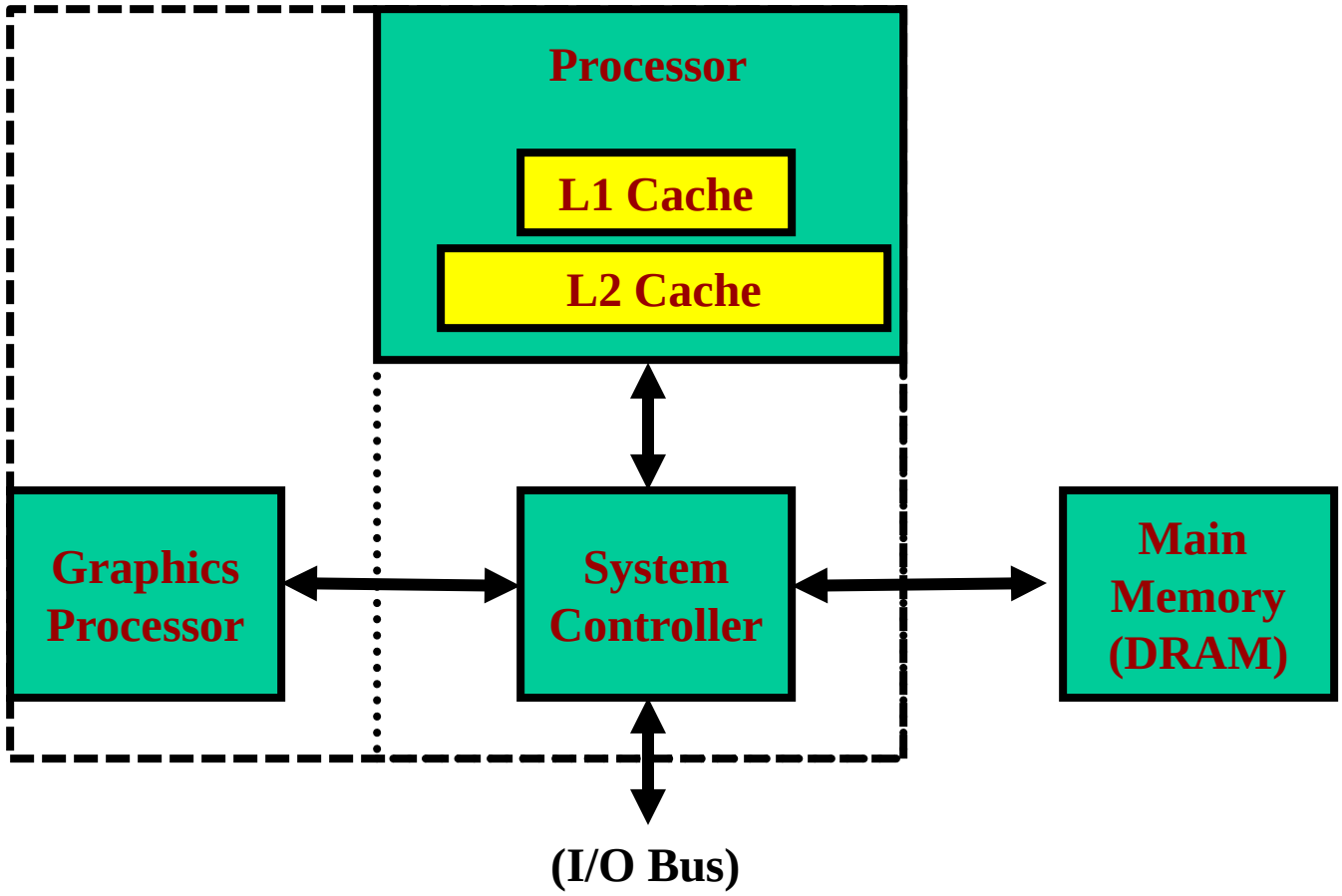
Addressing the Speed Gap

- Latency depends on physical limitations
- Bandwidth can be increased using:
  - Parallelism - transfer more bits / word
  - Burst transfers - transfer successive words on each cycle
- So... use bandwidth to support memory hierarchy!
  - Use cache to support locality of reference
  - Design hierarchy to transfer large blocks of memory

Current DRAM Parts

- **Synchronous DRAM (SDRAM)** - clocked transfer of bursts of data starting at a specific address
  - Double-Data Rate SDRAM - transfer two bits/clock cycle
  - Quad-Data Rate SDRAM - transfer four bits / clock cycle
  - Rambus RDRAM - High-speed interface for fast transfers
- **Current PCs** use some form of SDRAM
  - DDR3 – 1,066MHz (= 1GHz) memory bus

Memory Configuration in Current PCs



Outline - Memory Systems

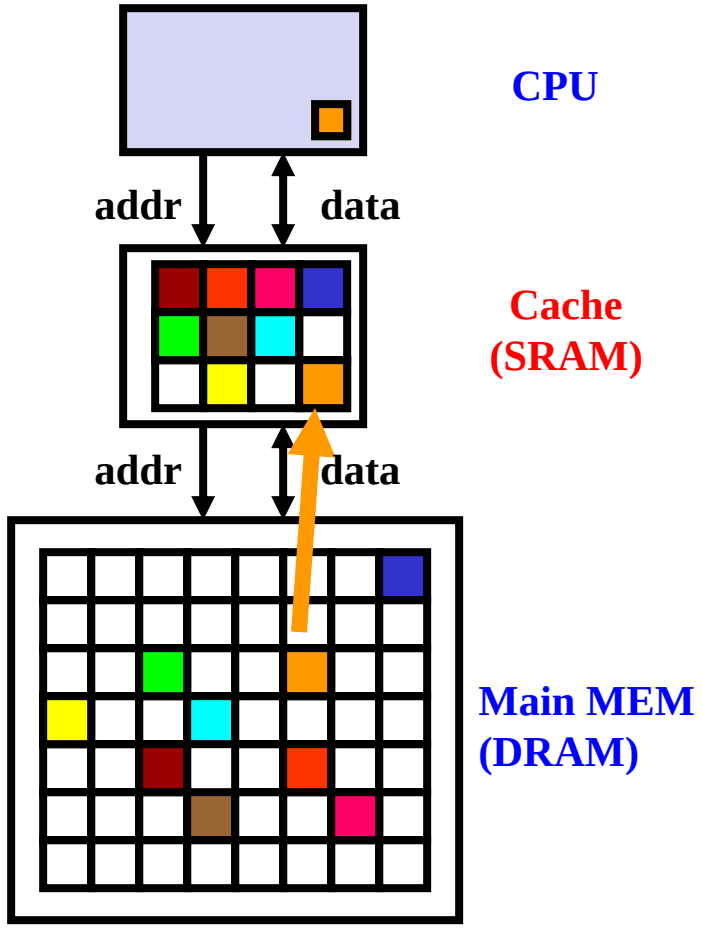
- Overview
  - Motivation
  - General Structure and Terminology
- Memory Technology
  - Static RAM
  - Dynamic RAM
- Cache Memory
- Virtual Memory

Cache Operation

- Inserted between CPU and Main Mem.
- Implement with fast Static RAM
- Holds some of a program's
  - data
  - instructions
- **Operation:**

  Hit: Data in Cache (no penalty)

  Miss: Data not in Cache (miss penalty)



Four Key Cache Questions:

1. **Where can block be placed in cache?**
   (block placement)

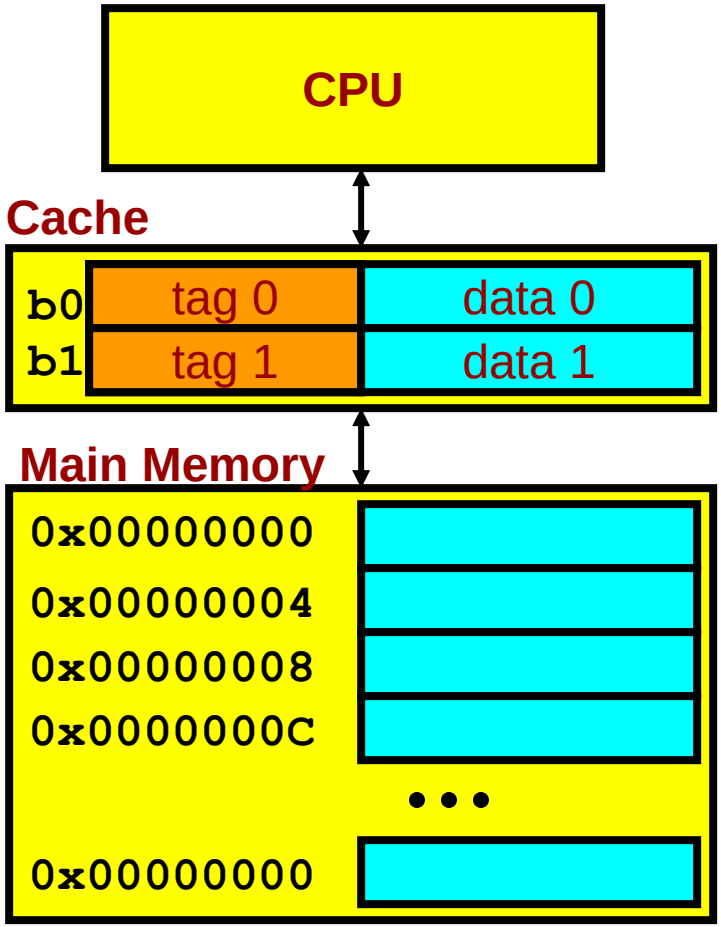
2. **How can block be found in cache?**
   (block identification)

3. **Which block should be replaced on a miss?**
   (block replacement)

4. **What happens on a write?**
   (write strategy)

Basic Cache Design

- Organized into blocks or lines
- Block Contents
  - tag - extra bits to identify block (part of block address)
  - data - data or instruction words - contiguous memory locations
- Our example:
  - One-word (4 byte) block size
  - 30-bit tag
  - Two blocks in cache

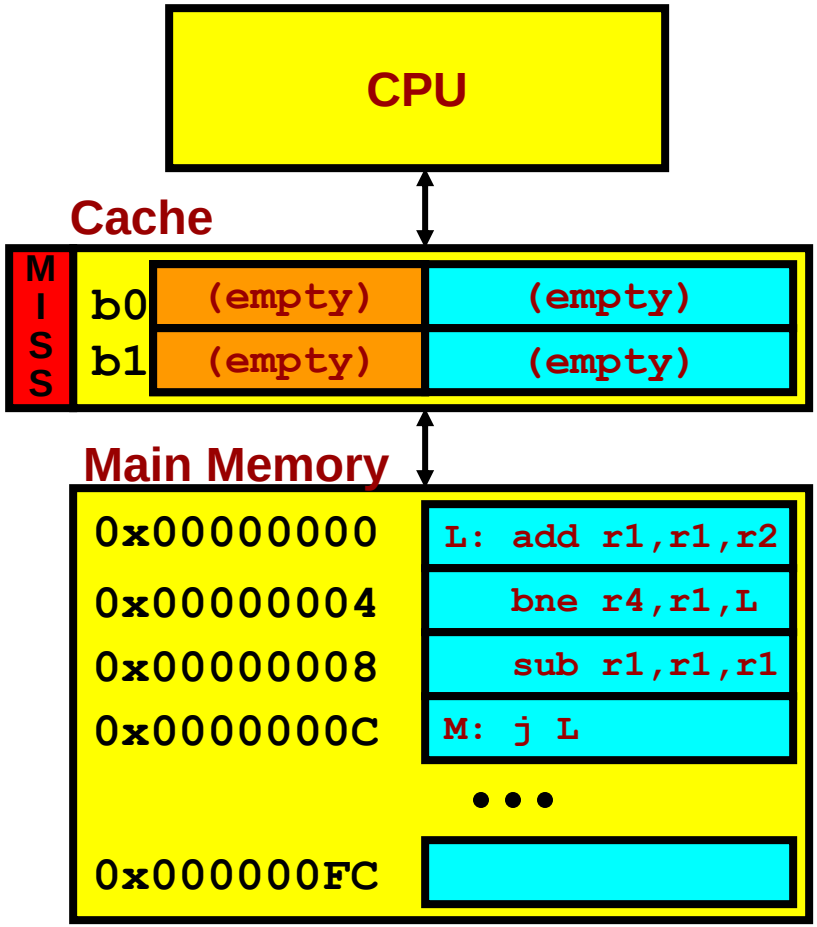


Cache Example (2)

- Assume:
  - `r1==0, r2==1, r4==2`
  - 1 cycle for cache access
  - 5 cycles for main. mem. access
  - 1 cycle for instr. execution

Assume that the fetch of next instruction can start during the execution of current instruction

- At cycle 1 - `PC=0x00`
  - Fetch instruction from memory
    - look in cache
    - MISS - fetch from main mem (5 cycle penalty)

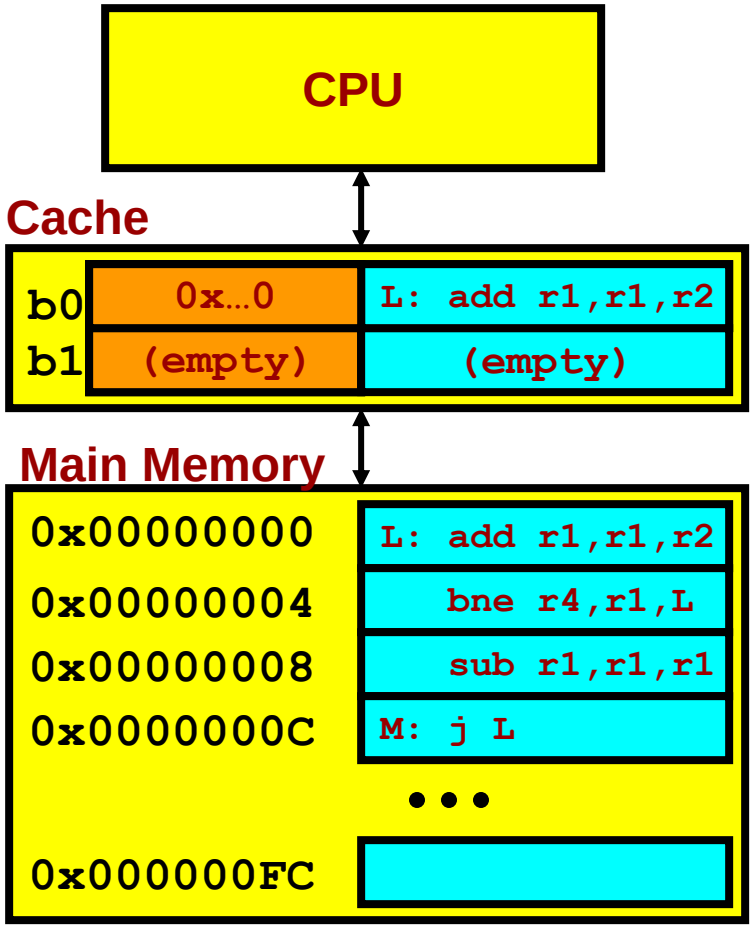


Cache Example (3)

`r1==1, r2==1, r4==2`

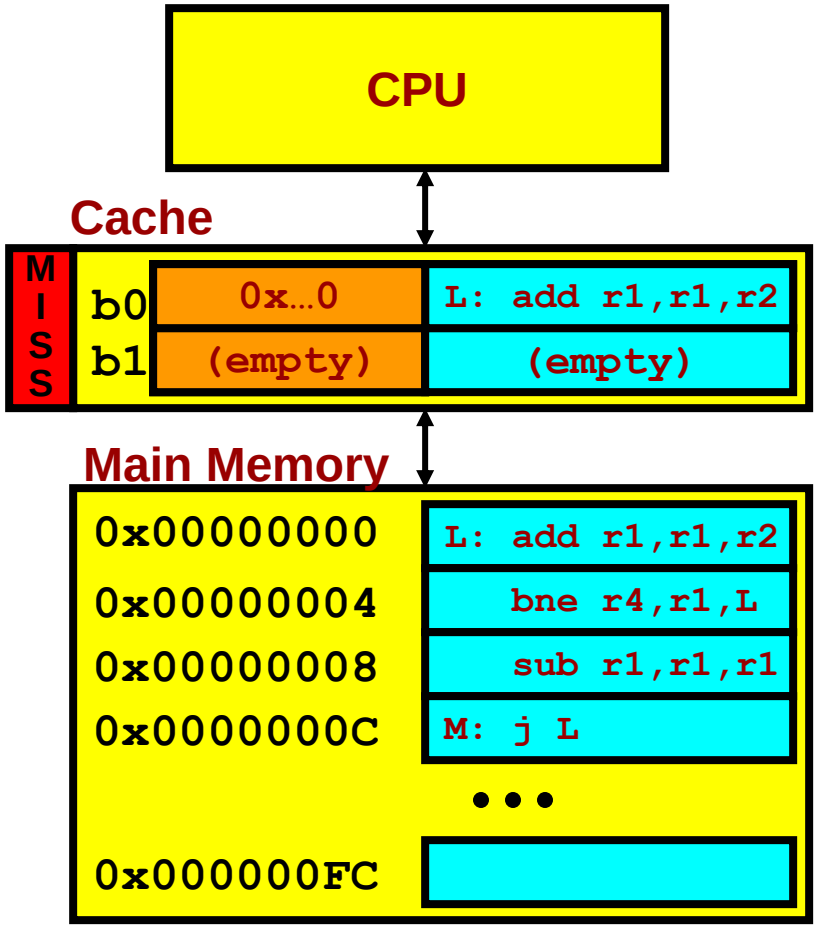
Cycle	Address	Op/Instr.	r1
1-5		`FETCH 0x…0`	
6	0x…0	`add r1,r1,r2`	1

- **At cycle 6**
  - Execute instr. `add r1,r1,r2`



Cache Example (4)

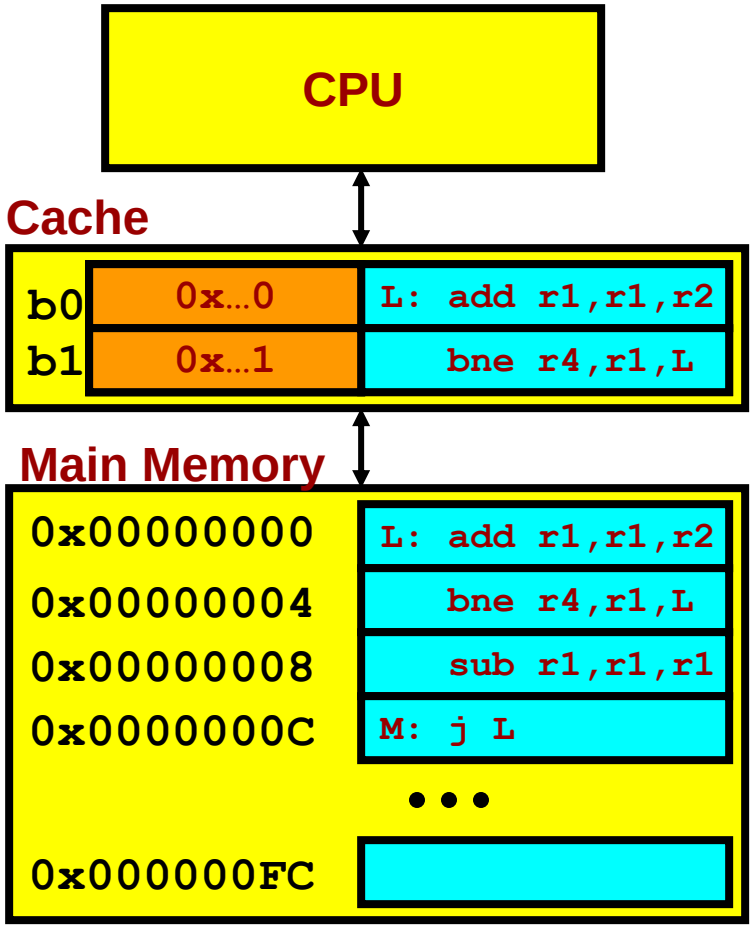
Cycle	Address	Op/Instr.	r1
1-5		`FETCH 0x…0`	
6	0x…0	`add r1,r1,r2`	1
6-10		`FETCH 0x…4`	



- **At cycle 6 - PC=0x04**
  - **Fetch instruction from memory**
    - **look in cache**
    - **MISS - fetch from main mem (5 cycle penalty)**

Cache Example (5)

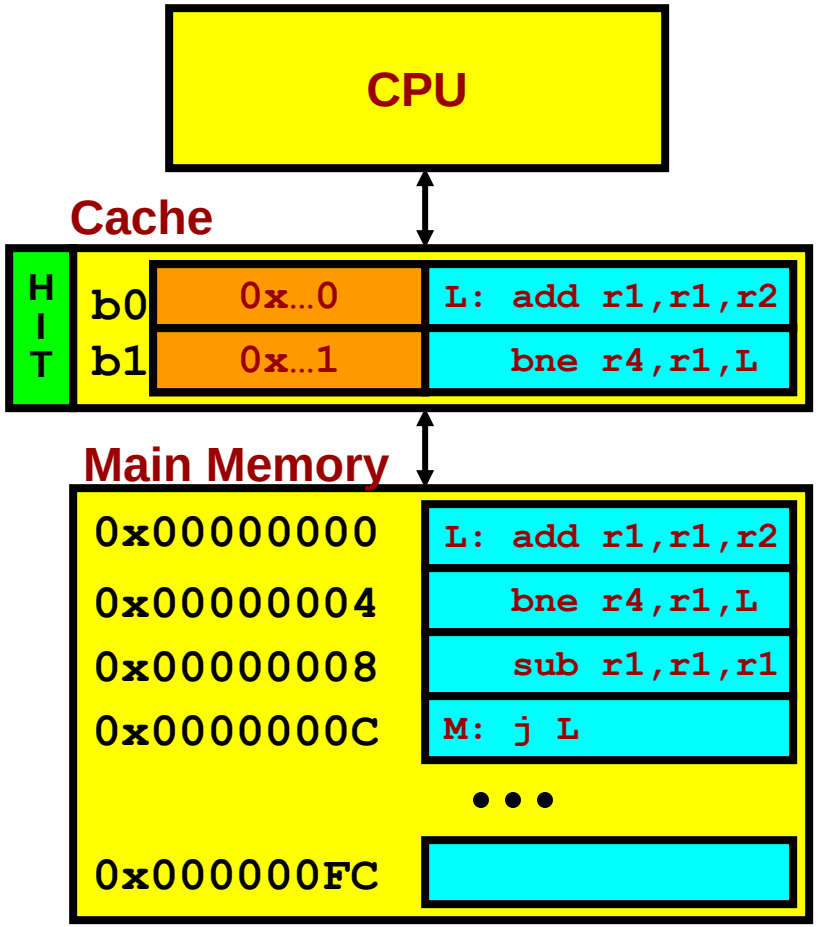
Cycle	Address	Op/Instr.	r1
1-5		FETCH 0x…0	
6	0x…0	add r1,r1,r2	1
6-10		FETCH 0x…4	
11	0x…4	bne r4,r1,L	1



- **At cycle 11**
  - Execute instr. `bne r4,r1,L`

Cache Example (6)

Cycle	Address	Op/Instr.	r1
1-5		FETCH 0x…0	
6	0x…0	add r1,r1,r2	1
6-10		FETCH 0x…4	
11	0x…4	bne r4,r1,L	1
11		FETCH 0x…0	1



- **At cycle 11 - PC=0x00**
  - **Fetch instruction from memory**
  - **HIT - instruction in cache**

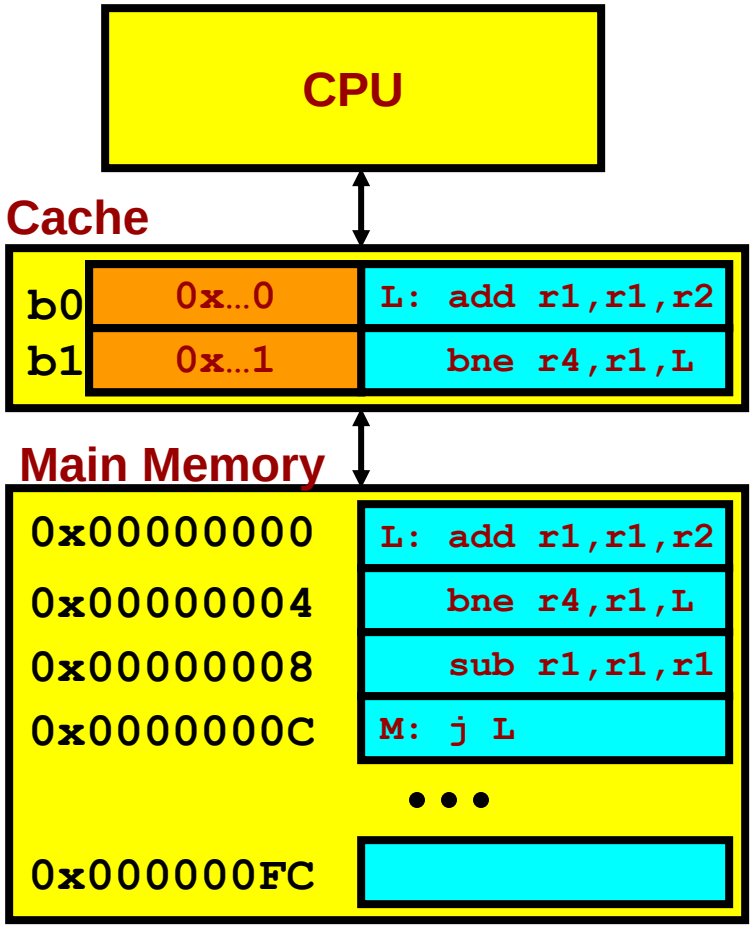
Cache Example (7)

r1==1, r2==1, r4==2

r1==2, r2==1, r4==2

Cycle	Address	Op/Instr.	r1
1-5		FETCH 0x…0	
6	0x…0	add r1,r1,r2	1
6-10		FETCH 0x…4	
11	0x…4	bne r4,r1,L	1
11		FETCH 0x…0	1
12		add r1,r1,r2	2

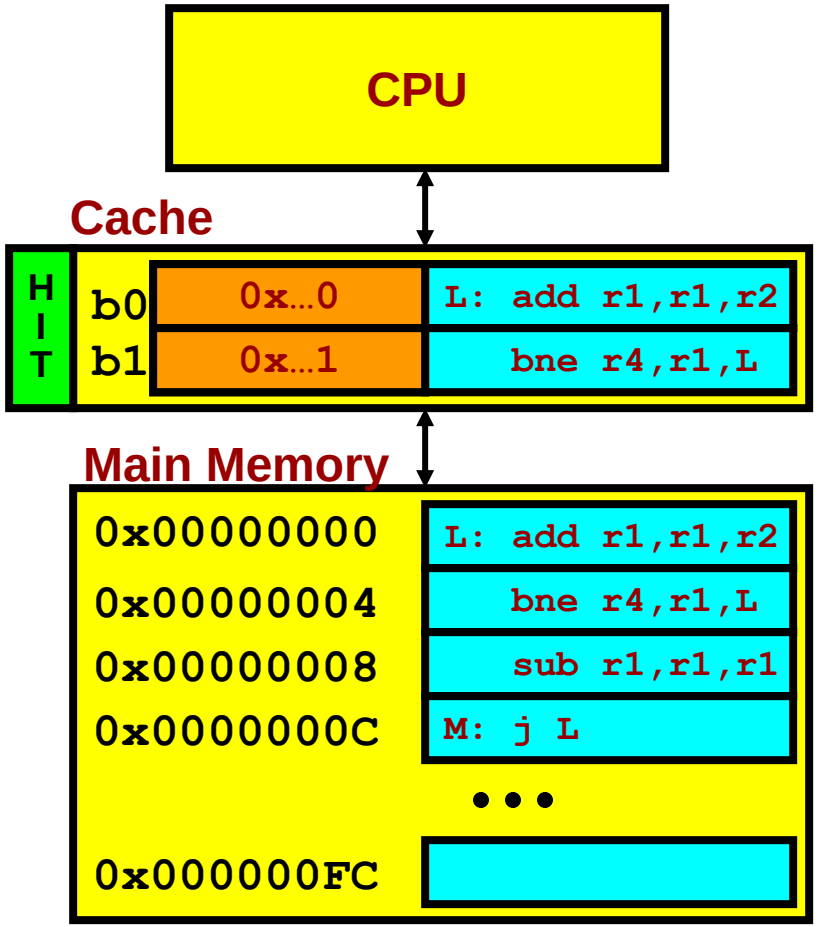
- **At cycle 12**
  - **Execute `add r1, r1, r2`**



Cache Example (8)

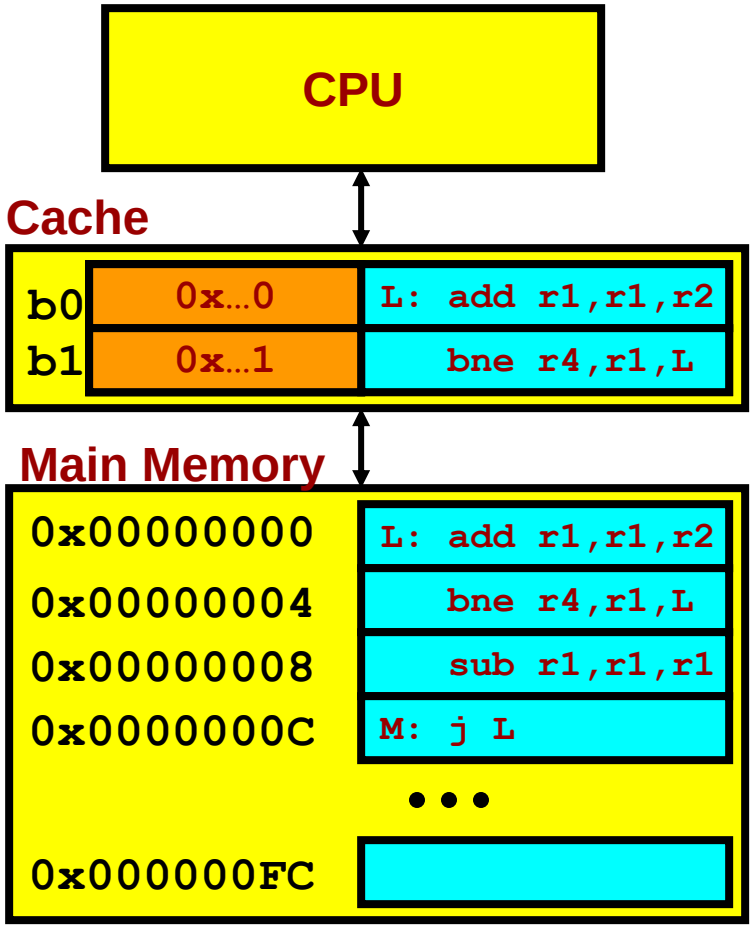
Cycle	Address	Op/Instr.	r1
1-5		FETCH 0x…0	
6	0x…0	add r1,r1,r2	1
6-10		FETCH 0x…4	
11	0x…4	bne r4,r1,L	1
11		FETCH 0x…0	1
12		add r1,r1,r2	2
12		FETCH 0x04	

- **At cycle 12 - PC=0x04**
  - **Fetch instruction from memory**
  - **HIT - instruction in cache**



Cache Example (9)

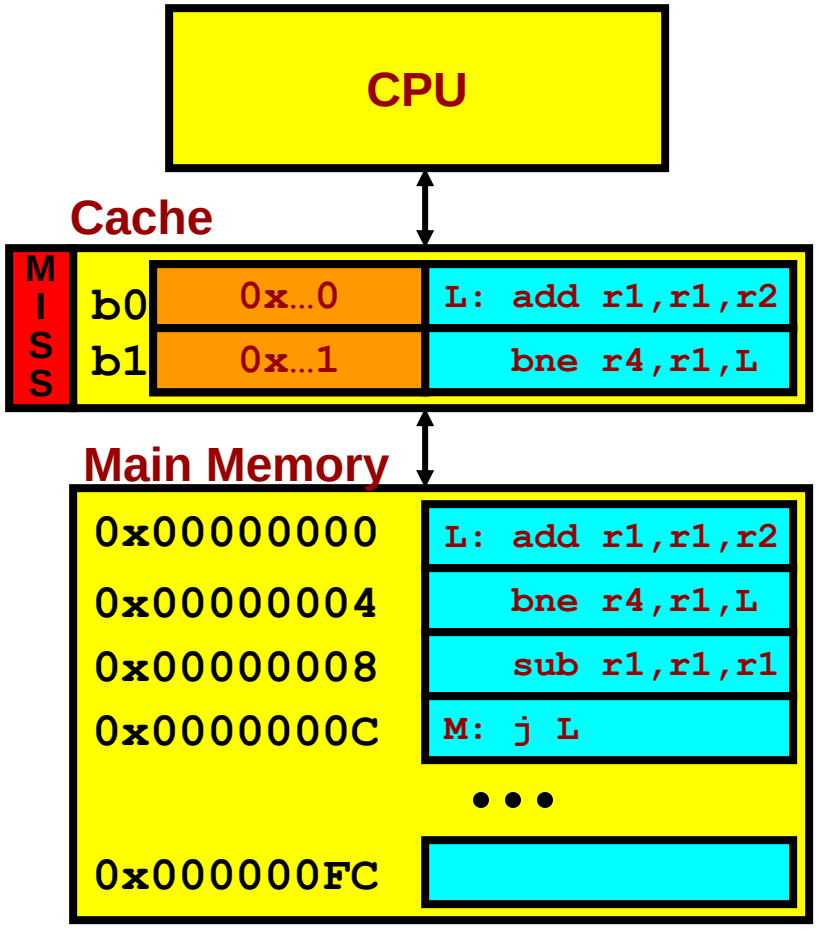
Cycle	Address	Op/Instr.	r1
1-5		FETCH 0x…0	
6	0x…0	add r1,r1,r2	1
6-10		FETCH 0x…4	
11	0x…4	bne r4,r1,L	1
11		FETCH 0x…0	1
12		add r1,r1,r2	2
12		FETCH 0x04	
13		bne r4, r1, L	



- **At cycle 13**
  - Execute instr. `bne r4, r1, L`
  - Branch not taken

Cache Example (10)

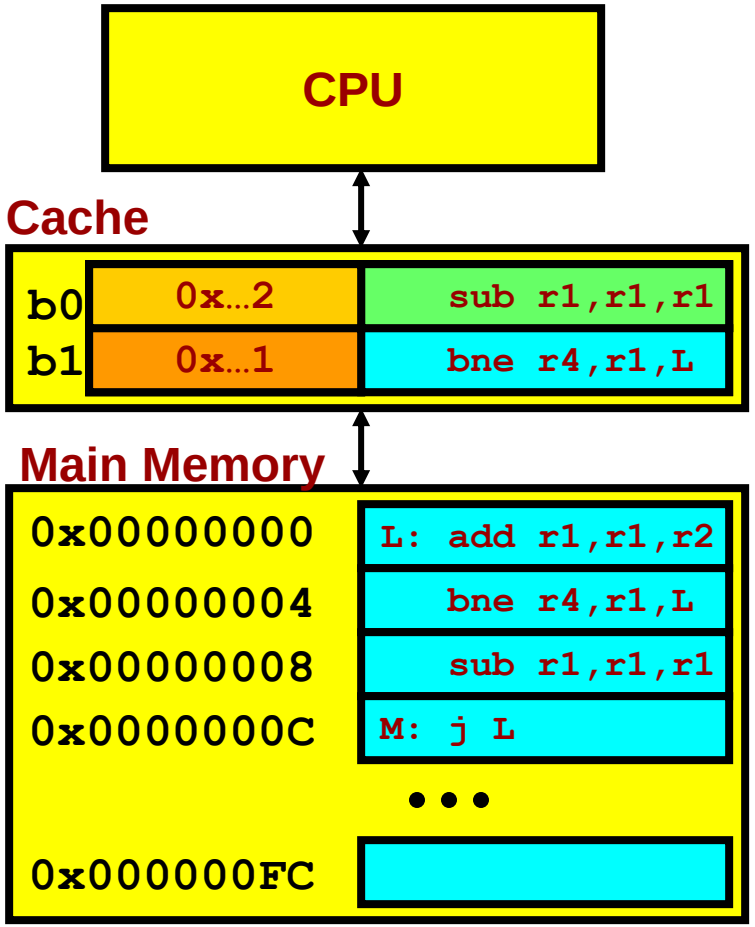
Cycle	Address	Op/Instr.	r1
1-5		FETCH 0x…0	
6	0x…0	add r1,r1,r2	1
6-10		FETCH 0x…4	
11	0x…4	bne r4,r1,L	1
11		FETCH 0x…0	1
12		add r1,r1,r2	2
12		FETCH 0x04	
13		bne r4, r1, L	
13		FETCH 0x08	



- **At cycle 13 - PC=0x08**
  - **Fetch Instruction from Memory**
  - **MISS - not in cache**

Cache Example (11)

Cycle	Address	Op/Instr.	r1
1-5		FETCH 0x…0	
6	0x…0	add r1,r1,r2	1
6-10		FETCH 0x…4	
11	0x…4	bne r4,r1,L	1
11		FETCH 0x…0	1
12		add r1,r1,r2	2
12		FETCH 0x04	
13		bne r4, r1, L	
13-17		FETCH 0x08	



- **At cycle 17 - PC=0x08**
  - Put instruction into cache
  - Replace existing instruction

Cache Example (12)

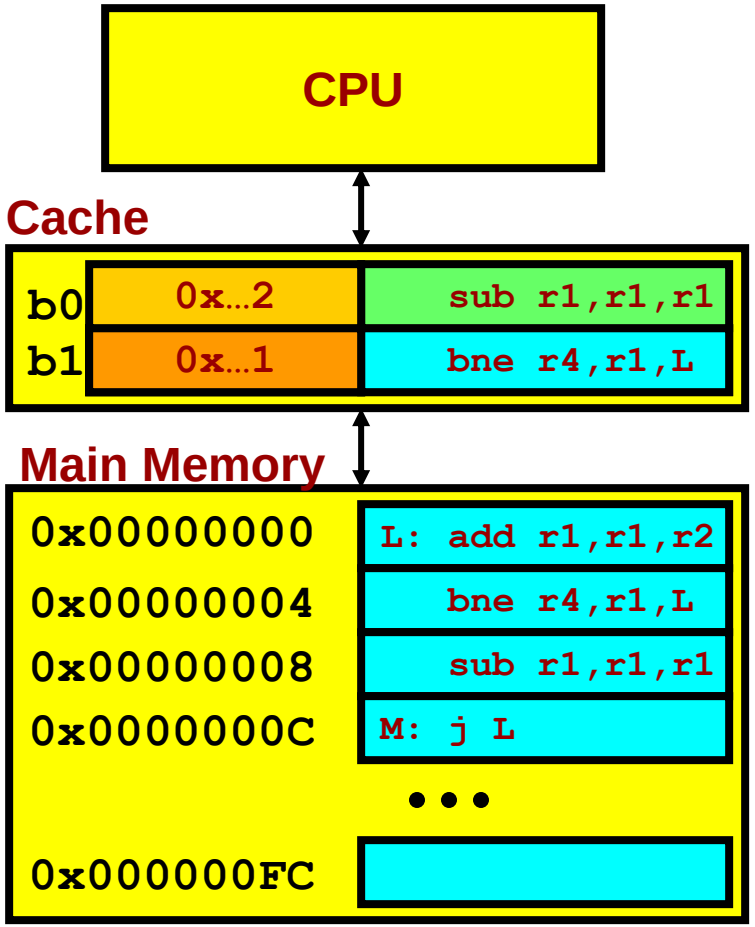
r1==2, r2==1, r4==2

↓

r1==0, r2==1, r4==2

Cycle	Address	Op/Instr.	r1
1-5		FETCH 0x…0	
6	0x…0	add r1,r1,r2	1
6-10		FETCH 0x…4	
11	0x…4	bne r4,r1,L	1
11		FETCH 0x…0	1
12		add r1,r1,2	2
12		FETCH 0x04	2
13		bne r4, r1, L	2
13-17		FETCH 0x08	2
18		sub r1, r1, r1	0

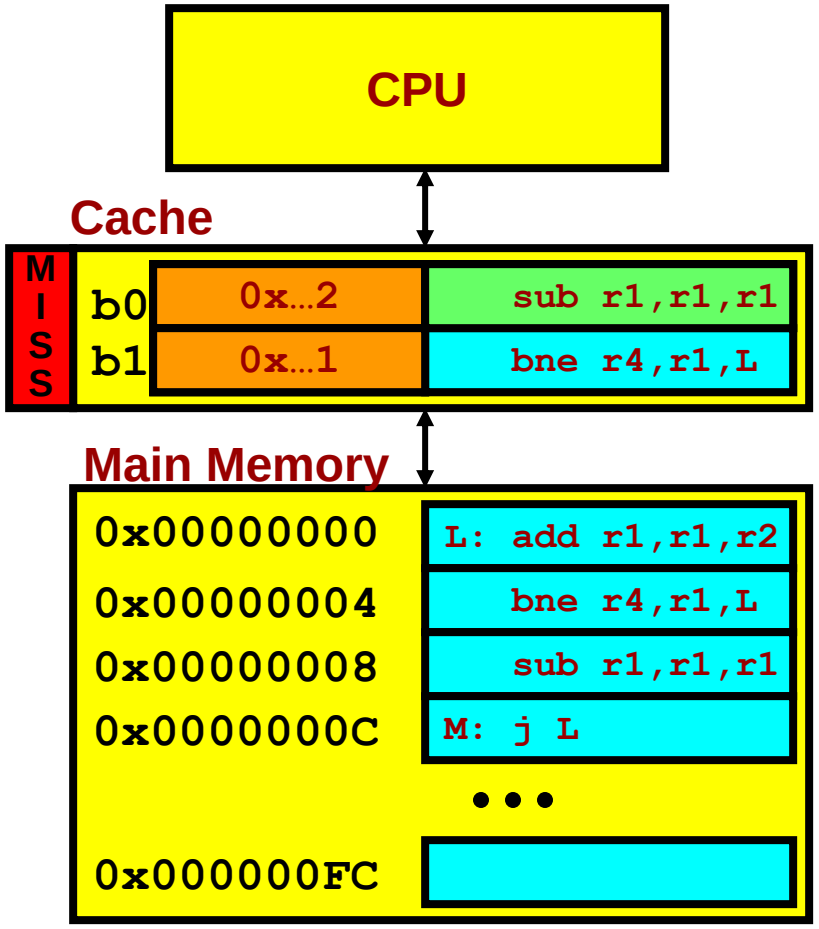
- **At cycle 18**
  - Execute `sub r1, r1, r1`



Cache Example (13)

Cycle	Address	Op/Instr.	r1
1-5		FETCH 0x…0	
6	0x…0	add r1,r1,r2	1
6-10		FETCH 0x…4	
11	0x…4	bne r4,r1,L	1
11		FETCH 0x…0	1
12		add r1,r1,r2	2
12		FETCH 0x04	2
13		bne r4, r1, L	2
13-17		FETCH 0x08	2
18		sub r1, r1, r1	0
18		FETCH 0x0C	

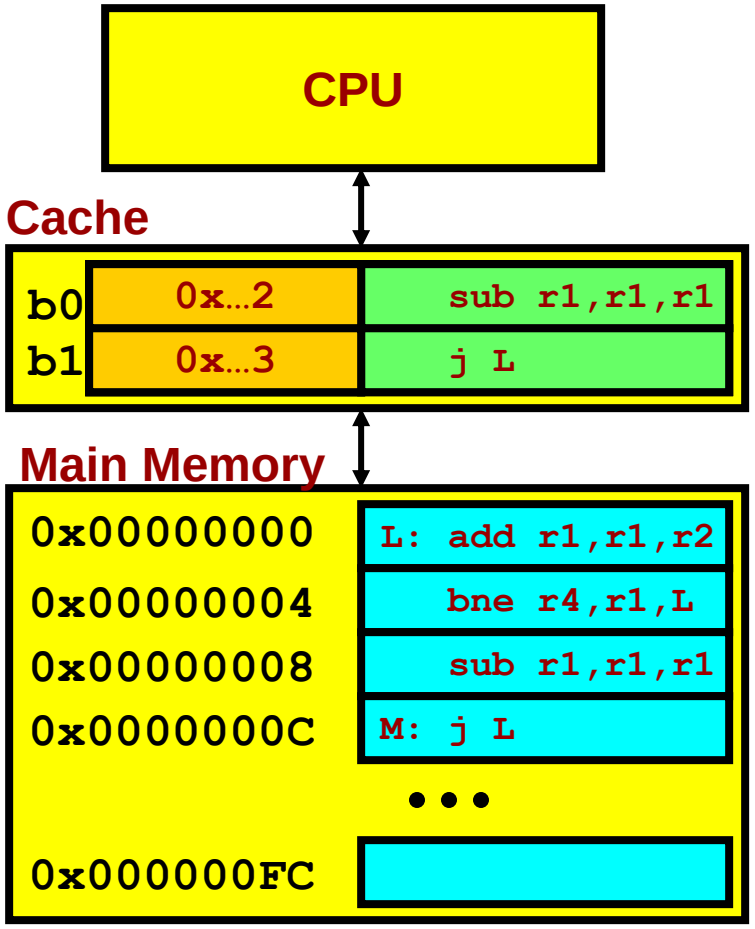
- **At cycle 18**
  - Fetch instruction from memory
  - MISS - not in cache



Cache Example (14)

Cycle	Address	Op/Instr.	r1
1-5		FETCH 0x…0	
6	0x…0	add r1,r1,r2	1
6-10		FETCH 0x…4	
11	0x…4	bne r4,r1,L	1
11		FETCH 0x…0	1
12		add r1,r1,r2	2
12		FETCH 0x04	2
13		bne r4, r1, L	2
13-17		FETCH 0x08	2
18		sub r1, r1, r1	0
18-22		FETCH 0x0C	

- **At cycle 22**
  - **Put instruction into cache**
  - **Replace existing instruction**

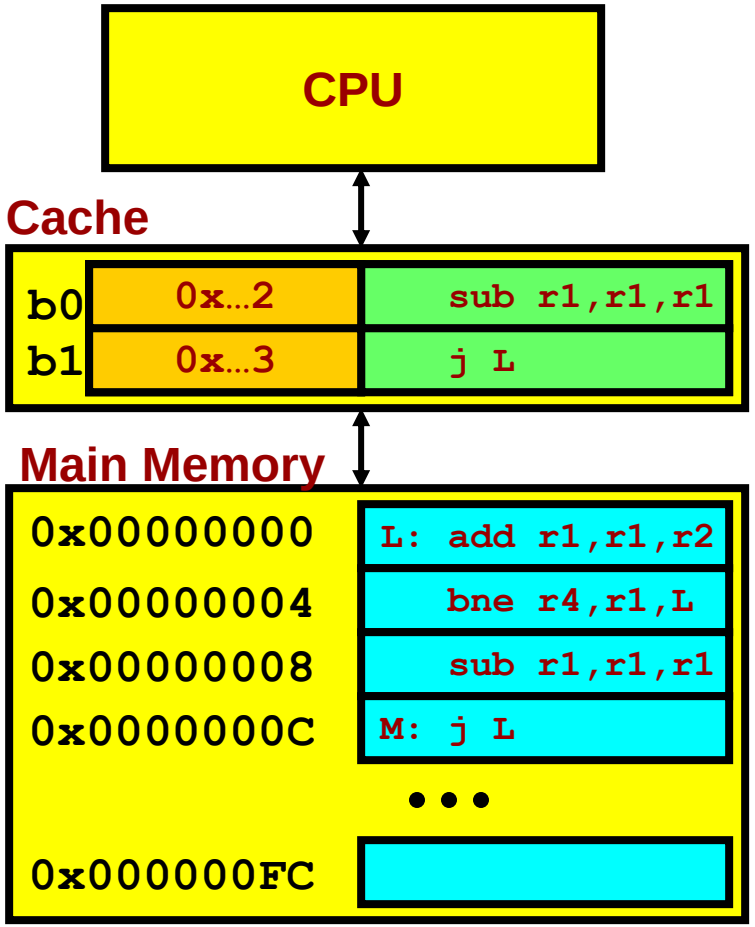


Cache Example (15)

Cycle	Address	Op/Instr.	r1
1-5		FETCH 0x…0	
6	0x…0	add r1,r1,r2	1
6-10		FETCH 0x…4	
11	0x…4	bne r4,r1,L	
11		FETCH 0x…0	
12	0x…8	add r1,r1,r2	2
12		FETCH 0x…4	
13	0x…4	bne r4,r1,L	
13-17		FETCH 0x…8	
18	0x…8	sub r1,r1,r1	0
18-22		FETCH 0x..C	
23	0x…8	j L	

At cycle 23

Execute `j L`



Compare No-cache vs. Cache

NO CACHE

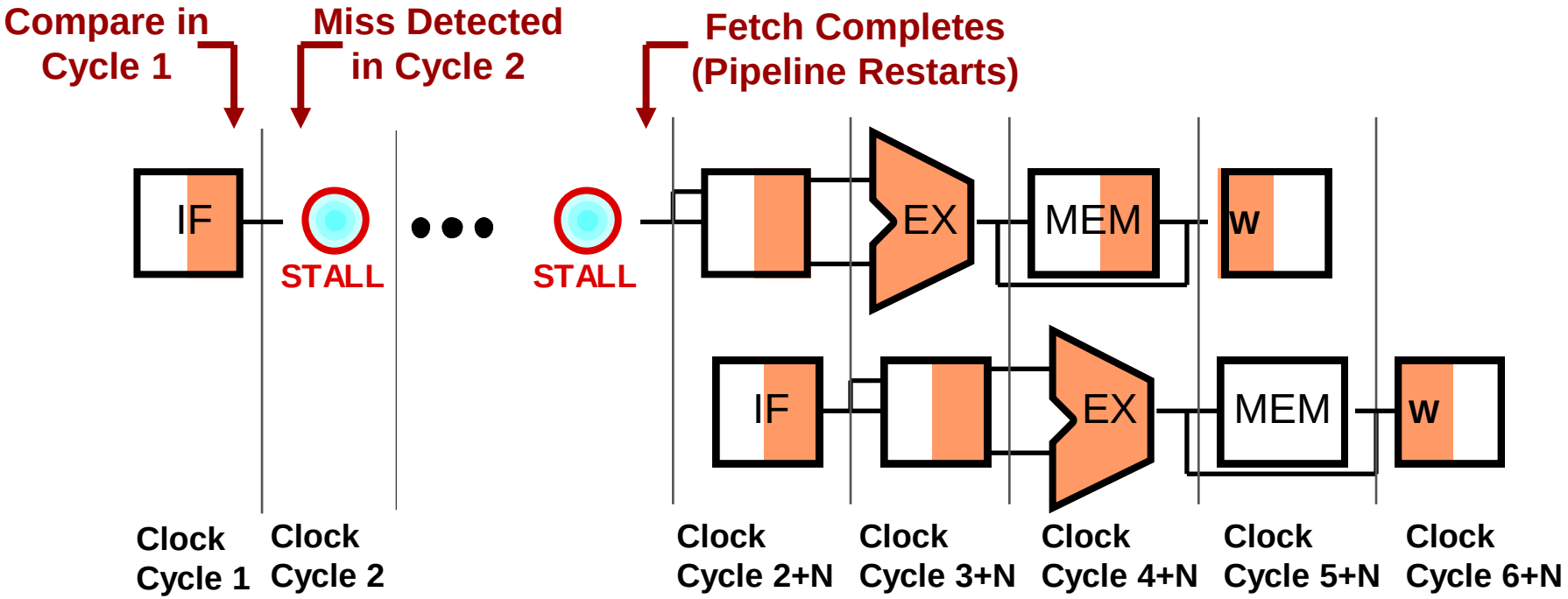
Cycle	Address	Op/Instr.
1-5		**FETCH 0x…0**
6	0x…0	**add r1,r1,r2**
6-10		**FETCH 0x…4**
11	0x…4	**bne r4,r1,L**
11-15		**FETCH 0x…0**
16	0x…0	**add r1,r1,2**
16-20		**FETCH 0x…4**
21	0x…4	**bne r4,r1,L**
21-25		**FETCH 0x…8**
26	0x…8	**sub r1,r1,r1**
26-30		**FETCH 0x..C**
31	0x…C	**j L**

CACHE

	Cycle	Address	Op/Instr.
M	1-5		**FETCH 0x…0**
	6	0x…0	**add r1,r1,r2**
M	6-10		**FETCH 0x…4**
	11	0x…4	**bne r4,r1,L**
H	11		**FETCH 0x…0**
	12	0x…0	**add r1,r1,r2**
H	12		**FETCH 0x…4**
	13	0x…4	**bne r4,r1,L**
M	13-17		**FETCH 0x…8**
	18	0x…8	**sub r1,r1,r1**
M	18-22		**FETCH 0x..C**
	23	0x…C	**j L**

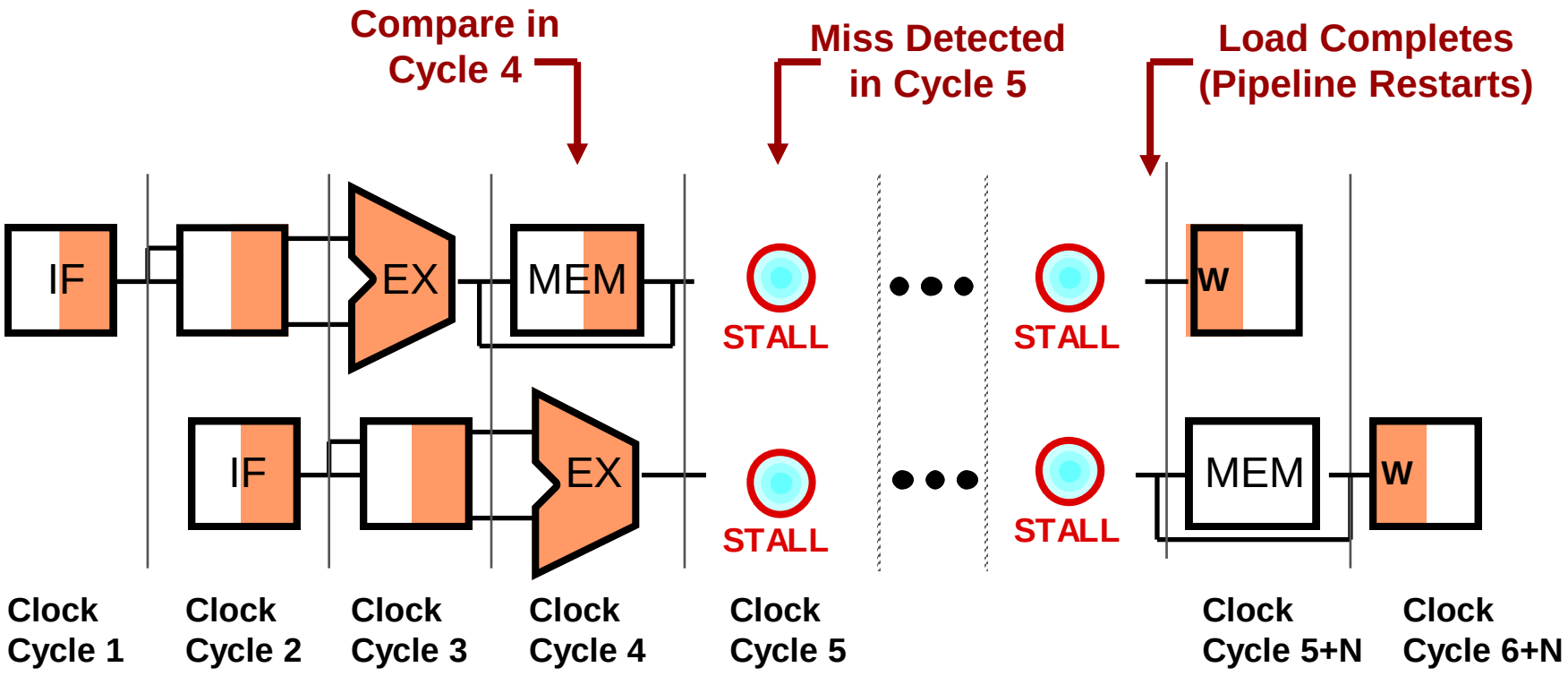
Cache Miss and the MIPS Pipeline

- Instruction Fetch



Cache Miss and the MIPS Pipeline

- **Load Instruction**



Coming Up: Four Key Cache Questions:

1. **Where** can block be **placed** in cache?
   (block placement)
2. **How** can block be **found** in cache? **…using a tag**
   (block identification)
3. **Which** block should be **replaced** on a miss?
   (block replacement)
4. **What happens** on a **write**?
   (write strategy)