

---

## Chapter Five (1/3)

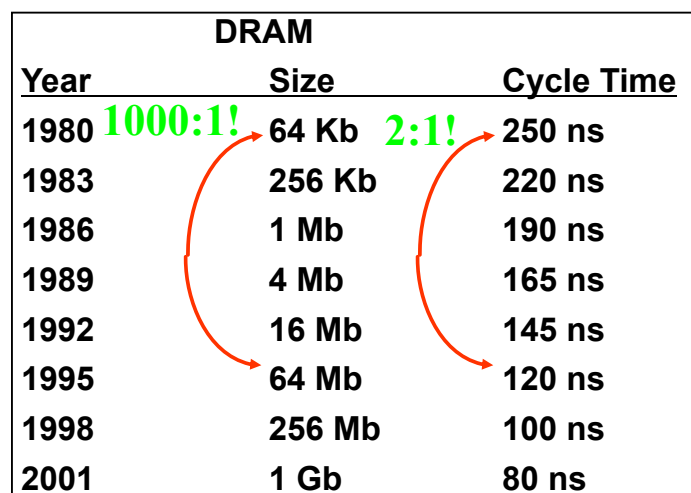
1

### Technology Trends (from 1st lecture)

---

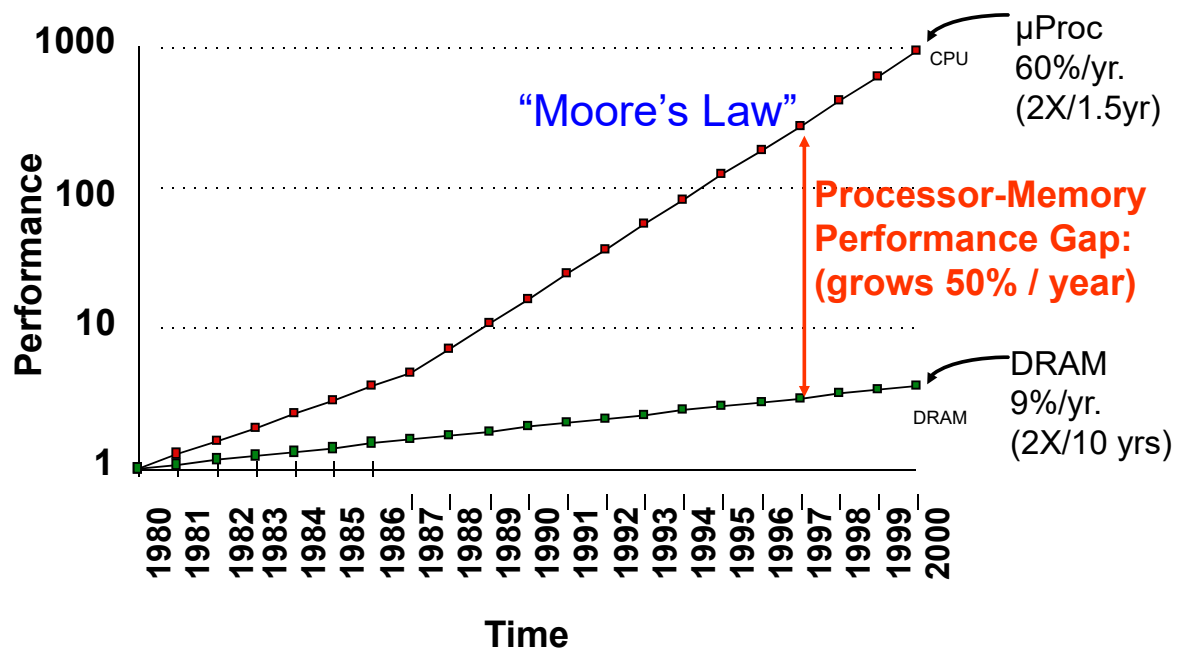
	Capacity	Speed (latency)
Logic:	2x in 3 years	2x in 3 years
DRAM:	4x in 3 years	2x in 10 years
Disk:	4x in 3 years	2x in 10 years

DRAM		
Year	Size	Cycle Time
1980	64 Kb	250 ns
1983	256 Kb	220 ns
1986	1 Mb	190 ns
1989	4 Mb	165 ns
1992	16 Mb	145 ns
1995	64 Mb	120 ns
1998	256 Mb	100 ns
2001	1 Gb	80 ns



# Who Cares About Memory?

## Processor-DRAM Memory Gap (latency)



CE, KWU Prof. S.W. LEE 3

## Today’s Situation: Microprocessors

- Microprocessor-DRAM performance gap
  - Microprocessor Clock Rates

Name	Date	Transistors	Microns	Clock speed	Data width
8080	1974	6,000	6	2 MHz	8 bits
8088	1979	29,000	3	5 MHz	16 bits 8-bit bus
80286	1982	134,000	1.5	6 MHz	16 bits
80386	1985	275,000	1.5	16 MHz	32 bits
80486	1989	1,200,000	1	25 MHz	32 bits
Pentium	1993	3,100,000	0.8	60 MHz	32 bits 64-bit bus
Pentium II	1997	7,500,000	0.35	233 MHz	32 bits 64-bit bus
Pentium III	1999	9,500,000	0.25	450 MHz	32 bits 64-bit bus
Pentium 4	2000	42,000,000	0.18	1.5 GHz	32 bits 64-bit bus
Pentium 4 "Prescott"	2004	125,000,000	0.09	3.8 GHz	32 bits 64-bit bus

CE, KWU Prof. S.W. LEE 4

# Today's Situation: Microprocessors

---

- Rely on caches to bridge gap
- Cache is a high-speed memory between the processor and main memory
- Microprocessor-DRAM performance gap
  - time of a full cache miss in instructions executed

Standard name	Memory clock	Cycle time	I/O Bus clock	Data transfers per second	Module name	Peak transfer rate	Timings tRAS-tCAS-tACC
DDR-200	100 MHz	10 ns	100 MHz	200 Million	PC-1600	1600 MB/s	50 ns
DDR-266	133 MHz	7.5 ns	133 MHz	266 Million	PC-2100	2100 MB/s	50 ns
DDR-333	166 MHz	6 ns	166 MHz	333 Million	PC-2700	2700 MB/s	40 ns
DDR-400	200 MHz	5 ns	200 MHz	400 Million	PC-3200	3200 MB/s	40 ns
DDR2-400	100 MHz	10 ns	200 MHz	400 Million	PC2-3200	3200 MB/s	30 ns
DDR2-533	133 MHz	7.5 ns	266 MHz	533 Million	PC2-4300	4266 MB/s	30 ns
DDR2-667	166 MHz	6 ns	333 MHz	667 Million	PC2-5300	5333 MB/s	30 ns
DDR2-800	200 MHz	5 ns	400 MHz	800 Million	PC2-6400	6400 MB/s	25 ns
DDR2-1066	266 MHz	3.75 ns	533 MHz	1066 Million	PC2-8600	8533 MB/s	23 ns

- 1980: no cache in  $\mu$ -proc;  
1997 2-level cache, 60% trans. on Alpha 21164  $\mu$ -proc

CE, KWU Prof. S.W. LEE 5

---

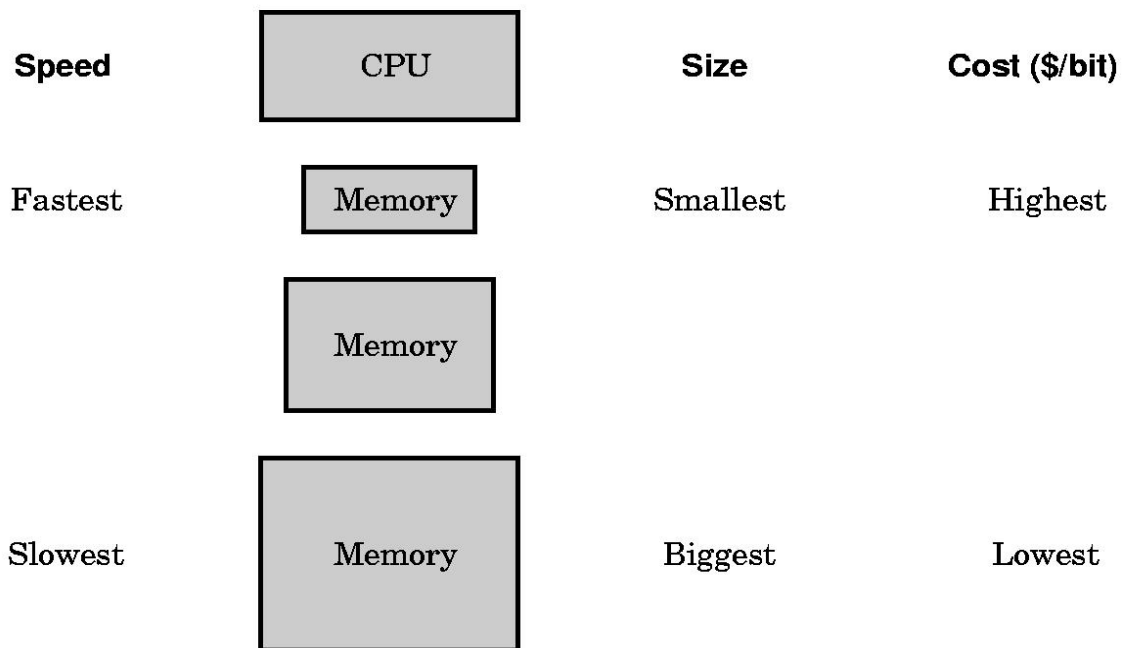
## The memory dilemma

---

- Assumption: on-chip instruction and data memories hold the entire program and its data and can be accessed in one cycle
- Reality check
  - In high performance machines, programs may require 100's of megabytes or even *gigabytes* of memory to run
  - Embedded processors have less needs but there is also less room for on-chip memory
- Basic problem
  - We need much more memory than can fit on the microprocessor chip
  - But we do not want to incur stall cycles every time the pipeline accesses instructions or data
  - At the same time, we need the memory to be economical for the machine to be competitive in the market

CE, KWU Prof. S.W. LEE 6

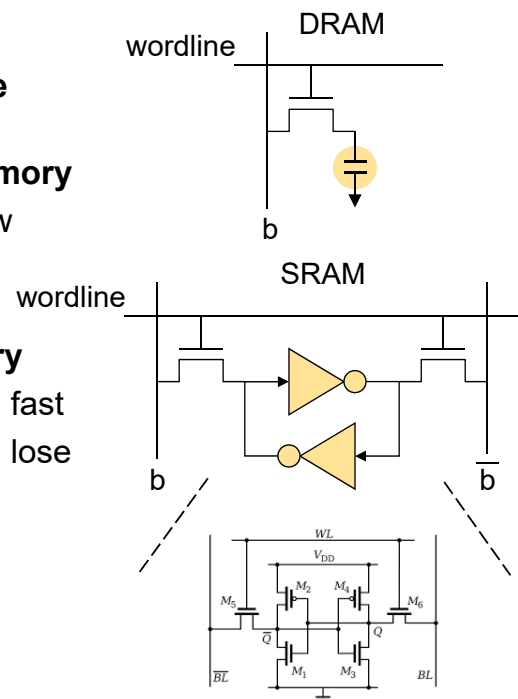
# Solution: a hierarchy of memories



CE, KWU Prof. S.W. LEE 7

## Memory Technology

- **Random Access:**
  - “Random” is good: access time is the same for all locations
  - **DRAM:** Dynamic Random Access Memory
    - High density, low power, cheap, slow
    - Dynamic: need to be “refreshed” regularly
  - **SRAM:** Static Random Access Memory
    - Low density, high power, expensive, fast
    - Static: content will last “forever”(until lose power)
- **“Non-so-random” Access Technology:**
  - Access time varies from location to location and from time to time
  - Examples: Disk, CDROM
- **Sequential Access Technology:** access time linear in location (e.g., Tape)



CE, KWU Prof. S.W. LEE 8

# Typical characteristics of each level

---

- First level (L1) is separate on-chip *instruction* and *data* caches placed where our instruction and data memories reside
  - 16-64KB for each cache (desktop/server machine)
  - Fast, power-hungry, not-so-dense, static RAM (SRAM)
- Second level (L2) consists of another larger *unified* cache
  - Holds both instructions and data
  - 256KB-4MB SRAM
  - On or off-chip
- Third level (L3) is yet another larger unified *shared* cache
  - Works for multiple cores
  - Off-Chip, Multichip package
  - 12MB-24MB SRAM
- Fourth level is *main memory*
  - 64MB-64GB
  - Slower, lower-power, denser dynamic RAM (DRAM)
- Final level is I/O (e.g., disk)

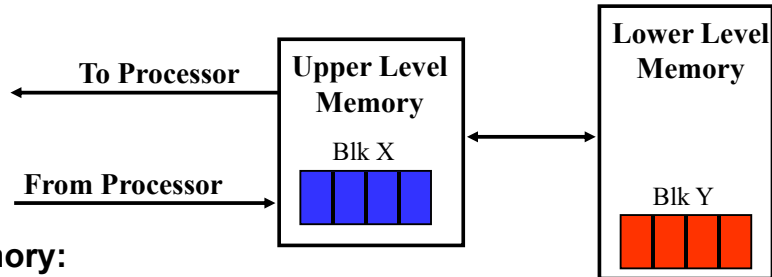
# Principle of Locality

---

- **Locality: Programs access a small proportion of their address space at any time**
  - **Temporal locality**
    - Items accessed recently are likely to be accessed again soon
    - e.g., instructions in a loop, induction variables
  - **Spatial locality**
    - Items near those accessed recently are likely to be accessed soon
    - E.g., sequential instruction access, array data
- **Taking Advantage of Locality**
  - **Memory hierarchy**

# Memory Hierarchy: How Does it Work?

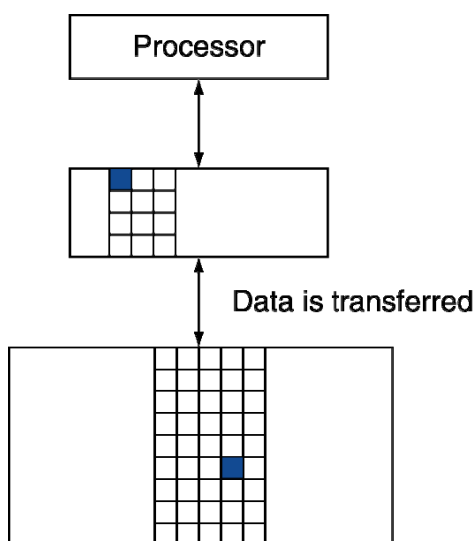
- **Temporal Locality** (Locality in Time):
  - Keep most recently accessed data items closer to the processor
- **Spatial Locality** (Locality in Space):
  - Move blocks consists of contiguous words to the upper levels



- **Registers  $\leftrightarrow$  Memory:**
  - by compiler (programmer?)
- **cache  $\leftrightarrow$  memory:**
  - by the hardware
- **memory  $\leftrightarrow$  disks:**
  - by the hardware and operating system (virtual memory)
  - by the programmer (files)

CE, KWU Prof. S.W. LEE 11

## Memory Hierarchy Levels



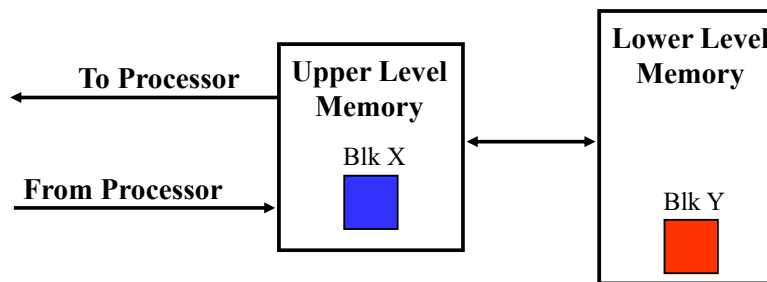
- **Block (aka line): unit of copying**
  - May be multiple words
- **If accessed data is present in upper level**
  - **Hit: access satisfied by upper level**
    - Hit ratio: hits/accesses
- **If accessed data is absent**
  - **Miss: block copied from lower level**
    - Time taken: miss penalty
    - Miss ratio: misses/accesses  
 $= 1 - \text{hit ratio}$
  - **Then accessed data supplied from upper level**

CE, KWU Prof. S.W. LEE 12

# Memory Hierarchy: Terminology

---

- **Hit**: data appears in some block in the upper level (example: Block X)
  - **Hit Rate**: the fraction of memory access found in the upper level
  - **Hit Time**: Time to access the upper level which consists of RAM access time + Time to determine hit/miss
- **Miss**: data needs to be retrieve from a block in the lower level (Block Y)
  - **Miss Rate** = 1 - (Hit Rate)
  - **Miss Penalty**: Time to replace a block in the upper level + Time to deliver the block the processor
- Hit Time << Miss Penalty



CE, KWU Prof. S.W. LEE

13

# General Principles of Memory

---

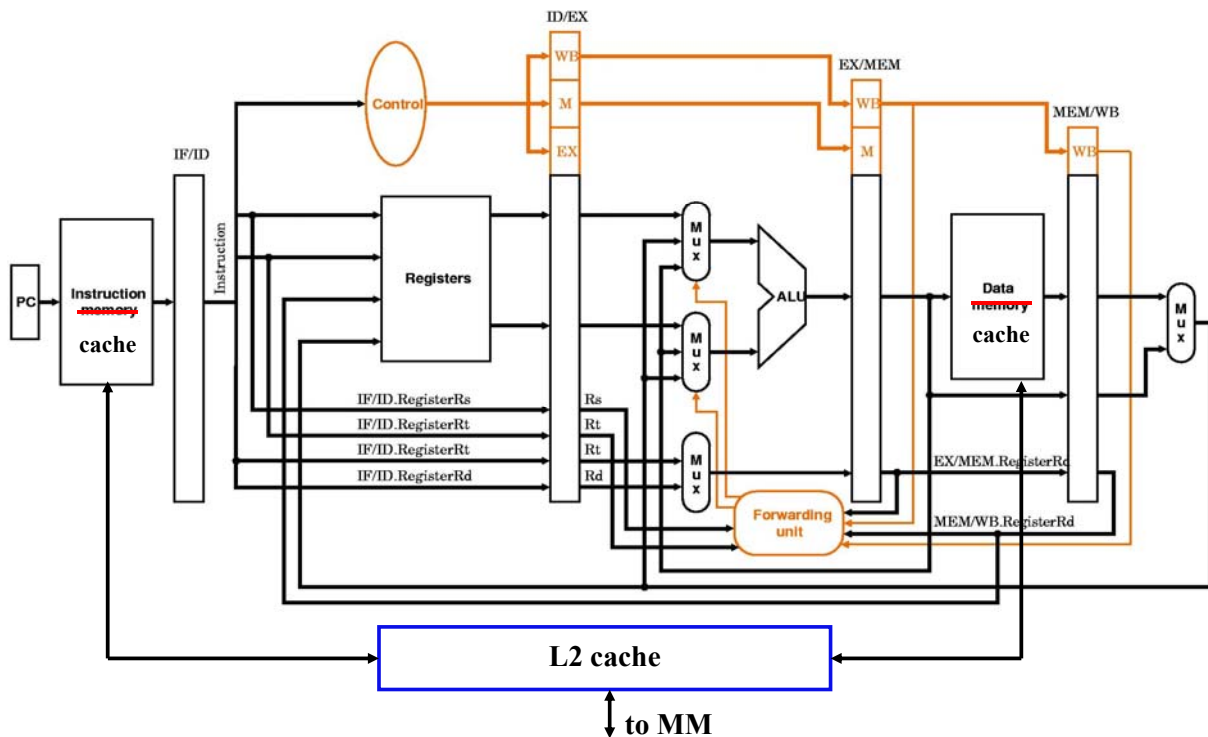
- **Locality**
  - **Temporal Locality**: referenced memory is likely to be referenced again soon (e.g. code within a loop)
  - **Spatial Locality**: memory close to referenced memory is likely to be referenced soon (e.g., data in a sequentially access array)
- **Definitions**
  - **Upper**: memory closer to processor
  - **Block**: minimum unit that is present or not present
  - **Block address**: location of block in memory
  - **Hit**: Data is found in the desired location
  - **Hit time**: time to access upper level
  - **Miss rate**: percentage of time item not found in upper level
- **Locality + smaller HW is faster = memory hierarchy**
  - **Levels**: each smaller, faster, more expensive/byte than level below
  - **Inclusive**: data found in upper leve also found in the lower level

CE, KWU Prof. S.W. LEE

14

# Caches and the pipeline

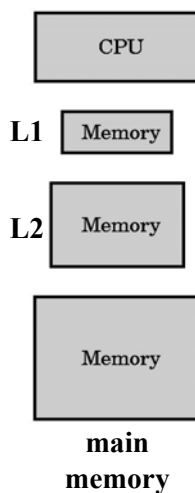
- L1 instruction and data caches and L2 cache



CE, KWU Prof. S.W. LEE

15

## Memory hierarchy operation



- (1) Search L1 for the instruction or data  
If found (*cache hit*), done
- (2) Else (*cache miss*), search L2 cache  
If found, place it in L1 and repeat (1)
- (3) Else, search main memory  
If found, place it in L2 and repeat (2)
- (4) Else, get it from I/O (Chapter 8)

Steps (1)-(3) are performed in hardware

- 1-3 cycles to get from L1 caches
- 5-20 cycles to get from L2 cache
- 50-200 cycles to get from main memory

CE, KWU Prof. S.W. LEE

16



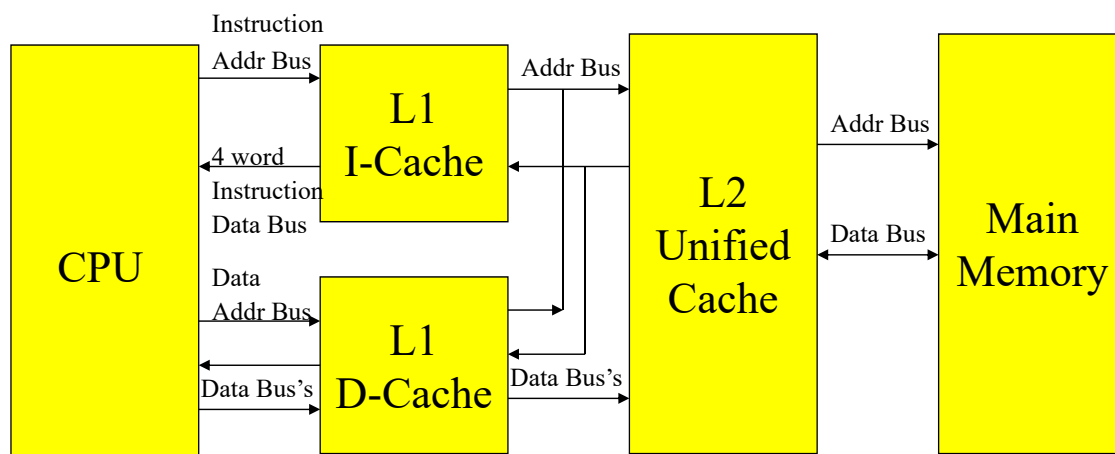
# Principle of Locality (POL) → Caches

- **Programs access a small portion of memory within a short time period**
  - **Temporal & Spatial locality:**
    - A large percentage of the time (typically >90%) the instruction or data is found in L1, the fastest memory
  - **Cheap, abundant main memory is accessed more rarely**
  - **Memory hierarchy operates at nearly the speed of expensive on-chip SRAM with the cost-effectiveness of DRAM**
- **Caches are small, fast, memories that hold recently accessed instructions and/or data**
  - **Separate L1 instruction and L1 data caches**
    - Need simultaneous access of instructions and data in pipelines
  - **L2 cache holds both instructions and data**
    - Simultaneous access not as critical since >90% of the instructions and data will be found in L1
    - PC or effective address from L1 is sent to L2 to search for the instruction or data

CE, KWU Prof. S.W. LEE 17

## Cache Structure

- **The L1 cache is usually separated into separate Data and Instruction caches**
- **The L2 and higher caches are normally unified caches, i.e. both instructions and data are in the same cache**

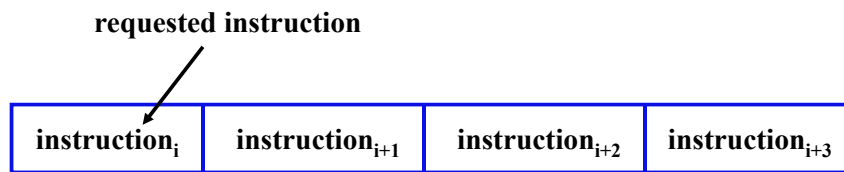


CE, KWU Prof. S.W. LEE 18

# How caches exploit the POL

---

- On a cache miss, a **block** of several instructions or data, including the requested item, are returned



- The entire block is placed into the cache so that future searches for items in the block will be successful
- Consider sequence of instructions and data accesses in this loop with a block size of 4 words

```
Loop: lw    $t0, 0($s1)
      addu  $t0, $t0, $s2
      sw    $t0, 0($s1)
      addi  $s1, $s1, -4
      bne   $s1, $zero, Loop
```

---

## Four Questions for Memory Hierarchy

---

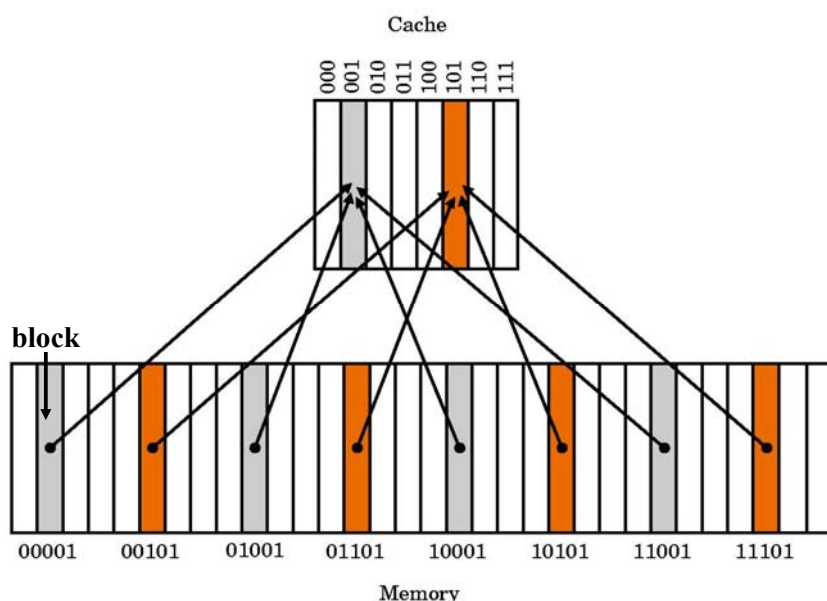
- Q1: Where can a block be placed in the upper level?
  - (Block placement)
- Q2: How is a block found if it is in the upper level?
  - (Block identification)
- Q3: Which block should be replaced on a miss?
  - (Block replacement)
- Q4: What happens on a write?
  - (Write strategy)

## Q1: Where can a block be placed?

- **Direct mapped:** each block can be placed in only one cache location
  - Each location in memory maps to only one location in cache
  - Easy to build since the memory mapping is unique
- **Set associative:** each block can be placed in any of n cache locations
  - Each Set is direct mapped to memory
  - Cache is fully associative between sets
  - If there are n blocks in a set, the cache placement is called n-way set associative
- **Fully associative:** each block can be placed in any cache location
  - Any location in cache can map to any location in memory
  - Difficult to build since all cache locations must be checked for a match to any specific memory value

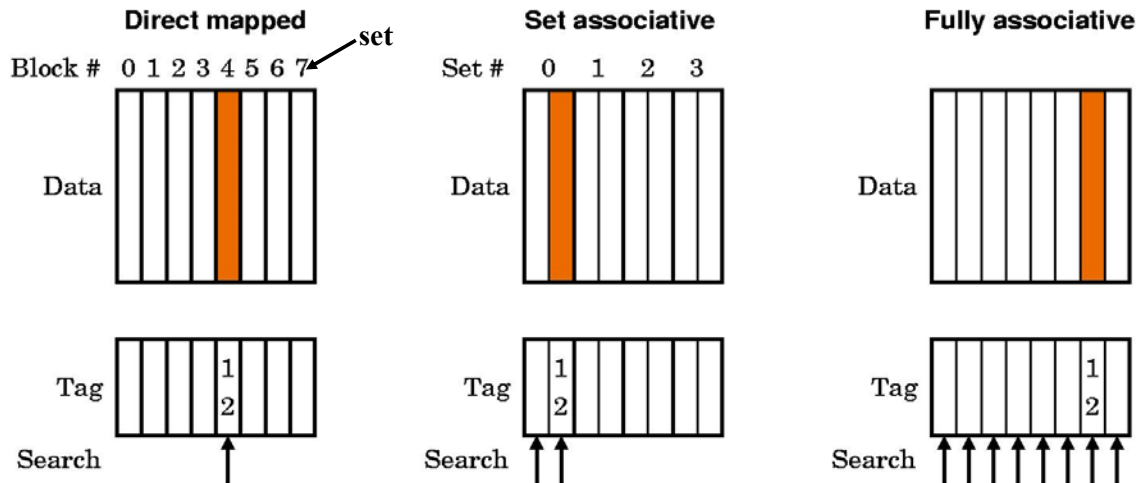
## Placing blocks in cache

- The cache is much smaller than main memory
  - Multiple memory blocks must share the same cache location



# Block placement

- Searching for and placing block 12 in caches of size 8 blocks
  - Direct mapped: Block no. = (Block addr.) mod (No. of blocks)
  - 2-way set associative: Set no. = (Block addr.) mod (No. of sets)
  - Fully associative: Block 12 can go anywhere



CE, KWU Prof. S.W. LEE

23

## Q2: How Is a Block Found?

- The address can be divided into two main parts
  - **Block offset:** selects the data from the block
    - offset size =  $\log_2(\text{block size})$
  - **Block address: tag + index**
    - index: selects set in cache
      - index size =  $\log_2(\# \text{blocks/associativity})$
    - tag: compared to tag in cache to determine hit
      - tag size = address size - index size - offset size
- Each block has a valid bit that tells if the block is valid - the block is in the cache if the tags match and the valid bit is set.

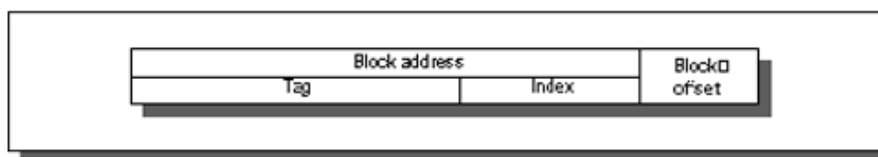


FIGURE 5.3 The three portions of an address in a set-associative or direct-mapped cache.

# Addressing a direct mapped cache



- Need  $\log_2$  (number of sets) of the address bits (called the *index*) to select the block location
- *block offset* used to select the desired byte, half-word, or word within the block
- Remaining bits (called the tag) are used to distinguish between different blocks that share the same cache location
- Block is placed in the set *index*
- number of sets = cache size/block size

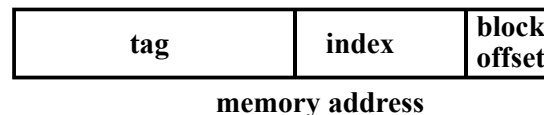
assume data cache  
with 16 byte blocks

8 sets

4 block offset bits

3 index bits

25 tag bits

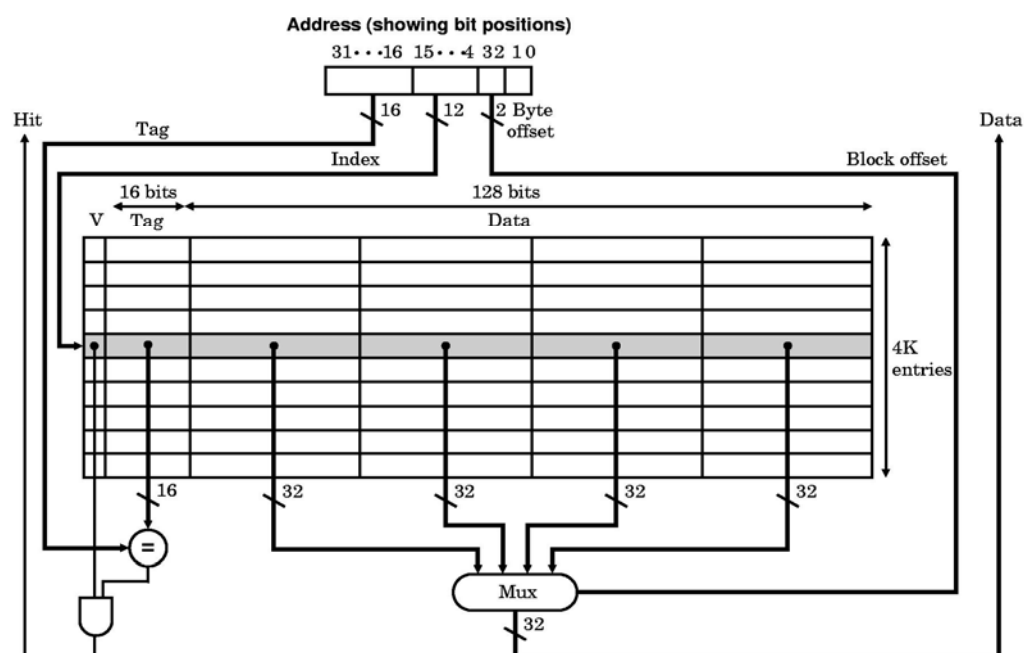


CE, KWU Prof. S.W. LEE

25

## Direct mapped cache organization

- 64KB instruction cache with 16 byte (4 word) blocks



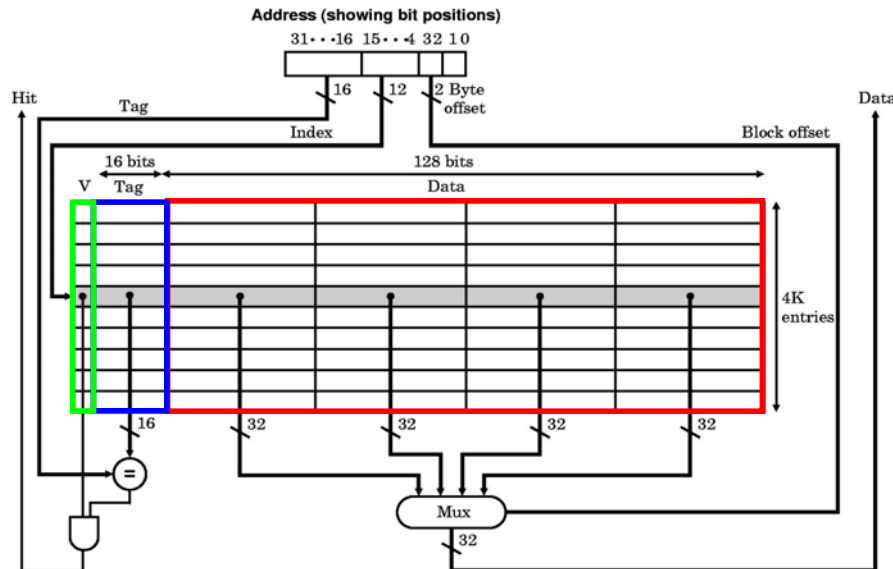
– 4K sets (64KB/16B) → need 12 address bits to pick

CE, KWU Prof. S.W. LEE

26

# Direct mapped cache organization

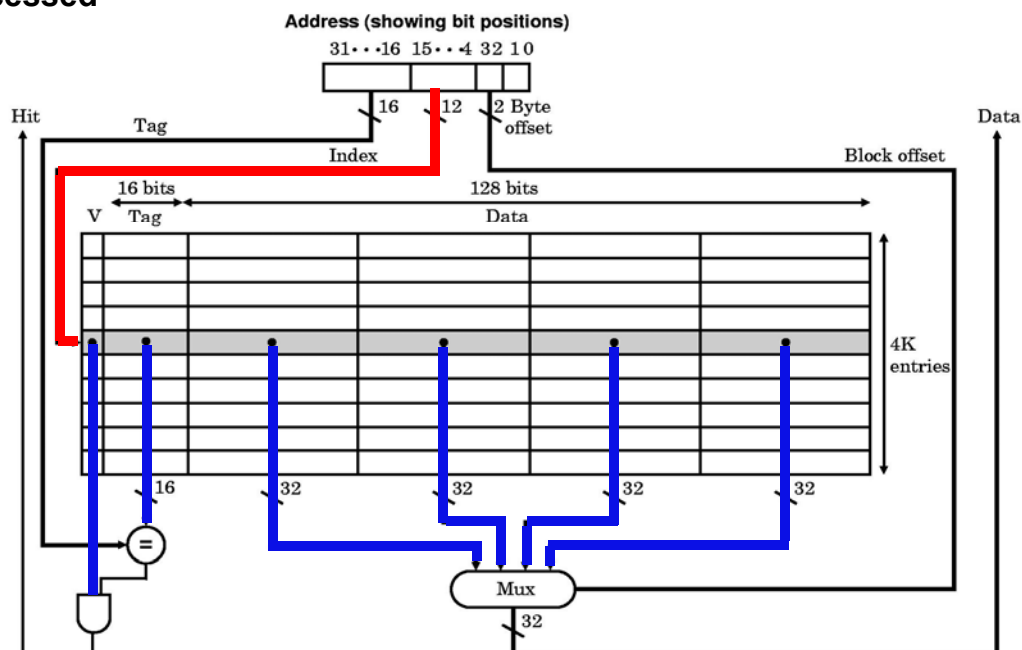
- The **data section** of the cache holds the instructions
- The **tag section** holds the part of the memory address used to distinguish different blocks
- A **valid bit** associated with each set indicates if the instructions are valid or not



CE, KWU Prof. S.W. LEE 27

# Direct mapped cache access

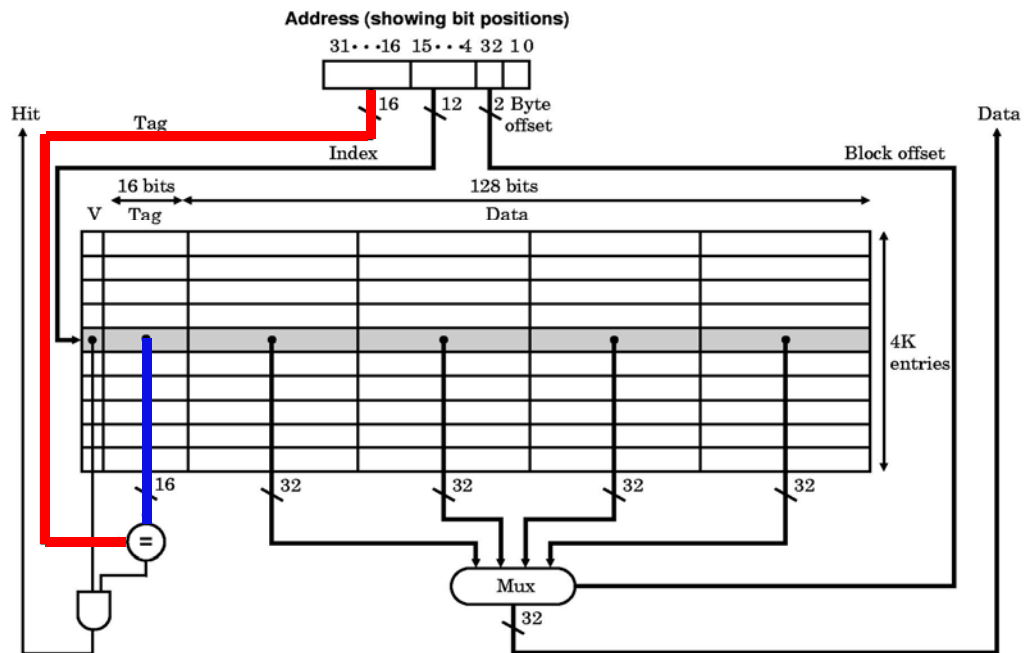
- The index bits are used to select one of the sets
- The data, tag, and Valid bit from the selected set are simultaneously accessed



CE, KWU Prof. S.W. LEE 28

# Direct mapped cache access

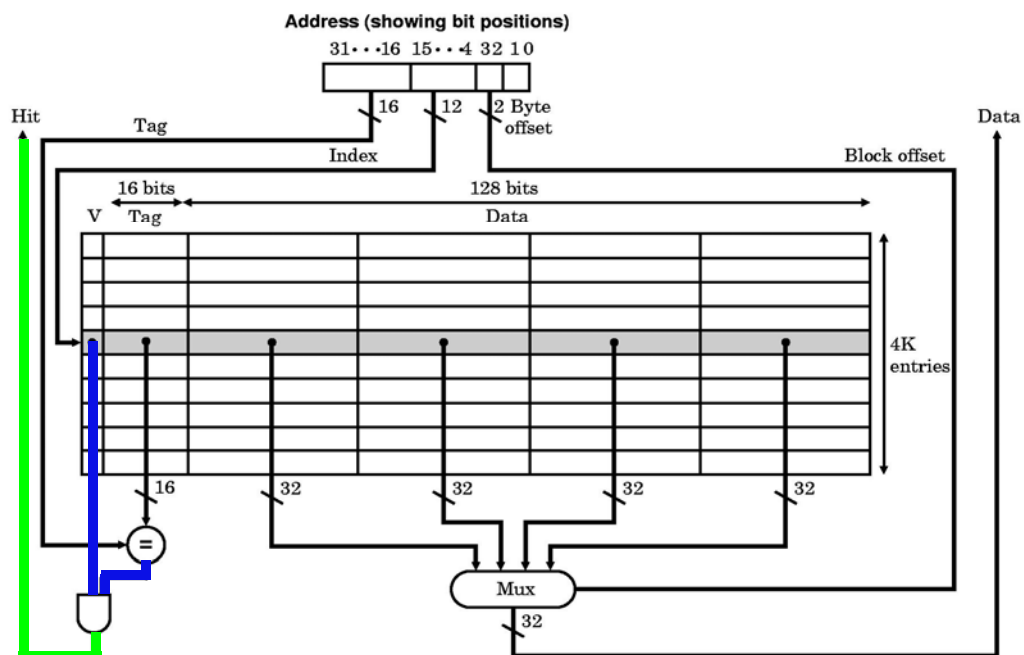
- The tag from the selected entry is compared with the tag field of the address



CE, KWU Prof. S.W. LEE 29

# Direct mapped cache access

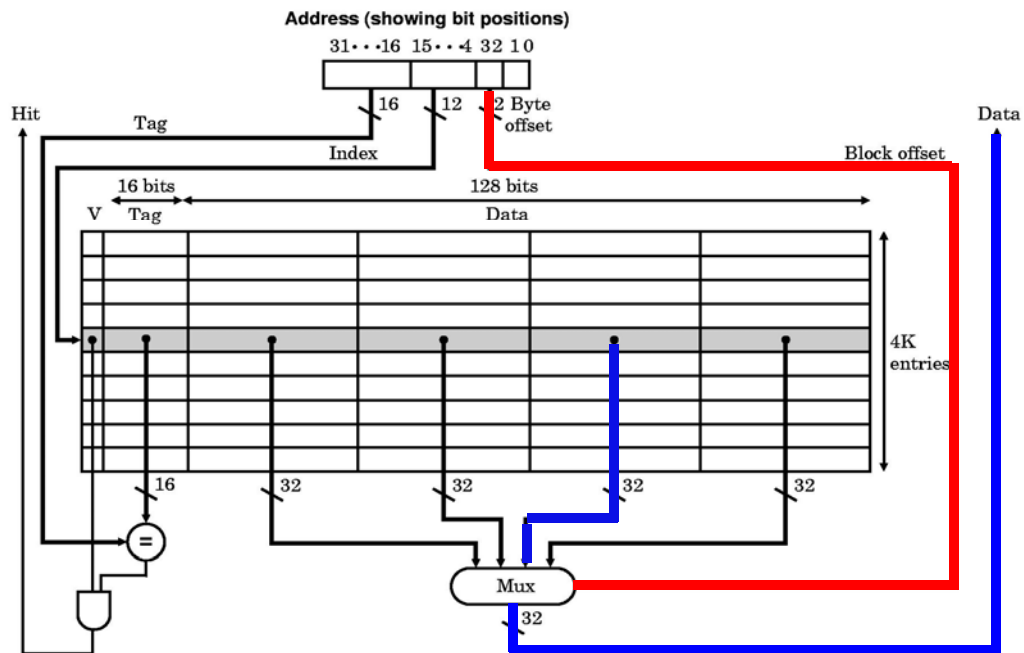
- A match between the tags and a Valid bit that is set indicates a cache hit



CE, KWU Prof. S.W. LEE 30

# Direct mapped cache access

- The block offset selects the desired instruction



CE, KWU Prof. S.W. LEE 31

## Cache Example

CACHE

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	10	Mem[10010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Memory Access

Time	Word addr	Binary addr	Hit/miss	Cache block
1	22	10 110	Miss	110
2	26	11 010	Miss	010
3	22	10 110	Hit	110
4	26	11 010	Hit	010
5	16	10 000	Miss	000
6	3	00 011	Miss	011
7	16	10 000	Hit	000
8	18	10 010	Miss	010
...	...			

CE, KWU Prof. S.W. LEE 32