Chapter 8

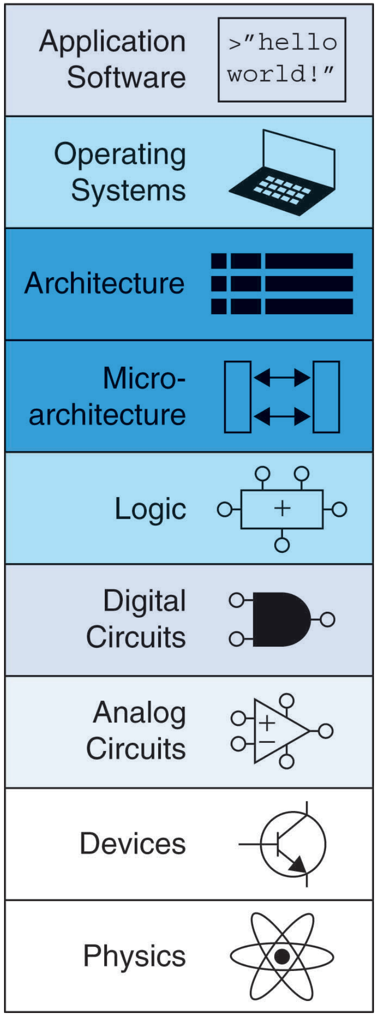
MEMORY & I/O SYSTEMS

***Digital Design and Computer Architecture*, 2nd Edition**

David Money Harris and Sarah L. Harris

Chapter 8 :: Topics

- **Introduction**
- **Memory System Performance Analysis**
- **Caches**
- **Virtual Memory**
- **Memory-Mapped I/O**
- **Summary**

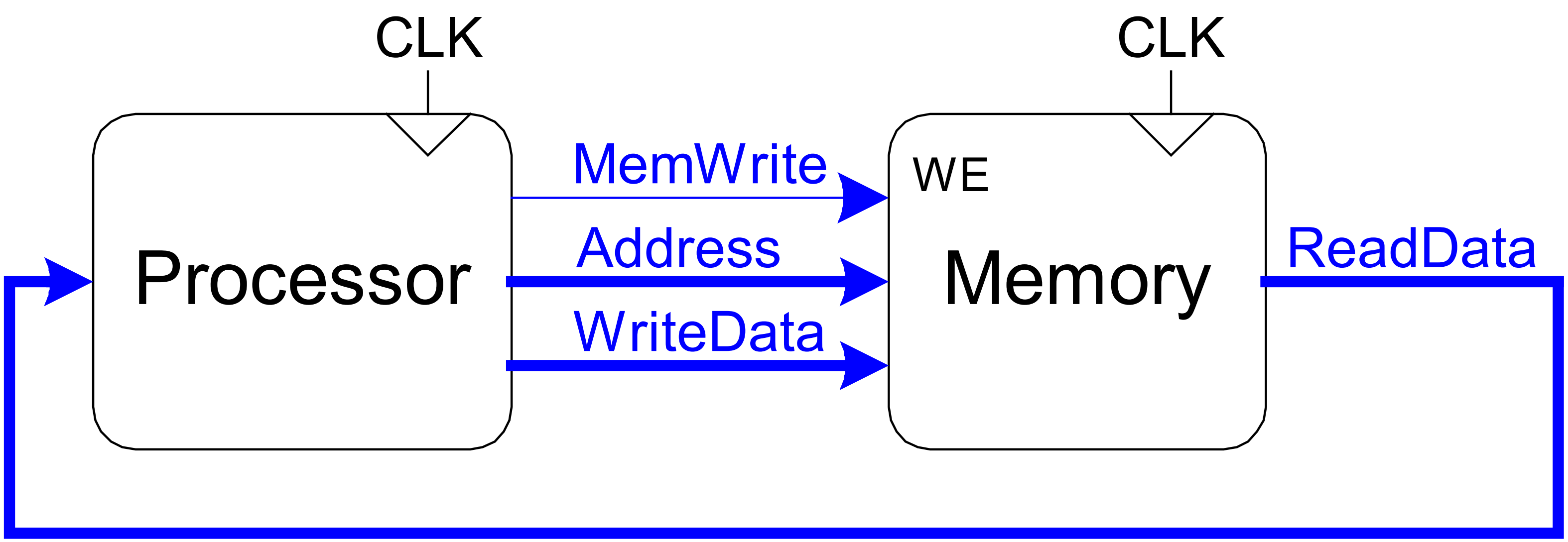


ELSEVIER

Introduction

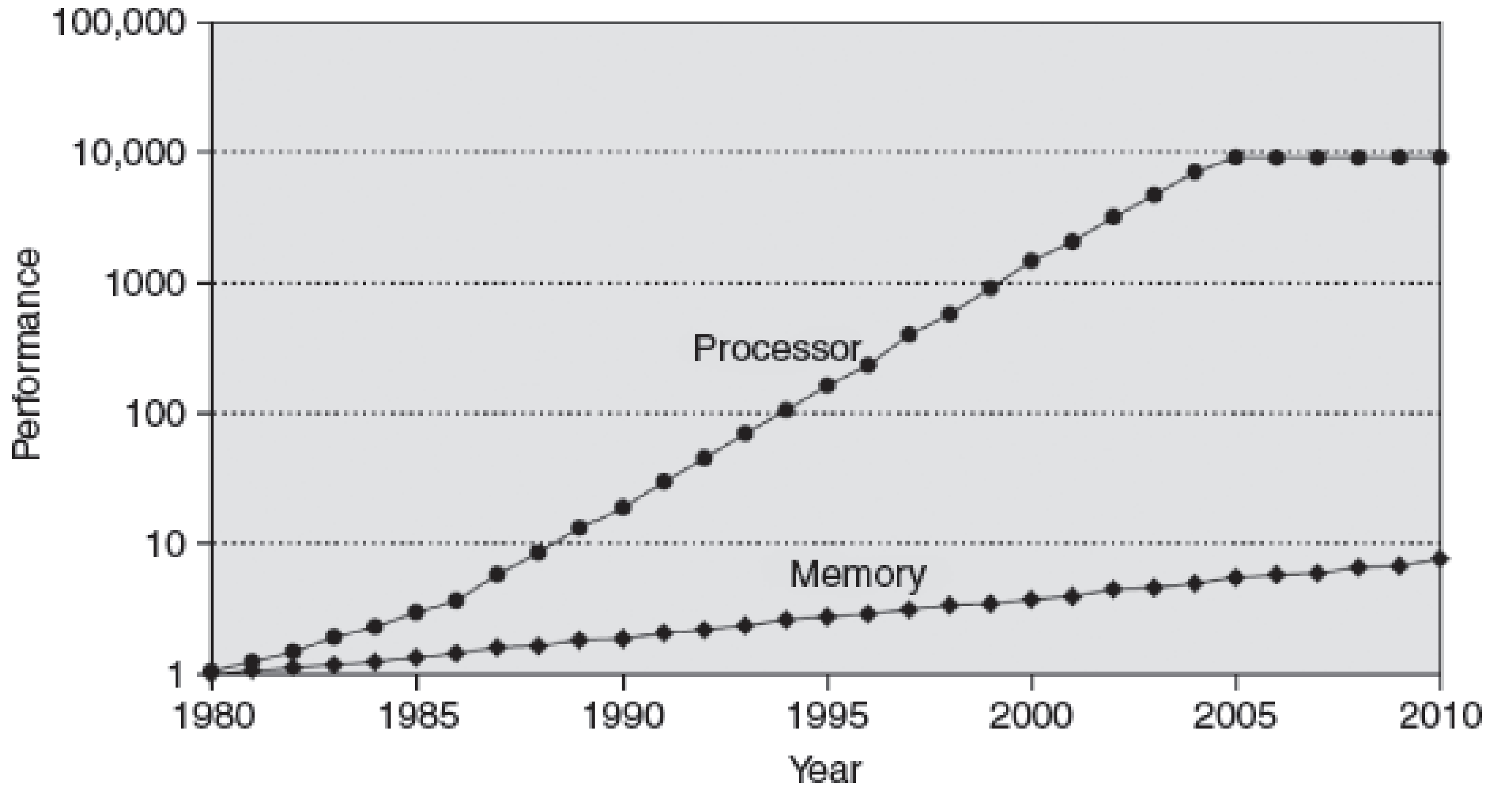
- Computer performance depends on:
  - Processor performance
  - Memory system performance

Memory Interface



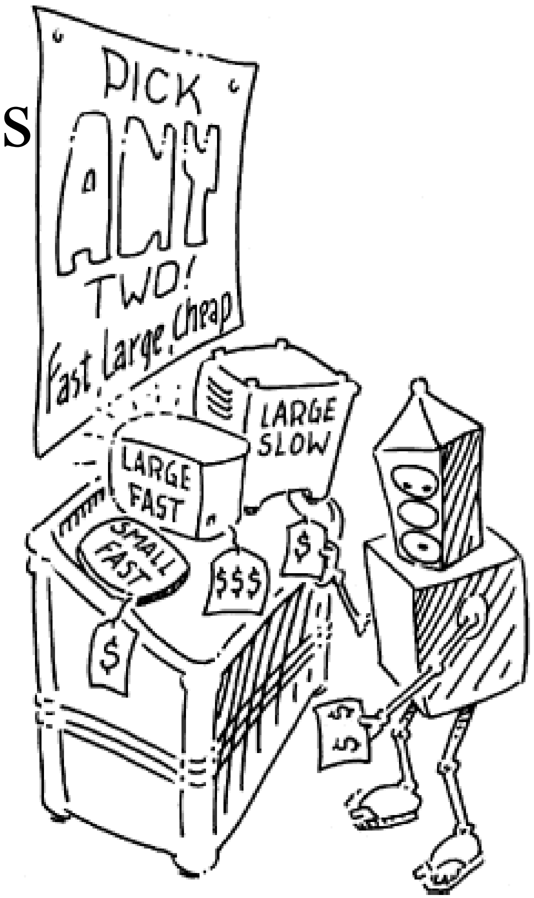
Processor-Memory Gap

In prior chapters, assumed access memory in 1 clock cycle – but hasn't been true since the 1980's



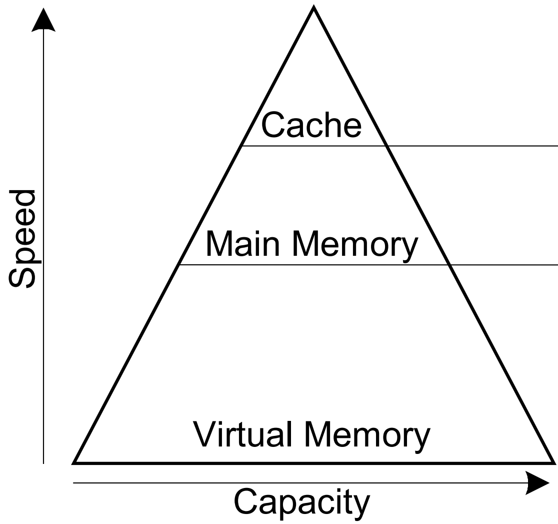
Memory System Challenge

- Make memory system appear as fast as processor
- Use hierarchy of memories
- Ideal memory:
  - Fast
  - Cheap (inexpensive)
  - Large (capacity)



But can only choose two!

Memory Hierarchy



Technology	Price / GB	Access Time (ns)	Bandwidth (GB/s)
SRAM	$10,000	1	25+
DRAM	$10	10 - 50	10
SSD	$1	100,000	0.5
HDD	$0.1	10,000,000	0.1

Locality

Exploit locality to make memory accesses fast

- **Temporal Locality:**
  - Locality in time
  - If data used recently, likely to use it again soon
  - **How to exploit:** keep recently accessed data in higher levels of memory hierarchy
- **Spatial Locality:**
  - Locality in space
  - If data used recently, likely to use nearby data soon
  - **How to exploit:** when access data, bring nearby data into higher levels of memory hierarchy too

Memory Performance

- **Hit:** data found in that level of memory hierarchy
- **Miss:** data not found (must go to next level)

Hit Rate = # hits / # memory accesses
= 1 – Miss Rate

Miss Rate = # misses / # memory accesses
= 1 – Hit Rate

- **Average memory access time (AMAT):** average time for processor to access data

$$\text{AMAT} = t_{\text{cache}} + MR_{\text{cache}}[t_{MM} + MR_{MM}(t_{VM})]$$

MEMORY & I/O SYSTEMS

Memory Performance Example 1

- A program has 2,000 loads and stores
- 1,250 of these data values in cache
- Rest supplied by other levels of memory hierarchy
- **What are the hit and miss rates for the cache?**

Memory Performance Example 1

- A program has 2,000 loads and stores
- 1,250 of these data values in cache
- Rest supplied by other levels of memory hierarchy
- **What are the hit and miss rates for the cache?**

Hit Rate = 1250/2000 **= 0.625**

Miss Rate = 750/2000 = **0.375** = 1 – Hit Rate

Memory Performance Example 2

- Suppose processor has 2 levels of hierarchy: cache and main memory
- $t_{\text{cache}} = 1$ cycle, $t_{MM} = 100$ cycles
- **What is the AMAT of the program from Example 1?**

Memory Performance Example 2

- Suppose processor has 2 levels of hierarchy: cache and main memory
- $t_{\text{cache}} = 1$ cycle, $t_{MM} = 100$ cycles
- **What is the AMAT of the program from Example 1?**

$$\begin{aligned} \textbf{AMAT} \quad &= t_{\text{cache}} + MR_{\text{cache}}(t_{MM}) \\ &= [1 + 0.375(100)] \text{ cycles} \\ &= \mathbf{38.5\ cycles} \end{aligned}$$

MEMORY & I/O SYSTEMS

ELSEVIER

Gene Amdahl, 1922-

- **Amdahl's Law:** the effort spent increasing the performance of a subsystem is wasted unless the subsystem affects a large percentage of overall performance
- Co-founded 3 companies, including one called Amdahl Corporation in 1970

Cache

- Highest level in memory hierarchy
- Fast (typically ~ 1 cycle access time)
- Ideally supplies most data to processor
- Usually holds most recently accessed data

Cache Design Questions

- What data is held in the cache?
- How is data found?
- What data is replaced?

Focus on data loads, but stores follow same principles

What data is held in the cache?

- Ideally, cache anticipates needed data and puts it in cache
- But impossible to predict future
- Use past to predict future – temporal and spatial locality:
  - **Temporal locality:** copy newly accessed data into cache
  - **Spatial locality:** copy neighboring data into cache too

Cache Terminology

- **Capacity (*C*):**
  - number of data bytes in cache
- **Block size (*b*):**
  - bytes of data brought into cache at once
- **Number of blocks ($B = C/b$):**
  - number of blocks in cache: $B = C/b$
- **Degree of associativity (*N*):**
  - number of blocks in a set
- **Number of sets ($S = B/N$):**
  - each memory address maps to exactly one cache set

How is data found?

- Cache organized into *S* sets
- Each memory address maps to exactly one set
- Caches categorized by # of blocks in a set:
  - **Direct mapped:** 1 block per set
  - ***N*-way set associative:** *N* blocks per set
  - **Fully associative:** all cache blocks in 1 set
- Examine each organization for a cache with:
  - Capacity (*C* = 8 words)
  - Block size (*b* = 1 word)
  - So, number of blocks (*B* = 8)

MEMORY & I/O SYSTEMS

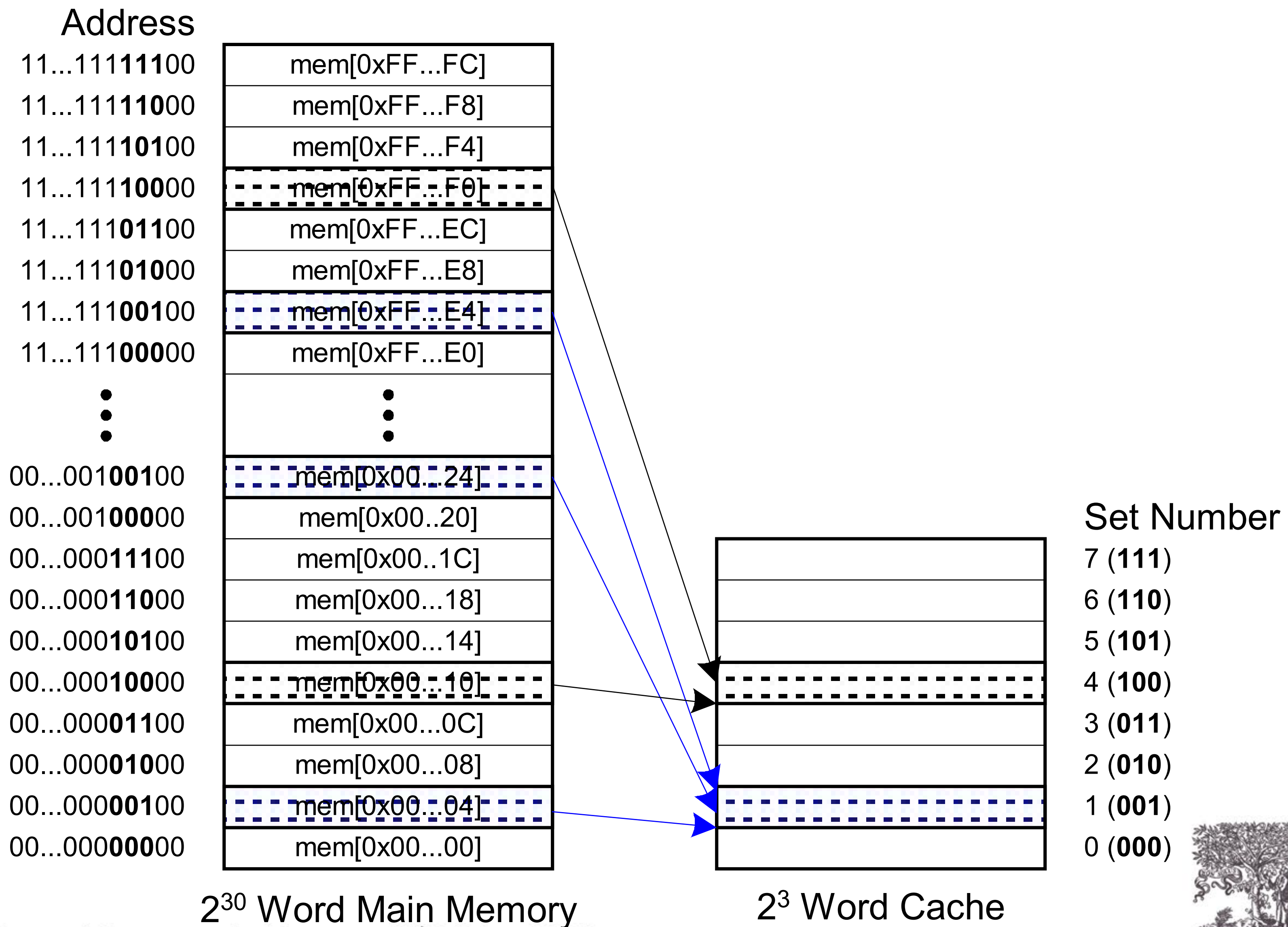
ELSEVIER

Example Cache Parameters

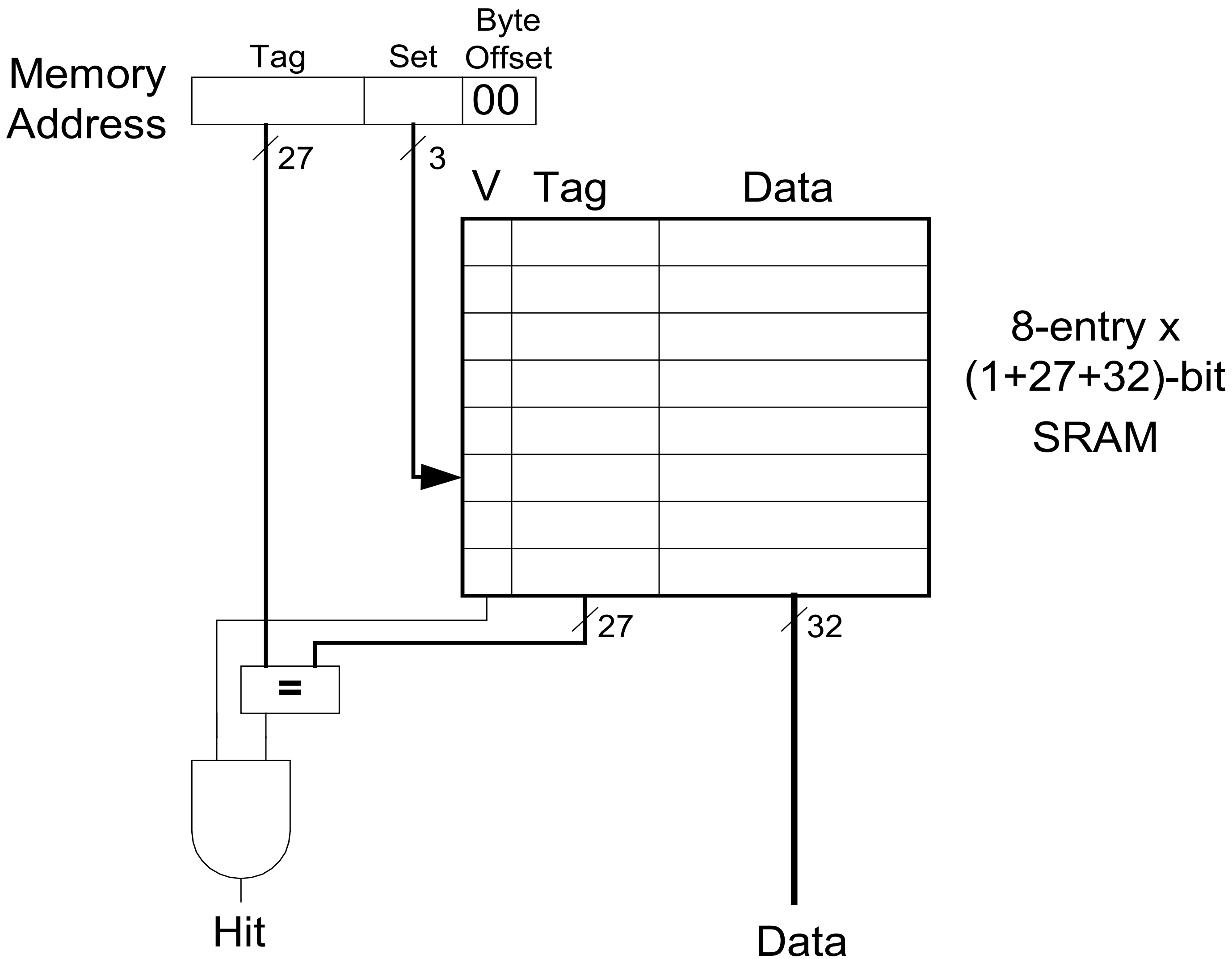
- ***C*** = **8** words (capacity)
- ***b*** = **1** word (block size)
- So, ***B*** = **8** (# of blocks)

Ridiculously small, but will illustrate organizations

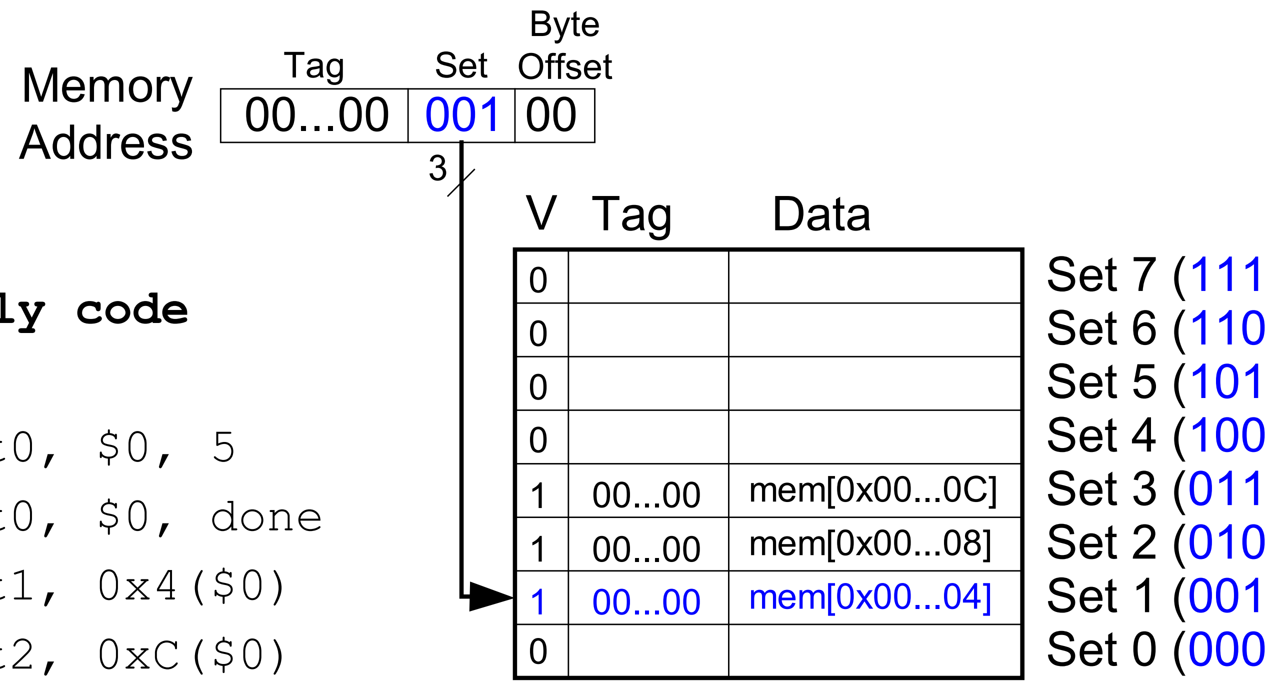
Direct Mapped Cache



Direct Mapped Cache Hardware



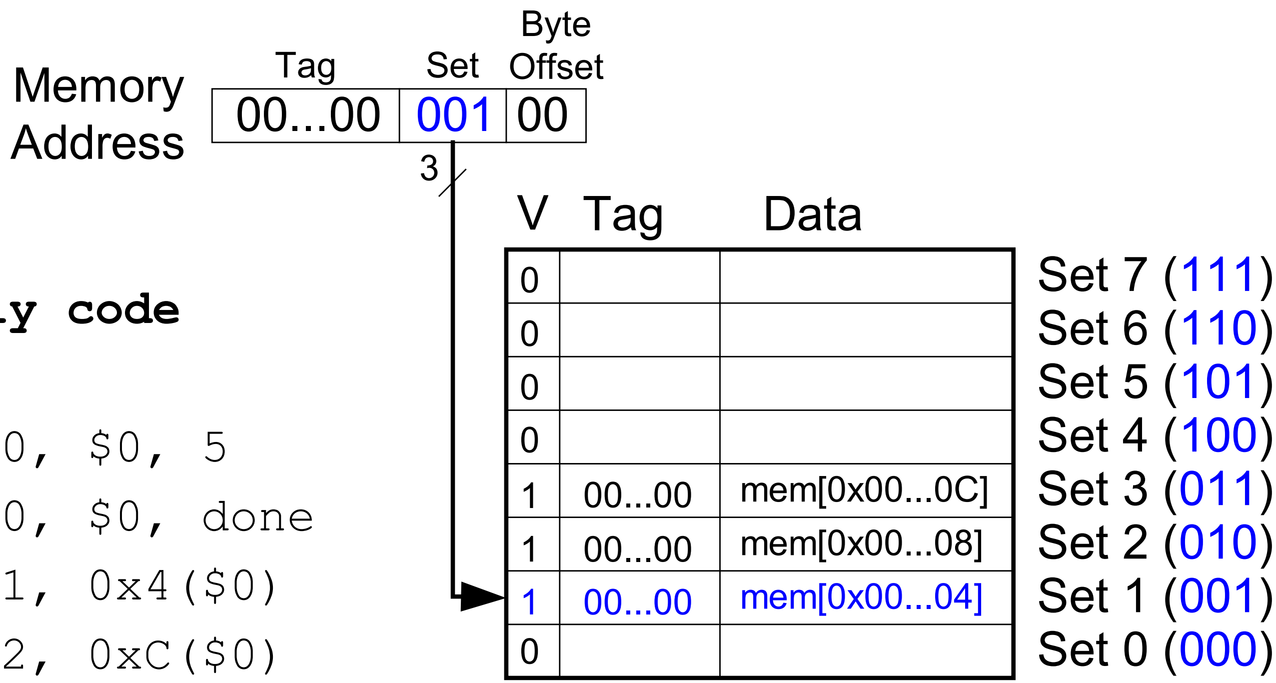
Direct Mapped Cache Performance




```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0xC($0)
        lw   $t3, 0x8($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = ?

Direct Mapped Cache Performance




```
# MIPS assembly code

       addi $t0, $0, 5
loop:  beq  $t0, $0, done
       lw   $t1, 0x4($0)
       lw   $t2, 0xC($0)
       lw   $t3, 0x8($0)
       addi $t0, $t0, -1
       j    loop
done:
```

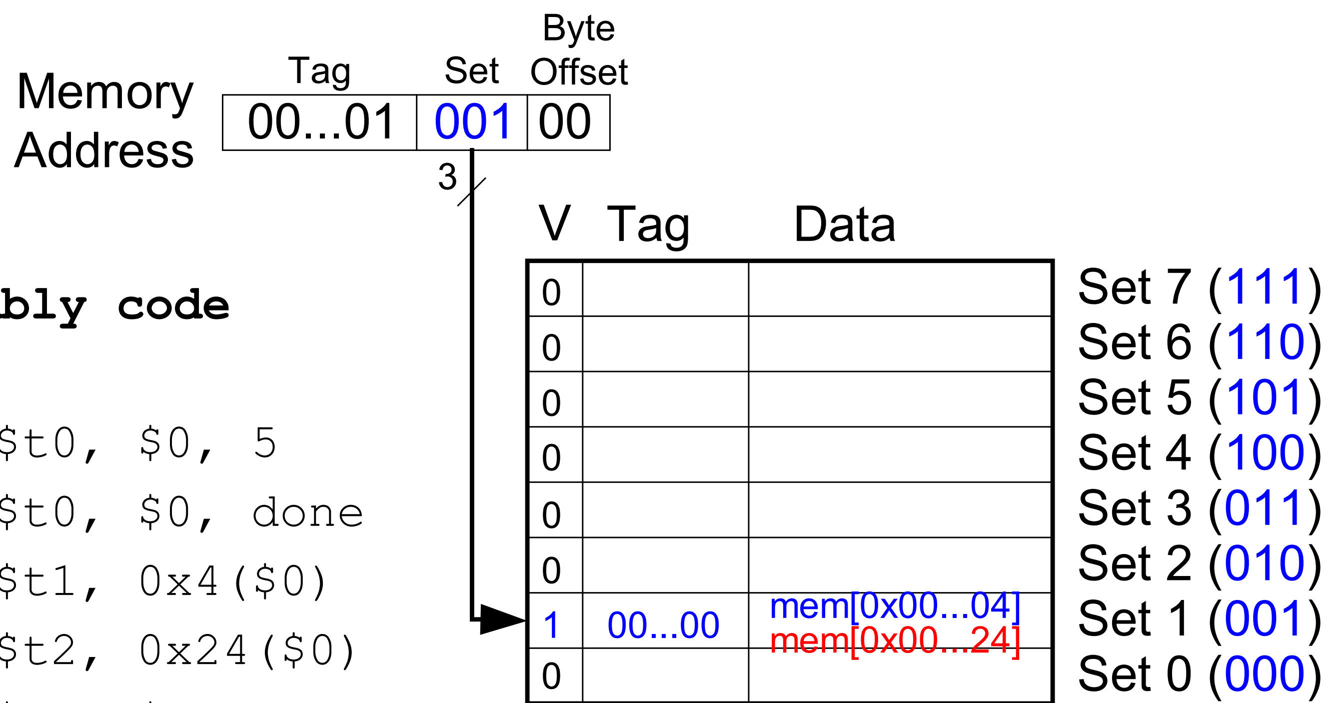
Miss Rate = 3/15

= 20%

Temporal Locality

Compulsory Misses

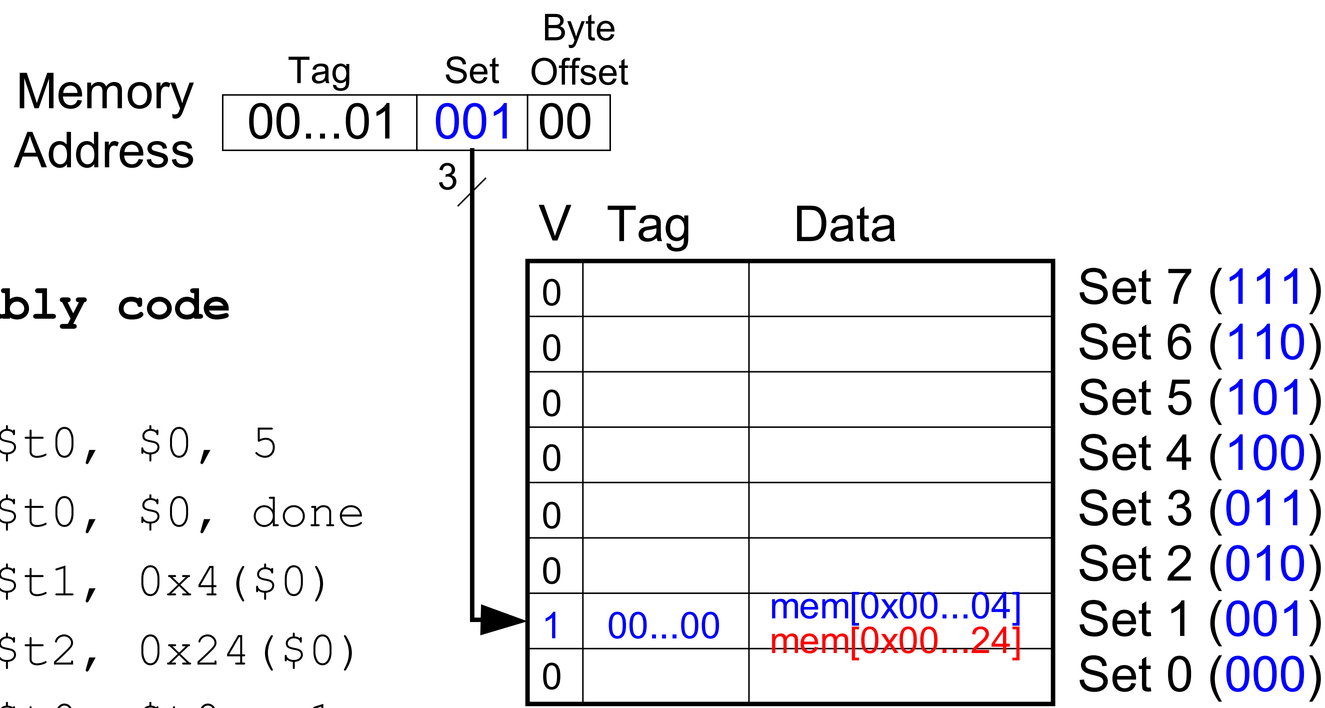
Direct Mapped Cache: Conflict




```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = ?

Direct Mapped Cache: Conflict



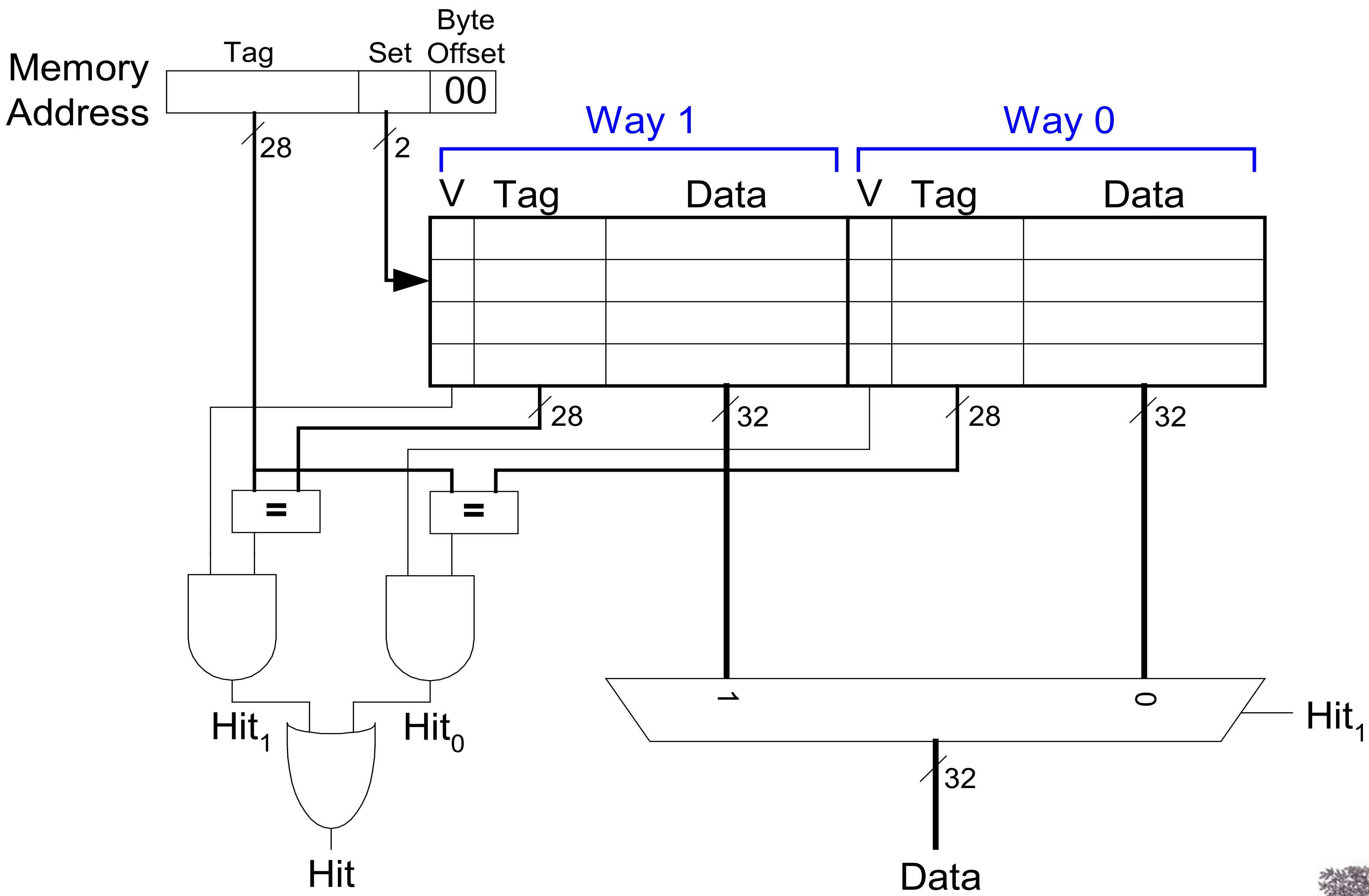

```
# MIPS assembly code

       addi $t0, $0, 5
loop:  beq  $t0, $0, done
       lw   $t1, 0x4($0)
       lw   $t2, 0x24($0)
       addi $t0, $t0, -1
       j    loop
done:
```

Miss Rate = 10/10

= 100%

Conflict Misses

N-Way Set Associative Cache



N-Way Set Associative Performance

```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = ?

Way 1 V	Way 1 Tag	Way 1 Data	Way 0 V	Way 0 Tag	Way 0 Data	
0			0			Set 3
0			0			Set 2
0			0			Set 1
0			0			Set 0

N-Way Set Associative Performance

```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```

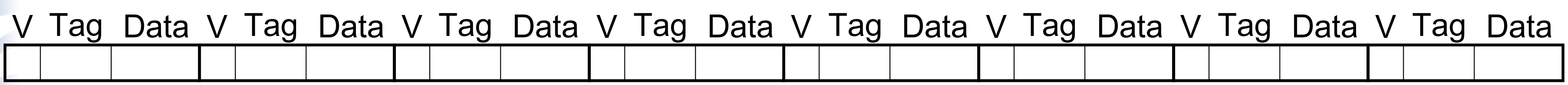
Miss Rate = 2/10

= 20%

Associativity reduces conflict misses

	Way 1			Way 0		
	V	Tag	Data	V	Tag	Data
Set 3	0			0		
Set 2	0			0		
Set 1	1	00...10	mem[0x00...24]	1	00...00	mem[0x00...04]
Set 0	0			0		

Fully Associative Cache

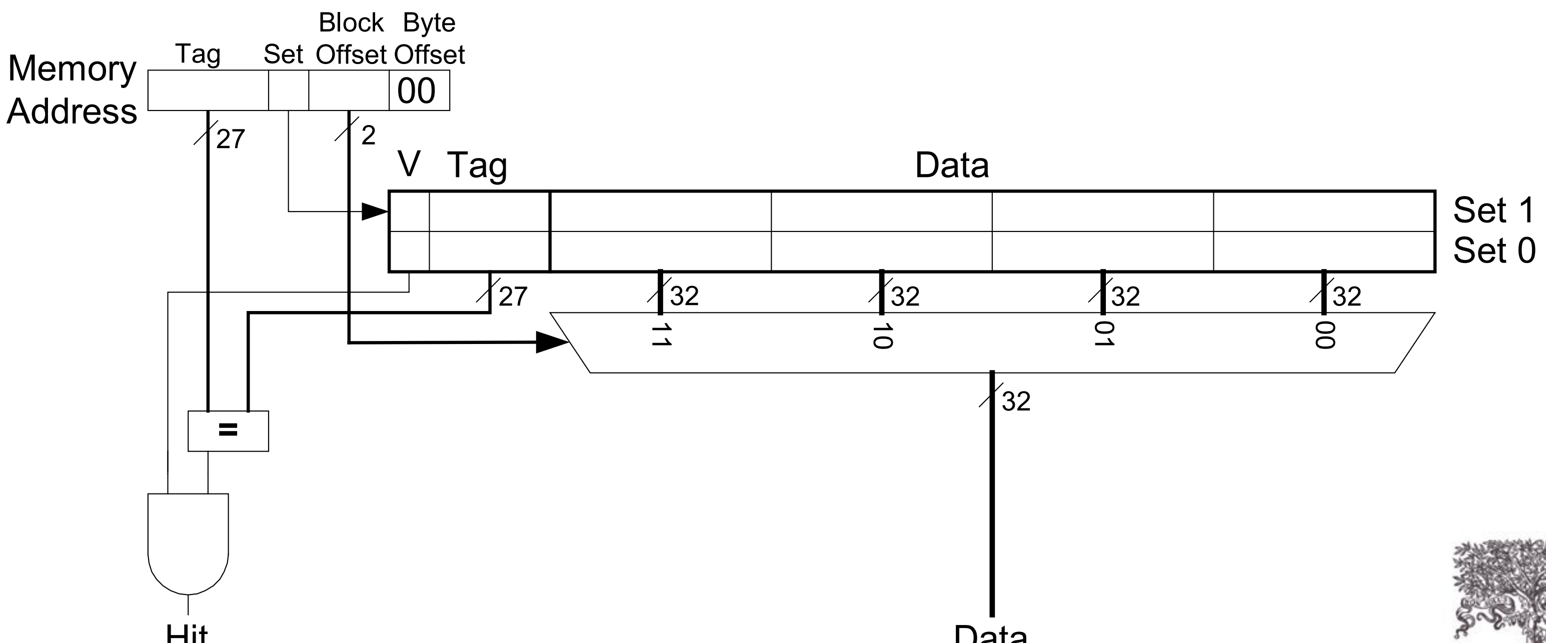


Reduces conflict misses

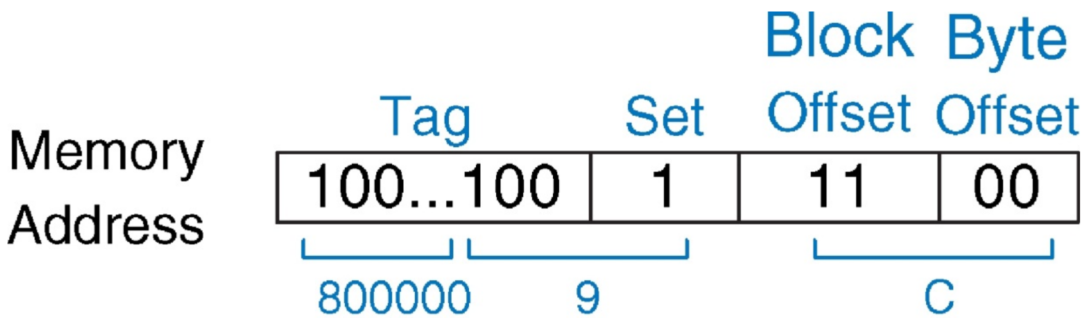
Expensive to build

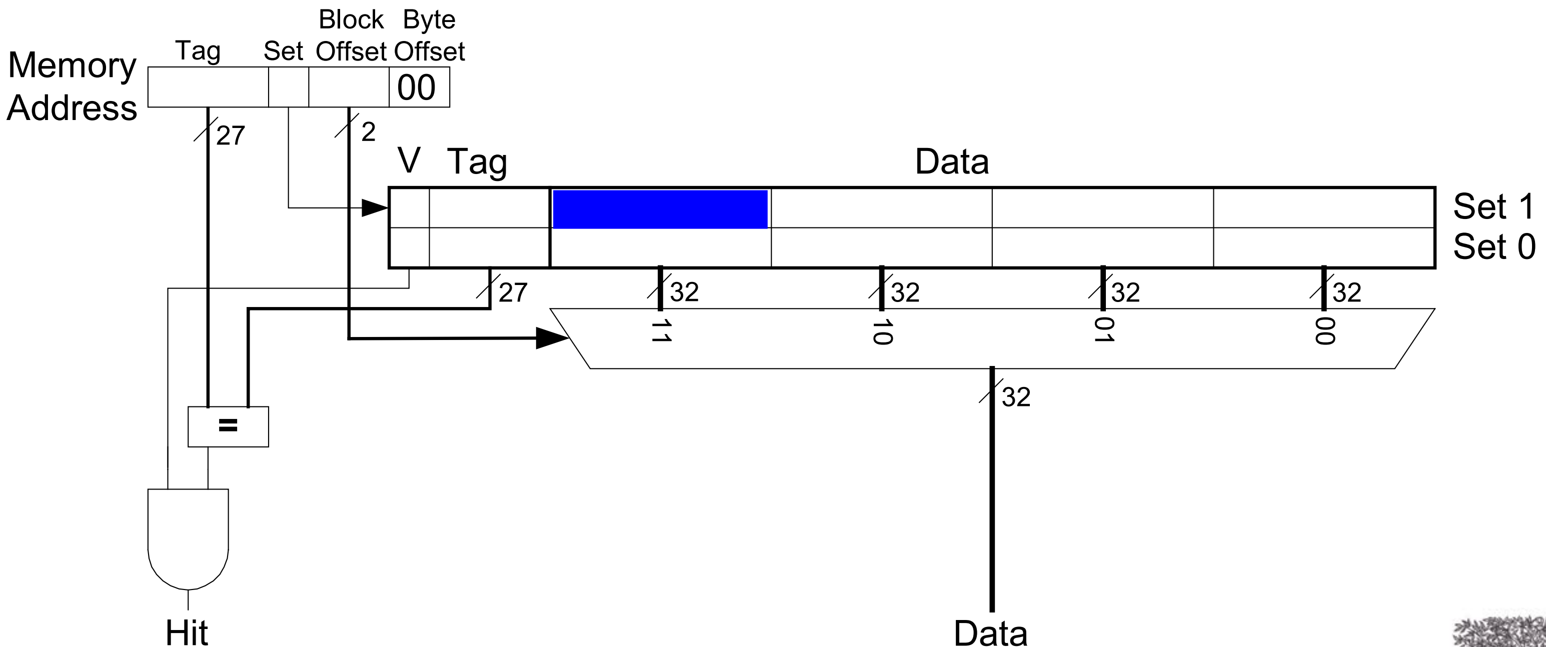
Spatial Locality?

- Increase block size:
  - Block size, ***b* = 4 words**
  - *C* = 8 words
  - Direct mapped (1 block per set)
  - Number of blocks, ***B* = 2** (*C*/*b* = 8/4 = 2)



Cache with Larger Block Size





Direct Mapped Cache Performance


```
       addi $t0, $0, 5
loop:  beq  $t0, $0, done
       lw   $t1, 0x4($0)
       lw   $t2, 0xC($0)
       lw   $t3, 0x8($0)
       addi $t0, $t0, -1
       j    loop
done:
```

Miss Rate = ?

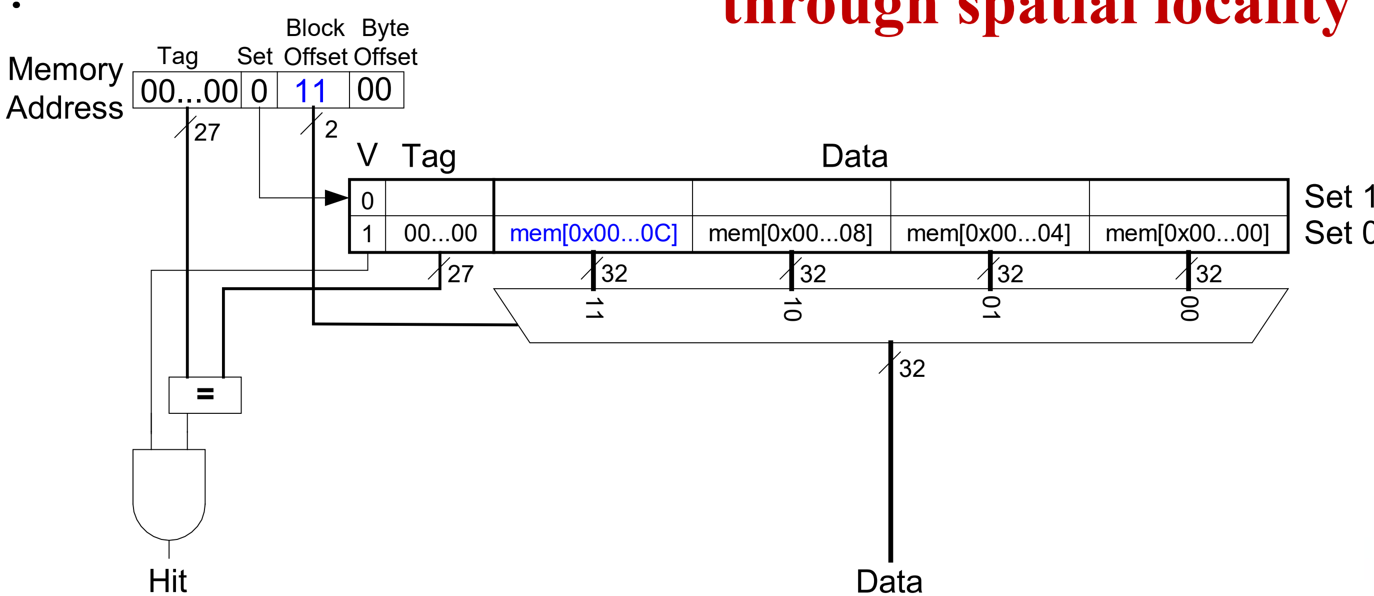
Direct Mapped Cache Performance

```
       addi $t0, $0, 5
loop:  beq  $t0, $0, done
       lw   $t1, 0x4($0)
       lw   $t2, 0xC($0)
       lw   $t3, 0x8($0)
       addi $t0, $t0, -1
       j    loop
done:
```

Miss Rate = 1/15

= 6.67%

Larger blocks reduce compulsory misses through spatial locality



Cache Organization Recap

- Capacity: C
- Block size: b
- Number of blocks in cache: $B = C/b$
- Number of blocks in a set: N
- Number of sets: $S = B/N$

Organization	Number of Ways (N)	Number of Sets ($S = B/N$)
Direct Mapped	1	B
N-Way Set Associative	$1 < N < B$	B / N
Fully Associative	B	1

MEMORY & I/O SYSTEMS

Capacity Misses

- Cache is too small to hold all data of interest at once
- If cache full: program accesses data X & evicts data Y
- ***Capacity miss*** when access Y again
- How to choose Y to minimize chance of needing it again?
- **Least recently used (LRU) replacement:** the least recently used block in a set evicted

Types of Misses

- **Compulsory:** first time data accessed
- **Capacity:** cache too small to hold all data of interest
- **Conflict:** data of interest maps to same location in cache

Miss penalty: time it takes to retrieve a block from lower level of hierarchy

LRU Replacement

```
# MIPS assembly

lw $t0, 0x04($0)
lw $t1, 0x24($0)
lw $t2, 0x54($0)
```

Way 1				Way 0			
V	U	Tag	Data	V	Tag	Data	
0	0			0			Set 3 (11)
0	0			0			Set 2 (10)
0	0			0			Set 1 (01)
0	0			0			Set 0 (00)

ELSEVIER

LRU Replacement

```
# MIPS assembly

lw $t0, 0x04($0)
lw $t1, 0x24($0)
lw $t2, 0x54($0)
```

(a)

Way 1				Way 0			
V	U	Tag	Data	V	Tag	Data	
0	0			0			Set 3 (11)
0	0			0			Set 2 (10)
1	0	00...010	mem[0x00...24]	1	00...000	mem[0x00...04]	Set 1 (01)
0	0			0			Set 0 (00)

(b)

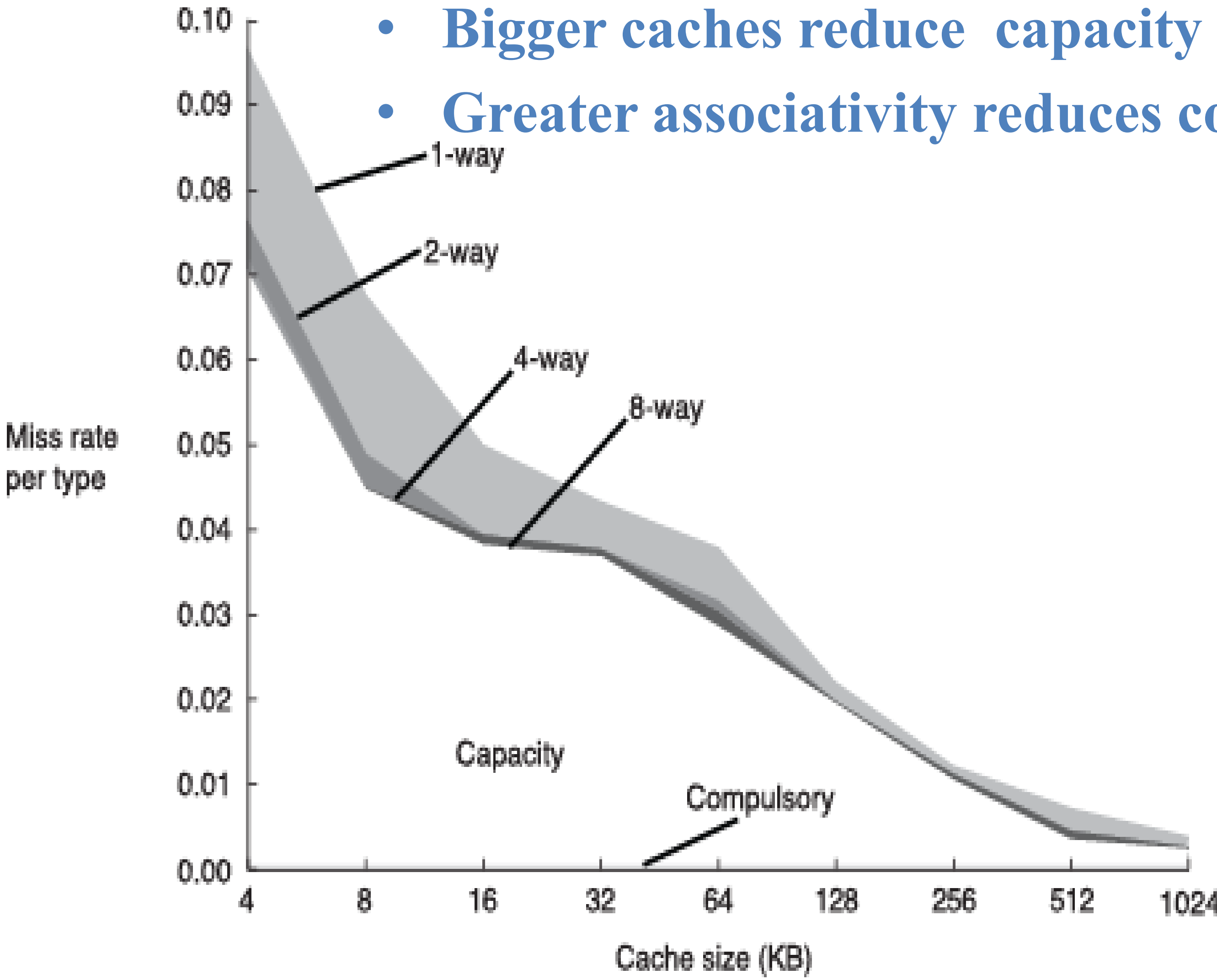
Way 1				Way 0			
V	U	Tag	Data	V	Tag	Data	
0	0			0			Set 3 (11)
0	0			0			Set 2 (10)
1	1	00...010	mem[0x00...24]	1	00...101	mem[0x00...54]	Set 1 (01)
0	0			0			Set 0 (00)

Cache Summary

- **What data is held in the cache?**
  - Recently used data (temporal locality)
  - Nearby data (spatial locality)
- **How is data found?**
  - Set is determined by address of data
  - Word within block also determined by address
  - In associative caches, data could be in one of several ways
- **What data is replaced?**
  - Least-recently used way in the set

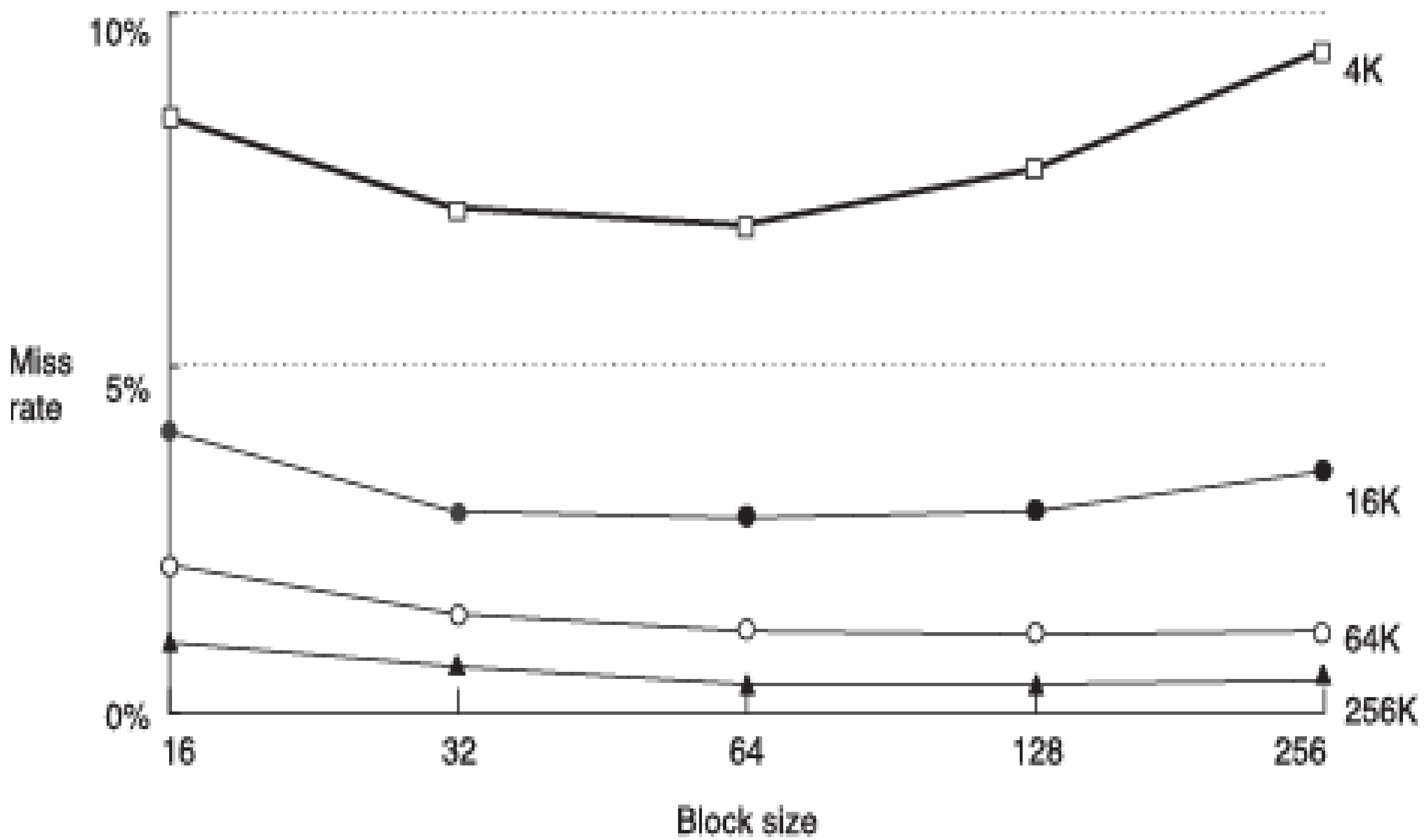
Miss Rate Trends

- **Bigger caches reduce capacity misses**
- **Greater associativity reduces conflict misses**



Adapted from Patterson & Hennessy, *Computer Architecture: A Quantitative Approach*, 2011

Miss Rate Trends

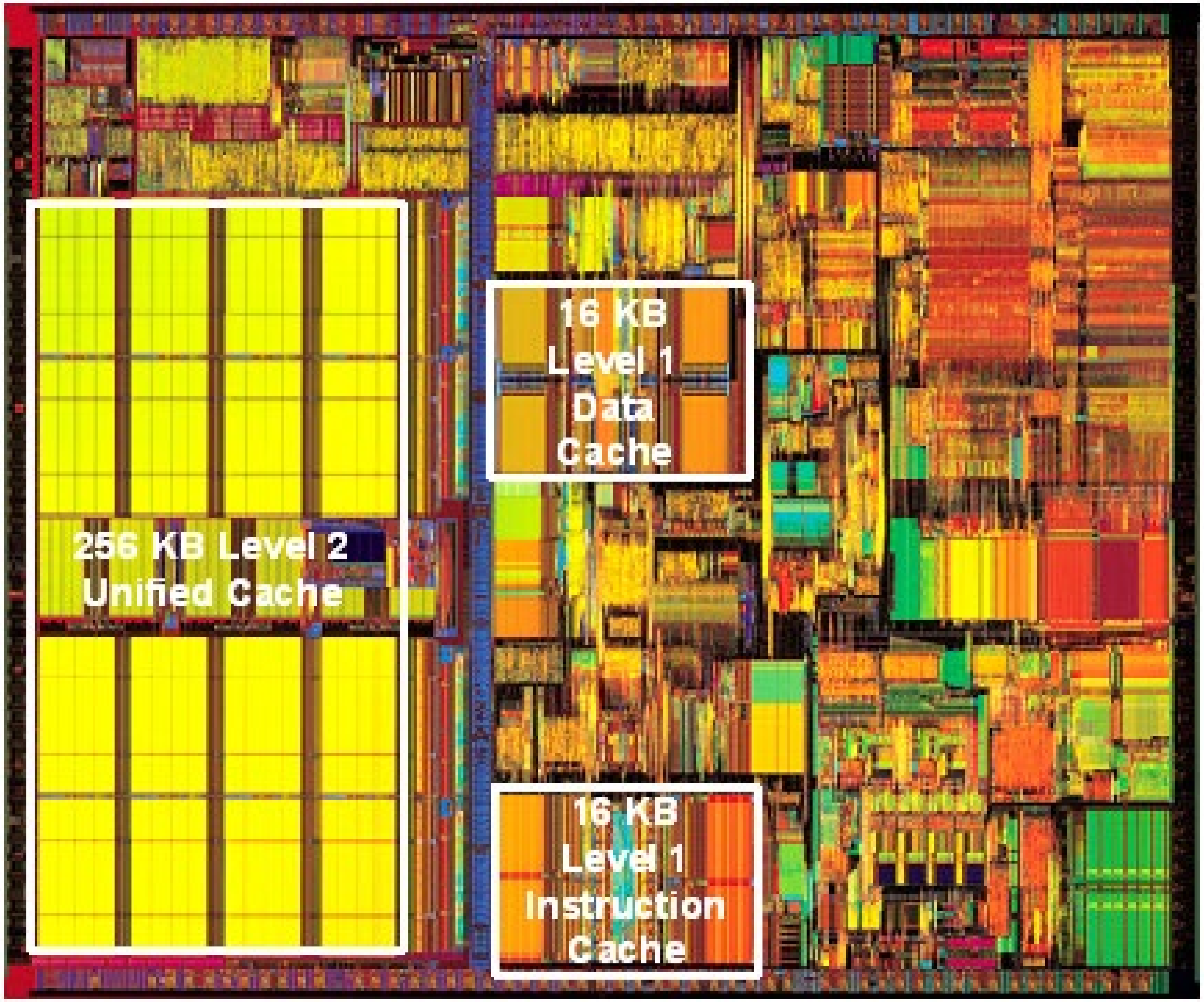


- Bigger blocks reduce compulsory misses
- Bigger blocks increase conflict misses

Multilevel Caches

- Larger caches have lower miss rates, longer access times
- Expand memory hierarchy to multiple levels of caches
- Level 1: small and fast (e.g. 16 KB, 1 cycle)
- Level 2: larger and slower (e.g. 256 KB, 2-6 cycles)
- Most modern PCs have L1, L2, and L3 cache

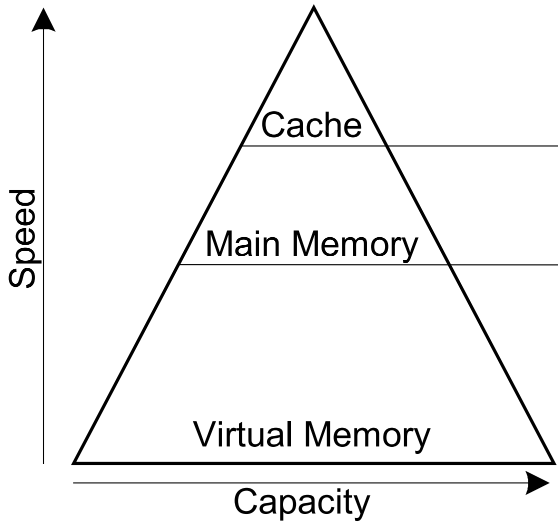
Intel Pentium III Die



Virtual Memory

- Gives the illusion of bigger memory
- Main memory (DRAM) acts as cache for hard disk

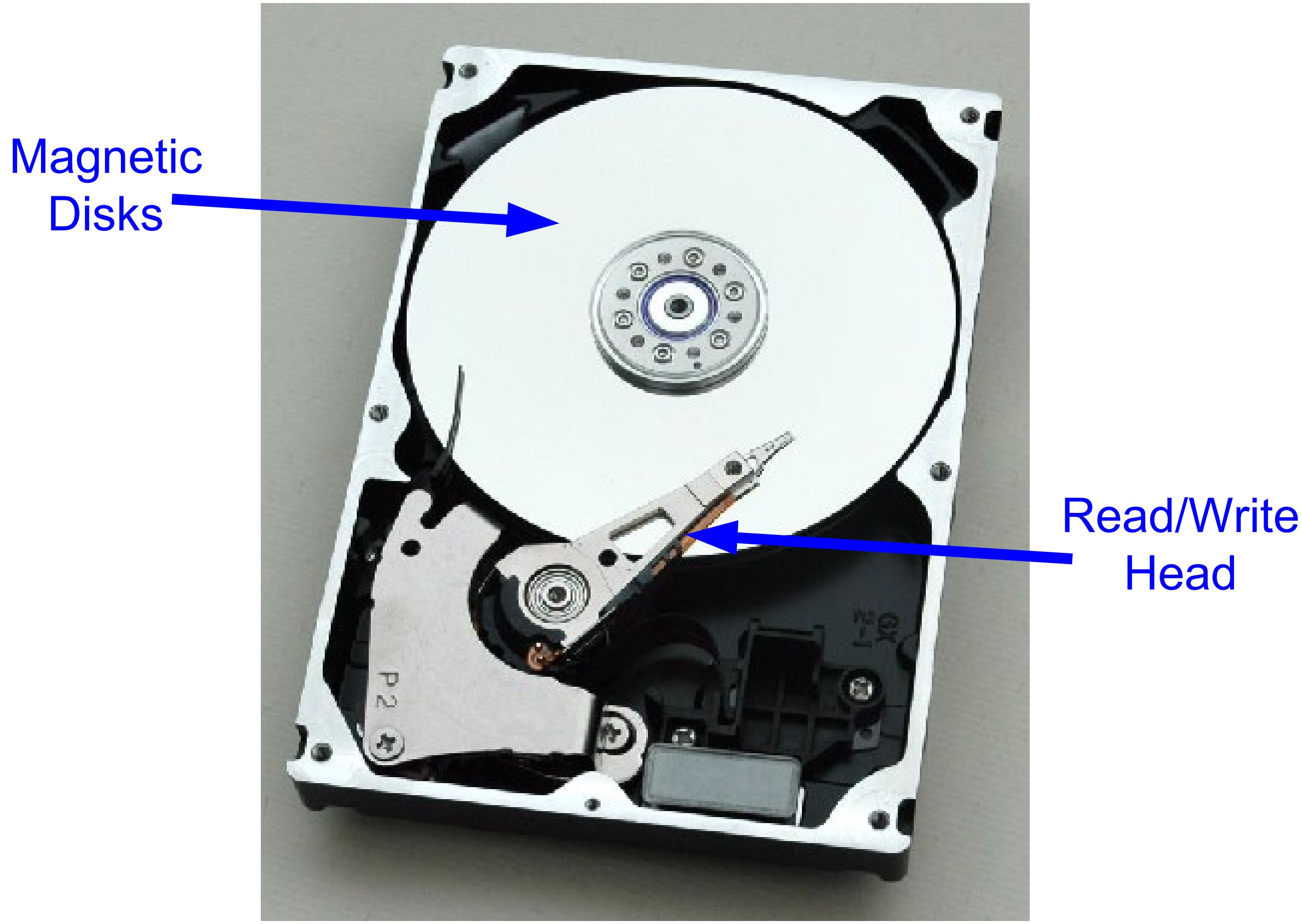
Memory Hierarchy



Technology	Price / GB	Access Time (ns)	Bandwidth (GB/s)
SRAM	\$10,000	1	25+
DRAM	\$10	10 - 50	10
SSD	\$1	100,000	0.5
HDD	\$0.1	10,000,000	0.1

- **Physical Memory:** DRAM (Main Memory)
- **Virtual Memory:** Hard drive
  - Slow, Large, Cheap

Hard Disk



Takes milliseconds to *seek* correct location on disk

Virtual Memory

- **Virtual addresses**
  - Programs use virtual addresses
  - Entire virtual address space stored on a hard drive
  - Subset of virtual address data in DRAM
  - CPU translates virtual addresses into ***physical addresses*** (DRAM addresses)
  - Data not in DRAM fetched from hard drive
- **Memory Protection**
  - Each program has own virtual to physical mapping
  - Two programs can use same virtual address for different data
  - Programs don't need to be aware others are running
  - One program (or virus) can't corrupt memory used by another

Cache/Virtual Memory Analogues

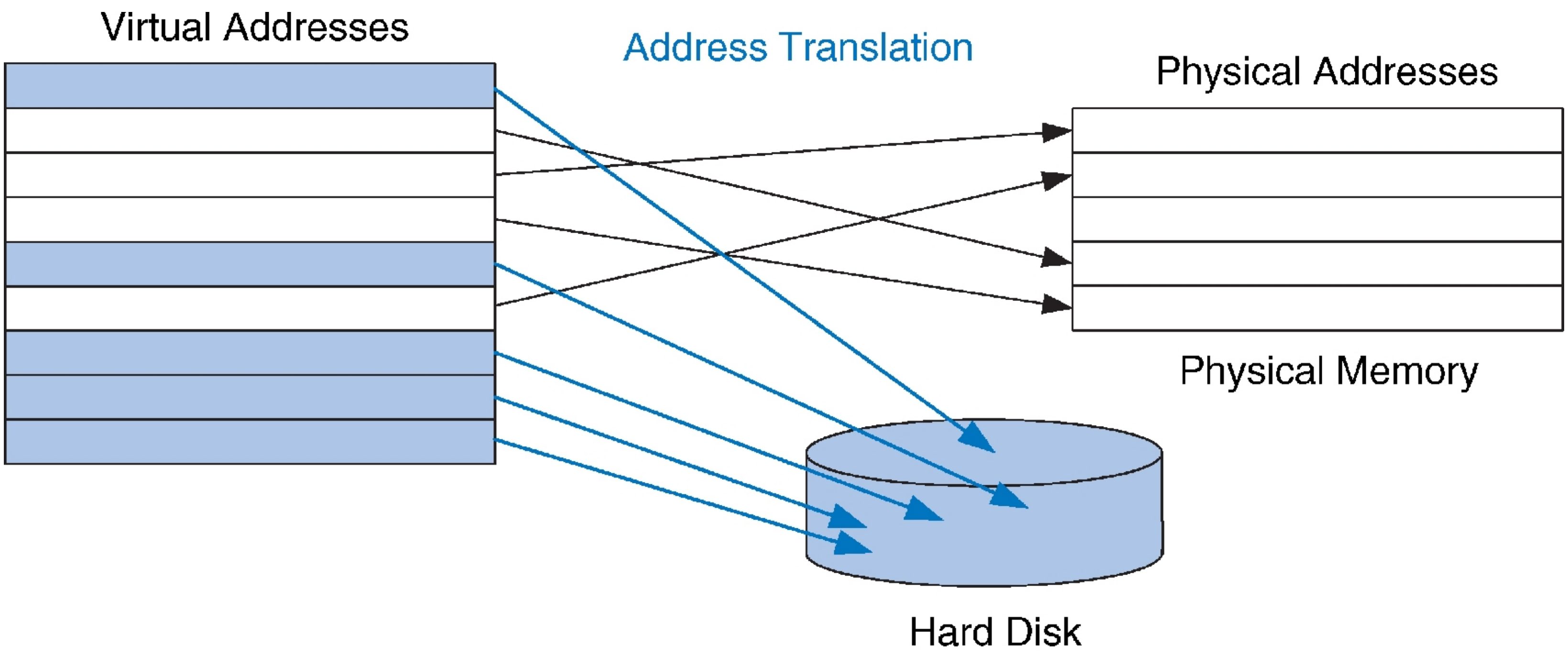
Cache	Virtual Memory
Block	Page
Block Size	Page Size
Block Offset	Page Offset
Miss	Page Fault
Tag	Virtual Page Number

Physical memory acts as cache for virtual memory

Virtual Memory Definitions

- **Page size:** amount of memory transferred from hard disk to DRAM at once
- **Address translation:** determining physical address from virtual address
- **Page table:** lookup table used to translate virtual addresses to physical addresses

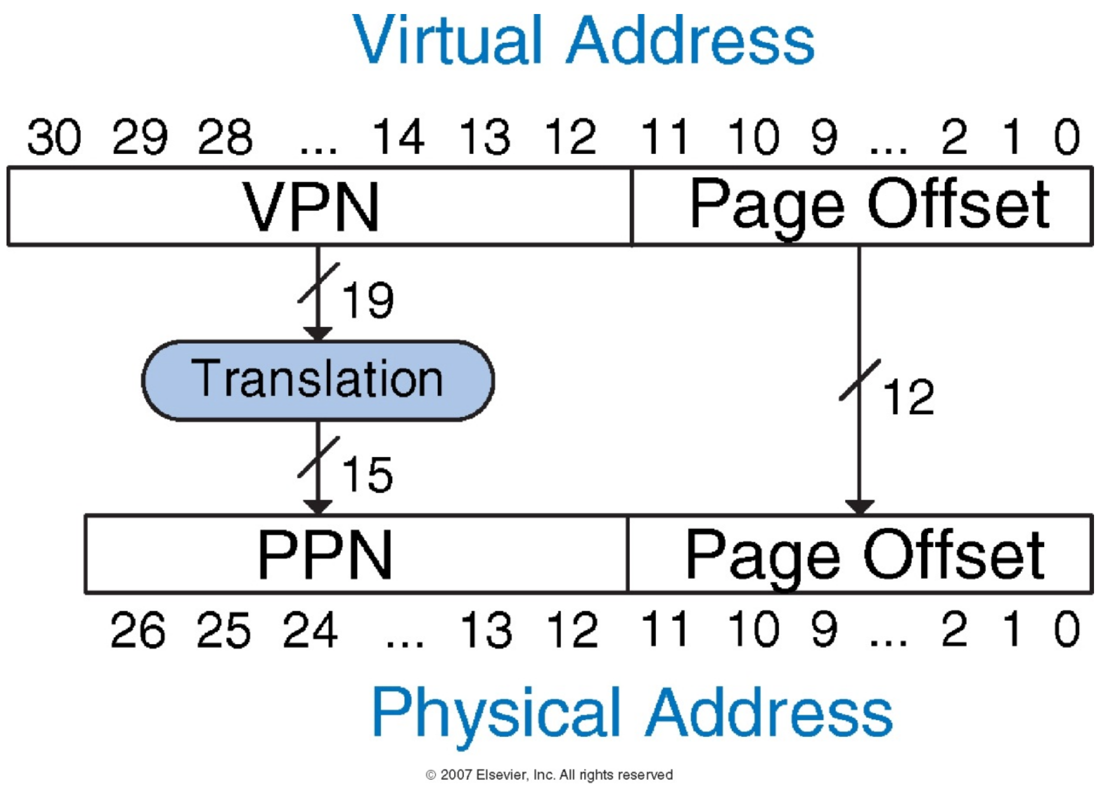
Virtual & Physical Addresses



Most accesses hit in physical memory

But programs have the large capacity of virtual memory

Address Translation



Virtual Memory Example

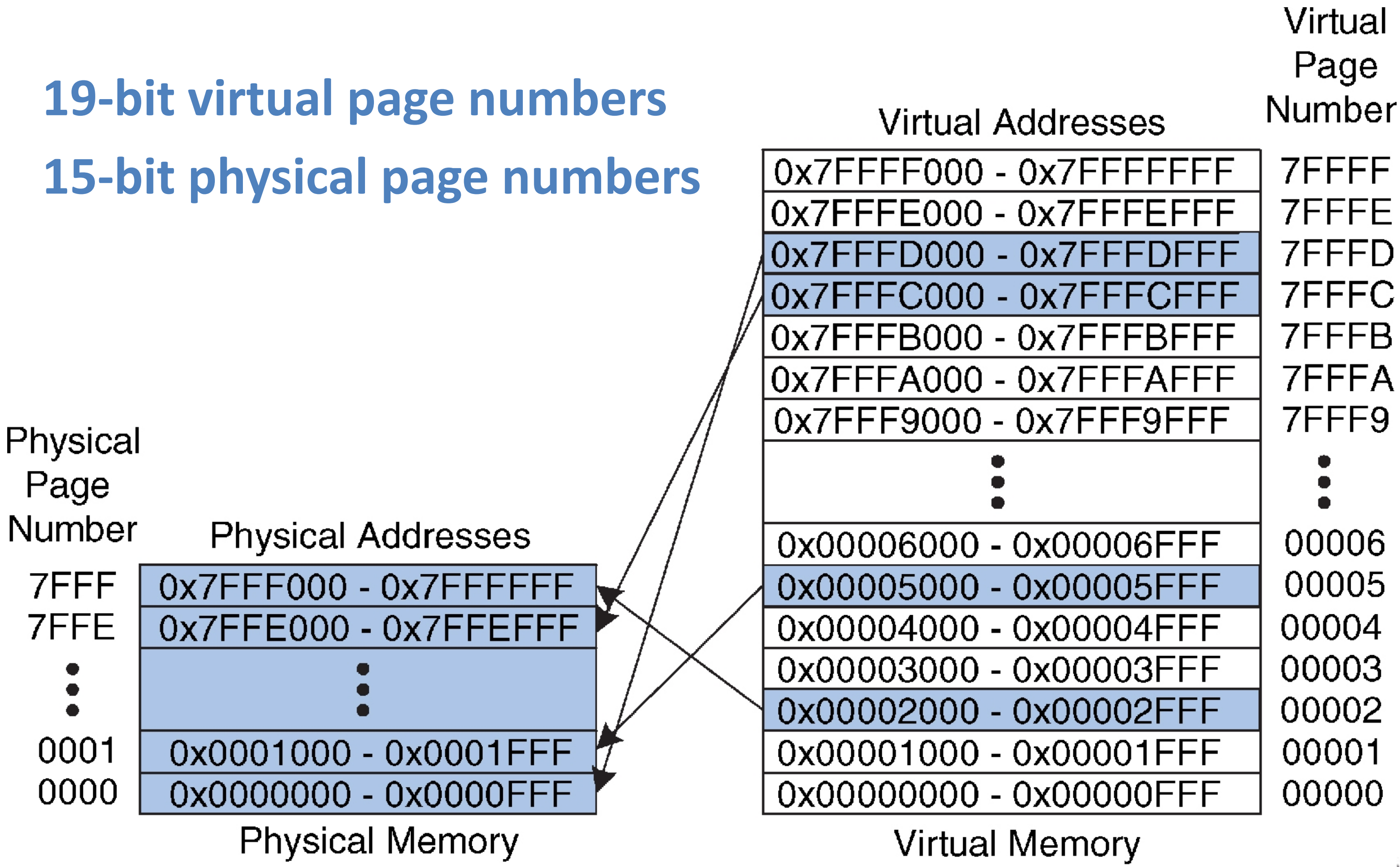
- **System:**
  - Virtual memory size: 2 GB = 2^{31} bytes
  - Physical memory size: 128 MB = 2^{27} bytes
  - Page size: 4 KB = 2^{12} bytes

Virtual Memory Example

- **System:**
  - Virtual memory size: 2 GB = 2^{31} bytes
  - Physical memory size: 128 MB = 2^{27} bytes
  - Page size: 4 KB = 2^{12} bytes
- **Organization:**
  - Virtual address: **31** bits
  - Physical address: **27** bits
  - Page offset: **12** bits
  - # Virtual pages = $2^{31}/2^{12}$ = $\mathbf{2^{19}}$ (VPN = 19 bits)
  - # Physical pages = $2^{27}/2^{12}$ = $\mathbf{2^{15}}$ (PPN = 15 bits)

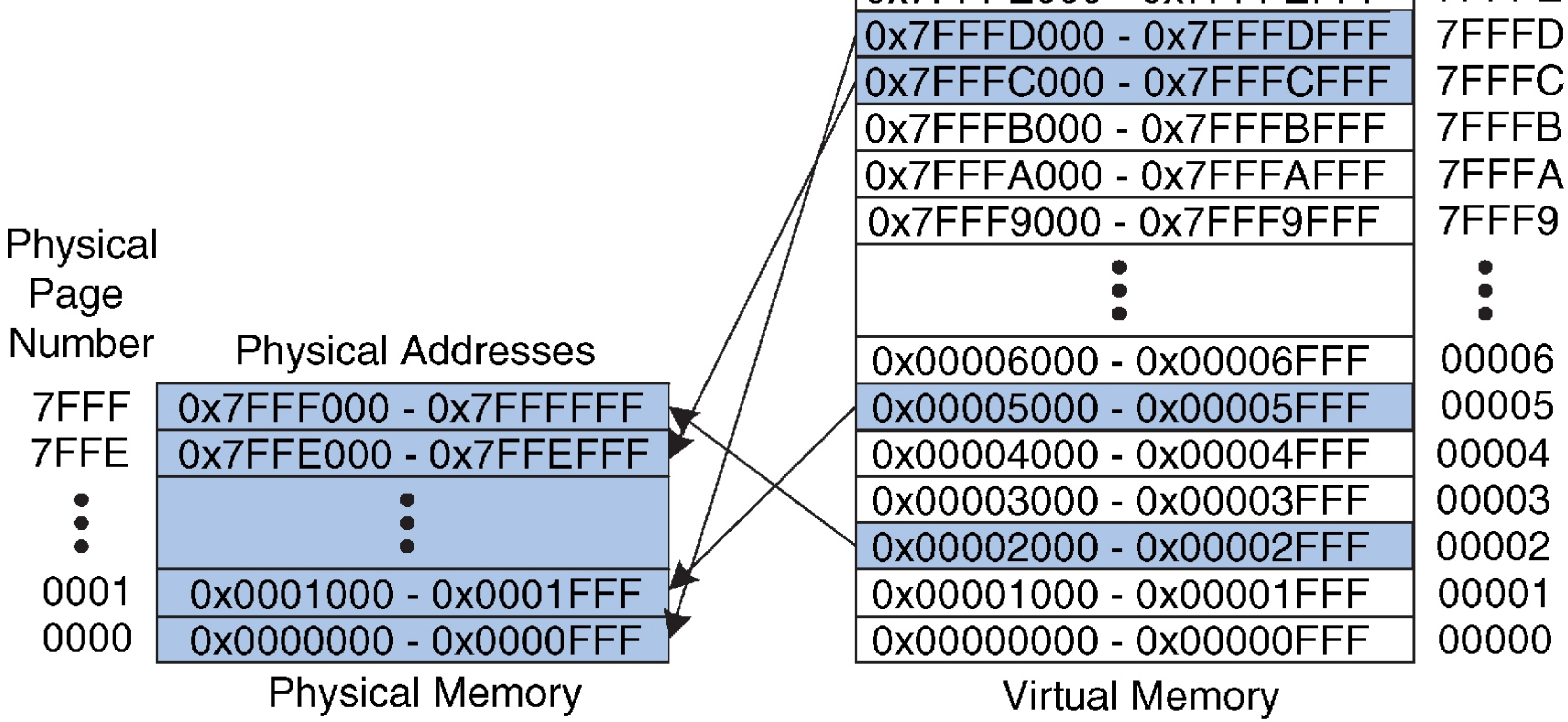
Virtual Memory Example

- **19-bit virtual page numbers**
- **15-bit physical page numbers**



Virtual Memory Example

What is the physical address of virtual address **0x247C**?



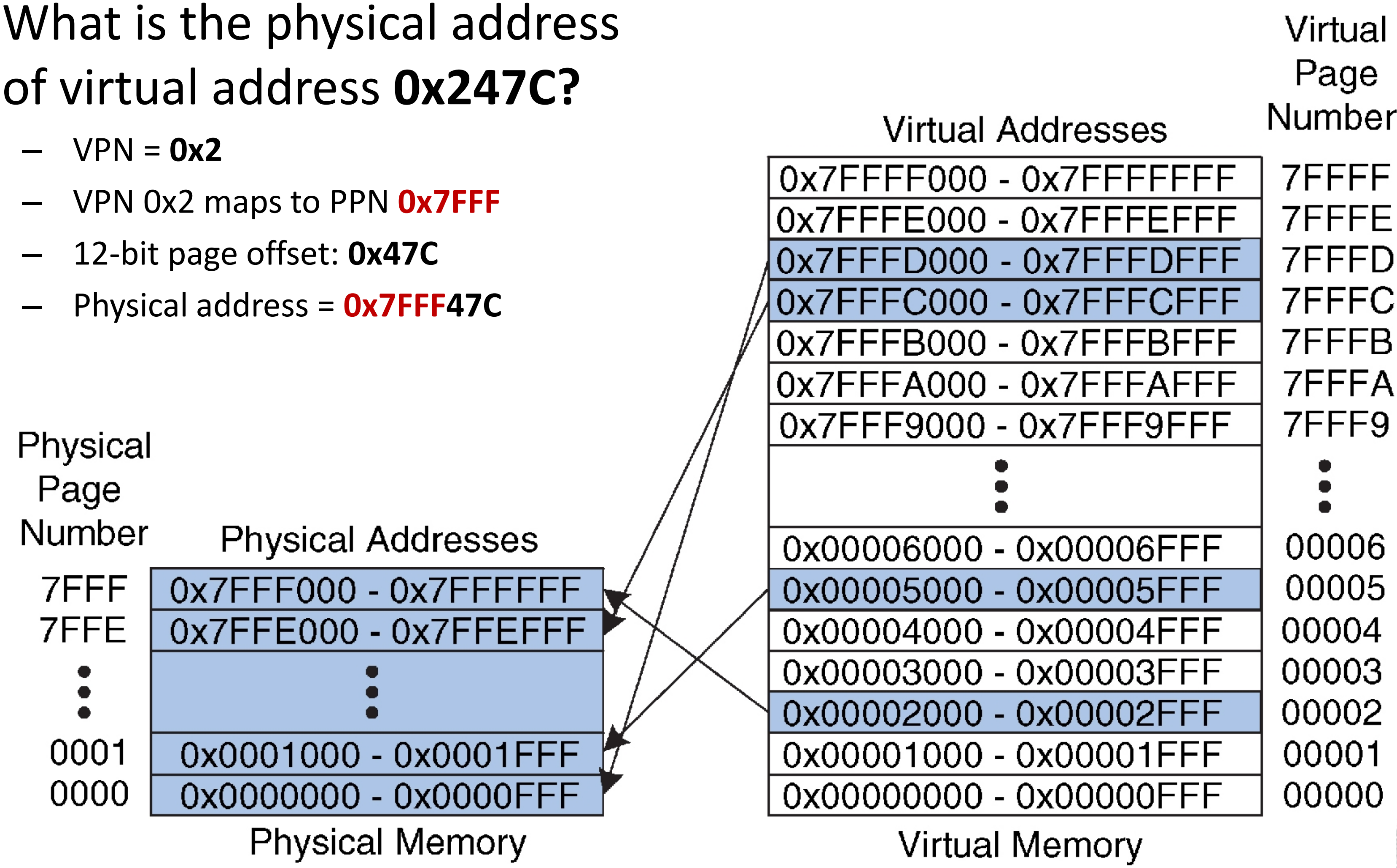
MEMORY & I/O SYSTEMS

ELSEVIER

Virtual Memory Example

What is the physical address of virtual address **0x247C?**

- VPN = **0x2**
- VPN 0x2 maps to PPN **0x7FFF**
- 12-bit page offset: **0x47C**
- Physical address = **0x7FFF47C**

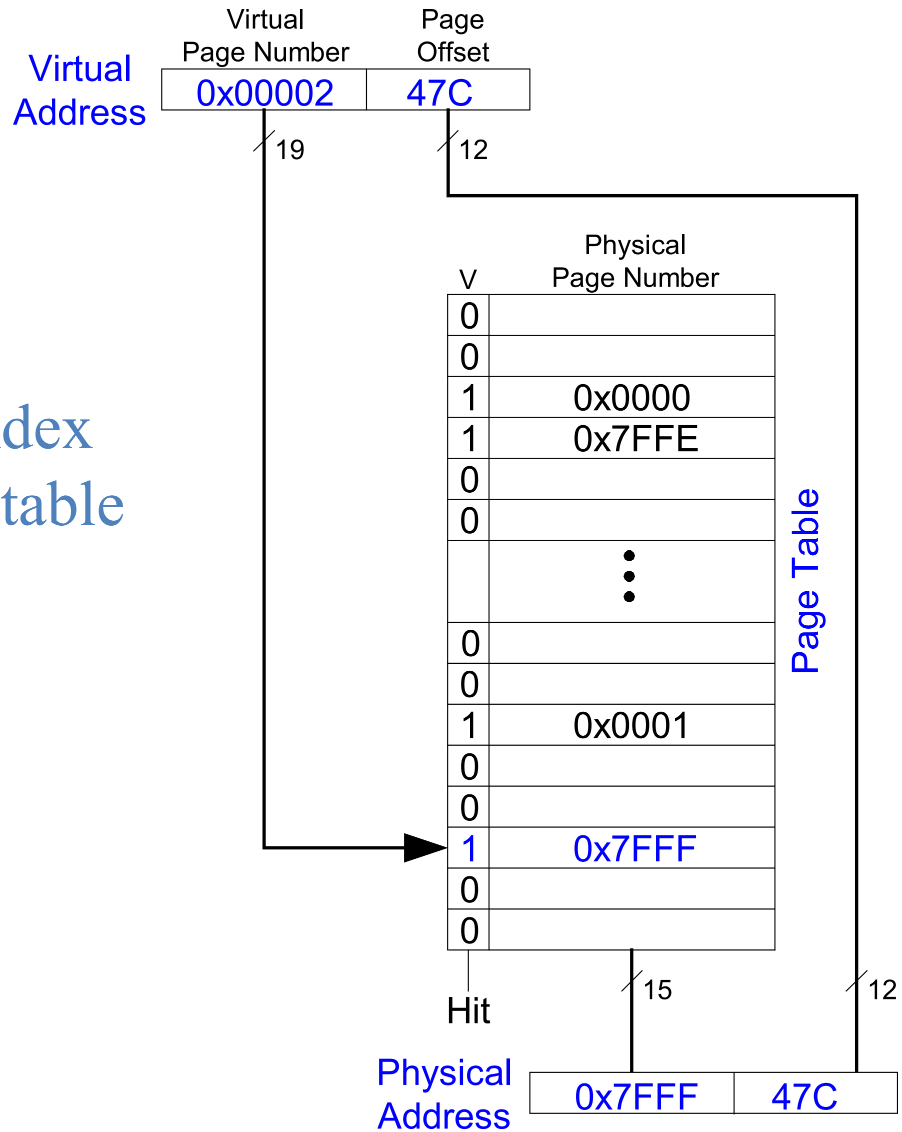


How to perform translation?

- **Page table**
  - Entry for each virtual page
  - Entry fields:
    - **Valid bit:** 1 if page in physical memory
    - **Physical page number:** where the page is located

Page Table Example

VPN is index into page table



MEMORY & I/O SYSTEMS

ELSEVIER

Page Table Example 1

What is the physical address of virtual address **0x5F20**?

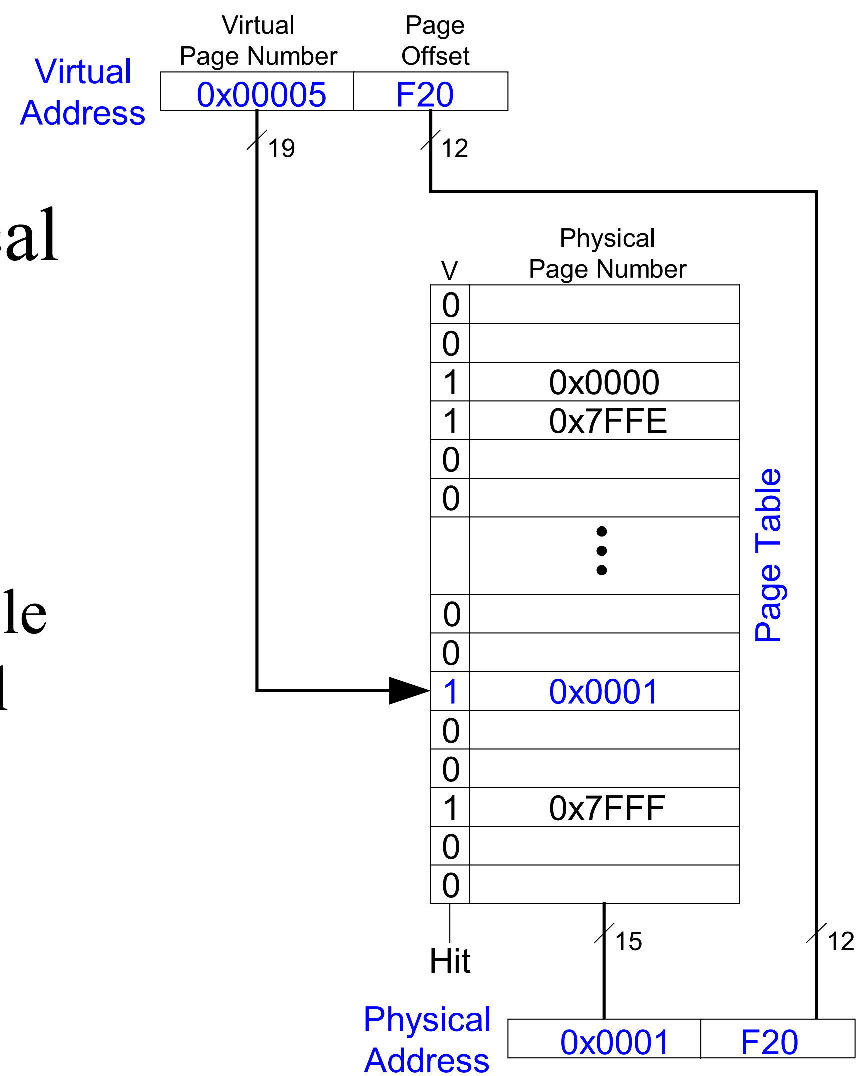
Page Table

V	Physical Page Number
0	
0	
1	0x0000
1	0x7FFE
0	
0	
	⋮
0	
0	
1	0x0001
0	
0	
1	0x7FFF
0	
0	

Page Table Example 1

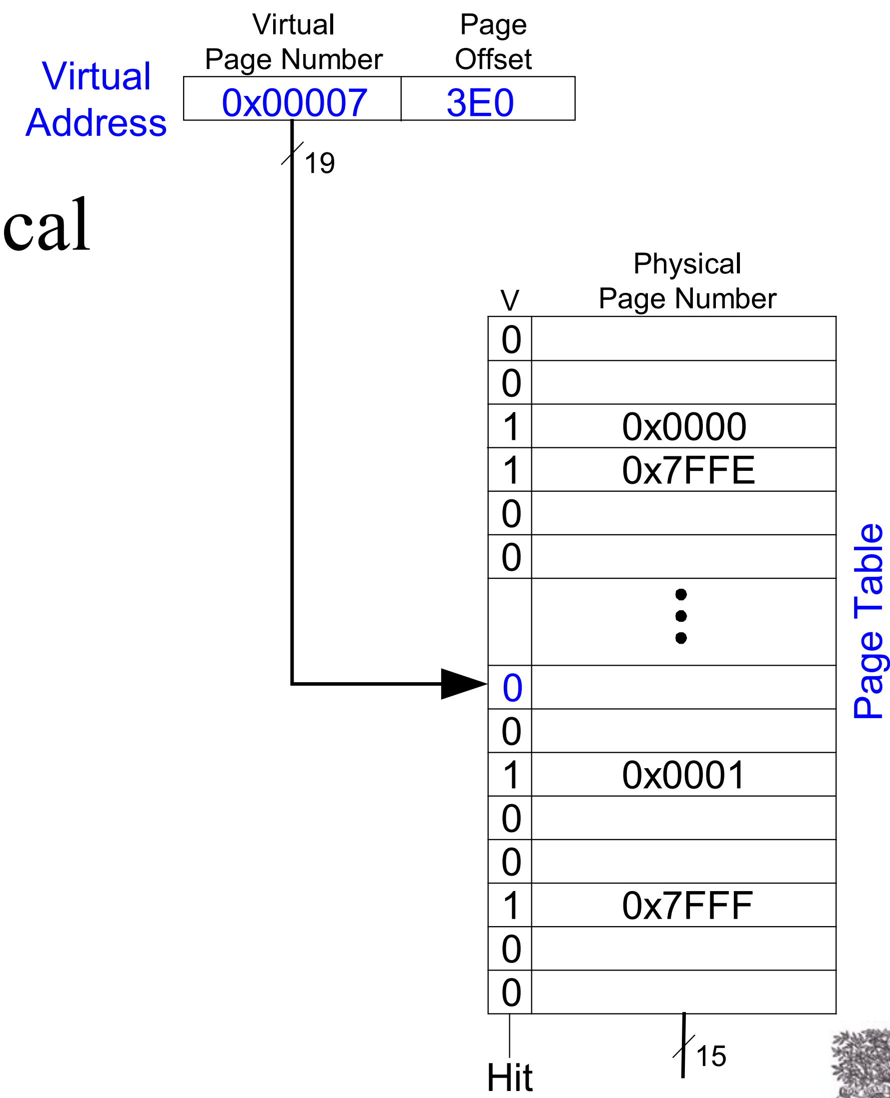
What is the physical address of virtual address **0x5F20**?

- VPN = **5**
- Entry 5 in page table VPN 5 => physical page **1**
- Physical address: **0x1F20**



Page Table Example 2

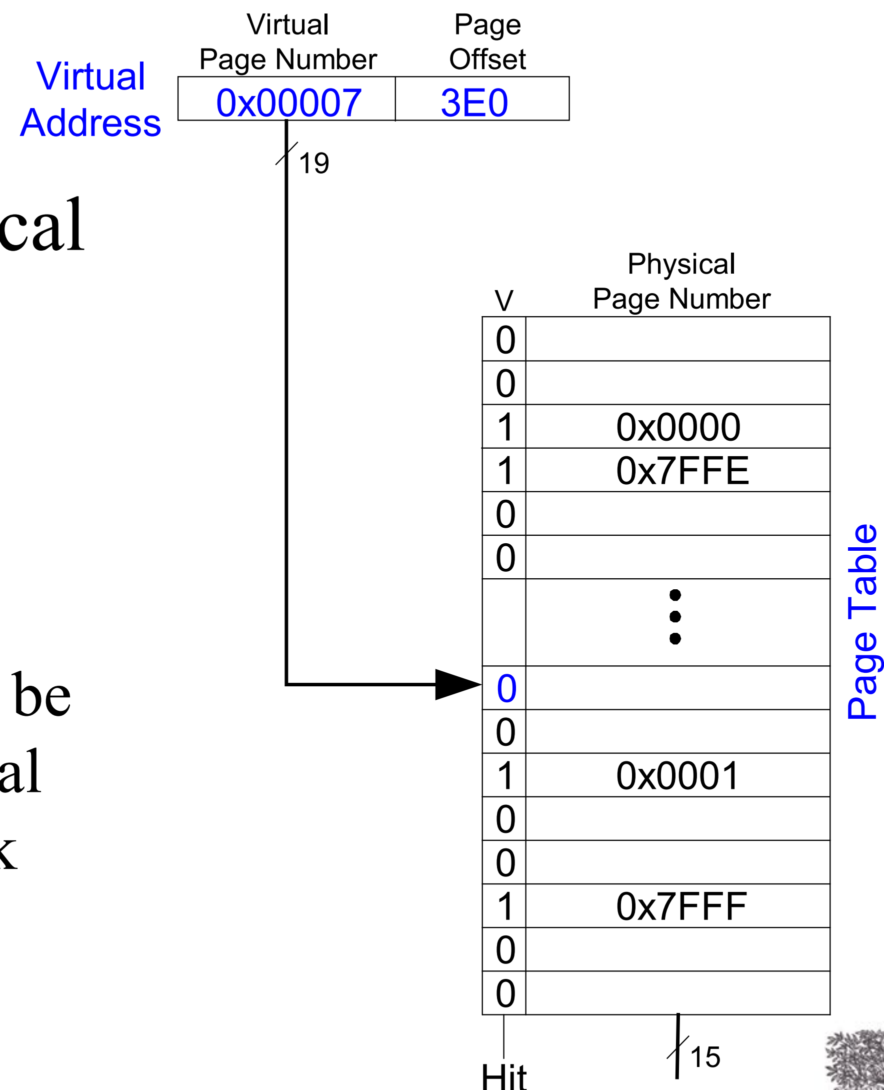
What is the physical address of virtual address **0x73E0**?



Page Table Example 2

What is the physical address of virtual address **0x73E0**?

- VPN = **7**
- Entry 7 is invalid
- Virtual page must be ***paged*** into physical memory from disk



Page Table Challenges

- **Page table is large**
  - usually located in physical memory
- Load/store requires 2 main memory accesses:
  - one for translation (page table read)
  - one to access data (after translation)
- Cuts memory performance in half
  - ***Unless we get clever…***

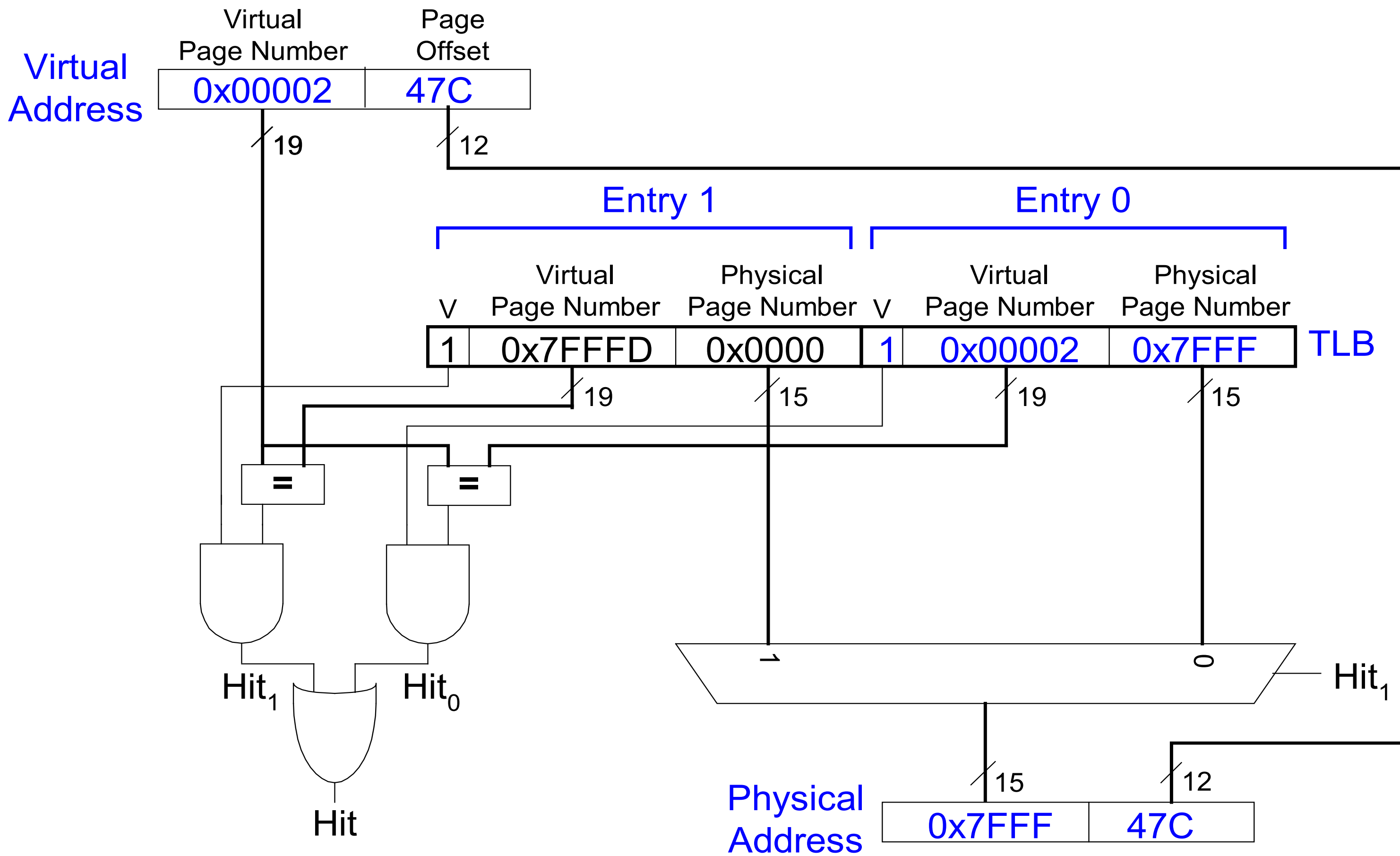
Translation Lookaside Buffer (TLB)

- Small cache of most recent translations
- Reduces # of memory accesses for *most* loads/stores from 2 to 1

TLB

- Page table accesses: high temporal locality
  - Large page size, so consecutive loads/stores likely to access same page
- TLB
  - Small: accessed in < 1 cycle
  - Typically 16 - 512 entries
  - Fully associative
  - > 99 % hit rates typical
  - Reduces # of memory accesses for most loads/stores from 2 to 1

Example 2-Entry TLB



ELSEVIER

Memory Protection

- Multiple processes (programs) run at once
- Each process has its own page table
- Each process can use entire virtual address space
- A process can only access physical pages mapped in its own page table

Virtual Memory Summary

- Virtual memory increases **capacity**
- A subset of virtual pages in physical memory
- **Page table** maps virtual pages to physical pages – address translation
- A **TLB** speeds up address translation
- Different page tables for different programs provides **memory protection**

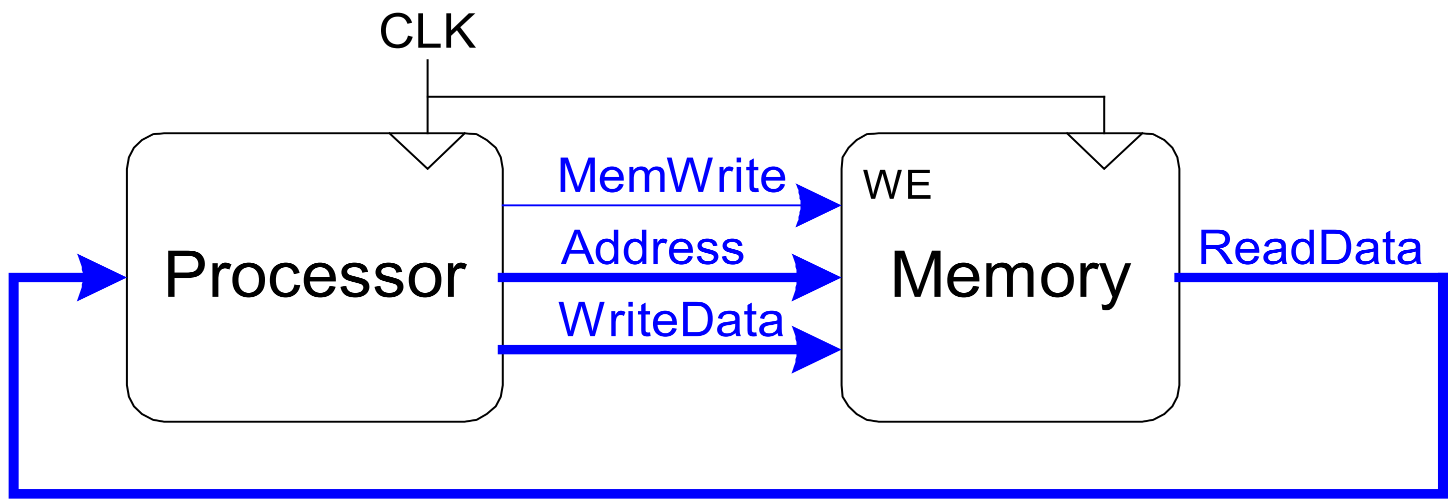
Memory-Mapped I/O

- Processor accesses I/O devices just like memory (like keyboards, monitors, printers)
- Each I/O device assigned one or more address
- When that address is detected, data read/written to I/O device instead of memory
- A portion of the address space dedicated to I/O devices

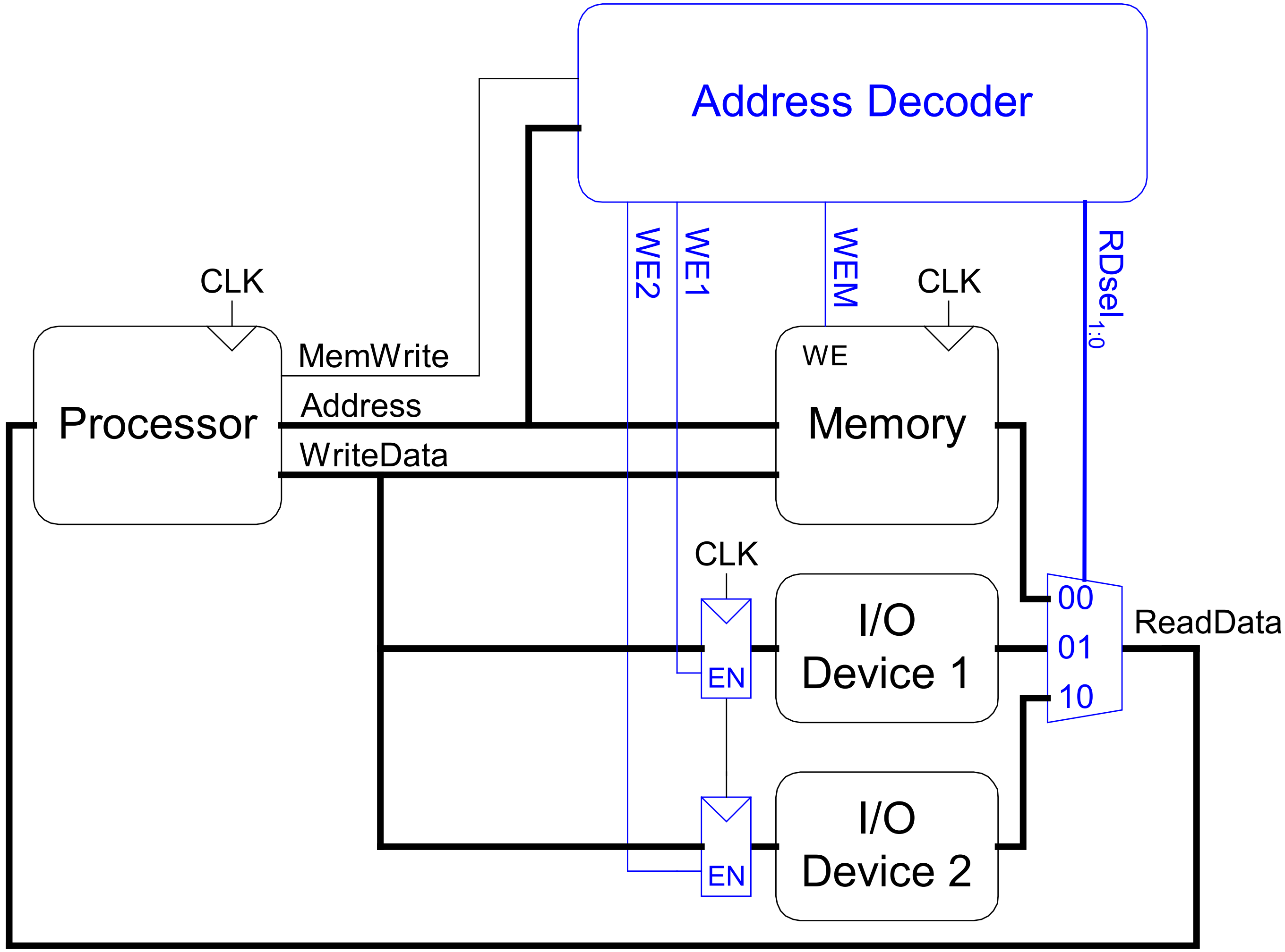
Memory-Mapped I/O Hardware

- **Address Decoder:**
  - Looks at address to determine which device/memory communicates with the processor
- **I/O Registers:**
  - Hold values written to the I/O devices
- **ReadData Multiplexer:**
  - Selects between memory and I/O devices as source of data sent to the processor

The Memory Interface



Memory-Mapped I/O Hardware



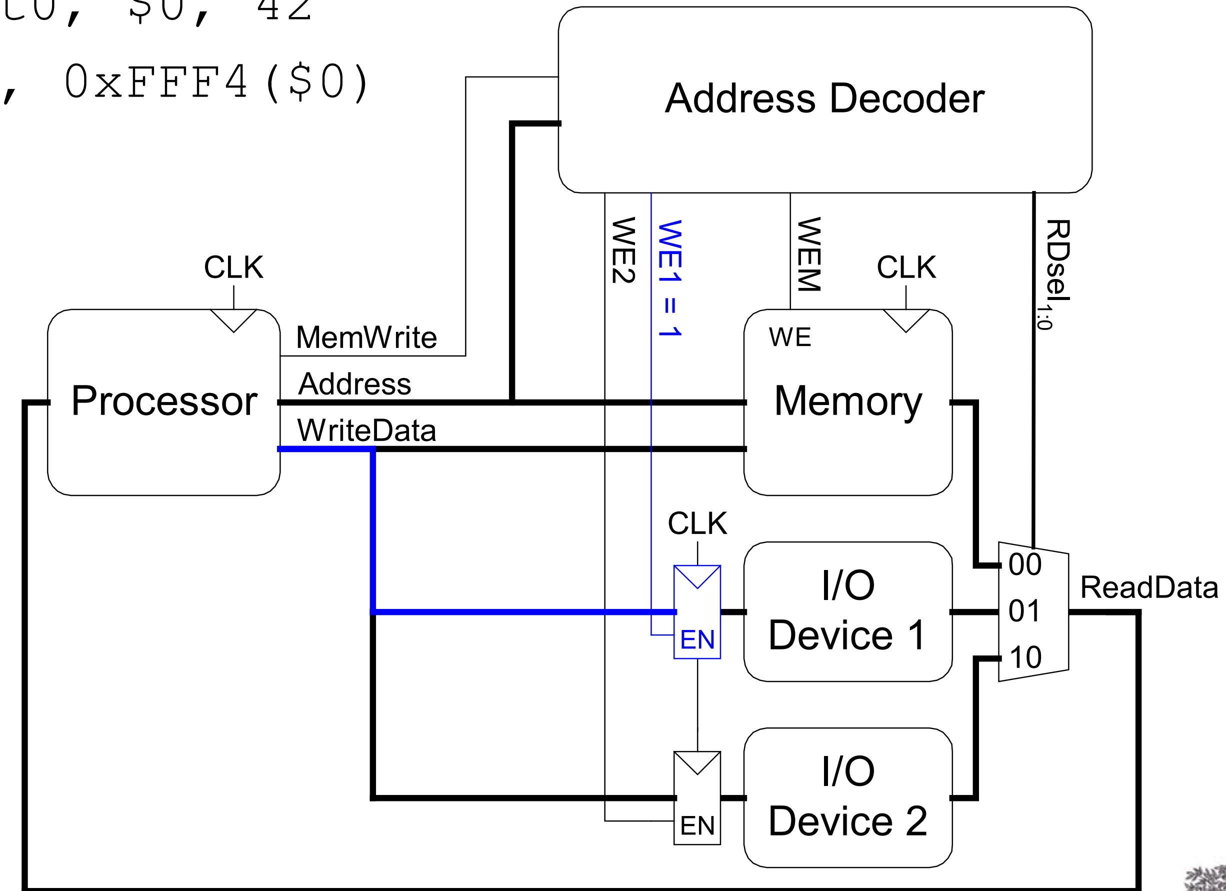
Memory-Mapped I/O Code

- Suppose I/O Device 1 is assigned the address 0xFFFFFFF4
  - **Write the value 42 to I/O Device 1**
  - **Read value from I/O Device 1 and place in `$t3`**

Memory-Mapped I/O Code

- **Write the value 42 to I/O Device 1 (0xFFFFFFF4)**

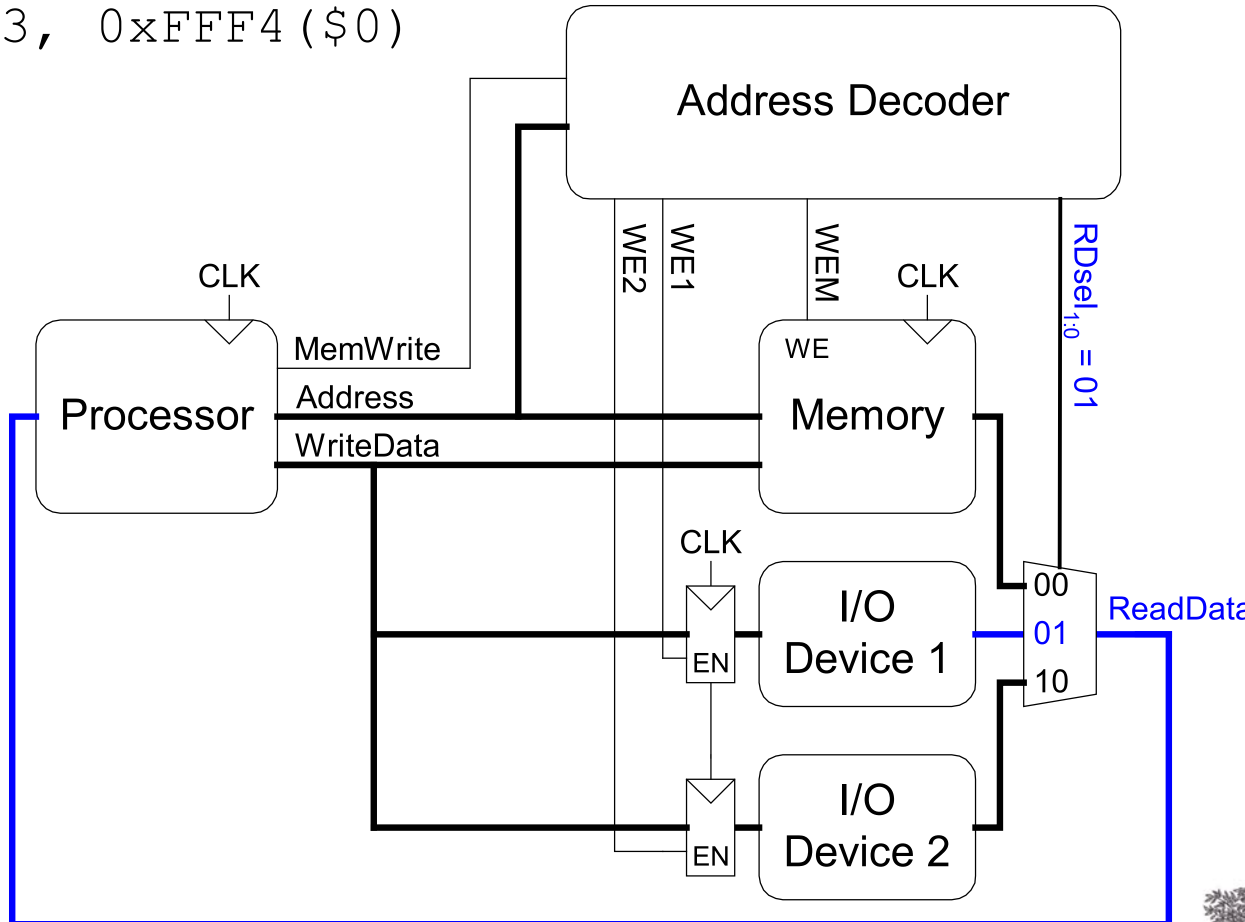
```
addi $t0, $0, 42
sw $t0, 0xFFF4($0)
```



Memory-Mapped I/O Code

- **Read the value from I/O Device 1 and place in `$t3`**

```
lw $t3, 0xFFF4($0)
```



Input/Output (I/O) Systems

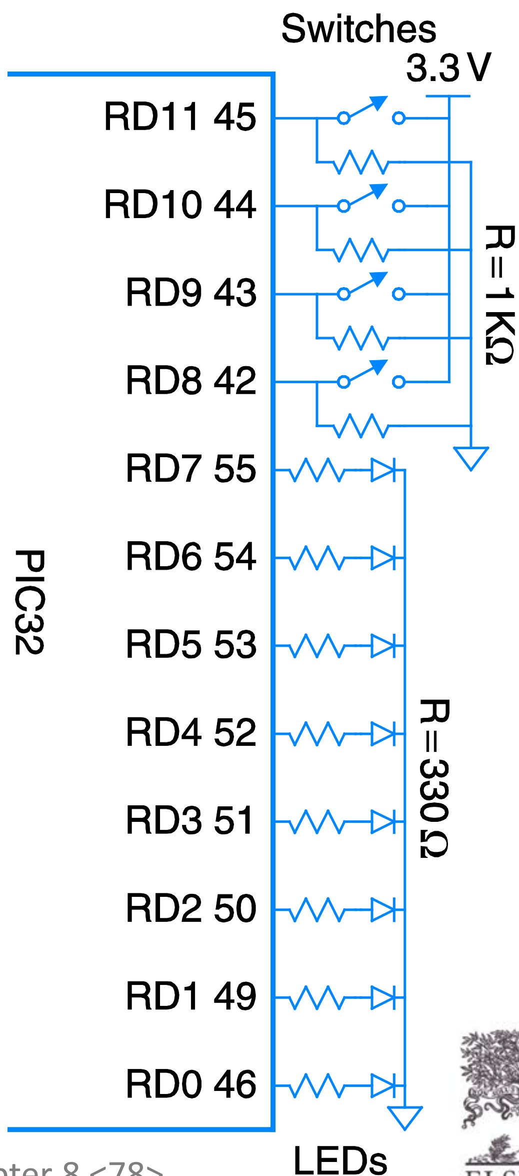
- Embedded I/O Systems
  - Toasters, LEDs, etc.
- PC I/O Systems

Embedded I/O Systems

- Example microcontroller: PIC32
  - microcontroller
  - 32-bit MIPS processor
  - low-level peripherals include:
    - serial ports
    - timers
    - A/D converters

Digital I/O

```
// C Code
#include <p3xxxx.h>

int main(void) {
  int switches;
  TRISD = 0xFF00;        // RD[7:0] outputs
                         // RD[11:8] inputs
  while (1) {
    // read & mask switches, RD[11:8]
    switches = (PORTD >> 8) & 0xF;
    PORTD = switches;  // display on LEDs
  }
}
```

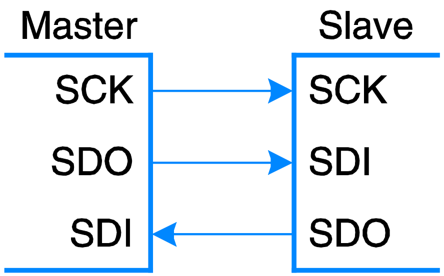


Serial I/O

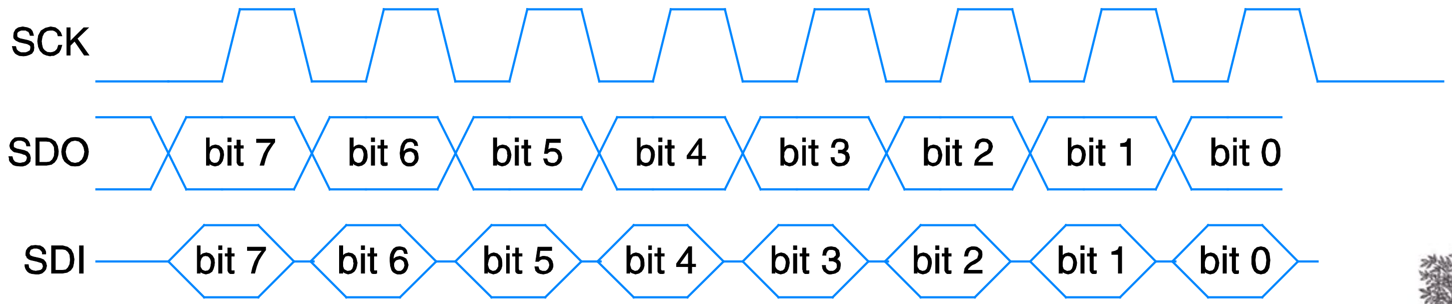
- Example serial protocols
  - **SPI:** Serial Peripheral Interface
  - **UART:** Universal Asynchronous Receiver/Transmitter
  - Also: I^2C, USB, Ethernet, etc.

SPI: Serial Peripheral Interface

- Master initiates communication to slave by sending pulses on SCK
- Master sends SDO (Serial Data Out) to slave, msb first
- Slave may send data (SDI) to master, msb first



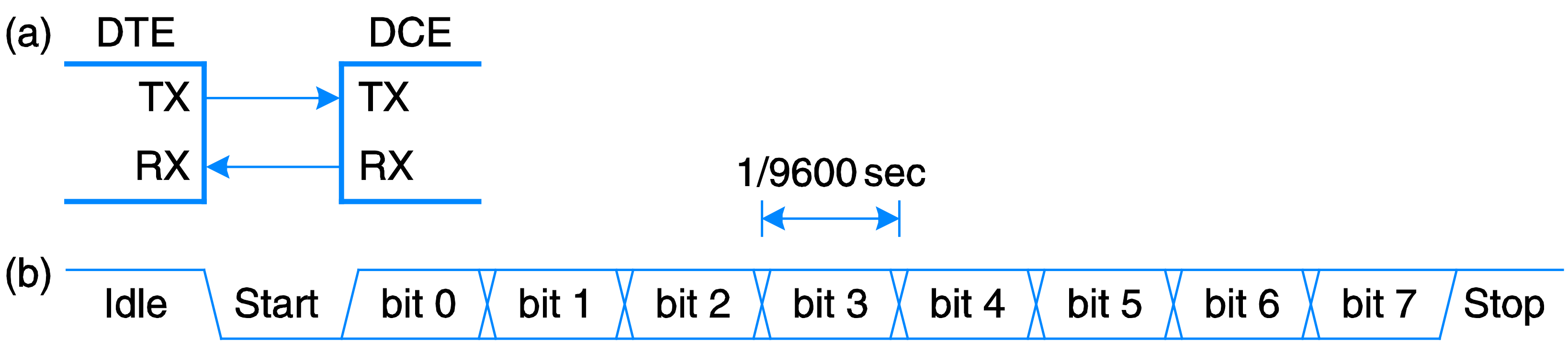
(a)



(b)

UART: Universal Asynchronous Rx/Tx

- Configuration:
  - start bit (0), 7-8 data bits, parity bit (optional), 1+ stop bits (1)
  - data rate: 300, 1200, 2400, 9600, …115200 baud
- Line idles HIGH (1)
- Common configuration:
  - 8 data bits, no parity, 1 stop bit, 9600 baud



Timers

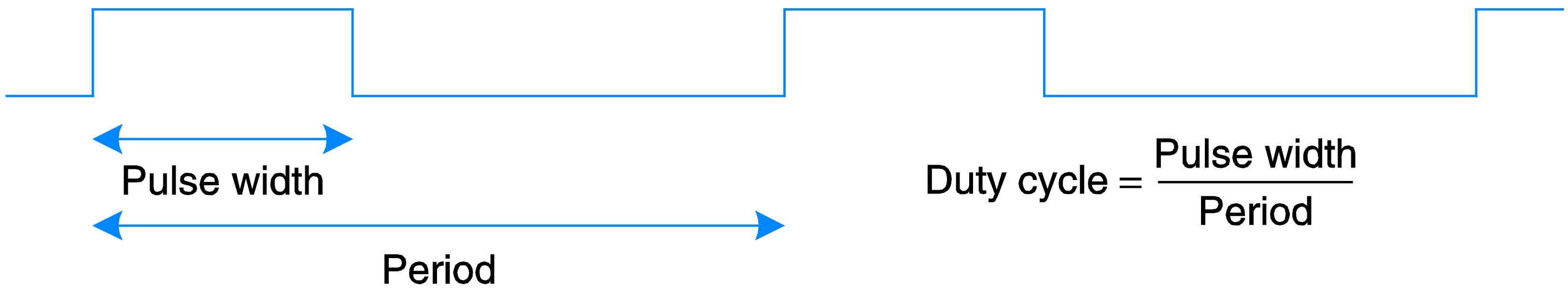
```
// Create specified ms/us of delay using built-in timer
#include <P32xxxx.h>

void delaymicros(int micros) {
  if (micros > 1000) {          // avoid timer overflow
    delaymicros(1000);
    delaymicros(micros-1000);
  }
  else if (micros > 6){
    TMR1 = 0;                   // reset timer to 0
    T1CONbits.ON = 1;           // turn timer on
    PR1 = (micros-6)*20;        // 20 clocks per microsecond
                                // Function has overhead of ~6 us
    IFS0bits.T1IF = 0;          // clear overflow flag
    while (!IFS0bits.T1IF);     // wait until overflow flag set
  }
}

void delaymillis(int millis) {
  while (millis--) delaymicros(1000); // repeatedly delay 1 ms
}                                     // until done
```

Analog I/O

- Needed to interface with outside world
- **Analog input:** Analog-to-digital (A/D) conversion
  - Often included in microcontroller
  - N-bit: converts analog input from V_{ref-}-V_{ref+} to 0-2^{N-1}
- **Analog output:**
  - Digital-to-analog (D/A) conversion
    - Typically need external chip (e.g., AD558 or LTC1257)
    - N-bit: converts digital signal from 0-2^{N-1} to V_{ref-}-V_{ref+}
  - Pulse-width modulation

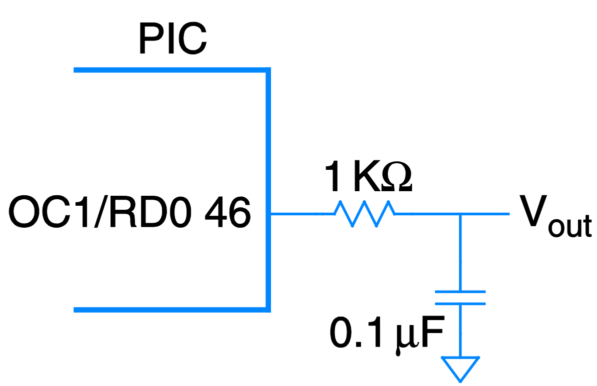
Pulse-Width Modulation (PWM)

- Average value proportional to duty cycle



$$\text{Duty cycle} = \frac{\text{Pulse width}}{\text{Period}}$$

- Add high-pass filter on output to deliver average value



Other Microcontroller Peripherals

- Examples
  - Character LCD
  - VGA monitor
  - Bluetooth wireless
  - Motors

Personal Computer (PC) I/O Systems

- USB: Universal Serial Bus
  - USB 1.0 released in 1996
  - standardized cables/software for peripherals
- PCI/PCIe: Peripheral Component Interconnect/PCI Express
  - developed by Intel, widespread around 1994
  - 32-bit parallel bus
  - used for expansion cards (i.e., sound cards, video cards, etc.)
- DDR: double-data rate memory

Personal Computer (PC) I/O Systems

- TCP/IP: Transmission Control Protocol and Internet Protocol
  - physical connection: Ethernet cable or Wi-Fi
- SATA: hard drive interface
- Input/Output (sensors, actuators, microcontrollers, etc.)
  - Data Acquisition Systems (DAQs)
  - USB Links