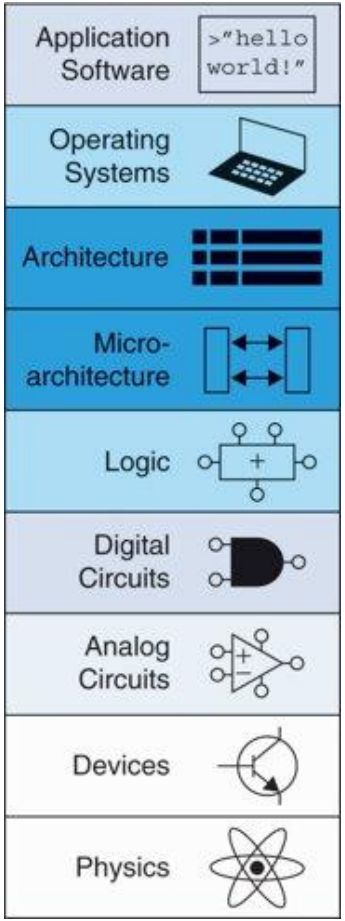
CPE 221: Computer Organization

17 Memory Systems

rahul.bhadani@uah.edu

Memory Systems

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



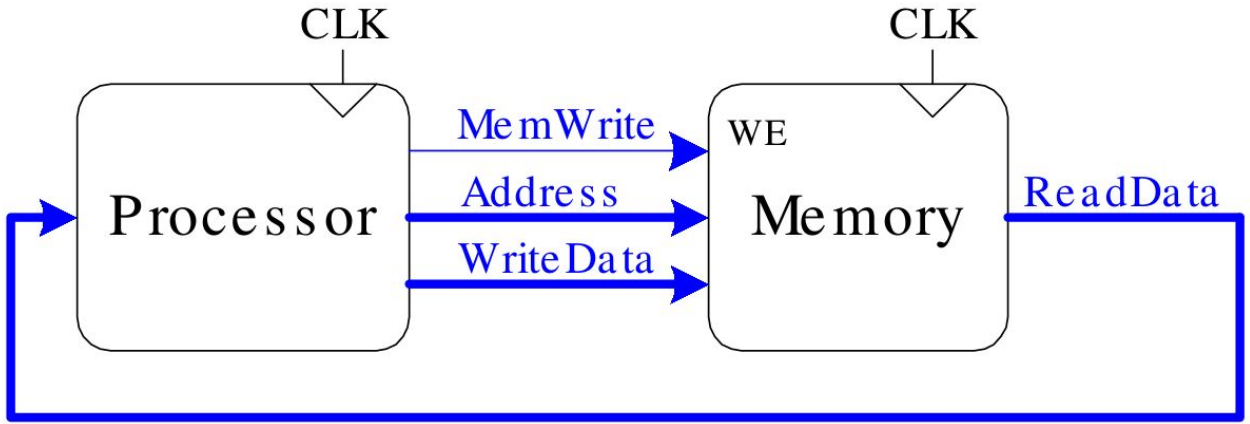
Introduction

Computer performance depends on:

- Processor performance
- Memory system performance

The processor sends an address over the Address bus to the memory system. For a read, MemWrite is 0 and the memory returns the data on the ReadData bus. For a write, MemWrite is 1 and the processor sends data to memory on the WriteData bus.

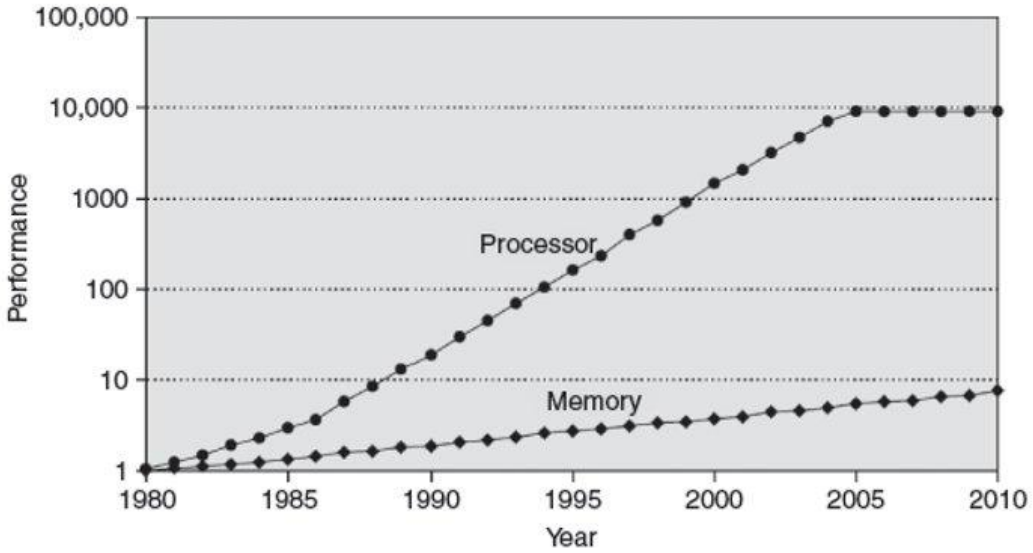
Memory Interface



Processor-Memory Gap

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

Diverging processor and memory performance



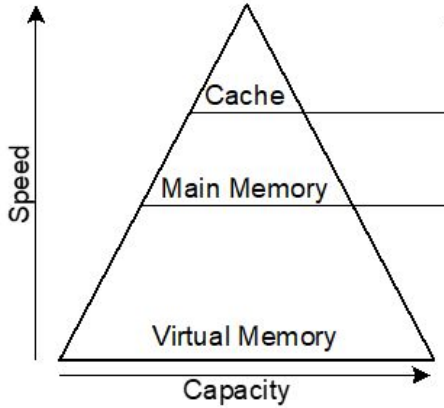
DRAM speed has improved by only about 7% per year, whereas processor performance has improved at a rate of 25 to 50% per year

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

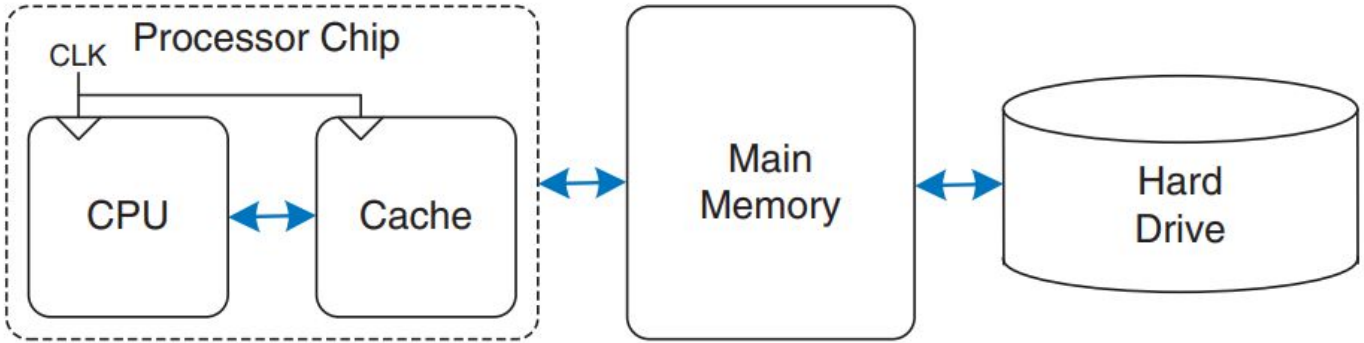
Computers store the most commonly used instructions and data in a faster but smaller memory, called a cache

The cache is usually built out of SRAM on the same chip as the processor.

The cache speed is comparable to the processor speed, because SRAM is inherently faster than DRAM, and because the on-chip memory eliminates lengthy delays caused by traveling to and from a separate chip.

A typical memory hierarchy

The hard drive provides an illusion of more capacity than actually exists in the main memory. It is thus called virtual memory.



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

Designers (and computer buyers) need quantitative ways to measure the performance of memory systems to evaluate the cost-benefit trade-offs of various alternatives.

- **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

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

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 = # 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

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

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?**

$$\textbf{Hit Rate} = 1250/2000 = \textbf{0.625}$$

$$\textbf{Miss Rate} = 750/2000 = \textbf{0.375} = 1 - \text{Hit Rate}$$

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?**

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

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

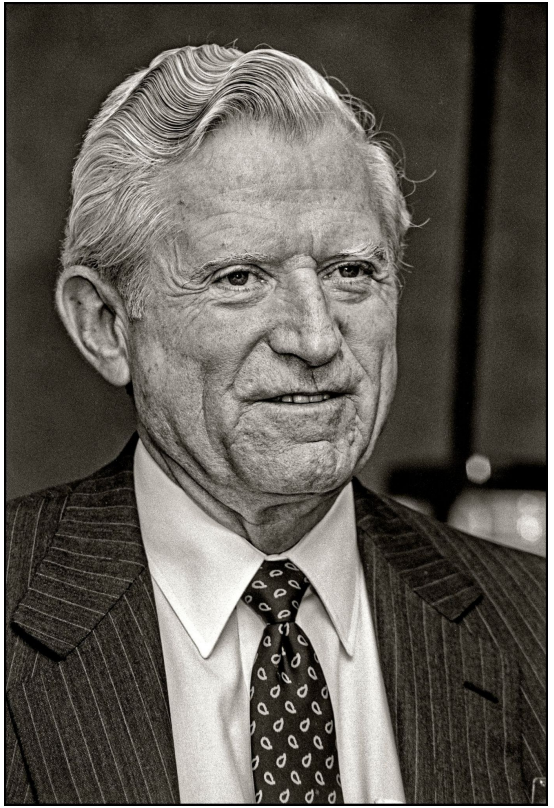
Memory Performance Example 2

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

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

Gene Amdahl, 1922-2015

- **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



Dick Barnatt/Getty Images

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 Hit and Miss

If the processor requests data that is available in the cache, it is returned quickly. This is called a cache hit.

Otherwise, the processor retrieves the data from main memory (DRAM). This is called a cache miss.

> "If the cache hits most of the time, then the processor seldom has to wait for the slow main memory, and the average access time is low."

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?

- An ideal cache would anticipate all of the data needed by the processor and fetch it from main memory ahead of time so that the cache has a zero miss rate.
- 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 Block/Cache Line

When the cache fetches one word from memory, it may also fetch several adjacent words. This group of words is called a cache block or cache line.

Cache Terminology

- **Capacity (C):**
  - A cache holds commonly used memory data. The number of data bytes that it can hold is called capacity.
- **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

General Working Principle of Cache

"When the processor attempts to access data, it first checks the cache for the data. If the cache hits, the data is available immediately. If the cache misses, the processor fetches the data from main memory and places it in the cache for future use. To accommodate the new data, the cache must replace old data."

Caches use spatial and temporal locality to predict what data will be needed next. If a program accesses data in a random order, it would not benefit from a cache.

How is data found?

- A cache is organized into S sets, each of which holds one or more blocks of data.
- The relationship between the address of data in main memory and the location of that data in the cache is called the **mapping**.
- 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 ($B = 8$)

In a direct mapped cache, each set contains exactly one block, so the cache has S = B sets.

In an N-way set associative cache, each set contains N blocks.

Example Cache Parameters

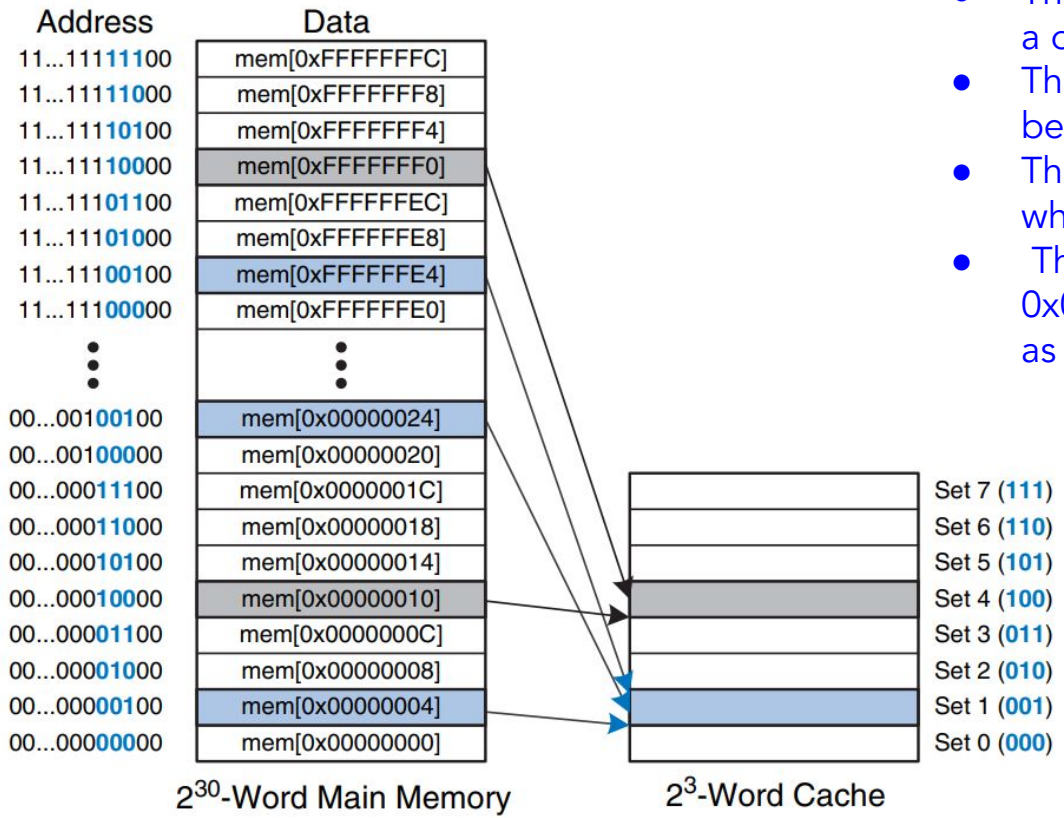
- $C = 8$ words (capacity)
- $b = 1$ word (block size)
- So, $B = 8$ (# of blocks)

Ridiculously small, but will illustrate organizations

Direct Mapped Cache

- A direct mapped cache has one block in each set, so it is organized into S=B sets.
- To understand the mapping of memory addresses onto cache blocks, imagine main memory as being mapped into b-word blocks, just as the cache is.
- An address in block 0 of main memory maps to set 0 of the cache.
- An address in block 1 of main memory maps to set 1 of the cache, and so forth until an address in block B - 1 of main memory maps to block B - 1 of the cache.
- There are no more blocks of the cache, so the mapping wraps around, such that block B of main memory maps to block 0 of the cache

Direct Mapped Cache

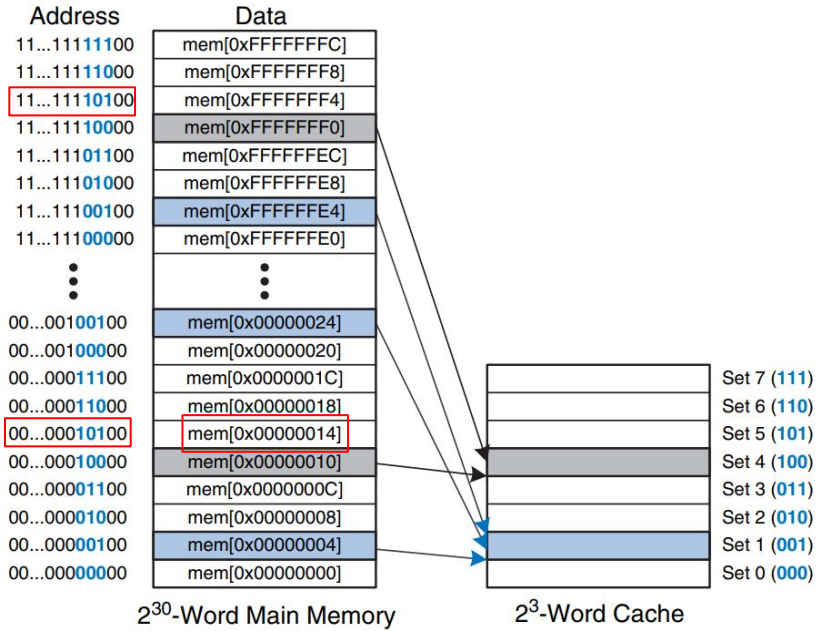


- A capacity of eight words and a block size of one word.
- The cache has eight sets, each of which contains a one-word block.
- The bottom two bits of the address are always 00, because they are word aligned.
- The next $\log_2 8 = 3$ bits indicate the set onto which the memory address maps.
- Thus, the data at addresses 0x00000004, 0x00000024, . . . , 0xFFFFFFE4 all map to set 1, as shown in blue.

- Likewise, data at addresses 0x00000010, . . . , 0xFFFFFFF0 all map to set 4, and so forth.
- Each main memory address maps to exactly one set in the cache.

Example

To what cache set in the Figure does the word at address 0x00000014 map? Name another address that maps to the same set.



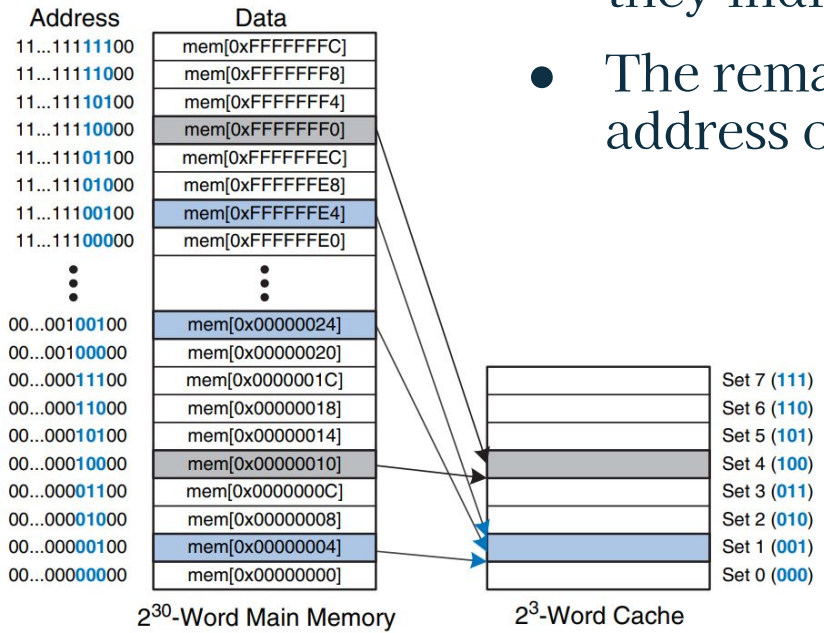
The two least significant bits of the address are 00 (4 = 100), because the address is word aligned. The next three bits are 101, so the word maps to set 5. Words at addresses 0x34, 0x54, 0x74, . . . , 0xFFFFFFF4 all map to this same set.

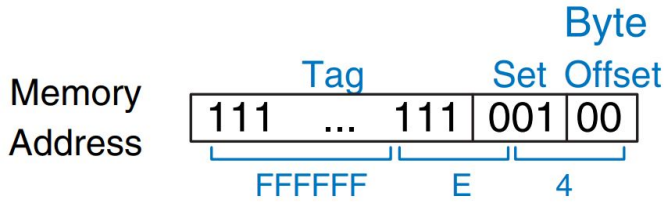
Cache fields

- Because many addresses map to a single set, the cache must also keep track of the address of the data actually contained in each set.
- The least significant bits of the address specify which set holds the data.
- The remaining most significant bits are called the tag and indicate which of the many possible addresses is held in that set.

Cache fields

- The two least significant bits of the 32-bit address are called the **byte offset** because they indicate the byte within the word.
- The next three bits are called the **set bits**, because they indicate the set to which the address maps.
- The remaining 27 **tag bits** indicate the memory address of the data stored in a given cache set.





Cache fields for address 0xFFFFFFE4 when mapping to the cache

Example

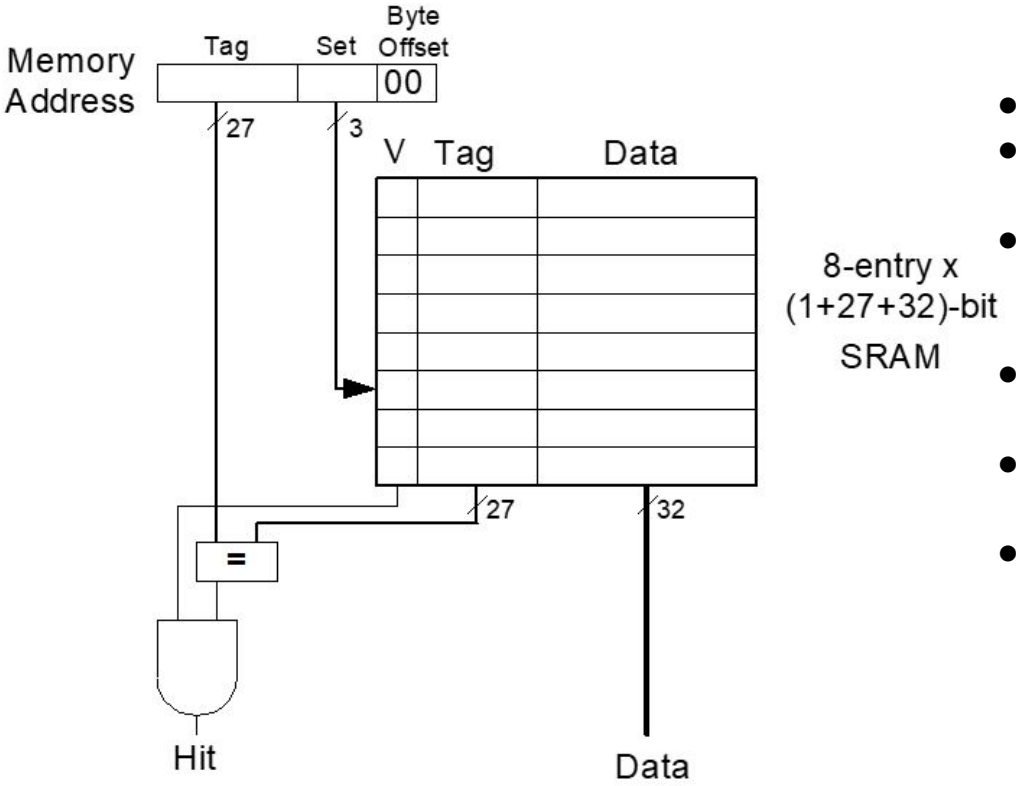
Find the number of set and tag bits for a direct mapped cache with 1024 (2^{10}) sets and a one-word block size. The address size is 32 bits.

A cache with 2^{10} sets requires $\log2(2^{10}) = 10$ set bits. The two least significant bits of the address are the byte offset, and the remaining 32 - 10 – 2 = 20 bits form the tag.

Valid Bit

Sometimes, such as when the computer first starts up, the cache sets contain no data at all. The cache uses a valid bit for each set to indicate whether the set holds meaningful data. If the valid bit is 0, the contents are meaningless.

Direct Mapped Cache Hardware

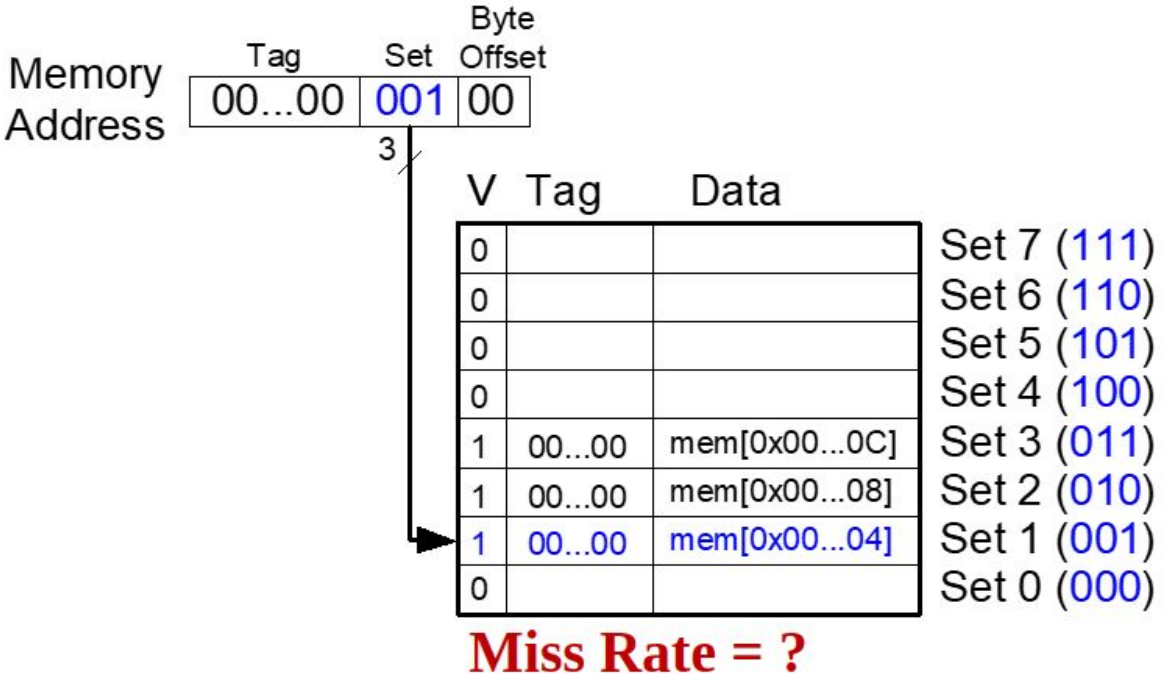


- The cache is constructed as an eight-entry SRAM.
- Each entry, or set, contains one line consisting of 32 bits of data, 27 bits of tag, and 1 valid bit.
- The cache is accessed using the 32-bit address. The two least significant bits, the byte offset bits, are ignored for word accesses.
- The next three bits, the set bits, specify the entry or set in the cache.
- A load instruction reads the specified entry from the cache and checks the tag and valid bits.
- If the tag matches the most significant 27 bits of the address and the valid bit is 1, the cache hits and the data is returned to the processor. Otherwise, the cache misses and the memory system must fetch the data from main memory.

Direct Mapped Cache Performance

ARM Assembly Code

```
 MOV R0, #5
MOV R1, #0
LOOP CMP R0, #0
                BEQ DONE
                LDR R2, [R1, #4]
                LDR R3, [R1, #12]
                LDR R4, [R1, #8]
                SUB R0, R0, #1
                B   LOOP
DONE
```

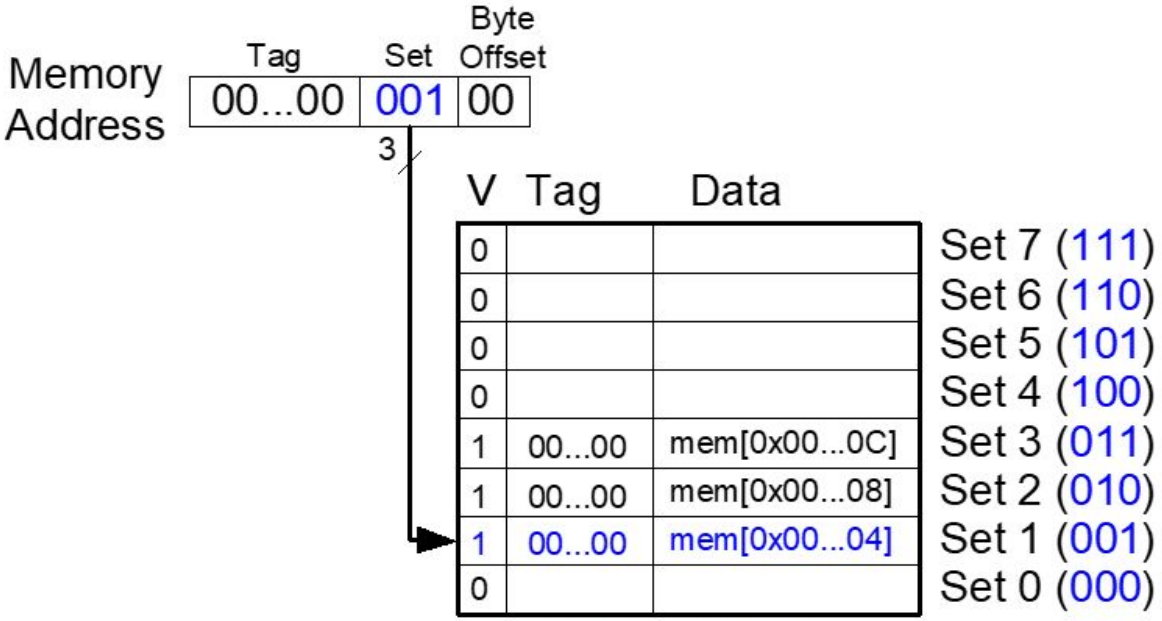


Miss Rate = ?

Direct Mapped Cache Performance

ARM Assembly Code

```
MOV R0, #5
MOV R1, #0
LOOP CMP R0, #0
                BEQ DONE
                LDR R2, [R1, #4]
                LDR R3, [R1, #12]
                LDR R4, [R1, #8]
                SUB R0, R0, #1
                B   LOOP
DONE
```



Temporal Locality

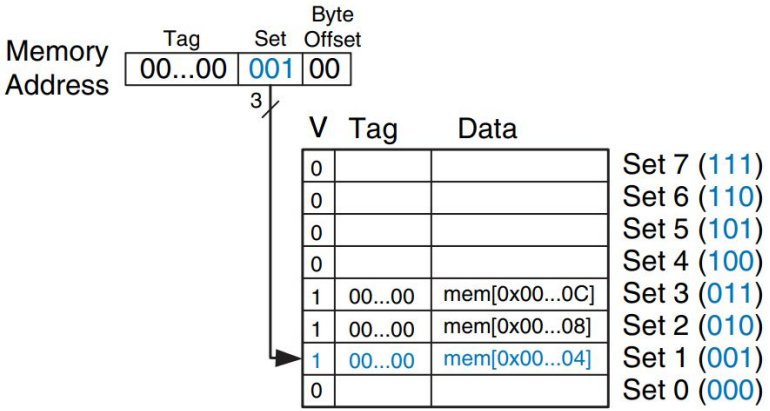
Compulsory Misses

Miss Rate = 3/15 = 20%

Temporal Locality With A Direct Mapped Cache

```
MOV R0, #5
MOV R1, #0
LOOP CMP R0, #0
     BEQ DONE
     LDR R2, [R1, #4]
     LDR R3, [R1, #12]
     LDR R4, [R1, #8]
     SUB R0, R0, #1
     B LOOP
DONE
```

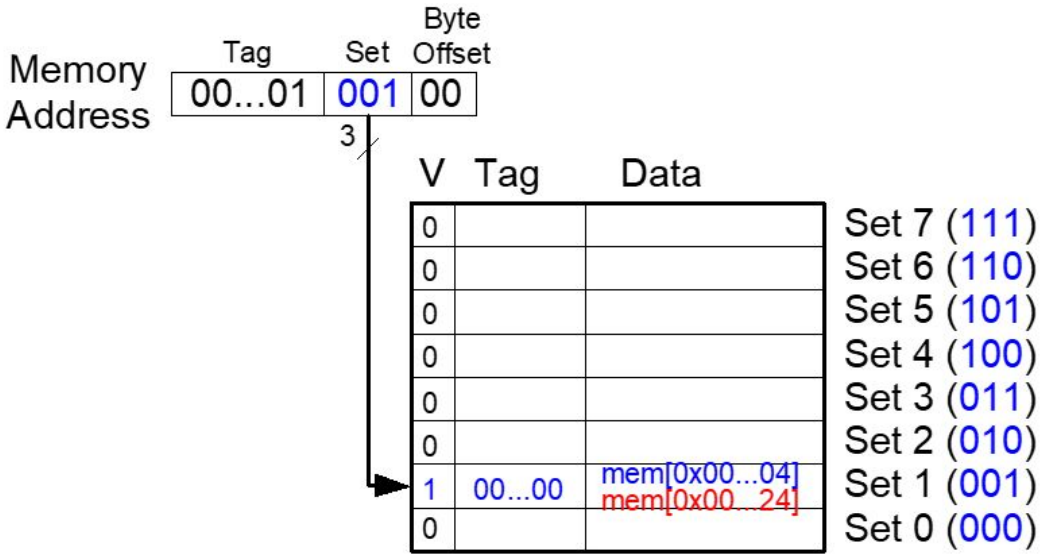
The program contains a loop that repeats for five iterations.
Each iteration: 3 memory access
First iteration: cache empty, data fetched, the next four times the loop executes, the data found in the cache. Miss rate: (3*1)/(3*5) = 0.2 = 20%



Direct Mapped Cache: Conflict

ARM Assembly Code

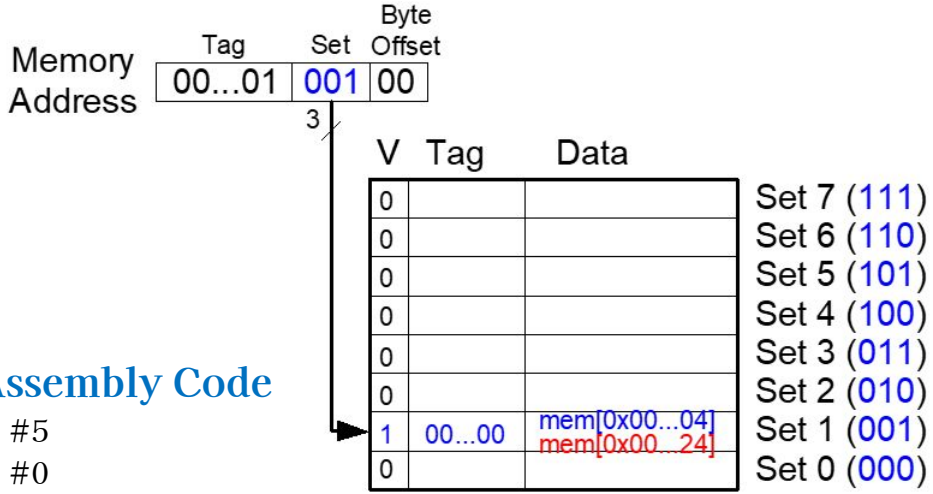
```
MOV R0, #5
MOV R1, #0
LOOP CMP R0, #0
        BEQ DONE
        LDR R2, [R1, #0x4]
        LDR R3, [R1, #0x24]
        SUB R0, R0, #1
        B LOOP
DONE
```



Miss Rate = ?

Direct Mapped Cache: Conflict

- Memory addresses 0x4 and 0x24 both map to set 1.
- During the initial execution of the loop, data at address 0x4 is loaded into set 1 of the cache. Then data at address 0x24 is loaded into set 1, evicting the data from address 0x4.
- Upon the second execution of the loop, the pattern repeats and the cache must refetch data at address 0x4, evicting data from address 0x24.
- The two addresses conflict, and the miss rate is 100%.



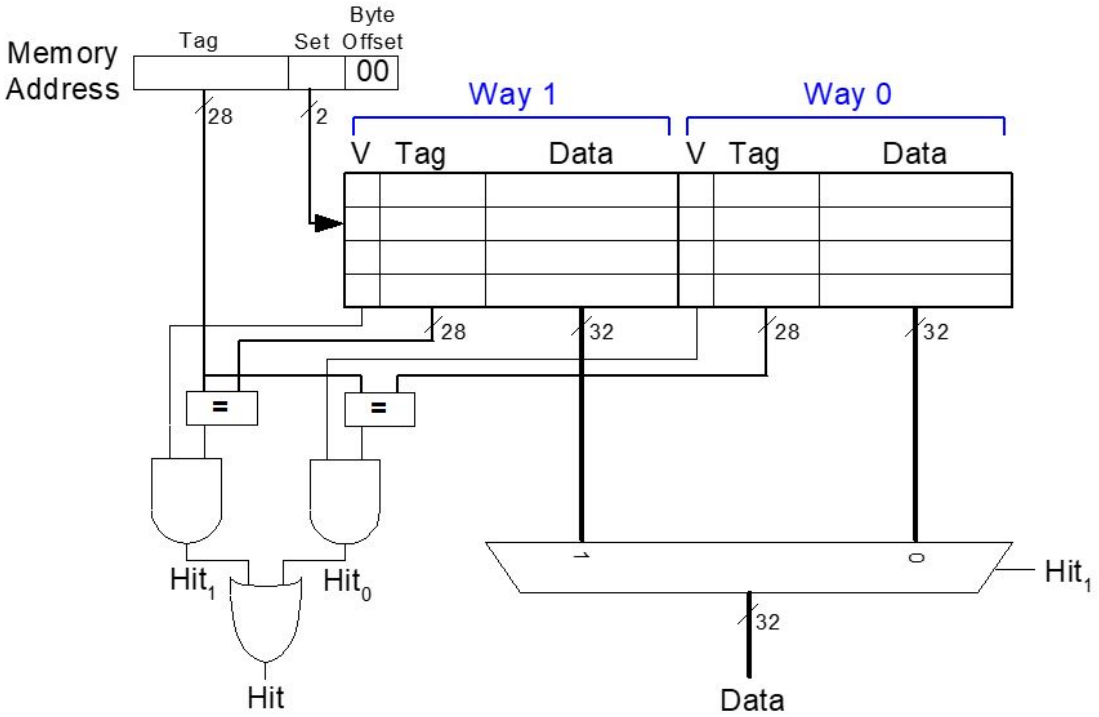
ARM Assembly Code

```
MOV R0, #5
MOV R1, #0
LOOP CMP R0, #0
        BEQ DONE
        LDR R2, [R1, #0x4]
        LDR R3, [R1, #0x24]
        SUB R0, R0, #1
        B LOOP
DONE
```

Miss Rate = 10/10 = 100%

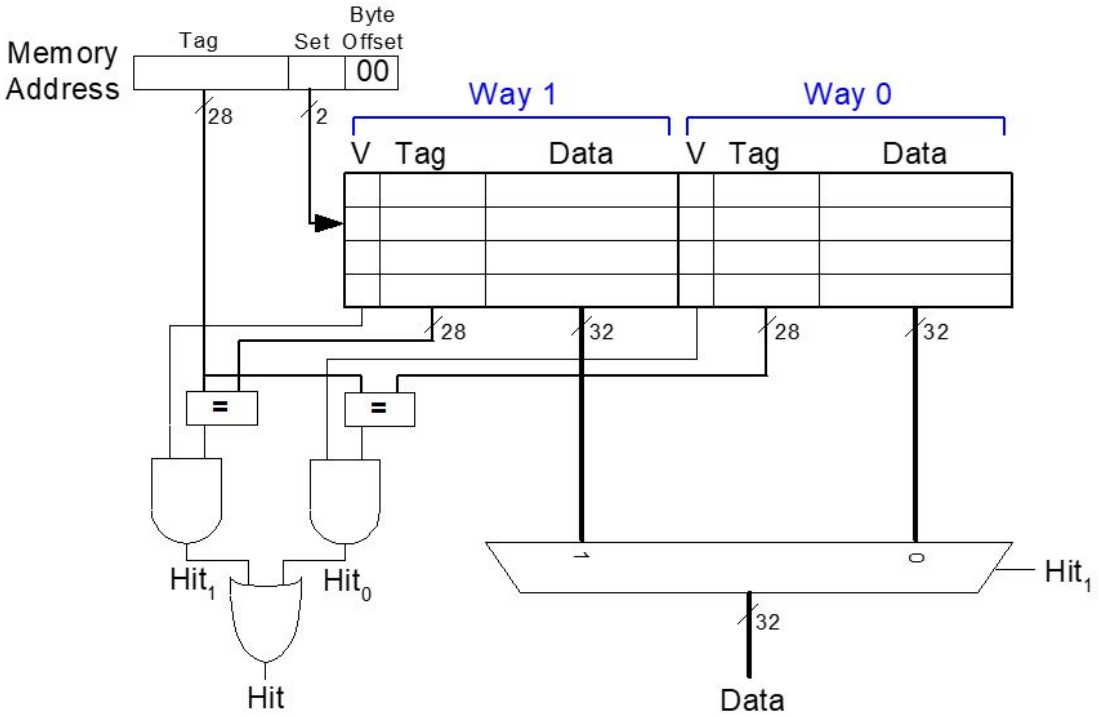
Conflict Misses

N-Way (or Multi-way) Set Associative Cache



- An N-way set associative cache reduces conflicts by providing N blocks in each set where data mapping to that set might be found.
- Each memory address still maps to a specific set, but it can map to any one of the N blocks in the set.
- Hence, a direct mapped cache is another name for a one-way set associative cache. N is also called the degree of associativity of the cache.

N-Way (or Multi-way) Set Associative Cache



- C = 8-word, N = 2-way set associative cache. The cache now has only S = 4 sets rather than 8.
- Thus, only $\log_2 4 = 2$ set bits rather than 3 are used to select the set. The tag increases from 27 to 28 bits.
- Each set contains two ways or degrees of associativity. Each way consists of a data block and the valid and tag bits.
- The cache reads blocks from both ways in the selected set and checks the tags and valid bits for a hit. If a hit occurs in one of the ways, a multiplexer selects data from that way.

N-Way (or Multi-way) Set Associative Cache

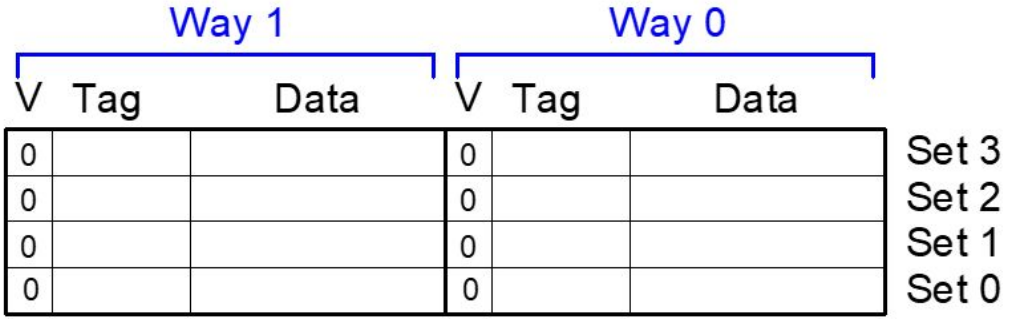
- Set associative caches generally have lower miss rates than direct mapped caches of the same capacity, because they have fewer conflicts.
- However, set associative caches are usually slower and somewhat more expensive to build because of the output multiplexer and additional comparators.
- They also raise the question of which way to replace when both ways are full
- Most commercial systems use set associative caches.

N-Way Set Associative Performance

Miss Rate = ?

ARM Assembly Code

```
        MOV R0, #5
        MOV R1, #0
LOOP CMP R0, 0
        BEQ DONE
        LDR R2, [R1, #0x4]
        LDR R3, [R1,
   #0x24]
        SUB R0, R0, #1
        B   LOOP
DONE
```



N-Way Set Associative Performance

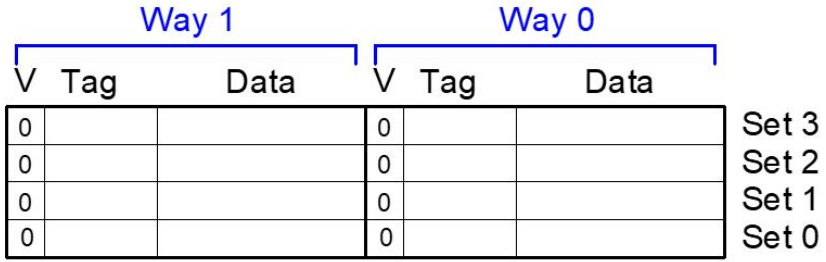
ARM Assembly Code

```
                MOV R0, #5
                MOV R1, #0
LOOP CMP R0, 0
                BEQ DONE
                LDR R2, [R1, #0x4]
                LDR R3, [R1, #0x24]
                SUB R0, R0, #1
                B   LOOP
DONE
```

Associativity reduces conflict misses

Miss Rate = 2/10 = 20%

- The cache has two ways, so it can accommodate data from both addresses.
- During the first loop iteration, the empty cache misses both addresses and loads both words of data into the two ways of set 1
- On the next four iterations, the cache hits. Hence, the miss rate is 2/10 = 20%.



Announcement

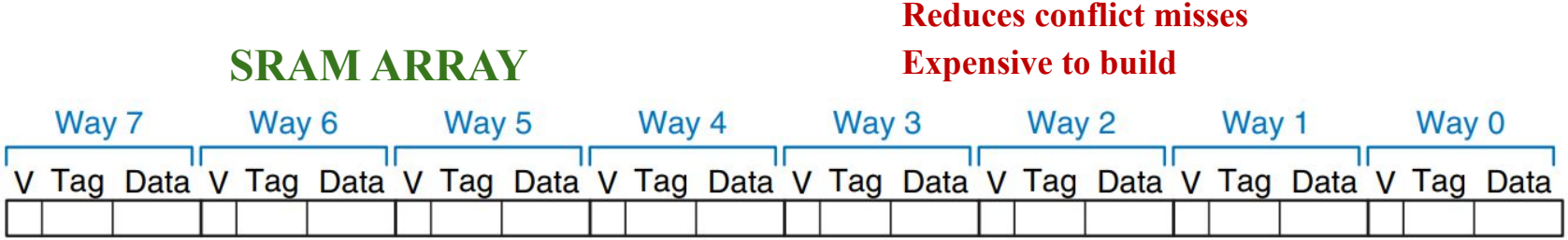
- I will post midterm solution today
- HW04 (Oct 30, 11:59 pm due) and HW05 (Nov 17, 11:59 pm due) have been posted.
- Please use this link to schedule oral exam: https://calendly.com/rkb0022/15-minute-meeting?month=2023-11. Please select only for November month.
    - If it doesn't work for you, please let me know
- I will post a take-home make up exam that can make up for maximum 12.5% (maximum is half of what you could get by doing midterm) of your grade. It will appear as bonus 12.5% on the Canvas Assignment. Questions will be similar to one appeared in mid-term. However, they will all be subjective questions. You will have 10 days to complete. No extension will be given. Paper copy only.

Fully Associative Cache

A fully associative cache contains a single set with B ways, where B is the number of blocks. A memory address can map to a block in any of these ways. A fully associative cache is another name for a B-way set associative cache with one set.

Reduces conflict misses

Expensive to build



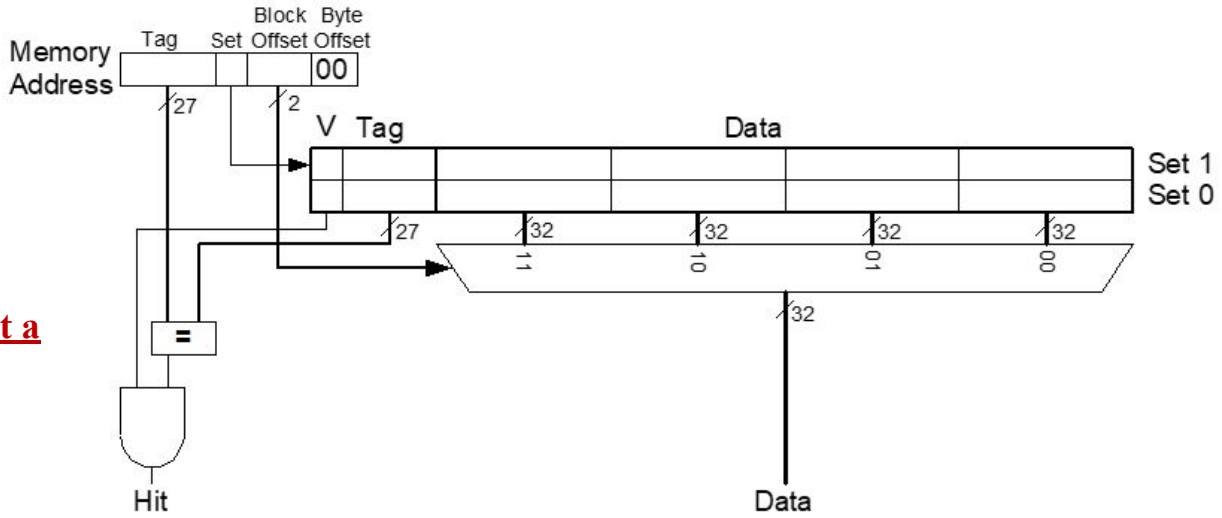
- SRAM array of a fully associative cache with eight blocks.
- Fully associative caches tend to have the fewest conflict misses for a given cache capacity, but they require more hardware for additional tag comparisons.
- They are best suited to relatively small caches because of the large number of comparators.
- Upon a data request, eight tag comparisons (not shown) must be made, because the data could be in any block.

Spatial Locality and Block Size

To exploit spatial locality, a cache uses larger blocks to hold several consecutive words.

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

The advantage of a block size greater than one is that when a miss occurs and the word is fetched into the cache, the adjacent words in the block are also fetched. Therefore, subsequent accesses are more likely to hit because of spatial locality.



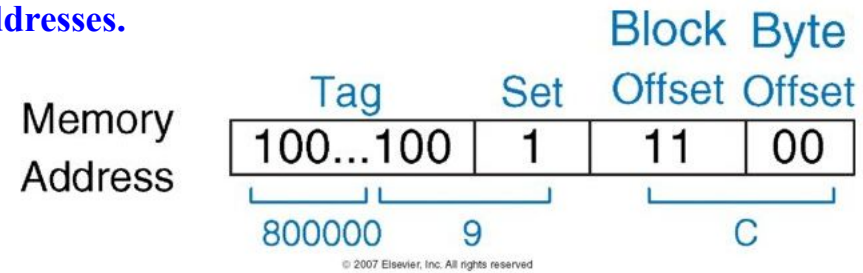
$\log_2 2 = 1$ bit is used to select a set.

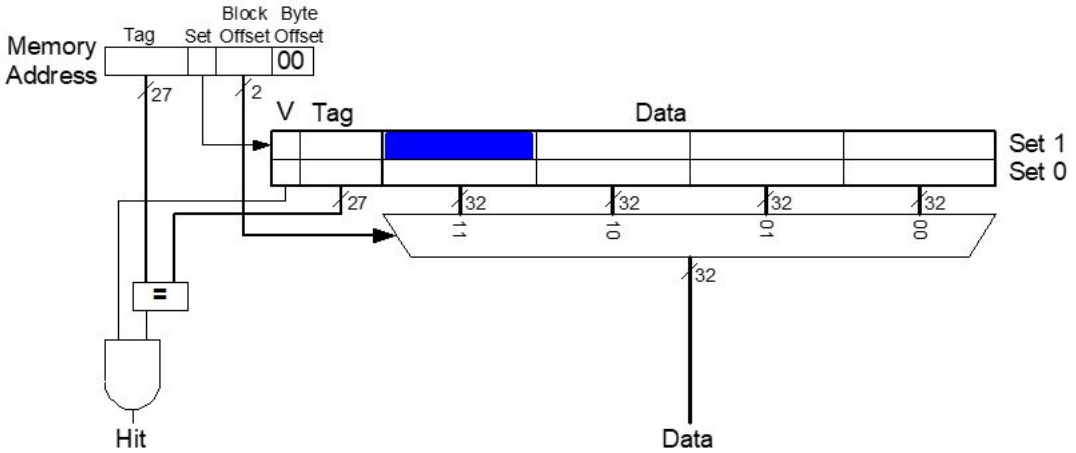
But we need a mux to select a word within the set.

Cache with Larger Block Size

- A large block size means that a fixed-size cache will have fewer blocks.
- This may lead to more conflicts, increasing the miss rate.
- Moreover, it takes more time to fetch the missing cache block after a miss, because more than one data word is fetched from main memory.
- The time required to load the missing block into the cache is called the **miss penalty**.

Only one tag is needed for the entire block, because the words in the block are at consecutive addresses.





Direct Mapped Cache Performance

ARM assembly code

```
MOV R0, #5
MOV R1, #0
LOOP CMP R0, 0
          BEQ DONE
          LDR R2, [R1, #4]
          LDR R3, [R1, #12]
          LDR R4, [R1, #8]
          SUB R0, R0, #1
          B   LOOP
DONE
```

Miss Rate = ?

Consider we have the eight-word direct mapped cache with a four-word block size.

Direct Mapped Cache Performance

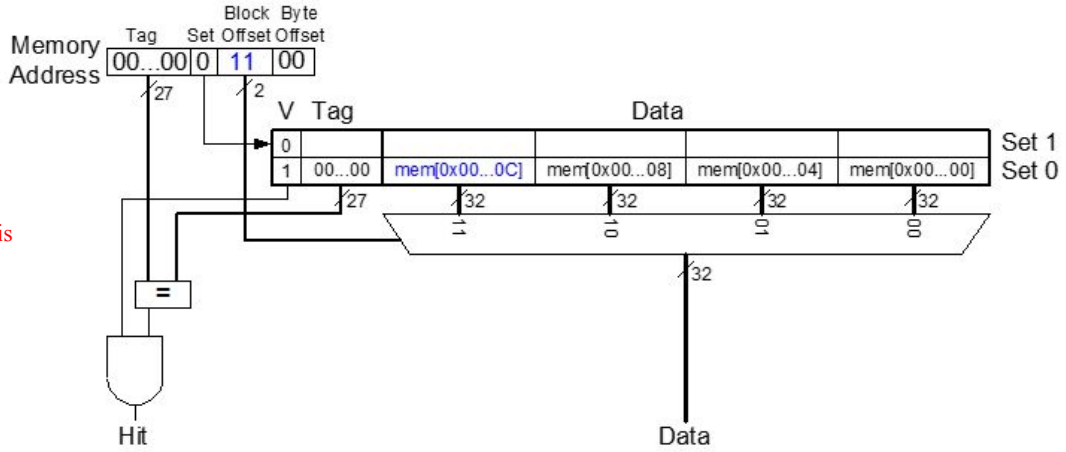
ARM assembly code

```
MOV R0, #5
MOV R1, #0
LOOP CMP R0, 0
          BEQ DONE
          LDR R2, [R1, #4]
          LDR R3, [R1, #12]
          LDR R4, [R1, #8]
          SUB R0, R0, #1
          B   LOOP
DONE
```

On the first loop iteration, the cache misses on the access to memory address 0x4. This access loads data at addresses 0x0 through 0xC into the cache block. All subsequent accesses (as shown for address 0xC) hit in the cache.

Miss Rate = 1/15 = 6.67%

Larger blocks
reduce compulsory misses
through spatial locality



Cache Organization Summary

- Caches are organized as two-dimensional arrays. The rows are called sets, and the columns are called ways.
- Each entry in the array consists of a data block and its associated valid and tag bits. Caches are
- characterized by
  - 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

What data is replaced?

The principle of temporal locality suggests that the best choice is to evict the least recently used block, because it is least likely to be used again soon. Hence, most associative caches have a **least recently used (LRU) replacement policy**.

Use bit

In a two-way set associative cache, a use bit, U, indicates which way within a set was least recently used.

- Each time one of the ways is used, U is adjusted to indicate the other way.
- For set associative caches with more than two ways, tracking the least recently used way becomes complicated.
- To simplify the problem, the ways are often divided into two groups and U indicates which group of ways was least recently used.
- Upon replacement, the new block replaces a random block within the least recently used group. Such a policy is called pseudo-LRU and is good enough in practice.

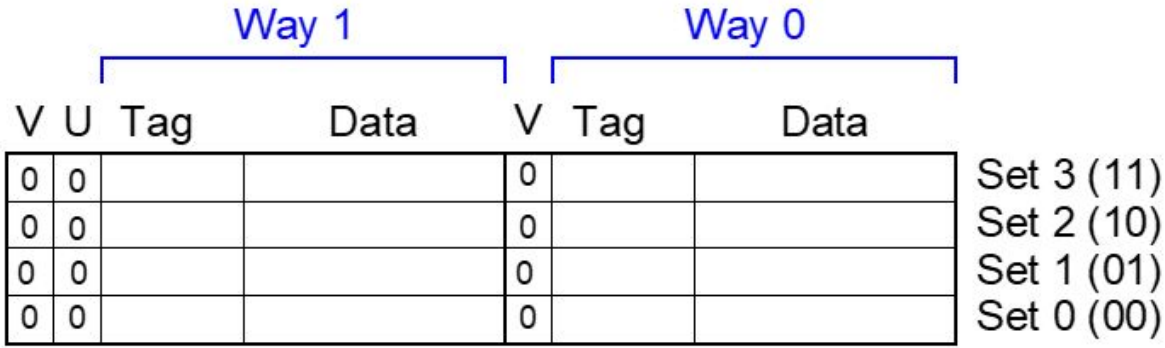
Capacity Misses

- 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

LRU Replacement

ARM Assembly Code

```
MOV R0, #0
LDR R1, [R0, #4]
LDR R2, [R0, #0x24]
LDR R3, [R0, #0x54]
```



LRU Replacement

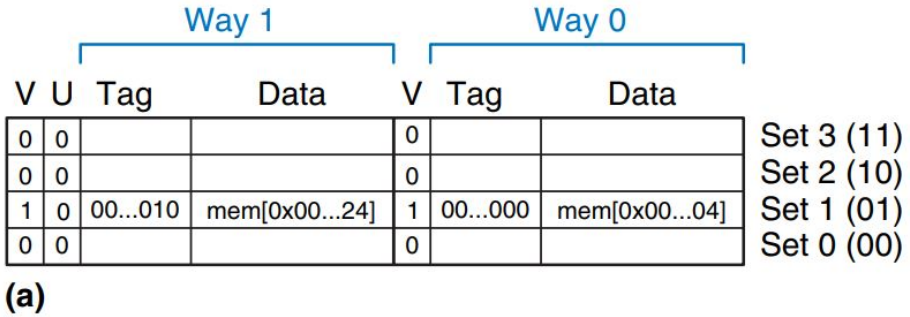
```
MOV R0, #0
LDR R1, [R0, #4]
LDR R2, [R0, #0x24]
LDR R3, [R0, #0x54]
```

The first two instructions load data from memory addresses 0x4 and 0x24 into set 1 of the cache.

U = 0 indicates that data in way 0 was the least recently used.

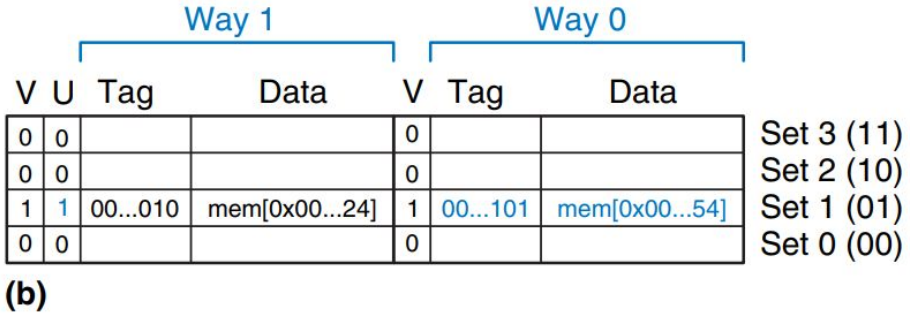
The next memory access, to address 0x54, also maps to set 1 and replaces the least recently used data in way 0.

The use bit U is set to 1 to indicate that data in way 1 was the least recently used.



		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)

(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	1	00...010	mem[0x00...24]	1	00...101	mem[0x00...54]	Set 1 (01)
0	0			0			Set 0 (00)

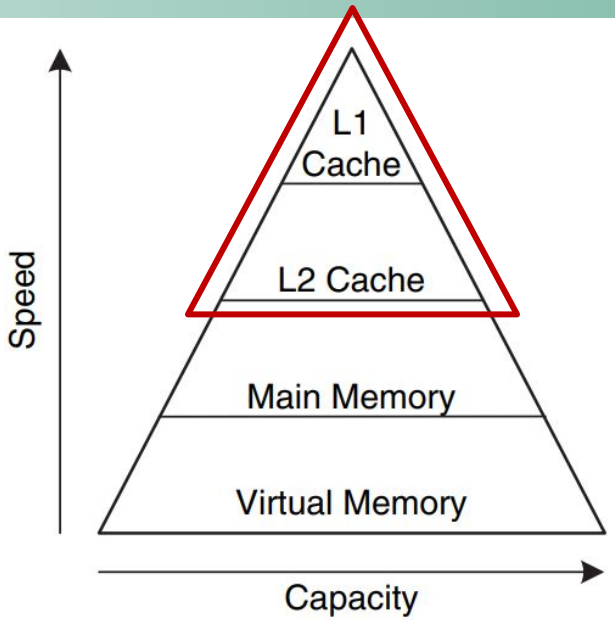
(b)

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

Multilevel Caches

- Larger caches have lower miss rates, longer access times
- Modern systems often use at least two levels of caches
- 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



The processor first looks for the data in the L1 cache. If the L1 cache misses, the processor looks in the L2 cache. If the L2 cache misses, the processor fetches the data from main memory.

Example: System With An L2 Cache

Consider 1, 10, and 100 cycles for the L1 cache, L2 cache, and main memory. Assume that the L1 and L2 caches have miss rates of 5% and 20%, respectively. Specifically, of the 5% of accesses that miss the L1 cache, 20% of those also miss the L2 cache. What is the average memory access time (AMAT)?

Example: System With An L2 Cache

Consider 1, 10, and 100 cycles for the L1 cache, L2 cache, and main memory. Assume that the L1 and L2 caches have miss rates of 5% and 20%, respectively. Specifically, of the 5% of accesses that miss the L1 cache, 20% of those also miss the L2 cache. What is the average memory access time (AMAT)?

1 cycle + 0.05[10 cycles + 0.2(100 cycles)] = 2.5 cycles

Reducing Miss Rate

Cache misses can be reduced by changing capacity, block size, and/or associativity.

The first step to reducing the miss rate is to understand the causes of the misses.

The misses can be classified as compulsory, capacity, and conflict.

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

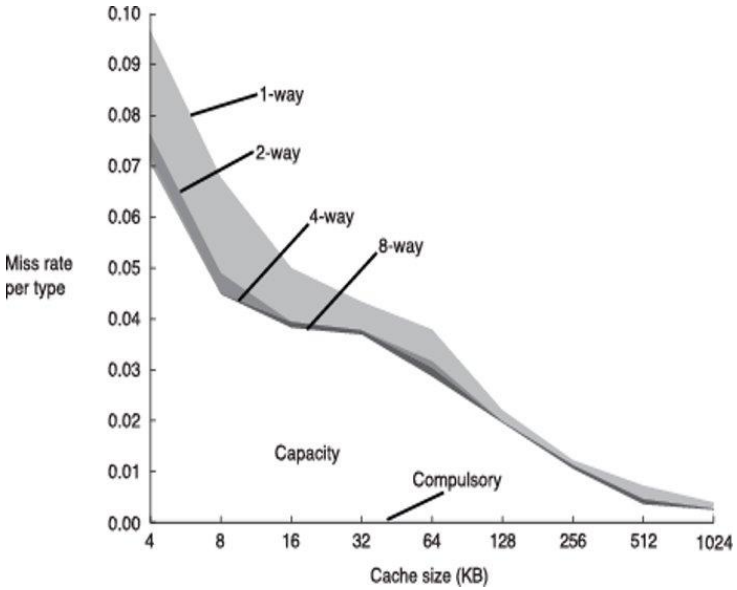
Impact of Changing Cache Configuration

Changing cache parameters can affect one or more type of cache miss.

- For example, increasing cache capacity can reduce conflict and capacity misses, but it does not affect compulsory misses.
- On the other hand, increasing block size could reduce compulsory misses (due to spatial locality) but might actually increase conflict misses (because more addresses would map to the same set and could conflict).

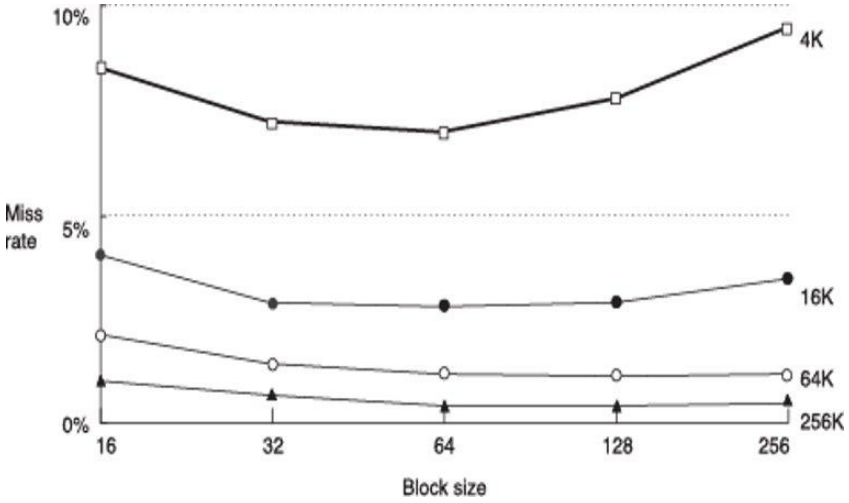
Miss Rate Trends

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

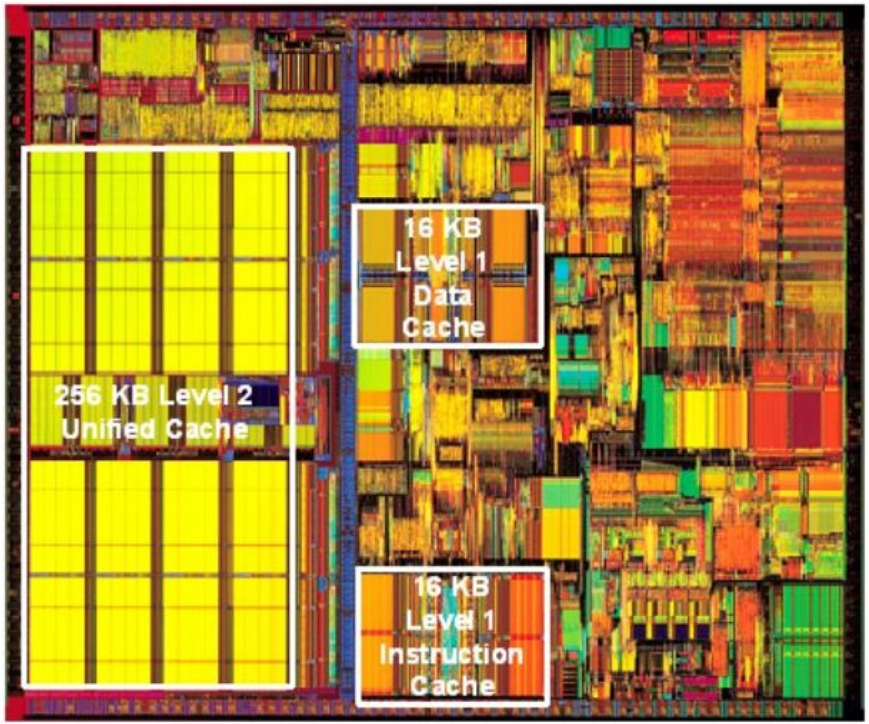


Miss Rate Trends

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

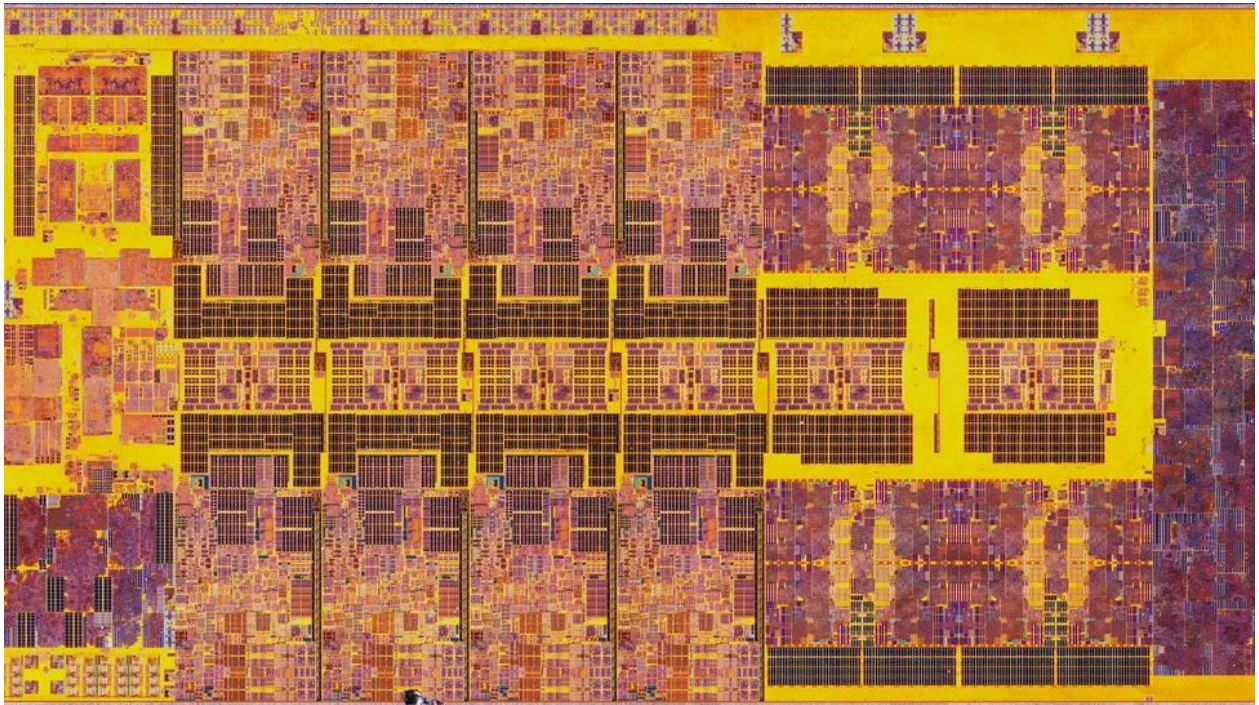


Intel Pentium III Die



Intel Core i9

https://www.pcgamer.com/oh-sorry-i-was-busy-admiring-these-gorgeous-die-shots-of-the-intel-core-i9-13900k/



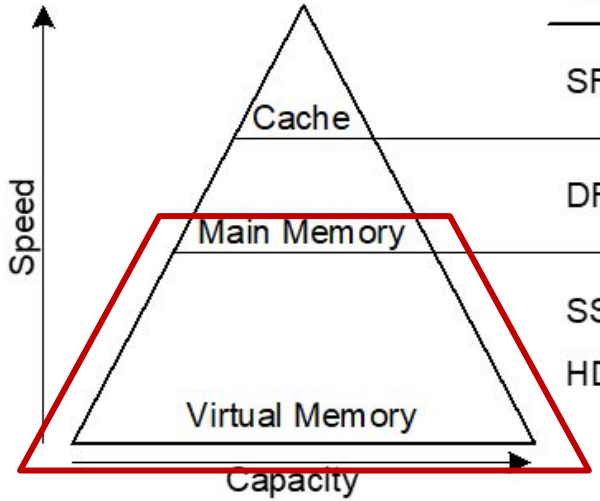
Virtual Memory

Virtual Memory

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

Memory Hierarchy

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



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

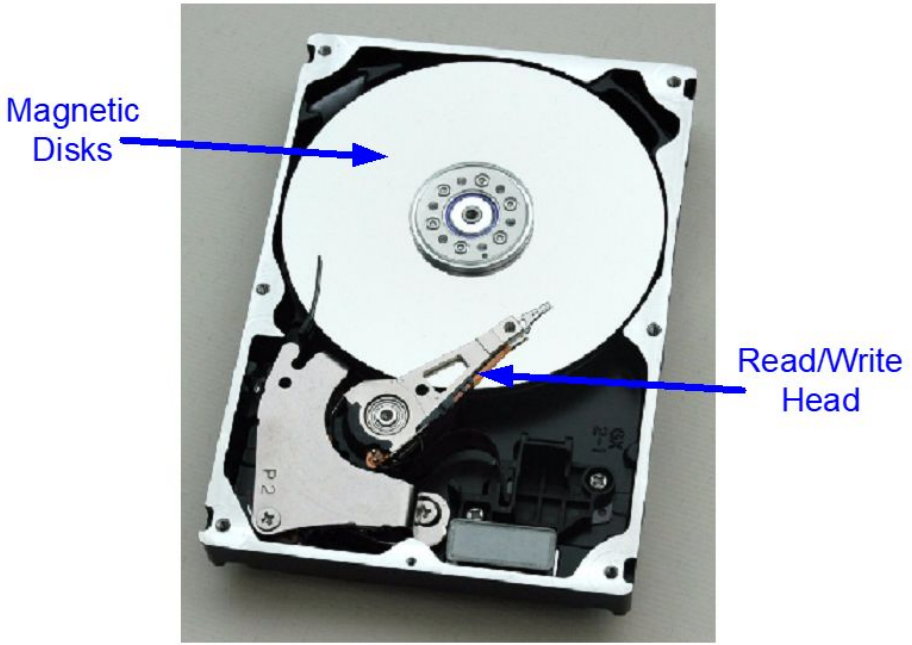
Hard Disk

Thee hard disk contains one or more rigid disks or platters, each of which has a read/write head on the end of a long triangular arm.

The head moves to the correct location on the disk and reads or writes data magnetically as the disk rotates beneath it.

The head takes several milliseconds to seek the correct location on the disk, which is fast from a human perspective but millions of times slower than the processor.

SSD is better in terms read/write speed and have no mechanical failure.



Virtual Memory

These numbers are just examples!!!

A computer with only 128 MB of DRAM, for example, could effectively provide 2 GB of memory using the hard drive. This larger 2-GB memory is called virtual memory, and the smaller 128-MB main memory is called physical 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

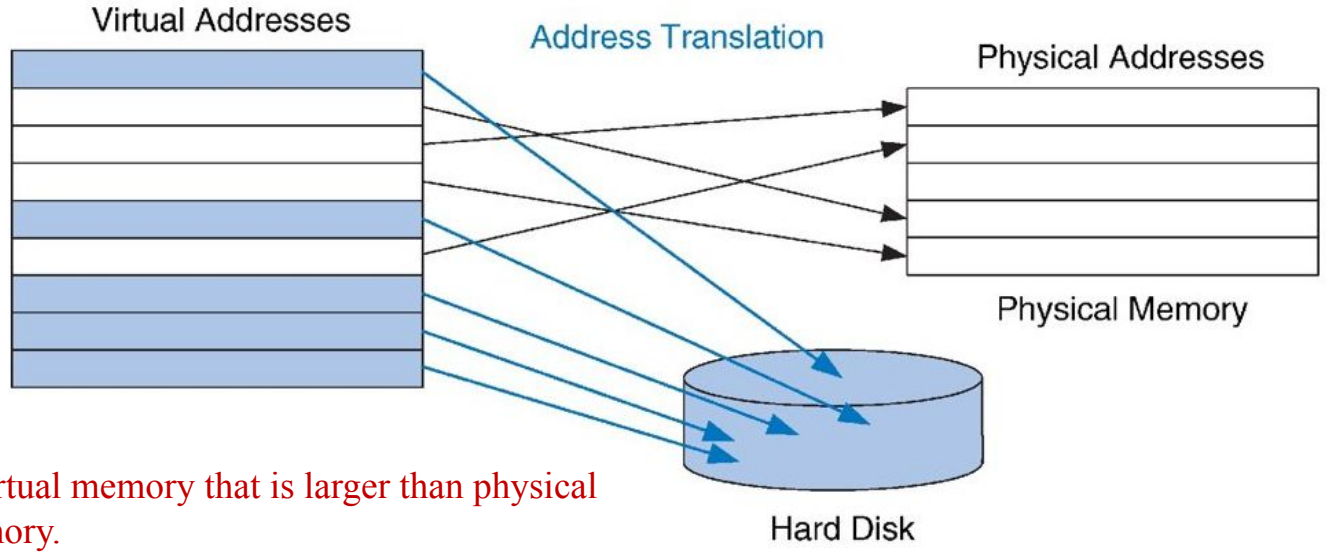
Physical memory acts as cache for virtual memory

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

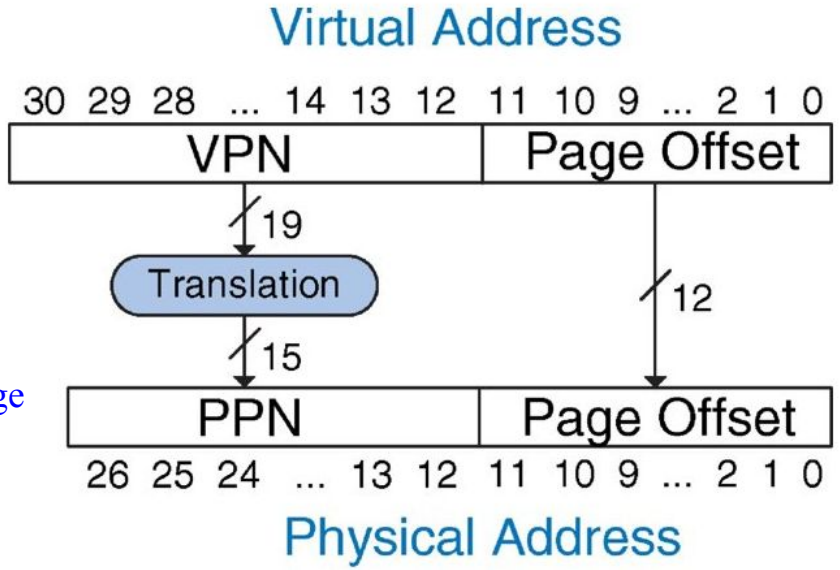


A virtual memory that is larger than physical memory.

Address Translation

The process of determining the physical address from the virtual address is called **address translation**.

If the processor attempts to access a virtual address that is not in physical memory, a page fault occurs, and the operating system loads the page from the hard drive into physical memory.



To avoid page faults caused by conflicts, any virtual page can map to any physical page. In other words, physical memory behaves as a fully associative cache for virtual memory.

Mapping virtual address to physical address

Each physical page would need a comparator to check the most significant virtual address bits against a tag to determine whether the virtual page maps to that physical page.

But this is very expensive and may require millions of comparators.

Page Table

- In practical implementation, the virtual memory system uses a page table to perform address translation.
- A page table contains an entry for each virtual page, indicating its location in physical memory or that it is on the hard drive.
- Each load or store instruction requires a page table access followed by a physical memory access.
- The page table access translates the virtual address used by the program to a physical address. The physical address is then used to actually read or write the data.

Translation Lookaside Buffer

But page table is also large – they are located in physical memory.

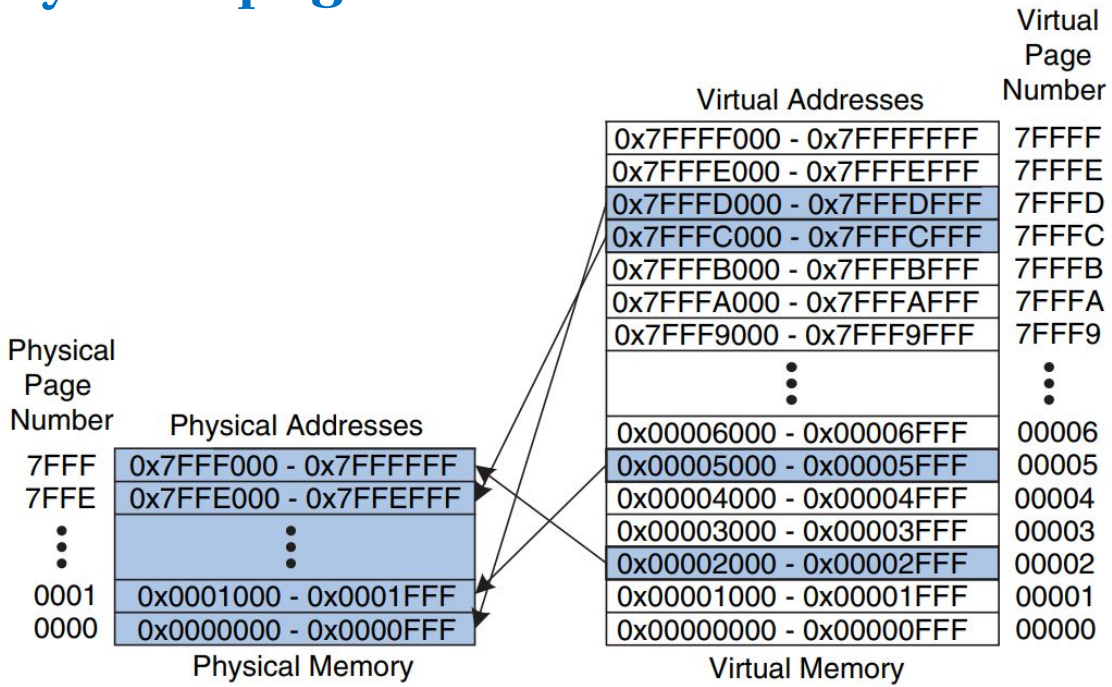
Each load or store involves two physical memory accesses: a page table access, and a data access.

To speed up address translation, a translation lookaside buffer (TLB) caches the most commonly used page table entries.

Address Translation

Virtual Memory Example

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



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

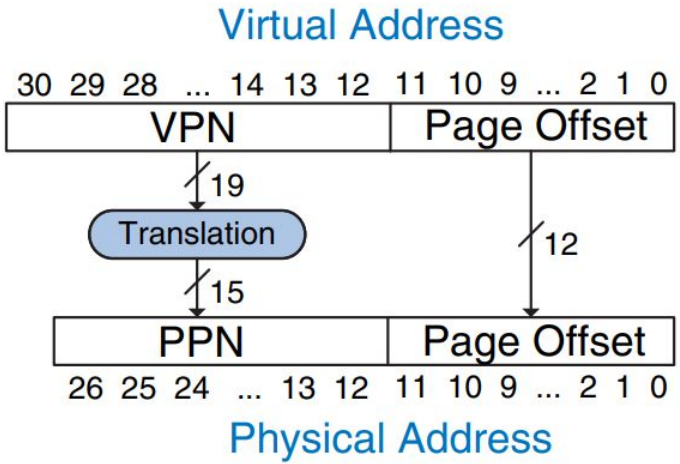
- **Organization:**
  - Virtual address: 31 bits
  - Physical address: 27 bits
  - Page offset: 12 bits
  - # Virtual pages = $2^{31}/2^{12} = 2^{19}$ (VPN = 19 bits)
  - # Physical pages = $2^{27}/2^{12} = 2^{15}$ (PPN = 15 bits)

Page Number and Page Offset

The most significant bits of the virtual or physical address specify the virtual or physical page number. The least significant bits specify the word within the page and are called the page offset.

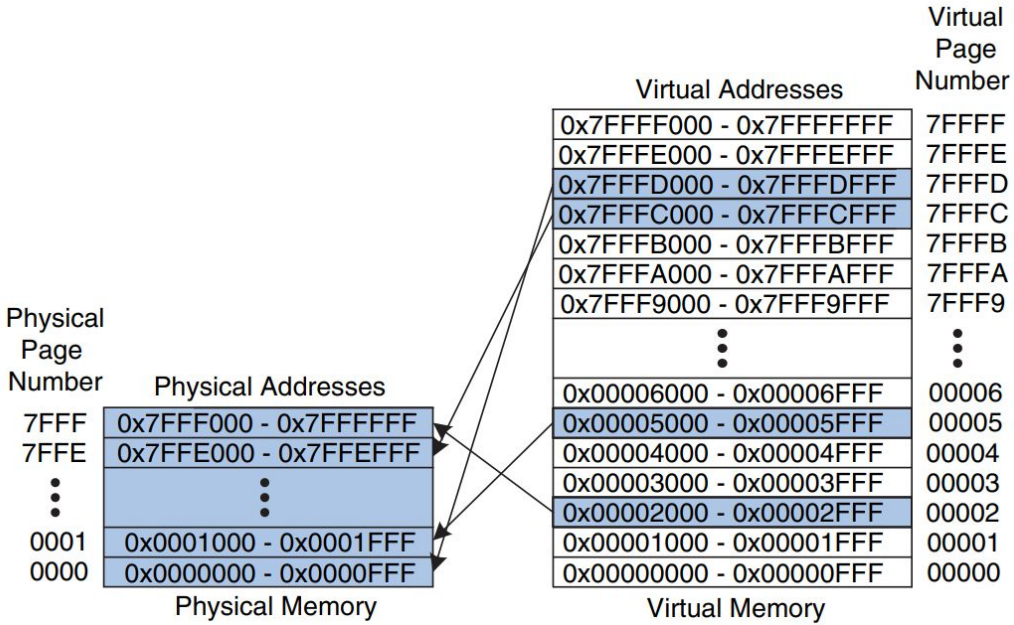
Translation Of A Virtual Address To A Physical Address

- The least significant 12 bits indicate the page offset and require no translation.
- The upper 19 bits of the virtual address specify the virtual page number (VPN) and are translated to a 15-bit physical page number (PPN).



Virtual Memory Example

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



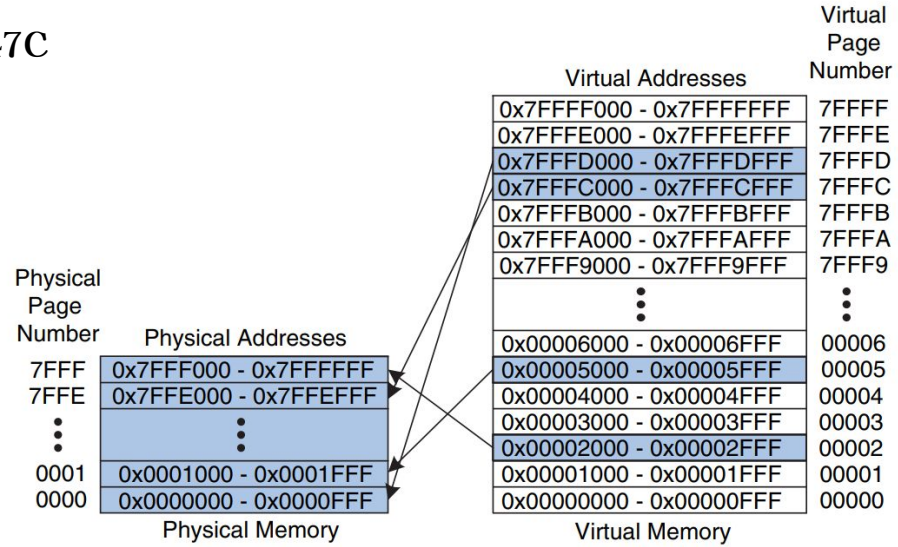
Virtual Memory Example

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**

The 12-bit page offset (0x47C) requires no translation. The remaining 19 bits of the virtual address give the virtual page number, so virtual address 0x247C is found in virtual page 0x2.

Virtual page 0x2 maps to physical page 0x7FFF. Thus, virtual address 0x247C maps to physical address 0x7FFF47C.



How to perform translation?

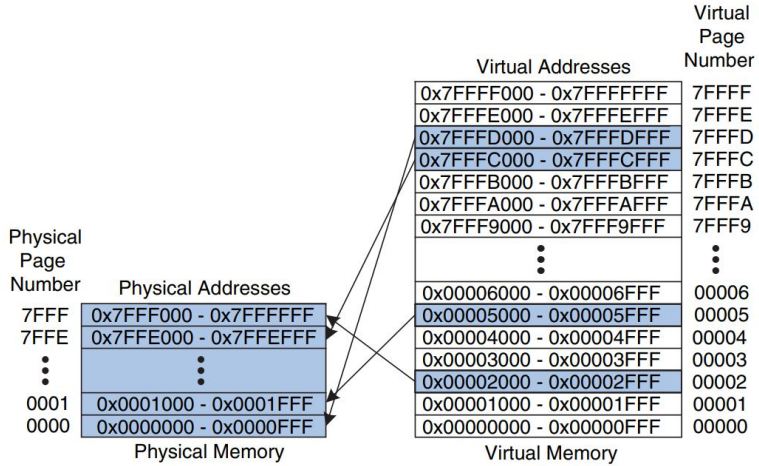
- **Page table**
  - The processor uses a page table to translate virtual addresses to physical addresses.
    - Entry for each virtual page
    - This entry contains a physical page number and a valid bit
    - Entry fields:
      - **Valid bit:** If the valid bit is 1, the virtual page maps to the physical page specified in the entry. Otherwise, the virtual page is found on the hard drive.
      - **Physical page number:** where the page is located

Page Table Example

V	Physical Page Number	Virtual Page Number
0		7FFFF
0		7FFFE
1	0x0000	7FFFD
1	0x7FFE	7FFFC
0		7FFFB
0		7FFFA
	⋮	⋮
0		00007
0		00006
1	0x0001	00005
0		00004
0		00003
1	0x7FFF	00002
0		00001
0		00000

Page Table

VPN is index into page table.

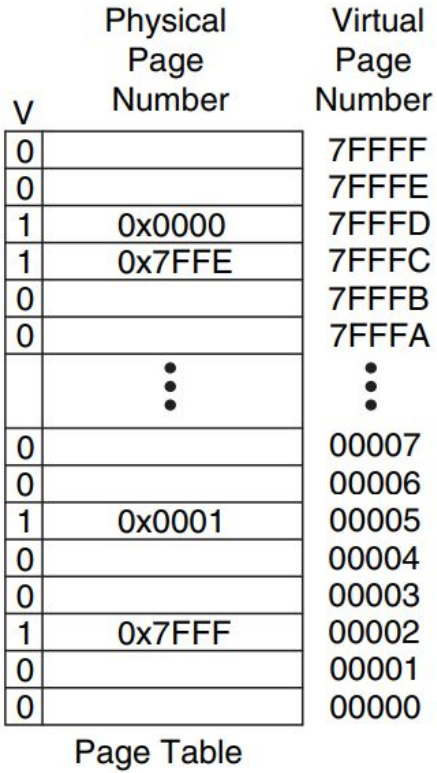


Let us assume for now that it is stored as a contiguous array.
This page contains mapping of memory system shown above on the right side.

The page table is indexed with the virtual page number (VPN). For example, entry 5 specifies that virtual page 5 maps to physical page 1. Entry 6 is invalid (V = 0), so virtual page 6 is located on the hard drive.

Page Table Example 1

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



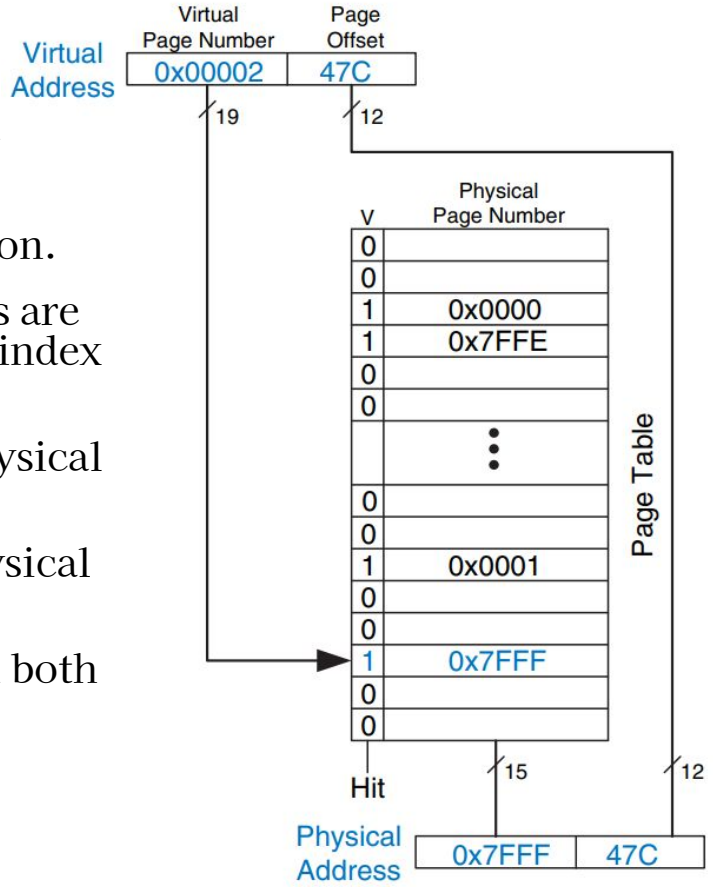
V	Physical Page Number	Virtual Page Number
0		7FFFF
0		7FFFE
1	0x0000	7FFFD
1	0x7FFE	7FFFC
0		7FFFB
0		7FFFA
⋮	⋮	⋮
0		00007
0		00006
1	0x0001	00005
0		00004
0		00003
1	0x7FFF	00002
0		00001
0		00000

Page Table

Page Table Example 1

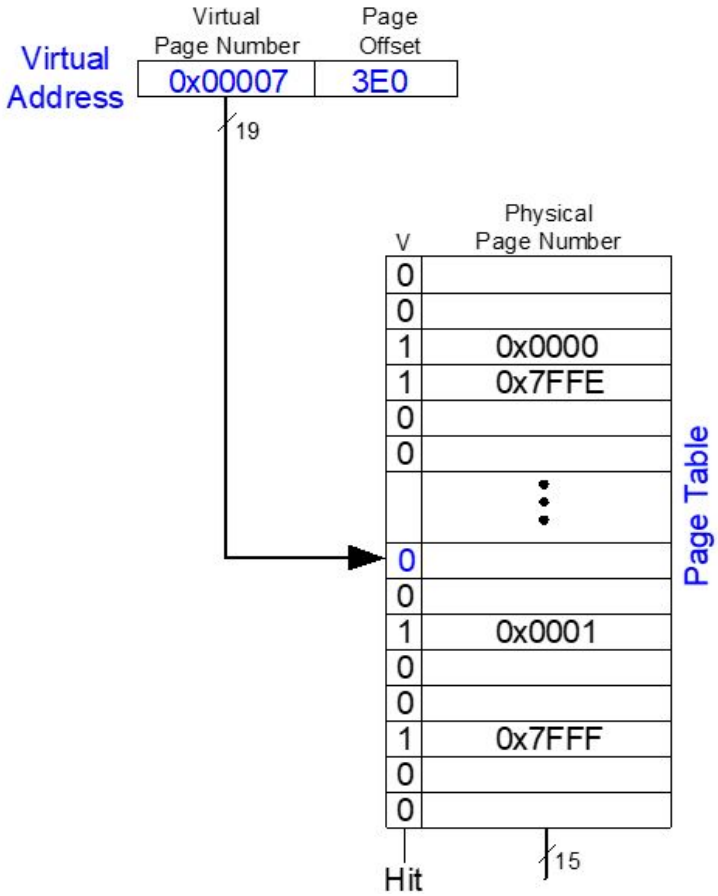
What is the physical address of virtual address **0x247C**?

- The 12-bit page offset requires no translation.
- The remaining 19 bits of the virtual address are the virtual page number, 0x2, and give the index into the page table.
- The page table maps virtual page 0x2 to physical page 0x7FFF.
- Hence, virtual address 0x247C maps to physical address 0x7FFF47C.
- The least significant 12 bits are the same in both the physical and the virtual address.



Page Table Example 2

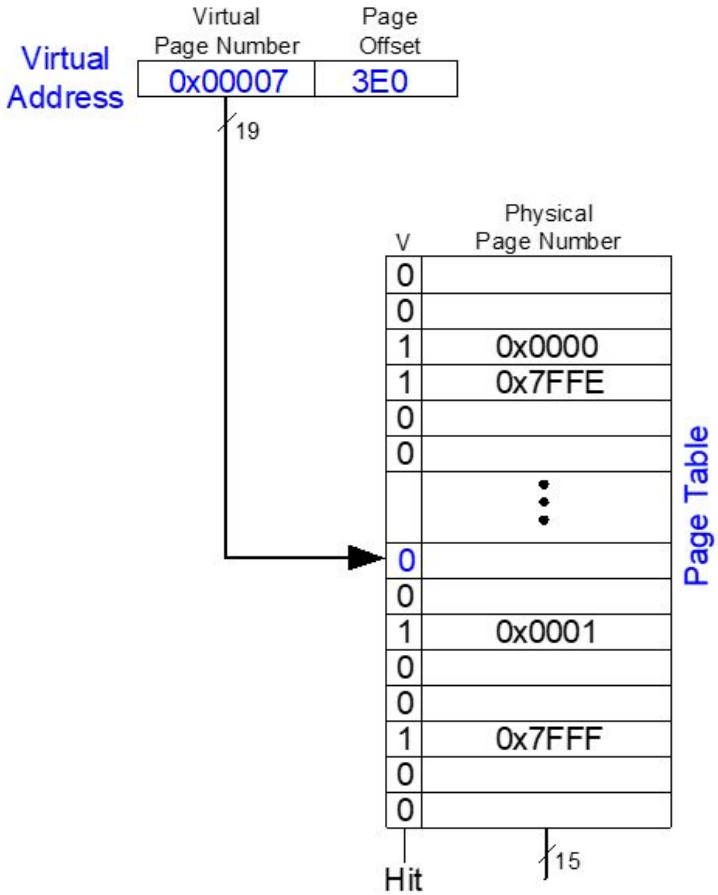
What is the physical address of virtual address **0x73E0**?



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)

Virtual memory would have a severe performance impact if it required a page table read on every load or store, doubling the delay of loads and stores.

- But page table accesses have great temporal locality.
- The temporal and spatial locality of data accesses and the large page size mean that many consecutive loads or stores are likely to reference the same page.
- Therefore, if the processor remembers the last page table entry that it read, it can probably reuse this translation without rereading the page table.
- In general, the processor can keep the last several page table entries in a small cache called a translation lookaside buffer (TLB).
- The processor “looks aside” to find the translation in the TLB before having to access the page table in physical memory.

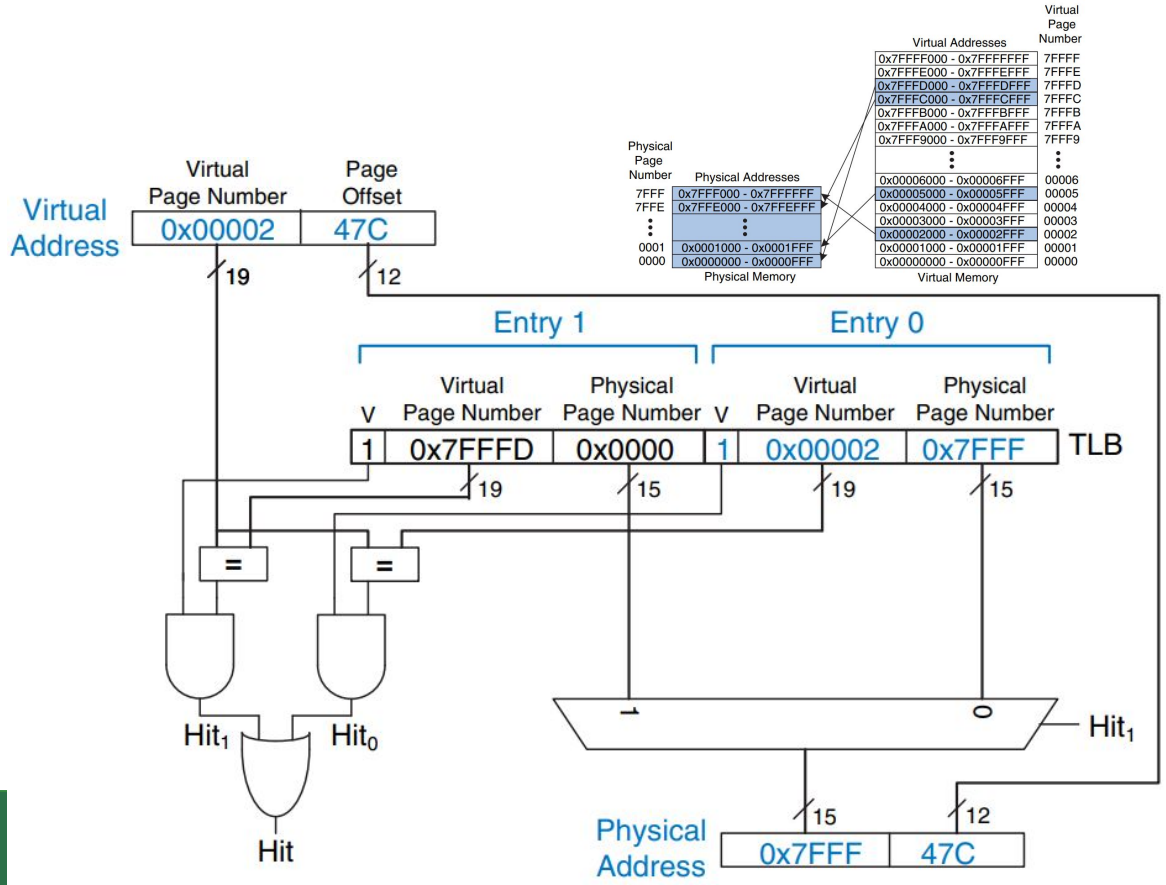
Translation Lookaside Buffer (TLB)

- 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



- The two-entry TLB with the request for virtual address 0x247C.
- The TLB receives the virtual page number of the incoming address, 0x2, and compares it to the virtual page number of each entry. Entry 0 matches and is valid, so the request hits.
- The translated physical address is the physical page number of the matching entry, 0x7FFF, concatenated with the page offset of the virtual address.
- As always, the page offset requires no translation. The request for virtual address 0x5FB0 misses in the TLB. So, the request is forwarded to the page table for translation.

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

No program should be able to access another program's memory without permission. This is called memory protection.

Virtual memory systems provide memory protection by giving each program its own virtual address space.

Virtual Address Space

- Each program can use as much memory as it wants in that virtual address space, but only a portion of the virtual address space is in physical memory at any given time.
- Each program can use its entire virtual address space without having to worry about where other programs are physically located.
- However, a program can access only those physical pages that are mapped in its page table.
- In this way, a program cannot accidentally or maliciously access another program's physical pages, because they are not mapped in its 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**