MODULE-6

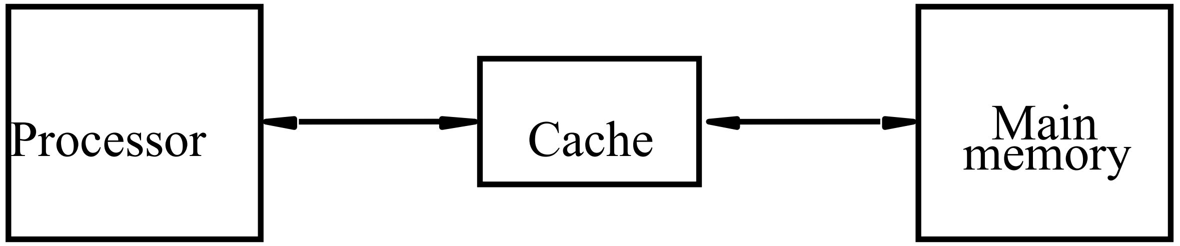
CACHE MEMORY, VIRTUAL MEMORY

Cache Memories

The Memory System

Cache Memories

- Processor is much faster than the main memory.
  - As a result, the processor has to spend much of its time waiting while instructions and data are being fetched from the main memory.
  - Major obstacle towards achieving good performance.
- Speed of the main memory cannot be increased beyond a certain point.
- Cache memory is an architectural arrangement which makes the main memory appear faster to the processor than it really is.(to increase the effective speed of the memory system)
- Cache contains a copy of portions of main memory.
- Cache memory is based on the property of computer programs known as “locality of reference”.



Locality of Reference

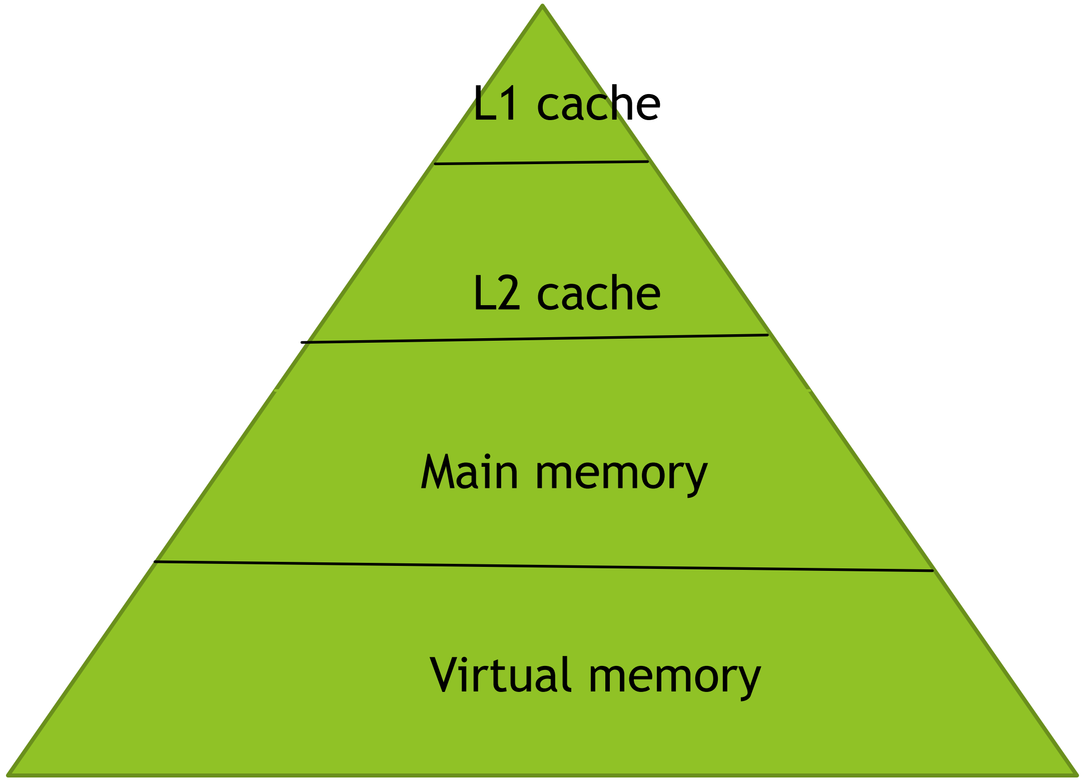
- Analysis of programs indicates that many instructions in localized areas of a program are executed repeatedly during some period of time, while the others are accessed relatively less frequently.
  - These instructions may be the ones in a loop, nested loop or few procedures calling each other repeatedly.
  - This is called "locality of reference".
- Temporal locality of reference:
  - Recently executed instruction is likely to be executed again very soon. This occurs when a program loop is executed, the same set of instructions are referenced and fetched repeatedly.
- Spatial locality of reference:
  - Instructions with addresses close to a recently instruction are likely to be executed soon. This occurs as the instructions are stored in consecutive memory locations.

Levels of cache

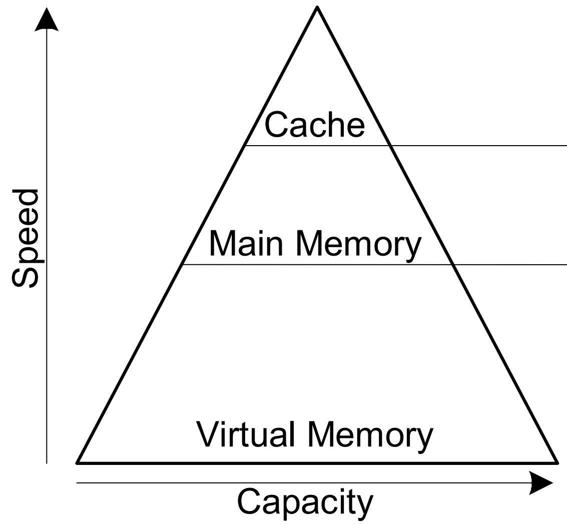
- To decrease memory time, low miss rate

L1 –level-1 cache memory built onto microprocessor chip(small,fast)

L2 –level-2 cache memory is on a separate chip (large,slower than L1)

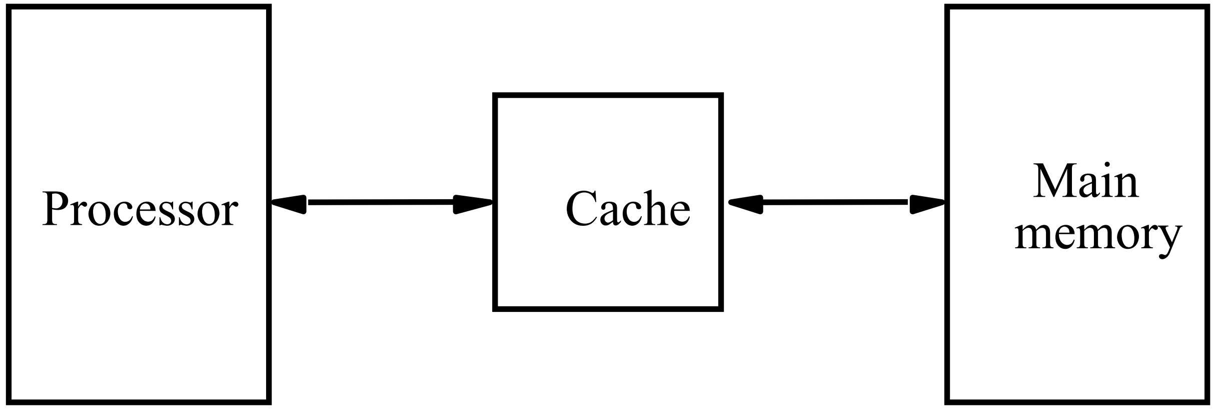


Memory Hierarchy



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

Cache memories



- *Processor issues a Read request, a block of words is transferred from the main memory to the cache, one word at a time.*
- *Subsequent references to the data in this block of words are found in the cache.*
- *At any given time, only some blocks in the main memory are held in the cache. Which blocks in the main memory are in the cache is determined by a “mapping function”.*
- *When the cache is full, and a block of words needs to be transferred from the main memory, some block of words in the cache must be replaced. This is determined by a “replacement algorithm”.*

Cache hit

- *Existence of a cache is transparent to the processor. The processor issues Read and Write requests in the same manner.*
- *If the data is in the cache it is called a Read or Write hit.*
- *Read hit:*
  - *The data is obtained from the cache.*
- *Write hit:*
  - *Cache has a replica of the contents of the main memory.*
  - *Contents of the cache and the main memory may be updated simultaneously each time CPU writes into cache. This is the write-through protocol.Main memory always contain the same version of data as the cache contains.*
  - *Update only the contents of the cache during a write operation, and mark it as updated by setting a bit known as the dirty bit or modified bit(flag). The contents are copied to the main memory whenever this block is replaced. This is write-back or copy-back protocol.*

Cache miss

- *If the data is not present in the cache, then a <u>Read miss or Write miss</u> occurs.*

- *Read miss:*
  - *Block of words containing this requested word is transferred from the memory.*
  - *After the block is transferred, the desired word is forwarded to the processor.*
  - *The desired word may also be forwarded to the processor as soon as it is transferred without waiting for the entire block to be transferred. This is called <u>load-through or early-restart.</u>*

- *Write-miss:*
  - *Write-through protocol is used, then the contents of the main memory are updated directly.*
  - *If write-back protocol is used, the block containing the addressed word is first brought into the cache. The desired word is overwritten with new information.*

Cache Design Questions

- What data is held in the cache?

Ideally, cache anticipates needed data and puts it in cache from main memory.

But impossible to predict future. Use past to predict future –exploits temporal and spatial locality:

  - **Temporal locality:** copy recently accessed data into cache
  - **Spatial locality:** copy neighboring data into cache too. If 1 word is fetched ,group of words are fetched(cache block),**B=C(capacity)/b(block size)**

- How is data found?

Cache organized into *S* sets

Each memory address maps to exactly one set

Caches categorized by no: of blocks in a set:

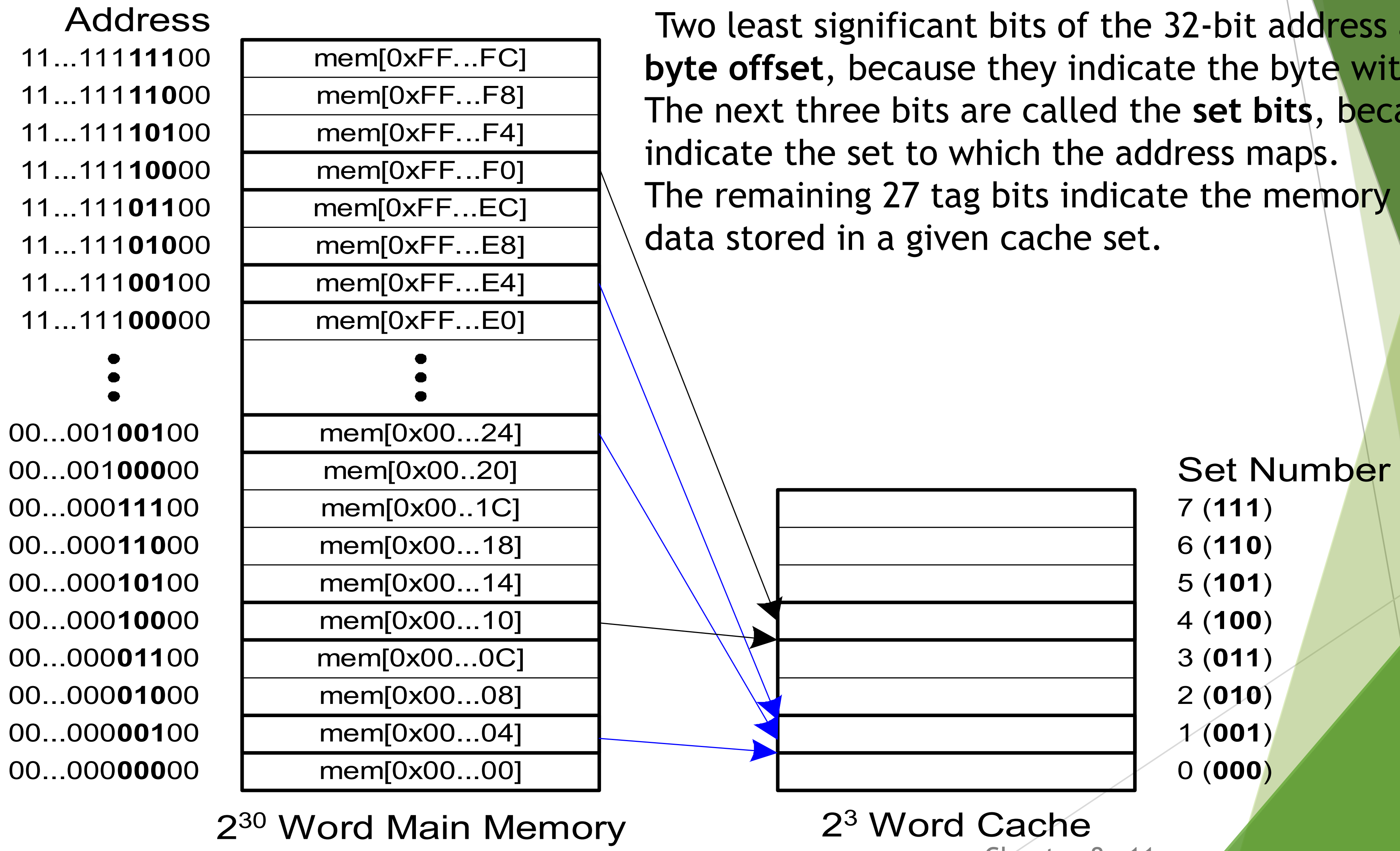
  - **Direct mapped:** 1 block per set(S=B)
  - ***N*-way set associative:** *N* blocks per set(S=B/N)
  - **Fully associative:** all cache blocks in 1 set(S=1)

- What data is replaced to make room for new data when the cache is full?

Least-recently used way in the set

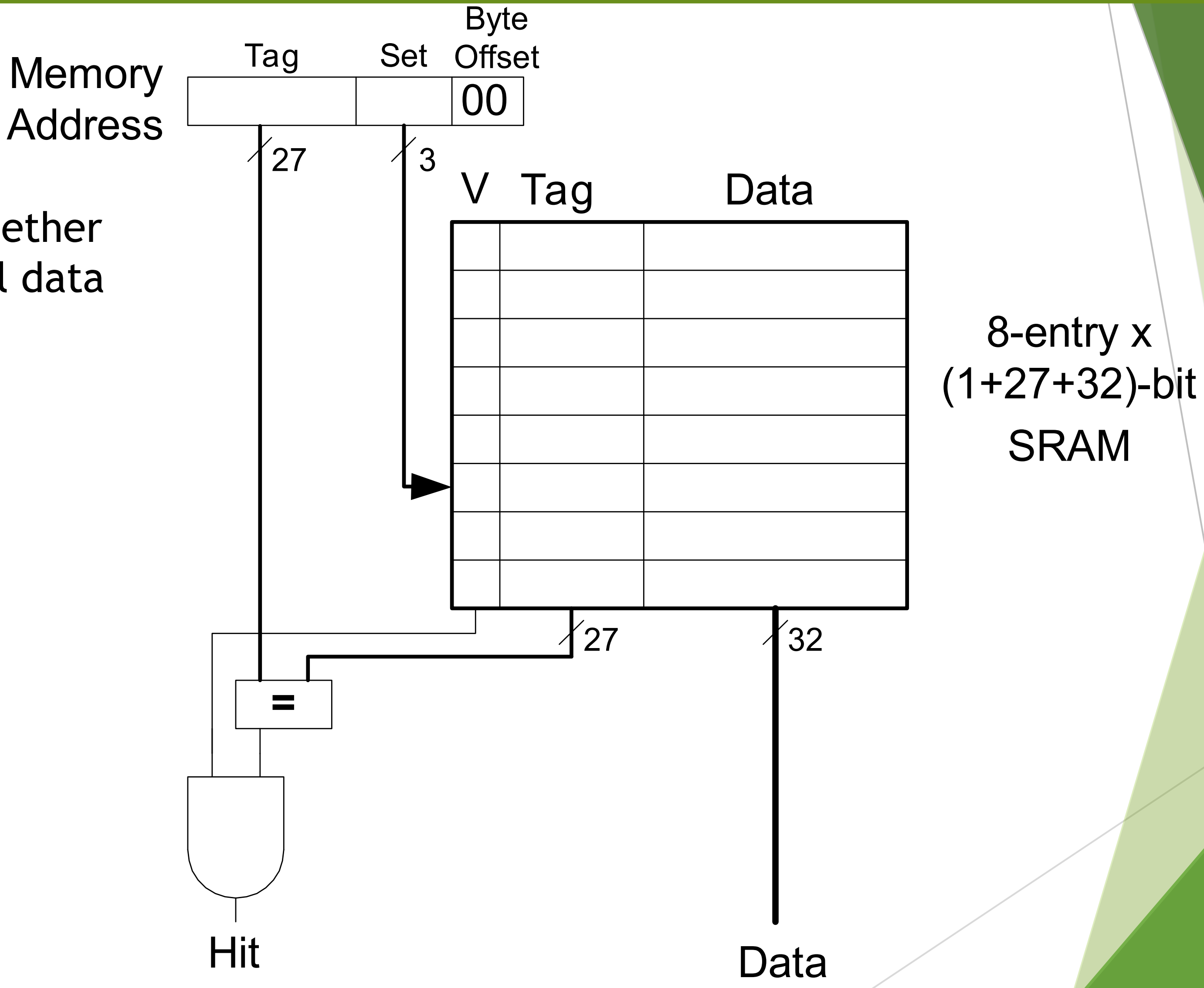
Direct Mapped Cache

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.



Hardware

V-valid bit –indicate whether the set hold meaningful data



Examples:-

1.Find the number of set and tag bits for a direct mapped cache with 1024 sets and a one-word block size. The address size is 32 bits.

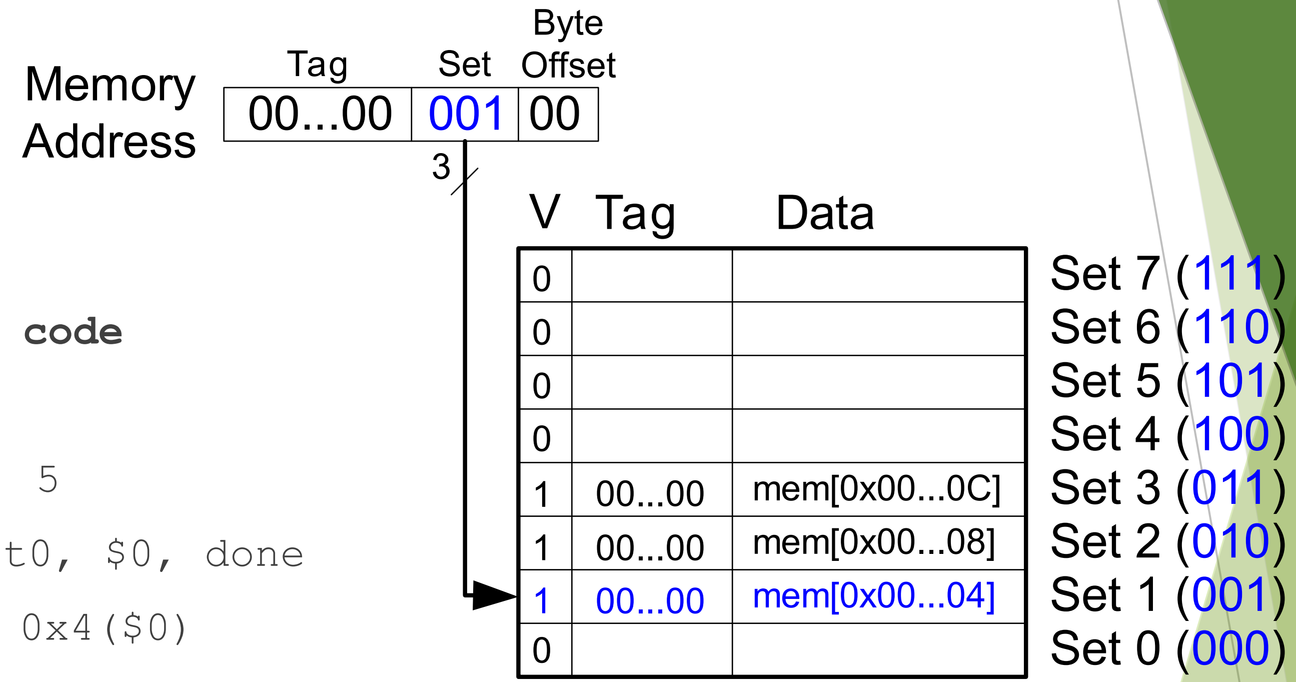
Solution: A cache with 2^10 sets requires $\log_2(2^{10}) = 10$ set bits. The two least significant bits of the address are the byte offset, and the remaining 32 - 10 - 2 = 20 bits form the tag.

2.Suppose a computer has 4K(2^12) main memory and 1K cache memory.To address a word in main memory,12 bit is needed.To address a word in cache memory ,10 bit is needed.

Solution:10 bit set and 12-10=2 bit tag

Performance

```
# MIPS assembly code

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



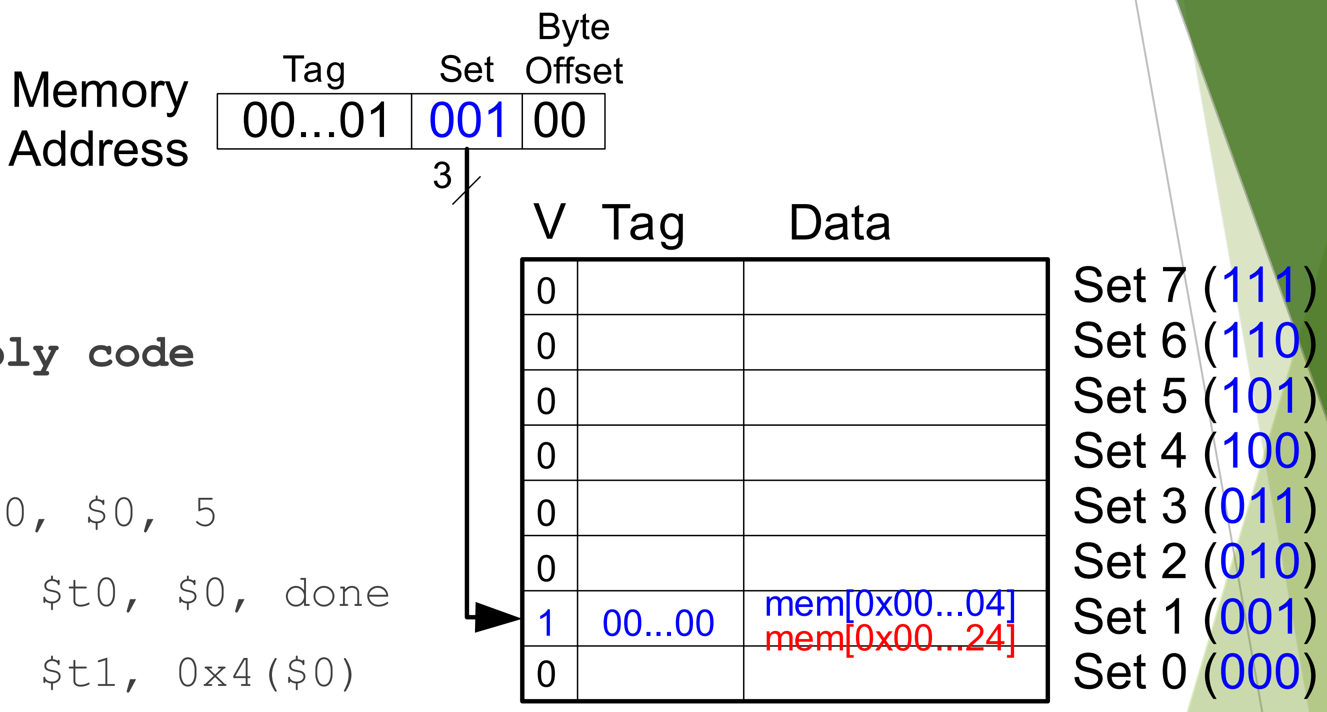
Miss Rate = 3/15

= 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 = 10/10
= 100%

Conflict Misses

N-Way Set Associative Cache

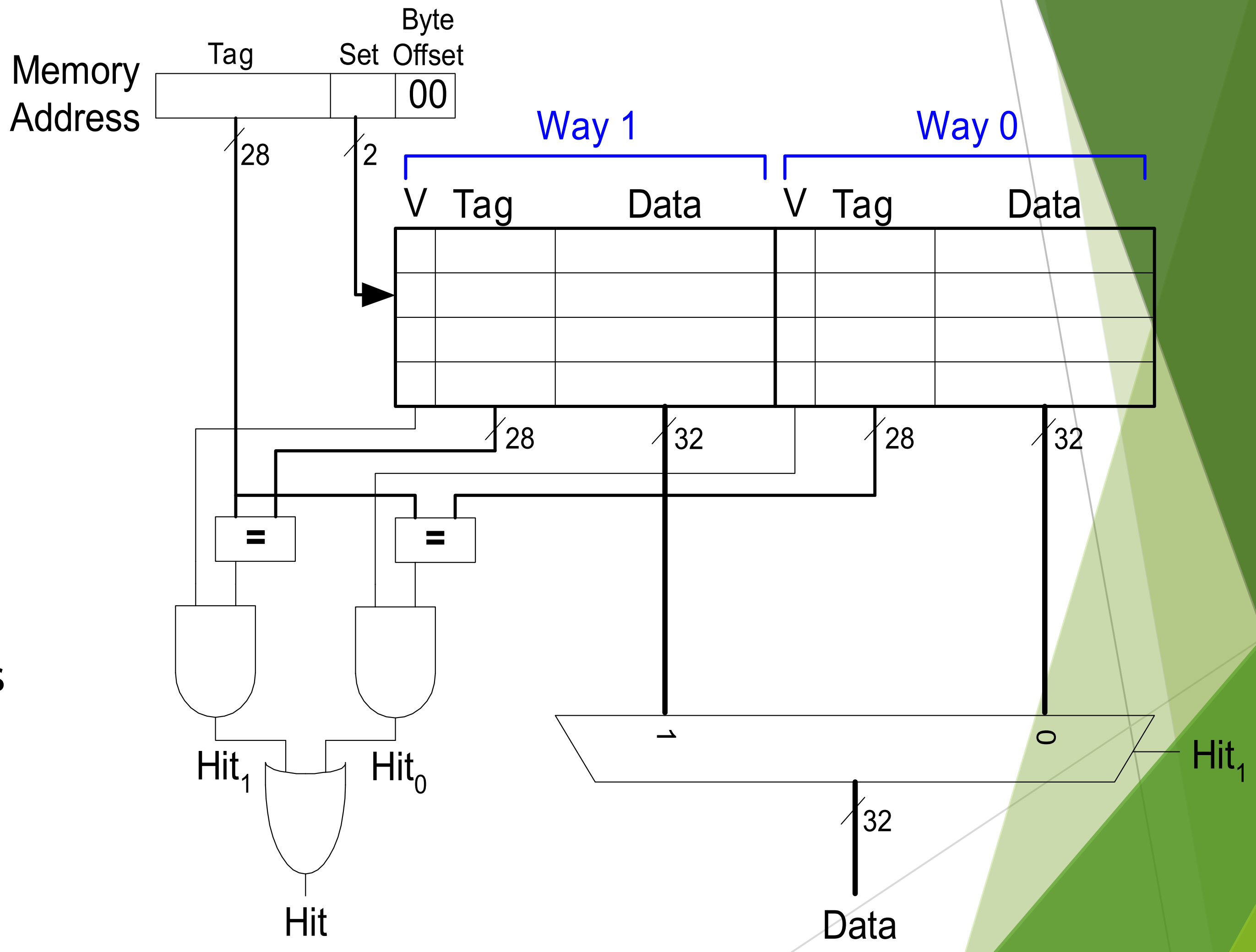
An N-way set associative cache reduces conflicts by providing N blocks in each set where data mapping to that set might be found.

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.

Low miss rate,expensive to build,slower.



Performance

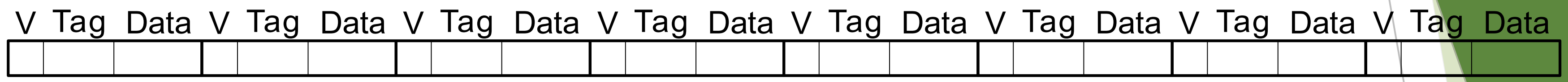
```
# MIPS assembly code

       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	
0			0			Set 3
0			0			Set 2
1	00...10	mem[0x00...24]	1	00...00	mem[0x00...04]	Set 1
0			0			Set 0

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

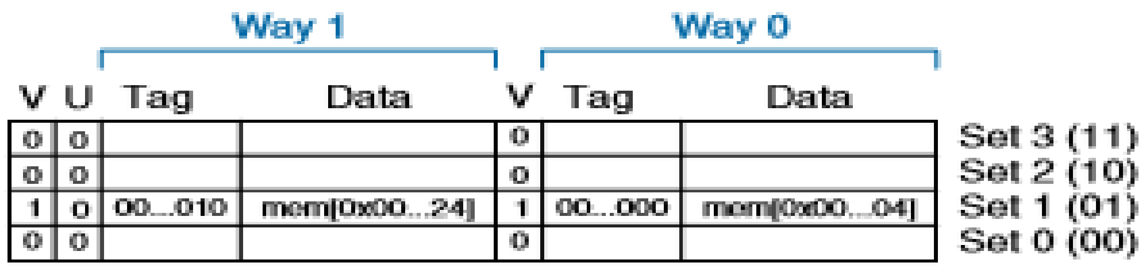
Replacement Methods:-

- In a direct mapped cache, each address maps to a unique block and set. If a set is full when new data must be loaded, the block in that set is replaced with the new data.
- In set associative and fully associative caches, the cache must choose which block to evict when a cache set is full. Temporal locality says LRU is used because it is least likely to be used again soon.
- 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.
- Most common replacement algorithms used are:
- Random replacement
- FIFO
- LRU
- 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.

Show the contents of an eight-word two-way set associative cache after executing the following code. Assume LRU replacement, a block size of one word, and an initially empty cache.

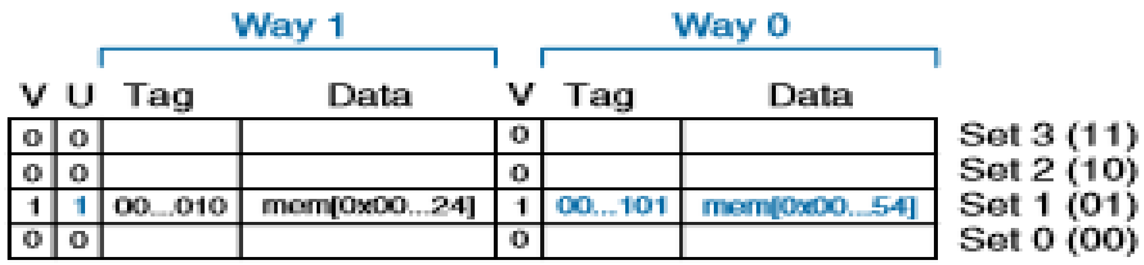
```
lw $t0, 0x04($0)
lw $t1, 0x24($0)
lw $t2, 0x54($0)
```

Solution: The first two instructions load data from memory addresses 0x4 and 0x24 into set 1 of the cache, shown in Figure 8.15(a). $U = 0$ 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, as shown in Figure 8.15(b), 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)

Figure 8.15 Two-way associative cache with LRU replacement

Virtual Memories

The Memory System

Virtual memories

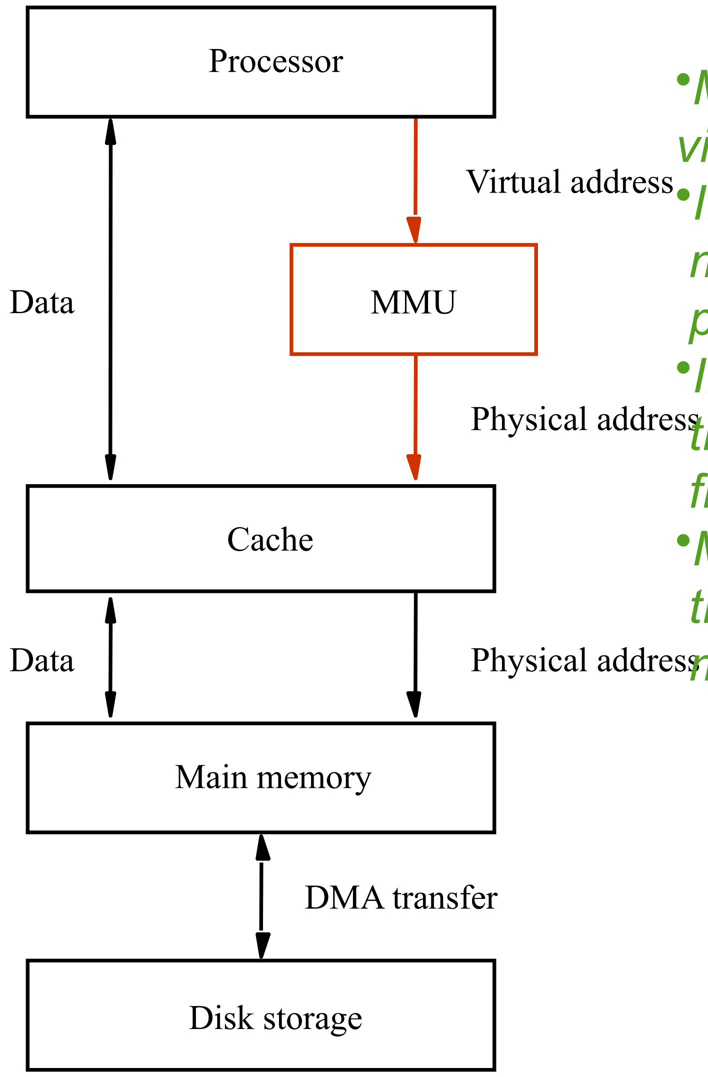
- Recall that an important challenge in the design of a computer system is to provide a large, fast memory system at an affordable cost.
- Architectural solutions to increase the effective speed and size of the memory system.
- Cache memories were developed to increase the effective speed of the memory system.
- Virtual memory is an architectural solution to increase the effective size of the memory system.
- Recall that the addressable memory space depends on the number of address bits in a computer.
  - For example, if a computer issues 32-bit addresses, the addressable memory space is 4G bytes.
- Physical main memory in a computer is generally not as large as the entire possible addressable space.
  - Physical memory typically ranges from a few hundred megabytes to 1G bytes.
- Large programs that cannot fit completely into the main memory have their parts stored on secondary storage devices such as magnetic disks.
  - Pieces of programs must be transferred to the main memory from secondary storage before they can be executed.

Virtual memories (contd..)

- When a new piece of a program is to be transferred to the main memory, and the main memory is full, then some other piece in the main memory must be replaced.
  - Recall this is very similar to what we studied in case of cache memories.
- Operating system automatically transfers data between the main memory and secondary storage.
  - Application programmer need not be concerned with this transfer.
  - Also, application programmer does not need to be aware of the limitations imposed by the available physical memory.
- Techniques that automatically move program and data between main memory and secondary storage when they are required for execution are called <u>virtual-memory</u> techniques.
- Programs and processors reference an instruction or data independent of the size of the main memory.
- Processor issues binary addresses for instructions and data.

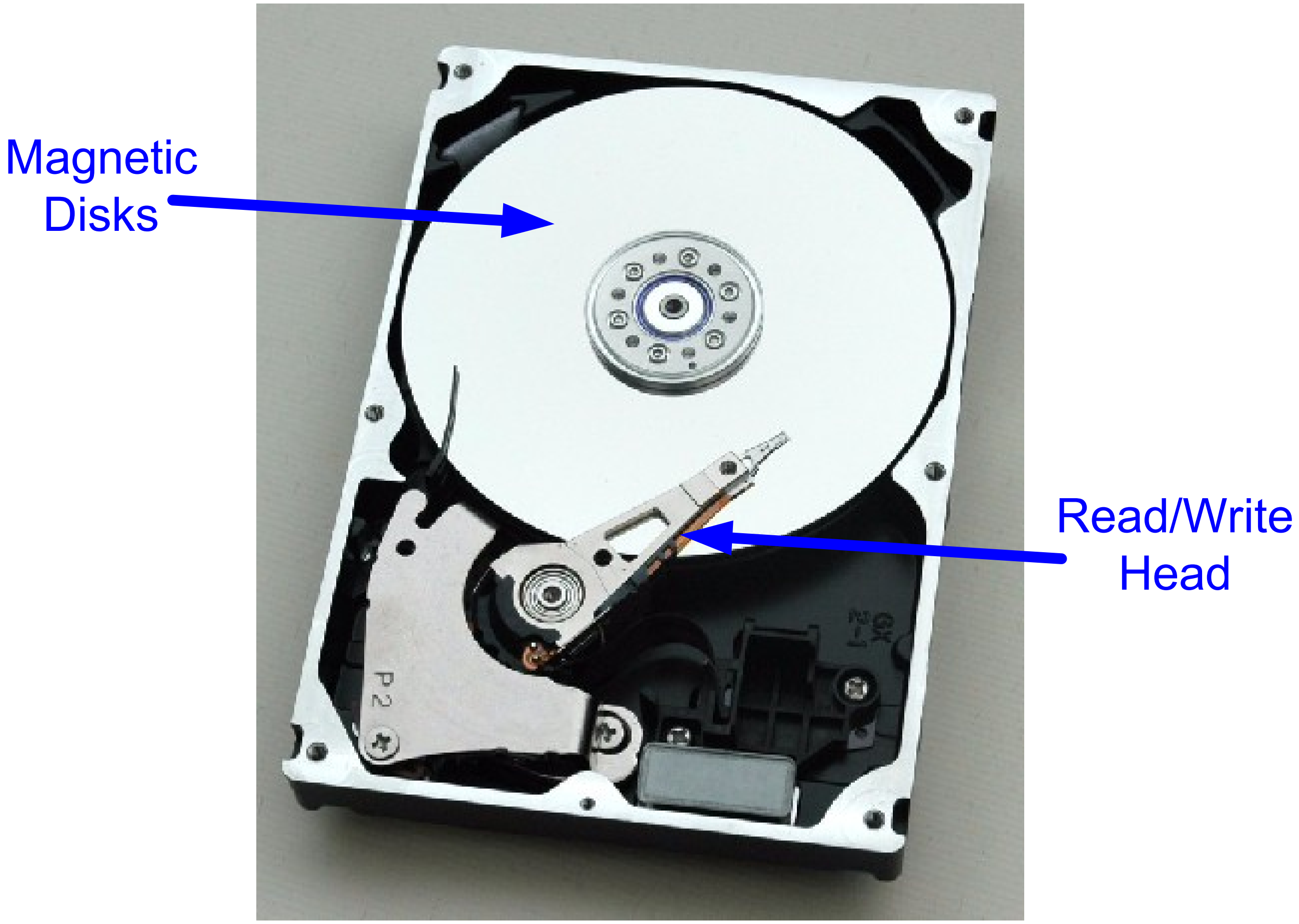
  These binary addresses are called logical or virtual addresses.
- Virtual addresses are translated into physical addresses by a combination of hardware and software subsystems.
  - If virtual address refers to a part of the program that is currently in the main memory, it is accessed immediately.
  - If the address refers to a part of the program that is not currently in the main memory, it is first transferred to the main memory before it can be used.

Virtual memory organization



- *Memory management unit (MMU) translates virtual addresses into physical addresses.*
- *If the desired data or instructions are in the main memory they are fetched as described previously.*
- *If the desired data or instructions are not in the main memory, they must be transferred from secondary storage to the main memory.*
- *MMU causes the operating system to bring the data from the secondary storage into the main memory.*

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

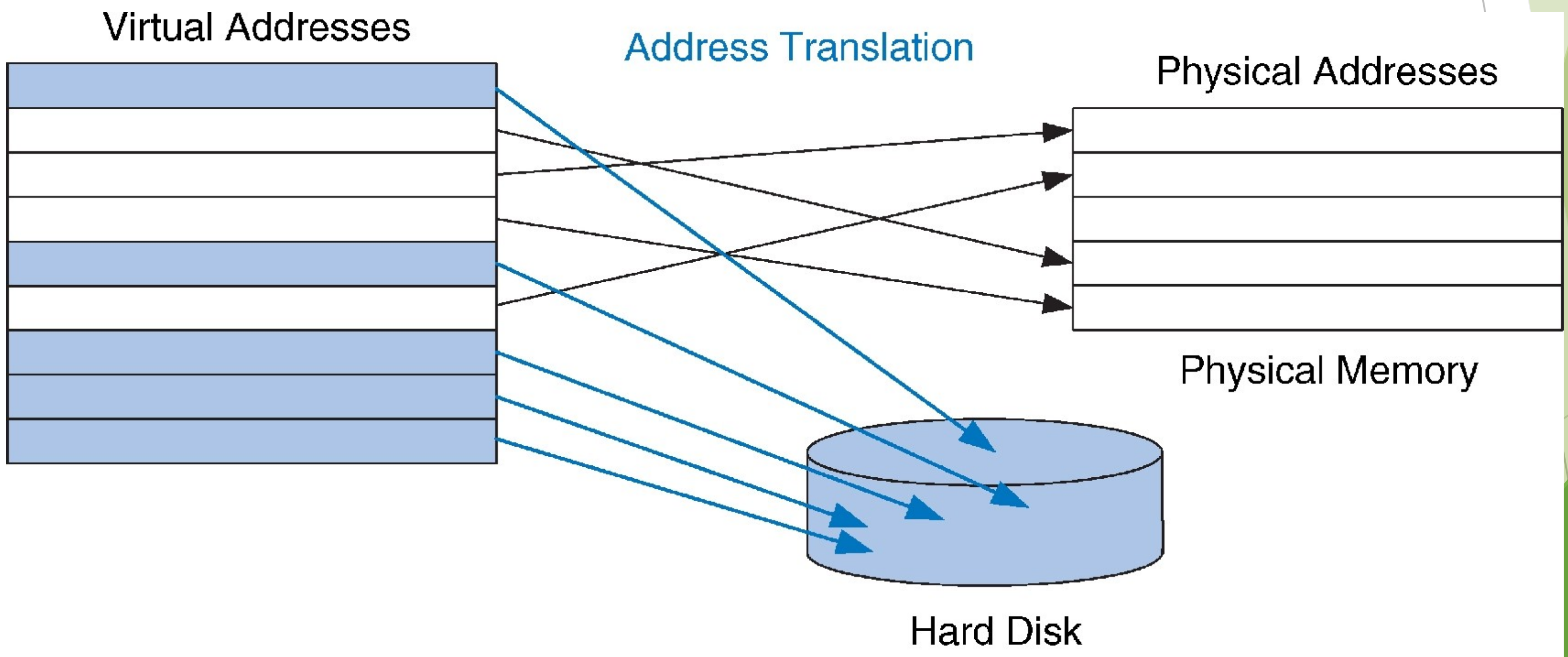
Cache/Virtual Memory Analogues

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

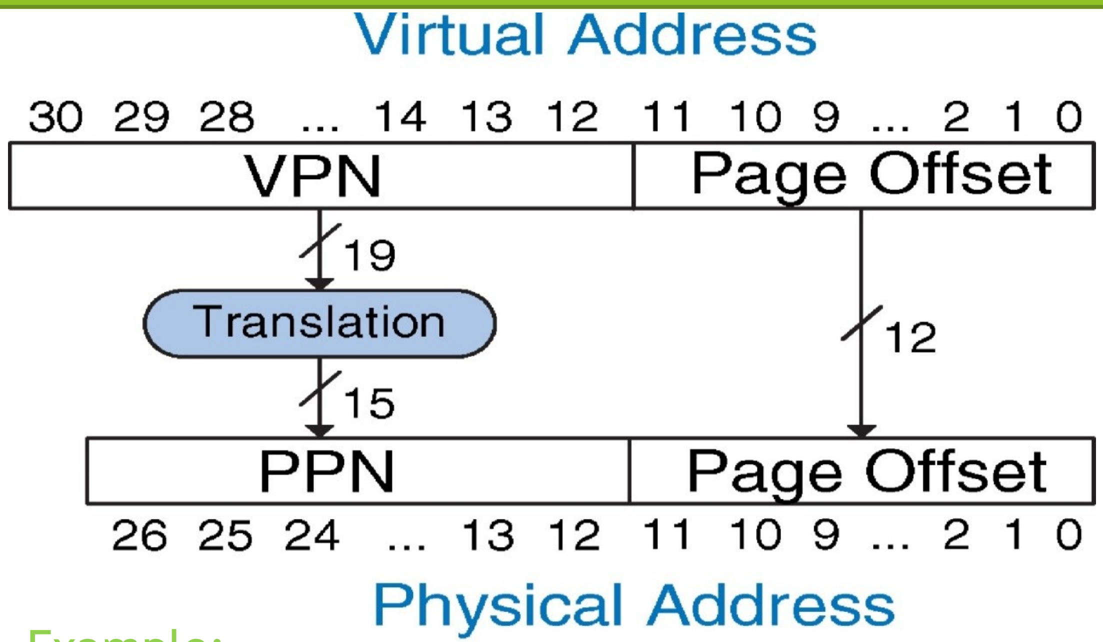
Physical memory acts as cache for virtual memory

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



Address Translation



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

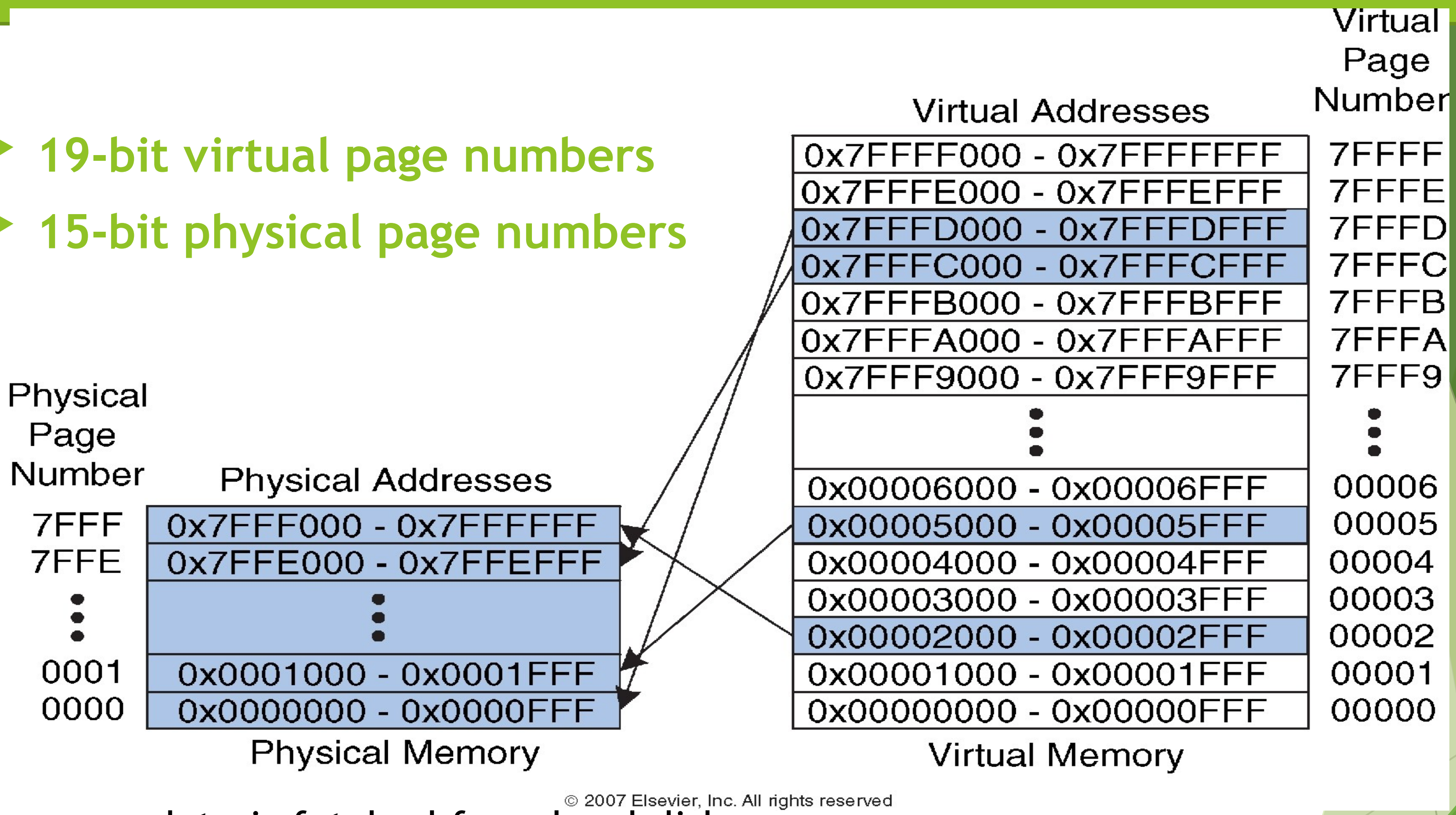
Organization:--Virtual address: 31 bits
Physical address: 27 bits
Page offset: 12 bits
# Virtual pages = $2^{31}/2^{12}$ = $\mathbf{2^{19}}$ (VPN = 19 bits)
# Physical pages = $2^{27}/2^{12}$ = $\mathbf{2^{15}}$ (PPN = 15 bits)

Continued……….

- 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 on the disk.
- 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. The page table is so large that it is 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.

Virtual Memory Example

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

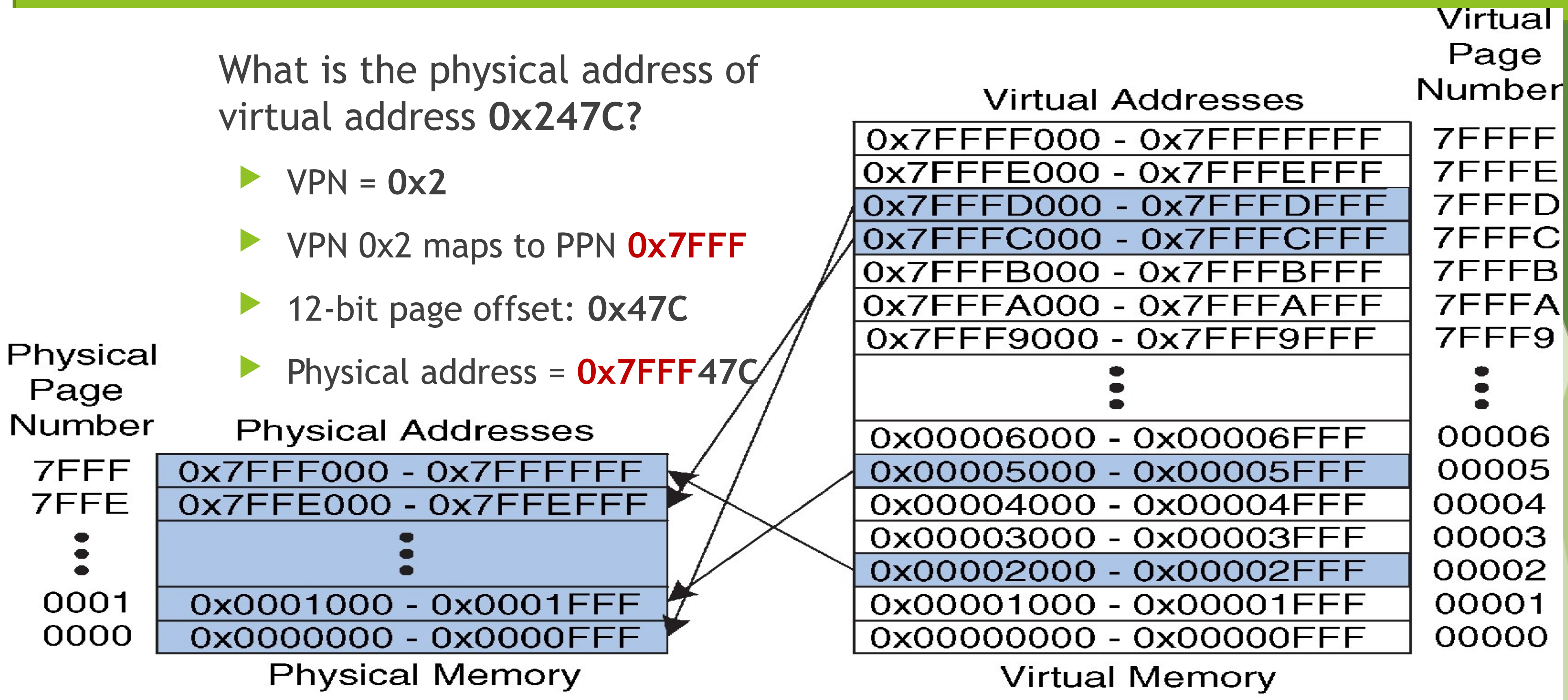


If a page fault occurs,data is fetched from hard disk.
The MSB of the virtual or physical address specify the virtual or physical page number. The LSB specify the word within the page and are called the page offset.
Physical memory can only hold up to 1/16th of the virtual pages at any given time. The rest of the virtual pages are kept on disk.

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**

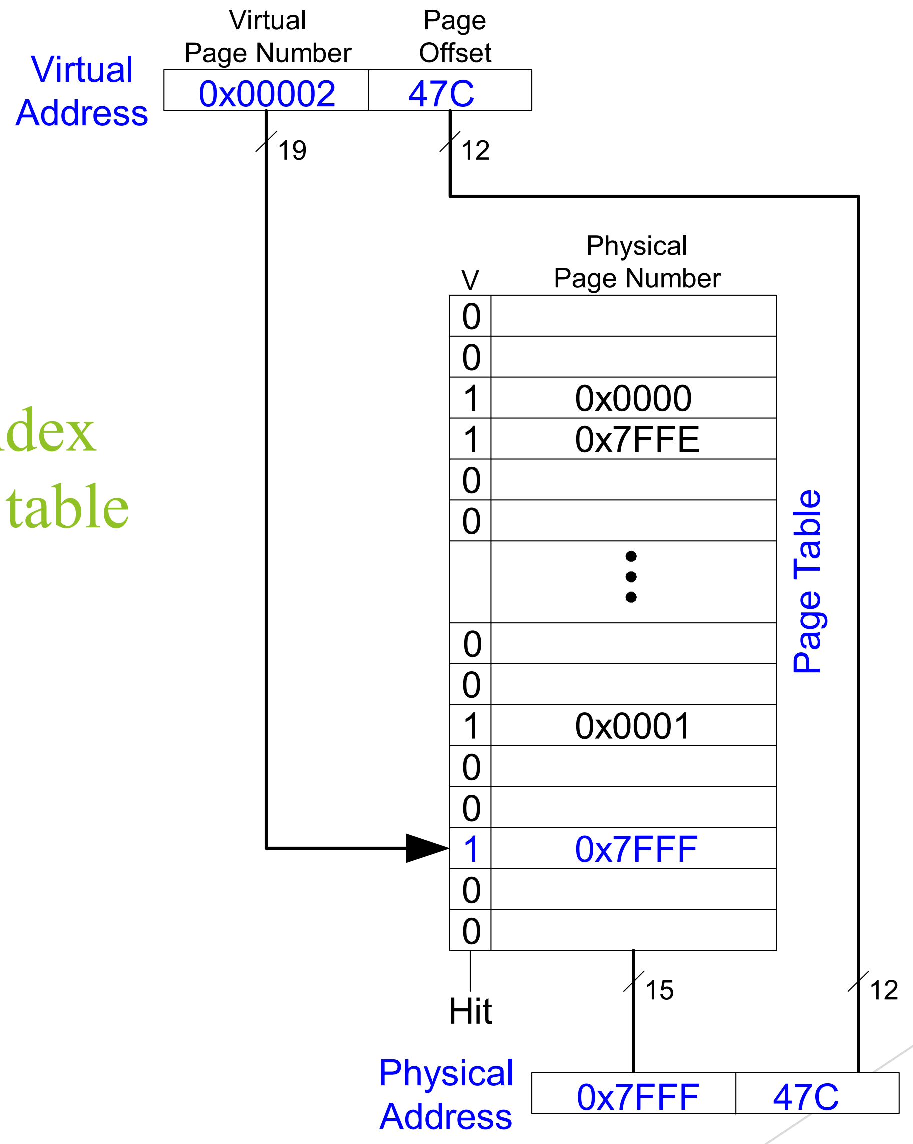


Example:---Virtual address 0x53F8 (an offset of 0x3F8 within virtual page 5) maps to physical address 0x13F8 (an offset of 0x3F8 within physical page 1).

- The least significant 12 bits of the virtual and physical addresses are the same (0x3F8) and specify the page offset.
- Only the page number needs to be translated to obtain the physical address from the virtual address.

Page Table Example

VPN is index into page table



Page Table Example 1

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

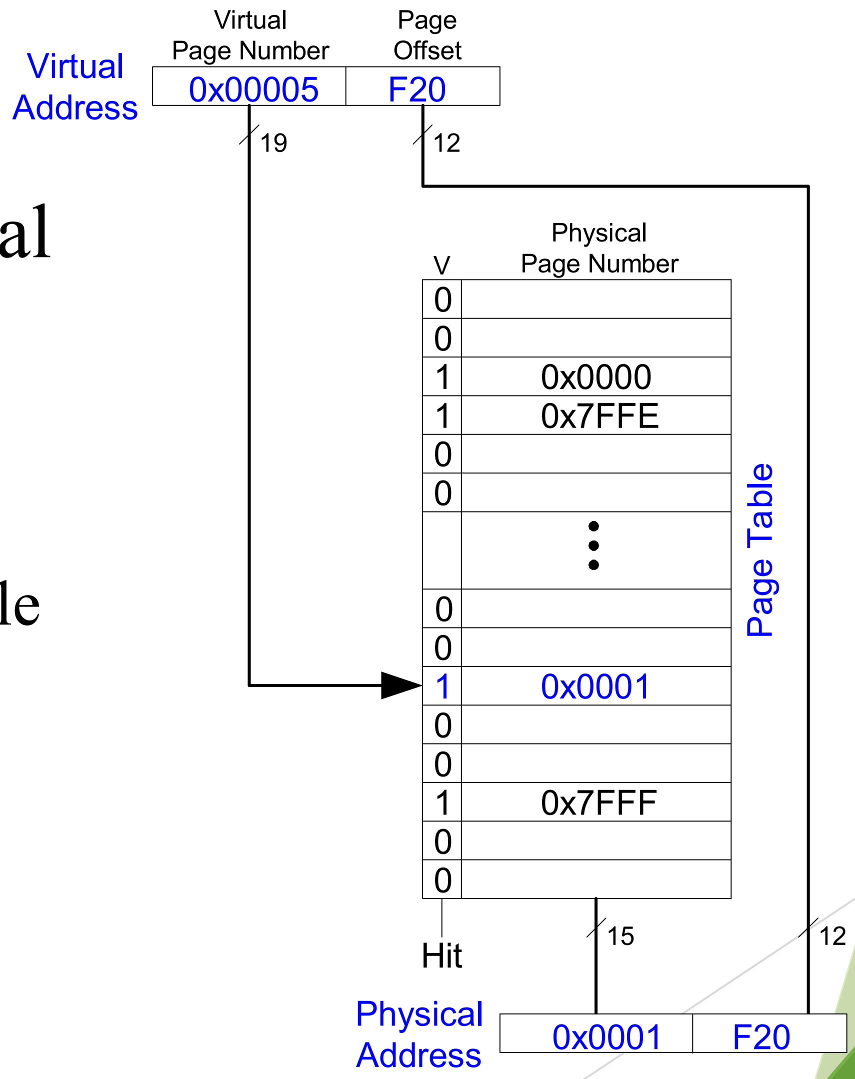
V	Physical Page Number
0	
0	
1	0x0000
1	0x7FFE
0	
0	
	⋮
0	
0	
1	0x0001
0	
0	
1	0x7FFF
0	
0	

Page Table

Page Table Example 1

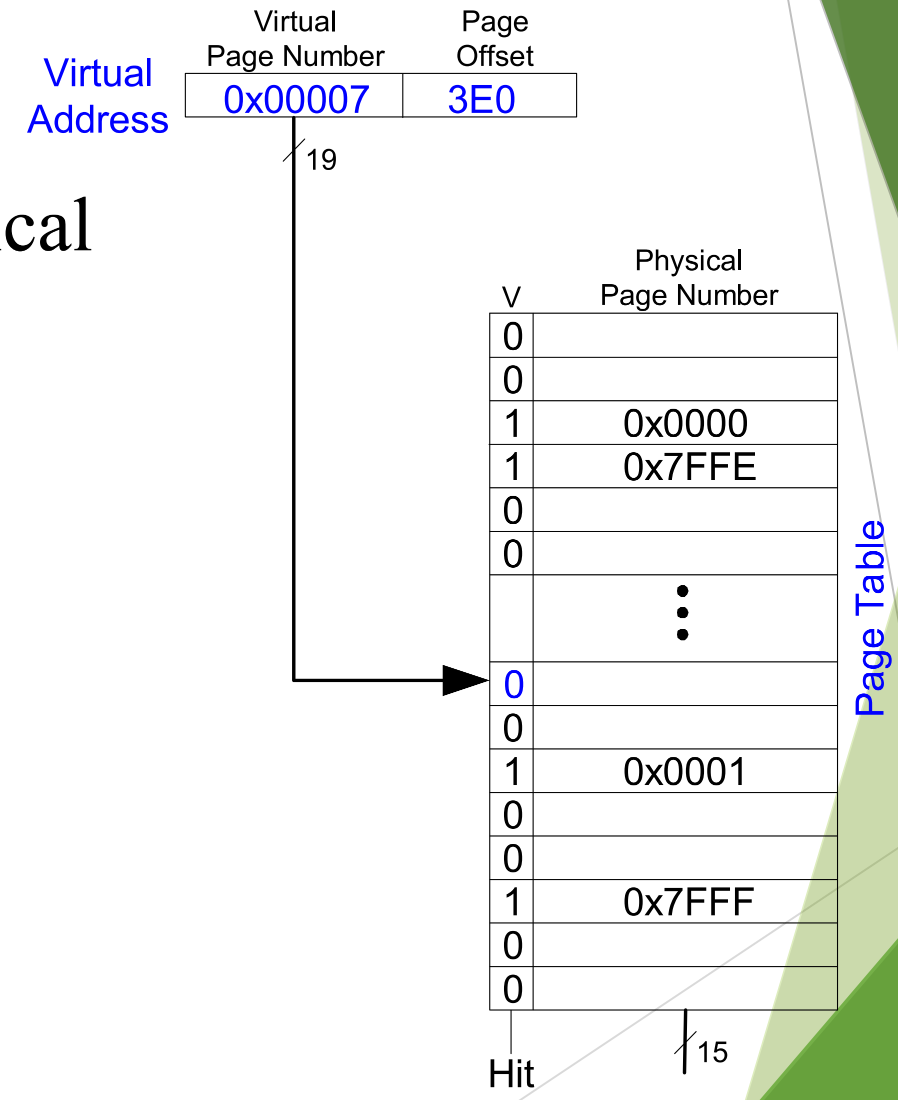
What is the physical address of virtual address **0x5F20**?

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



Page Table Example 2

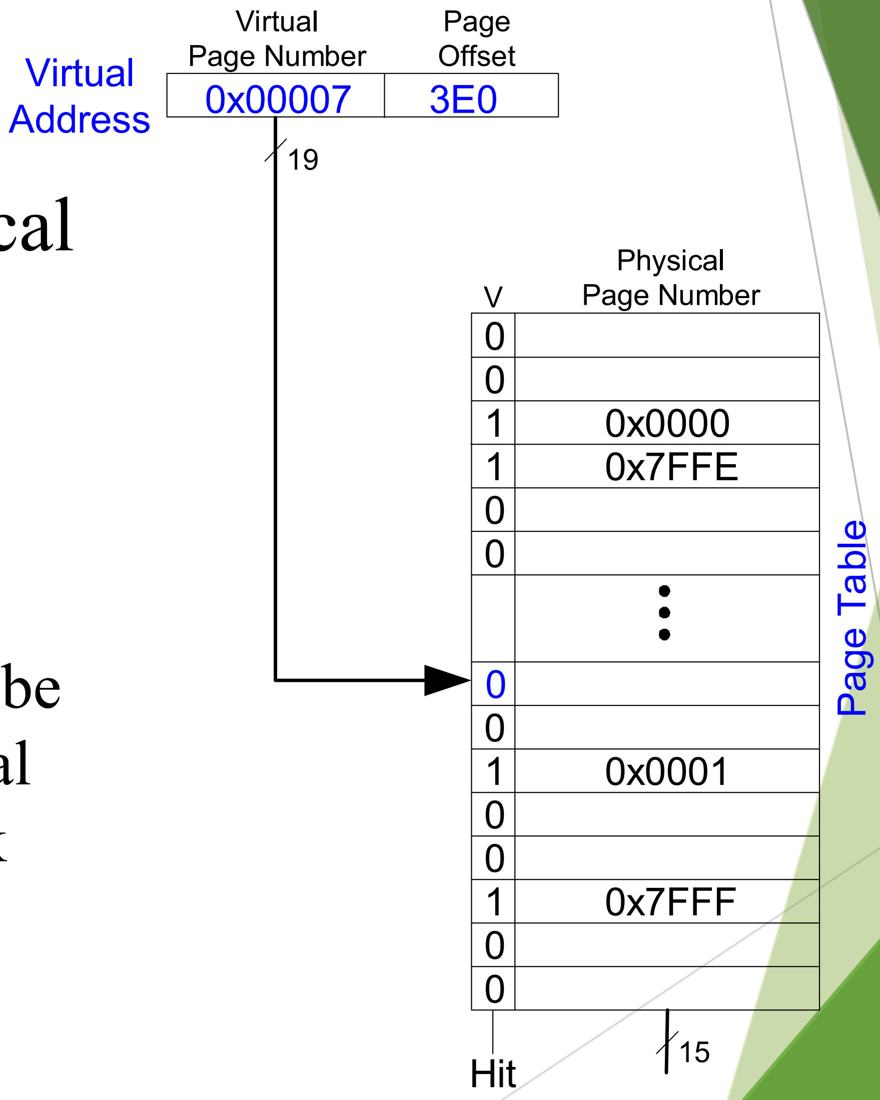
What is the physical address of virtual address **0x73E0**?



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



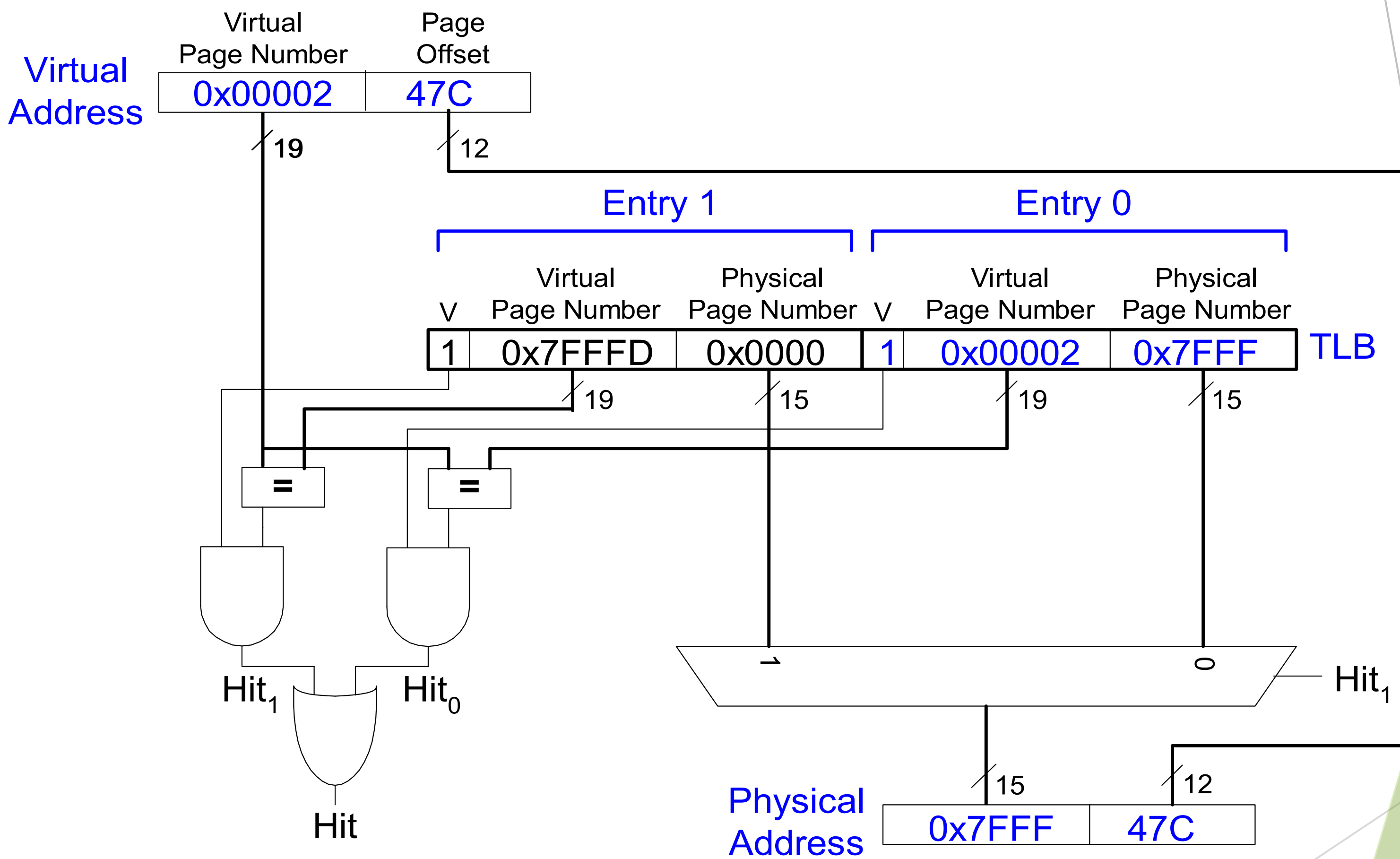
Page Table Challenges

- **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.

TLB-Translation Look aside Buffer

- 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).
- Reduces no:of memory accesses for *most* loads/stores from 2 to 1
- Each TLB entry holds a virtual page number and its corresponding physical page number.
- The TLB is accessed using the virtual page number. If the TLB hits, it returns the corresponding physical page number.
- A **TLB** speeds up address translation.
- TLB
  - Small: accessed in < 1 cycle
  - Typically 16 - 512 entries
  - Fully associative
  - > 99 % hit rates typical

Example 2-Entry TLB



Segmentation

- Memory Management Technique-memory is divided into variable sized chunks which can be allotted to processes.
- Each chunk is called Segment.It is a set of logically related instruction or data element associated with given name.
- Segments are generated by programmer or OS
- It is another memory protection method

Differences:

Segmentation	Paging
Program is divided into variable size segments	Program is divided into fixed size pages
User is responsible for division	Division performed by OS
Slower than paging	Paging is Faster
Visible to user	Invisible to user
Eliminates internal fragmentation	Suffers from internal fragmentation
Page numbers,offset is used to calculate absolute address	Segment number,offset is used to calculate absolute address
Variable length,2- dimension address	Fixed length,1-dimension address