

# Memory

## Where do we keep data?

- Database applications
  - Web server
  - Digital library
- Scientific computations
  - Bioinformatics
- Multimedia applications
  - Games
- Need large and fast storage

## Computer System

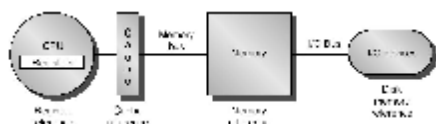


- **Computation system: Processor (central processor unit)**
  - Add numbers, test numbers, signal I/O devices to active
    - Control: determine operations of datapath, memory, I/O devices
    - Datapath: perform arithmetic operations
- **Memory system:**
  - Main memory
  - Secondary memory
- **I/O system: Secondary and tertiary storage I/O, network I/O**

## Memories: Review

- **Static random access memory (SRAM):**
  - very fast but takes up more space than DRAM
- **Dynamic random access memory (DRAM):**
  - very small but slower than SRAM

## Memory Hierarchy

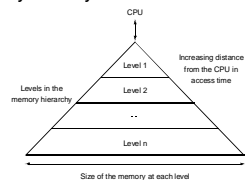


Level	1	2	3	4
Called	Registers	Cache	Main memory	Disk storage
Typical size	~1 KB	~4 MB	~1 GB	~1 TB
Implementation technology	Custom memory with multiple ports, SRAM or DRAM	On-chip or off-chip SRAM	CMOS DRAM	Magnetic disk
Access time (ns)	2-5	1-10	10-100	5,000,000
Bandwidth (in MB/sec)	8000-12,000	400-500	400-2000	4-32
Managed by	Compiler	Hardware	Operating system	Operating system/user
Backed by	Cache	Main memory	Disk	Tape

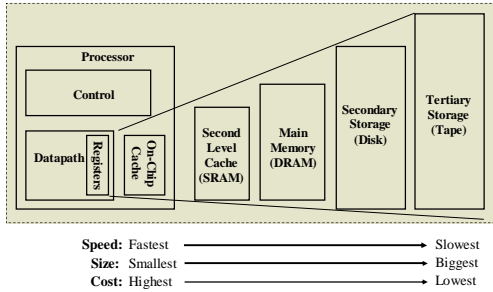
## Exploiting Memory Hierarchy

- **Users want large and fast memories! – cost??**  
 SRAM access times are .5 – 5ns at cost of \$4000 to \$10,000 per GB.  
 DRAM access times are 50-70ns at cost of \$100 to \$200 per GB.  
 Disk access times are 5 to 20 million ns at cost of \$.50 to \$2 per GB.

- **Try and give it to them anyway**
  - build a memory hierarchy



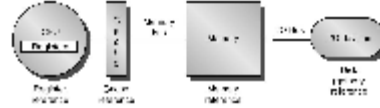
## Memory Hierarchy



7

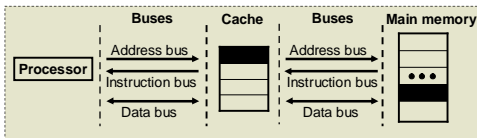
## Memory System Components

- Storage components: Store information
  - Registers
  - TLB, Caches
  - Main memory
- Communication components: Communicate information
  - On- and off-chip buses
  - I/O buffers and signaling/driving circuitry
  - I/O pads
  - Pins



8

## Memory Accesses



- Higher level: subset of lower level
  - e.g., cache: subset of main memory
  - Block: minimum unit present or not present in the two-level hierarchy
- Memory accesses: load and store
  - First step: referenced address
  - 32-bit bus (e.g., 0000 0000 0000 0000 0000 0000 1000<sub>1</sub>)
  - Second step: load instructions or data; store data
- Hit and miss
  - Hit: instruction/data requested appears
  - Miss: instruction/data requested not found

9

## Locality

- A principle that makes having a memory hierarchy a good idea
- If an item is referenced,
  - temporal locality: it will tend to be referenced again soon
  - spatial locality: nearby items will tend to be referenced soon
- Our initial focus: two levels (upper, lower)
  - block: minimum unit of data
  - hit: data requested is in the upper level
  - miss: data requested is not in the upper level
- Reading
  - 7.3 Measuring and Improving Cache Performance

10

## Locality

- Temporal locality: it will tend to be referenced again soon
- Spatial locality: nearby items will tend to be referenced soon
  - For example
    - Same instructions are accessed again and again

```

Loop: lw $t0, 0($t1)    # $t0 = A[i]      ← PC
      addi $t3, $t0, $t2 # $t3 = $t0 + $t2 ← PC + 4
           # $t3 = A[i] + C
      sw $t3, 0($t1)    # A[j] = $t3 = A[i] + C
      addi $t1, $t1, 4  # $t1 = $t1 + 4
           # i++
      bne $t1, $t4, Loop # $t1 = ? = $t4
    
```

11

## Cache

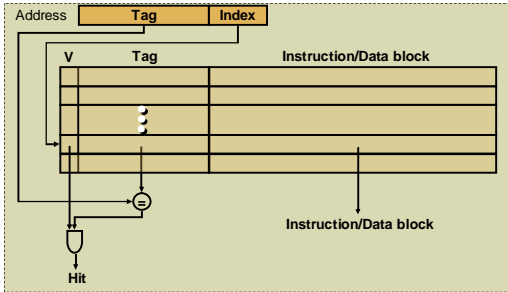
- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- Our first example:
  - block size is one word of data
  - "direct mapped"

For each item of data at the lower level, there is exactly one location in the cache where it might be.

e.g., lots of items at the lower level share locations in the upper level

12

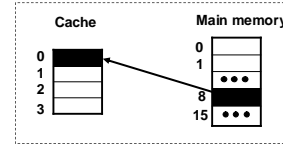
### Cache Logic



13

### Cache Block Placement

- Block position in cache
  - $(\text{Block number}) \bmod (\text{Number of cache blocks})$

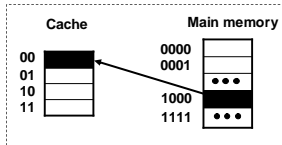


Number of cache blocks: 4  
 Memory block number: 8  
 Block position in cache:  $(8) \bmod (4) = 0$

14

### Cache Block Placement

- Block position in cache
  - Lower order bits used for block number

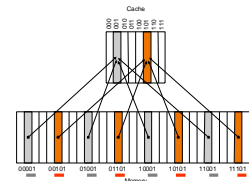


Number of cache blocks: 4  $\rightarrow$  need 2 bits  
 Memory block number: 1000  
 Block position in cache: 1000  $\rightarrow$  00

15

### Direct Mapped Cache

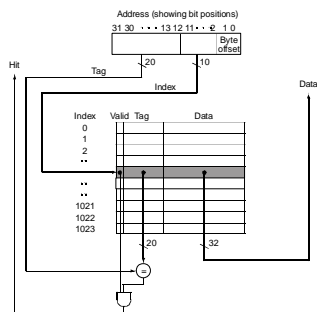
- Mapping: address is modulo the number of blocks in the cache
- $(\text{Block number}) \bmod (\text{Number of cache blocks})$



16

### Direct Mapped Cache

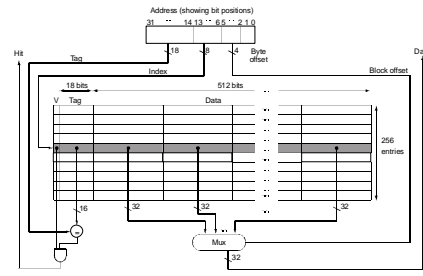
- For MIPS:



17

### Direct Mapped Cache

- Taking advantage of spatial locality: 16KB caches contains 256 blocks with 16 words per block



18

## Hits vs. Misses

- **Read hits**
  - this is what we want!
- **Read misses**
  - stall the CPU, fetch block from memory, deliver to cache, restart
- **Write hits:**
  - can replace data in cache and memory (write-through)
    - to avoid cache and memory inconsistent
  - write the data only into the cache (write-back the cache later)
    - Write-through: at least 100 cpu cycles → slow down processor  
SPEC CPU2000 INT: 10% store instructions  
Solution → Write Buffer
- **Write misses:**
  - read the entire block into the cache, then write the word

19

## Improve Performance By Increasing Cache Bandwidth

- **Split caches or combined cache?**
  - For example
    - Total cache size: 32 KB
    - Split cache miss rate: 3.24%
    - Combined cache miss rate: 3.18%
      - Better, but not much
  - No fixed entries for instructions and data
- **Why use split caches?**
  - Improve bandwidth to reduce miss penalty – improve performance
    - For example
      - Double bandwidth by supporting both an instruction and data access simultaneously
- We can not use miss rate as sole measure of cache performance

20

## Design Memory System to Support Caches

- Miss penalty for accessing memory
  - Address + Accessing memory + Instruction/Data Transferring
- Clock rate of bus is much slower than processor
  - By as much as a factor of 10
- For example:
  - 1 memory bus clock cycle to send address
  - 15 memory bus clock cycles for each DRAM access initiated
  - 1 memory bus clock cycle to send a word of data

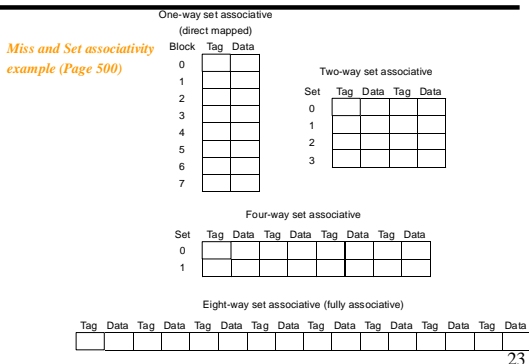
21

## Reduce Cache Misses by More Flexible Placement of Blocks

- **Direct mapped**
  - One entry for a block
  - (block #) modulo (# of cache blocks)
- **Set associative**
  - n-way set-associative: n locations for a block
  - (block #) modulo (# of cache sets)
- **Fully associative**
  - any entry for a block
  - Make search practical
    - Done in parallel with a comparator for each entry
    - Increase hardware complexity
- **Location of a memory block**
  - address is 12
  - cache with 8 blocks
  - direct mapped: (12) modulo (8) = 4
  - 4-way set associative: (12) modulo (2) = 0
  - fully associative: any blocks
  - P. 500
    - Where can a block be placed?
    - How is the block is found?
    - Which block should be replaced?

22

## Decreasing miss ratio with associativity



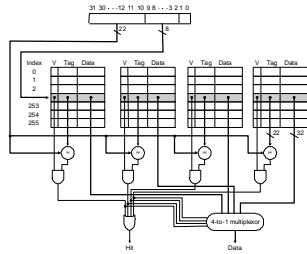
23

## Reduce Cache Misses by Replacement Policy

- First in first out (FIFO)
- Least recently used (LRU)

24

## An implementation



25

- Cache
    - 64 blocks
    - Block size 16 bytes
    - What block # does byte address 1200 map to?
  - (Block address) modulo (# of cache blocks)
  - $=(\text{Byte address}/\text{Bytes per block}) \text{ modulo } (\# \text{ of cache blocks})$
  - A block contains addresses
    - $\lfloor \text{Byte address}/\text{Bytes per block} \rfloor \times \text{Bytes per block}$
    - $\lfloor \text{Byte address}/\text{Bytes per block} \rfloor \times \text{Bytes per block} + (\text{Bytes per block} - 1)$
- $= \lfloor 1200/16 \rfloor \text{ modulo } 64 = 11$  {Block maps to [1200-1215]}
- ↑ block address

26

## Cache Size

- Number of entries in cache
- Each entry
  - Number of tag bits
  - Number of data bits
  - Valid bit
- Naming convention
  - Count only size of data
  - 32KB, 128KB, 256KB

27

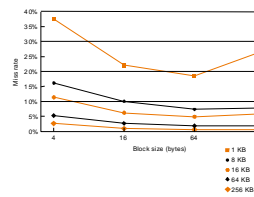
## Performance

- Simplified model:
  - execution time = (execution cycles + stall cycles) · cycle time
  - stall cycles = # of instructions · miss ratio · miss penalty
- Two ways of improving performance:
  - decreasing the miss rate
  - decreasing the miss penalty

28

## Performance

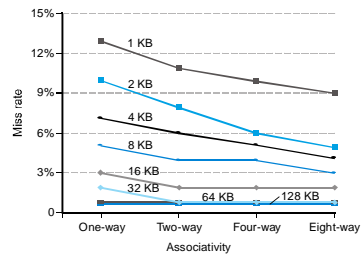
- Increasing the block size tends to decrease miss rate:



- Larger blocks exploit spatial locality to lower miss rate
  - Miss rate may go up eventually if block size becomes a *significant fraction* of cache size
    - Less blocks can be held in cache
    - Block might be replaced before many of its words are accessed
  - Increase Miss penalty

29

## Performance



30

## Decreasing miss penalty with multilevel caches

- **Multilevel Caches**
  - **Add a second level cache:**
    - often primary cache is on the same chip as the processor
    - use SRAMs to add another cache above primary memory (DRAM)
  - **miss penalty goes down if data is in 2nd level cache**
- **Using multilevel caches:**
  - try and optimize the hit time on the 1st level cache
  - try and optimize the miss rate on the 2nd level cache
- **Prefetching**
  - Global history buffer
- **Compression**

31

## Decreasing miss penalty with multilevel caches

- **Example:**
    - CPI of 1.0 on a 5 GHz machine with a 2% miss rate per instruction, 100ns DRAM access
    - Adding 2nd level cache with 5ns access time decreases miss rate to .5% per instruction
    - How much faster?
- miss penalty to main memory:**  $100\text{ns}/0.2=500$  clock cycles  
**Total CPI = Base CPI + Memory-stall cycles per instruction::**  
 $= 1.0 + 2\% * 500 = 11$
- miss penalty to second-level cache:**  $5\text{ns}/0.2 = 25$  clock cycles  
**Total CPI = Base CPI + Primary stalls per instruction + Secondary stalls per instruction**  
 $= 1.0 + 2\% * 25 + 0.5\% * 500 = 1 + 0.5 + 2.5 = 4.0$   
 $11/4=2.8$

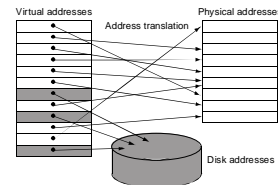
32

## Virtual Memory

33

## Virtual Memory – Address Mapping/Translation

- **Main memory can act as a cache for the secondary storage (disk)**



- **Advantages:**
  - illusion of having more physical memory
  - program relocation
  - protection

34

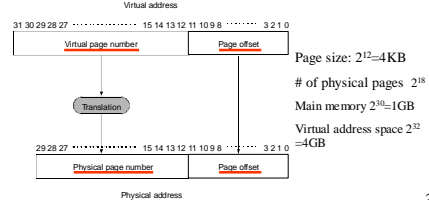
## Virtual memory

- **Multiple programs running at once on a computer**
  - **For each program**
    - Total memory required maybe be **much larger** than the amount of memory available in computer system
      - A fraction of memory is **actively being used** at any point in time
  - **Multiple programs**
    - Share memory
- **Virtual memory**
  - **Allow efficient and safe sharing of memory**
    - Keep active data for each program
  - **Remove burdens of small and limited amount of main memory**
- **Programs sharing memory change dynamically**
  - **Compile each program into its own address space**

35

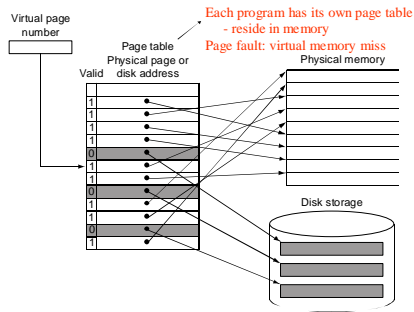
## Pages: virtual memory blocks

- **Relocation maps virtual addresses to different physical addresses**
- **Page faults:** the data is not in memory, retrieve it from disk
  - **Huge miss penalty, thus pages should be fairly large (e.g., 4KB)**
    - Millions of clock cycles
  - **Reducing page faults is important (LRU is worth the price)**
  - **using write-through is too expensive so we use writeback**



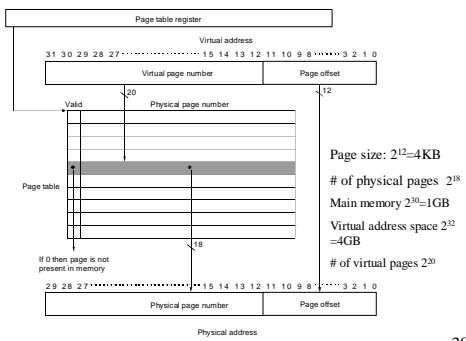
36

## Page Tables



37

## Page Tables



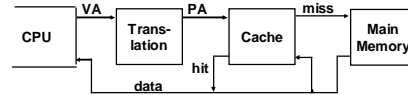
38

## virtual memory

- With a 32-bit virtual address, 4 KB pages, and 4 bytes per page table entry
    - Page table size:
      - # of page table entries =  $2^{32}/4KB=2^{28}/2^{12}=2^{16}$
      - Size of page table =
        - $2^{16}$  page table entries \* 2<sup>2</sup> bytes/page table entry = **4MB**
    - With a 64-bit virtual address:  $2^{64}/2^{12}=2^{52}$  Entries
    - 10 to 100 programs and fixed page tables – Memory??**
- Limit register – page table grows if necessary
    - only large when many pages of virtual address space used
  - Two limit register – page tables grow in two directions
  - Different data structure – hash table – more complex
  - Multi-level page tables
    - Two level -- higher order bits for first table (segment table)
      - next set of higher order bits for second table
    - Mapping – more complex
- Allow page tables to be paged

39

## Virtual Address and a Cache: Step backward???

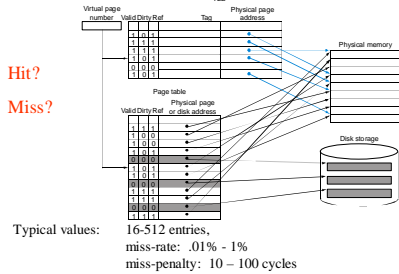


- Virtual memory seems to be really **slow**:
  - we have to access memory on every access – even cache hits!
  - Worse → if translation not completely in memory, may need to go to **disk** before hitting in cache!
- Solution: Caching!
  - Keep track of most common translations and place them in a “Translation Lookaside Buffer” (TLB)

40

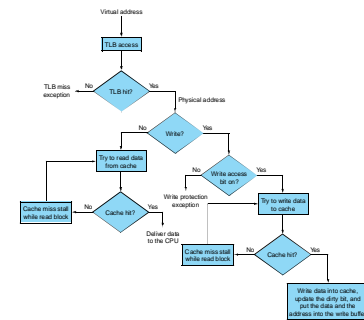
## Making Address Translation Fast

- Page tables in main memory → get physical address, access data
- A **cache** for address translations: **translation lookaside buffer (TLB)**



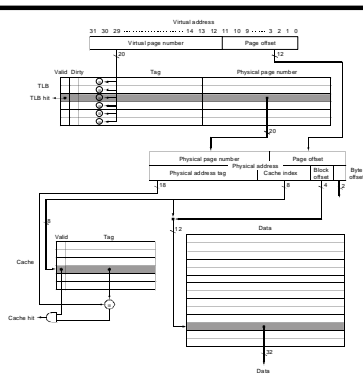
41

## TLBs and caches



42

## TLBs and Caches



43

## Advantages of Virtual Memory System

- **Translation:**
  - Program can be given consistent view of memory, even though physical memory is scrambled
  - Makes multithreading reasonable (now used a lot!)
  - Only the most important part of program ("Working Set") must be in physical memory.
  - Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later.
- **Protection:**
  - Different threads (or processes) protected from each other.
  - Different pages can be given special behavior
    - (Read Only, Invisible to user programs, etc).
  - Kernel data protected from User programs
  - Very important for protection from malicious programs
- **Sharing:**
  - Can map same physical page to multiple users ("Shared memory")

44