

## Reducing Miss Penalty

Five techniques:

- Read priority over write on miss
- Sub-block placement
- Early Restart and Critical Word First on miss
- Non-blocking caches (Hit Under Miss)
- Second level cache

## Miss Penalty Reduction: Second Level Cache

- L2 equations

$$\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1}$$

$$\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}$$

$$\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$$

- Definitions:

- *Local miss rate*: Misses in this cache divided by the total number of memory accesses to this cache ( $\text{Miss Rate}_{L2}$ )
- *Global miss rate*: Misses in this cache divided by the total number of memory accesses generated by the CPU ( $\text{Miss Rate}_{L1} \times \text{Miss Rate}_{L2}$ )

## Improving Hit Times

- Small and simple caches
- Avoiding address translation
- Pipelined writes
- Fast writes on misses via small sub-blocks

## Main Memory

- The second level of the memory hierarchy is typically the main memory (some systems have 2 levels of caches)
  - Main memory: implements the memory defined by the instruction set
  - Main memory is visible: there are instructions to manipulate the main memory
- Performance Measures
  - Bandwidth: bytes transferred per cycle
    - \* this measure is important for I/O
    - \* it is also important when cache block size is large (16 or more bytes)
  - Latency: time to get the data
    - \* access time: time between a read request and the arrival of data
    - \* cycle time: time between two successive read requests
    - \* the access time may be shorter than the cycle time

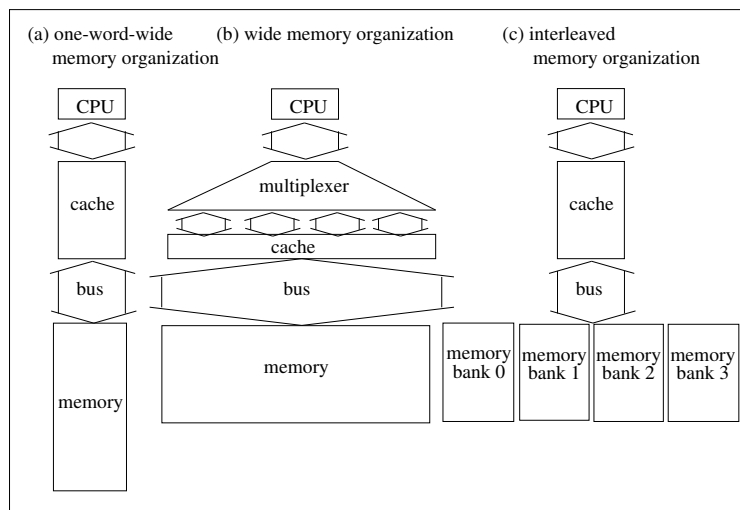
## Characteristics of RAM Technologies

- Static RAM (SRAM)
  - fast
  - expensive
  - requires no refresh
- Dynamic RAM (DRAM)
  - not as fast as static RAM
  - cheaper than static RAM
  - denser (per chip) than static RAM
  - requires refresh
  - address lines are usually multiplexed

Currently, caches typically use SRAM; main memory DRAM

## Improving Main Memory Performance

- Wider main memory
- Interleaved memory



## Using Wide Memory

- For illustration purposes assume that it takes
  - 1 cycle to place an address on the address bus
  - 6 cycles for the memory to perform a request
  - 1 cycle to obtain the memory data from the bus
  - Bandwidth assuming a 1-word bus:  $1/8$  word per cycle
- By using a 4 times wider bus and memory organization we can increase the memory bandwidth to  $1/2$  word per cycle, at extra hardware cost:
  - multiplexors between cache and CPU
  - cost of wider bus

## Using Interleaved Memories

- Split memory into multiple banks with successive word addresses falling in successive banks
- Example: split memory into 4 banks (each 1 word wide)
  - one memory address now yields 4 words
  - memory bus is kept 1 word wide
  - cost of a 4-word transfer:
    - \* 1 cycle to place address on bus
    - \* 6 cycles for memory access
    - \* 4 cycles for the 4 banks to reply
    - \* bandwidth =  $4/11$  words per cycle

## Main Memory

- The second level of the memory hierarchy is typically the main memory (some systems have 2 levels of caches)
  - Main memory: implements the memory defined by the instruction set
  - Main memory is visible: there are instructions to manipulate the main memory
- Performance Measures
  - Bandwidth: bytes transferred per cycle
    - \* this measure is important for I/O
    - \* it is also important when cache block size is large (16 or more bytes)
  - Latency: time to get the data
    - \* access time: time between a read request and the arrival of data
    - \* cycle time: time between two successive read requests
    - \* the access time may be shorter than the cycle time

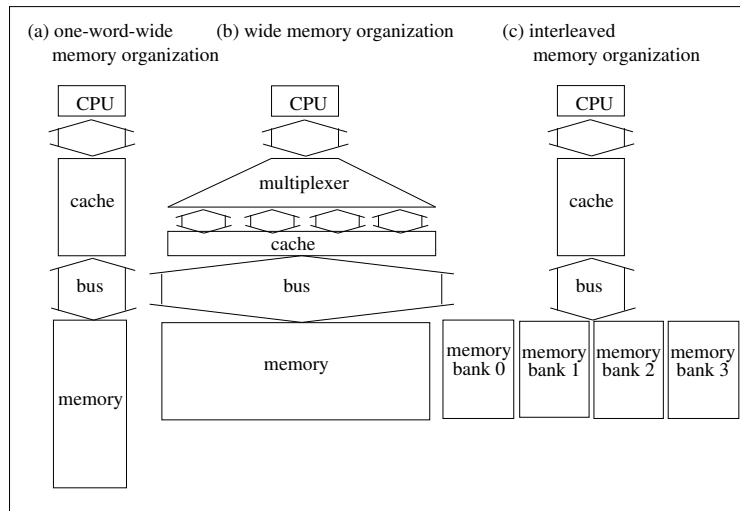
## Characteristics of RAM Technologies

- Static RAM (SRAM)
  - fast
  - expensive
  - requires no refresh
- Dynamic RAM (DRAM)
  - not as fast as static RAM
  - cheaper than static RAM
  - denser (per chip) than static RAM
  - requires refresh
  - address lines are usually multiplexed

Currently, caches typically use SRAM; main memory DRAM

## Improving Main Memory Performance

- Wider main memory
- Interleaved memory



## Using Wide Memory

- For illustration purposes assume that it takes
  - 1 cycle to place an address on the address bus
  - 6 cycles for the memory to perform a request
  - 1 cycle to obtain the memory data from the bus
  - Bandwidth assuming a 1-word bus:  $1/8$  word per cycle
- By using a 4 times wider bus and memory organization we can increase the memory bandwidth to  $1/2$  word per cycle, at extra hardware cost:
  - multiplexors between cache and CPU
  - cost of wider bus

## Using Interleaved Memories

- Split memory into multiple banks with successive word addresses falling in successive banks
- Example: split memory into 4 banks (each 1 word wide)
  - one memory address now yields 4 words
  - memory bus is kept 1 word wide
  - cost of a 4-word transfer:
    - \* 1 cycle to place address on bus
    - \* 6 cycles for memory access
    - \* 4 cycles for the 4 banks to reply
    - \* bandwidth = 4/11 words per cycle

## Virtual Memory: Motivation

- We have a machine with 16 Mbytes of main memory
- We have a problem requiring 20 Mbytes of memory
  - **Solution:** use the disk for back up and juggle pieces in and out of main memory under user control
- This technique is called memory overlay (same memory chunk is “overlayed” with different chunks that are kept on the disk (or tape)). This technique is:
  - difficult to program
  - error prone
  - non-portable (if we buy more memory, we must recode the program!)

## Virtual Memory - Definition

- Virtual memory (VM) is a hardware/software mechanism (under OS control) that gives the user the illusion of a memory system that is much larger than the physical (primary) memory of the machine.
  - The illusion of a large memory is accomplished by making use of a large secondary memory (disk/paging device/swap space) as backup for the primary memory
- VM give the appearance of a memory that is as large as the swapping space (very large) and as fast as the physical memory (much faster than the swapping device). It allows:
  - Dynamic size adjustment of process space
  - Better utilization of primary memory
  - Convenient sharing of process space
  - A large memory is essential for multiprogramming

## Swap Unit

- What is the “swap” unit in VM?
  - segments (variable size)
  - pages
  - paged segmented virtual memory

	Page	Segment
Words per address	One	Two (segment and offset)
Programmer visible?	Invisible to application programmer	May be visible to application programmer
Replacing a block	Trivial (all blocks are the same size)	Hard (must find contiguous, variable-size, unused portion of main memory)
Memory use inefficiency	Internal fragmentation (unused portion of page)	<i>External fragmentation</i> (unused pieces of main memory)
Efficient disk traffic	Yes (adjust page size to balance access time and transfer time)	Not always (small segments may transfer just a few bytes )



## Paging Versus Segmentation

- Both can waste memory, depending on the block size and how well the segments fit together in main memory.
- Programming languages with unrestricted pointers require both the segment and the address to be passed.
- A hybrid approach, called *paged segments*, shoots for the best of both worlds:
  - segments are composed of pages, so replacing a block is easy
  - a segment may be treated as a logical unit

## Paged Virtual Memories

- The unit of transfer between primary and secondary memory is called the page (similar to the block in the case of caches)
- The primary memory is logically divided into page frames
- The secondary memory is logically divided into pages (page frame size = page size)
- A main memory page frame is either
  - vacant
  - occupied by a copy of some secondary memory page

## Paged Virtual Memories – Typical Parameters

- Page size = 0.5 to 8 Kbytes
- Main memory hit time: 1 to 10 cycles
- Main memory miss penalty: 100,000 to 600,000 cycles (if it is not overlapped with the execution of another job)
- Main memory miss rate: 0.0001% to 0.001%
- Main memory size: 4 Mbytes to 2 Gbytes
- Secondary memory size: 16 Mbytes to 2 Gbytes (per process)

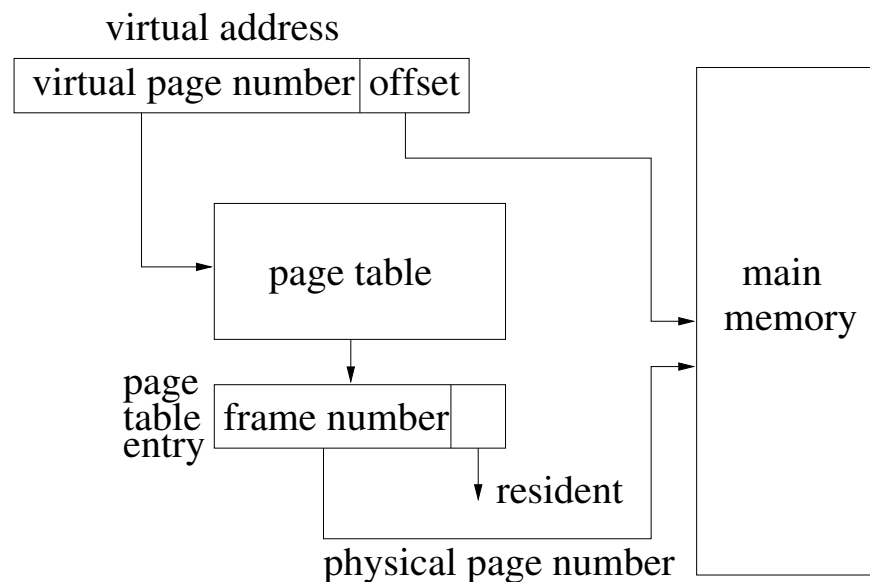
## Address Translation

- CPU generates virtual addresses
- A virtual address is divided into
  - page number
  - offset within page
- Translation of virtual page number to physical page done using:
  - page tables, or
  - page registers (inverted page tables)
- In physical page is not resident in main memory, a page fault occurs
  - the page number is set to the secondary memory
  - a frame is identified to hold the missing page
  - the missing page is transferred into the frame

## Address Translation using Page Tables

- A table with one entry per VM page is maintained by OS
- The virtual page number is used to index the page table, and retrieve the corresponding page table entry
  - Each entry contains
    - \* residence bit: whether or not a copy of the page is currently in main memory
    - \* frame number: the frame containing the copy of the page if it is resident
    - \* other information (protection bits)
  - Entry will either indicate a fault or provide physical page address be concatenated with offset and used to access main memory

## Page Tables – Address Translation



## Page Tables – Replacement Policy

- A complex replacement policy is used for pages in main memory because of more expensive miss penalty associated with secondary memories.
  - To lower miss ratio, LRU (or a simpler variant) is used. Trace driven simulations show that LRU is quite close to optimum. “Oracle” solution evicts the frame which won't be used for the longest period in the future.
- Write policy: always write back
- If corresponding dirty bit is set, a page is written back before it is evicted.

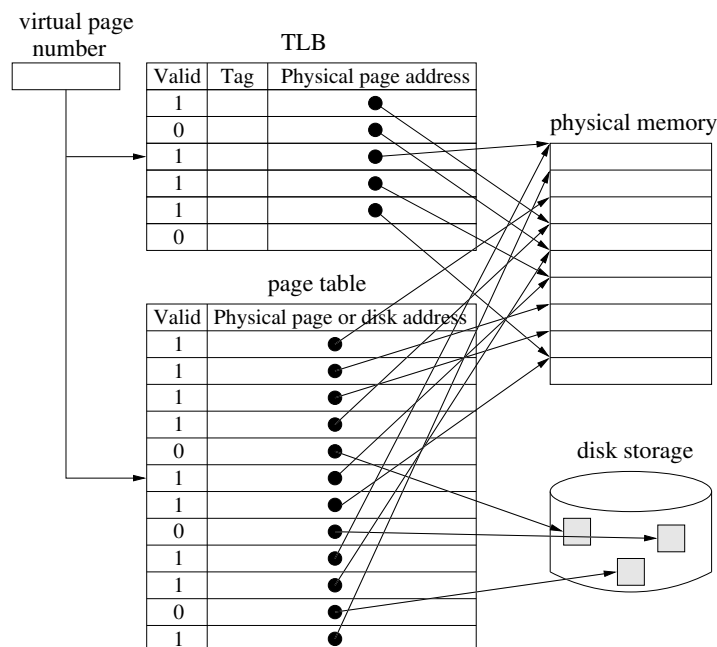
## Speeding up Page Table Lookup

- Each memory reference takes two memory transactions
  - one access to get the page table entry
  - one access to get the data (provided there was no page fault)
- Page table is large; a part of it may be in disk!
- **Solution:** Cache most recently used page table entries
- Such a translation cache is called the **translation look-aside buffer (TLB)**
  - 8 to 64 entries
  - full or set associative
  - has better than 99 percent hit ratio because of locality

## Translation with TLB

- For a TLB hit, the physical page number is obtained in 1 cycle
  - Note: Tag may be extended to include a process identifier (so TLB need not be always flushed on context switch)
- If we miss, then the regular translation mechanism is used (many cycles), and TLB is updated with new page-number/page-table-entry pair

## Translation with TLB (Cont'd.)



## Page Registers (Inverted Page Tables)

- Each frame has associated with it a register which contains
  - residence bit: whether or not the frame is occupied
  - occupier: page number of page occupying the frame
  - protection bits
- Page Register – Example
  - Main memory size: 16 Mbytes
  - Page size: 4096 bytes
  - Number of frames: 4096
  - Space used for page registers (assuming 8 bytes/register): 32 Kbytes
  - Percentage of main memory used for page registers: 0.2%
  - Size of virtual memory: irrelevant

## Page Registers

- Advantages:
  - Size of translation table is a very small fraction (less than 1%) of primary memory
  - Size of translation table is independent of size of virtual memory
- Disadvantage: We have the reverse of the information we need
  - How to perform translation?
  - Search the translation table (the set of page registers) for the desired page number

## Searching the Inverted Page Table (IPT)

- If the number of frames is small, then the page registers can be placed in an associative memory
- The page number is looked up in the associative memory:
  - hit: frame number is extracted
  - miss: page fault
- Limitations:
  - large associative memory is expensive
  - memory expansion is non-trivial

## Searching the Inverted Page Table

Use a proven fast search technique: hash tables

- Page registers are placed in an array (at a reserved area of primary memory)
- Page  $i$  is placed in frame number  $f(i)$  where  $f$  is an agreed upon “hashing function”
- To lookup page  $i$ , we perform the following:
  - compute  $f(i)$  and use it as an index into table of page registers
  - extract the corresponding page register
  - check if register is currently holding  $i$ ; if so, we have a hit
  - otherwise, we have a miss

## Searching the Inverted Page Table (Cont.)

### Minor complication

- Since the number of items (page numbers) is usually much larger than the number of slots in the hash table, two or more items will “hash” to the same table entry
- In other words, it is possible to have  $i \neq j$  and  $f(i) = f(j)$ 
  - if pages  $i$  and  $j$  are resident at the same time and  $f(i) = f(j)$ , then we have a problem
- Two different “keys” that hash to the same hash table entries are said to “collide”
- There are many standard techniques for dealing with collisions
  - use a linked list of items that hash to a particular table entry
  - rehash index until the key is found or an empty table entry is reached
  - ...

## Protection

- Each process has its own virtual address space, but physical memory is shared
  - a multi-programmed machine must provide protection to the users
- Solution: OS manages page tables to ensure that physical memory maps are disjoint if so desired
  - requires at least two modes of execution: OS in executive/kernel/supervision mode, other in user mode
    - \* supervisor mode can access “privileged” instructions, used to implement a protection policy
    - \* need instructions to switch modes (e.g., system call exception)



## Protection: A Simple Scheme

- The processor includes two address bounding registers
  - a base register
  - a bound register
- If address produced by the CPU is out of bounds (compared by hardware), a protection interrupt is generated
- The bounding registers can be changed only in system mode:
  - when the operating system launches a process, it sets its bounding registers
  - the OS saves/restores the bounding registers between context switches
  - bound checking is not performed in system mode
- Shortcoming: a virtual space of a process must be contiguous

## Flexible Page Level Protection

With paged virtual memories, protection is typically enforced by the translation mechanism, under OS control

- Each process gets the full address space (two processes can use the same virtual address)
- Translation scheme ensures that physical addresses are process specific
  - Each page table entry also contains protection bits (read; write; execute)
  - Each memory access is checked against the protection bits
  - An access (protection) violation interrupt is generated if the access does not match the page protection
- A similar technique is used for paged registers