ECE 341

Lecture # 18

Instructor: Zeshan Chishti
zeshan@ece.pdx.edu

December 1, 2014

Portland State University

Lecture Topics

- The Memory System
  - Cache Memories
    - Performance Considerations
      - Hit Ratios and Miss Penalty
      - How to Increase Cache Hit Ratio?
  - Virtual Memory
    - Memory and Storage
    - Virtual Memory Organization
    - Address Translation
    - Translation Lookaside Buffer (TLB)
    - Page Faults and Replacements
- References:
  - Chapter 8: Sections 8.7, 8.8

How to Handle Stale Data?

- A "valid bit" is associated with each cache block
- If the block contains valid data, this bit is set to 1, otherwise it is set to 0

- Valid bits are set to 0, when the power is just turned on
- When a block is loaded into the cache the first time, the valid bit is set to 1
- The processor fetches data from a cache block only if its valid bit is 1

- Data transfers between main memory & disk can occur directly through direct memory access (DMA), bypassing cache
- What happens if the memory contents change and a copy of the data is also resident in the cache?
  - The valid bit of cache block is set to "0" to indicate that this is a stale copy

Hit Ratio and Miss Penalty

- Cache hit ratio = No. of cache hits / No. of cache accesses
- Cache miss ratio = No. of cache misses / No. of cache accesses = 1- hit ratio
- Cache hit latency = No. of cycles it takes to read data from the cache
- Cache miss penalty = No. of cycles the processor has to wait when required data is not found in the cache

- For a system with only one level of cache, average memory access time is:

$$t_{avg} = hC + (1 - h)M$$

where:
h = Cache hit ratio
C = cache hit latency
M = Cache miss penalty

To reduce memory access time, it is important to have both high cache hit ratios and low cache miss penalties

Example

- Consider a computer with one level of cache, that has the following parameters: Cache access time is 1 cycle. When a cache miss occurs, a block of 8 words is transferred from memory to cache. It takes 10 cycles to transfer the first word of block, and the remaining words are transferred at the rate of 1 word per cycle. The miss penalty also includes the initial access to the cache and another delay of 1 cycle to transfer the word to the processor after the block has been loaded into the cache. Assume that the cache hit rate is 95%. Calculate the average memory access time.

- Solution:
  Cache hit rate h = 95%
  Cache hit latency C = 1
  Cache miss latency M = 1 + 10 + (8 – 1)(1) + 1 = 19
  Average memory access time t_{avg} = hC + (1-h)M = (0.95)(1) + (1-0.95)(19)
  = 1.9 cycles

Refer to Example 8.1 (Page 301) for a more detailed example

How to Increase Cache Hit Ratio?

- Make the cache larger
  - Allows more data to be kept in the cache
  - But increases both cost and cache access time

- Increase the block size, while keeping the cache size constant
  - Helps if there is high spatial locality in the program
  - But larger blocks are effective only up to a certain size
    - Very large blocks need longer to transfer => higher miss penalty
  - Block size should be neither too large nor too small
    - Typical choice for block sizes is in the 64 to 128 byte range

- Add a second level cache
  - First level (L1) cache provides fast access but small capacity
  - Second level (L2) cache can be slower but much larger than L1 cache to ensure a high hit rate, so that fewer accesses go to main memory

Two-level Cache Hierarchy

- In a processor with two levels of caches, the average memory access time is given by:

$$t_{avg} = h_1C_1 + (1 - h_1)(h_2C_2 + (1-h_2)M_2)$$

where:
h_1 = Hit ratio of L1 cache
h_2 = Hit ratio of L2 cache
C_1 = Access time of L1 cache
C_2 = Miss penalty to transfer data from L2 cache to L1 cache
M = Miss penalty to transfer data from memory to L2 cache

- Assume that $h_1 = h_2 = 0.9$, then only 0.1*0.1 = 1% of memory accesses need to go to main memory
- With a single level cache, 10% accesses would have gone to main memory

Prefetching

- New data is usually brought into the cache when it is needed
- Processor has to pay miss penalty if required data is not in cache
- To avoid this penalty, processor can *prefetch* the data into the cache *before* it is actually needed.
- Prefetching can be accomplished through software by including a special “prefetch” instruction in the processor instruction set
- Prefetching can also be accomplished using hardware:
  - Circuitry that attempts to discover patterns in memory references and then prefetches addresses according to these patterns

Virtual Memory

Introduction

- Recall that the most important challenge in memory system design is to provide a computer with as large and fast a memory as possible, within a given cost target
- Two key architectural solutions are used to increase the effective speed and size of the memory system
  - Cache memories are used to increase the effective speed of the memory system
  - **Virtual memory** is used to increase the effective size of the memory system

Memory and Storage

- Recall that the addressable memory space depends on the number of address bits in a computer
  - E.g., if a computer issues 40-bit addresses, addressable memory space is 1 TB
- But due to cost constraints, main memory in a computer is generally not as large as the entire possible addressable space
- Application programmers do not want to be constrained by the limited size of main memory
- To mitigate the limited main memory, large programs that cannot fit in the main memory have their parts stored on secondary storage devices, such as magnetic disks

Memory and Storage (cont.)

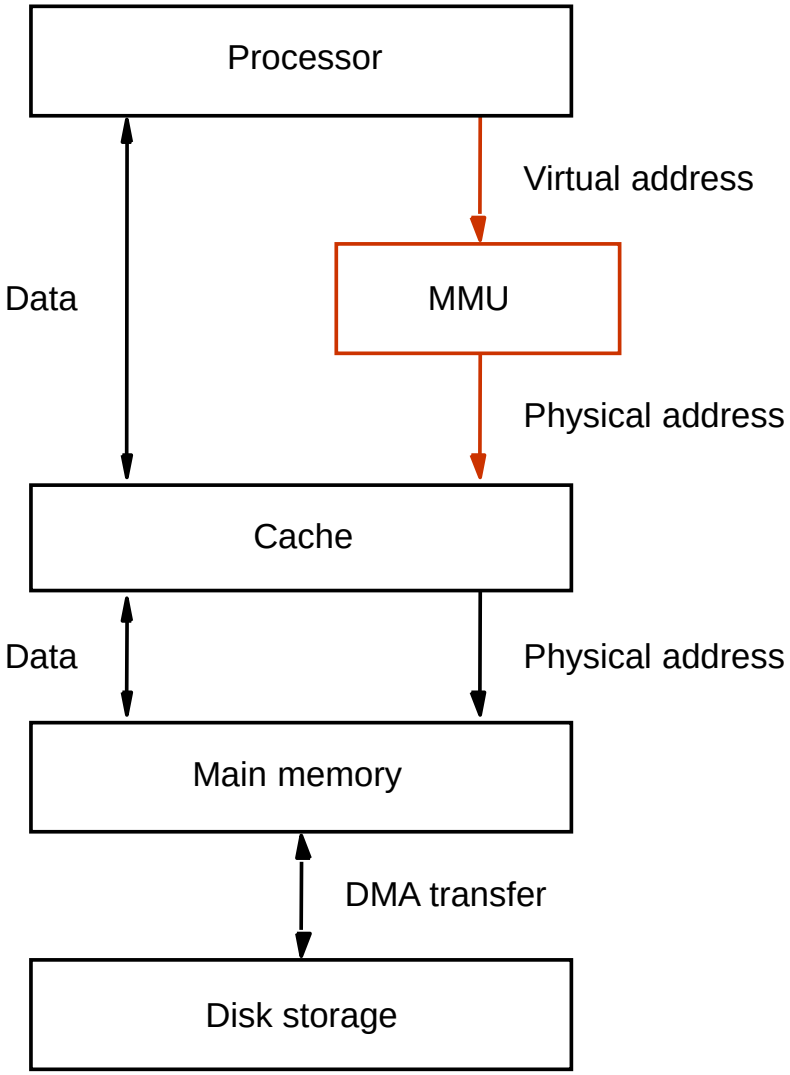
- Pieces of programs must be transferred from secondary storage to the main memory before they can be executed
- When a new piece of data is to be transferred to the main memory, and the main memory is full, then some other data in the main memory must be replaced
  - This is similar to the replacement issue 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

Virtual Memory

- Techniques used to manage data transfer between main memory and secondary storage are called **virtual-memory** techniques

- Instruction and data references in a program do not depend on the physical size of the main memory. They depend only on the number of address bits used by the processor
- When executing a program, addresses issued by the processor are called **logical** or **virtual** addresses
- Virtual addresses are translated into **physical** addresses by a combination of hardware and software subsystems
- A physical address refers to the mapping of a virtual address in main memory
  - If the virtual address refers to a part of the program that is currently in the main memory, the translation provides the physical address, which is accessed immediately
  - If the virtual address refers to a part of the program that is not 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

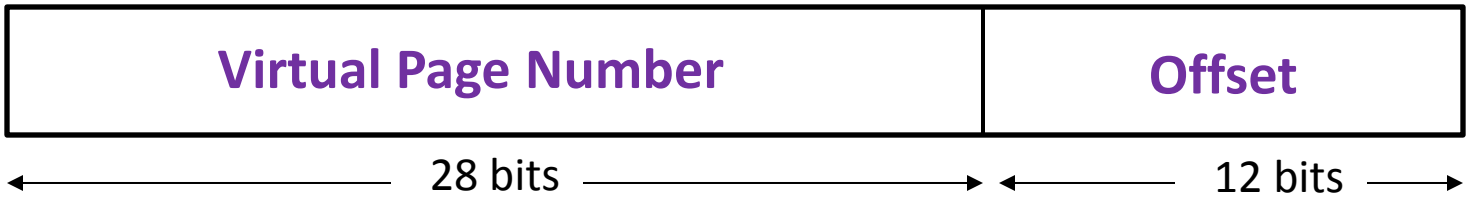
Memory Pages

- Assume that physical memory space is divided into fixed-length units called pages
- A page consists of a block of words that occupy contiguous locations in the main memory
- Page is a basic unit of information that is transferred between secondary storage and main memory.
- Size of a page commonly ranges from 2K to 16K bytes
  - Pages should not be too small, because the access time of a secondary storage device is much larger than the main memory
  - Pages should not be too large, else a large portion of the page may not be used, and it will occupy valuable space in the main memory

Address Translation

- A virtual address needs to be translated to a physical address, before the memory is accessed

- Each virtual address generated by the processor is interpreted as a virtual page number (high-order bits) plus a offset (low-order bits) that specifies the location of a particular byte within that page
- For example, if the processor uses 40 address bits and the page size is 4 kBytes, then the address bits are interpreted as follows:



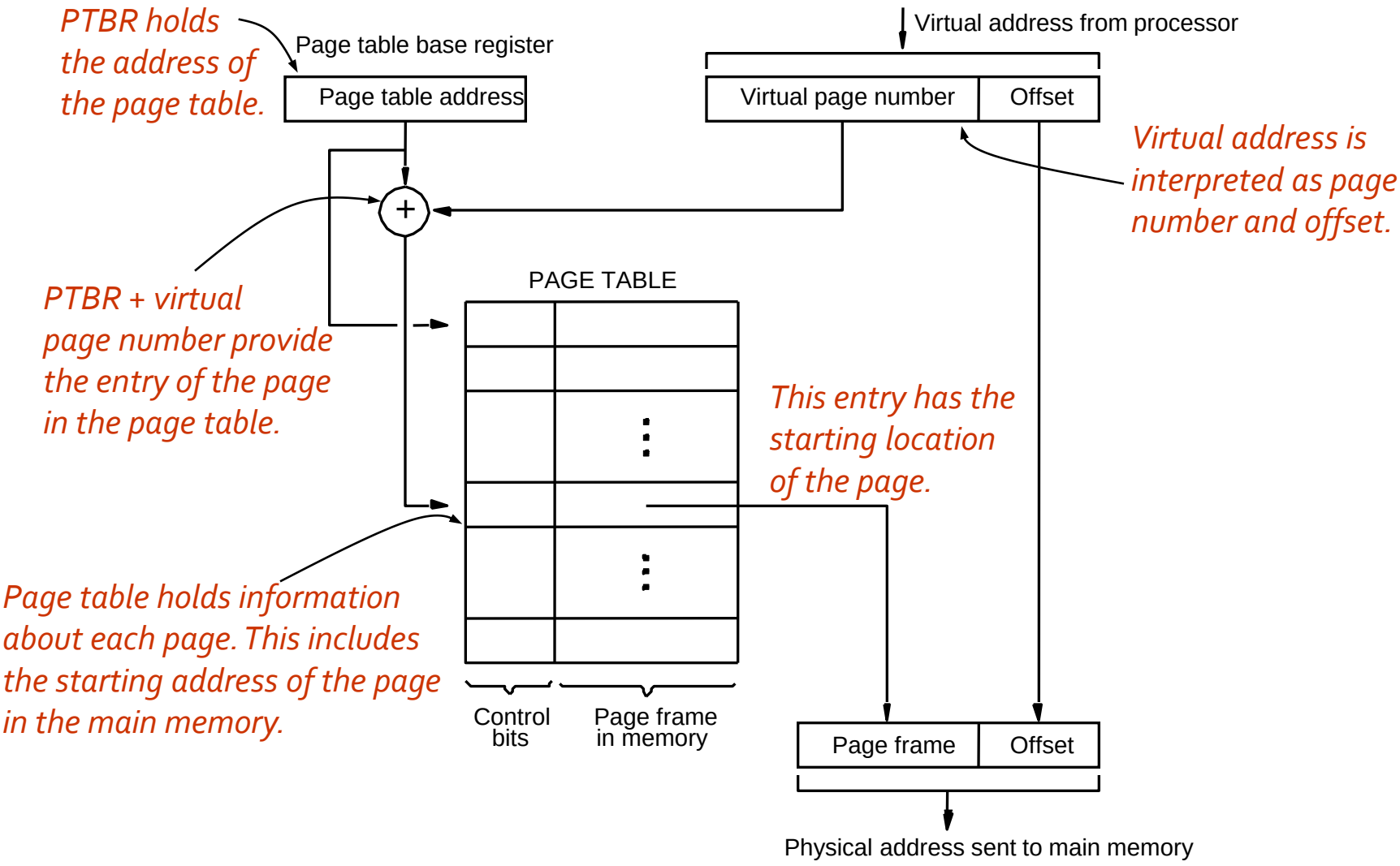
- The first step in address translation is to break down a virtual address into the above two fields

Address Translation (cont.)

- Information about each virtual page is kept in the **page table**
- Conceptually the page table has one entry for each virtual page
  - Is the virtual page currently mapped into the main memory?
  - If yes, what is the main memory address where the page is mapped to?
  - Current status of the page (dirty, LRU etc.)

- Starting address of the page table is kept in page table base register
- Virtual page number generated by the processor is added to the contents of the page table base register
  - This provides the address of the corresponding entry in the page table
- The contents of this page table entry provide the starting address of the page if the page is currently in the main memory

Address Translation (cont.)



Page Table Entry

- Page table entry for a page also includes some control bits which describe the status of the page while it is in the main memory
- "Valid" bit indicates the validity of the page
  - Indicates whether the page is actually loaded into the main memory
- "Dirty" bit indicates whether the page has been modified during its residency in the main memory
  - This bit determines whether the page should be written back to the disk when it is removed from the main memory
  - Similar to the "dirty" bit in case of cache memory

Page Table Location

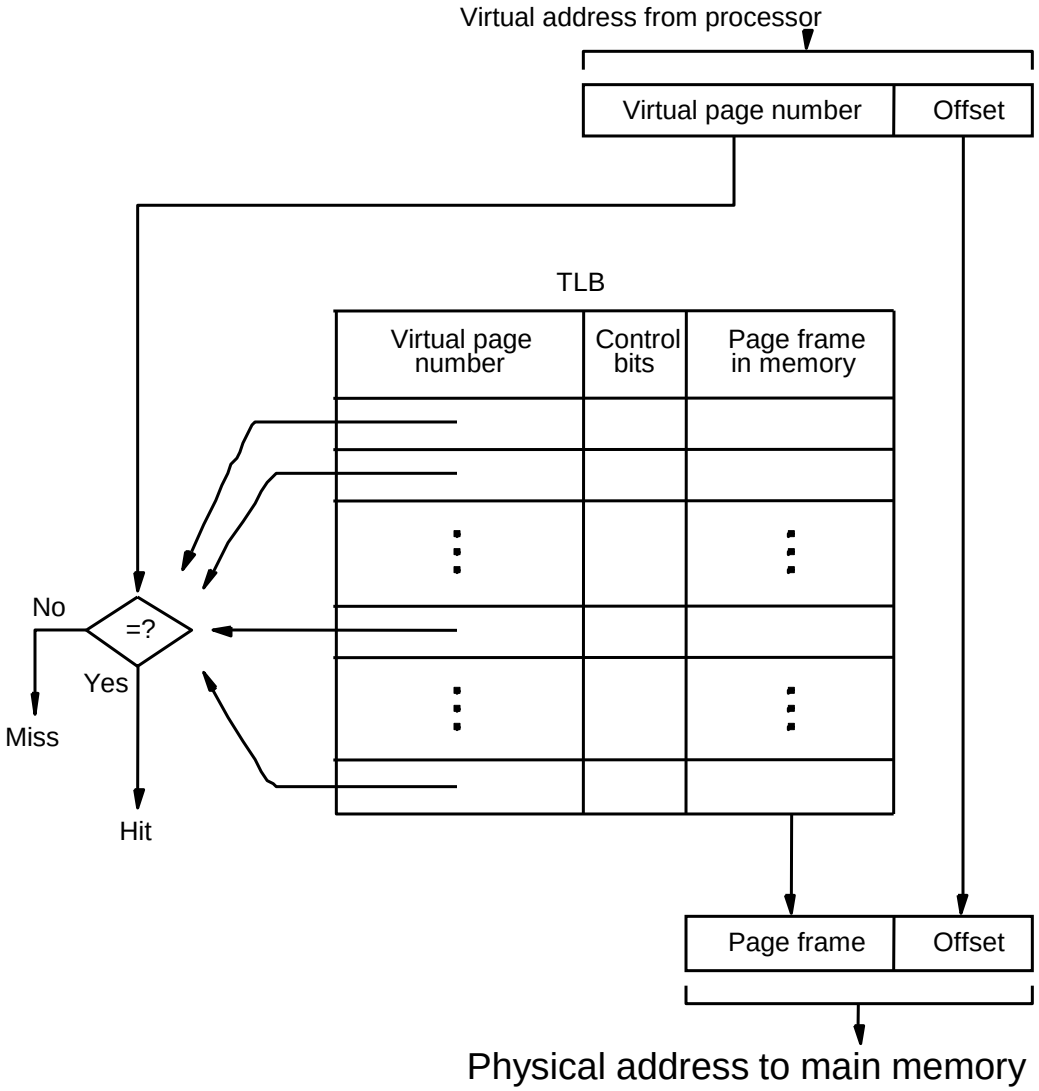
- Where should the page table be located?

- Recall that the page table is used by the MMU for every read and write access to the memory
  - Ideal location for the page table is within the MMU
- Page table is quite large
- MMU is implemented as part of the processor chip
- Impossible to include a complete page table on the chip
- Page table is kept in the main memory
- A copy of a small portion of the page table can be cached within the MMU
  - Cached portion consists of page table entries that correspond to the most recently accessed pages

Translation Lookaside Buffer (TLB)

- A small cache called as Translation Lookaside Buffer (TLB) is included in the MMU
  - TLB holds page table entries of the most recently accessed pages
- Recall that cache memory holds most recently accessed blocks from the main memory
  - The relationship between the TLB and page table is similar to the relationship between the cache and main memory
- Page table entry for a page includes:
  - Address of the page frame where the page resides in the main memory
  - Some control bits
- In addition to the above for each page, TLB must hold the virtual page number for each page

TLB (cont.)



Associative-mapped TLB

- High-order bits of the virtual address generated by the processor select the virtual page
- These bits are compared to the virtual page numbers in the TLB.
- If there is a match, a hit occurs and the corresponding address of the page frame is read.
- If there is no match, a miss occurs and the page table within the main memory must be consulted.
- Set-associative mapped TLBs are used in commercial processors.

TLB (cont.)

- How to keep the entries of the TLB coherent with the contents of the page table in the main memory?

- OS may change the contents of the page table in the main memory
  - Simultaneously it must also invalidate the corresponding entries in the TLB
- A control bit is provided in the TLB to invalidate an entry
- If an entry is invalidated, then the TLB gets the information for that entry from the page table

Page Fault

- What happens if a program generates an access to a page that is not in the main memory?

- In this case, a page fault is said to occur
  - Whole page must be brought into the main memory from the disk, before the execution can proceed

- Upon detecting a page fault by the MMU, following actions occur:
  - MMU asks the operating system to intervene by raising an exception
  - Processing of the active task which caused the page fault is interrupted
  - Control is transferred to the operating system
  - Operating system copies the requested page from secondary storage to the main memory
  - Once the page is copied, control is returned to the task which was interrupted

Page Replacement

- When a new page is to be brought into the main memory from secondary storage, the main memory may be full
  - Some page from the main memory must be replaced with this new page
- How to choose which page to replace?
  - This is similar to the replacement that occurs when the cache is full
  - The principle of locality of reference can also be applied here
  - A replacement strategy similar to LRU can be applied
- Since the size of the main memory is relatively larger compared to cache, a relatively large amount of programs and data can be held in the main memory
  - Minimizes the frequency of transfers between storage and main memory