CMSC 411
Computer Systems Architecture
Lecture 16
Memory Hierarchy 3
(Main Memory & Virtual Memory)

Main memory management

- Questions:
  - How big should main memory be?
  - How to handle reads and writes?
  - How to find something in main memory?
  - How to decide what to put in main memory?
  - If main memory is full, how to decide what to replace?

The scale of things

- Typically (as of 2000):
  - Registers: < 1 KB, access time .25 - .5 ns
  - Cache: < 8 MB, access time .5 - 25 ns
  - Main Memory: < 4 GB, access time 150 - 250 ns
  - Disk Storage: > 30 GB, access time 5,000,000 ns (5ms)
- Memory Technology: CMOS (Complementary Metal Oxide Semiconductor)
  - uses a combination of n- and p-doped semiconductor material to achieve low power dissipation.

Memory hardware

- DRAM: dynamic random access memory, typically used for main memory
  - One transistor per data bit
  - Each bit must be refreshed periodically (e.g., every 8 milliseconds), so maybe 5% of time is spent in refresh
  - Access time < cycle time
  - Address sent in two halves so that fewer pins are needed on chip (row and column access)

Memory hardware (cont.)

- SRAM: static random access, typically used for cache memory
  - 4-6 transistors per data bit
  - No need for refresh
  - Access time = cycle time
  - Address sent all at once, for speed

Bottleneck

- Main memory access will slow down the CPU unless the hardware designer is careful
- Some techniques can improve memory bandwidth, the amount of data that can be delivered from memory in a given amount of time:
  - wider main memory
  - interleaved memory
  - independent memory banks
  - avoiding memory bank conflicts

Wider main memory

- Wider cache lines
  - Cache miss: If a cache block contains k words, then each cache miss involves these steps repeated k times:
    - » Send the address to main memory
    - » Access the word (i.e., locate it)
    - » Send the word to cache, with the bits transmitted in parallel
  - Idea behind wider memory: the user thinks about 32 bit words, but physical memory can have longer words
  - Then the operations above are done only k/n times, where n is the number of 32 bit words in a physical word

Wider main memory (cont.)

- Extra costs:
  - a wider memory bus: hardware to deliver 32n bits in parallel, instead of 32 bits
  - a multiplexer to choose the correct 32 bits to transmit from the cache to the CPU

Interleaved memory

- Partition memory into banks, with each bank able to access a word and send it to cache in parallel
- Organize address space so that adjacent words live in different banks - called interleaving
- For example, 4 banks might have words with the following octal addresses:

Bank 0	Bank 1	Bank 2	Bank3
00	01	02	03
04	05	06	07
10	11	12	13
…	…	…	…

Interleaved memory (cont.)

- Note how nice interleaving is for write-through
- Also helps speed read and write-back
- *Note*: Interleaved memory acts like wide memory, except that words are transmitted through the bus sequentially, not in parallel

Independent memory banks

- Each bank of memory has its own address lines and (usually) a bus
- Can have several independent banks: perhaps
  - one for instructions
  - one for data
  - this is called a Harvard architecture
- Banks can operate independently without slowing others

Avoid memory bank conflicts

- By having a prime number of memory banks
- Since arrays frequently have even dimension sizes - and often dimension sizes that are a power of 2 - strides that match the number of banks (or a multiple) give very slow access

Example: Interleaving

```
int x[256][512];

for (j=0; j<512; j=j+1)
for (i=0; i<256; i=i+1)
   x[i][j] = 2 * x[i][j];
```

- First access the first column of **x**:
  - x[0][0], x[1][0], x[2][0], ... x[255][0]
- with addresses
  - K, K+512*4, K+512*8, ... K+512*4*255
- With 4 memory banks, all of the elements live in the same memory bank, so the CPU will stall in the worst possible way

How much good do these techniques do?

- Example: Assume a cache block of 4 words, and
  - 4 cycles to send address to main memory
  - 24 cycles to access a word, once the address arrives
  - 4 cycles to send a word back to cache
- Basic miss penalty: 4*32 = 128 cycles, since each of 4 words has the full 32 cycle penalty
- Memory with a 2-word width: 2*(32+4) = 72 cycle miss penalty
- Simple interleaved memory: address can be sent to each bank simultaneously, so miss penalty is 4 + 24 + 4*4 (for sending words) = 44 cycles
- Independent memory banks: 32 cycle miss penalty, as long as the words are in different banks, since each has its own address lines and bus

Virtual addressing

- Computers are designed so that multiple programs can be active at the same time
- At the time a program is compiled, the compiler has to assign addresses to each data item. But how can it know what memory addresses are being used by other programs?
- Instead, the compiler assigns virtual addresses, and expects the loader/OS to provide the means to map these into physical addresses

Memory protection

- Each program "lives" in its own virtual space, called its process
- When the CPU is working on one process, others may be partially completed or waiting for attention
- The CPU is time shared among the processes, working on each in turn
- And main memory is also shared among processes

In the olden days …

- The loader would locate an unused set of main memory addresses and load the program and data there
- There would be a special register called the relocation register, and all addresses that the program used would be interpreted as addresses relative to the base address in that register
- So if the program jumped to location 54, the jump would really be to 54 + contents of relocation register. A similar thing, perhaps with a second register, would happen for data references

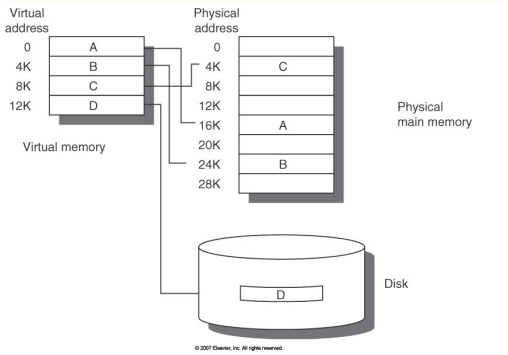
In the less-olden days ...

- It became difficult to find a contiguous segment of memory big enough to hold program and data, so the program was divided into pages, with each page stored contiguously, but different pages in any available spot, either in main memory or on disk
- This is the virtual addressing scheme
  - to the program, memory looks like a contiguous segment, but actually, data is scattered in main memory and perhaps on disk

But we know all about this!

- Already know that a program and data can be scattered between cache memory and main memory
- Now add the reality that its location in main memory is also determined in a scattered way, and some pages may also be located on disk
- So each page has its own relocation value

Virtual Memory – Fig. C.18



Parameters – Fig. C.19

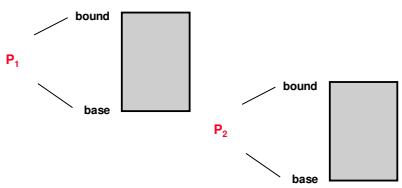
Parameter	First-level cache	Virtual memory
Block (page) size	16-128 bytes	4096-65,536 bytes
Hit time	1-3 clock cycles	50-150 cc
Miss penalty	8-200 cc	10^6-10^7 cc
(access time)	(6-160 cc)	(.8-8)*10^6 cc
(transfer time)	(2-40 cc)	(.2-2)*10^6 cc
Miss rate	0.1-10%	0.00001-0.001%
Address mapping	25-45 bit physical address to 14-20 bit cache address	32-64 bit virtual address to 25-45 bit physical address

Cache vs. virtual memory

Cache	Virtual memory
Cache miss handled by hardware	Page faults handled by operating system
Cache size fixed for a particular machine	Virtual memory size fixed for a particular program
Fundamental unit is a block	Fundamental unit is a fixed-length page or a variable-length segment
Cache fault	Page fault

Old Protection mode: Base & Bound

- User processes need to be protected from each other
- Two registers, base and bound test whether this virtual address belongs to this process
- If not, a memory protection violation exception is raised
- Users cannot change the base and bound registers

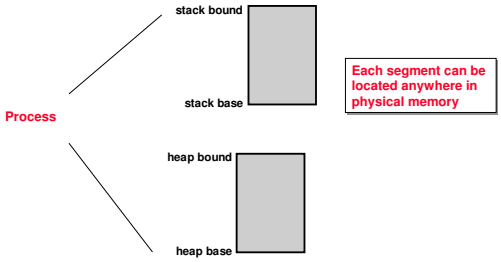


Who can change them?

- The operating system needs access to the base and bound registers
- So a process that is labeled kernel (also called supervisor or executive) can access any memory location and change the registers
- Kernel processes are accessed through system calls, and a return to user mode is like a subroutine return, restoring the state of the user process

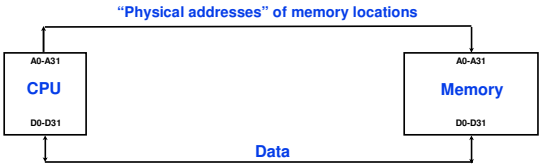
Segmentation

- Basically multiple base&bounds
  - w/ virtual memory



Paging

The Limits of Physical Addressing

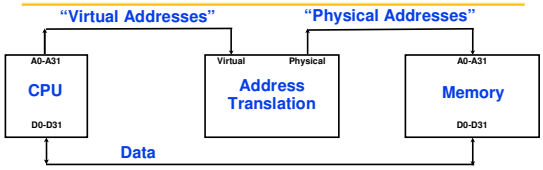


All programs share one address space:
The physical address space

Machine language programs must be aware of the machine organization

No way to prevent a program from accessing any machine resource

Solution: Add a Layer of Indirection



User programs run in an standardized virtual address space

Address Translation hardware managed by the operating system (OS) maps virtual address to physical memory

Hardware supports "modern" OS features: Protection, Translation, Sharing

Paging overview

- Fully-associative mapping, because page faults are really, really expensive
- Page is located using a page table, one entry per page in the virtual address space
  - Size is sometimes reduced by hashing, to make one entry per physical page in main memory – an inverted page table
- Since locality says that a page will be used multiple times, address translation usually tests the addresses of the recently referenced pages before looking in other places
- So address translation information is held in the translation look-aside buffer (TLB)

Paging overview (cont.)

- Most machines replace the LRU page
  - Moving pages between memory and disk is so slow that it's worth doing something close to real LRU
- Disks are *so slow* that machines use write-back, not write-through, and keep a *dirty bit* for each page

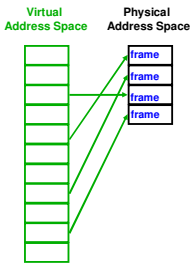
Advantages of Paged Virtual Memory

- Translation
  - Program can be given consistent view of memory, even though physical memory is scrambled
  - 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.

Advantages of Paged Virtual Memory

- 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")

Page tables encode virtual address spaces



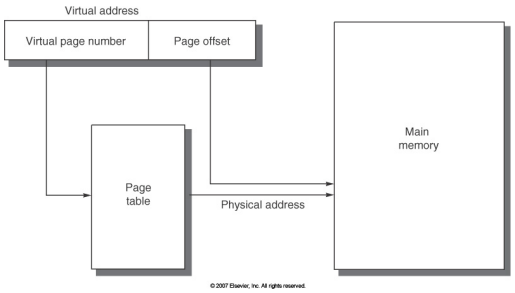
A virtual address space is divided into blocks of memory called pages

A machine usually supports pages of a few sizes (MIPS R4000):

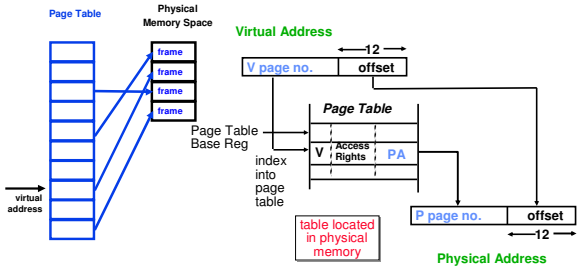
Page Size
4 Kbytes
16 Kbytes
64 Kbytes
256 Kbytes
1 Mbyte
4 Mbytes
16 Mbytes

A valid page table entry encodes physical memory "frame" address for the page

Page tables – Fig. C.22



Details of Page Table



- Page table maps virtual page numbers to physical frames
  - "PTE" = Page Table Entry
- Virtual memory => treat memory ≈ cache for disk

Two-Level Page Tables

Each process needs its own address space!

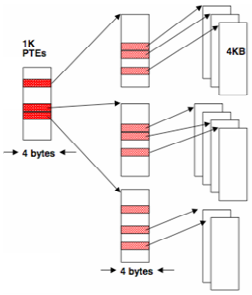
Two-level Page Tables

32 bit virtual address

31	22	21	12	11	0
P1 index		P2 index		Page Offset	

Top-level table *wired* in main memory

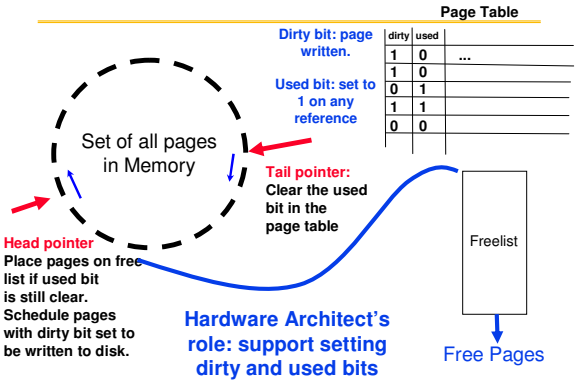
Subset of 1024 second-level tables in main memory; rest are on disk or unallocated



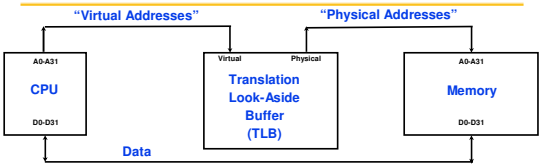
Choosing page size

- A large page size
  - keeps page table small.
  - reduces cache miss times, if accesses have locality
  - reduces start-up overhead in moving data from disk to memory
  - means fewer TLB misses
- but also
  - wastes memory (internal fragmentation)
  - increases the time to start up a program

VM and Disk: Page replacement policy



MIPS Address Translation: How does it work?

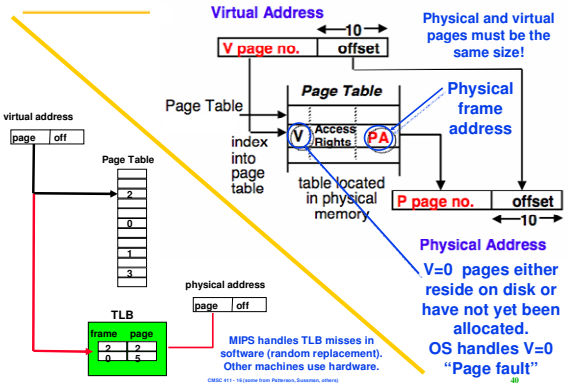


Translation Look-Aside Buffer (TLB)
A small fully-associative cache of mappings from virtual to physical addresses

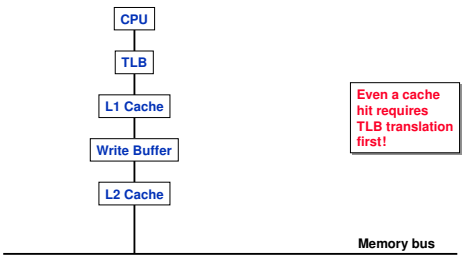
TLB also contains protection bits for virtual address

Fast common case: Virtual address is in TLB, process has permission to read/write it.

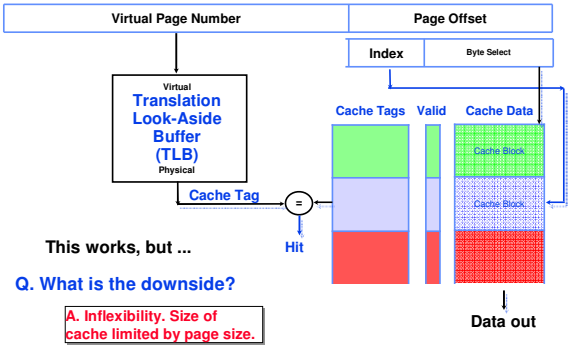
The TLB caches page table entries



Common Organization



Can TLB and caching be overlapped?



This works, but ...

Q. What is the downside?

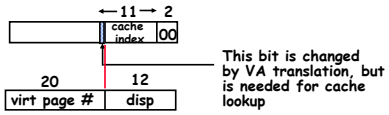
A. Inflexibility. Size of cache limited by page size.

Problems With Overlapped TLB Access

Overlapped access only works as long as the address bits used to index into the cache *do not change* as the result of VA translation

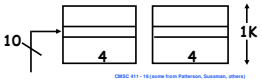
This usually limits things to small caches, large page sizes, or high n-way set associative caches if you want a large cache

Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:



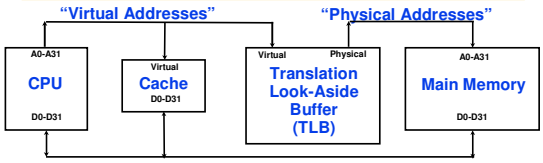
This bit is changed by VA translation, but is needed for cache lookup

Solutions:
go to 8K byte page sizes;
go to 2 way set associative cache; or
SW guarantee VA[13]=PA[13]



2 way set assoc cache

Use virtual addresses for cache?



Only use TLB on a cache miss !

Downside: a subtle, fatal problem. What is it?

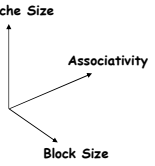
A. Synonym problem. If two address spaces share a physical frame, data may be in cache twice. Maintaining consistency is a nightmare.

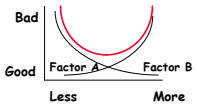
Paging vs. segmentation – Fig. C.21

	Page	Segment
Words per address	One	Two (segment/offset)
Programmer visible?	Invisible to app programmer	May be visible to app programmer
Replacing a block	Trivial (all blocks same size)	Hard (must find contiguous, variable-sized chunk)
Memory use inefficiency	Internal fragmentation (within page)	External fragmentation (in unused memory)
Efficient disk traffic	Yes (can adjust page size)	Not always (small segment problem)

Summary 1: The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation
- The optimal choice is a compromise
  - depends on access characteristics
    - » workload
    - » use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins





Summary 2: Caches

- The Principle of Locality:
  - Program accesses a relatively small portion of the address space in any short interval of time.
    - » Temporal Locality: Locality in Time
    - » Spatial Locality: Locality in Space
- Three Major Categories of Cache Misses:
  - Compulsory Misses: cold start misses
  - Capacity Misses: increase cache size
  - Conflict Misses: increase cache size and/or associativity. Nightmare Scenario: ping pong effect!
- Write Policy: Write Through vs. Write Back
- Today CPU time is a function of (ops, cache misses) vs. just f(ops): affects Compilers, Data structures, and Algorithms

Summary 3: TLB, Virtual Memory

- Page tables map virtual address to physical address
- TLBs are important for fast translation
- TLB misses are significant in processor performance
  - Funny times, as most systems can't access all of 2nd level cache without TLB misses!

Summary 3: TLB, Virtual Memory (cont.)

- Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions:
  - Where can block be placed?
  - How is block found?
  - What block is replaced on miss?
  - How are writes handled?
- Today VM allows many processes to share single memory without having to swap all processes to disk
  - Today VM protection is maybe even more important than memory hierarchy benefits, but computers still insecure