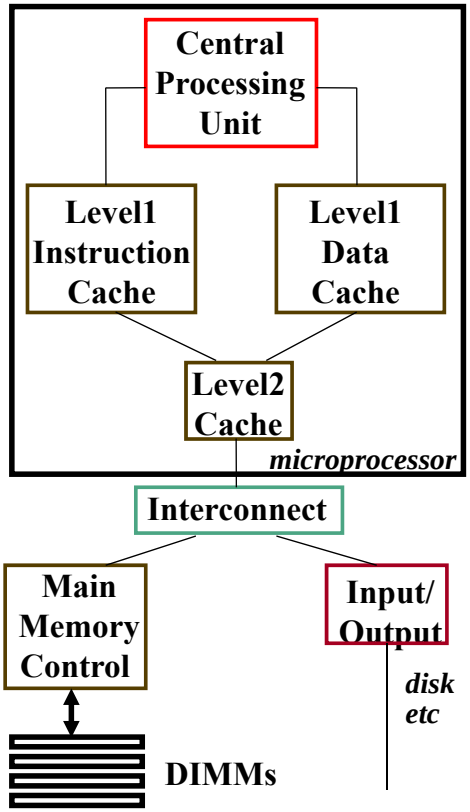
Chapter Five (3/3)

Main Memory Systems



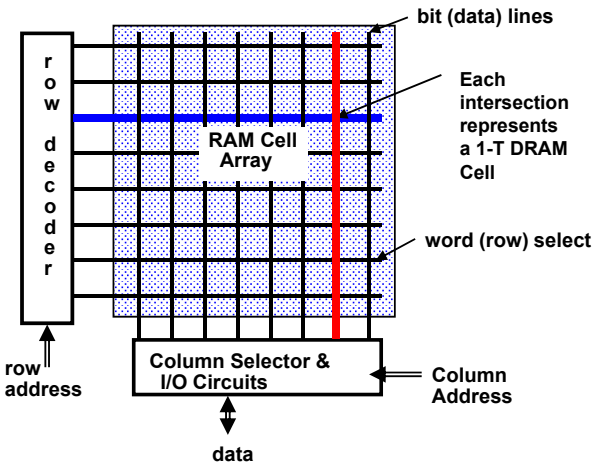
- The main memory (MM) lies in the memory hierarchy between the cache hierarchy and I/O (e.g., disk)
- In desktop systems, MM is too large to fit on the microprocessor die
  - MM controller may be on the microprocessor die or a separate chip
  - DRAMs lie on Dual In-line Memory Modules (DIMMs) on the printed circuit board
    - ~10-20 DRAMs per DIMM
- Interconnect width is typically ½ to ¼ of the cache block
  - Cache block is returned to L2 on multiple transfers

Main Memory Background

- **Performance of Main Memory:**
  - **Latency: Cache Miss Penalty**
    - Access Time: time between request and word arrives
    - Cycle Time: time between requests
  - **Bandwidth: I/O & Large Block Miss Penalty (L2)**
- **Main Memory is DRAM: Dynamic Random Access Memory**
  - **Dynamic since needs to be refreshed periodically (8 ms)**
  - **Addresses divided into 2 halves (Memory as a 2D matrix):**
    - RAS or Row Access Strobe
    - CAS or Column Access Strobe
- **Cache uses SRAM: Static Random Access Memory**
  - **No refresh (6 transistors/bit vs. 1 transistor /bit)**
  - **Address not divided**
- **Size: DRAM/SRAM 4-8, Cost/Cycle time: SRAM/DRAM 8-16**

Types of memories

- **RAM - Random Access Memory**
  - **DRAM - Dynamic RAM**
    - SIMM - 30 or 72 pin plug in memory module
    - DIMM - 168 pin memory module
  - **SRAM - Static Ram**
- **ROM - Read Only Memory (random access)**
  - **PROM - Programmable ROM**
  - **EPROM - Erasable PROM**
  - **Flash memory**

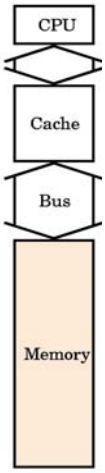


Types of DRAM memory

- **EDO Extended Data Output**
- **RAMBUS**
  - **Has dedicated 16-bit data bus. Can handle two operations per cycles. Dual channels.**
  - **Speeds – 300, 400, 800, 1066, 1200 MHz. Dual channel can handle 8 bytes per cycle.**
- **SDRAM**
  - **Synchronous DRAM Used with the front-side bus.**
  - **Speeds – 66, 100, 100, 133 MHz. On a 64 bit bus, it can handle 8 bytes per cycle.**
- **DDR2-SDRAM**
  - **Double Data Rate-Synchronous RAM version 2. SDRAM that sends data on both the rising and falling clock cycles**
  - **Speeds – 100, 133, 166 MHz, 16 bytes per cycle. May be named for the double speed or bits per second.**
- **ECC Error Correcting Code**
- **VRAM Video RAM**

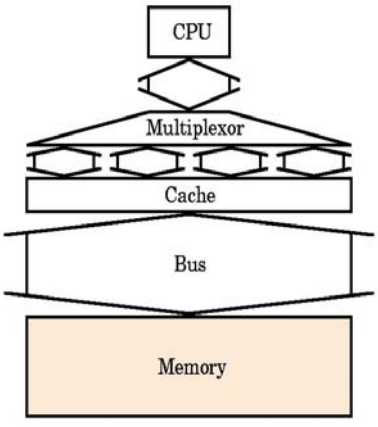
Improving MM performance

- **The latency to get the *first data* out of the MM is largely determined by**
  - **The speed of the DRAMs**
  - **The speed of the interconnect between the L2 and MM**
- **Focus is on reducing the time to return the *entire* cache block to L2**



1 cycle to transfer the address
15 cycles for each DRAM access
1 cycle to transfer the data
Memory and interconnect (bus) widths are ¼ that of the block
Cache miss penalty = 1 + 4 x (15 + 1) = 65 cycles

Baseline MM

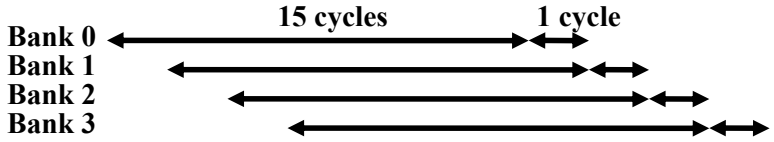


Wider MM

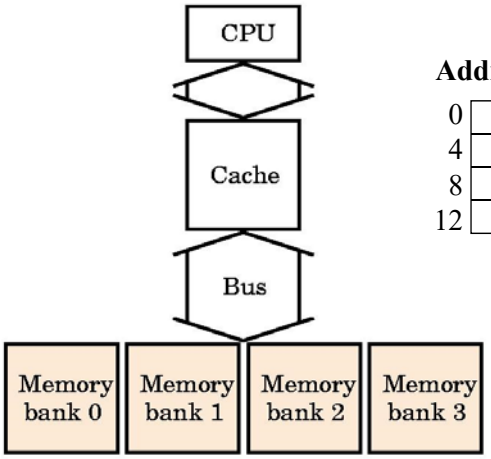
Increase MM and bus widths
Downsides
Hardware cost of wider bus
4X DRAMs or wider DRAMs
Cache miss penalty = 1 + (15 + 1) = 17 cycles for 4X width

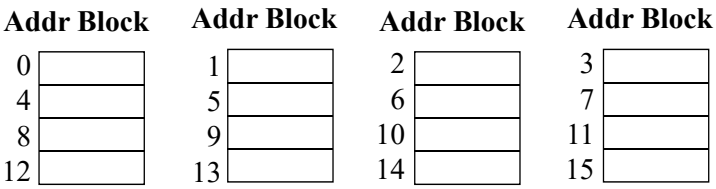
Interleaved memory

- **Divide the memory into banks (may be a DIMM)**
- **Each bank handles a fraction of the block on each access**
- **Bank accesses are started a cycle apart**



- **Cache miss penalty = 1 + 15 + 4 x 1 = 20 cycles**





Four-way interleaved memory address

Number of banks

- **How many banks?**
  number banks >= number clocks to access word in bank
  - **For sequential accesses, otherwise will return to original bank before it has next word ready**
  - **(like in vector case)**
- **Increasing DRAM => fewer chips => harder to have banks**
  - **64MB main memory**
    - 512 memory chips of 1-Mx1 (16 banks of 32 chips)
    - 8 64-Mx1-bit chips (maximum: one bank)
    - Wider paths (16 Mx4bits or 8Mx8bits)

Avoiding Bank Conflicts

- **Lots of banks**

```
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];
```

- **Even with 128 banks (512 mod 128=0), conflict on word accesses**
- **SW: loop interchange or array not power of 2 ("array padding")**
- **HW: Prime number of banks**
  - **bank number = address mod number of banks**
  - **address within bank = address / number of banks**
  - **modulo & divide per memory access with prime no. banks?**
  - **Let number of banks be = prime number of ($2^K \pm 1$)**
  - **address within bank = address mod number words in bank**
  - **easy if 2N words per bank → from chinese remainder theorem**

Virtual Memory

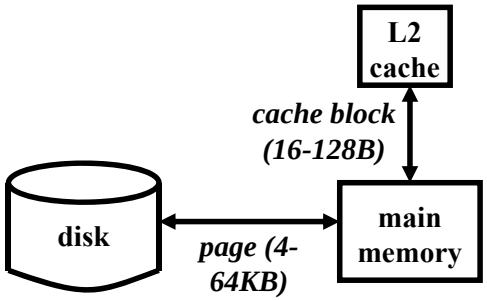
- **What is Virtual Memory?**
  - **Virtual memory => treat memory as a cache for the disk**
    - Terminology: blocks in this cache are called "Pages"
    - Typical size of a page: 1K ~ 8K
  - **Page table maps virtual page numbers to physical frames**
    - "PTE" = Page Table Entry

- **VM provides the following benefits**
  - **Allows multiple programs to share the same physical memory**
  - **Allows programmers to write code (or compilers to generate code) as though they have a very large amount of main memory**
  - **Automatically handles bringing in data from disk**

- **Cache terms vs. VM terms**
  - **Cache block => page**
  - **Cache Miss => page fault**

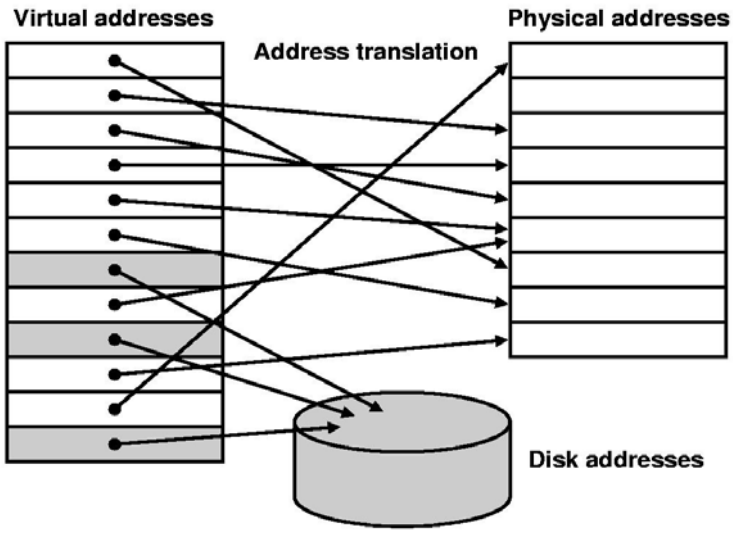
Memory Pages

- Transfers between disk and memory occur in *pages* whose size is defined in the ISA
- Page size is large to amortize the high cost of disk access (~5-10ms)
- Tradeoffs in increasing page size are similar as for cache block size



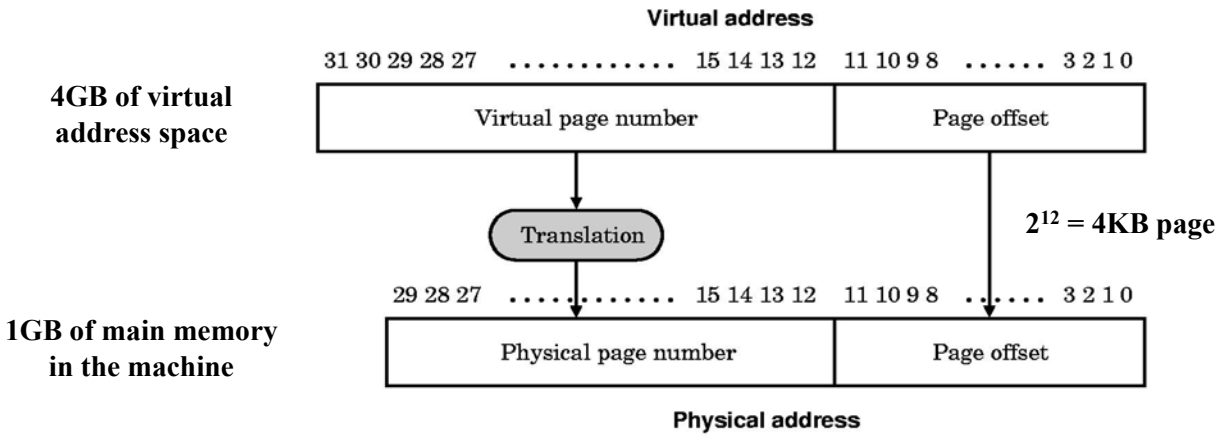
Virtual and physical addresses

- The *virtual addresses* in your program are *translated* during program execution into the *physical addresses* used to address memory
- Some virtual addresses may refer to data that is not in memory (not *memory resident*)



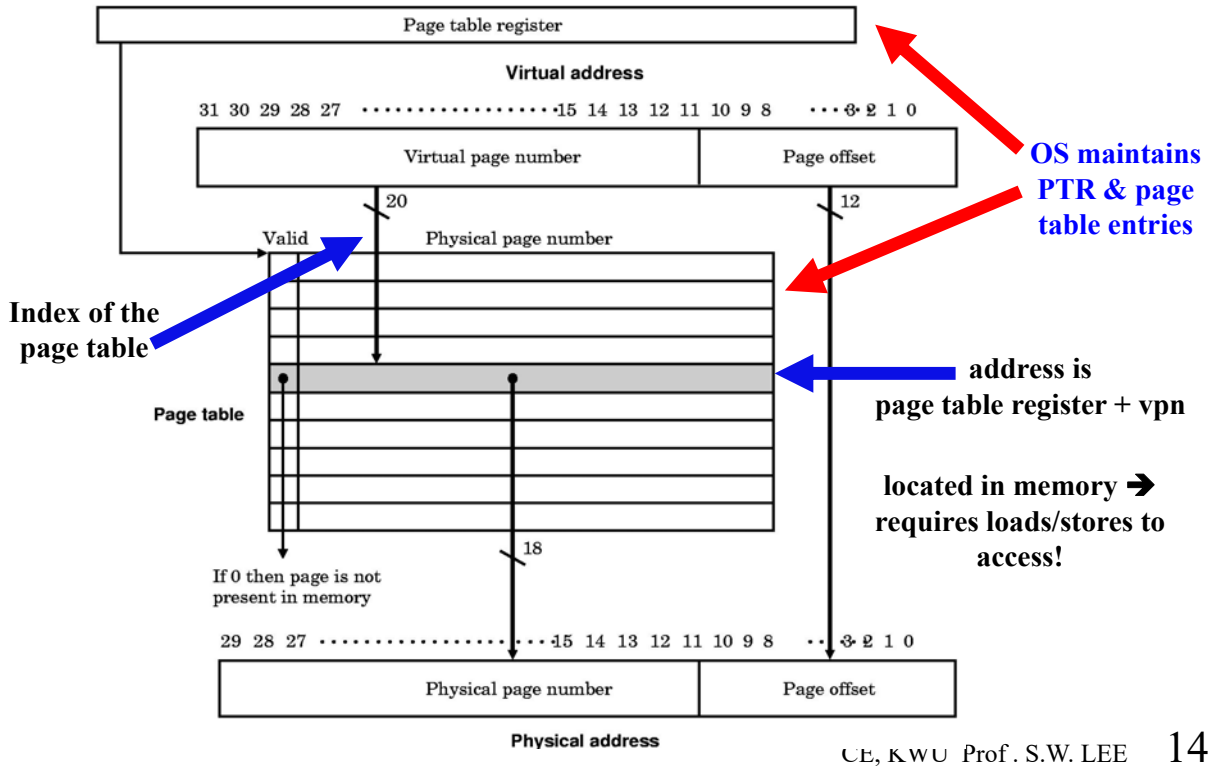
Address translation

- The *virtual page number (vpn)* part of the virtual address is translated into a *physical page number (ppn)* that points to the desired page
- The low-order *page offset* bits point to which byte is being accessed within a page
- The ppn + page offset form the physical address



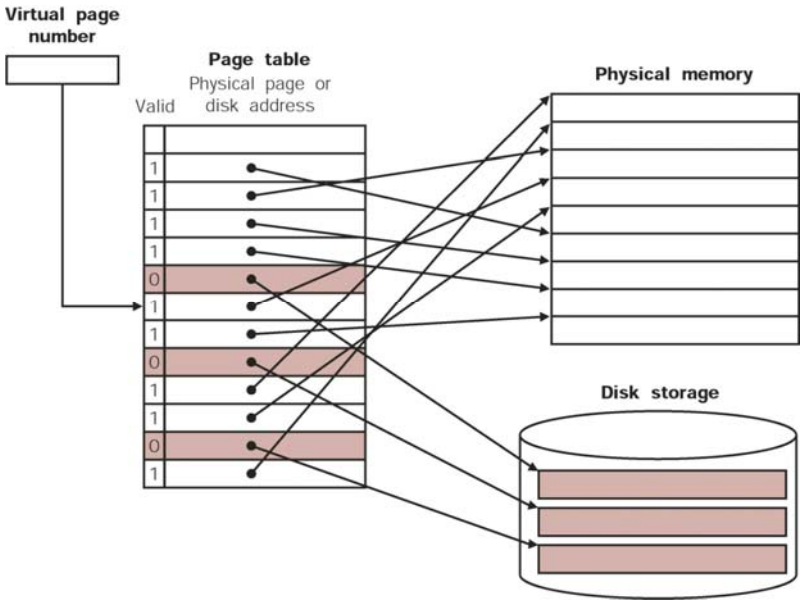
Address translation

- Page table access

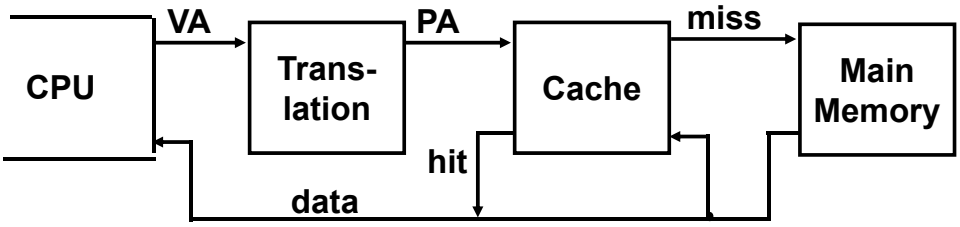


Address translation

- Page table access
  - If the valid bit of the page table is zero, this means that the page is not in main memory. → A page fault occurs, and the missing page is read in from disk.



The TLB: faster address translation



- Major problem: for each instruction or data access we have to first access the page table in memory to get the physical address

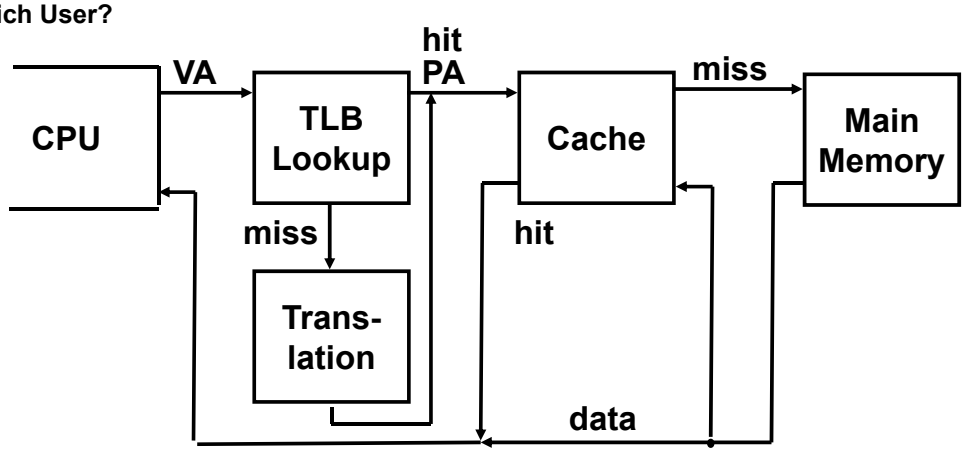
- Solution: *cache* the address translations in a ***Translation Lookaside Buffer (TLB)*** in hardware
  - The TLB holds the ppns of the most recently accessed pages
  - Hardware first checks the TLB for the ppn; if not found (*TLB miss*), then the page table is accessed to get the ppn
  - The ppn is loaded into the TLB for later access

TLB organization: include protection

- TLB usually organized as fully-associative cache
  - Lookup is by Virtual Address
  - Returns Physical Address + other info
- Other info.

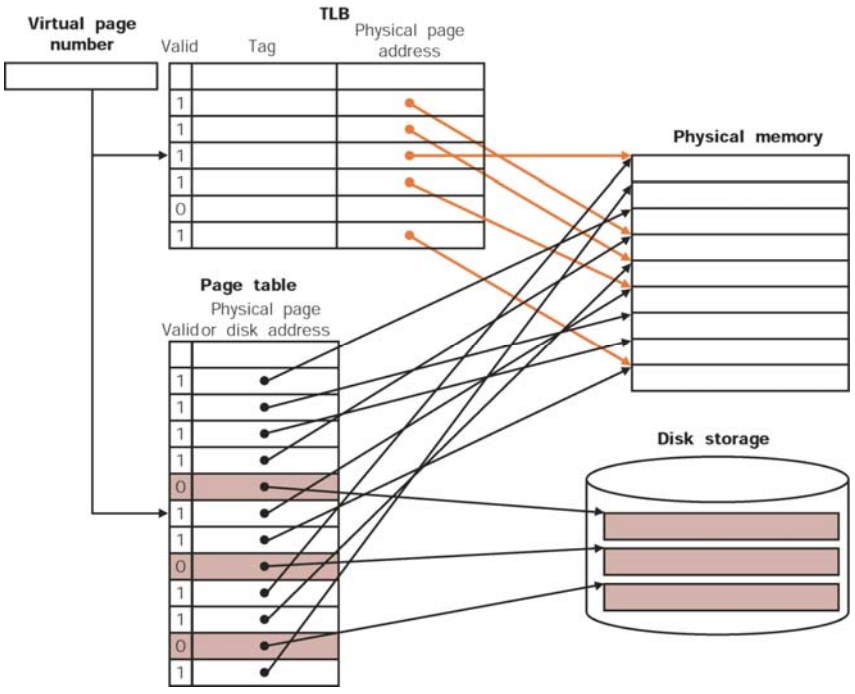
  Dirty => Page modified (Y/N)?
  Ref => Page touched (Y/N)?
  Valid => TLB entry valid (Y/N)?
  Access => Read? Write?
  ASID => Which User?

Translation with a TLB



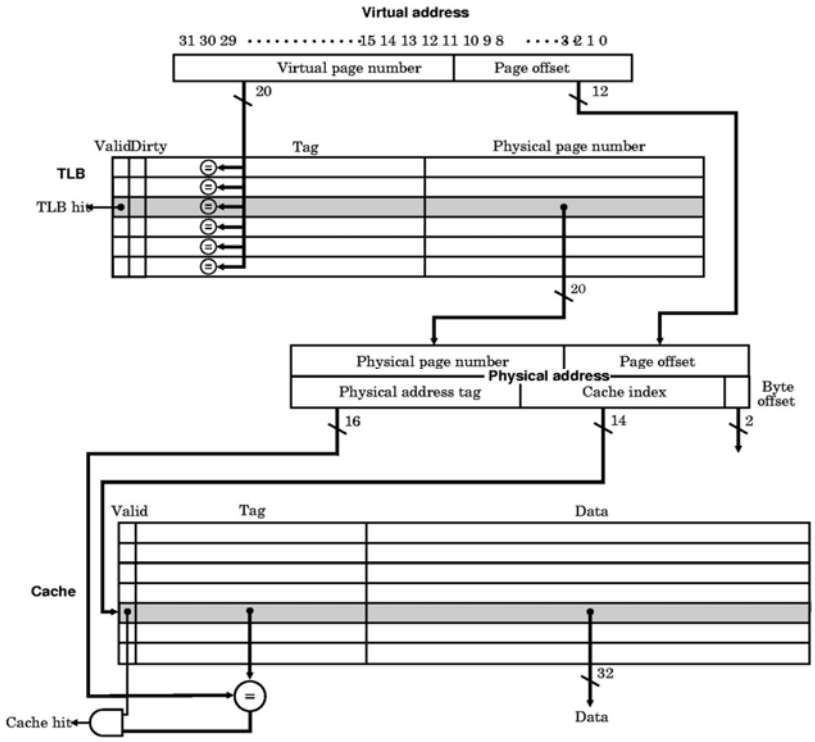
Address translation

- Page table and TLB access



Accessing the TLB and the cache

- DECStation 3100 fully associative TLB and Dcache



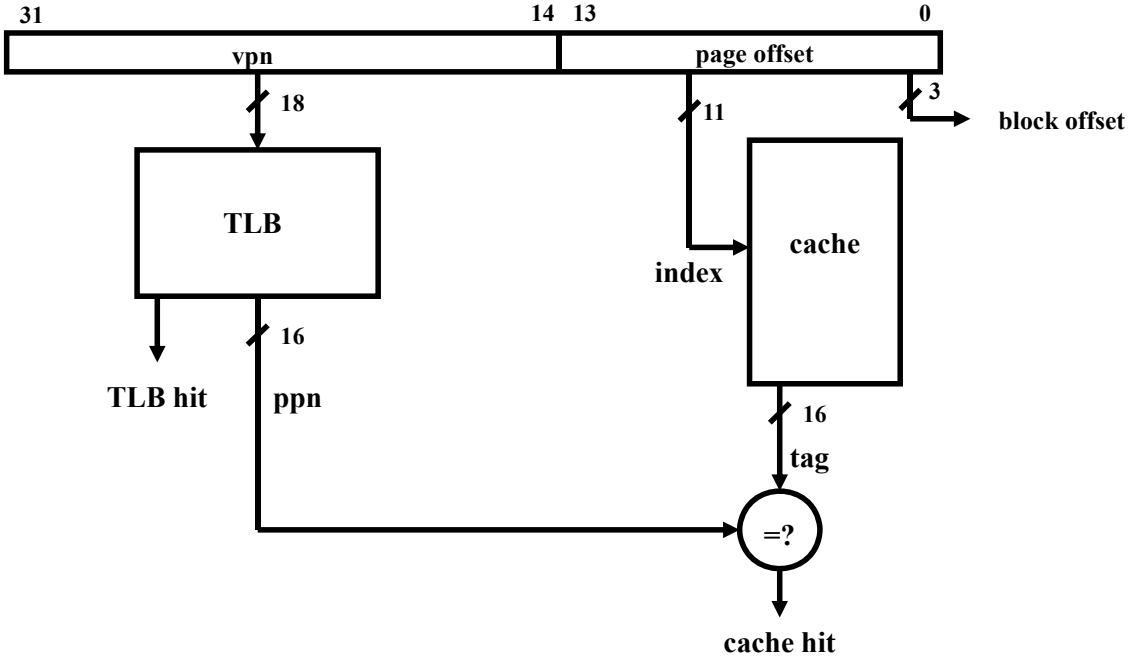
Accessing the TLB and the cache

- Drawback: have to access the TLB first before accessing the cache (increases cache hit time)

- *Virtually indexed, physically tagged cache*
  - Organize the cache so that only the page offset bits (or a subset) are necessary to index the cache
  - Access the cache and TLB in parallel
  - Compare the ppn from the TLB and remaining page offset bits with the tag read out of the cache to determine cache hit/miss

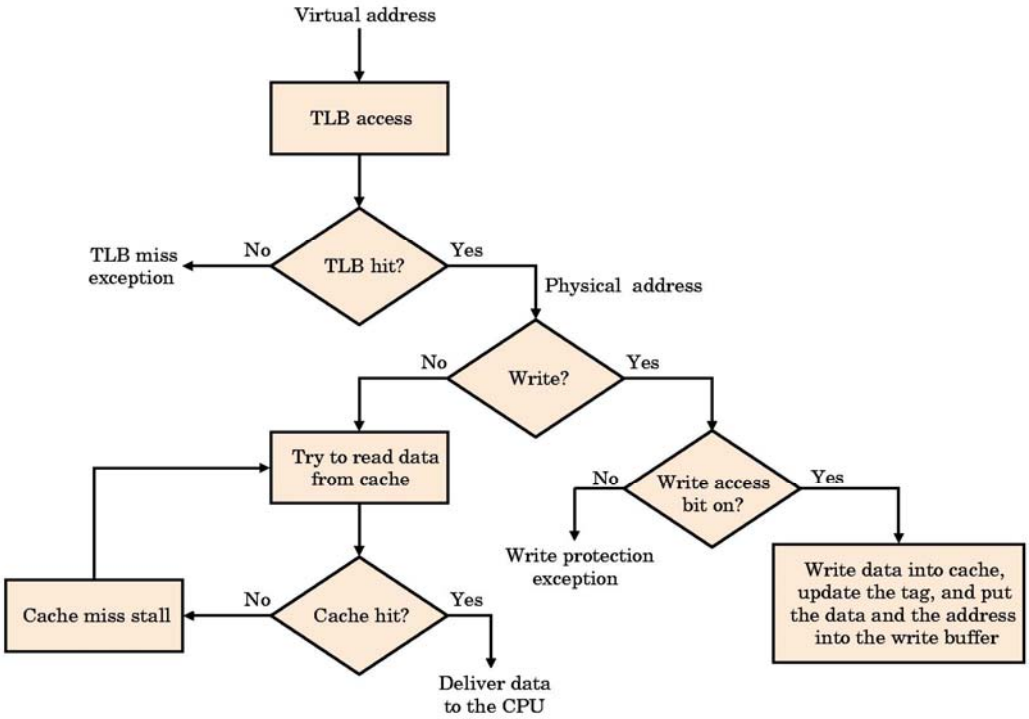
Virtually indexed, physically tagged cache

- **Example: 16KB page size, 16KB direct-mapped Dcache with 8B blocks, 1GB of main memory**



Accessing the TLB and the cache

- **TLB and Dcache operation**



Handling TLB Misses and Page Faults

- **When a TLB miss occurs, microcode or a small OS routine**
  - **Select an entry for replacement**
  - **Write back any special bits of the replaced entry that may have been modified (e.g., dirty bit) to the page table**
  - **Access the page table entry corresponding to the vpn**
  - **If Valid=0, generate a page fault exception**
  - **Otherwise, load the information from the page table into the TLB, and retry the access**

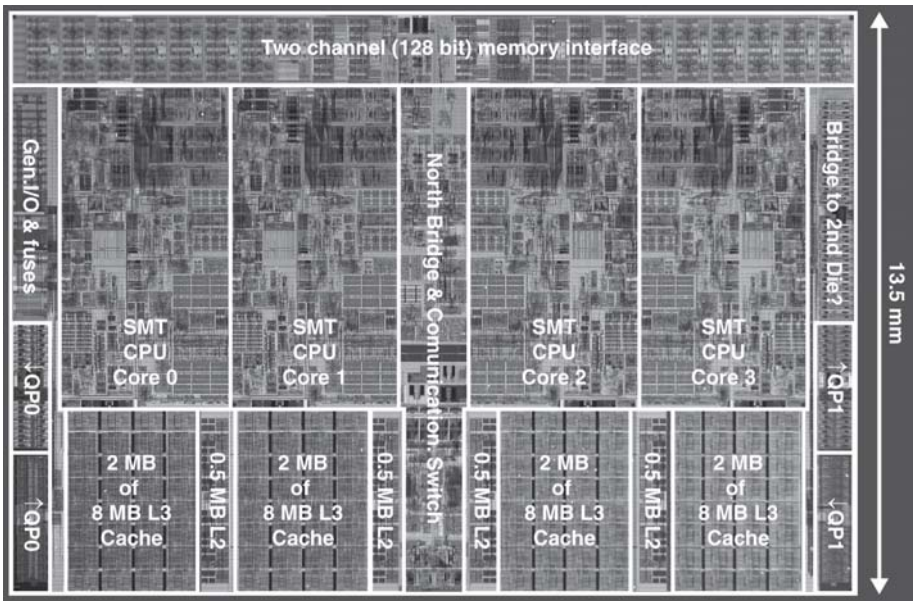
- **When a *page fault exception* invokes the OS, then the OS**
  - **Saves the state (registers) of the running process, changes its status to *idle***
  - **Selects another process to run on the CPU while the disk is accessed**
  - **Access the page table to determine the physical location of the page on disk**
  - **Chooses a physical page to replace - if the replaced page is dirty it is written to disk**
  - **Reads a page from disk into the chosen physical page in main memory**
  - **updates the page table and changes the status of the first process to *runnable***

Memory Protection

- **TLB and page table entries also have a *write access* bit that is set only if the process has permission to write the page**
  - **Example: you have read-only access to a text editor page, but both read and write access to the data page**
  - **Attempts to circumvent these mechanisms causes a *memory protection violation* exception**
- **A memory is often shared by several programs or processes**
- **Mechanisms for providing protection between multiple processes**
  - **Provide both user and supervisor (operating system) modes**
  - **Provide CPU state that the user can read, but cannot write**
    - user/supervisor bit, page table pointer, and TLB
  - **Provide method to go from user to supervisor mode and vice versa**
    - system call or exception : user to supervisor
    - system or exception return : supervisor to user
  - **Provide permissions for each page in memory**
  - **Store page tables in the operating systems address space - can't be accessed directly by user.**

Multilevel On-Chip Caches

Intel Nehalem 4-core processor



Per core: 32KB L1 I-cache, 32KB L1 D-cache, 512KB L2 cache

2-Level TLB Organization

	Intel Nehalem	AMD Opteron X4
Virtual addr	48 bits	48 bits
Physical addr	44 bits	48 bits
Page size	4KB, 2/4MB	4KB, 2/4MB
L1 TLB (per core)	L1 I-TLB: 128 entries for small pages, 7 per thread (2×) for large pages L1 D-TLB: 64 entries for small pages, 32 for large pages Both 4-way, LRU replacement	L1 I-TLB: 48 entries L1 D-TLB: 48 entries Both fully associative, LRU replacement
L2 TLB (per core)	Single L2 TLB: 512 entries 4-way, LRU replacement	L2 I-TLB: 512 entries L2 D-TLB: 512 entries Both 4-way, round-robin LRU
TLB misses	Handled in hardware	Handled in hardware

3-Level Cache Organization

	Intel Nehalem	AMD Opteron X4
L1 caches (per core)	L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU replacement, hit time n/a L1 D-cache: 32KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	L1 I-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, hit time 3 cycles L1 D-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, write-back/allocate, hit time 9 cycles
L2 unified cache (per core)	256KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	512KB, 64-byte blocks, 16-way, approx LRU replacement, write-back/allocate, hit time n/a
L3 unified cache (shared)	8MB, 64-byte blocks, 16-way, replacement n/a, write-back/allocate, hit time n/a	2MB, 64-byte blocks, 32-way, replace block shared by fewest cores, write-back/allocate, hit time 32 cycles

n/a: data not available

Miss Penalty Reduction

- **Return requested word first**
  - **Then back-fill rest of block**
- **Non-blocking miss processing**
  - **Hit under miss: allow hits to proceed**
  - **Mis under miss: allow multiple outstanding misses**
- **Hardware prefetch: instructions and data**
- **Opteron X4: bank interleaved L1 D-cache**
  - **Two concurrent accesses per cycle**