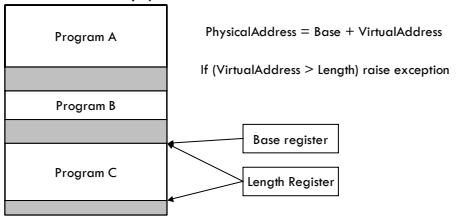
Address Translation & Virtual Memory

Evolution

- Initially, each program ran alone on the machine, using all of the available memory.
- It was linked at a known starting address (like 0).
- All memory addresses were *physical*.
- Problem: This single program model doesn't utilize resources well. When a program blocks for I/O, the CPU sits idle for a long time.
- Solution: Do some other work in the meantime: *multiprogramming*.
- Issues:
  - Protection, sharing, addressing.

Solution: Base & Length Registers

- Compile/link programs starting at address zero.
- Place programs into contiguous free blocks of memory and translate *virtual addresses* into *physical addresses*:



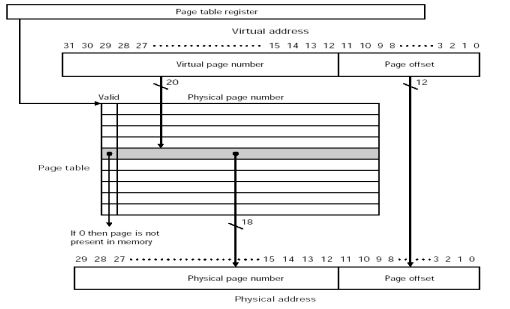
Relocation & Protection

- Base/Length registers support relocation & protection.
- Each program thinks it is the only program in memory, starting at address zero.
- All addresses are translated (by hardware) via the base register.
- The length register provides protection.
- *Fragmentation* is the main problem:
  - As programs come and go, memory get chopped up.
  - There may be enough total memory for a program to run, but it must be contiguous.

Paging

- Basic idea: divide the virtual address space up into equal sized chunks: *pages*.
- Divide physical memory into equal sized chunks: *frames*.
- Provide relocation information for every program, so any virtual page can be mapped to any physical frame.
- Memory hierarchy: physical memory acts like a fully associative cache between the processor and disk. Pages are blocks.
- Disk transfers are costly, so:
  - Make pages big, to amortize cost of transfer
  - Write-back policy is used.

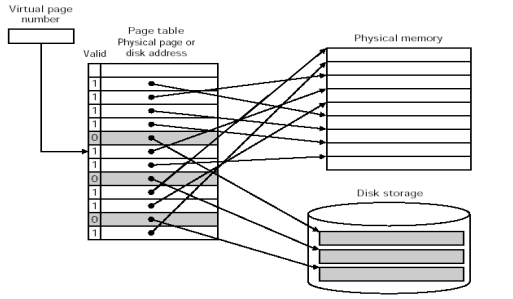
Software Mechanism: Page Table



Page Tables

- Paging allows for a virtual address space that is larger than the physical memory.
- Each page table entry (PTE) indicates:
  - where the virtual page lives (physical frame)
  - valid bit: is the page in memory?
  - dirty bit: has the page been modified
  - protection bits: used to control read/write access
  - reference bits: used for replacement policy
- A program can run without having all of its pages in memory. The unused pages reside on disk.

Page Tables



Processes

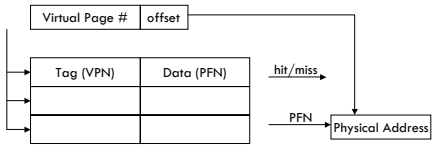
- A process (a program in execution) is defined by:
  - Registers: PC, stack pointer, general registers
  - Page table(s)
  - Bookkeeping: open files, process ID, time used, etc
- On a uniprocessor, only one process runs at a time. Switching from one process to another is called a *context switch*.
- Switching from A to B requires: saving A's state (registers, etc) and then restoring B's state, and jumping to B's PC.
- Different states: running, ready, waiting

Protection & Sharing

- *Protection*: a program cannot generate an address that accesses another program's data.
- Processes cannot be allowed to modify their own page tables, obviously...
- Sharing: If two PTE's from different process point to the same physical frame, then those processes can share that data.

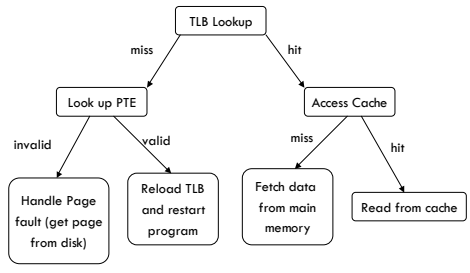
Speeding Translation: TLBs

- To do an address translation, we have to do a lookup in the page table.
- Translation costs (at least) one extra memory access.
- Solution: build special hardware (Translation Lookaside Buffer) to "cache" the PTEs



TLBs are usually small (~64 entries) and fully associative

Memory Accesses



TLB Organization

- TLBs are small caches holding TLBs.
- MIPS: fully associative, write-allocate, write-back, random replacement. 64 entries.
- Looking up a PTE (on a TLB miss) and putting it into the TLB is accomplished in software (10-30 cycles).
- What happens on a context switch? The PTEs are no longer valid for the new process. Options:
  - Flush the TLB on each context switch (expensive)
  - Append a process ID to the virtual address. This way, the TLB can hold entries for more than one process.

Page Faults

- Pages live in memory or on disk.
- Page fault: when a program references a page that is not in memory.
- Resolving the fault takes a long time, because we have to go to disk.
- The OS resolves it as follows:
  - Find a free physical frame (on a free list or a frame needs to be replaced)
  - Find where the faulting page resides on disk
  - Initiate the read from disk into the memory frame.
  - Now switch in a new process because the disk operation will take a long time.
  - When the transfer completes, modify the PTE to make it valid and restart the faulting program.

Summary

- VM is just another level of the memory hierarchy
- pages = blocks; faults = cache misses
- Misses are expensive. Keep the miss rate low by:
  - large blocks
  - fully associative mapping (need page tables)
  - careful replacement (see CSE451)
- Writes are expensive. Use a write-back scheme.
- Address translation is key: it provides protection, sharing, memory mapping.
- Translation is done in hardware, mostly.