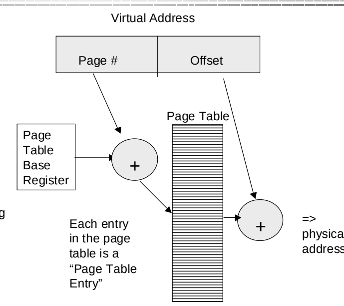
Paging and Segmentation

Brian Bershad

Paging

- Goals
  - » make allocation and swapping easier
- Make all chunks of memory the same size
  - » call each chunk a "PAGE"
  - » example page sizes are 512 bytes, 1K, 4K, 8K, etc
  - » pages have been getting bigger with time
  - » we'll discuss reasons why pages should be of a certain size as the week progresses.



An Example

- Pages are 1024 bytes long
  - » this says that bottom 10 bits of the VA is the offset
- PTBR contains 2000
  - » this says that the first page table entry for this process is at physical memory location 2000
- Virtual address is 2256
  - » this says "page 2, offset 256"
- Physical memory location 2004 contains 8192
  - » this says that each PTE is 4 bytes (1 word)
  - » and that the second page of this process's address space can be found at memory location 8192.
- So, we add 256 to 8192 and we get the real data!

What does a PTE contain?

1	1	1	1-2	about 20
M-bit	R-bit	V-bit	Protection bits	Page Frame Number

- The Modify bit says whether or not the page has been written.
  - » it is updated each time a WRITE to the page occurs.
- The Reference bit says whether or not the page has been touched
  - » it is updated each time a READ or a WRITE occurs
- The V bit says whether or not the PTE can be used
  - » it is checked each time the virtual address is used
- The Protection bits say what operations are allowed on this page
  - » READ, WRITE, EXECUTE
- The Page Frame Number says where in memory is the page

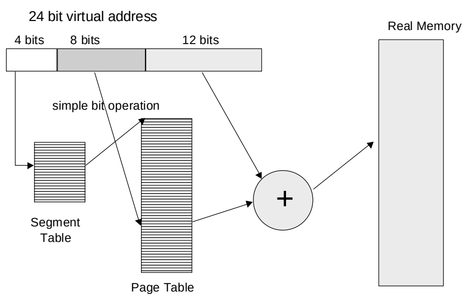
Evaluating Paging

- Easy to allocate memory
  - » memory comes from a free list of fixed size chunks.
  - » to find a new page, get anything off the free list.
  - » external fragmentation not a problem
- easy to swap out pieces of a program
  - » since all pieces are the same size.
  - » use valid bit to detect references to swapped pages
  - » pages are a nice multiple of the disk block size.
- Can still have internal fragmentation
- Table space can become a serious problem
  - » especially bad with small pages
    - – **eg, w/a 32bit address space and 1k size pages, that's 2^{22} pages or that many ptes which is a lot!**
- Memory reference overhead can be high
  - » 2 refs for every one

Segmentation and Paging at the Same Time

- Provide for two levels of mapping
- Use segments to contain logically related things
  - » code, data, stack
  - » can vary in size but are generally large.
- Use pages to describe components of the segments
  - » makes segments easy to manage and can swap memory between segments.
  - » need to allocate page table entries only for those pieces of the segments that have themselves been allocated.
- Segments that are shared can be represented with shared page tables for the segments themselves.

An Early Example -- IBM System 370

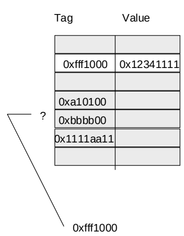


Lookups

- Each memory reference can be 3
  - » assuming no fault
- Can exploit locality to improve lookup strategy
  - » a process is likely to use only a few pages at a time
- Use Translation Lookaside buffer to exploit locality
  - » a TLB is a fast associative memory that keeps track of recent translations.
- The hardware searches the TLB on a memory reference
- On a TLB miss, either a hardware or software exception can occur
  - » older machines reloaded the TLB in hardware
  - » newer RISC machines tend to use software loaded TLBs
    - – **can have any structure you want for the page table**
    - – **fast handler computes and goes. Eg, the MIPS.**

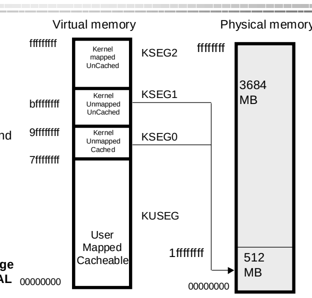
A TLB

- A small fully associative cache
- Each entry contains a tag and a value.
  - » tags are virtual page numbers
  - » values are physical page table entries.
- Problems include
  - » keeping the TLB consistent with the PTE in main memory
    - – **valid and ref bits, for example**
  - » keeping TLBs consistent on an MP.
  - » quickly loading the TLB on a miss.
- Hit rates are important.



MIPS R3000 VM Architecture

- User mode and kernel mode
  - » 2GB of user space
  - » When in user mode, can only
  - » access KUSEG.
- Three kernel regions
  - » KSEG0 contains kernel code and data, but is unmapped. Translations are direct.
  - » KSEG1 like KSEG0, but uncached. Used for I/O space.
  - » KSEG2 is kernel space, but cached and mapped. Contains page tables for KUSEG.
    - – **Implication is that the page tables are kept in VIRTUAL memory!**



Software Loaded TLB

- The MIPS TLB contains 64 entries
  - » fully associative
  - » random replacement
- On a TLB miss to KUSEG, a trap occurs
  - » control transfers to UTLBMISS handler
- If a miss occurs in the miss handler, a second fault occurs.
  - » control transfers to a KTLBMISS handler
  - » the missed VA is "pte"
  - » the translation for the page table is loaded into the TLB.

```
UTLBmiss(vm_offset_t va)
{
  PTE_t pte;
  pte = PCB.PTBR;
  pte = pte +  (va / PAGE_SIZE);
  TLB_dropin(va,
         *pte);  /* could miss here */
 }
```


Selecting a page size

- Small pages give you lots of flexibility but at a high cost.
- Big pages are easy to manage, but not very flexible.
- Issues include
  - » TLB coverage
    - – **product of page size and # entries**
  - » internal fragmentation
    - – **likely to use less of a big page**
  - » # page faults and prefetch effect
    - – **small pages will force you to fault often**
  - » match to I/O bandwidth
    - – **want one miss to bring in a lot of data since it will take a long time.**