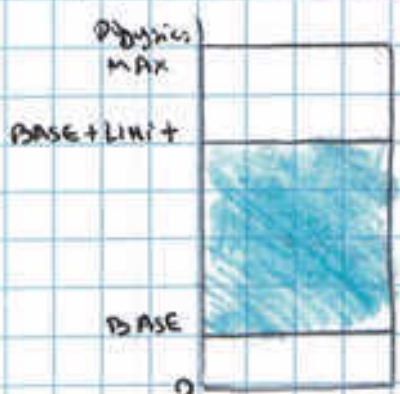
Relocation and Protection solution

- A solution is to equip machine with two registers called base and limit registers



- Relocate Addresses in Executable Binary
- Relocation: Address locations added to base value to map to physical address
- Protection: Address location larger than limit value is an error

- Additional benefit is that if a program is moved within memory, only its base register needs to be amended. this is a lot quicker than modifying every address reference within the program.

Which are the solution for relocation and protection

What kind of additional benefit you obtain?

[redacted] self study

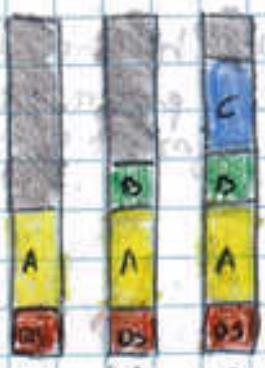
SWAPPING

- Physical memory may not be enough to accomodate needs of all processes

What is swapping

How does swapping work?

- swapping consist of bringing in each process in its entirety, running it for a while, then putting it back on the disk
  - Idle processes are mostly stored on disk, so they do not take up any memory when they are running (some of them wake up periodically to do their work, then go to sleep.



① Initially only process A is in memory (a)
② Then process B and C are created or swapped in from the disk (b) (c)

③ A is swapped out of disk (d)
④ D comes in and B goes out (e) (f)

⑤ Finally A comes in again

- since A is now at different location, addresses contained must be relocated

Swapping (Continue)

~~• Memory allocation~~

- When swapping creates multiple holes in memory, it is possible to combine them all into one big one by moving all process downward as far as possible. This is called memory Compaction
  - It is normally not done because it requires a lot of CPU time.

~~All processes created w~~

- If processes created with fixed size that never changes, then the allocation is simple: OS allocate exactly what is needed
- If processes data segments can grow (dynamic allocate memory) the problem goes when a process tried to grow
  - If a ~~process~~ hole is adjacent to the process, it can be allocated and the process allowed to grow in the hole
  - If the process is adjacent to another process, the growing process will either have to be moved to a hole in memory large enough for it
- Memory allocation changes as:
  - → Processes come into memory
  - → leave memory and are swapped out to disk
  - → re-enter memory by getting swapped-in from disk

← • Shared regions are unused memory

- When swapping processes disk, only the memory actually in use should be swapped

What happens when swapping create multiple holes in memory?

What happens when the process has a fixed size?

What happens when the process has a dynamic size?

How does memory allocation change?

Which parts should be swapped

[redacted] self study

VA - virtual Address

Managing Free Memory

- When memory is assigned dynamically, the OS must manage it, there are two ways: bitmaps & freelist.

Virtual Memory

- While swapping is useful, when the sum total of memory requirement of all processes is greater than DRAM available in the system but sometimes, a single process might require more memory than available DRAM in the system

- In cases where swapping is not enough, we need to break up the memory space of a process into smaller equal-size pieces (called pages)

- the OS decides which pages stay in memory and which get moved to disk

- Definition of Virtual Memory:
  Each process gets an illusion that it has more memory than the physical DRAM in the system

- the basic idea is that each program has its own address space, which is broken up into chunks called pages.

- Each page is a continuous range of addresses

62

Virtual Memory (continue)

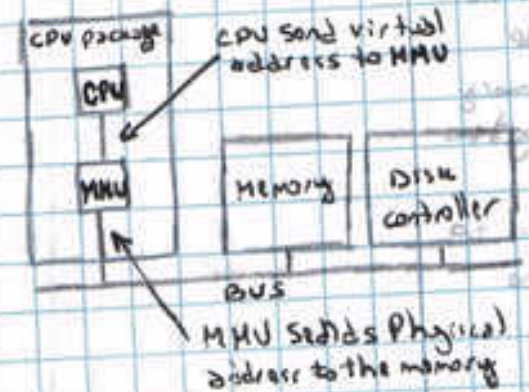
- Pages are mapped onto a physical memory but not all pages have to be in physical memory to run the program

- When a program references a part of its address space that is in physical memory, the hardware performs the mapping on the fly.

- When the program references a part of its address space that is not in physical memory, the OS is alerted to go get the missing piece and re-execute the instruction failed.

Do all the pages are loaded on the physical memory?

Virtual Memory and MMU (Memory Management Unit)



- MMU is part that accompanies the CPU,
- MMU converts Virtual Address to Physical Addresses
- Most virtual system use a technique called Paging

Is the MMU part of the CPU?

What does the MMU does?

Paging

1. let say a program execute an instruction like MOV REG,1000 which copy the content of memory address 1000 to REG

2. this program-generated addresses are called ~~virtual Address~~ Virtual Addresses.

What is Paging?

What is a virtual address?

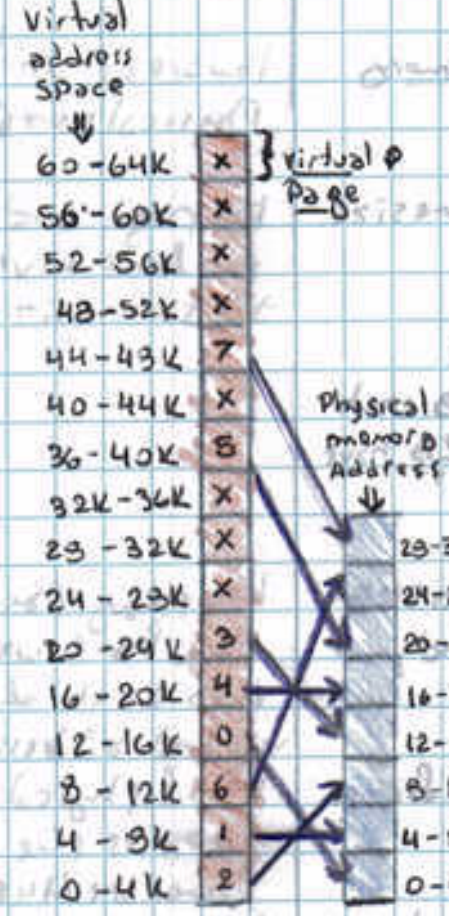
- On computers without virtual memory, virtual address is put directly onto the memory bus and it causes the physical memory word with the same address to be read or write

[redacted] self study

Paging (Continue)

- when virtual memory is used, the virtual address do not go directly to the memory bus. Instead they go to to an Memory Management Unit (MMU)

Memory Management Unit(MMU) and Paging



- the relation between virtual address and physical memory address given by page table.
- Every page begins on a multiple of 4096 and ends 4095 addresses higher so 4K-8K really means 4096-8191 and 8K-12K means 8192-12287.
- Memory Management unit(MMU) maps the virtual addresses onto the physical memory

Ex: ① let assume we have a computer that generates 16-bits addresses from 0 to 64K. These are virtual addresses

② However we only have 32K of physical memory.

what happens when virtual memory is used?

what provide the relation between the virtual address and physical memory address given?

Every page start with and with?

What does the MMU?

Memory Management Unit and Paging (continue)

(3) Although 64KB programs can be written, they cannot be loaded into memory in their entirety and run

(4) Copy of the core image of the program, up to 64KB must be present in Disk

(5) virtual address space is divided into fixed-sized unit called Pages

(6) the corresponding units in the physical memory are called page Frames

(7) Pages and Page Frames are (generally) the same size.

(8) Example:

(1) We have pages and page frames of 4KB each but pages sizes from 512 Bytes to 64 KB have been used in real systems

(2) With 64KB of virtual Address space and 32KB of physical memory, we get 16 pages and 8 page frames.

$$\frac{\text{virtual Address space } 64\text{KB}}{4\text{KB}} = 16 \text{ pages} \qquad \frac{\text{Physical memory } 32\text{KB}}{4\text{KB}} = 8 \text{ page frames}$$

Note: transfers between RAM and disk are always done in whole pages

Can 64KB programs be loaded into memory entirely?

Do the core image of the program up to 64KB be present in disk?

How virtual address space is divided?

How are units of physical memory?

Do pages and page frame have the same size?

How many pages or page frame we get with 64KB of virtual address and 32KB physical memory is the page frame are 4KB of fixed size?

Do transfers between RAM and disk are done in whole or half pages?

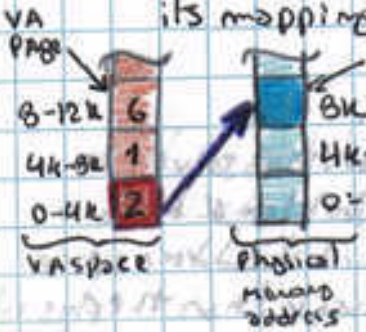
[redacted] self study

Memory Management Unit (MMU) and Paging (continue)

- Let the program try to access address 0, using instruction MOV REG,0

① Virtual Address 0 is sent to MMU.

② MMU sees that this virtual address falls in page 0 (0 to 4095), which according to its mapping is page frame 2 (8192 to 12287)

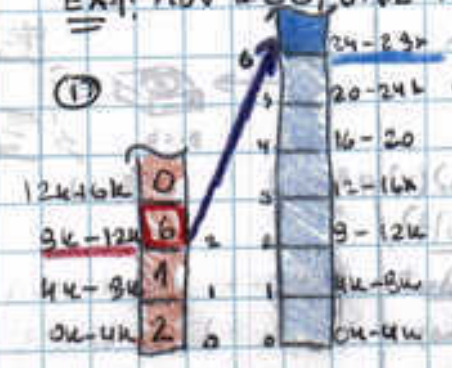


- transform the address 8192 and outputs address 8192 onto the bus.

- The memory doesn't know at all about the MMU and just sees a request for reading or writing 8192.

- The MMU mapped all virtual addresses between 0 and 4095 onto Physical address 8192 to 12287

Ex1: MOV REG, 8192 is transformed to MOV REG 24576



- Virtual Address 8192 is mapped onto 24574
- so virtual page 2 is mapped onto physical page frame 6

66 Memory Management Unit and Paging (cont. ne)

EX2

- Let say we have a virtual address 20500
- VA 20500 is 20 bytes from the start of virtual page 5 so (virtual page 5) 20480 + 20
- Therefore 20500 is between 20480 and 24575
- then of the physical address it would be 12288+20 = 12308

- One problem this ability to map 16 virtual pages onto any of the 8 page frames doesn't solve Problem that virtual address space is larger than physical memory
  - only 8 page frame can be used so only 8 virtual pages can be mapped.
- Present/absent bit: keep track which pages are physical present in memory
- the other virtual pages are unmapped (X)

How does the MMU solve the problem of having a virtual address space larger than the physical memory?

What does the present/absent bit does?

Ex:

EX3:

- MOV Reg 32780 which is byte 12 within page 8 (start at 32768).
- MMU notice that the page is unmapped (X) & cause the CPU to trap the OS (page fault)
- OS picks a little used page & writes it content back to disk (if not true), then it fetches the page just referenced into the page frame just freed, changes the map, and restart the trapped instruction

Memory Management Unit and Paging (continue)

Ex: ① OS decide evict page frame 1,

② OS would load virtual page 8 at physical address 8192

③ Then move two changes to the MMU map:

Ⓐ Mark virtual page 1's entry as unmapped (to trap any future accesses to virtual addresses between 4096 and 8191)

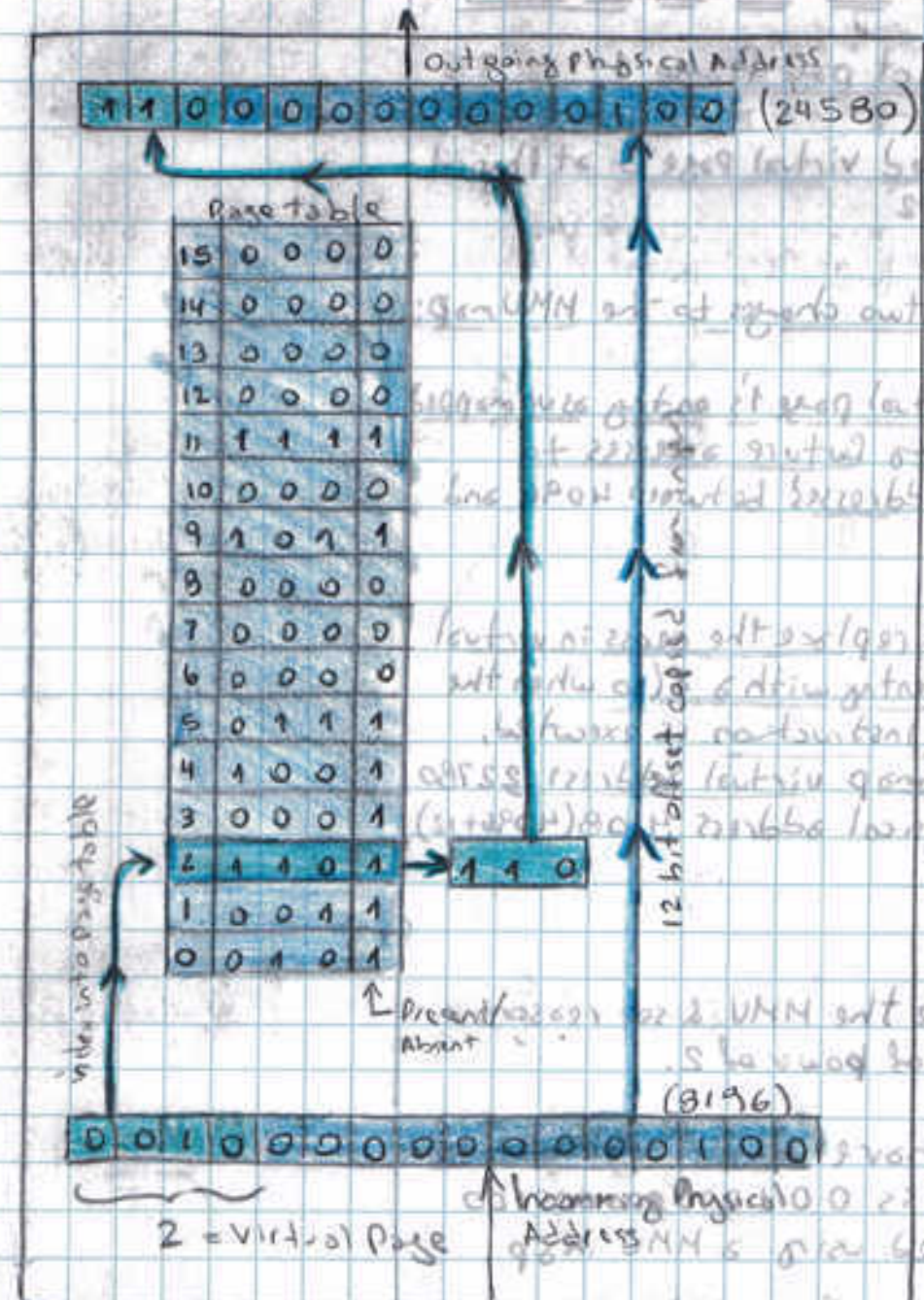
-then-

Ⓑ It would replace the cross in virtual page 8's entry with a 1 (so when the trapped instruction is executed, it will map virtual address 32780 to physical address 4108 (4096+12))

Page Table

- let us look inside the MMU & see reason to use a page size of power of 2.
- let assume we have a virtual addres 8196 in binary it is 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 and it is mapped using a MMU map

Page tables (continue)



Internal operation of the MMU with 16 4KB pages

① the incoming 16-bits virtual page address is split into 4-bits (index into page table) and 12 bits (offset)

② the page # is used as index into pagetable which yield the number of page frame corresponding to the virtual page

How does the pagetable work?

How do we the 16-bits virtual address split?

How use the page number?

What does the pagetable yield?

[redacted] self study

③ If the Present/Absent bit is 0 then trap the OS is caused
If the present/Absent bit is 1 then page frame number found in the page table is copied to the higher-order 3bits of the out put register along with the 12 bit offset (copied unmodified) from the incoming virtual address forming a 15-bit physical address

④ output register is then put onto the memory bus as the physical memory address

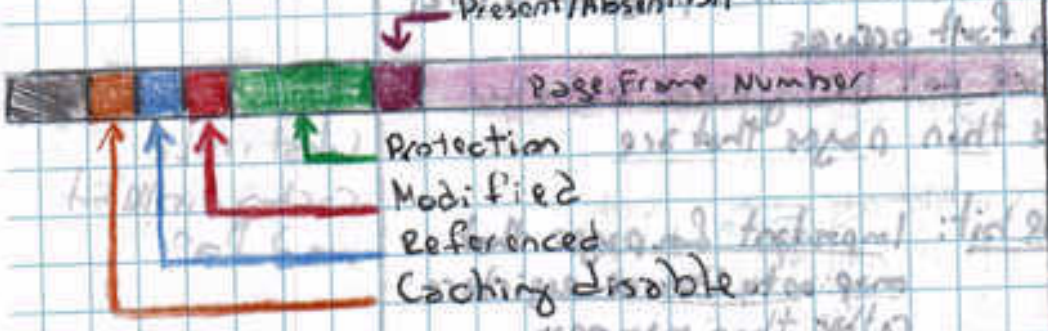
What is the present/absent bit used for?

What is the present/absent bit?

Where the output register goes after?

Structure of a Page Table entry

- this is a details of a single page entry of 32 bits
- Most important field is the Page frame number



What is the most important field in the structure of page table entry

- Present/Absent: 1 - entry is valid and can be used
  0 - virtual page to which the entry belongs is not currently in memory
  Accessing a page table entry w/ this bit set to 0 causes a page fault

What is the goal of page mapping

- Goal of page mapping is to output a correct page frame number.

Structure of a Page table entry (continue)

- **Protection bits**: tells what kind of access are permitted.
  - Contains 1 bit = 0 - read/write
    1 - read only
  - More sophisticate use 3 bits for writing, Ready, Execute

(dirty bit)
- **Modified & Referenced bits**: keep track of page usage
  → when page is written, hardware sets the modified bit
  this bit is of value for the OS when decides to reclaim page
  - If page was modified ("dirty") it must be written back to disk
  - If page was not being modified ("clean") it can just be abandoned since the disk copy still valid

~~→ not dirty~~

→ **Referenced bits**: set whenever a page is referenced for reading or writing
  - It is use for the OS to choose a page to be evict when a page fault occurs
  - Pages that are not being used are better candidates than pages that are

- **Caching disable bit**: Important for pages that map onto device registers rather than memory
  → if OS is sitting in a tight loop waiting for some I/O device to respond to a command it was just given, hardware need to keep fetching the word from device, and not use an old cached copy

What is the protection bit used for?

What is the modified bit used for?

What is the reference bit used for?

What the modified and the reference bits are reused together?

What is the caching disable bit used for?

[redacted] self studying

Speeding Up Paging

Two problems in paging systems:

1. The mapping from virtual Address to physical address must be fast because mapping must be done on every memory reference. Avoid bottleneck!
2. If the virtual address space is large, the page table will be large. Modern computers use virtual addresses of at least 32 bits (64 bits becoming common). With 4 KB page size, 32 bit address space equals 1 million pages in which each process needs its own page table.

What are the two problems in paging systems?

two designs for the need for large, fast pages:

1. One extreme: single page table consisting of an array of fast hardware registers, with one entry for each virtual page, indexed by virtual page number (as in the internal operation of the MMU with 16 4-KB pages).
   When a process is started up, the OS loads the registers w/ the process' page table, taken from a copy kept in main memory.
   During ~~execution~~ execution, no more memory references are needed for the page table.
   - Advantage: it is straightforward and requires no memory references during mapping.
   - Disadvantage: it is unbearably expensive if the page table is large. Another is having to load the full page table at every context switch hurts performance

Explain the extreme cases for large fixed pages.

What is the advantage of 2 loadings?

Speeding up paging (continue)

2. Other extreme: Page table can be entirely in main memory.
   Hardware needs single register that points to start of the page table. [this allows the virtual-to-physical map to be changed at a context switch by reloading one register.] Advantage
   Disadvantage: require one or more references to read page table entries during the execution of each instruction, making it very slow.

Translation Lookaside Buffers (TLB)

- schemes for speeding up paging and for handling large virtual address spaces, starting with the former.
- starting point is the page table is in memory. this design has enormous impact on performance:
  eg: 1 byte instruction that copies one register to another: in absence of paging, this makes only one memory reference, to fetch instruction
  - with paging, additional memory reference will be needed, to access the page table.
    Since execution speed is generally limited by the rate at which the CPU can get instruction and data out of memory, having to make two memory references per memory reference reduce performance by half!

What is the disadvantage of this extra

What does TLB mean?

What the TLB is used for?

have an example of how this design have a big impact on the performance

[redacted] self study

Translation Transitions Lookaside Buffers (TLB) (continue)

Solution: Based on observation that most programs have the tendency to make a large number of references to a small number of pages, and not only an small fraction of pages table entries are heavily read; the rest are barely used at all.
Equip computers with a small hardware device for mapping virtual addresses to physical addresses without going through the page table. Hardware device is called translation lookaside buffer (TLB) that directs at some times an associative memory

- TLB is normally inside the MMU.
- TLB consist of small number of entries
  → each entry contains information about one page, including the virtual page number
- A bit is set when the page is modified
- A bit is set when the page is protected to code (read/write/execute permissions)
- A physical page frame that indicate where the page is located
- Each field have one-to-one correspondence with the fields in the page table, except for virtual page number (which is no needed in the page table)
- A bit that indicate entry is valid or not

valid	virtual Page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50

- It is not a Data Cache or Instruction cache. These are separate

(12)

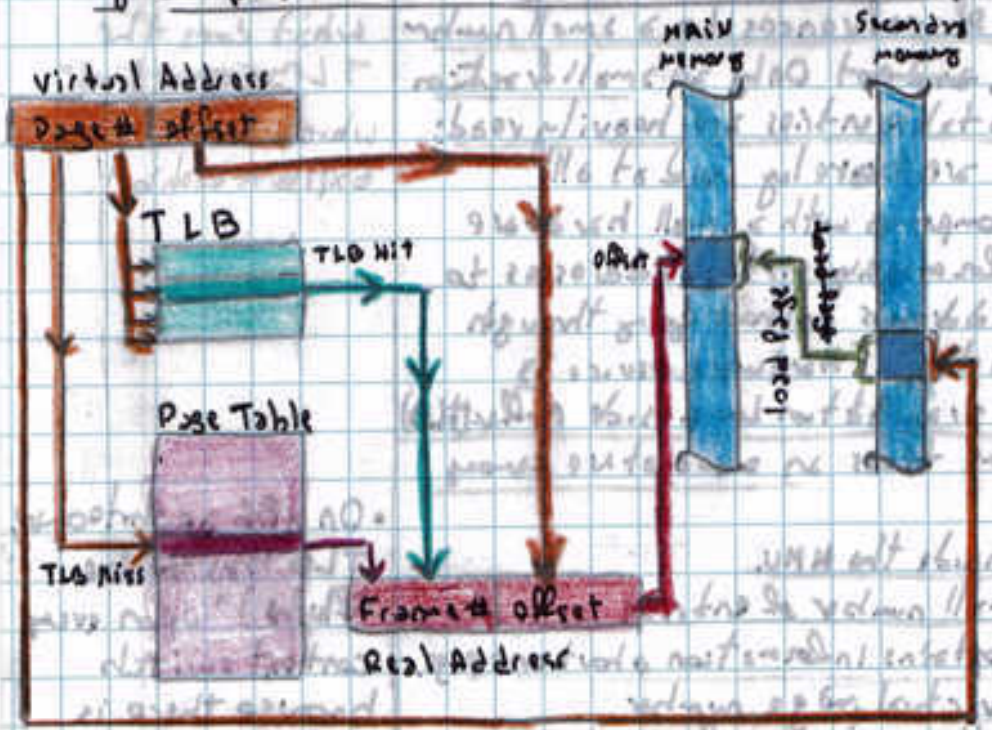
what is the solution implemented? what is the TLB? what does the TLB contain? which bits are set? Explain each bit.

- On x86 architecture, TLB has to be "flushed" upon every context switch because there is no field in TLB to identify the process context.
- TLB simple caches translations from virtual page # to Physical page # so that the MMU don't have to the MMU don't have to access page-table in memory too often.

← ASV

Translation Lookaside Buffer (TLB) (continue)

- When a virtual address is presented to the MMU for translation, the hardware first checks to see if its virtual page number is in the TLB by comparing it to all the entries simultaneously



- When the virtual page # is not in the TLB. The MMU detects the miss and does an ordinary page table lookup. it then evicts one of the entries from the TLB and replaces it with the page table entry just looked up.
  - if that page is used again soon, the second time it will result in a TLB hit rather than a miss.
- When an entry is purged from the TLB, the modified bit is copied back into the page entry in memory. the other values are already there, except the reference bit
- When TLB is loaded from the page table, all fields are taken from memory.

What does the hardware do when a virtual address is presented?

the bigger is the page size, the more virtual Address space is covered = so large page size will reduce miss rate of the TLB

What happens when the virtual page number is not present in the TLB

What happens when an entry is purged from the TLB

What happens when the TLB is loaded from the page table

Software translation lookup buffer (TLB) (continue)

- Many computers such as Risc, sparc, mips, and HPPA do nearly all of the page management in software.

- TLB entries are explicitly loaded by the OS.

- When miss occurs, instead of the MMU going to the ~~table~~ page tables to find and fetch the page reference needed, it generate a TLB fault and pass the problem into the lap of the OS.

  → the OS have to find the page, remove an entry for the TLB, enter a new one, and restart the instruction that faulted. this must be done in an handful of instr. that fault ~~fast~~ because TLBs misses occur much more frequently than page faults (mentioned)
  - if TLB is large to reduce the miss rate, software management of TLB turns out to be acceptably efficient.
  - again is a simpler MMU, which frees up an amount of area on the CPU chip for caches and other features to improve performance

- One approach that both reduces TLB misses and reducing the cost of a TLB miss when it does occurs
  → to reduce TLB misses, the OS use intuition to figure out which page are likely to be used next and to preload entries for them in the TLB.

- The normal way to process a TLB miss is to go to the page & perform the indexing operation to locate the page referenced.

which computer use page management in software?

How load the TLB entries?

what happens when a miss occurs

what does the TLB do to reduce miss rate when the TLB is large (64 entries)?

Explain one approach to reduce the TLB misses

what is the normal way to process a TLB miss?

Software Translation lookaside buffer (TLB) (continue)

→ disadvantages: Pages holding page table may not be in the TLB, which will cause a edit and TLB faults during processing
These faults can be reduced by maintaining a large software cache of TLB entries in a fixed location where page is always kept in the TLB.
By first checking the software cache the OS can substantially reduce TLB misses.

What are the disadvantages of a software TLB

- When software TLB management is used, there are two kind of misses:

What are the two kind of misses that could occur in a software TLB, describe each of

1. Soft miss: When the page referenced is not in the TLB, but is in memory. TLB need to be updated (No disk I/O is needed)

2. Hard miss: When page itself is not in memory (and not in TLB) A disk access is required to bring in the page, which takes several seconds. Hard miss is slower than soft miss