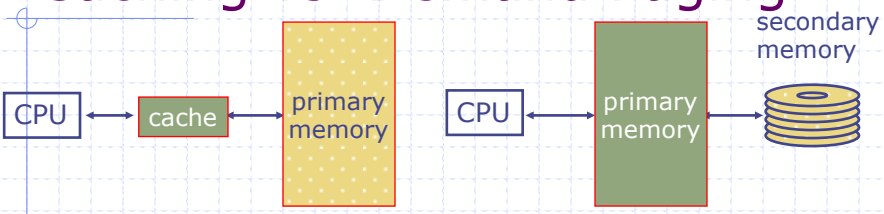
Constructive Computer Architecture

Virtual Memory and Interrupts

Arvind
Computer Science & Artificial Intelligence Lab.
Massachusetts Institute of Technology

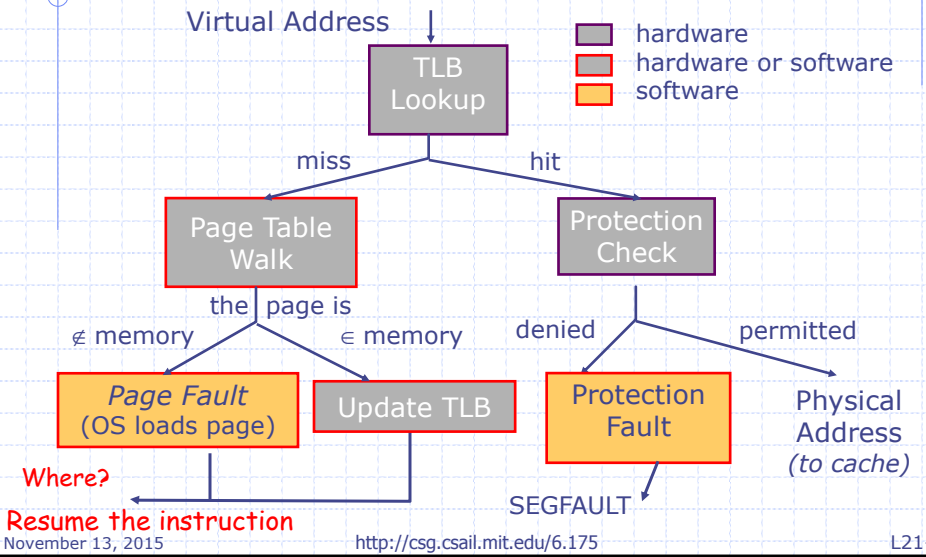
Caching vs. Demand Paging



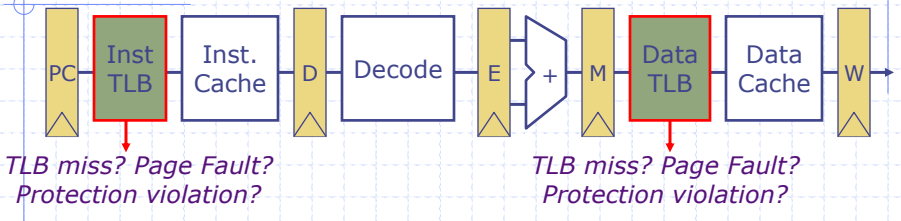
Caching
- cache slot
- cache line (~32 bytes)
- cache miss rate (1% to 20%)
- cache hit (~1 cycle)
- cache miss (~100 cycles)
- miss is handled in *hardware*

Demand paging
- page frame
- page (~4K bytes)
- page miss rate (<0.001%)
- page hit (~100 cycles)
- page miss (~5M cycles)
- miss is handled mostly in *software*

Address Translation:
putting it all together

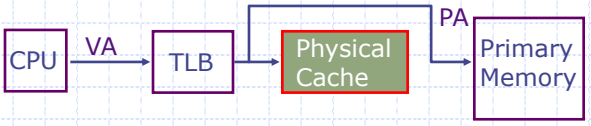


Address Translation in Pipeline Machines

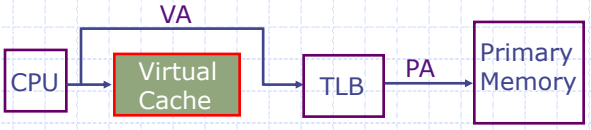


- Software handlers need a *restartable* exception on page fault or protection violation
- Handling a TLB miss needs a *hardware* or *software* mechanism to refill TLB
- Methods to overcome the additional latency of a TLB:
  - slow down the clock
  - pipeline the TLB and cache access
  - √ virtual address caches
  - √ parallel TLB/cache access

Physical *vs* Virtual Address Caches?



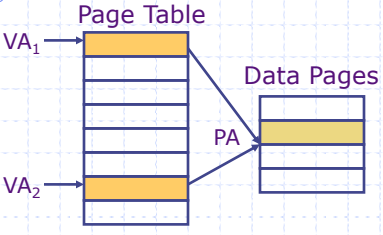
Alternative: place the cache before the TLB



(StrongARM)

- One cycle in case of a hit (+)
- cache needs to be flushed on a context switch unless address space identifiers (ASIDs) included in tags (-)
- *aliasing problems* due to the sharing of pages (-)

Aliasing in Virtual-Address Caches



Two virtual pages share one physical page

Tag	Data
VA_1	1st Copy of Data at PA
VA_2	2nd Copy of Data at PA

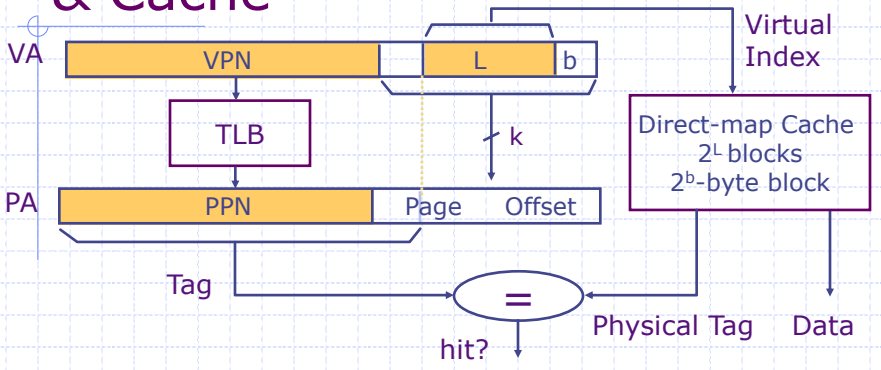
Virtual cache can have two copies of same physical data. Writes to one copy not visible to reads of other!

General Solution: *Disallow aliases to coexist in cache*

Software (i.e., OS) solution for direct-mapped cache

VAs of shared pages must agree in cache index bits; this ensures all VAs accessing same PA will conflict in direct-mapped cache (early SPARCs)

Concurrent Access to TLB & Cache



Index L is available without consulting the TLB

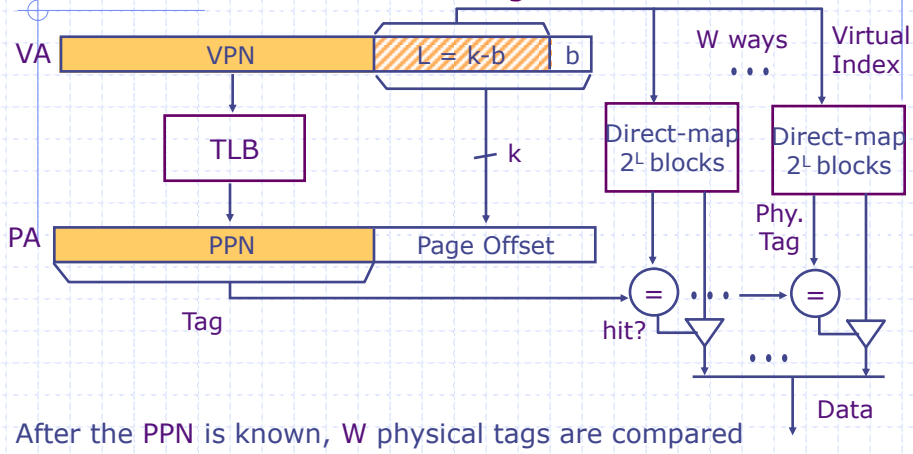
⇒ *cache and TLB accesses can begin simultaneously*

Tag comparison is made after both accesses are completed

Cases: L + b = k L + b < k

L + b > k what happens here? Partially VA cache!

Virtual-Index Physical-Tag Caches: Associative Organization

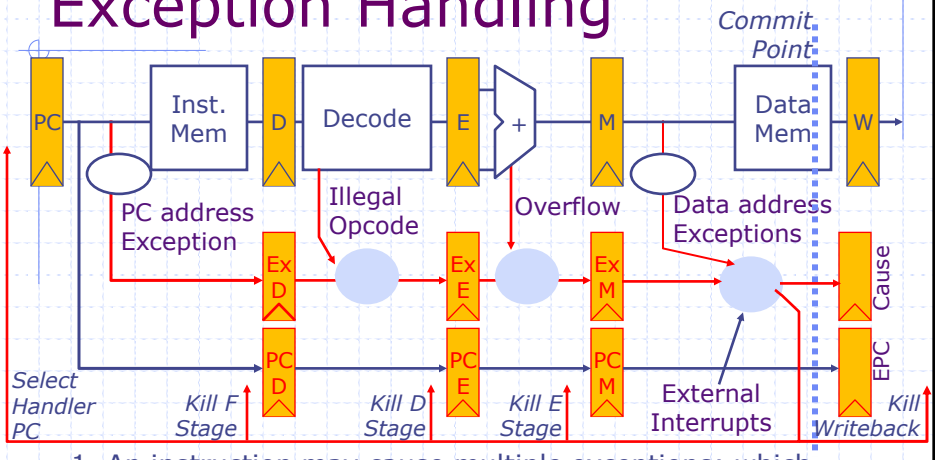


After the PPN is known, W physical tags are compared

Allows cache *size* to be greater than 2^{L+b} bytes

Exception handling in a pipeline machine

Exception Handling



1. An instruction may cause multiple exceptions; which one should we process? from the earliest stage
2. When multiple instructions are causing exceptions; which one should we process first? from the oldest instruction

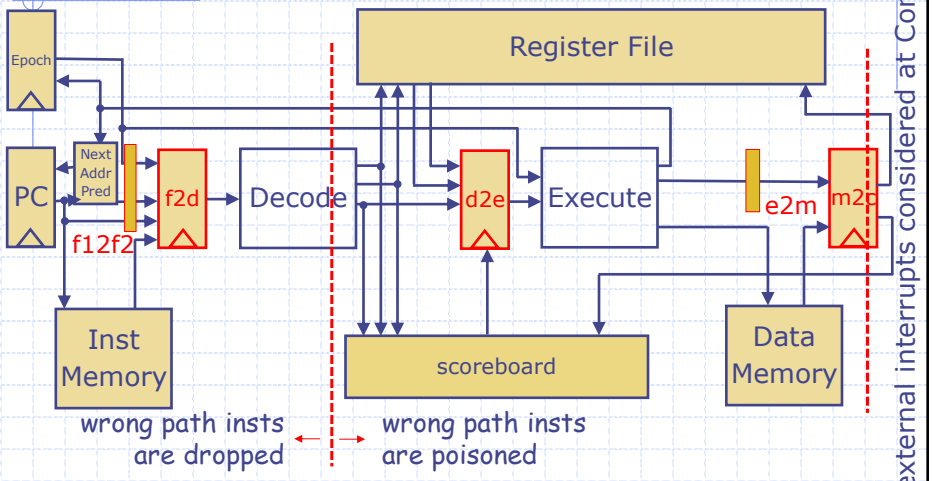
Interrupt processing

- Internal interrupts can happen at any stage but cause a redirection only at Commit
- External interrupts are considered only at Commit
- If an instruction causes an interrupt then the external interrupt, if present, is given a priority and the instruction is executed again
  - Some instructions, like Store, cannot be undone once launched. So an instruction is considered to have completed before an external interrupt is taken

Exception Handling

- When instruction *x* in $stage_i$ raises an exception, its cause is recorded and passed down the pipeline
- For a given instruction, exceptions from the later stages of the pipeline do not override cause of exception from the earlier stages
- If an exception is present at commit: Cause and EPC registers are set, and pc is redirected to the handler PC
  - Epoch mechanism takes care of redirecting the pc

Killing vs Poisoning



This affects whether an instruction is removed from sb in case of an interrupt

Interrupt processing at Execute

Incoming Interrupt

no

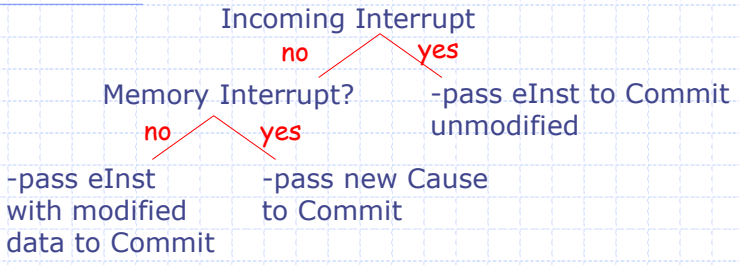
- if (mem type) issue Ld/St
- if (mispred) redirect
- pass eInst to M stage

yes

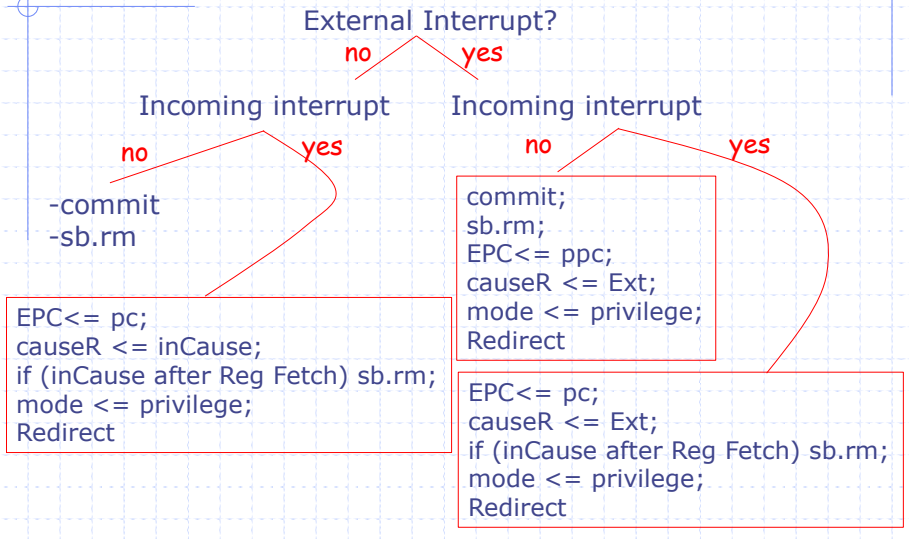
- pass eInst to M stage unmodified

eInst will contain information about any newly detected interrupts at Execute

Interrupt processing at Memory stage



Interrupt processing at Commit



Final comment

- There is generally a lot of machinery associated with a plethora of exceptions in ISAs
- Precise exceptions are difficult to implement correctly in pipelined machines
- Performance is usually not the issue and therefore sometimes exceptions are implemented using microcode

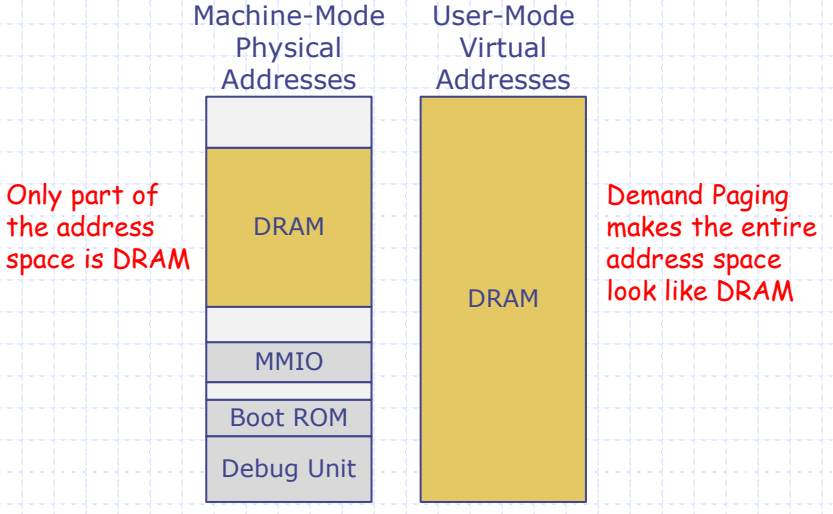
RISC-V Virtual Memory

Privileged ISA v. 1.9.1

RISC-V Privilege Levels

- Separation between low-level access to the hardware and high-level user programs
  - Machine-mode (M) – all addresses are physical addresses, has access to all addresses including memory-mapped IO devices
  - Supervisor-mode (S) – addresses are typically virtual addresses, can switch page-table in use (`sptbr`)
  - User-mode (U) – addresses are virtual, access to devices only through systemcalls

RISC-V Memory Maps



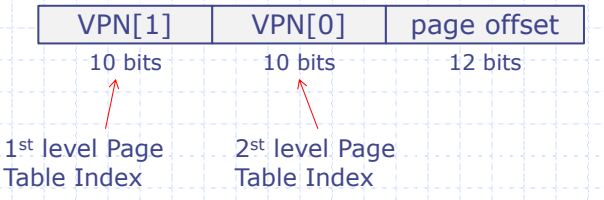
RISC-V Paged Virtual Memory

- Different modes for different systems:
  - Sv32: 32-bit VA, 34-bit PA
    - 4 GB virtual address space
    - 16 GB physical address space
    - 2-layer page table
  - Sv39: 39-bit VA, 50-bit PA
    - 512 GB virtual address space
    - 1 PB physical address space
    - 3-layer page table

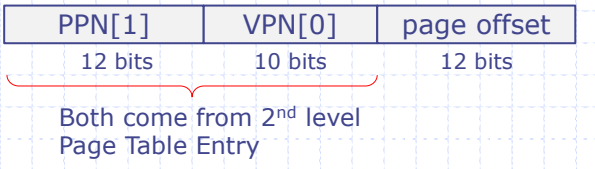
Requires RV64, 64-bit ISA

Sv32 Addresses

- Virtual Addresses:



- Physical Addresses:



Sv32 Page Table Entries

PPN[1]	PPN[0]	*SW Reserved*	D	A	G	U	X	W	R	V
12 bits	10 bits	2 bits								

Dirty – This page has been written to
Accessed - This page has been accessed
Global – Mapping exists in *all* virtual address spaces
User – User-mode programs can access this page
e**X**ecute – This page can be executed
Write – This page can be written to
Read – This page can be read from
Valid – This page valid and in memory

Sv32 Page Table Entries

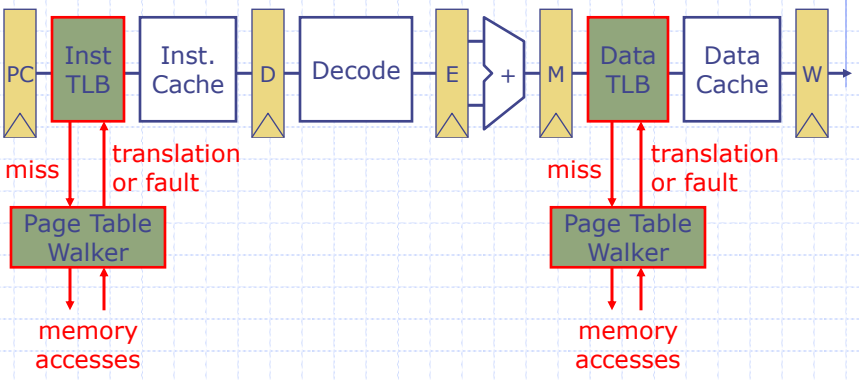
PPN[1]	PPN[0]	*SW Reserved*	D	A	G	U	X	W	R	V
12 bits	10 bits	2 bits								

If V = 1, but X, W, R == 0, PPN[] points to the 2nd level page table

If V = 0, page is either invalid or in disk. If in disk, the OS can reuse bits in the PTE to store the disk address (or part of it).

Disk Address	G	U	X	W	R	0
26 bits						

RISC-V Pipeline with VM



On a fault, an exception is raised and the OS takes over

RISC-V VM Instructions

- SFence.VM
  - Privileged instruction to synchronize TLB translation. Ensures that stores to data cache are seen by hardware page table walker
- CSRs – Control and Status Registers
  - Privileged registers for processor configuration
  - `sptbr` – Page Table Base Register
  - `mstatus.vm` – Virtual Memory mode (e.g. Sv32)
  - `mstatus.mxr`, `mstatus.pum`, `mstatus.mprv` – Fields for modifying privileges for memory accesses to emulate accesses at low privilege levels