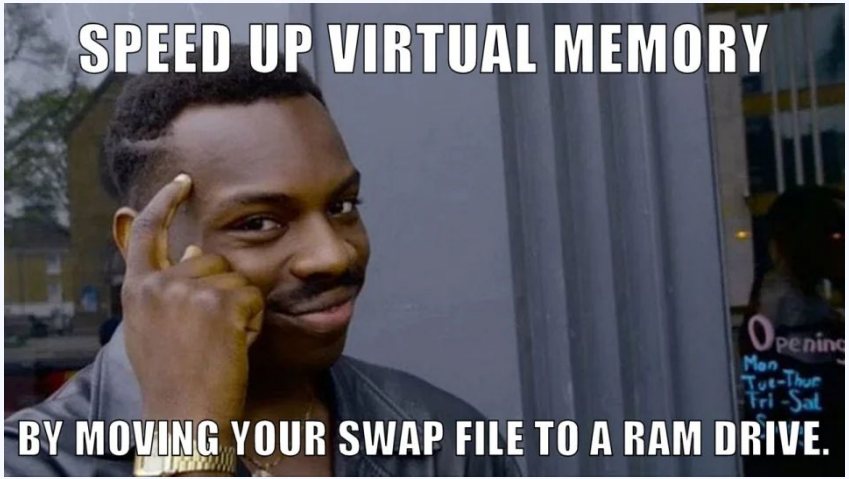
TLBs


SPEED UP VIRTUAL MEMORY
BY MOVING YOUR SWAP FILE TO A RAM DRIVE.

How does the hardware use/manage the page table? How does the OS?

CSRs: control status registers

RISCV has a “privileged architecture” that supports OS operations (access control, paging, special control registers)

CSRs track what the CPU is doing in each mode - helpful when switching between user mode and supervisor mode

4.1.1 Supervisor Status Register (sstatus)

The **sstatus** register is an SXLEN-bit read/write register formatted as shown in Figure 4.1 when SXLEN=32 and Figure 4.2 when SXLEN=64. The **sstatus** register keeps track of the processor's current operating state.

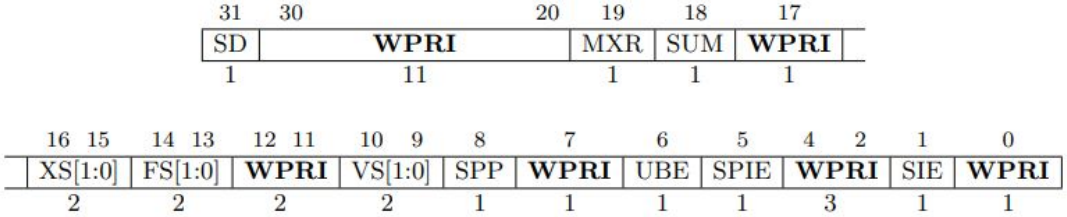


Figure 4.1: Supervisor-mode status register (**sstatus**) when SXLEN=32.

RISC-V spec vol. 2

Address translation in RISC-V

4.1.11 Supervisor Address Translation and Protection (satp) Register

The satp register is an SXLEN-bit read/write register, formatted as shown in Figure 4.14 for SXLEN=32 and Figure 4.15 for SXLEN=64, which controls supervisor-mode address translation and protection. This register holds the physical page number (PPN) of the root page table, i.e., its supervisor physical address divided by 4 KiB; an address space identifier (ASID), which facilitates address-translation fences on a per-address-space basis; and the MODE field, which selects the current address-translation scheme. Further details on the access to this register are described in Section 3.1.6.5.

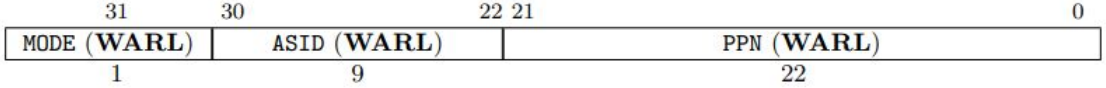


Figure 4.14: Supervisor address translation and protection register satp when SXLEN=32.

SXLEN=32		
Value	Name	Description
0	Bare	No translation or protection.
1	Sv32	Page-based 32-bit virtual addressing (see Section 4.3).

? ? ?

What should happen if our virtual address space is so big that the page table can’t efficiently fit in main memory?

Multi-level page table

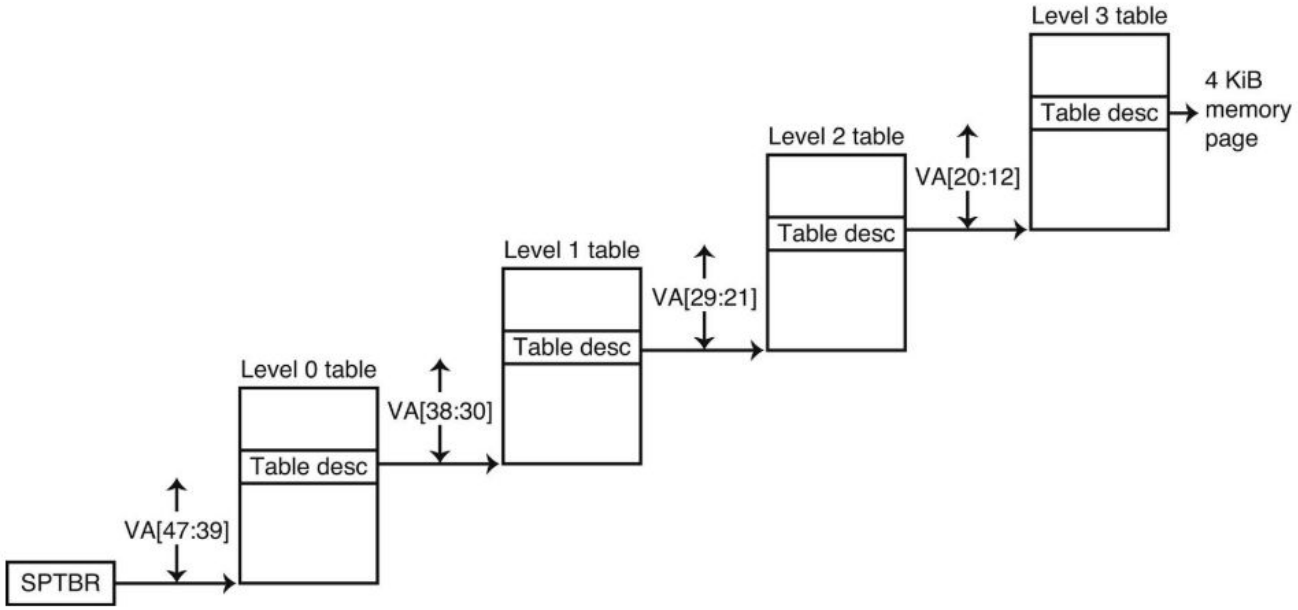


FIGURE 5.29 RISC-V uses four levels of tables to translate a 48-bit virtual address into a 40-bit physical address.

Virtual memory in RISC-V 32bit

32 bit virtual address space (4kb pages) → 20 bit virtual page numbers (VPNs)

22-bit physical page number (PPN)

Page tables are the size of a page

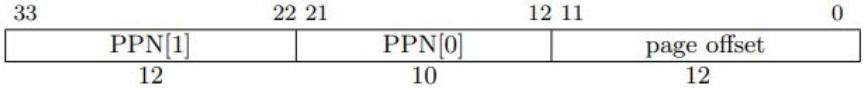


Figure 4.17: Sv32 physical address.

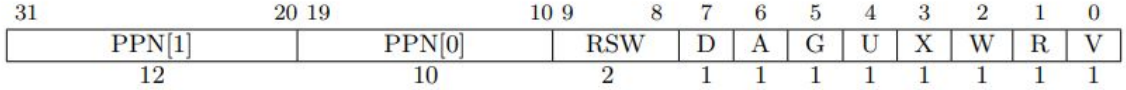
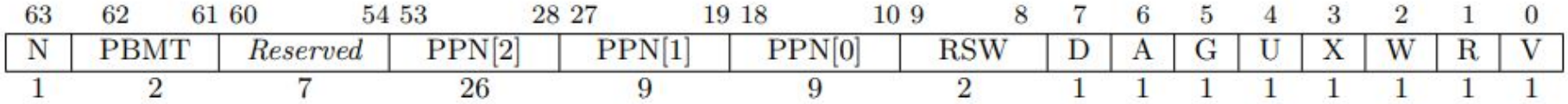


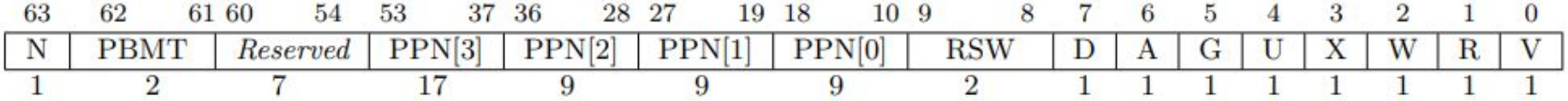
Figure 4.18: Sv32 page table entry.

Virtual memory in RISC-V 64bit



63	62 61	60 54	53 28	27 19	18 10	9 8	7	6	5	4	3	2	1	0
N	PBMT	*Reserved*	PPN[2]	PPN[1]	PPN[0]	RSW	D	A	G	U	X	W	R	V
1	2	7	26	9	9	2	1	1	1	1	1	1	1	1

Figure 4.21: Sv39 page table entry.



63	62 61	60 54	53 37	36 28	27 19	18 10	9 8	7	6	5	4	3	2	1	0
N	PBMT	*Reserved*	PPN[3]	PPN[2]	PPN[1]	PPN[0]	RSW	D	A	G	U	X	W	R	V
1	2	7	17	9	9	9	2	1	1	1	1	1	1	1	1

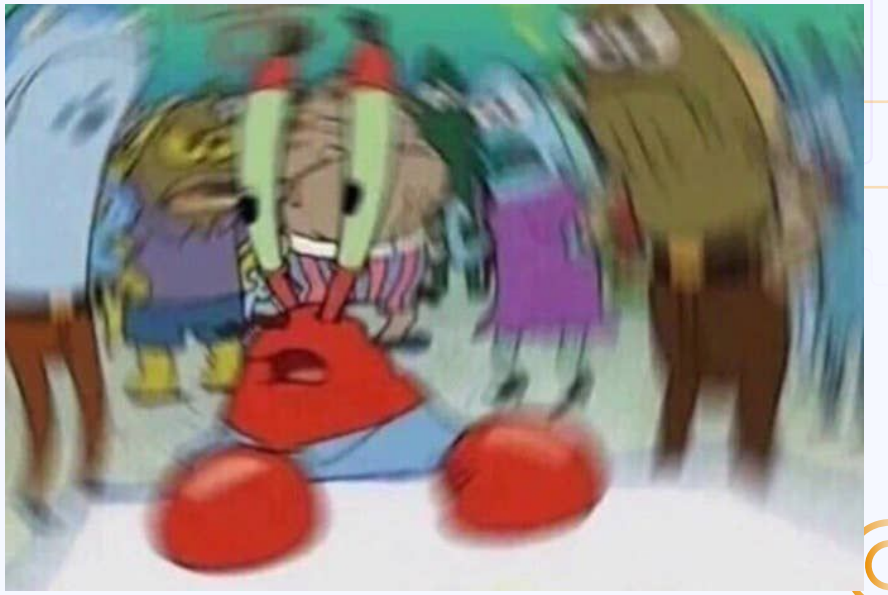
Figure 4.24: Sv48 page table entry.

In earlier lectures, we said memory is really slow, which means that virtual memory makes memory accesses *really really* slow (looking up translation in page table + then doing access). What can be done?

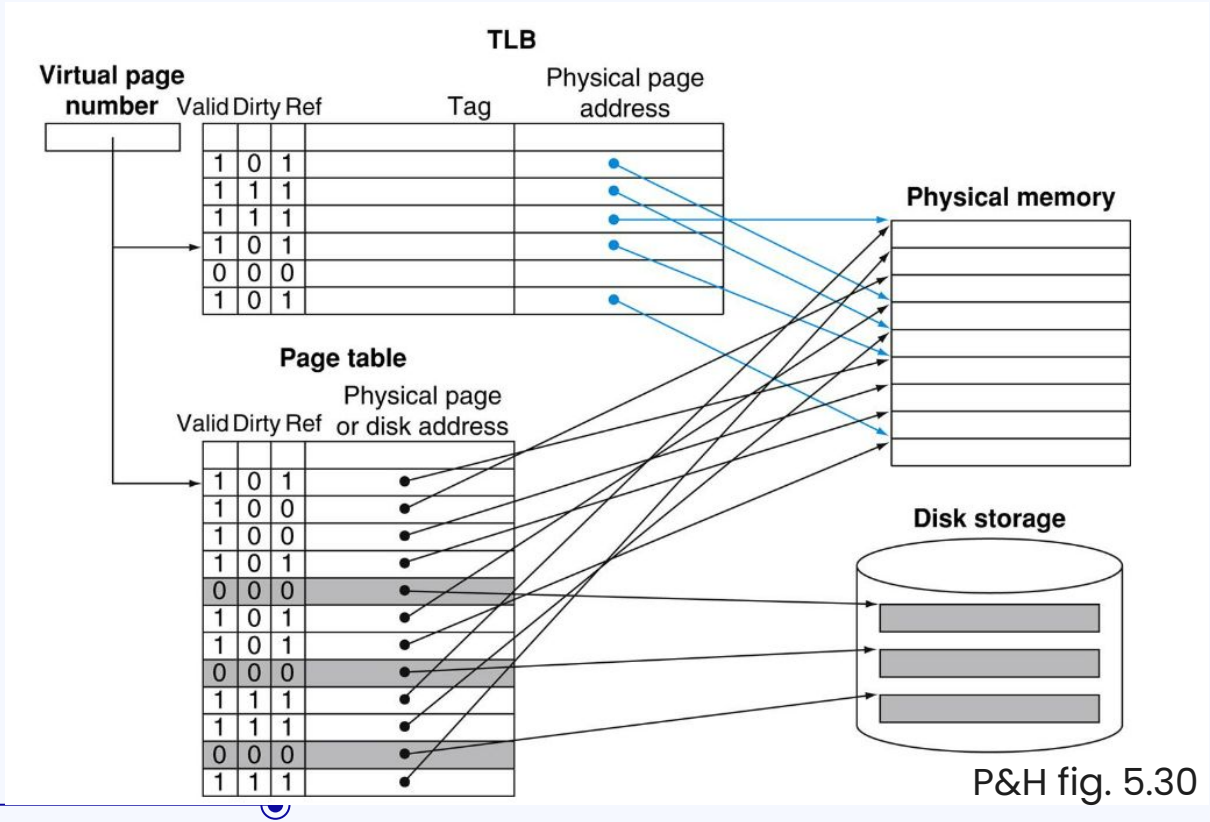
TLBs: a cache for the page table

For those counting: we have

- L1 I-cache
- L1 D-cache
- L2 cache
- L3 cache
- Main memory acting as a cache for disk
- TLBs acting as a cache for page tables (translation of virtual to physical addresses)
- ??? probably other caches in the future



TLBs: does this clear it up?



P&H fig. 5.30

For an instruction like `lw 10 0(sp)`, which do we do first?

- Check L1 cache
- Check main memory
- Check TLB
- Check page table

Interaction of TLB and cache (PIPT)

PIPT (explained on next slide) is physical cache (need to know physical address before indexing into cache)

P&H fig. 5.33

TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.

VIPT caches

PIPT (physically indexed, physically tagged) caches come at page translation cost

VIVT (virtually indexed, virtually tagged) caches cause aliasing issues (two virtual addresses mapping to same physical address)

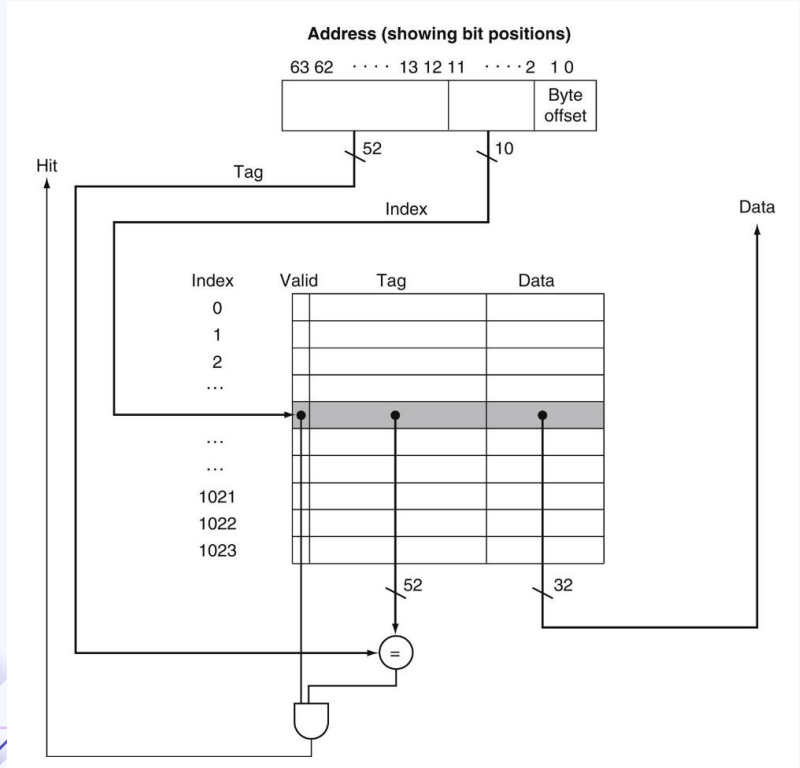
VIPT: perform cache lookup + TLB lookup in parallel

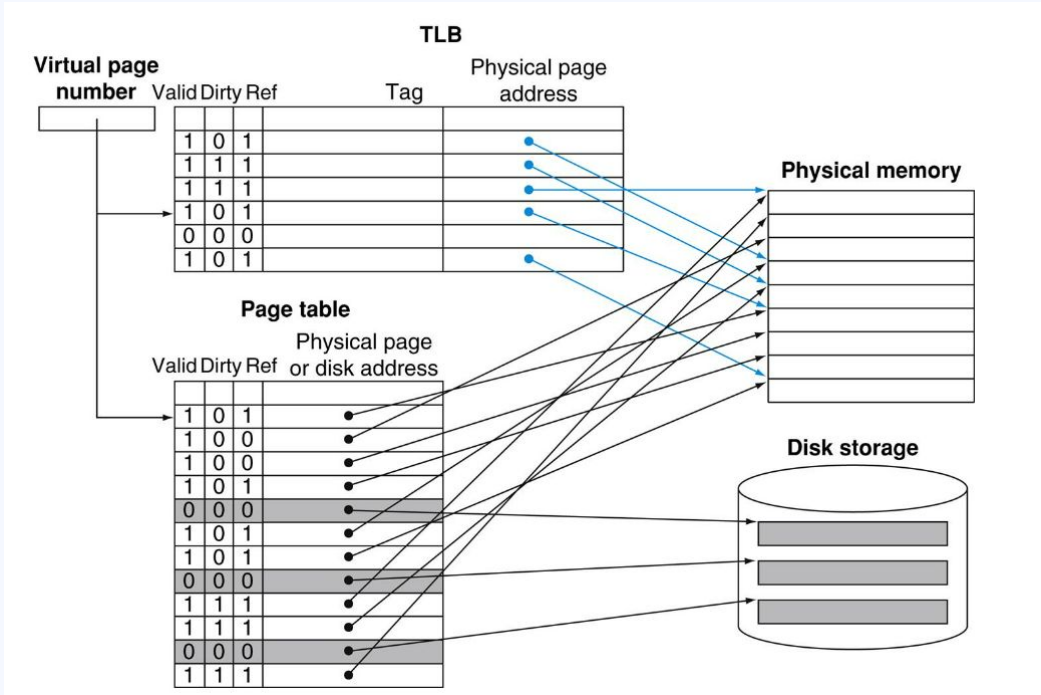
both virtual and physical address have the same page offset

cache size now limited to page size

virtual memory

cache





What options do we have when designing a memory hierarchy?

# of levels in hierarchy

block size

cache size

data, instruction, or unified

associativity

replacement policy

behavior on write (through/back; allocate; buffers)

how to find a block

Design tradeoffs

We designed single-stage CPU for correctness

We designed pipelined CPUs for performance (w/ some complexity tradeoffs)

With memory hierarchy, we encountered the space of *performance tradeoffs*

- Sometimes the answer is to compromise (multiple cache levels; n-way associativity)
- Sometimes the answer is to innovate (TLBs, write buffers, VIPT)

Throwing hardware at the problem has limits and costs ($$, energy, area)

Real world complicates things a *lot*

Using advanced tools like gem5 helps us navigate tradeoffs (with a giant caveat!)