Virtual-memory extensions proof of concept status

Yinan Xu

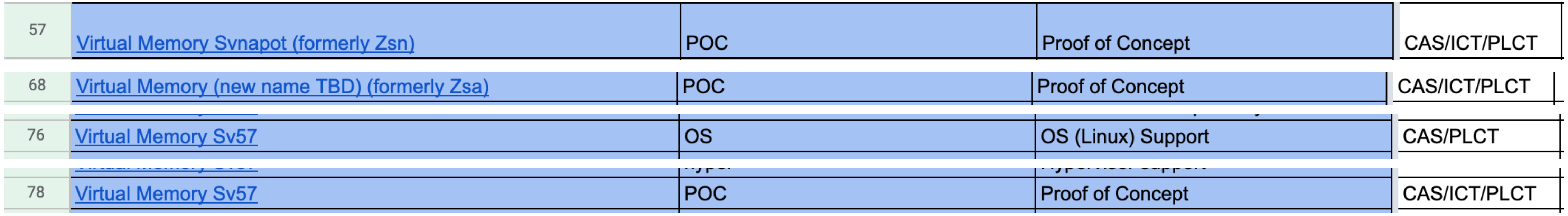
ICT, CAS

2021/5/14

POC for virtual-memory extensions

- Joint group from ICT, CAS and PLCT, ISCAS
  - Leaded by Prof. Yungang Bao and Prof. Yanjun Wu, Prof. Jiageng Yu, Prof. Wei Wu
  - Both ICT and PLCT are development partners (group contributors) with RVI

- The POC tasks we take from RVI



57	Virtual Memory Svnapot (formerly Zsn)	POC	Proof of Concept	CAS/ICT/PLCT
68	Virtual Memory (new name TBD) (formerly Zsa)	POC	Proof of Concept	CAS/ICT/PLCT
76	Virtual Memory Sv57	OS	OS (Linux) Support	CAS/PLCT
78	Virtual Memory Sv57	POC	Proof of Concept	CAS/ICT/PLCT

  - We didn’t know these are not considered to be sub-extensions …

Current implementation status

	Sv57	Svnapot	Zsa
Hardware implementation	Finished (Ziyue Zhang)	Finished (Xin Li)	In-progress (Bohan Hu)
Platform	NutShell	NutShell	rocket-chip
Verification	Hand-written tests	Hand-written tests	Not tested yet
Linux kernel support	In-progress (Hongyu Lin)	In-progress (Qinglin Pan)	(no need?)

* NutShell is an open-sourced five-stage in-order core by ICT (taped-out in 2019 & 2020) https://github.com/OSCPU/NutShell
* Rocket core is an in-order core developed by UCB https://github.com/chipsalliance/rocket-chip

- To share
  - our implementation details: how they work on hardware
  - our thoughts on some problems for hardware implementation
  - POC status for these three extensions (updates for priv-spec-1.12)

- To discuss
  - the SOWs for the tasks (needed by both RVI and TG)
  - future work on the POC (verification, physical design, silicon, …)
  - further participation (spec, hw/sw implementation, …)

- we will keep on contributing to the open-source community

- Thanks!

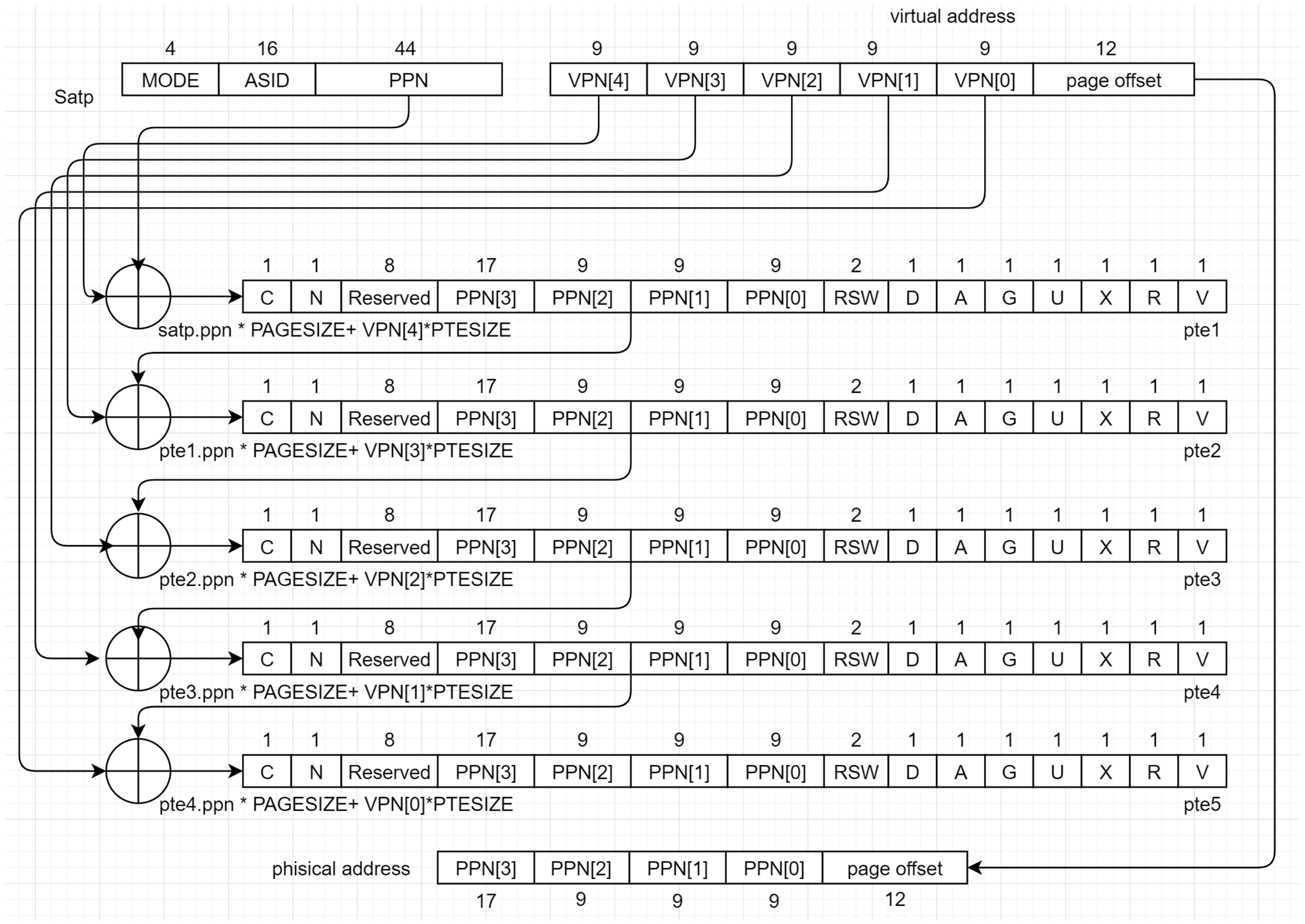
Sv57

Ziyue Zhang ICT, CAS

Current work

- Support sv57 on Nutshell.
- https://github.com/Ziyue-Zhang/nutshell

- Support sv57 on NEMU(riscv emulator).
- https://github.com/Ziyue-Zhang/nemu-rv64


virtual address
4
16
44
MODE
ASID
PPN
Satp
9
9
9
9
9
12
VPN[4]
VPN[3]
VPN[2]
VPN[1]
VPN[0]
page offset
1 1 8 17 9 9 9 2 1 1 1 1 1 1 1
C N Reserved PPN[3] PPN[2] PPN[1] PPN[0] RSW D A G U X R V
satp.ppn * PAGESIZE+ VPN[4]*PTESIZE
pte1
pte1.ppn * PAGESIZE+ VPN[3]*PTESIZE
pte2
pte2.ppn * PAGESIZE+ VPN[2]*PTESIZE
pte3
pte3.ppn * PAGESIZE+ VPN[1]*PTESIZE
pte4
pte4.ppn * PAGESIZE+ VPN[0]*PTESIZE
pte5
phisical address
PPN[3]
PPN[2]
PPN[1]
PPN[0]
page offset
17
9
9
9
12

- Only the page table walker part is modified to support 57-bit virtual addresses on Nutshell.

Test

- Through the way of hardware encoding, turn on the translation of virtual address to physical address on Nutshell.
- Add a 5-level page table for Sv57 in the memory.
- Initialize satp register with the address of 5-level page table.
- Run a simple assembly program in Nutshell, use 5-level page table to translate address when accessing memory, and compare the behavior with NEMU to verify the correctness.

Support Sv57 on Linux

PLCT, ISCAS —— Hongyu Lin

May 2021

Catalogue

Background

Latest linux version: 5.12, not support Sv48 and Sv57.

Other ISA: 5 level paging in x86 has been implemented.

Patch Author: Alex Ghiti

Function: implementing Sv48 in Linux, based on version 5.10.
(Patch link： https://patchwork.kernel.org/project/linux-riscv/list/?series=408843).

Current Work

Since could not find a complete repository of his releases, I manually integrated his patch set into the 5.10 kernel.
(Github link： https://github.com/LHY-24/Linux-Sv48)

However, this version has not been compiled yet, so its correctness cannot be verified：
The problem we are currently encountering is ar:usr/initramfs_data.o can not find, and we are checking the source of the problem.

Problem

1. When implementing Sv48, you only need to choose between Sv39 and Sv48 in RV64 when selecting configuration items. However, after adding Sv57, you need to choose between Sv57, Sv48 and Sv39, so the complexity of judgment is increased.

2. When implementing Sv57, how to degenerate from Sv57 to Sv48 and Sv39 needs to be dealt with.

3. At the same time, we may need to deal with the superpage in Sv57.

Schedule

① Solve the compilation problem of sv48 kernel. (May 19)

② Test and verify Sv48 kernel in nemu and Nutshell. (May 27)

③ If I can ensure the correctness of the implementation of Sv48, I will explore the implementation of Sv57 on its basis.

I will also contact Alex and ask for help in the community if I need it.

Thanks for listening!

Svnapot

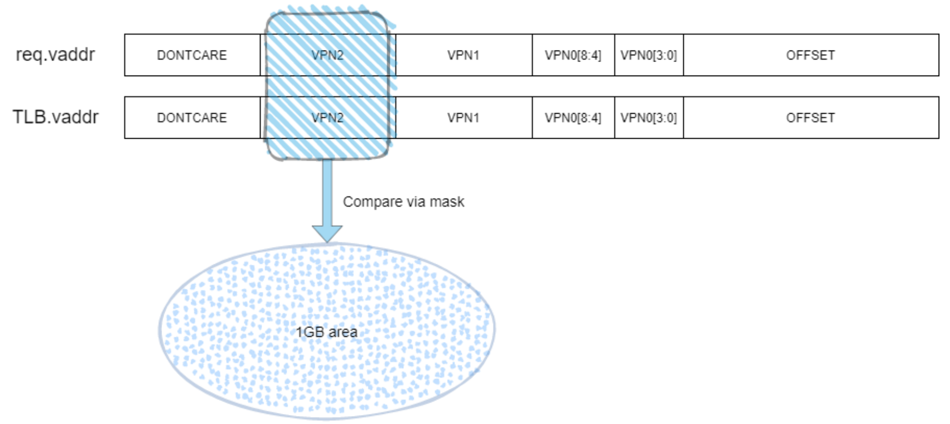
Lixin ICT,CAS

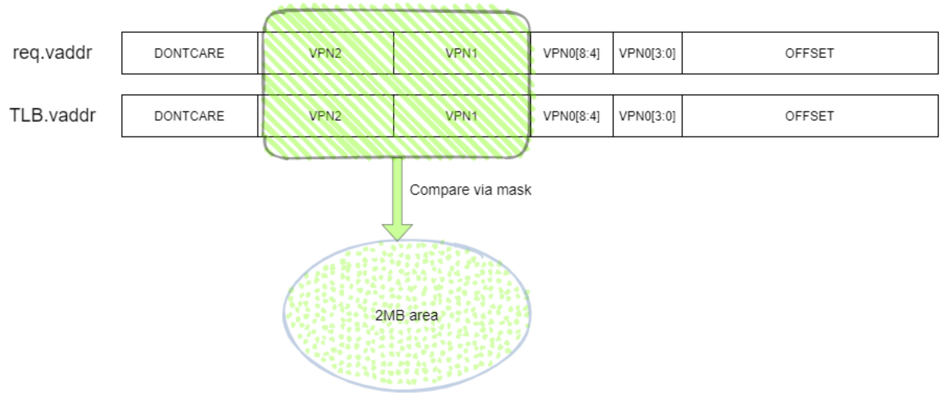
Current work(Svnapot extension)：

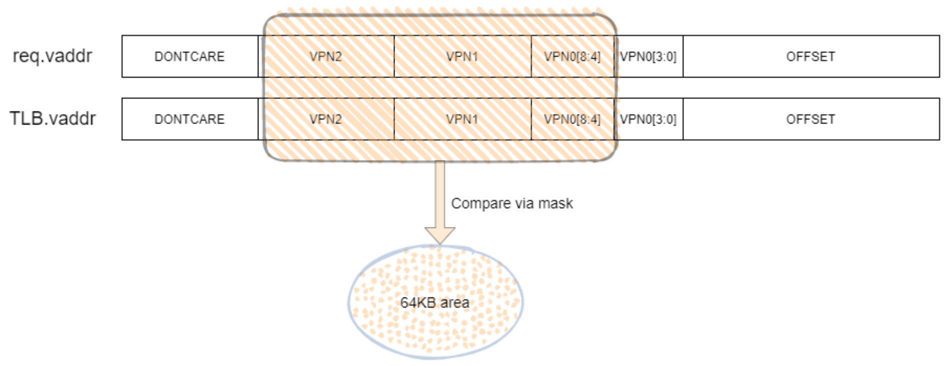
- Support Svnapot extension on Nutshell and add a configuration item to make it easier to open or close it.(https://github.com/happy-lx/NutShell)
- Support Svnapot extension on NEMU simulator.(https://github.com/happy-lx/nemu)
- Pass simple test cases like Microbench , CoreMark , Dhrystone when opening Svnapot extension. A few performance enhancement has been observed.

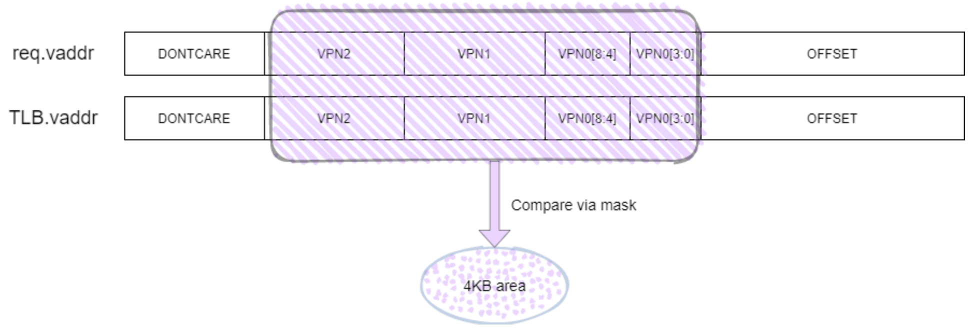
How we implement Svnapot on Nutshell?

- TLB of Nutshell is a fully-associated structure which uses three different masks to handle three situations of Sv39(1GB,2MB,4KB).
- I simply add one mask to indicate the 64KB page of napot extension.









Conclusion

- From the result of implementing and testing , it is convenient to realize the extension on a fully-associated TLB.
- The situation will be more complexed when it comes to a set-associative TLB.

Question about set-associated TLBs

- Many processors have L2 TLB which might be a set-associated structure.
- What if two addresses in the same 64KB area can not find the same entry in a set-associated TLB? (find different TLB sets due to different indexes)

Solutions after discussing

- It is a micro architecture problem . The solution varies with the microstructure.
- Use separate TLBs to store each size of PTEs(1GB,2MB,64KB,4KB). But if the software do not use napot extension then 64KB TLB storage is wasted.
- Use one set-associated TLB to store all size of PTEs . Access TLB with different hash functions each cycle .
- Store 64KB and 4KB PTEs together in a TLB. Use a clever hash function so that it is compatible with two strategies

To Implement Svnapot in Linux Kernel

Progress and Challenge

Qinglin Pan

PLCT, ISCAS

May 2021

Table of Contents

Table of Contents

State of development

- ■ mmap support for anonymous pages without fork
- ▲ fork support for anonymous pages
- □ swap support
- □ file cache support on basic of above

Implementation

- mmap support for anonymous pages without fork

Implementation

- mmap support for anonymous pages without fork
    - modify mmap syscall for napot flags. (simply mark vma)
    - check a napot vma in pgfault

Implementation

- mmap support for anonymous pages without fork
    - modify mmap syscall for napot flags. (simply mark vma)
    - check a napot vma in pgfault
- fork support for anonymous pages
    - implement copy operation for napot 64K page (when fix pgfault)

Implementation

- mmap support for anonymous pages without fork
  - modify mmap syscall for napot flags. (simply mark vma)
  - check a napot vma in pgfault
- fork support for anonymous pages
  - implement copy operation for napot 64K page (when fix pgfault)
- swap support
  - mark related PTEs by kernel reverse map machanism when swap
- file cache support on basic of above
  - still unknown

Table of Contents

Simple Test

```c
int main() {
  void *addr;
  addr = mmap(NULL, 64 * 1024, PROT_WRITE | PROT_READ,
      MAP_PRIVATE | MAP_ANONYMOUS | MAP_HUGE_64KB, -1, 0);
  long *ptr = (long *)addr;
  unsigned int i = 0;
  for(; i < 8 * 1024;i += 512) {
    printf("%lp \n", ptr);
    *ptr = 0xdeafabcd12345678 + i;
    ptr += 512;
  }
  ptr = (long *)addr;
  i = 0;
  for(; i < 8 * 1024;i += 512) {
    if (*ptr != (0xdeafabcd12345678 + i)) {
      printf("failed! 0x%lx \n", *ptr);
      break;
    }
    ptr += 512;
  }
}
```

Listing 1: An mmap example

Table of Contents

PTEs' batch edit

- (for 64K napot page) all 16 PTEs must be modified atomicly
- and a lock must be caught before modify a PTE in Linux Kernel
- for 64K napot support, 16 locks should be caught theoretically
  - for now, we lock the page which pmd points to, before modify PTEs
  - a initial solution for per-PTE-lock : when fixing a napot pgfault, each thread try to lock the lowest-address-PTE first.

Swap granularity

- For now (64K napot granularity), swap can be naturally low cost
- But when it is larger, shall the napot page batch be swaped out?
    - may cost much time if do so

The Implementation of Some New Features about RISC-V Virtual Memory

Bohan Hu
ICT, CAS

Current Work

- Implemented hardware management of A/D bits on Rocket core
- Hardware modules modified:
  - Page table walker
  - TLB
- Boot Linux (Got stuck. Debug in progress)
- No software support is needed

Hardware Modifications

- Extra states for Page Table Walker’s FSM
- Coherency between TLB and memory
- New LR/SC-like reservation stations (optional)

Hardware Modifications

- Compare-and-Swap or LR/SC
  - Using LR/SC-style to monitor any modifications to the in-memory PTE between read and update (Suggested by Dan Lustig)
  - Re-using the existing LR/SC datapath can be accepted
  - Better to add a new LR/SC-like reservation station dedicated for page table walk
  - Tilelink doesn’t offer compare-and-swap operation
  - LR/SC-style is easier to implement

Hardware Modifications

- Extra states for Page Table Walker’s FSM
    - a “Bitset” state is added
    - a LR-like operation is used at each PTW memory access instead of “Load”
    - After retrieving the leaf PTE, instead of returning it to TLB immediately, send a “SC” to the corresponding address with the modified PTE.
    - Retry if LR/SC fails (not match)

Coherency

- Software method: use SFENCE.VMA to sync
- Updating A/D bits does not automatically trigger SFENCE.VMA
- Do changes in local TLBs need to be visible to other harts?
  - Changes can only be the set of A/D bits
  - Hardware doesn’t clear A/D bits and make other changes to PTEs
  - Setting a bit repeatedly doesn’t matters

Coherency

- Consider the situation:
    - The operation is a "store"
    - TLB hits
    - the "D" bit is not set yet
- How to keep the in-memory PTEs and the ones in TLB coherent?

Coherency

- “Hit Invalidate” TLB
  - When encountering such situation, the “hit” TLB entry should be invalidated
  - Then, the PTW should “walk” the page table according to the original memory access (store)
  - The TLB will then be refilled with the new PTE
- Pros:
  - Minimized modification to existing hardware (Similar to SFENCE.VMA)
- Cons:
  - need an extra page table walk when such situation happens

Coherency

- “Write Through” or “Write Back” TLB
- Pros:
  - No more page table walk is needed
- Cons:
  - More complex hardware: Write updated PTEs to D$ requires the TLB keep track of the address of PTEs inside it

Coherency

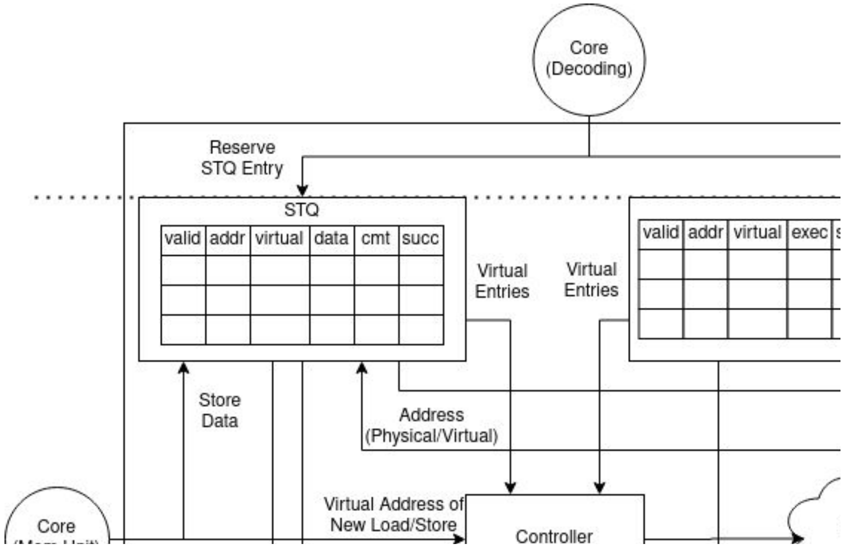
- Actually, the problem can be ignored
  - An extra page fault will be triggered
  - Will cause no functional problems

Implementation in O3 Processors: Issues (Cont’d)

- There are many in-flight “store”s
- If VA -> PA translation is executed when the “store” is in the pipeline:
  - The memory access is speculative: We can’t set “D” bit at that time!
- Any solutions?
  - A new hardware to keep track of every in-flight “store” and its PTE address (or adding new fields in store buffer)
  - Update the “D” bits atomically when the “store” retires
  - Or ...

Implementation in O3 Processors: Issues

- It’s not suggested to do VA-PA translation when executing “store” in O3 pipelines
- BOOM keeps VA instead of PA in STQ, which means address translation will be done upon a “store” retires
- Easier to implement Zsa



Thank you!