Commodity CPU support for virtualisation

COSC349—Cloud Computing Architecture

David Eyers

Learning objectives

- Can describe some challenges that **old x86 CPU architectures** caused for virtualisation
- Understand the relationship between **CPU protected mode** (including long mode) and **CPU guest mode**
- Appreciate that **virtual addresses** map to **physical addresses** in memory blocks called pages
- Explain potential performance problems caused running VMs if CPU does not support **nested paging**

Cloud computing relies on mass production

- Cloud data centres' computers are generic
  - Contrasts with expensive servers in the past
  - Distributed coordination is now more important than individual server functions, so the servers can and should be cheap

- Intel, AMD, helped facilitate x86/x64 virtualisation
  - As noted earlier, x86 virtualisation presented numerous barriers
  - CPU support facilitated further spread of virtualisation:
    - *e.g.*, easier support for virtualisation of Microsoft Windows

Lots of challenges for x86/x64 virtualisation

- Dynamic recompilation should always be able to work
  - … but might be slow, and timing might actually be important
    - (Important either to users, or regarding interactions with devices)
- Challenges include:
  - CPU **protected mode** and CPU **long mode** (64-bit operation)
    - These protected modes weren't designed to be virtualised
  - **Hidden CPU state**
    - VMM can't save/restore this state later when switching VMs
  - **Memory management inefficiency**
  - **I/O interactions**—again, designed without virtualisation in mind

Hardware virtualisation support x86/x64

- Twenty year gap between Intel's first effort, and the hardware virtualisation support we care about
- Intel 8086—the IBM PC CPU—had no protected mode
  - Failures in one application could take down the whole OS!
- Intel 80286 booted **real mode**; added **protected mode**
  - transition from real mode to protected mode was one-way
  - not widely useful: didn't allow hosting 8086 applications
- **Virtual 8086 mode** introduced in Intel 80386—allowed running virtual 8086 environments—MS-DOS on Win 3.1!

Intel x64 (x86-64) versus x86 behaviour

- Protected mode facilitates memory protection
  - Allows OSs to set up CPU to isolate kernel and userspace
    - isolation in terms of CPU share, RAM access, device access
- On 64-bit processors, protected mode feels like 32-bit
- Need to enable CPU's **long mode** to get 64-bit features
  - In long mode, memory access uses 64-bit addresses
    - (Note though that no current CPUs use 64-bit physical addresses
    - … no computer can practically contain that much RAM yet)
  - Also allows access to full CPU register set

Hardware virtualisation support x86/x64

- Intel x86 protected mode itself did not virtualise well
- **Intel VT-x** released in 2005 within some Pentium 4 CPUs
  - Subsequent CPUs include it (except some Atom processors)
  - AMD equivalent released in 2006
- CPUs gain a **guest mode** within protected mode:
  - For guests, guest mode looks like protected mode
  - For hosts, guest mode is lower privilege than protected mode
- CPU capability flags: vmx for Intel, svm for AMD
- More memory virtualisation support was still to come…

Two key obstacles to virtualisation of x86

- Information about privilege level **leaks to guest**
  - CPU code segment selector `%cs` reveals the current privilege level in its two low-order bits—guest should not see this
- Some privileged instructions **do not generate traps** when run in user mode—e.g., `POPF` instruction
  - `POPF` allows OS kernels to change interrupt handling flag `IF`
  - But if OS kernel is running virtualised, it is **not in protected mode**
  - Intel CPUs used not to generate a trap—VMM can't intercept!
- CPU guest mode fixes these problems

VMCS—Virtual Machine Control Structure

- VMCS gives fine-grained control over abilities of guests
  - Often a VMM wants to exert complete device control
  - Sometimes VMM wants a guest VM to directly access hardware
- Trap to VMM if guests attempt restricted operations
  - CPU explicitly records information useful to the host: *e.g.*,
    - indicates the value to be written to a control register
    - indicates value and I/O port to which data was being written
- Intel Haswell adds VMCS shadowing: nested virtualisation

Another x86 virtualisation challenge: RAM

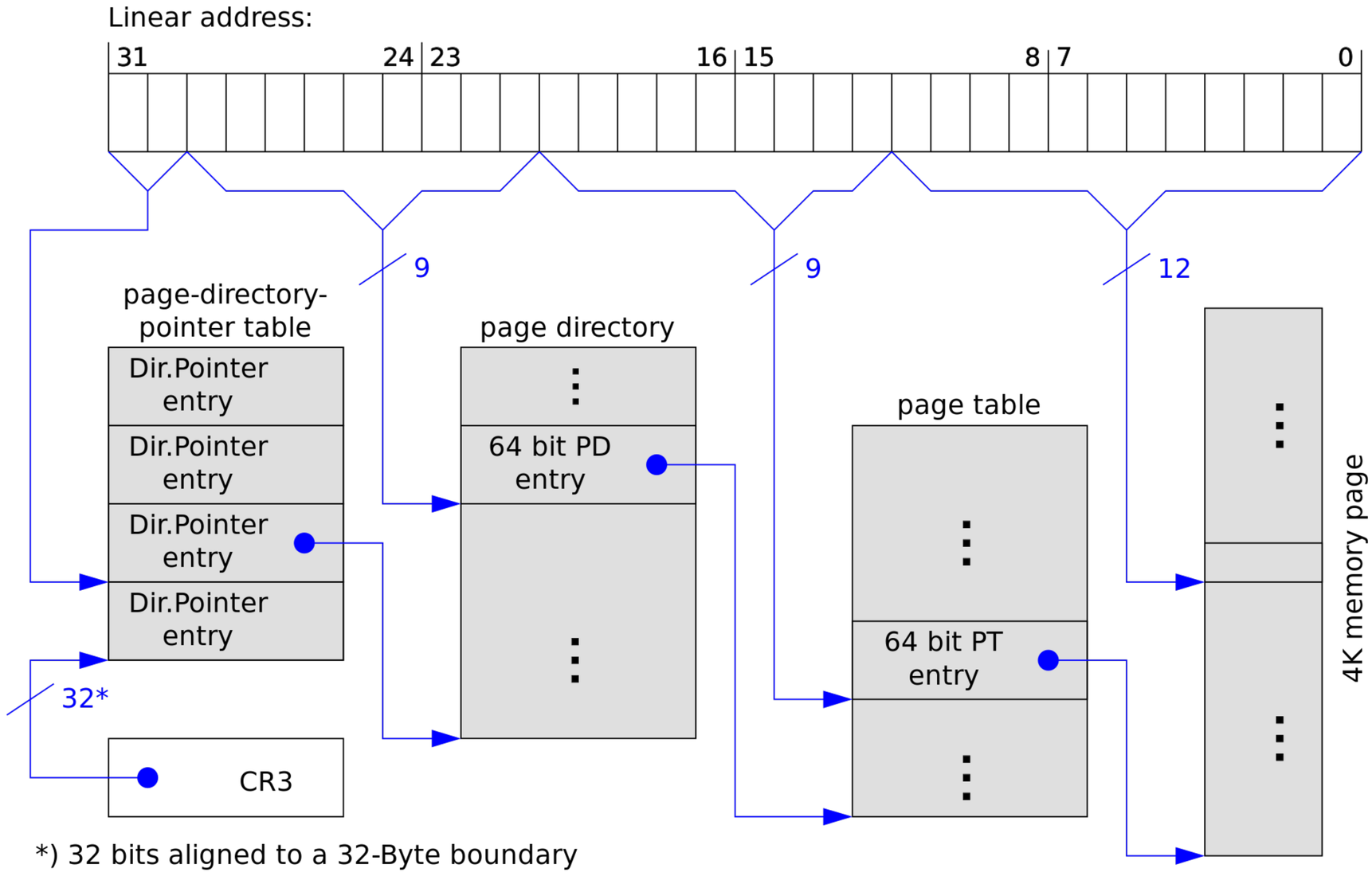
- How CPUs access RAM can be surprisingly complex
- History: CPU's address pins indicate word to read/write
  - e.g., MOS 6502 has 16 address wires, thus 64kB RAM (2^{16} bytes)
  - (even so, can use bank-switching to access more than 64kB)
- Early Intel 80x86 chips addressed offsets of segments
  - Thankfully segmented memory model has died off in x64
  - … so you don't need to think about it
- Intel 80386 added page-based memory mapping…

Page-based memory access

- Modern CPUs manage memory using **pages**
  - CPU memory management unit (MMU) does the work of translating **virtual addresses** into **physical addresses**
- **Page tables** describe virtual to physical mapping
  - …but these page tables are actually stored in memory
  - Page tables define process' address space—may be many!
- Virtual addresses help OSs manage processes' memory
  - Swap parts of an address space **in & out of physical memory**
  - **Memory-mapped files**: process access file using virtual address

FYI “Long mode” paged memory on x64

- Linear address:
  - with 4kB pages
  - using PAE
- 40-bit physical addresses?
  - Gives 1TB RAM
- 48-bit physical addresses now common



Virtualising paged memory—nested paging

- Page tables themselves are managed by guest OSs
- Older CPUs: VMM must store **shadow page tables**
  - Deny guest OSs all memory page access
  - Run software in VMM to distinguish between:
    - Genuine denial of page access
    - Page access should have succeeded, but VM host is intervening
  - VMM software updates shadow page tables and guests' view
- Newer CPUs: SLAT / nested paging support in hardware
  - guest's physical addresses treated as a host virtual address
    - Good caching of virtual to physical address translation important!

Translation lookaside buffer—TLB

- **TLB**s cache virtual to physical memory mappings
  - Specifically, TLB contains recent used entries from page tables
  - Locality of access means TLBs significantly boost performance
- But TLBs don't say which address space an entry is for
  - Thus, when switching OS processes, OS needs to flush the TLB
  - Further, when switching VMs the VMM needs to flush the TLB
- OS manages TLB, thus need to virtualise TLB control
  - TLB in x86 is supposed to be hardware-based:
    - software emulation is very slow

TLB tagging and virtualised DMA

- Since 2008 Intel and AMD have facilitated **TLB tagging**
  - Intel Virtual Processor IDs (VPIDs) allow VMM to assign VM IDs
- Instead of flushing TLB, hardware checks tag matches
  - So switching VM and switching back may leave TLB entries
  - Significant boost to memory access speed
- Finally, I/O support in MMUs can now virtualise DMA
  - PCI Passthrough—safe DMA from device to guest memory