ARM Virtualization: CPU & MMU Issues

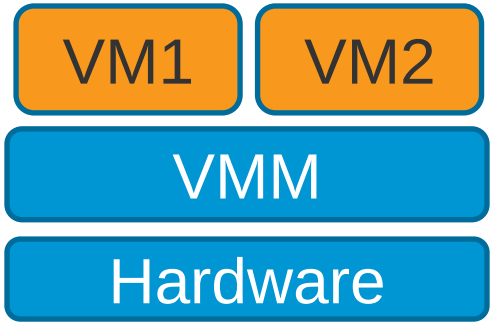
Prashanth Bungale, Sr. Member of Technical Staff

Overview

- **Virtualizability and Sensitive Instructions**
- **ARM CPU State**
- **Sensitive Instructions in ARM**
  - Dealing with Sensitive Instructions
    - In place binary translation
- **MMU Virtualization: Why & How?**
- **Comparison of ARM & x86 Virtualization**

Virtualizability and Sensitive Instructions

- **Defined in the context of a particular virtualization technique**
- **Example: Trap and Emulate Model**
  - Let VM execute most of its instructions directly on the h/w
  - Except for some sensitive instructions that trap into the VMM and are emulated
  - Sensitive instructions are those that interfere with:
    - Correct emulation of the VM, or
    - Correct functioning of the VMM
      - E.g. “Halt Machine” instruction



Virtualizability and Sensitive Instructions

- **Sensitive Instructions as defined by Goldberg [1]**
  - Mode Referencing Instructions
  - Sensitive Register/Memory Access Instructions
  - Storage Protection System Referencing Instructions
  - All I/O Instructions
- **Popek and Goldberg's Theorem about *strict virtualizability* [2]**
  - *For any conventional third generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.*

[1] Goldberg. Architectural Principles for Virtual Computer Systems. Ph.D. thesis, Harvard University, 1972.

[2] Popek and Goldberg, Formal requirements for virtualizable third generation architectures. In SOSP '73

ARM CPU State

System level views

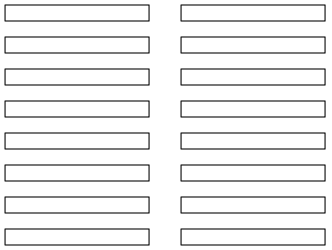
Privileged modes

Exception modes

Application level view	User mode	System mode	Supervisor mode	Monitor mode ‡	Abort mode	Undefined mode	IRQ mode	FIQ mode
R0	R0_usr							
R1	R1_usr							
R2	R2_usr							
R3	R3_usr							
R4	R4_usr							
R5	R5_usr							
R6	R6_usr							
R7	R7_usr							
R8	R8_usr							R8_fiq
R9	R9_usr							R9_fiq
R10	R10_usr							R10_fiq
R11	R11_usr							R11_fiq
R12	R12_usr							R12_fiq
SP	SP_usr		SP_svc	SP_mon ‡	SP_abt	SP_und	SP_irq	SP_fiq
LR	LR_usr		LR_svc	LR_mon ‡	LR_abt	LR_und	LR_irq	LR_fiq
PC	PC							
APSR	CPSR							
			SPSR_svc	SPSR_mon ‡	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

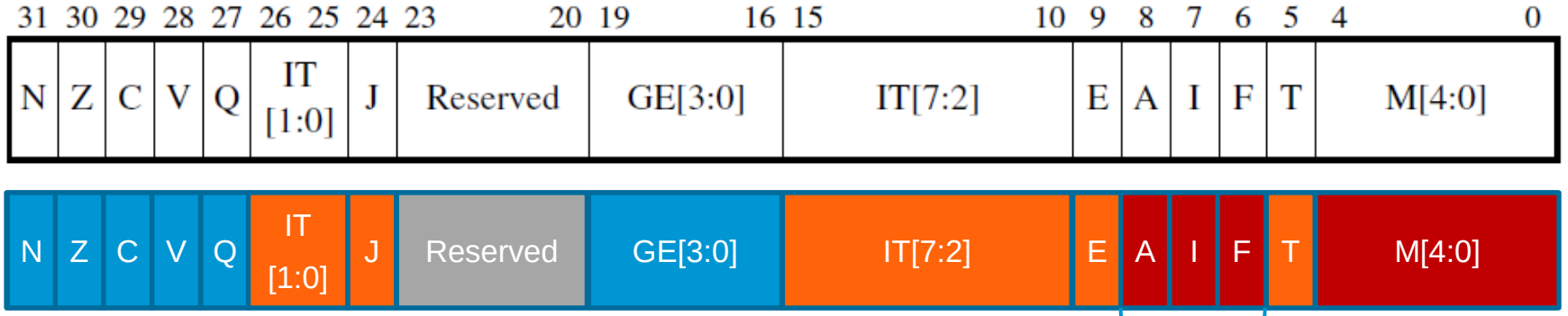
‡ Monitor mode, and the associated banked registers, are implemented only as part of the Security Extensions

Figure B1-1 Organization of general-purpose registers and Program Status Registers

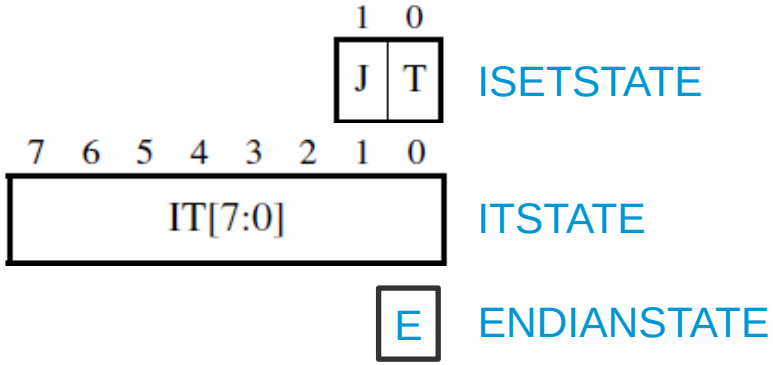


Coprocessor Registers

CPSR: Current Program Status Register



Execution State Registers:



Privileged-only Access Registers

Execution State Registers

Condition Flags

Sensitive Instructions on ARM

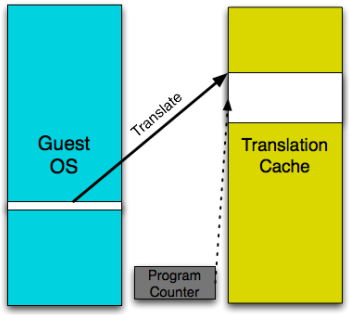
- **Coprocessor Access Instructions**
  - `MRC` / `MCR` / `CDP` / `LDC` / `STC`
- **SIMD/VFP System Register Access Instructions**
  - `VMRS` / `VMSR`
- **TrustZone Secure State Entry Instructions**
  - `SMC`
- **Memory-Mapped I/O Access Instructions**
  - Load/Store instructions from/into memory-mapped I/O locations
- **Direct (Explicit/Implicit) CPSR Access Instructions**
  - `MRS` / `MSR` / `CPS` / `SRS` / `RFE` / `LDM(Exc. Return)` / `DPSPC`
- **Indirect CPSR Access Instructions**
  - `LDRT` / `STRT` – Load/Store Unprivileged (“As User”)
- **Banked Register Access Instructions**
  - `LDM/STM(User mode registers)`

Dealing with Sensitive Instructions

- **Hardware Techniques**
  - Privileged Instruction Semantics dictated by ISA - 1, 2, 3
  - MMU-enforced traps (e.g., page fault) - 4
  - Tracing/debug support (e.g., bkpt)
  - Hardware-assisted Virtualization (e.g., extra privileged mode)

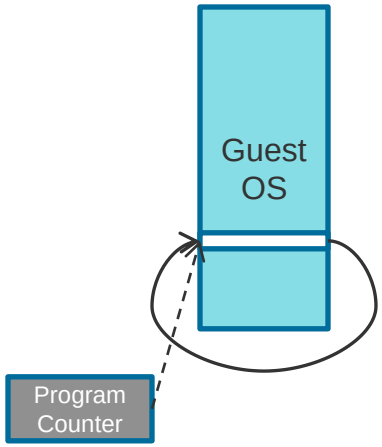
Dealing with Sensitive Instructions

- **Software Techniques**
  - Interpretation / Full Emulation
  - Binary Translation
    - Cached Translation →
    - In-Place Translation
  - Para-Virtualization
    - Shallow Para-Virtualization: replace sensitive instructions
    - Deep Para-Virtualization: replace sensitive subsystems (e.g., process model, page-table management, etc.)
  - Binary Patching / Pre-Virtualization



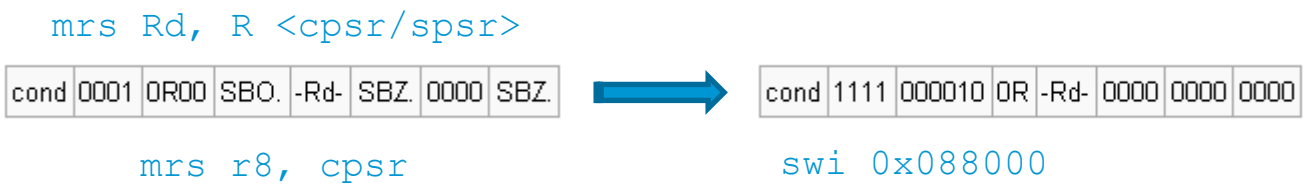
In-place binary translation

- **ARM has a fixed instruction size**
  - 32-bit in ARM mode and 16-bit in Thumb mode
- **Perform binary translation in-place**
  - Instead of in a separate cache
  - Follow control-flow
    - Translate basic block (if not already translated) at the current PC
    - Ensure interposition at end of translated sequence
    - All writes (but not reads) to PC now become dangerous instructions
  - Replace dangerous (i.e., sensitive but unprivileged) instructions 1-1 with hypercalls to force trap and emulate
- **Guest transparency issues**



1-1 Hypercalls

- **Replace dangerous instructions with 1-1 Hypercalls**
  - Use trap instruction to issue hypercall
  - Encode hypercall type & original instruction bits in hypercall *hint*
  - Example:



- **Trap and Emulate Semantics**
  - Upon trapping into the monitor, decode the hypercall type and the original instruction bits, and emulate instruction semantics

MMU Virtualization: Why?

- **Translation Virtualization**
  - VA→PA; PA→MA
- **Protection Virtualization**
  - Access permissions, domains
  - Multiplex available hardware MMU protection support among:
    - Protecting the hypervisor from the guests
    - Protecting the guest kernel from guest user
- **Cross-architectural versions (e.g., ARMv5 on ARMv6)**
  - Dealing with differences in cache, TLB architectures, page-table layout, etc.

MMU Virtualization: How?

- **Shadow PT**
  - Intercept guest MMU events of interest
    - Data/Prefetch Aborts, TTBR deltas, PT deltas, TLB ops
  - Maintain (lazily) hypervisor-controlled, trusted shadow PT
  - Options:
    - TLB Coherency-based Shadow PTs / Cached Shadow PTs
    - In-place Shadow PTs
- **Para-Virtualized trusted guest PT**
  - Highly intrusive to guest MMU software
- **Hardware virtualization support**
  - Nested / 2-stage Page Tables: VA->PA; PA->MA

Comparison of ARM vs. x86 Virtualizability

- **Sensitive Instructions**

Type of Sensitive Instructions	Violating Goldberg's Requirement #	X86 [3]	ARM
Sensitive Register Access	3B	SGDT, SIDT, SLDT, SMSW, PUSHF/POPF	-
Protection System References	3C	LAR, LSL, VERR, VERW, PUSH/POP, CALL, JMP, INT n, RET, STR, MOVE	LDM/STM (user regs), LDRT/STRT ("As User")
Both	3B & 3C	-	MRS, MSR, CPS, SRS, RFE, DPSPC, LDM (exc. return)

[3] John Scott Robin and Cynthia Irvine, Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor, USENIX Security Symposium, 2000.

Comparison of ARM vs. x86 Virtualizability

- **Ring compression – protection mechanisms**
  - `x86`: Segmentation + Paging
  - `ARM`: Paging (+ domains?)
- **Instruction execution vs. Data Read/Write protection**
  - `x86`: `CS` for instruction fetch vs. `DS`/other for data access
  - `ARM`: No explicit distinction b/w execute and read protection
- **Cache architecture**
  - `x86`: Largely transparent; PIPT across all versions
  - `ARM`: Exposes a lot of the cache architecture; VIVT/VIPT/PIPT

Comparison of ARM vs. x86 Virtualizability

- **Instruction size**
  - `x86`: Variable
  - `ARM`: Fixed -> enables more in-place patching mechanisms

- **I/O**
  - `x86`: I/O instructions + memory-mapped I/O
  - `ARM`: Only memory-mapped I/O