Intel SGX explained

Monday, April 10, 2017 2:46 AM

这个文档里主要记录我新学习的内容，方便以后查阅。

2. Intel architecture

2.2 计算模型

内存的指令可以分为write read

CPU之间和memory通过System bus连接, 发出指令send 和从bus上read.

SEND(op, $addr$, $data$) $\rightarrow \varnothing$ Place a message containing the operation code op, the bus address $addr$, and the value $data$ on the bus.
READ() $\rightarrow$ (op, $addr$, $value$) Return the message that was written on the bus at the beginning of this clock cycle.

Figure 7: The system bus abstraction

2.4 Address space

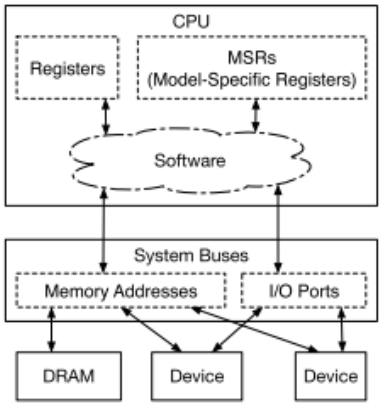

Figure 9: The four physical address spaces used by an Intel CPU. The registers and MSRs are internal to the CPU, while the memory and I/O address spaces are used to communicate with DRAM and other devices via system buses.

如上图所示, Intel中有4种物理地址,分别是寄存器,MSRs,内存和I/O port.

物理内存地址通常有2^{36}到2^{40}的地址空间,通常用来访问内存或MMIO.

MSRs包括2^{32}个MSRs, CPU提供读写MSR地址空间的指令如RDTSC和RDTSCP

2.5 address translations.

4级地址转换页表:

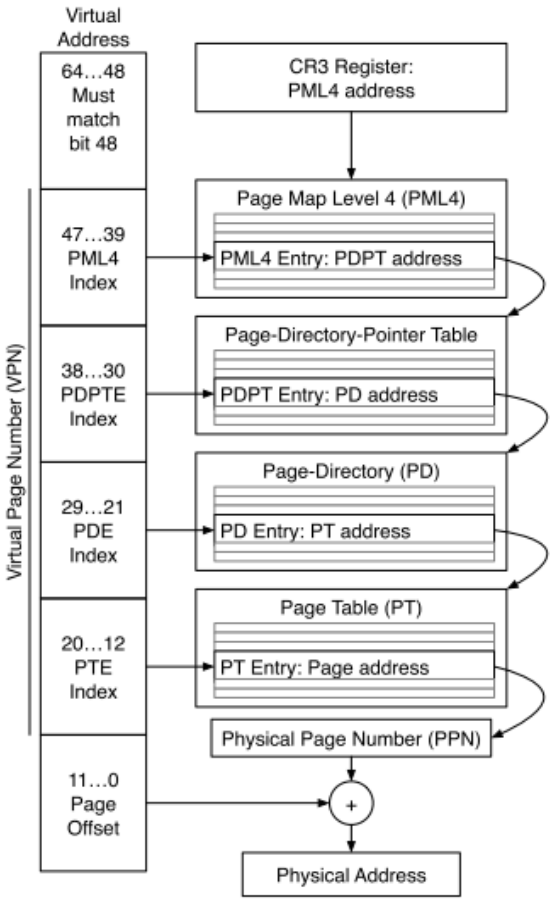

Figure 12: IA-32e address translation takes in a 48-bit virtual address and outputs a 52-bit physical address.

以及虚拟化条件下最高访存次数

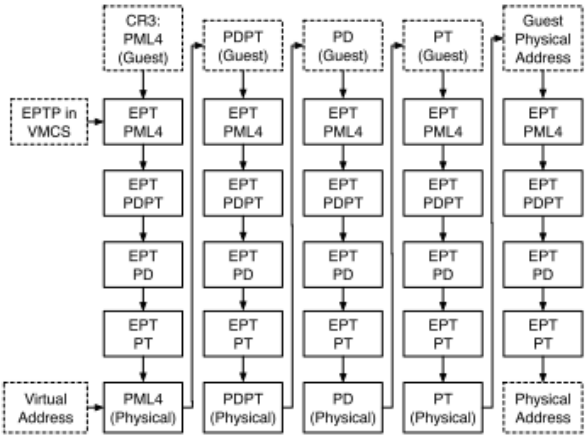

Figure 15: Address translation when hardware virtualization is enabled. The kernel-managed page tables contain guest-physical addresses, so each level in the kernel's page table requires a full walk of the hypervisor's extended page table (EPT). A translation requires up to 20 memory accesses (the bold boxes), assuming the physical address of the kernel's PML4 is cached.

2.6 Execution context

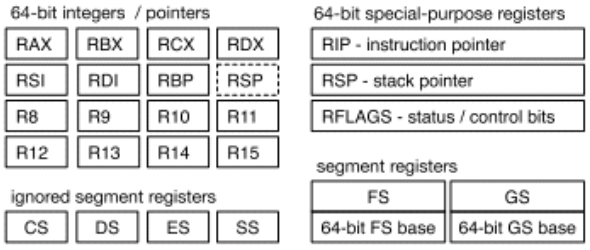

Figure 16: CPU registers in the 64-bit Intel architecture. RSP can be used as a general-purpose register (GPR), e.g., in pointer arithmetic, but it always points to the top of the program's stack. Segment registers are covered in § 2.7.

整数和内存地址被存在16个通用寄存器中.

RSP指向程序调用栈的最高点.它的值和他指向的值会被CPU自动修改,如 Call 和RET指令,或PUSH POP指令.

Intel 平台提供XSAVE指令,根据RFBM的值,将bitmap中为1的寄存器保持在内存中,同时保存的还有RFBM.XRSTOR 将其恢复.

2.7 段寄存器.

现代操作系统中,通常将整个地址空间包括在一个segment中来disable段机制.,一般包括一个code segment(CS) 和一个data segment(DS,SS,ES). FS和GS寄存器通常用来保留线程本地存储(TLS)的段.

2.9 a computer Map

计算机主板的构造图

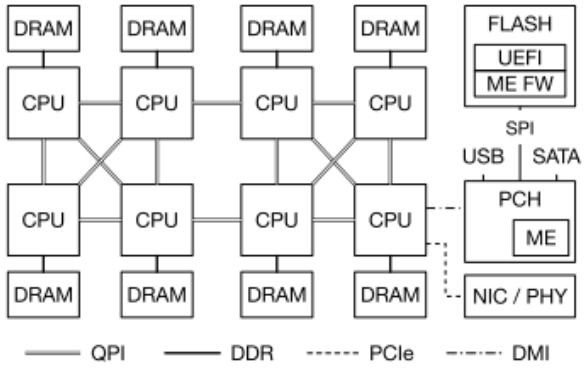

Figure 20: The motherboard structures that are most relevant in a system security analysis.

PCH负责连接低速设备.像SPI,STAT,Usb 等.

同时有一个FLASH设备存储着UEFI的固件,包括启动代码和SMM代码.

Quick-Path Interconnect (QPI)

负责CPU内部的互联

PCIe负责高速设备的互联.

NIC中有一个PHY模块将模拟

信号转换为数字信号.MAC模块实现网络层协议.

Intel ME

ME就是在系统软掉电时依然可以访问的一个小的系统.

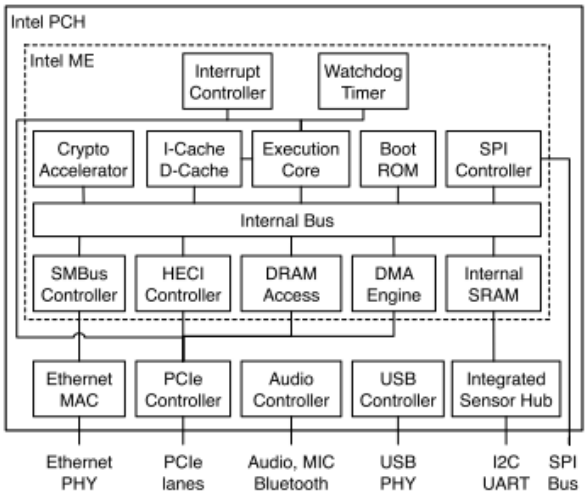

Figure 21: The Intel Management Engine (ME) is an embedded computer hosted in the PCH. The ME has its own execution core, ROM and SRAM. The ME can access the host's DRAM via a memory controller and a DMA controller. The ME is remotely accessible over the network, as it has direct access to an Ethernet PHY via the SMBus.

ME有自己的执行core和启动ROM以及内部的RAM.me的core是一个Argonaut RISC core,频率为200-400MHz.内部的SRAM有640KB,和ISH core共享.没有CPU支持时, SMBus运行在1 MHz 模式同时Ethernet PHY运行在10Mbps模式.

The Processor Die

Intel的芯片可以分为两个部分:

1. Core area实现指令的执行.
2. uncore提供以往独立芯片但如今集成到CPU中的功能.

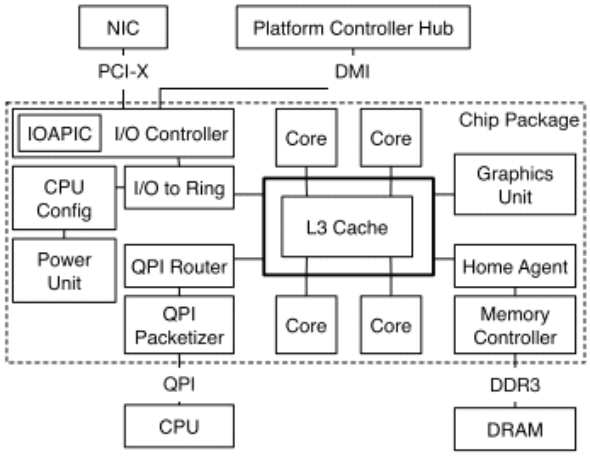

Figure 22: The major components in a modern CPU package. § 2.9.3 gives an uncore overview. § 2.9.4 describes execution cores. § 2.11.3 takes a deeper look at the uncore.

值得注意的是,DDR bus的控制器和PCIe控制器都在CPU内部实现,因此可以做很多安全相关的扩展.

以下是CPU core的图解.

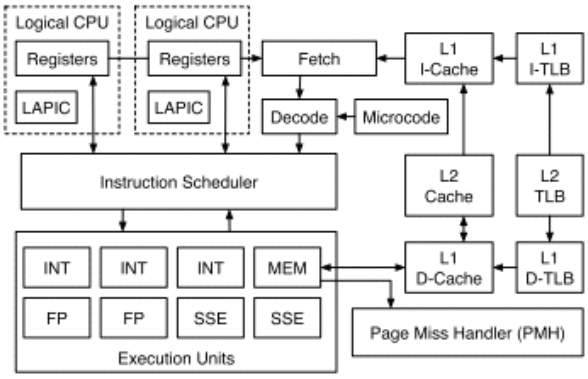

Figure 23: CPU core with two logical processors. Each logical processor has its own execution context and LAPIC (§ 2.12). All the other core resources are shared.

2.11 Cache

当前Cache的访问速度

Memory	Size	Access Time
Core Registers	1 KB	no latency
L1 D-Cache	32 KB	4 cycles
L2 Cache	256 KB	10 cycles
L3 Cache	8 MB	40-75 cycles
DRAM	16 GB	60 ns

Table 8: Approximate sizes and access times for each level in the memory hierarchy of an Intel processor, from [127]. Memory sizes and access times differ by orders of magnitude across the different levels of the hierarchy. This table does not cover multi-processor systems.

Cache 一致性协议的实现:

Cache 一致性协议MESIF实现在CPU内部和QPI bus上.

CPU内部的构造:

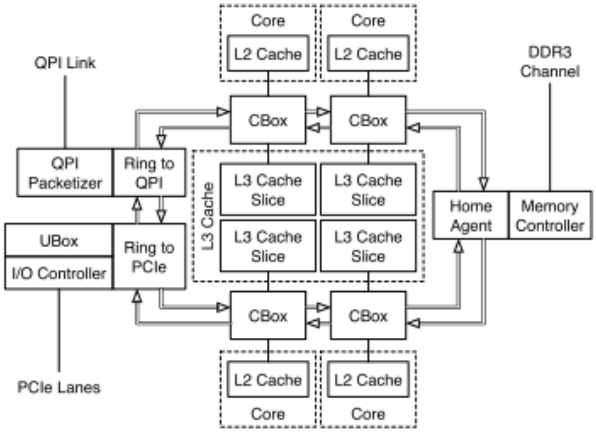

Figure 27: The stops on the ring interconnect used for inter-core and core-uncore communication.

L3又被称为Last level cache(LLC), 在L2 和L1的cache一定在L3中存在.

每个CPU都有cache agents，，连接LLC到processor，和home agent，连接memory controller。Cache miss 时Cache agent 发送请求给home agent，同时home agent对cache line 数据的ownership进行track，或从其他cache agent获取cache数据。QPI支持多个agent，每个core有自己的cache agent，至少有一个home agent。

LLC被分成Core number分LLC slice，同时Cache line 间有着复杂的hash关系。

CBox转发Core的LLC访问。CBox负责Cache一致性协议，即cache agent的功能。
UBox是uncore configuration agent，负责读写物理上的分布在uncore上的寄存器，同时也接收中断并dispatch到相应的core上。
最近的Intel 芯片中，uncore部分还包括一个内存管理器（iMC或Mbox）。这给intel 芯片带来了对DMA操作的过滤功能。
Home agent 中有一个Target Address Decoder（TAD)，将DRAM的地址映射到具体的DRAM芯片上。

Cache和MMIO
Intel将内存地址空间划分为区域，可以对区域的cache属性进行赋值。
Uncacheable，write combining, write through 和Write Protected.
Cache的属性通常由MTRR和PAT管理，同时还受到CR0中Cache disable 和Not-Write through bits影响。

TLB的访问时间

Memory	Entries	Access Time
L1 I-TLB	128 + 8 = 136	1 cycle
L1 D-TLB	64 + 32 + 4 = 100	1 cycle
L2 TLB	1536 + 8 = 1544	7 cycles
Page Tables	$2^{36} \approx 6 \cdot 10^{10}$	18 cycles - 200ms

Table 9: Approximate sizes and access times for each level in the TLB hierarchy, from [4].

TLB没有实现Cache一致性协议。
值得注意的是，L1 Cache中set index并没有用地址转换会影响的位，所以L1 set lookup 可以和TLB 查找同步进行。

Interrupt
外设通常使用总线特定的协议发起终端，比如PCIe通过MSI协议；而后终端信号被PCH中的I/O APIC接收。IOAPIC向一个或多个LAPIC发送终端请求。每个Core都有自己的LAPIC来接收中断。每个IOAPIC转发一个8bit的interrupt vector来指明中断来源，同时32bit的APIC ID用来指明接收的LAPIC。
每个LAPIC用一个256bit的Interrupt Request Register(IRR)来跟踪没有响应的中断。当Core available时，将最高优先级的中断从IRR拷贝到in-service register（ISR)中。
在CPU的处理中，重用fault handling的机制，ISR中的值用来定位中断服务程序。服务程序完成必要的操作后，向LAPIC的end of interrupt寄存器写以通知对方。
同时，系统软件可以通过将target core的APIC ID写入 LAPIC的interrupt command register(ICR)中来做IP I

Boot流程
更多信息：
Booting an Intel Architecture System, Part I: Early Initialization

From <http://www.drdobbs.com/go-parallel/article/print?articleId=232300699>

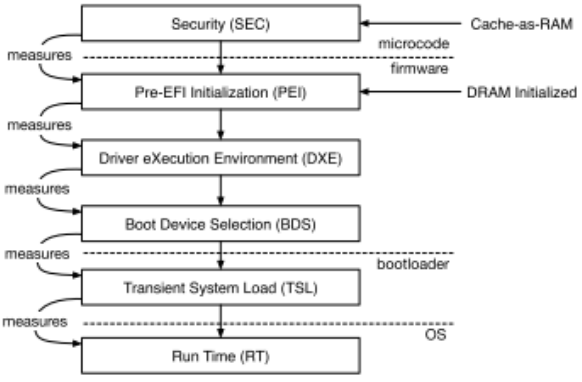

Figure 30: The phases of the Platform Initialization process in the UEFI specification.

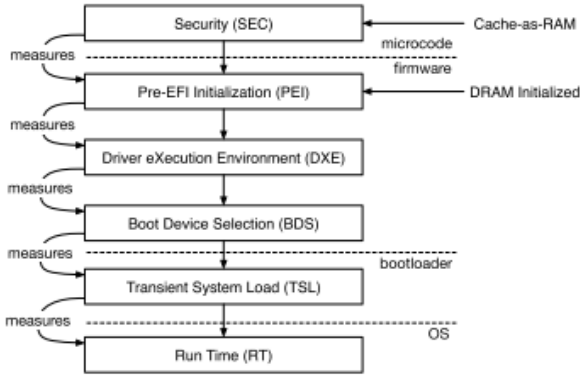

Figure 30: The phases of the Platform Initialization process in the UEFI specification.

上图是UEFI的启动流程

当机器启动的时候，进入security phase（SEC)，这是在CPU微指令实现的内容，这个阶段负责建立一块临时的内存存储然后将下一阶段的固件加载进来。
最为系统最先执行的代码，他代表着系统的可信根，同时也是建立系统安全的第一步。SEC会measure PEI。
接下来PEI阶段初始化系统的DRAM，将自己从临时存储区域拷贝到DRAM，销毁临时存储区域。同时PEI初始化存有UEFI的非易失性存储区域并且将下一阶段的固件加载。
接下来是DXE阶段和BDS阶段。TSL阶段将会加载BDS阶段选择的操作系统，OS loader将控制权限移交给操作系统，进入RT阶段。
如果是从sleep中启动，PEI先初始化存有系统快照的非易失性存储区域，然后启动快速初始化代码。

在系统上电的时候，电路的电力供应根据power sequencing 来启动存储器。PCH 上的Intel ME在CPU cores 启动之前就先power。
ME启动时，开始执行自己的boot 代码，设置好SPI bus，SPI bus 连接的Flash memory中存有ME的固件和UEFI固件。然后ME将自己的操作系统和应用程序加载。
之后，ME设置一些主板上的硬件，如PCH bus clock，然后开始CPU的启动流程。
CPU启动流程的开始是SEC阶段，在cpu电路中实现的。所有的逻辑CPU都需要执行硬件初始化流程，无效所有的cache TLB，执行硬件自检，设置所有的寄存器为初始值。
之后，LP执行Multi-Processor 初始化算法（MP)，一个LP变成启动处理器，其他的变成Application Processor(AP）。
MP算法基本来说就是所有的Core试图去操作QPI bus，谁成功了就是BP。

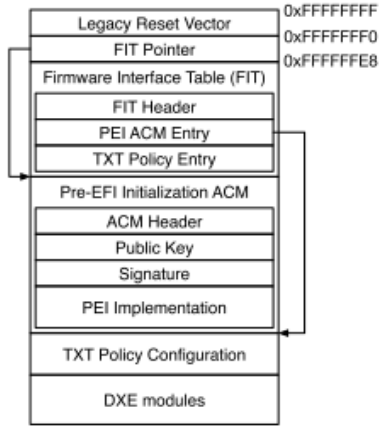

Figure 31: The Firmware Interface Table (FIT) in relation to the firmware's memory map.

现在的Intel 64位处理器均使用FIT来加载。如上图所示

CPU microcode
超过四个micro-ops 就说明这个指令是用Microcode实现的。