CPU vulnerabilities

EuroOpen Žatec

05/2018

Vít Šembera

What is a CPU ?

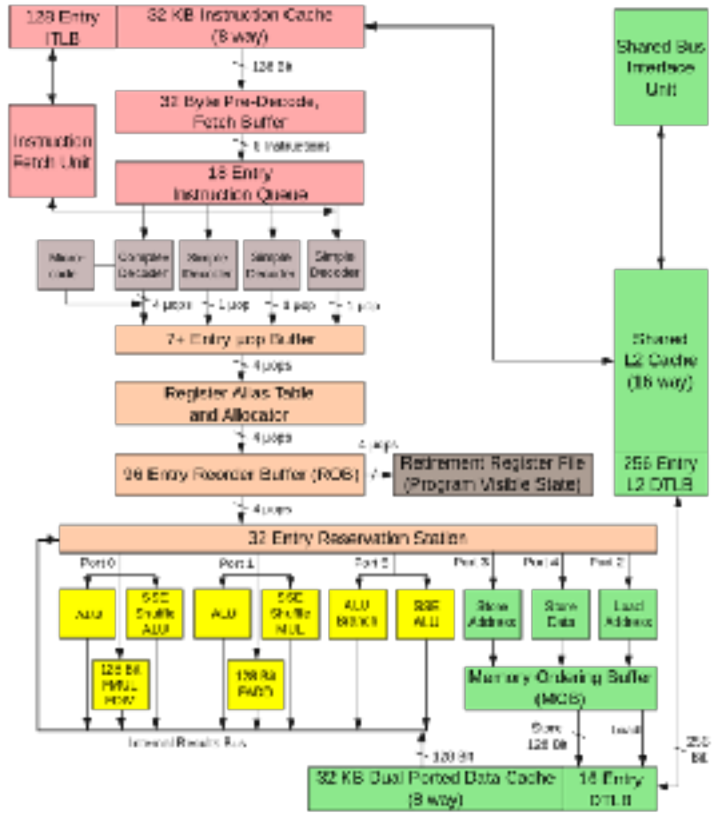
- Wikipedia: A **central processing unit** (**CPU**) is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

It is that fat expensive chip that has a big cooler attached.

CPU executes instructions…

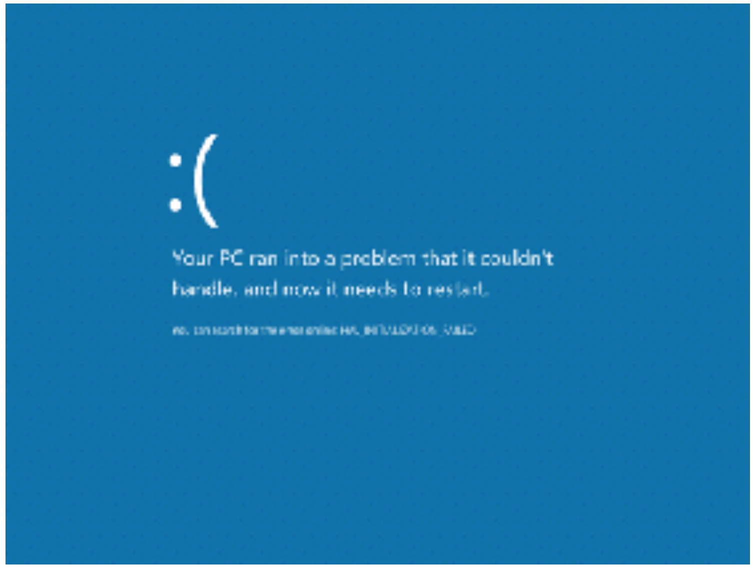
- Instructions format and behavior are strictly defined in vendor specification
- State of a CPU is defined by previous state, instruction and inputs
- Result is always predictable

…really?


128 Entry ITLB
32 KB Instruction Cache (8 way)
Shared Bus Interface Unit
Instruction Fetch Unit
32 Byte Pre-Decode, Fetch Buffer
18 Entry Instruction Queue
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
7+ Entry µop Buffer
Register Alias Table and Allocator
96 Entry Reorder Buffer (ROB)
Retirement Register File (Program Visible State)
Shared L2 Cache (16 way)
256 Entry L2 DTLB
32 Entry Reservation Station
ALU
SSE Shuffle ALU
ALU
SSE Shuffle MUL
ALU Branch
SSE ALU
Store Address
Store Data
Load Address
Memory Ordering Buffer (MOB)
32 KB Dual Ported Data Cache (8 way)
16 Entry DTLB
Intel Core 2 Architecture

Try this on Pentium B1 stepping:

- Binary code F0 0F C7 C8
- Instruction: *lock cmpxchg8b eax*
- Instruction can be executed with any privilege
- The instruction is invalid – operand should be memory reference
- Result should be invalid instruction exception
- Instead CPU halts and must be reset to recover



:(

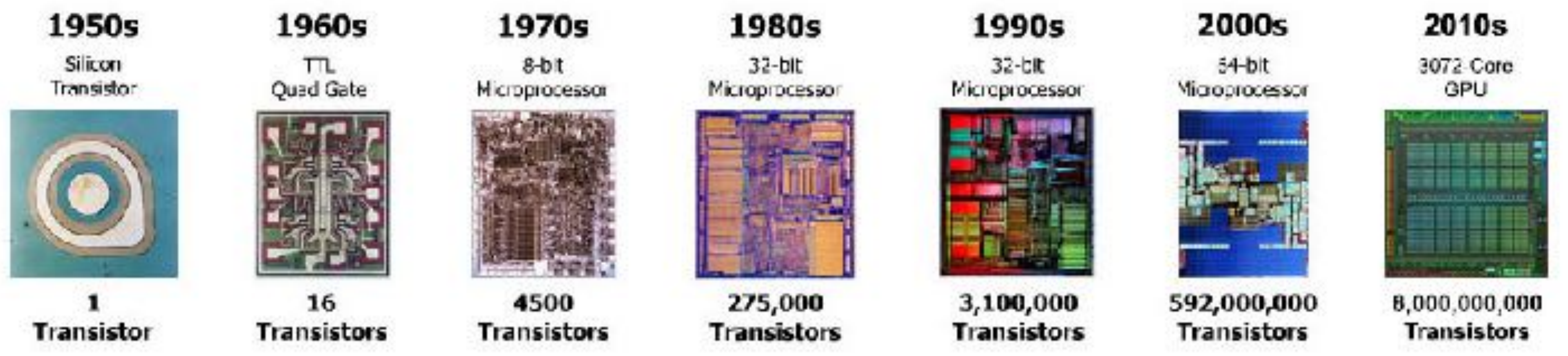
Your PC ran into a problem that it couldn't handle, and now it needs to restart.

[illegible]

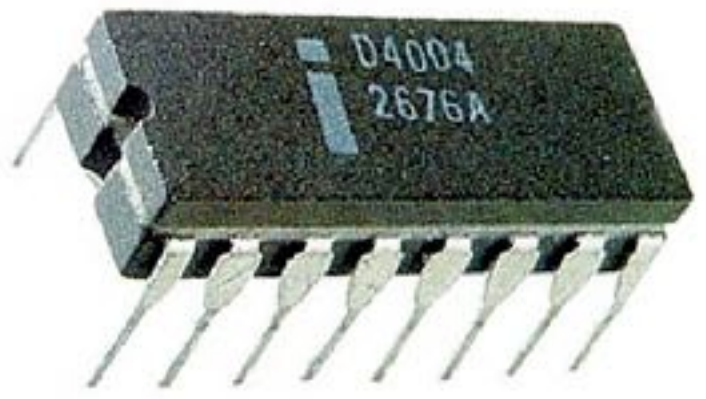
Erratum

- A correction of a published text
- As a general rule, publishers issue an erratum for a production error
- Design errors and mistakes in a CPU's hardwired logic may also be documented and described as errata
- Can be fixed by new silicone stepping, new µcode or BIOS/kernel update
- F00F bug is erratum #81 in Intel‘s „Pentium specification update“ doc
- *„Invalid operand with locked CMPXCHG8B instruction“*
- Fixed in stepping B2

A little bit of history…



1971: i4004 – the Beginning



- 4 bit architecture
- 256 bytes ROM
- 32 bit RAM
- 16 4bits registers
- 10 bit shift register
- PMOS, 2,300 transistors
- 60,000 ops
- maximum operating frequency 740 KHz.
- Ucc 15V
- Family of 7 supporting chips
- No errata ☺️

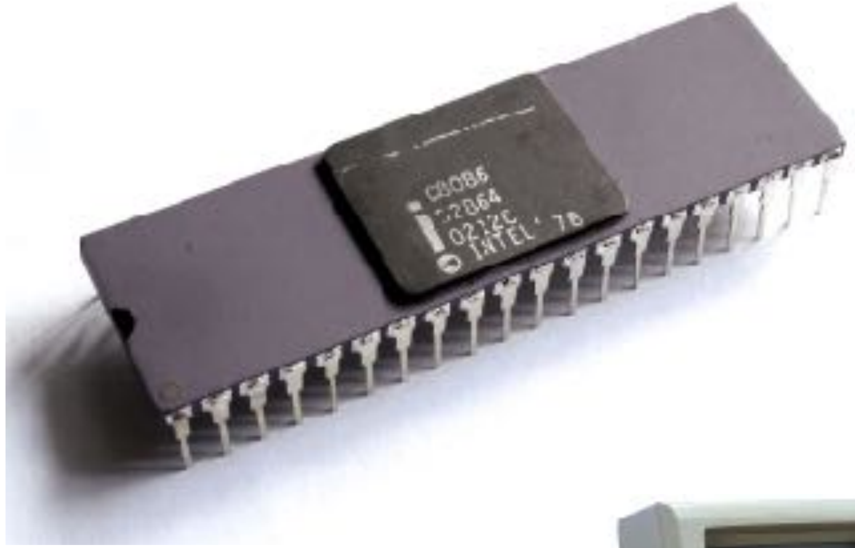
1972: i8008

- 8 bit
- 16kB memory
- 8bit registers A,B-E, H,L + 14b
- Internal 7 level stack
- Clock 500-800 kHz
- Up to 80k ops
- PMOS, 3500 transistors
- Still needs significant amount of supporting logic
- No errata

1974: i8080

- Instruction set and register set compatible with i8008
- Opcodes not binary compatible
- Stack pointer register SP
- Registers can be coupled to 16b
- 64 kB memory
- NMOS, 6k transistors
- 0,64 MIPS
- Ucc +12v, +5V, -5V
- Clock 2-3.125 MHz
- Needs only 2 supporting chips
- Still no errata

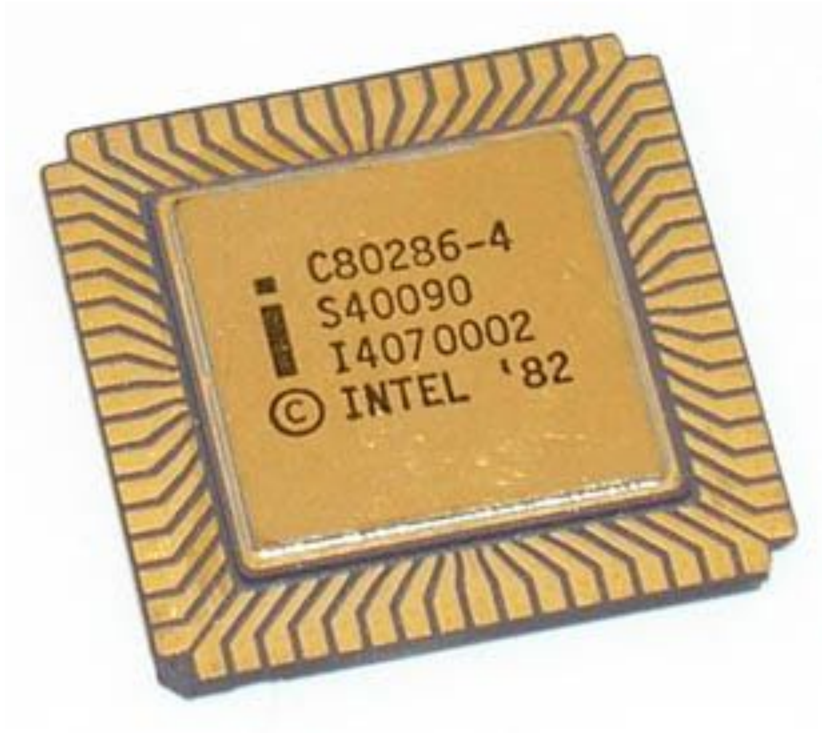
1978: i8086



- First fully 16 bit CPU – x86 is raising
- Source compatible with i8080
- First µCode
- 1MB memory
- Segment regs (gen regs are still 16b)
- External FPU
- Clock 5-10 MHz
- HMOS (NMOS), 29k transistors
- Ucc 5V
- First known (not official) CPU design bugs:
  - Interrupts Following MOV SS,xxx and POP SS Instructions May Corrupt Memory
  - Interrupted String Instructions With Multiple Prefixes Do Not Resume Correctly

1982: i80286

- 16 bits
- First MMU (protected mode)
- Still separate FPU
- 16MB memory
- NMOS, 134k transistors
- Clock up to 12.5 MHz
- Up to 2.66 MIPS
- 9 unofficial bugs (i.e. POPF) for B-2, B-3 steppings



1985: i386

- Entering 32b era – 32b gen registers
- Paging, VM86, new DR and CR regs
- Adressable 4GB memory
- Initialy 12MHz, later up to 33MHz
- CHMOS, 275k transistors
- External FPU and cache
- 11 MIPS
- Ucc 5V, later 3 - 3.3V
- No official errata, unofficial sources for A1 stepping counts 31 bugs
- Well known Multiply bug - fixed CPUs are marked ΣΣ

1989: i486

- Faster i386 with FPU integrated
- Internal 8-16 kB WT/WB cache and FPU
- New instructions (i.e. CMPXCHG)
- Clock 20-100 MHz
- 1.2M transistors
- Ucc 2.5-5V
- No errata found

1993: Pentium, Pentium MMX (P5)

- Improved cache (asociativity, separate I&D)
- Dual ALU pipeline
- Branch predictor (BPB 256-512 entries)
- 57 new MMX instructions
- 64b external bus
- Much faster FPU
- Clock 60-300MHz
- BiCMOS, 3.3M transistors
- Ucc 3.3-5V
- First official specification update: counts 81 errata
    - FDIV bug – replacement program
    - F00F bug

1995: Pentium Pro to Pentium III (P6)

- Speculative execution, out of order completion
- Register renaming
- Extended pipeline (from 5 to 14 stages)
- SSE
- L1 cache 2x16kB
- L2 cache up to 2MB
- Clock 150-1400 MHz
- PSN – privacy issues
- 90 errata (37 fixed)
  - #5: Fast Strings REP MOVS may not transfer all data

2000: Pentium 4

- Aka Netburst
- Hyperthreading
- Rapid execution (ALU clock doubled)
- Execution trace cache (µop cache)
- Power dissipation problems (1 core TDP 115W)
- Max clock 3.8 GHz (failed to reach 10GHz planned)
- Hyperpipeline (20-31 stages) – improved branch predictor
- Planned 40-50 pipeline stages, abandoned
- Errata N1-N100 (49 fixed)
  - N29: REP MOV Instruction with Overlapping Source and Destination May Result in Data Corruption

2006: Intel Core

- Return back to P6 after NetBurst fail
- Multiple cores (1 to 6)
- Larger L1 cache (32+32kB)
- No hyperthreading (P6)
- VT-x
- SSE3
- clock 2.13-3.3 GHz
- Errata Ax1-Ax129
- Huge public (Theo de Raadt, Linus Torvalds) response for errata
  - AI21: Global Pages in the Data Translation Look-Aside Buffer (DTLB) May Not Be Flushed by RSM instruction before Restoring the Architectural State from SMRAM

Following Intel Core generations

2008 Intel Pentium Dual Core (Nehalem) : AN1-AN112 (26 fixed)

2011 Intel Core 2nd gen (Sandy Bridge) : BJ1-BJ138 (2 planned to fix)

2012 Intel Core 3rd gen (Ivy Bridge) : BV1-BV116 (none fixed)

2013 Intel Core 4th gen (Haswell) : HSD1-HSD173 (none fixed)

2014 Intel Core 5th gen (Broadwell) : BDM1-BDM129 (9 fixed)

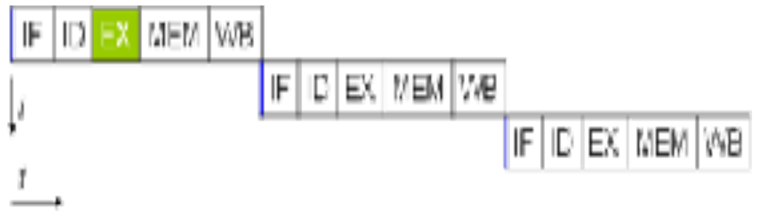
2015 Intel Core 6th gen (Skylake) : SKL1-SKL159 (none fixed)

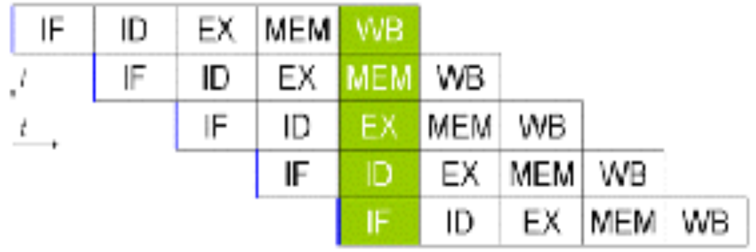
2016 Intel Core 7th gen (Kaby Lake) : KBL001-KBL103 (1 fixed)

2017 Intel Core 8th gen (Cofee Lake) : 001-090 (none fixed)

CPU pipeline

- Since 1961 (IBM stretch project)
- Bubbles (data dependency, branching)
- Multiple execution units, out of order execution, register renaming (Thomasulo algorithm - 1967)
- Speculation, instruction retirement, branch prediction





Intel pipeline optimization

- 4004 – 8080: each instruction takes several clock cycles
- 8086 – 80386: prefetching queue (6,8,16 bytes), loosely coupled units
- i486: first tightly coupled pipelining (simple instruction per cycle)
- Pentium: dual integer 5 stage pipeline (up to 2 instructions per cycle), 2-bit saturated counter branch predictor
- P6: speculative execution, out of order completion, register renaming (pipeline 10-14 stages), 2-level local branch predictor (4-bits history buffer)
- Netburst: HT (2 register sets to fill pipeline 20-31 stages long), 2 level BTB, RSB, IB target array, loop detector
- Core: 14-19 stage pipeline, larger BTB, improved branch predictor – details unknown

Speculation side effects

- Internal CPU state: registers, caches, counters etc.
- Can be accessible directly or indirectly (side channel method)
- CPU state should be modified only on instruction retirement
- State is modified (cache, branch predictors) also by unretired instructions

-> problem

- Short malicious code can be executed speculatively and modified CPU state can be read out
- There are several new attacks using speculative state modification in combination with side channel state exfiltration
- All CPUs with speculative execution are vulnerable (i.e. Intel CPUs since P6)

Spectre

- All current CPUs with speculative execution are vulnerable
- Attack stages:
    1. **Preparation:** branch predictor training with valid parameters, side channel init
    2. **Speculation**: speculative execution with invalid parameters
    3. **Extraction**: read out modified state through side channel

Internal CPU structures

- Side channel can be
  1. Cache (flush&reload, prime&probe, evict&time…)
  2. Pattern History Table (2-level branch predictor)
  3. Branch Target Buffer
  4. Return Stack Buffer
  5. Translate Lookaside Buffer …
- Side channel is affected by noise (usually 1-3%)

What data can be read ?

- Different process memory space
- Kernel memory space
- Hypervisor space
- Different VM space
- GSX enclave
- SMM

How it works – Spectre v1

```
for (i = 0; i < predictor_buffer_len; i++)    // number of repetitions architecture dependent
  func(1);                                     // predictor training for index validity check
clflush();                                     // eviction can be used
func(attack_index);                            // index is out of range
for (i = 0; i < 256; i++)                      // go through 256 cache lines
   time = measure_access(array1[i]);           // one access (cache hit) will be shorter

void func(index) {                             // victim code
  if (index < array2_size)                     // validity check
    tmp ^= array1[array2[index]*clsize] ;      // 2x access to memory needed. Array1 is indexed
}                                              // memory content at addr &array2[]+index.
                                               // Corresponding cache line to byte value is
loaded
```

How it works - Spectre v2

- Find a gadget like tmp ^= array1[array2[index]] in a victim function
- Find an indirect jump/call in a victim function
- The victim function must be callable from attacker space (a shared library can be used)
- Make a copy of a page containing gadget in attacker space (CoW)
- Replace the gadget code with the RET instruction
- Train IBP buffer in attacker space with series of calls targeting gadget address (former code replaced by RET)
- Prepare index value, flush cache containing address of indirect jump and call victim function
- Mistrained predictor will jump to a gadget with attacker index value
- Side channel read is same as with Spectre v1

Spectre – mitigation 1/4

- Masking index value before test:

```
tmp = array1[array2[index & 0xff]];
if (index < maxsize)
    …
```

- Used in linux kernel after source code static analysis

Spectre – mitigation 2/4

- Retpolines
- Created at google zero labs
- Inserted by compiler
- Example:

```
jmp *%r11
```

```
  call set_up_target;     (1)
capture_spec:             (4)
  pause;
  jmp capture_spec;
set_up_target:
  mov %r11, (%rsp);       (2)
  ret;                    (3)
```

Spectre – mitigation 3/4

- Fence:

```
if (index < maxsize) {
    asm („lfence“);
    tmp = array1[array2[index]];
}
```

- LFENCE instruction does not execute until all prior instructions have completed locally, and no later instruction begins execution until LFENCE completes
- C/C++ equivalent is _mm_lfence(void)

Spectre – mitigation 4/4

- New control bits in MSR – IA32_SPEC_CTRL.IBRS, IA32_SPEC_CTRL.STIBP, IA32_PRED_CMD.IBPB
- Introduced by µcode update on 2/2018 for almost all Core2 CPUs
- Indirect Branch Restricted Speculation (IBRS): Restricts speculation of indirect branches.
- Indirect Branch Predictor Barrier (IBPB): Ensures that earlier code's behavior does not control later indirect branch predictions.
- Single Thread Indirect Branch Predictors (STIBP): Prevents indirect branch predictions from being controlled by the sibling Hyperthread.
- IBRS – huge performance impact, refused for linux kernel (retpolines used instead).

Meltdown

- Specific for Intel CPU
- CPU checks access priviledges at the end of speculative execution (before retirement)
- Approach same as Spectre, but simpler – user space code can speculatively read kernel memory.

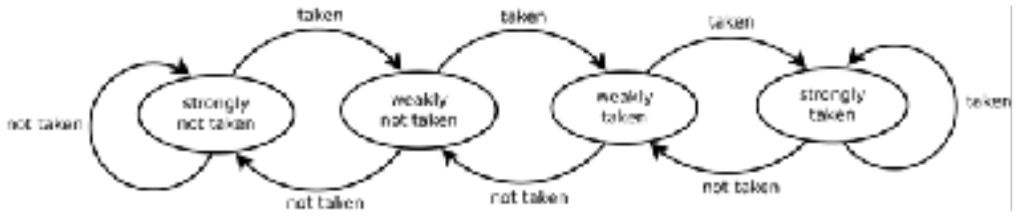
```
retry: mov al, byte [rcx]
       shl rax, 0xc
       jz retry
       mov rbx, qword [rbx + rax]
```

Meltdown – mitigation

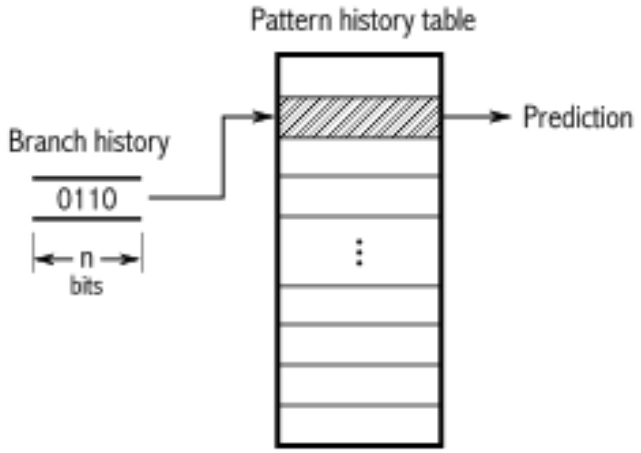
- Kernel page table isolation (KPTI) – separate PTT for user and kernel spaces
- User space code cannot see kernel pages – no mapping exists
- Performance impact – for each syscall CR3 must be reloaded and TLB flushed (new CPUs has PCID feature – only part of TLB must be flushed)

BranchScope

- Attacking BPU itself
- BPU consists of BTB and DP
- Side channel is Pattern History Table – part of BPU
- Ability to determine victims branch taken/not taken state



1st level predictor – saturating counter (FSM)



2nd level adaptive predictor – each entry is FSM

How it works - BranchScope

- Forces to use 1-level predictor (T/nT decision dependent only on PC)
- Slows down victim proces (i.e. modifies scheduler) to allow only single branch execution during context switch
- Stages:
    1. Prime PHT entries (series of random branches – about 100k branch and/or nop instructions needed)
    2. Victim code execution (PHT state change) with attacker branch intensive code running to keep victim using 1-level predictor
    3. Probe PHT entry (examine branch target) – FSM uses 2-bit counter, by TT, NN probe can be current state determined. Resulted time is measured to decide miss/hit.

Branchscope mitigation

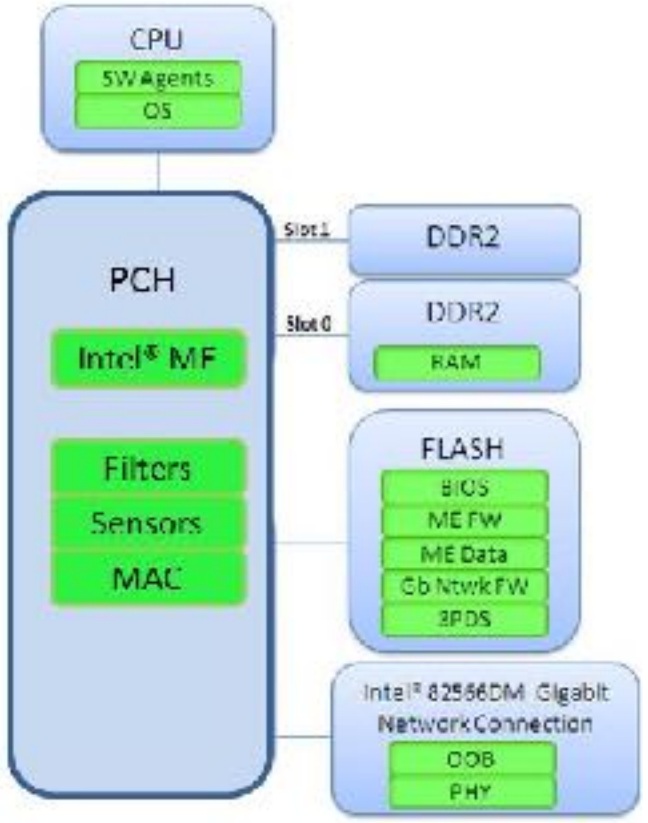
- SW based:
  - Conditional instructions (i.e. CMOV) instead of branches
  - Change algorithms to remove branch dependecies on data
- HW based:
  - PHT randomization (adding a random input to index function)
  - Explicit BPU disablement for sensitive branches
  - BPU partitioning (i.e. separate BPU for user, kernel and GSX)

Intel SGX

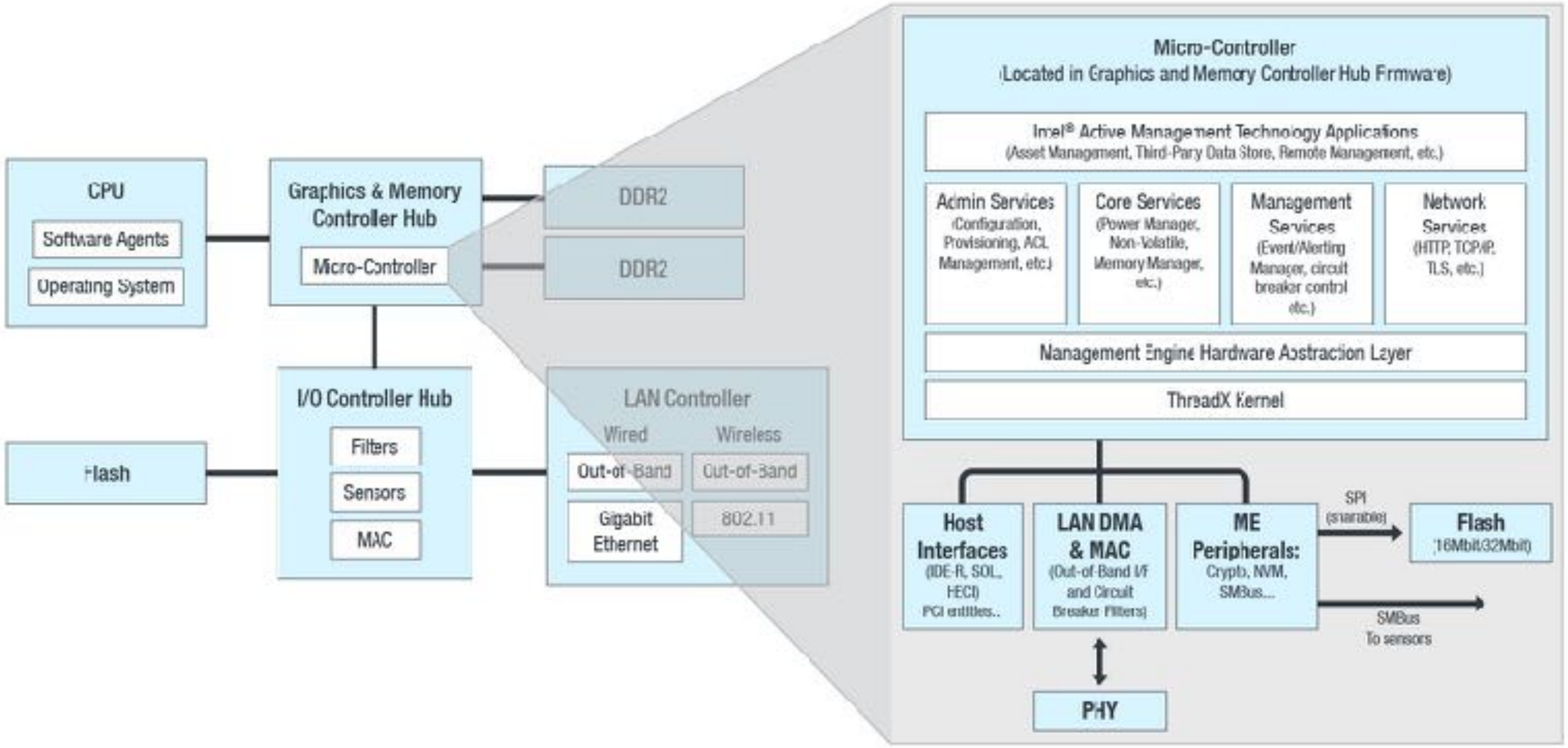
- Software Guard Extensions
- Since Skylake
- CPU circuits and instructions providing protection against compromised user and system code
- Even kernel and hypervisor has no access to Secure Enclave
- Protection against physical memory attacks (encrypted memory)
- Data are encapsulated and code can be entered and leaved only by special instructions through Call Gate
- SGX is proved to be Spectre and BranchScope vulnerable

Intel ME

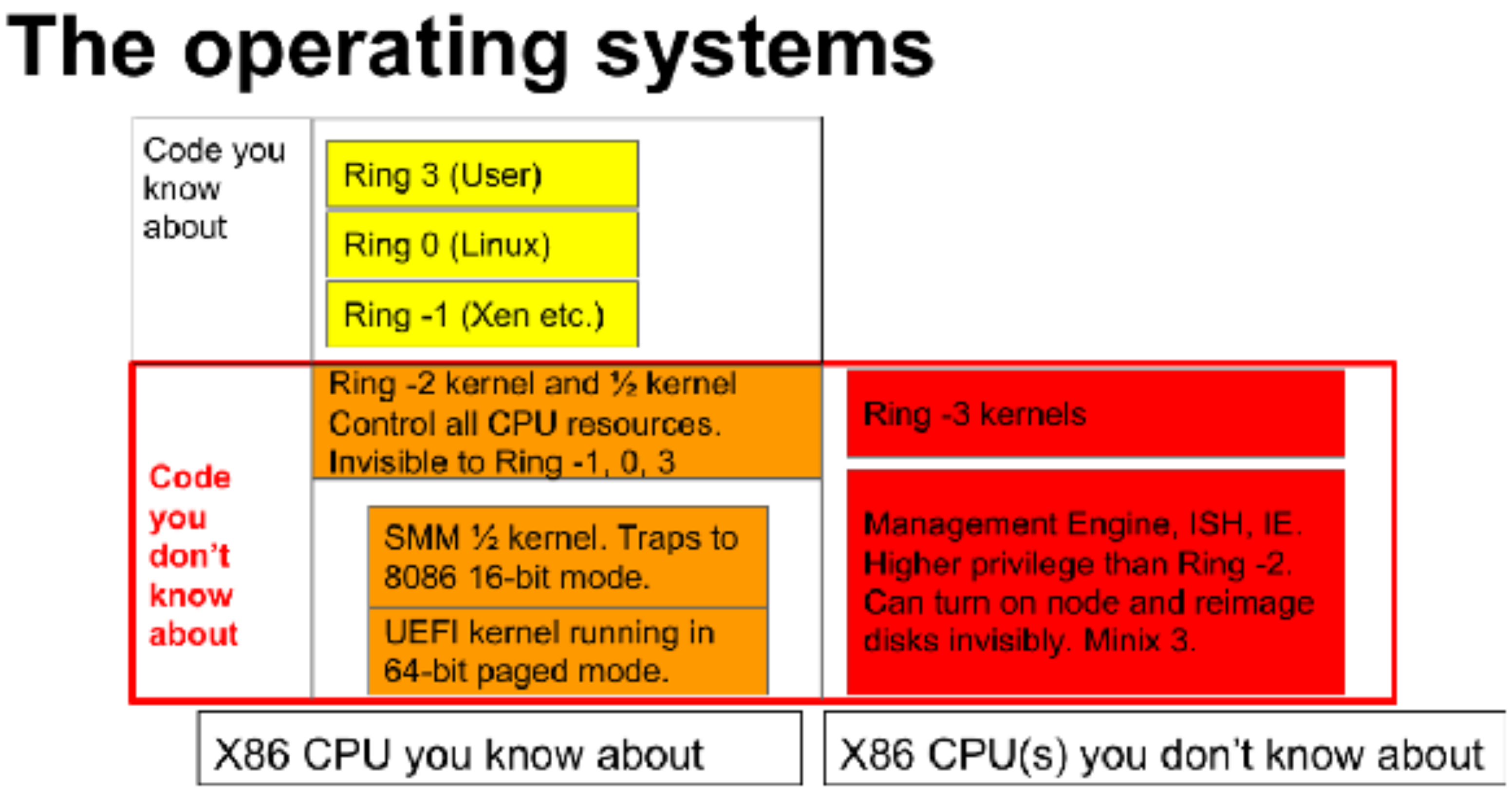
- Computer embedded in the PCH (initially MCH)
- Own MAC and IP address (OOB interface)
- All network traffic goes to ME first
- Communicates through HCI with host OS
- Serial host console can be attached
- First implemented inside of ICH7 in 2005 for high-end systems
- Since 2008 embedded in all chipsets
- Non optional, part of booting process and power management
- Running independently on the power state of main CPU
- In fact there are 3 x86 CPUs inside the chipset
- Initially ARC core with ThreadX RTOS
- Since ME 11.x Intel Quark x86 with MINIX 3



Intel AMT


CPU
Software Agents
Operating System
Graphics & Memory Controller Hub
Micro-Controller
DDR2
DDR2
I/O Controller Hub
Filters
Sensors
MAC
Flash
LAN Controller
Wired
Wireless
Out-of-Band
Out-of-Band
Gigabit Ethernet
802.11
Micro-Controller
(Located in Graphics and Memory Controller Hub Firmware)
Intel® Active Management Technology Applications
(Asset Management, Third-Pary Data Store, Remote Management, etc.)
Admin Services
(Configuration, Provisioning, ACL Management, etc.)
Core Services
(Power Manager, Non-Volatile, Memory Manager, etc.)
Management Services
(Event/Alerting Manager, circuit breaker control etc.)
Network Services
(HTTP, TCP/IP, TLS, etc.)
Management Engine Hardware Abstraction Layer
ThreadX Kernel
Host Interfaces
(IDE-R, SOL, HECI)
PCI entities..
LAN DMA & MAC
(Out-of-Band I/F and Circuit Breaker Filters)
ME Peripherals:
Crypto, NVM, SMBus...
SPI
(sharable)
Flash
(16Mbit/32Mbit)
SMBus
To sensors
PHY

Rings of privileges


The operating systems
Code you know about
Ring 3 (User)
Ring 0 (Linux)
Ring -1 (Xen etc.)
Code you don't know about
Ring -2 kernel and ½ kernel
Control all CPU resources.
Invisible to Ring -1, 0, 3
SMM ½ kernel. Traps to 8086 16-bit mode.
UEFI kernel running in 64-bit paged mode.
Ring -3 kernels
Management Engine, ISH, IE.
Higher privilege than Ring -2.
Can turn on node and reimage disks invisibly. Minix 3.
X86 CPU you know about
X86 CPU(s) you don't know about

What Intel does to secure ME code ?

- Some ME and AMT modules encoded with Huffman code since ME 6.x
- ARC core Huffman dictionary was decoded already
- Quark (ME 11.x+) dictionary is not publicly known yet
- It was claimed on June 2017 that 89% of code and 86% of data are decoded
- Intel released INTEL-SA_00086 advisory for ME 6.x-11.x in Q3 2017
    - CVE-2017-5705
    - CVE-2017-5708
    - CVE-2017-5711
    - CVE-2017-5712

-> Attacker can execute arbitrary code on ME

Silent Bob is Silent

- AMT uses digest auth protocol for Admin user
- In Q2 2017 critical vulnerability was published
- CVE-2017-5689: when empty auth response is sent, Admin is always authenticated
- It is enough to have http proxy and clear a response value
- CVSSv3 score of 9.8 out of 10
- There is > 5000 AMT accessible over internet (found by Shodan)

Summary

- Modern CPUs contain a tens of “errata” in each generation causing unpredictable CPU behavior
- All vendors have same problems
- Speculative execution has exploitable side effects – fix will need architectural redesign
- Most risks can be mitigated on µcode, BIOS, compiler or OS level but have performance impact
- CPU management subsystems have vulnerabilities and are dangerous when exploited