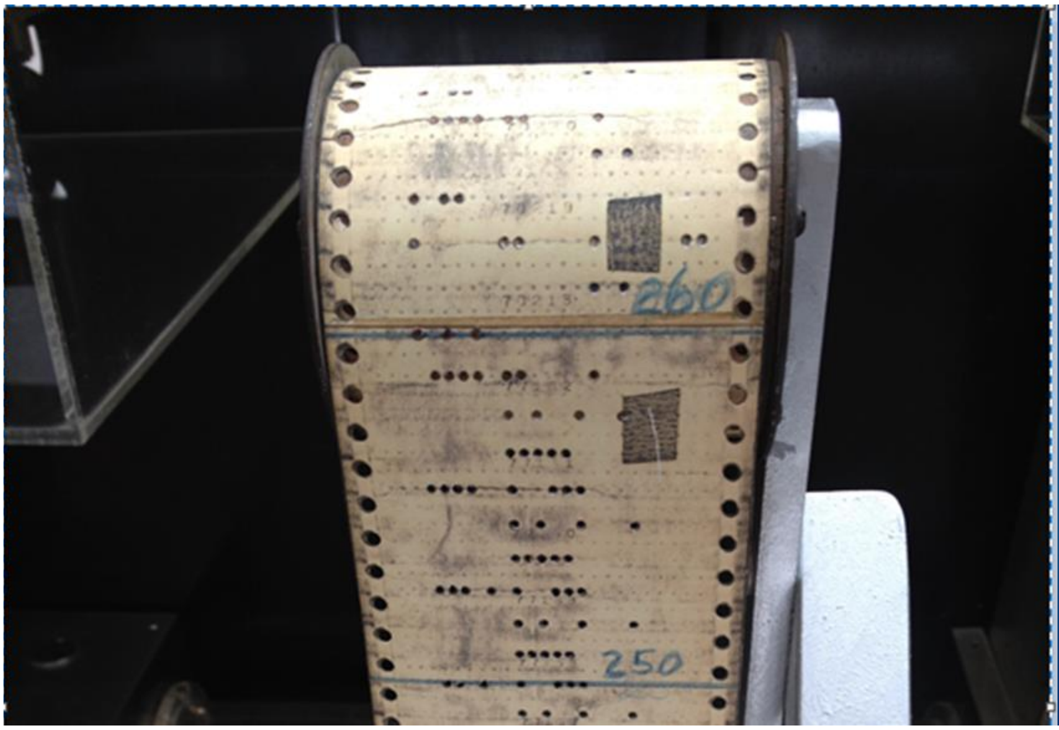
在调试器下理解ARMv8

——虚拟内存

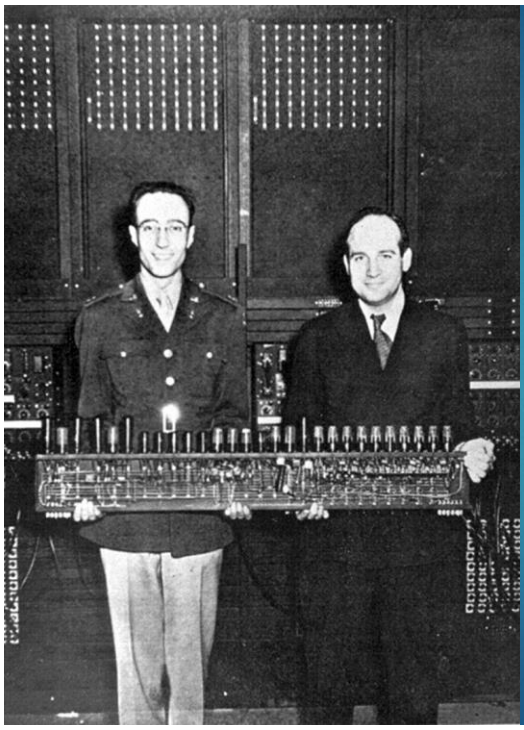
张银奎 2022-01-09 上海

IBM公司与哈佛大学合作的马克一号(~1940)

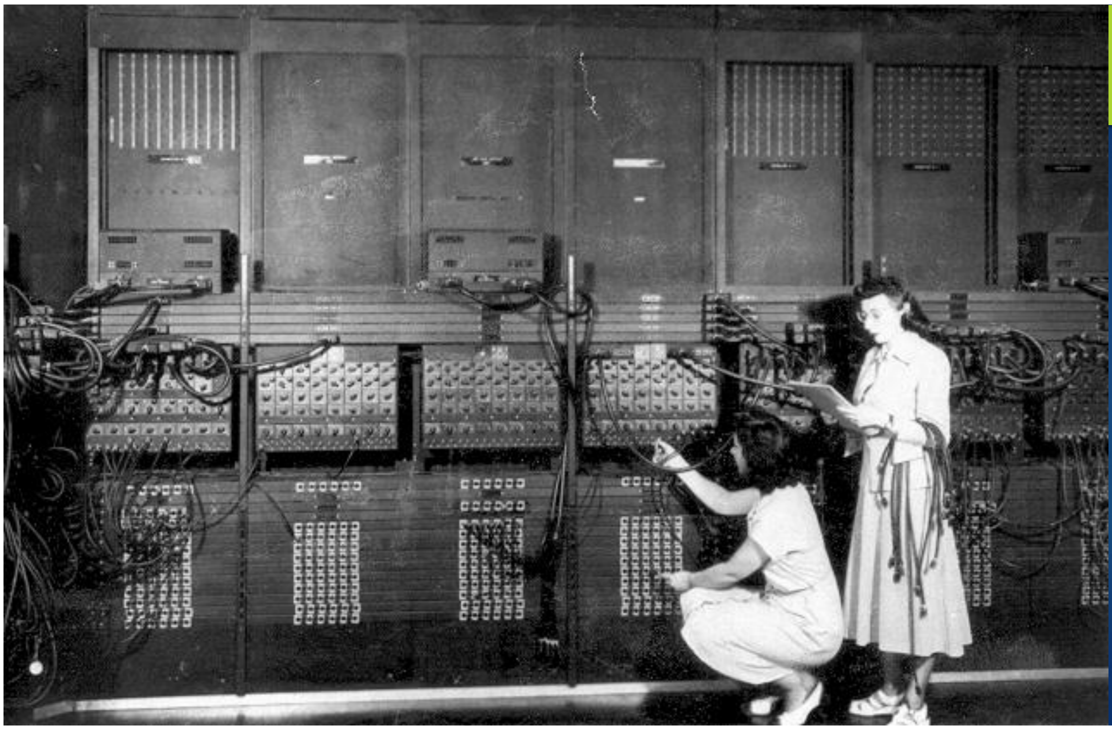


马克一号的程序纸带，两处黑色是真正的“补丁”，把某些孔封上做修改

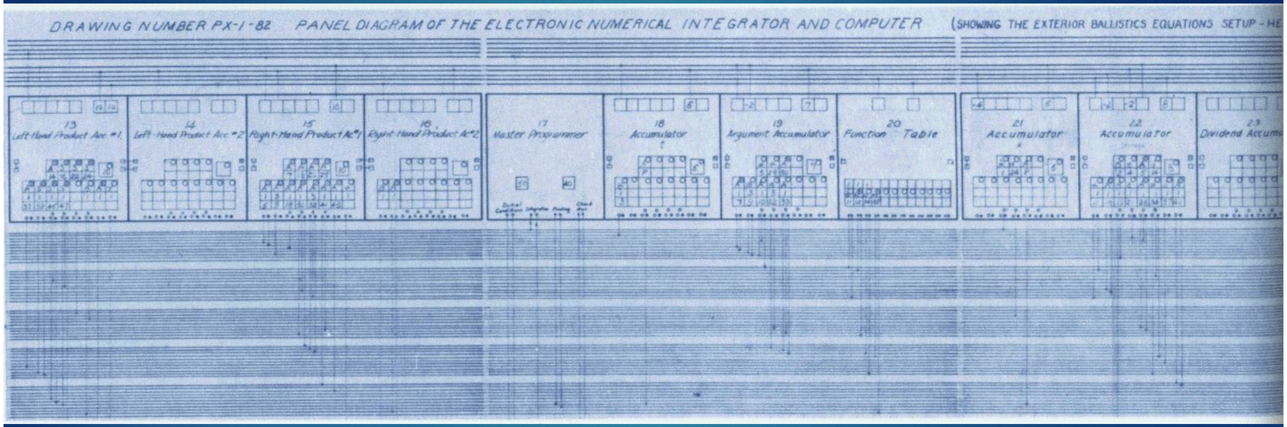
(1940 - 1943)



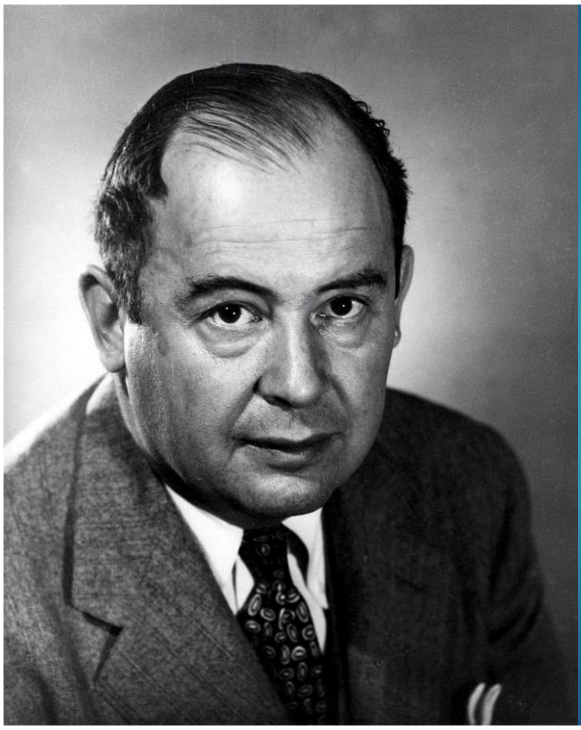
在戈德斯坦和埃克特的合影中，他们身后有ENIAC的四个机柜，其中左面三个便是累加器。每个累加器面板的上方有10X10的灯泡（neon）阵列，用来显示10个戴刻德的值，每一列10个灯泡，上面标有0-9十个数字，与一个戴刻德对应，点亮的灯泡代表当前值。



为ENIAC
加载程序


DRAWING NUMBER PX-1-82 PANEL DIAGRAM OF THE ELECTRONIC NUMERICAL INTEGRATOR AND COMPUTER
(SHOWING THE EXTERIOR BALLISTICS EQUATIONS SETUP - HE
13 Left-Hand Product Acc #1
14 Left-Hand Product Acc #2
15 Right-Hand Product Acc #1
16 Right-Hand Product Acc #2
17 Master Programmer
18 Accumulator
19 Argument Accumulator
20 Function Table
21 Accumulator
22 Accumulator
23 Dividend Accumu

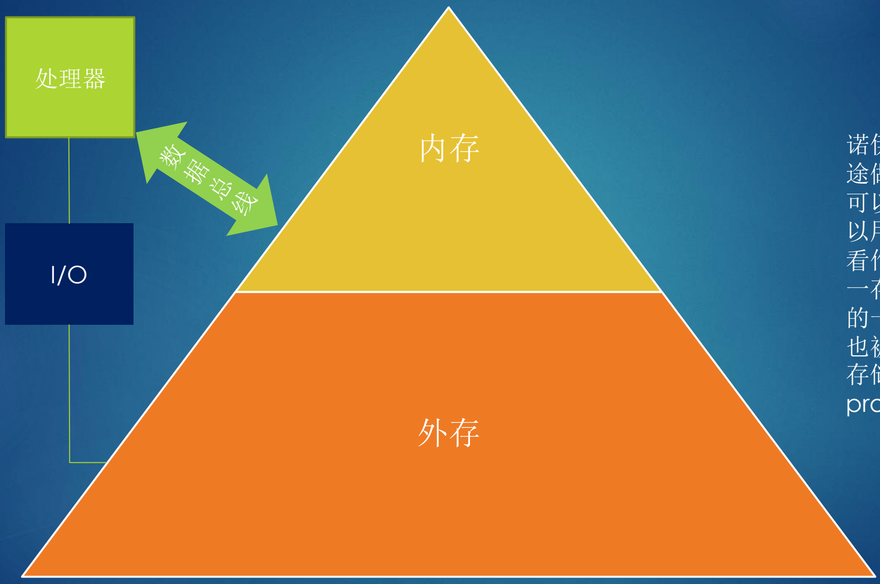
求解弹道方程的ENIAC程序



今天的计算机系统大多都是冯·诺伊曼架构

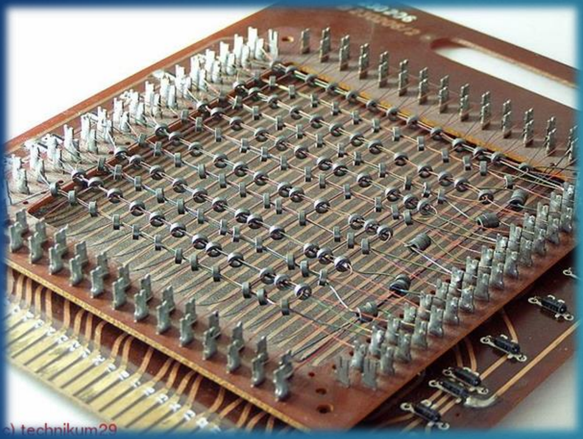
John von Neumann

1903-1957



诺伊曼把内存的角色和用途做了明确定义，它不仅可以用来存储数据，也可以用来存储代码。把代码看作是一种特殊的数据统一存储，这是诺伊曼架构的一大特色。这样的方式也被称为中央程序，也叫存储程序（stored program）。

Core Memory



Magnetic-core memory was the predominant form of random-access computer memory for 20 years between about 1955 and 1975. Such memory is often just called core memory, or, informally, core

VMS
(Virtual Memory System)

VAX
(Virtual Address Extension)

1975年4月，代号为Star的VAX项目开始
1975年6月，Cutler、Dick-Histvedt和Peter Lipman被任命为项目Leader，开发代号为Starlet的操作系统

VMS (Virtual Memory System)

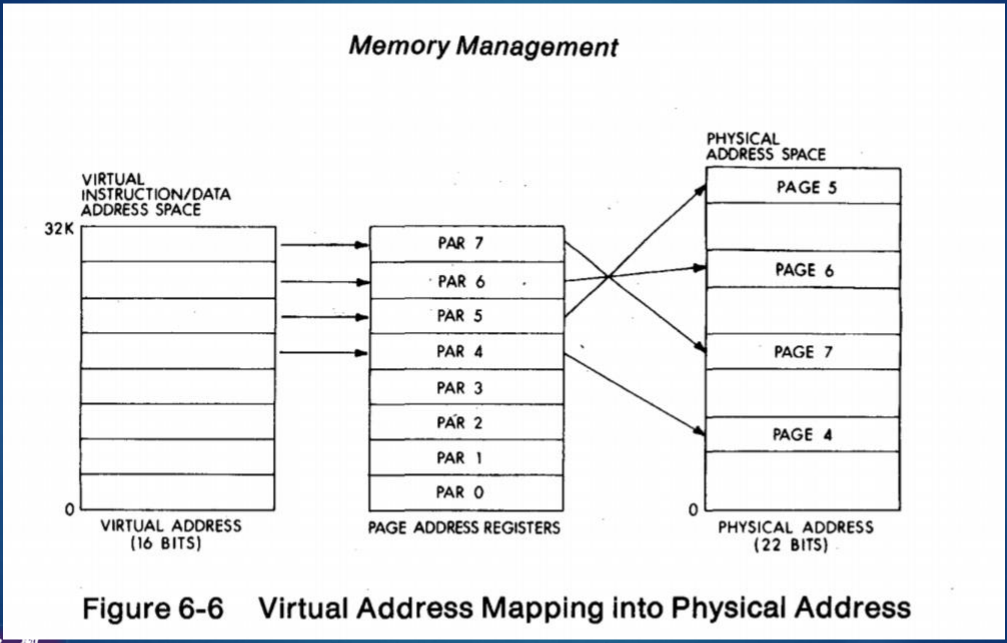
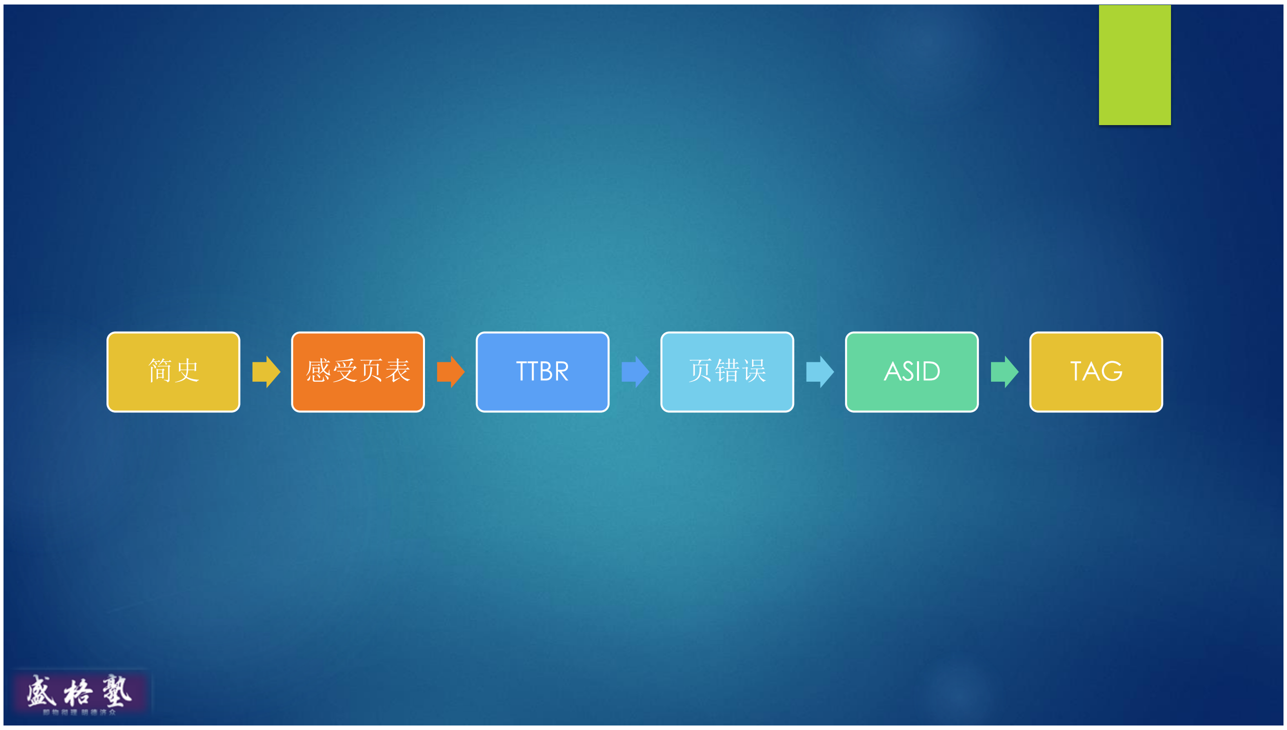


Figure 6-6 Virtual Address Mapping into Physical Address

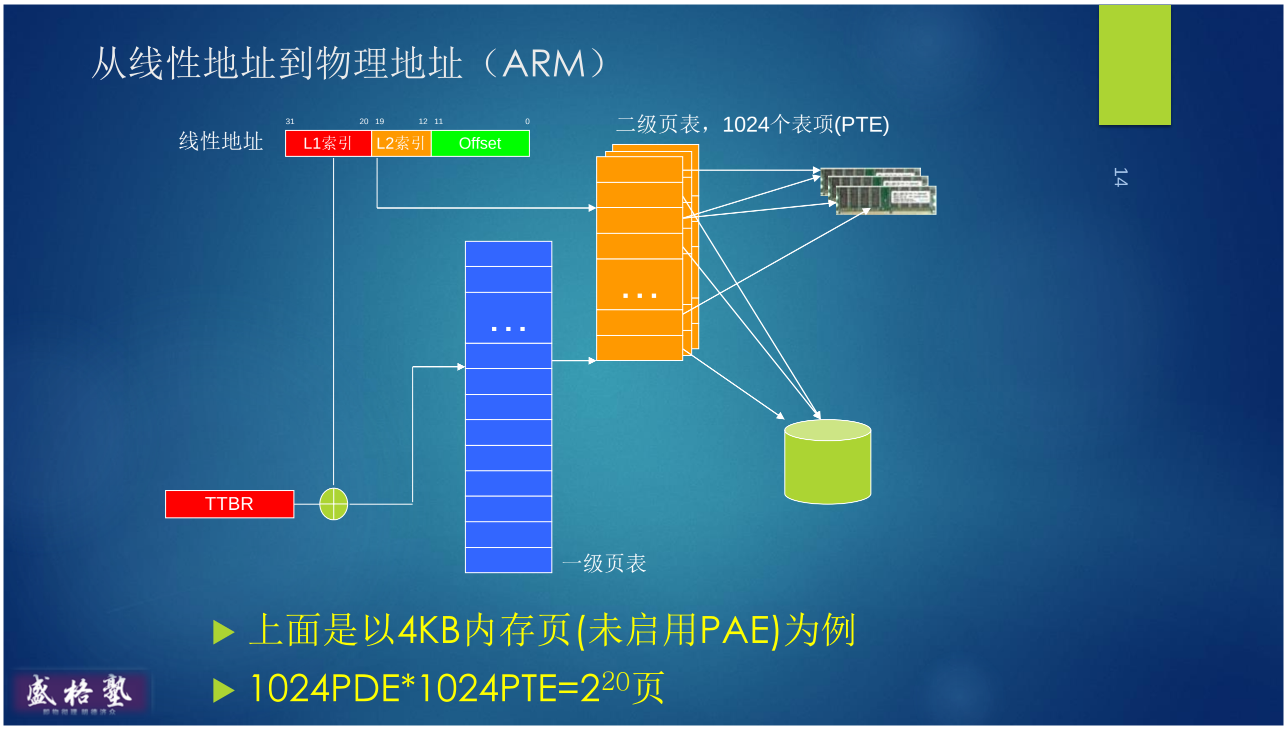
1979年的PDP-11手册中关于虚拟内存的插图

Page Address Register (PAR)

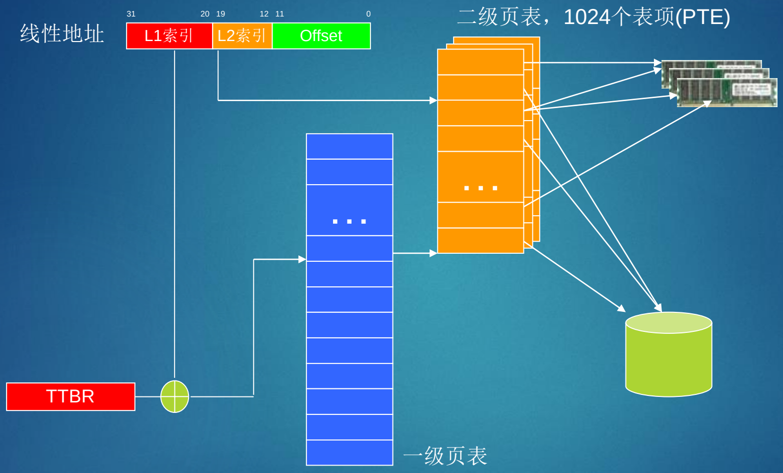


简史 ➡ 感受页表 ➡ TTBR ➡ 页错误 ➡ ASID ➡ TAG

盛格塾

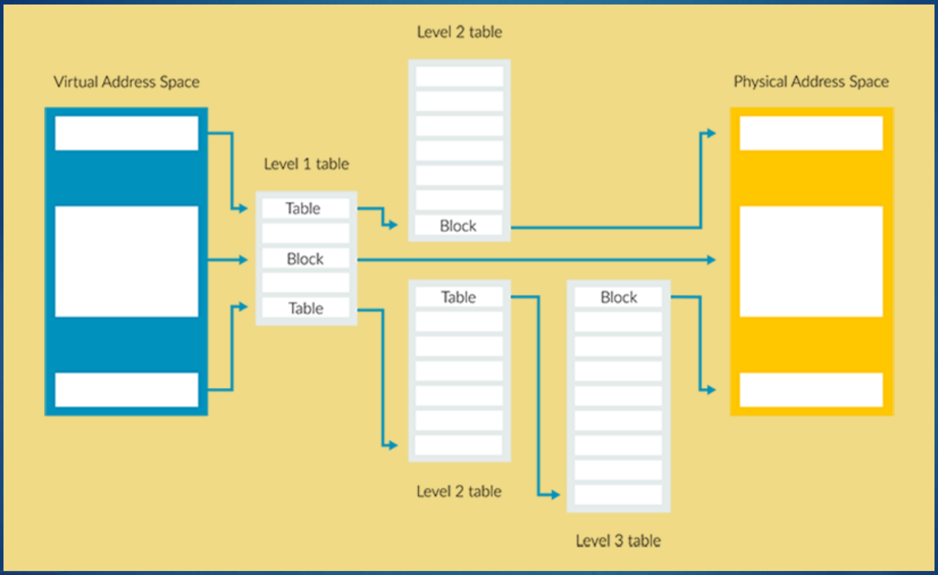


从线性地址到物理地址（ARM）



- 上面是以4KB内存页(未启用PAE)为例
- 1024PDE*1024PTE=2^{20}页

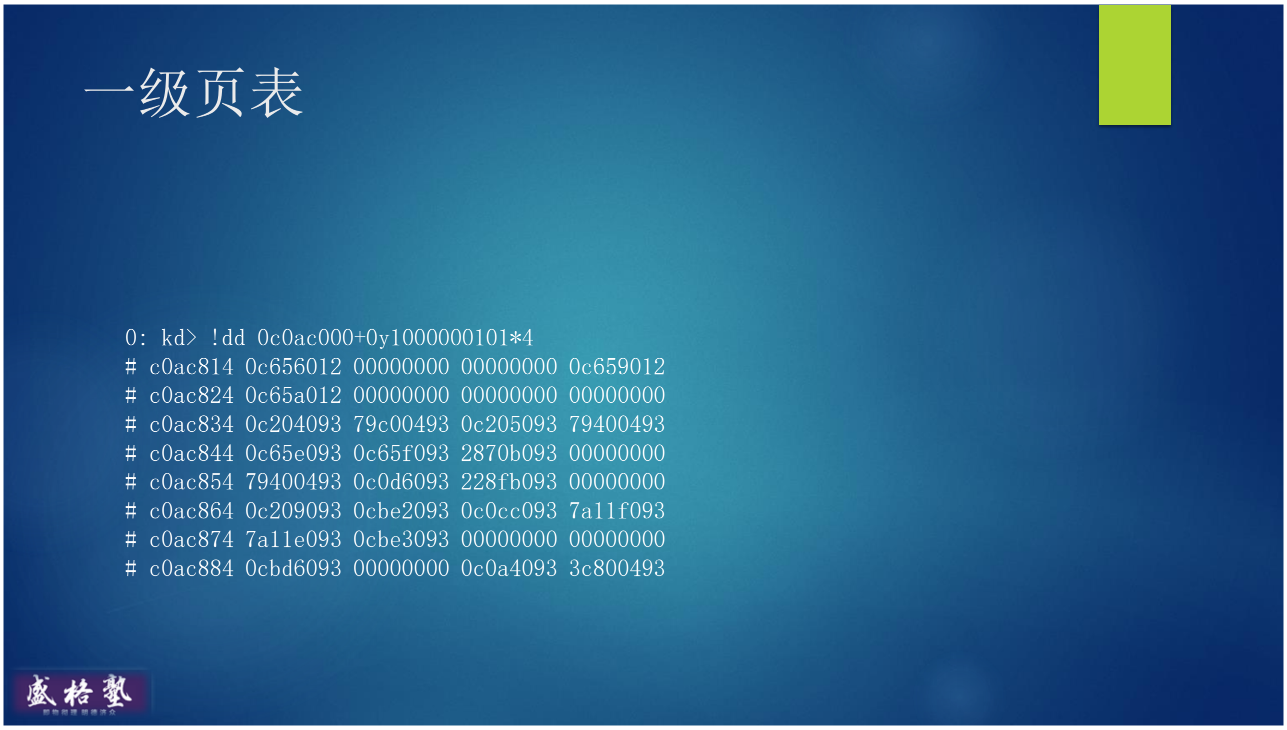
盛格塾



表项分为两种：
- 指向内存块(block)
- 指向表

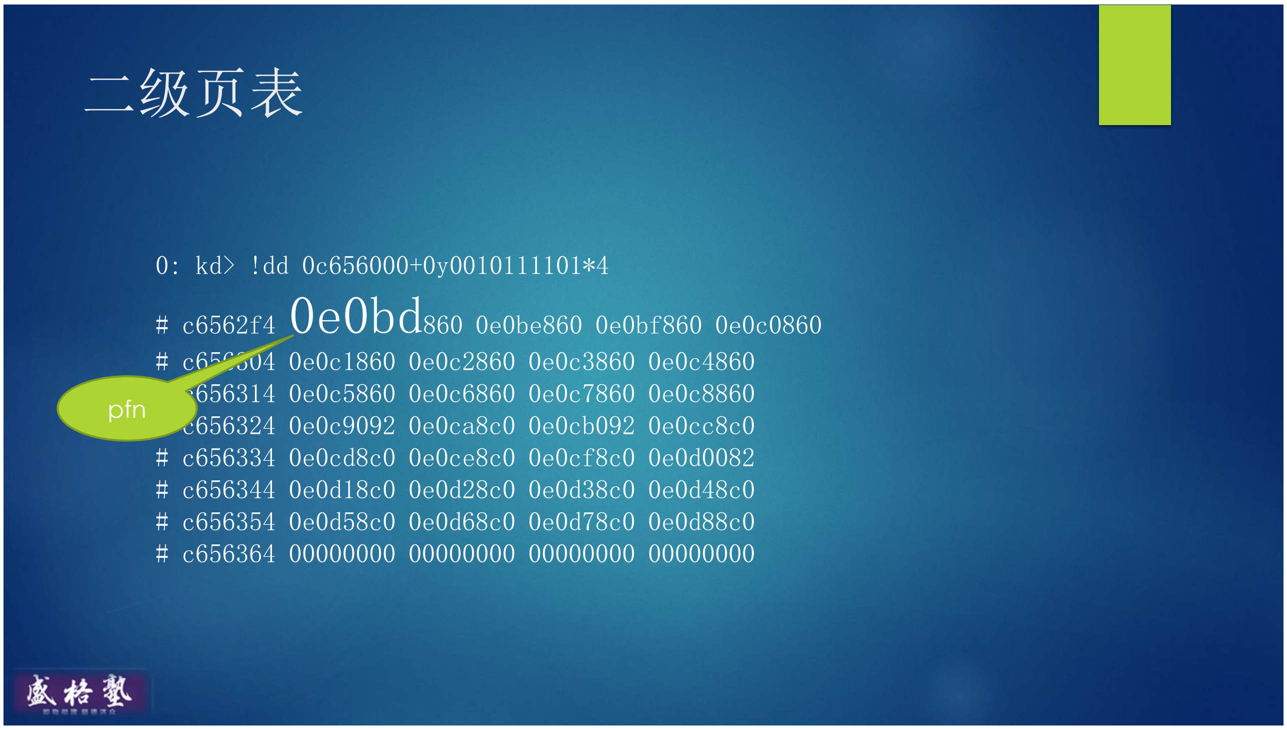
10+10+12

```
0: kd> .formats 814bdb7e
Evaluate expression:
  Hex:     814bdb7e
  Decimal: -2125735042
  Octal:   20122755576
  Binary:  10000001 01001011 1101 1011 01111110
  Chars:   .K.~
  Time:    ***** Invalid
  Float:   low -3.74427e-038 high -1.#QNAN
  Double:  -1.#QNAN
```



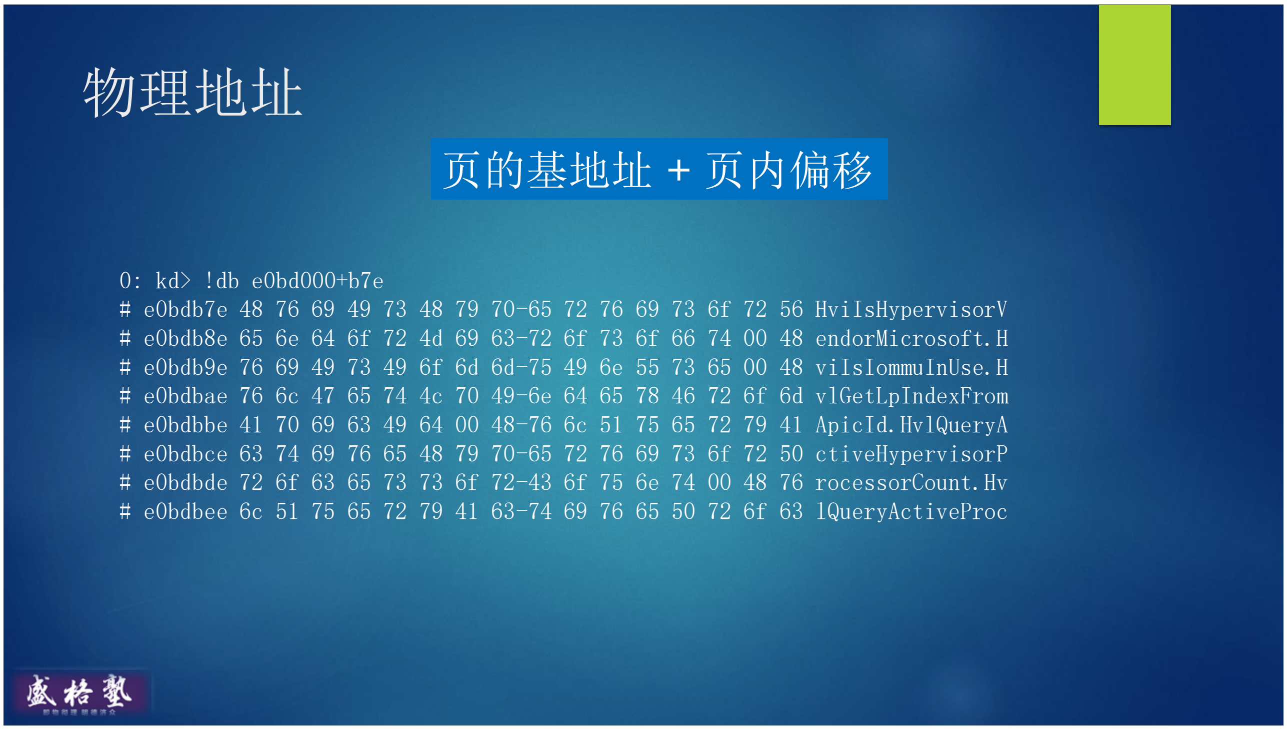
一级页表

```
0: kd> !dd 0c0ac000+0y1000000101*4
# c0ac814 0c656012 00000000 00000000 0c659012
# c0ac824 0c65a012 00000000 00000000 00000000
# c0ac834 0c204093 79c00493 0c205093 79400493
# c0ac844 0c65e093 0c65f093 2870b093 00000000
# c0ac854 79400493 0c0d6093 228fb093 00000000
# c0ac864 0c209093 0cbe2093 0c0cc093 7a11f093
# c0ac874 7a11e093 0cbe3093 00000000 00000000
# c0ac884 0cbd6093 00000000 0c0a4093 3c800493
```



二级页表

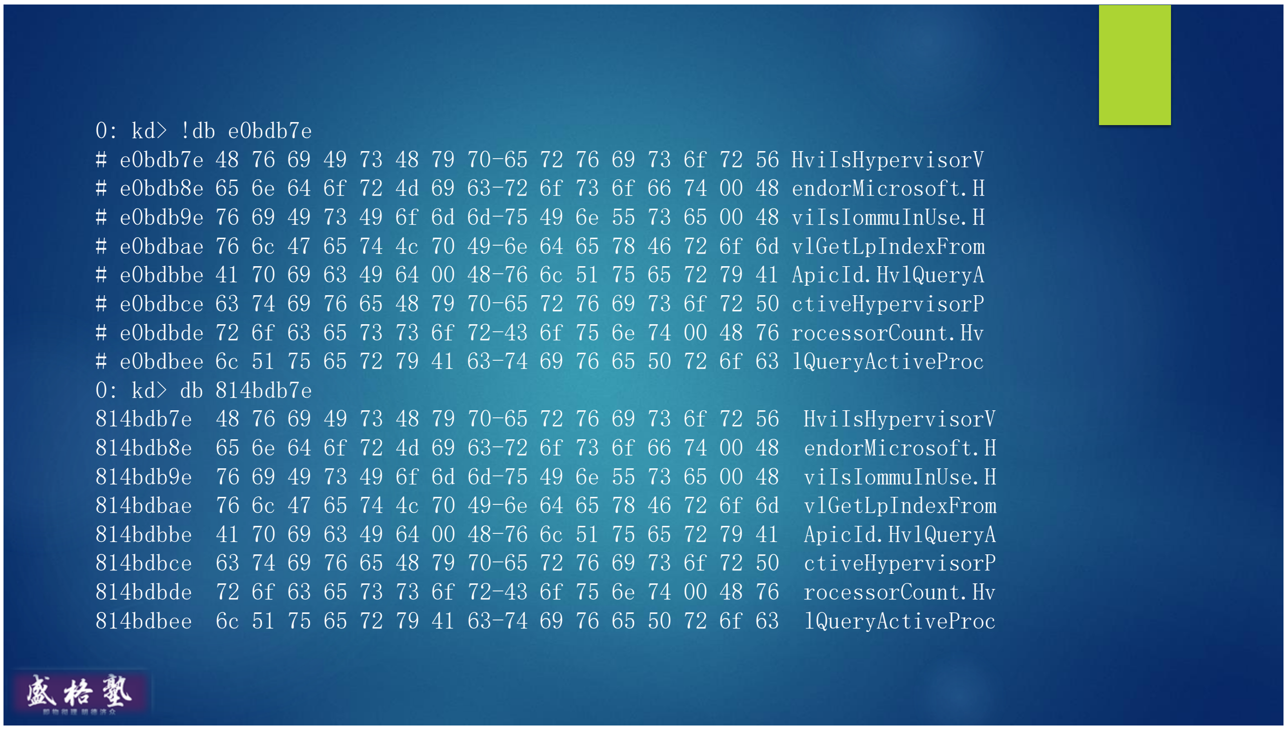
```
0: kd> !dd 0c656000+0y0010111101*4
# c6562f4 0e0bd860 0e0be860 0e0bf860 0e0c0860
# c65[illegible]04 0e0c1860 0e0c2860 0e0c3860 0e0c4860
[illegible]656314 0e0c5860 0e0c6860 0e0c7860 0e0c8860
[illegible]c656324 0e0c9092 0e0ca8c0 0e0cb092 0e0cc8c0
# c656334 0e0cd8c0 0e0ce8c0 0e0cf8c0 0e0d0082
# c656344 0e0d18c0 0e0d28c0 0e0d38c0 0e0d48c0
# c656354 0e0d58c0 0e0d68c0 0e0d78c0 0e0d88c0
# c656364 00000000 00000000 00000000 00000000
```



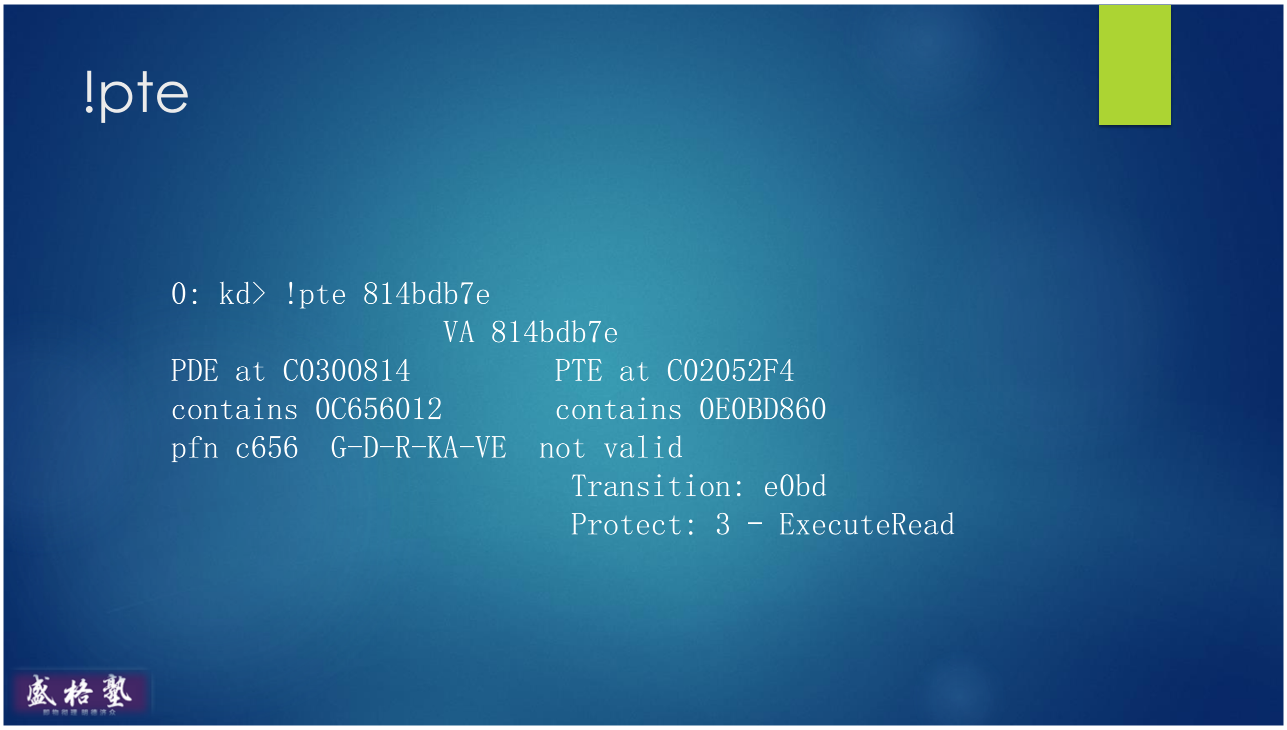
物理地址

页的基地址 + 页内偏移

```
0: kd> !db e0bd000+b7e
# e0bdb7e 48 76 69 49 73 48 79 70-65 72 76 69 73 6f 72 56 HviIsHypervisorV
# e0bdb8e 65 6e 64 6f 72 4d 69 63-72 6f 73 6f 66 74 00 48 endorMicrosoft.H
# e0bdb9e 76 69 49 73 49 6f 6d 6d-75 49 6e 55 73 65 00 48 viIsIommuInUse.H
# e0bdbae 76 6c 47 65 74 4c 70 49-6e 64 65 78 46 72 6f 6d vlGetLpIndexFrom
# e0bdbbe 41 70 69 63 49 64 00 48-76 6c 51 75 65 72 79 41 ApicId.HvlQueryA
# e0bdbce 63 74 69 76 65 48 79 70-65 72 76 69 73 6f 72 50 ctiveHypervisorP
# e0bdbde 72 6f 63 65 73 73 6f 72-43 6f 75 6e 74 00 48 76 rocessorCount.Hv
# e0bdbee 6c 51 75 65 72 79 41 63-74 69 76 65 50 72 6f 63 lQueryActiveProc
```



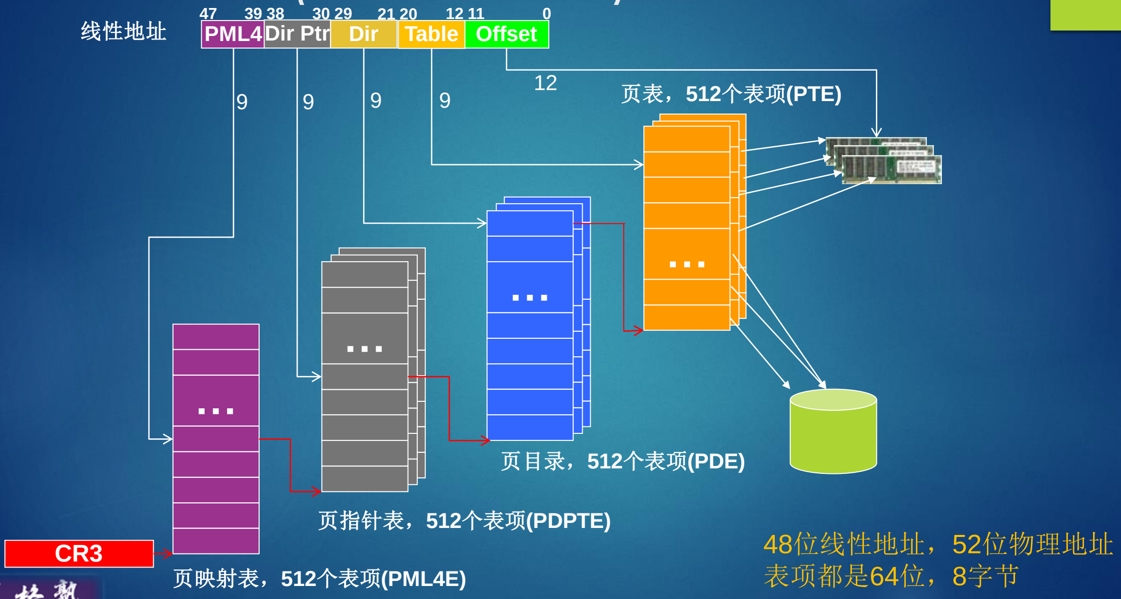
```
0: kd> !db e0bdb7e
# e0bdb7e 48 76 69 49 73 48 79 70-65 72 76 69 73 6f 72 56 HviIsHypervisorV
# e0bdb8e 65 6e 64 6f 72 4d 69 63-72 6f 73 6f 66 74 00 48 endorMicrosoft.H
# e0bdb9e 76 69 49 73 49 6f 6d 6d-75 49 6e 55 73 65 00 48 viIsIommuInUse.H
# e0bdbae 76 6c 47 65 74 4c 70 49-6e 64 65 78 46 72 6f 6d vlGetLpIndexFrom
# e0bdbbe 41 70 69 63 49 64 00 48-76 6c 51 75 65 72 79 41 ApicId.HvlQueryA
# e0bdbce 63 74 69 76 65 48 79 70-65 72 76 69 73 6f 72 50 ctiveHypervisorP
# e0bdbde 72 6f 63 65 73 73 6f 72-43 6f 75 6e 74 00 48 76 rocessorCount.Hv
# e0bdbee 6c 51 75 65 72 79 41 63-74 69 76 65 50 72 6f 63 lQueryActiveProc
0: kd> db 814bdb7e
814bdb7e  48 76 69 49 73 48 79 70-65 72 76 69 73 6f 72 56  HviIsHypervisorV
814bdb8e  65 6e 64 6f 72 4d 69 63-72 6f 73 6f 66 74 00 48  endorMicrosoft.H
814bdb9e  76 69 49 73 49 6f 6d 6d-75 49 6e 55 73 65 00 48  viIsIommuInUse.H
814bdbae  76 6c 47 65 74 4c 70 49-6e 64 65 78 46 72 6f 6d  vlGetLpIndexFrom
814bdbbe  41 70 69 63 49 64 00 48-76 6c 51 75 65 72 79 41  ApicId.HvlQueryA
814bdbce  63 74 69 76 65 48 79 70-65 72 76 69 73 6f 72 50  ctiveHypervisorP
814bdbde  72 6f 63 65 73 73 6f 72-43 6f 75 6e 74 00 48 76  rocessorCount.Hv
814bdbee  6c 51 75 65 72 79 41 63-74 69 76 65 50 72 6f 63  lQueryActiveProc
```



!pte

```
0: kd> !pte 814bdb7e
                    VA 814bdb7e
PDE at C0300814            PTE at C02052F4
contains 0C656012          contains 0E0BD860
pfn c656   G-D-R-KA-VE   not valid
                            Transition: e0bd
                            Protect: 3 - ExecuteRead
```

地址翻译 (64位, 4KB页)



48位线性地址，52位物理地址
表项都是64位，8字节

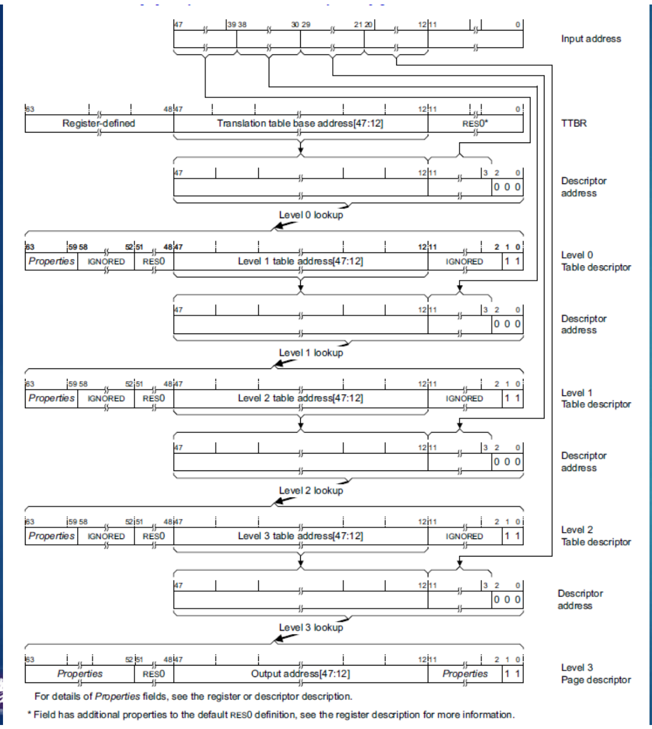
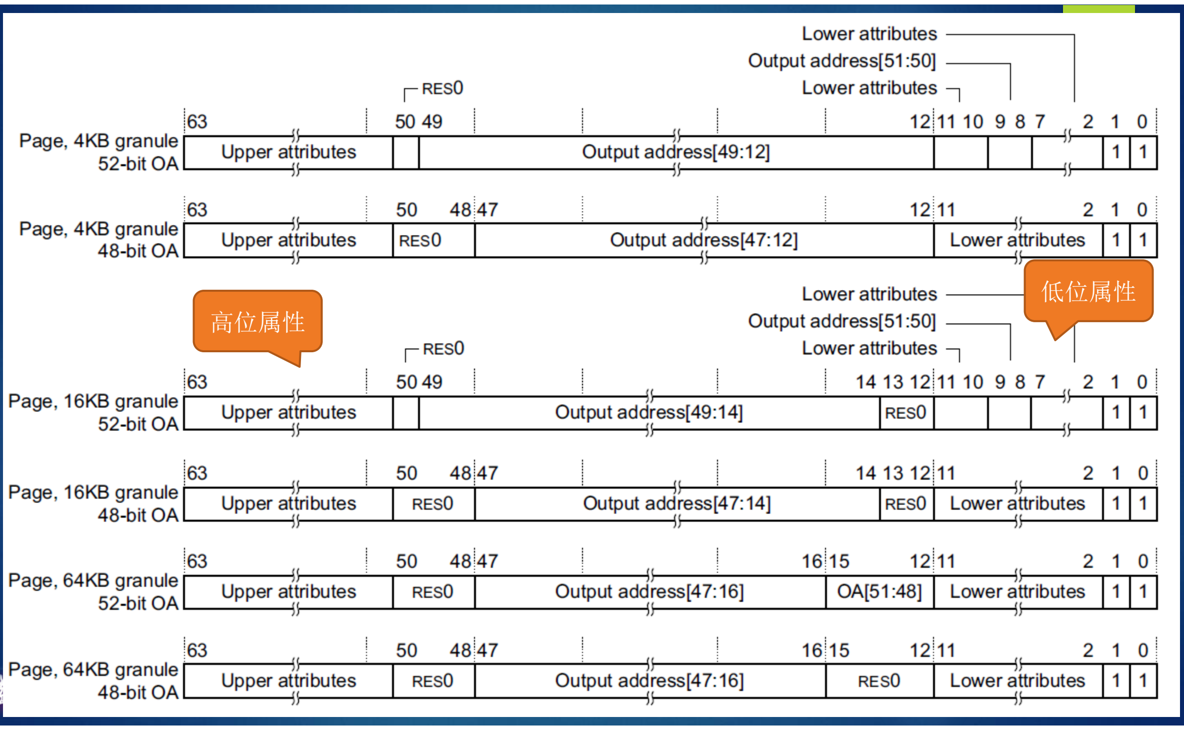
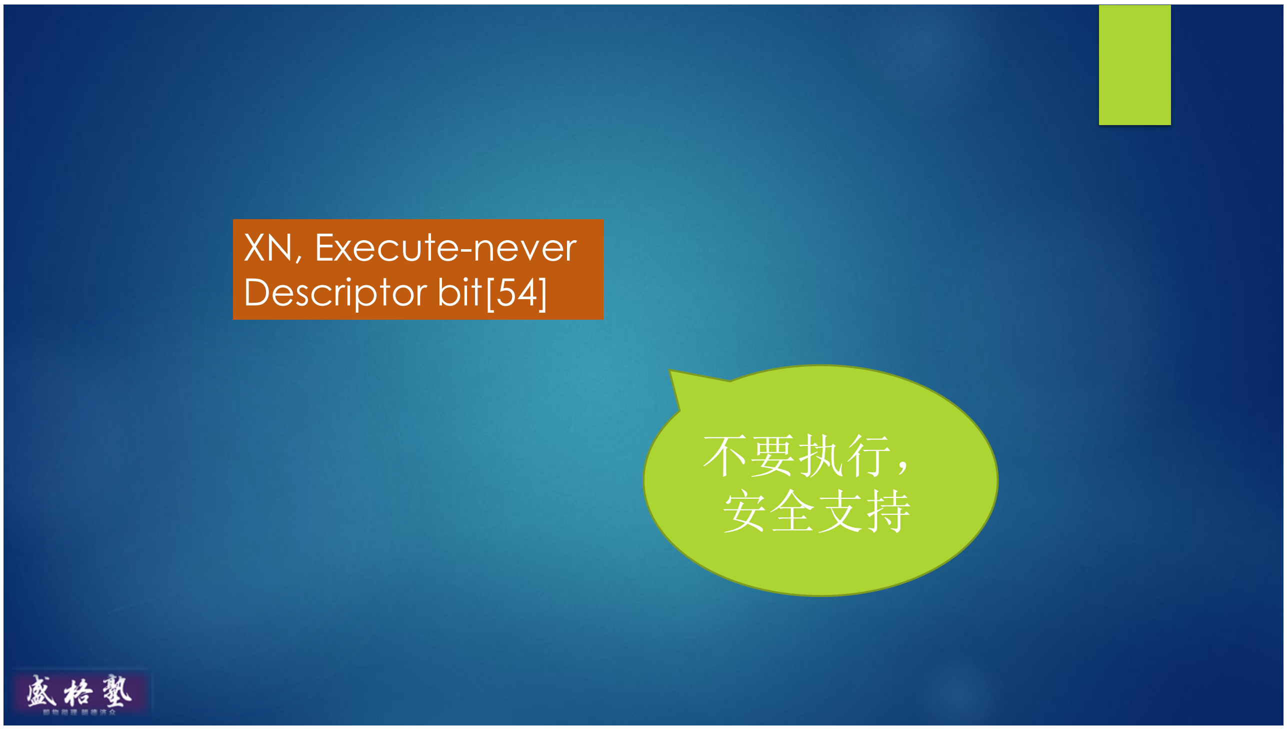

Input address
TTBR
Register-defined
Translation table base address[47:12]
RES0*
Descriptor address
Level 0 lookup
Level 0 Table descriptor
Properties
IGNORED
RES0
Level 1 table address[47:12]
Level 1 lookup
Level 1 Table descriptor
Level 2 table address[47:12]
Level 2 lookup
Level 2 Table descriptor
Level 3 table address[47:12]
Level 3 lookup
Level 3 Page descriptor
Output address[47:12]
For details of Properties fields, see the register or descriptor description.
* Field has additional properties to the default RES0 definition, see the register description for more information.

Figure K7-10 (ARMv8 ARM)

Full translation flow using the 4KB granule and starting at level 0





XN, Execute-never
Descriptor bit[54]

不要执行，
安全支持

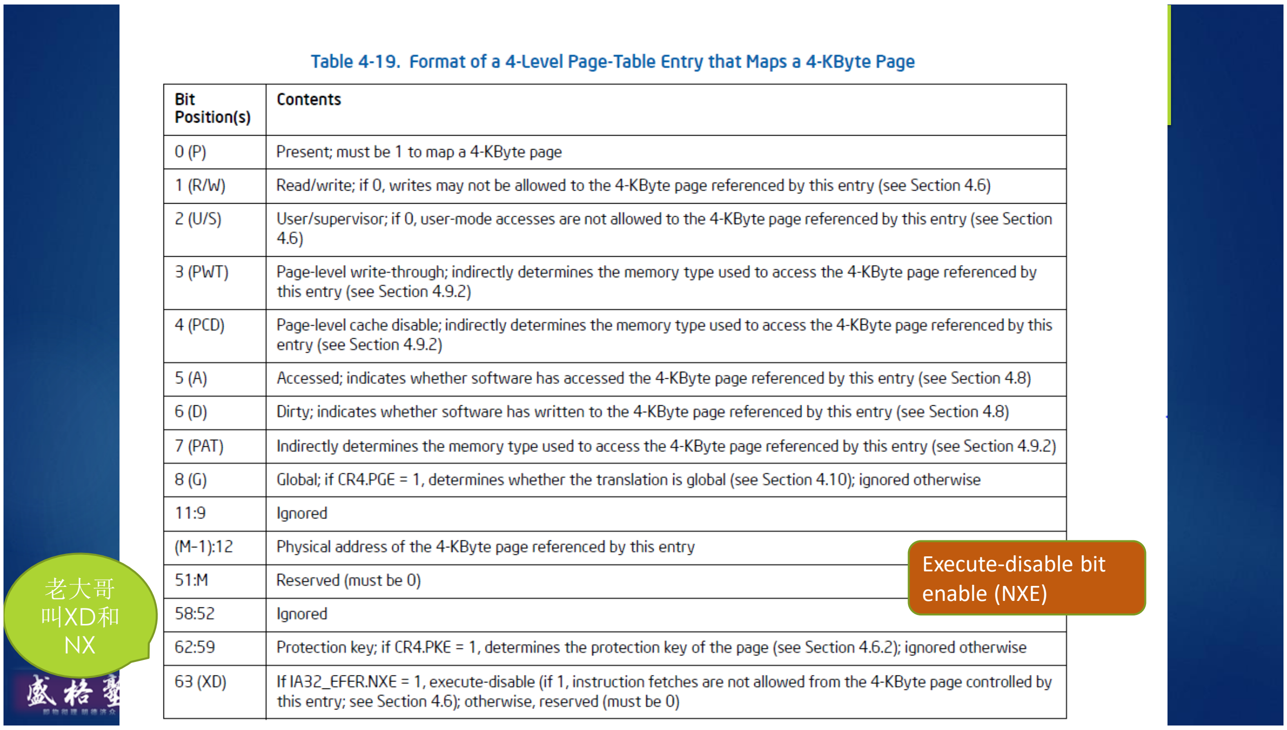
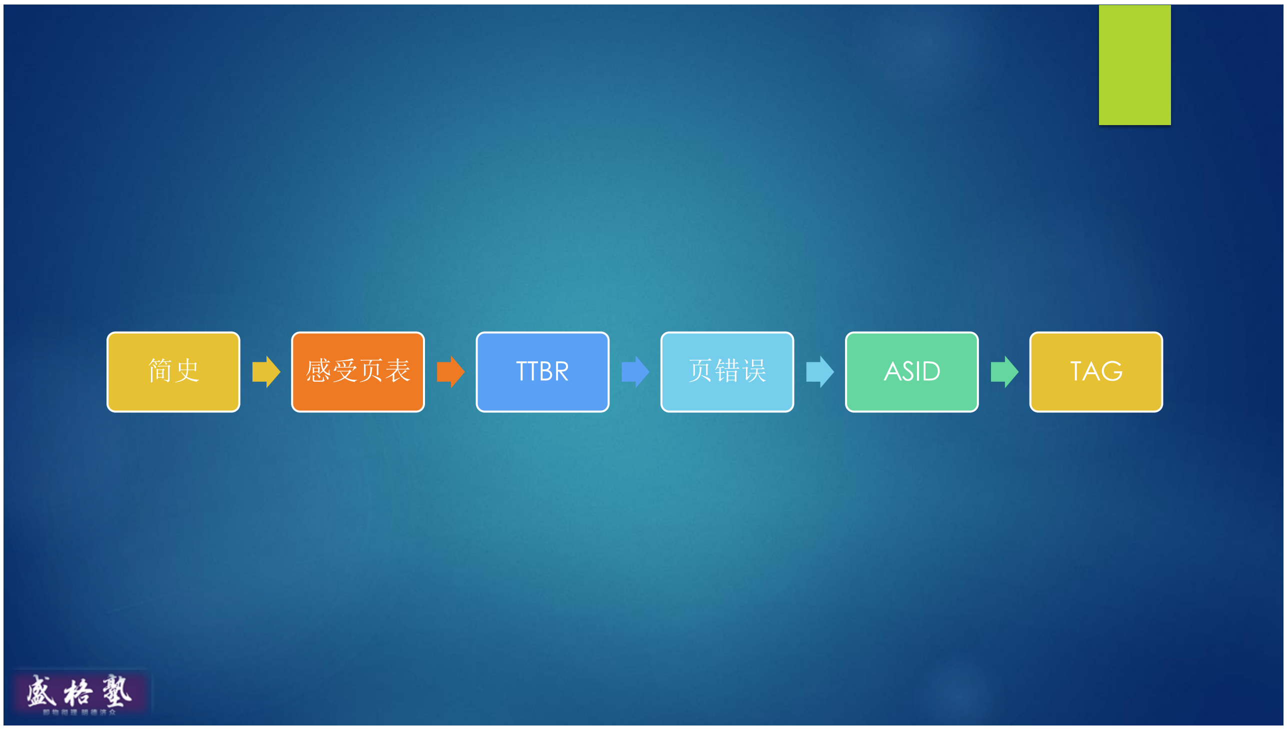


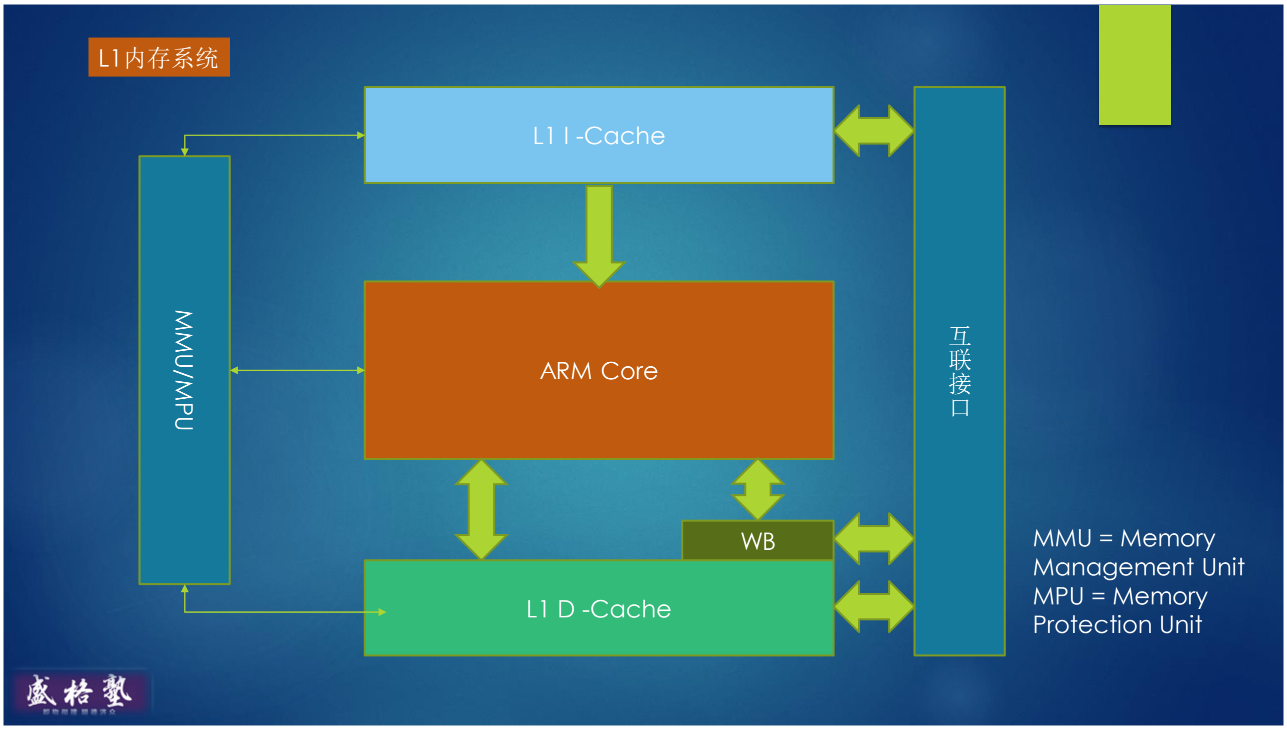
Table 4-19. Format of a 4-Level Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	Indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
(M-1):12	Physical address of the 4-KByte page referenced by this entry
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1, determines the protection key of the page (see Section 4.6.2); ignored otherwise
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Execute-disable bit enable (NXE)

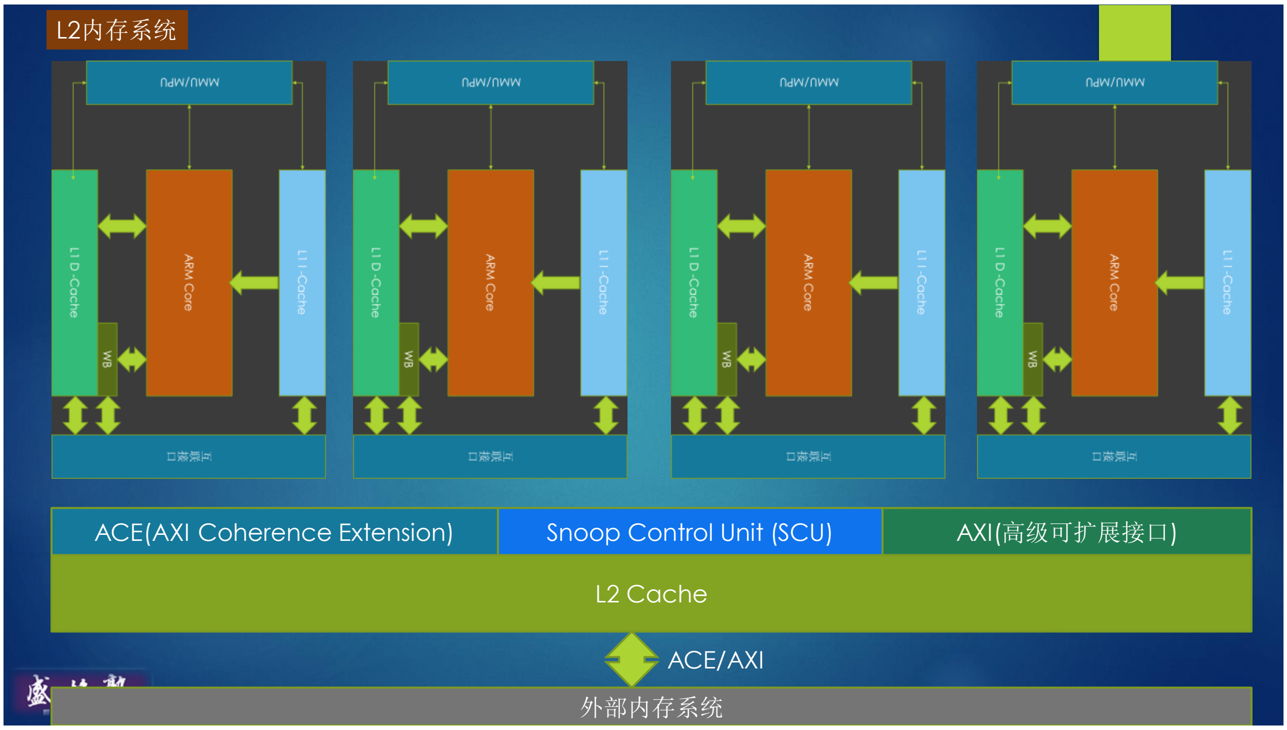
老大哥
叫XD和
NX

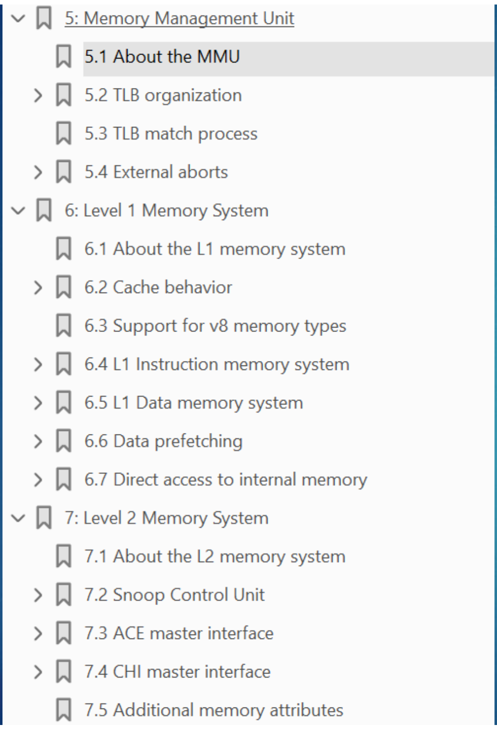




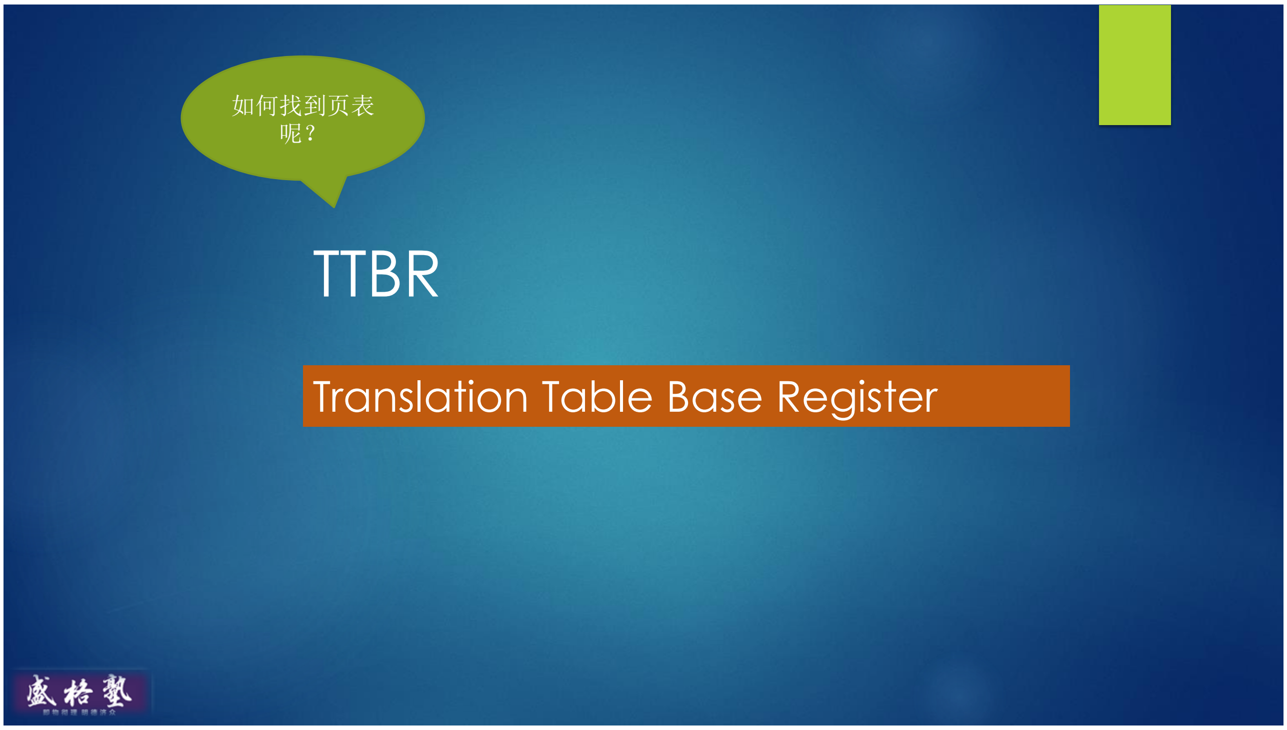
MMU = Memory Management Unit
MPU = Memory Protection Unit

L2内存系统





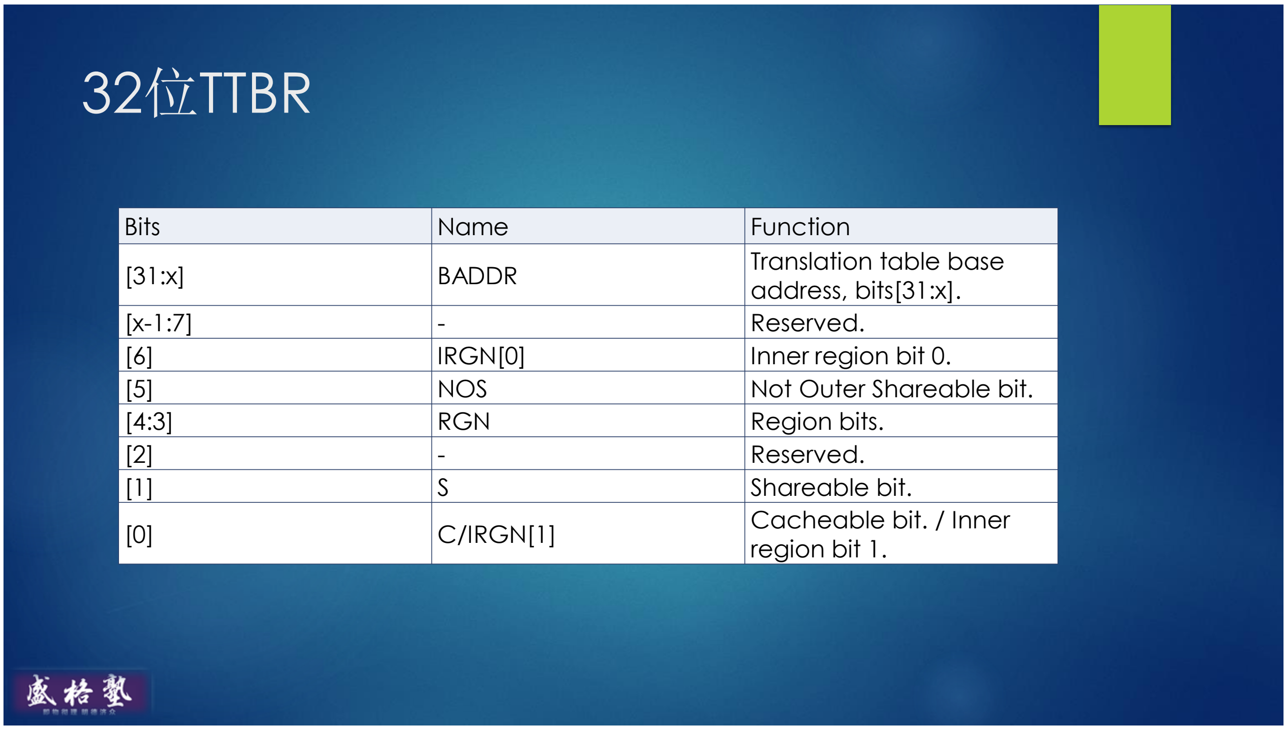
ARM® Cortex®-A53 MPCore Processor
Revision: r0p2
Technical Reference Manual



如何找到页表呢？

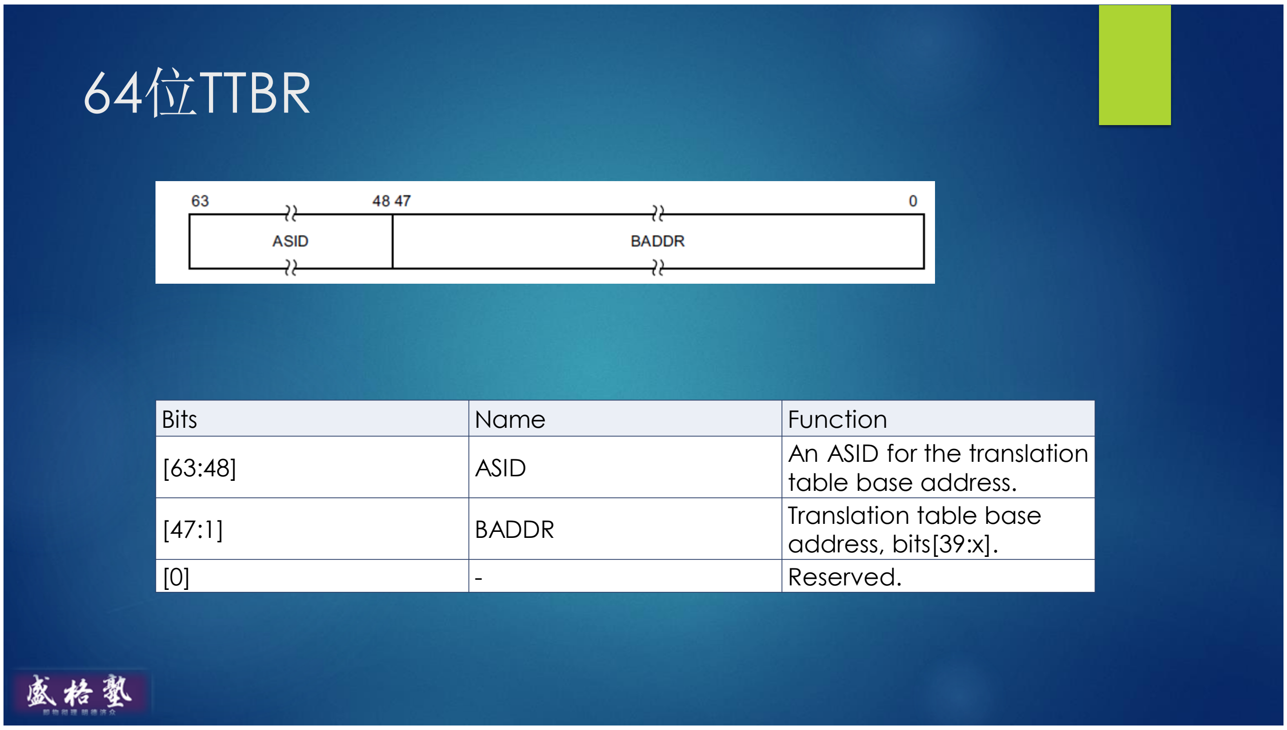
TTBR

Translation Table Base Register

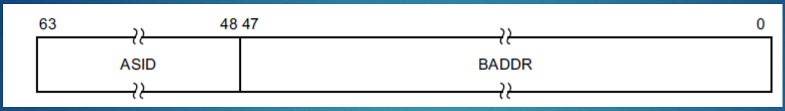


32位TTBR

Bits	Name	Function
[31:x]	BADDR	Translation table base address, bits[31:x].
[x-1:7]	-	Reserved.
[6]	IRGN[0]	Inner region bit 0.
[5]	NOS	Not Outer Shareable bit.
[4:3]	RGN	Region bits.
[2]	-	Reserved.
[1]	S	Shareable bit.
[0]	C/IRGN[1]	Cacheable bit. / Inner region bit 1.



64位TTBR



Bits	Name	Function
[63:48]	ASID	An ASID for the translation table base address.
[47:1]	BADDR	Translation table base address, bits[39:x].
[0]	-	Reserved.

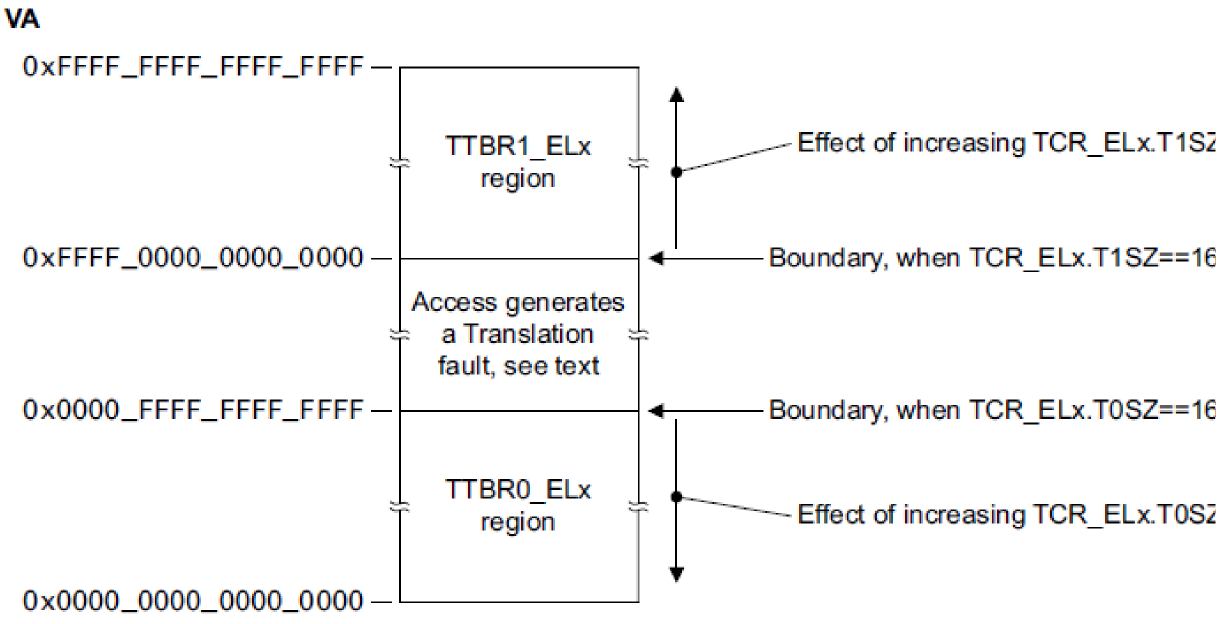
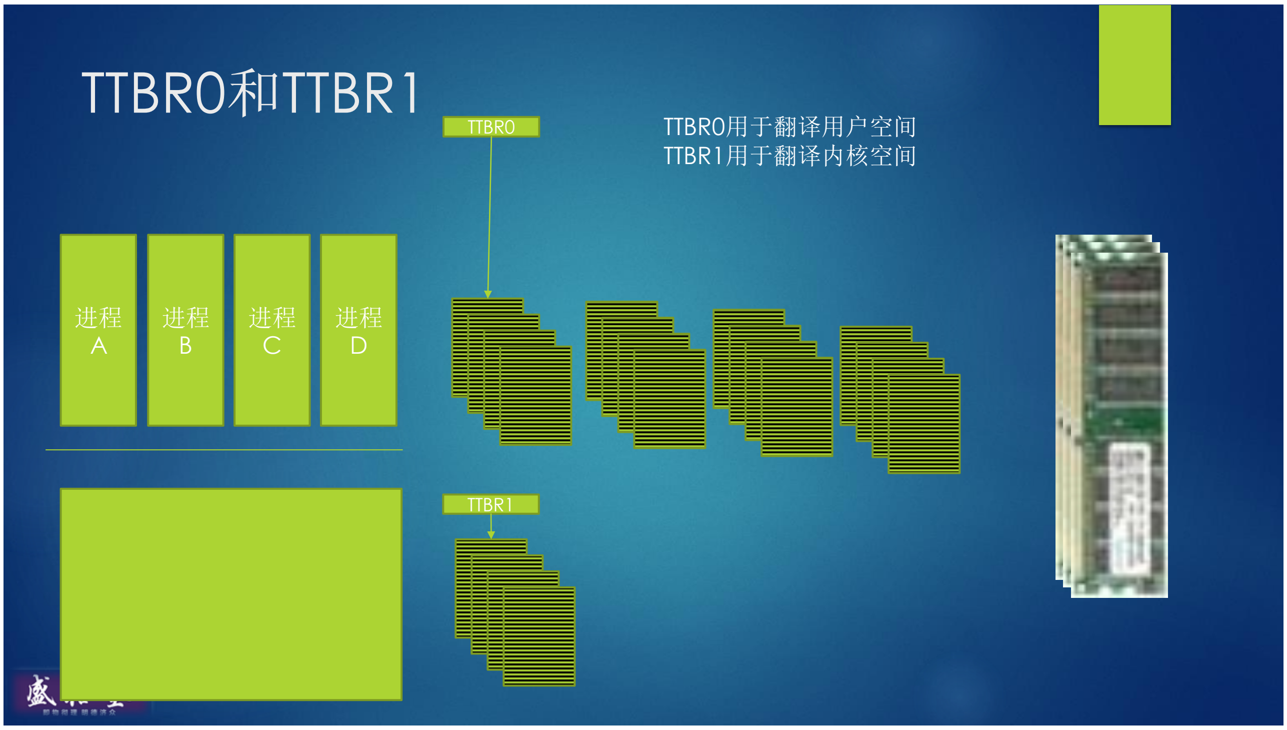
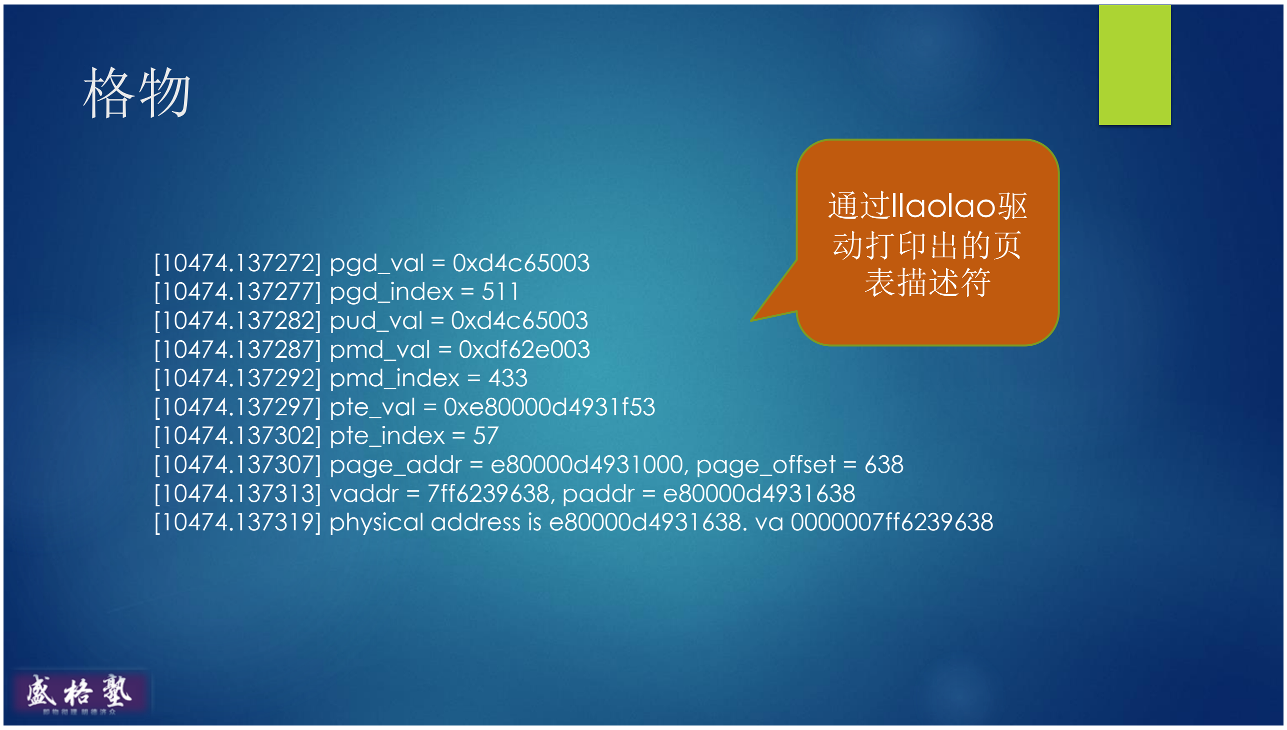

VA
0xFFFF_FFFF_FFFF_FFFF
0xFFFF_0000_0000_0000
0x0000_FFFF_FFFF_FFFF
0x0000_0000_0000_0000
TTBR1_ELx region
Access generates a Translation fault, see text
TTBR0_ELx region
Effect of increasing TCR_ELx.T1SZ
Boundary, when TCR_ELx.T1SZ==16
Boundary, when TCR_ELx.T0SZ==16
Effect of increasing TCR_ELx.T0SZ

Figure D4-14 AArch64 TTBR*n* boundaries and VA rang



TTBR0和TTBR1

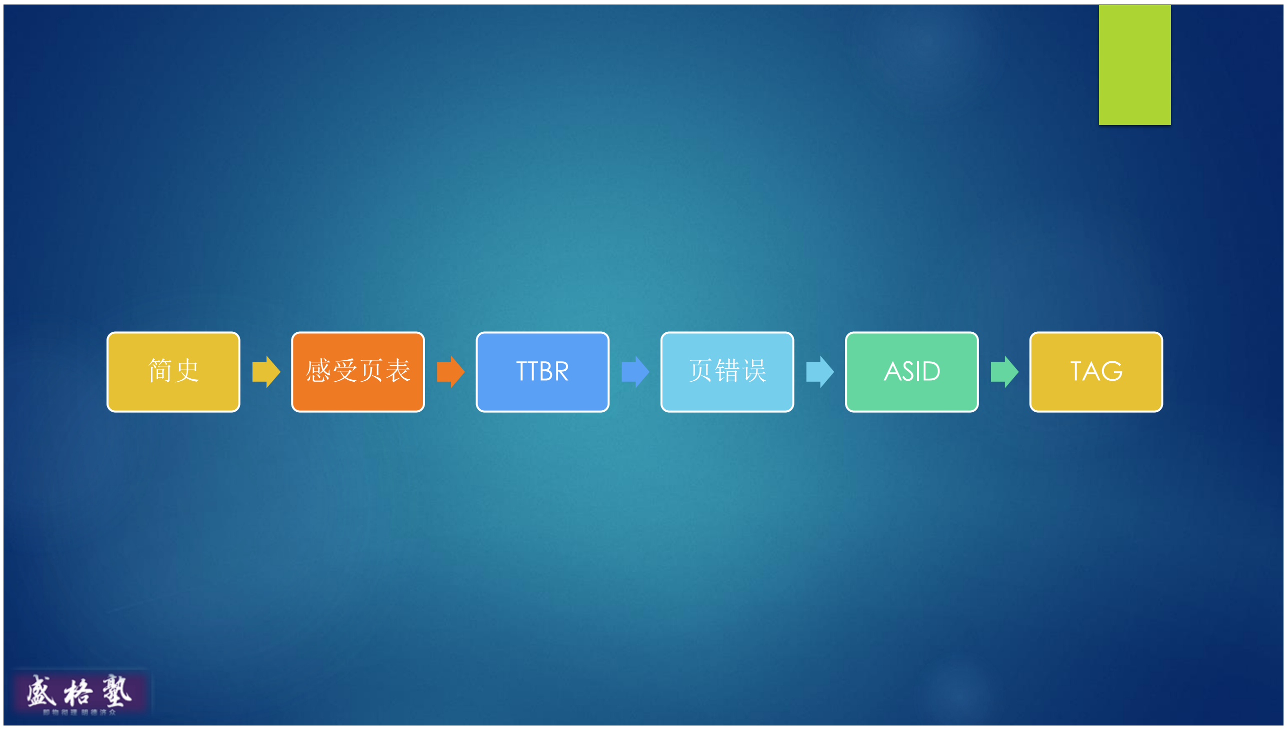
TTBR0用于翻译用户空间
TTBR1用于翻译内核空间

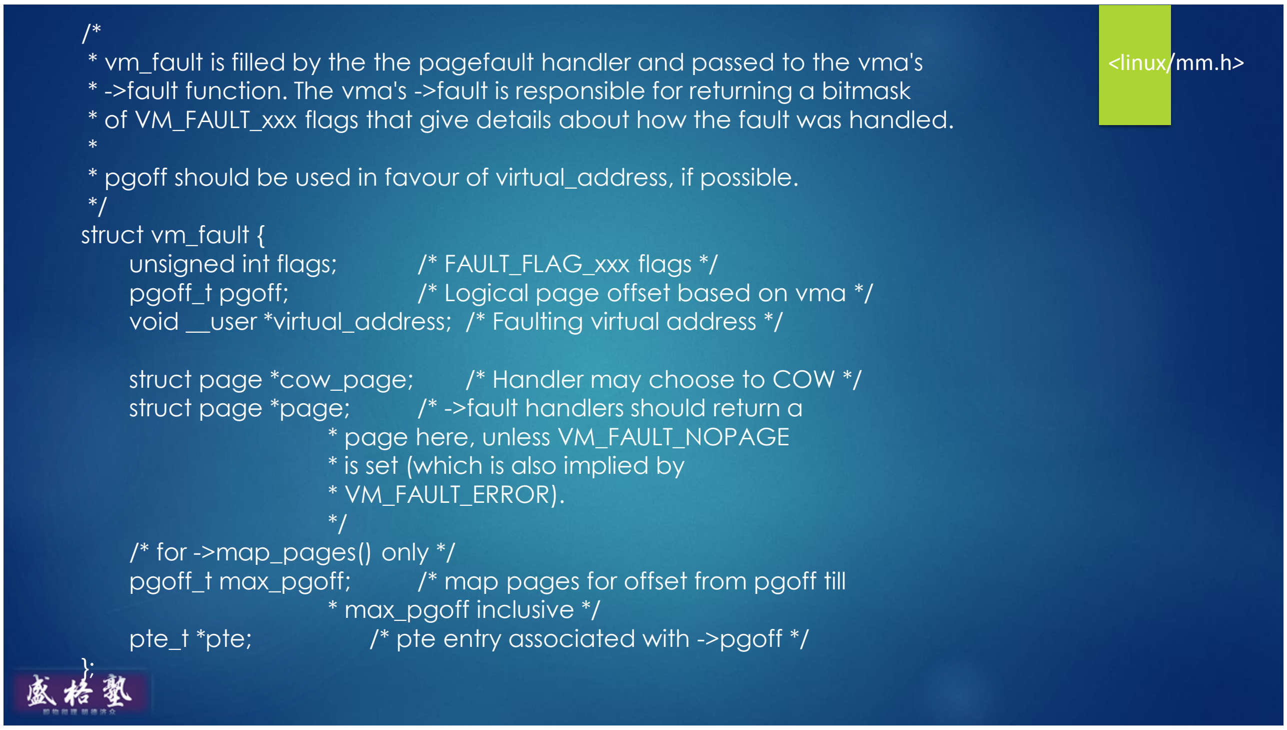


格物

```
[10474.137272] pgd_val = 0xd4c65003
[10474.137277] pgd_index = 511
[10474.137282] pud_val = 0xd4c65003
[10474.137287] pmd_val = 0xdf62e003
[10474.137292] pmd_index = 433
[10474.137297] pte_val = 0xe80000d4931f53
[10474.137302] pte_index = 57
[10474.137307] page_addr = e80000d4931000, page_offset = 638
[10474.137313] vaddr = 7ff6239638, paddr = e80000d4931638
[10474.137319] physical address is e80000d4931638. va 0000007ff6239638
```

通过llaolao驱动打印出的页表描述符

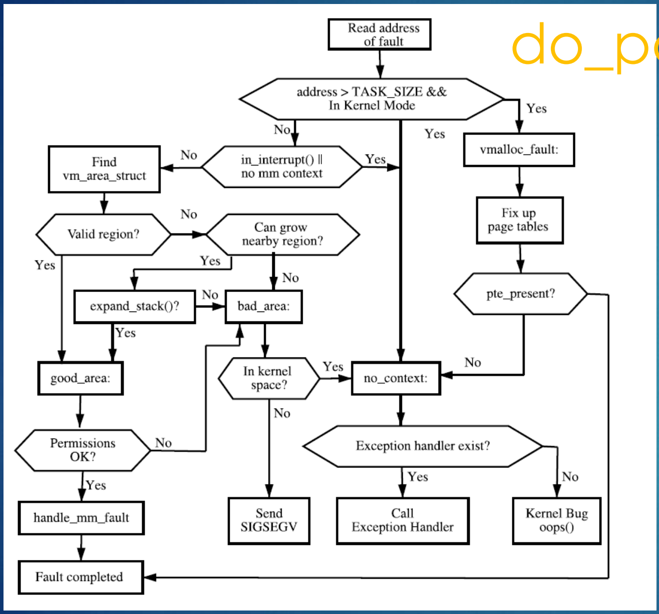




<linux/mm.h>

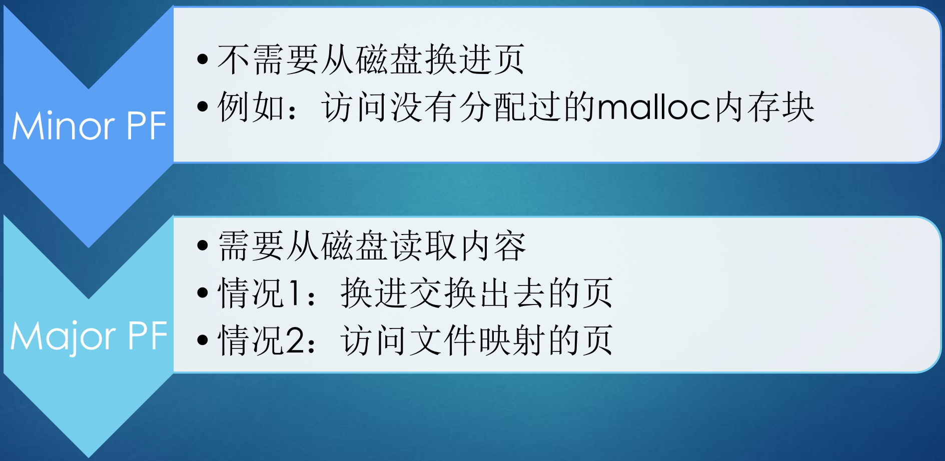
```
/*
 * vm_fault is filled by the the pagefault handler and passed to the vma's
 * ->fault function. The vma's ->fault is responsible for returning a bitmask
 * of VM_FAULT_xxx flags that give details about how the fault was handled.
 *
 * pgoff should be used in favour of virtual_address, if possible.
 */
struct vm_fault {
	unsigned int flags;		/* FAULT_FLAG_xxx flags */
	pgoff_t pgoff;			/* Logical page offset based on vma */
	void __user *virtual_address;	/* Faulting virtual address */

	struct page *cow_page;		/* Handler may choose to COW */
	struct page *page;		/* ->fault handlers should return a
					 * page here, unless VM_FAULT_NOPAGE
					 * is set (which is also implied by
					 * VM_FAULT_ERROR).
					 */
	/* for ->map_pages() only */
	pgoff_t max_pgoff;		/* map pages for offset from pgoff till
					 * max_pgoff inclusive */
	pte_t *pte;			/* pte entry associated with ->pgoff */
};
```

do_page_fault()



盛格塾

细分



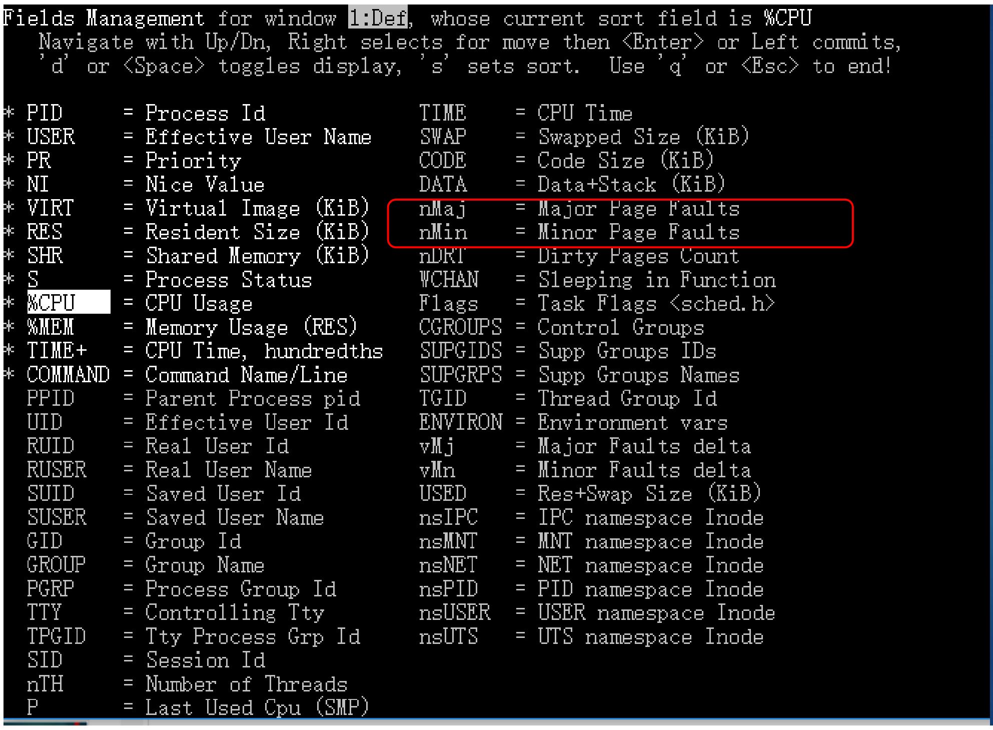
Minor PF

- 不需要从磁盘换进页
- 例如：访问没有分配过的malloc内存块

Major PF

- 需要从磁盘读取内容
- 情况1：换进交换出去的页
- 情况2：访问文件映射的页

盛格塾

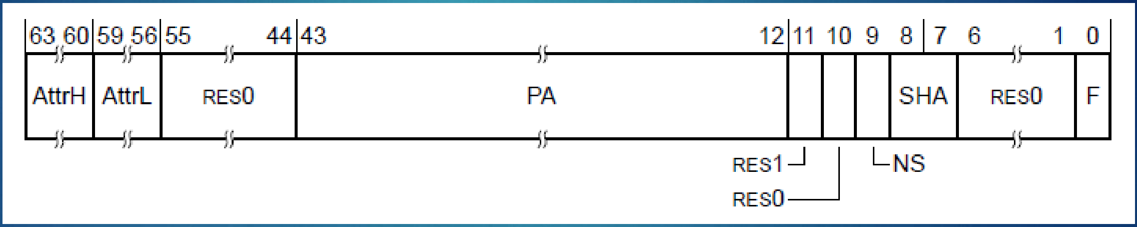


```
Fields Management for window 1:Def, whose current sort field is %CPU
   Navigate with Up/Dn, Right selects for move then <Enter> or Left commits,
   'd' or <Space> toggles display, 's' sets sort.  Use 'q' or <Esc> to end!

* PID     = Process Id            TIME    = CPU Time
* USER    = Effective User Name   SWAP    = Swapped Size (KiB)
* PR      = Priority              CODE    = Code Size (KiB)
* NI      = Nice Value            DATA    = Data+Stack (KiB)
* VIRT    = Virtual Image (KiB)   nMaj    = Major Page Faults
* RES     = Resident Size (KiB)   nMin    = Minor Page Faults
* SHR     = Shared Memory (KiB)   nDRT    = Dirty Pages Count
* S       = Process Status        WCHAN   = Sleeping in Function
* %CPU    = CPU Usage             Flags   = Task Flags <sched.h>
* %MEM    = Memory Usage (RES)    CGROUPS = Control Groups
* TIME+   = CPU Time, hundredths  SUPGIDS = Supp Groups IDs
* COMMAND = Command Name/Line     SUPGRPS = Supp Groups Names
  PPID    = Parent Process pid    TGID    = Thread Group Id
  UID     = Effective User Id     ENVIRON = Environment vars
  RUID    = Real User Id          vMj     = Major Faults delta
  RUSER   = Real User Name        vMn     = Minor Faults delta
  SUID    = Saved User Id         USED    = Res+Swap Size (KiB)
  SUSER   = Saved User Name       nsIPC   = IPC namespace Inode
  GID     = Group Id              nsMNT   = MNT namespace Inode
  GROUP   = Group Name            nsNET   = NET namespace Inode
  PGRP    = Process Group Id      nsPID   = PID namespace Inode
  TTY     = Controlling Tty       nsUSER  = USER namespace Inode
  TPGID   = Tty Process Grp Id    nsUTS   = UTS namespace Inode
  SID     = Session Id
  nTH     = Number of Threads
  P       = Last Used Cpu (SMP)
```

需要较高版本的TOP命令(WSL的可以)

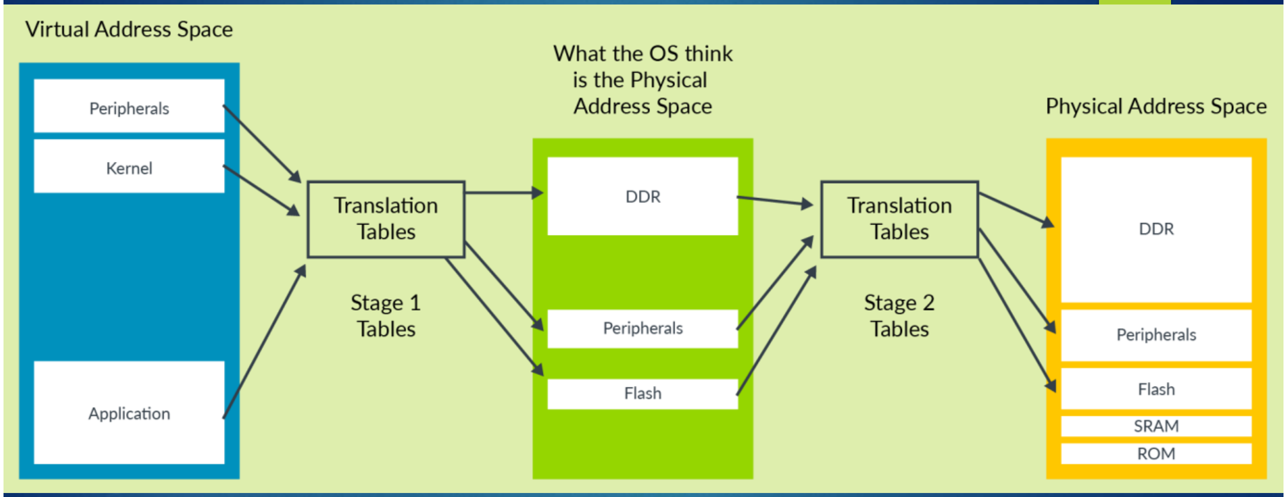
PAR_EL1

- **Physical Address Register, EL1**

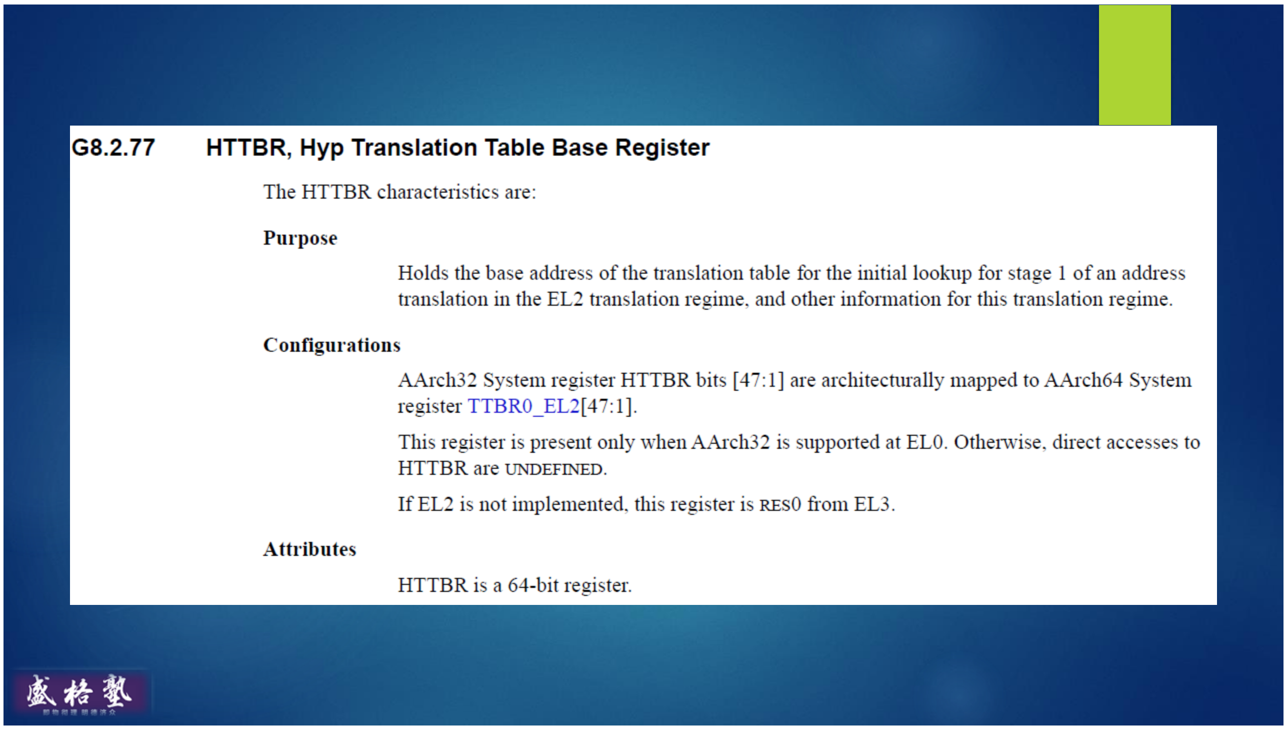


CPUMERRSR_EL1

- **CPU Memory Error Syndrome Register, EL1**
- Holds the number of memory errors that have occurred in the following L1 and L2 RAMs:
- • L1-I Tag RAM.
- • L1-I Data RAM.
- • L1-D Tag RAM.
- • L1-D Data RAM.
- • L2 TLB RAM.



https://developer.arm.com/documentation/102142/0100/Stage-2-translation



G8.2.77 HTTBR, Hyp Translation Table Base Register

The HTTBR characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of an address translation in the EL2 translation regime, and other information for this translation regime.

Configurations

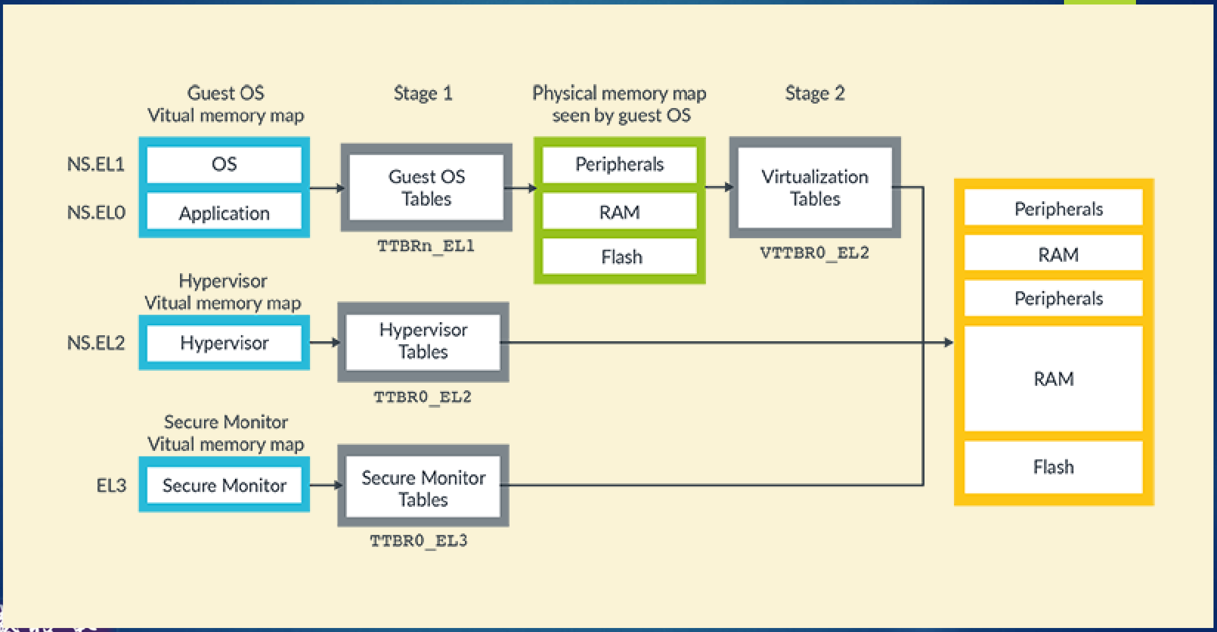
AArch32 System register HTTBR bits [47:1] are architecturally mapped to AArch64 System register TTBR0_EL2[47:1].

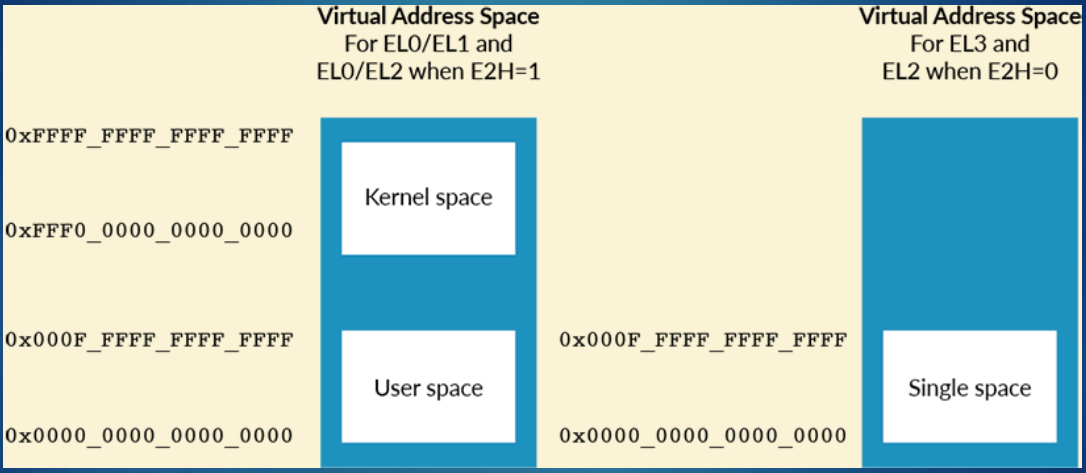
This register is present only when AArch32 is supported at EL0. Otherwise, direct accesses to HTTBR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HTTBR is a 64-bit register.

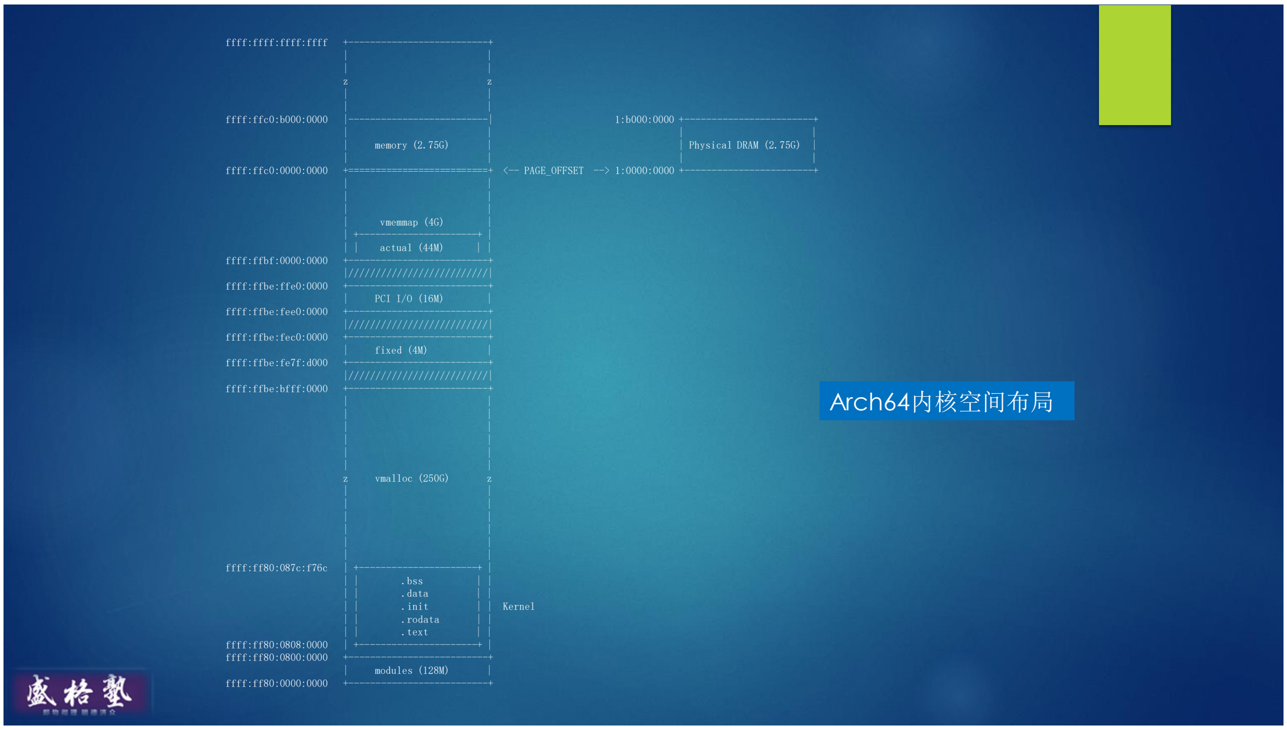




```
--------------------> +--------------------------+
ffff:ffff:ffff:ffff   |                          |
                      |                          |
ffff:ffc0:0000:0000   |----- Kernel (512G)-------|  <-- PAGE_OFFSET
                      |                          |
ffff:ff80:0000:0000   |                          |
--------------------> +--------------------------+
ffff:ff7f:ffff:ffff   |//////////////////////////|
                      |//////////////////////////|
                      |//////////////////////////|
                      |//////////////////////////|
                      |//////////////////////////|
                      |//////////////////////////|
                      z ///////////////////////// z
                      |//////////////////////////|
                      |//////////////////////////|
                      |//////////////////////////|
                      |//////////////////////////|
                      |//////////////////////////|
0000:0080:0000:0000   |//////////////////////////|
--------------------> +--------------------------+
0000:007f:ffff:ffff   |                          |
                      |                          |
                      |      User (512G)         |
                      |                          |
                      |                          |
0000:0000:0000:0000   |                          |
--------------------> +--------------------------+
```

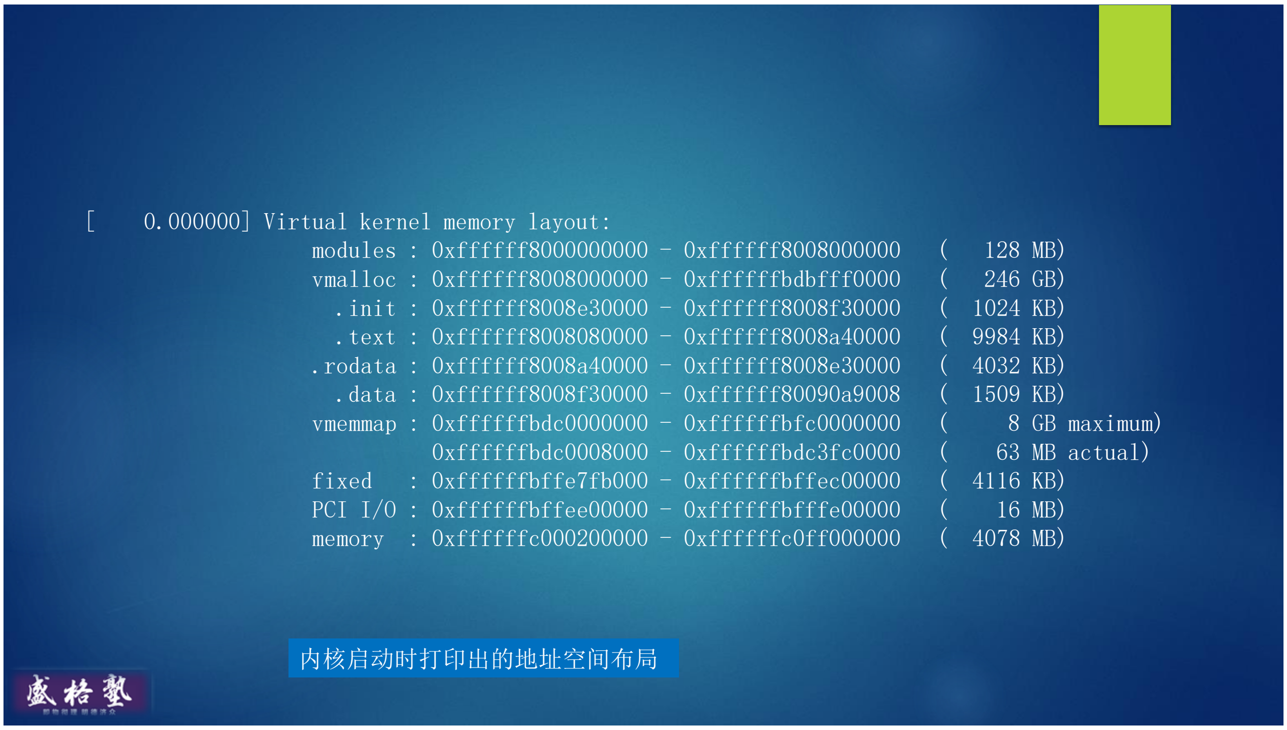
Arch64地址空间布局

CONFIG_ARM64_PAGE_SHIFT=12
CONFIG_PGTABLE_LEVELS=3
CONFIG_ARM64_VA_BITS=39



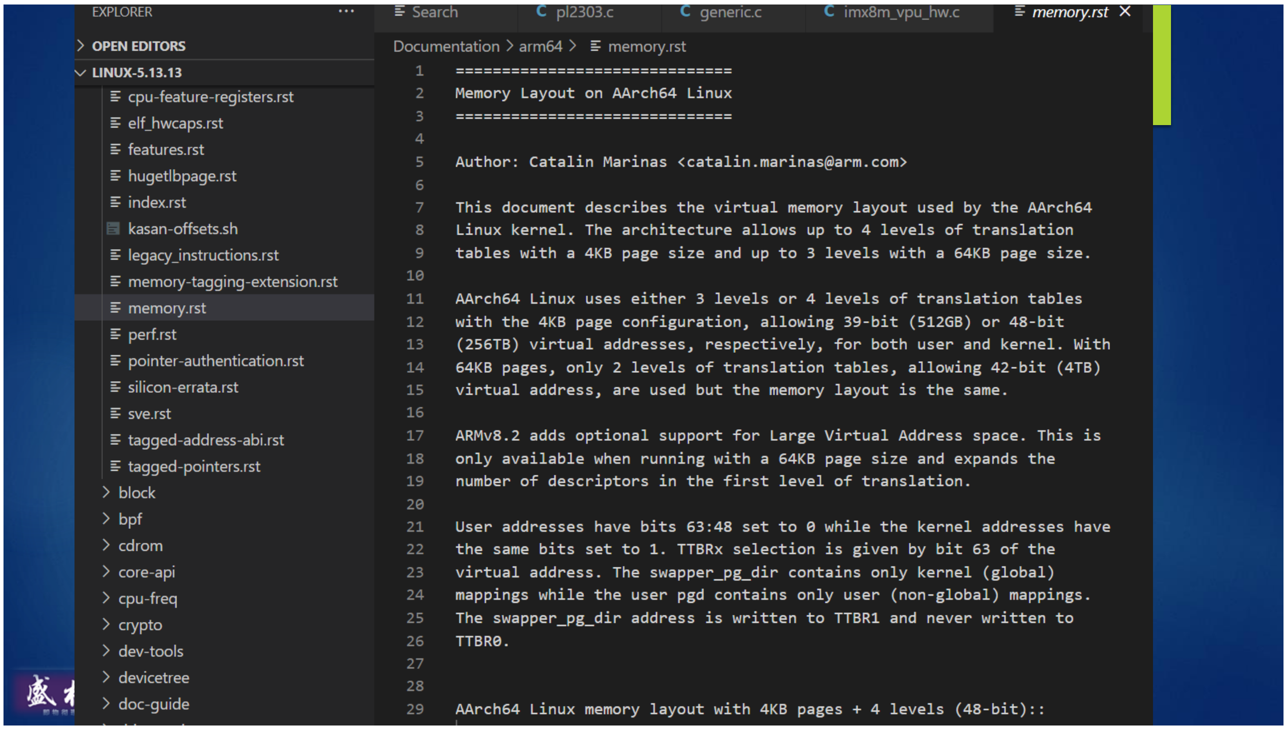
```
ffff:ffff:ffff:ffff   +--------------------------+
                      |                          |
                      |                          |
                      z                          z
                      |                          |
                      |                          |
ffff:ffc0:b000:0000   |--------------------------|                  1:b000:0000 +------------------------+
                      |                          |                              |                        |
                      |     memory (2.75G)       |                              | Physical DRAM (2.75G)  |
                      |                          |                              |                        |
ffff:ffc0:0000:0000   +==========================+  <-- PAGE_OFFSET  --> 1:0000:0000 +------------------------+
                      |                          |
                      |                          |
                      |                          |
                      |      vmemmap (4G)        |
                      | +----------------------+ |
                      | |    actual (44M)      | |
ffff:ffbf:0000:0000   +--------------------------+
                      |//////////////////////////|
ffff:ffbe:ffe0:0000   +--------------------------+
                      |     PCI I/O (16M)        |
ffff:ffbe:fee0:0000   +--------------------------+
                      |//////////////////////////|
ffff:ffbe:fec0:0000   +--------------------------+
                      |     fixed (4M)           |
ffff:ffbe:fe7f:d000   +--------------------------+
                      |//////////////////////////|
ffff:ffbe:bfff:0000   +--------------------------+
                      |                          |
                      |                          |
                      |                          |
                      |                          |
                      |                          |
                      |                          |
                      z     vmalloc (250G)       z
                      |                          |
                      |                          |
                      |                          |
                      |                          |
                      |                          |
                      |                          |
ffff:ff80:087c:f76c   | +----------------------+ |
                      | |        .bss          | |
                      | |        .data         | |
                      | |        .init         | |  Kernel
                      | |        .rodata       | |
                      | |        .text         | |
ffff:ff80:0808:0000   | +----------------------+ |
ffff:ff80:0800:0000   +--------------------------+
                      |     modules (128M)       |
ffff:ff80:0000:0000   +--------------------------+
```

Arch64内核空间布局

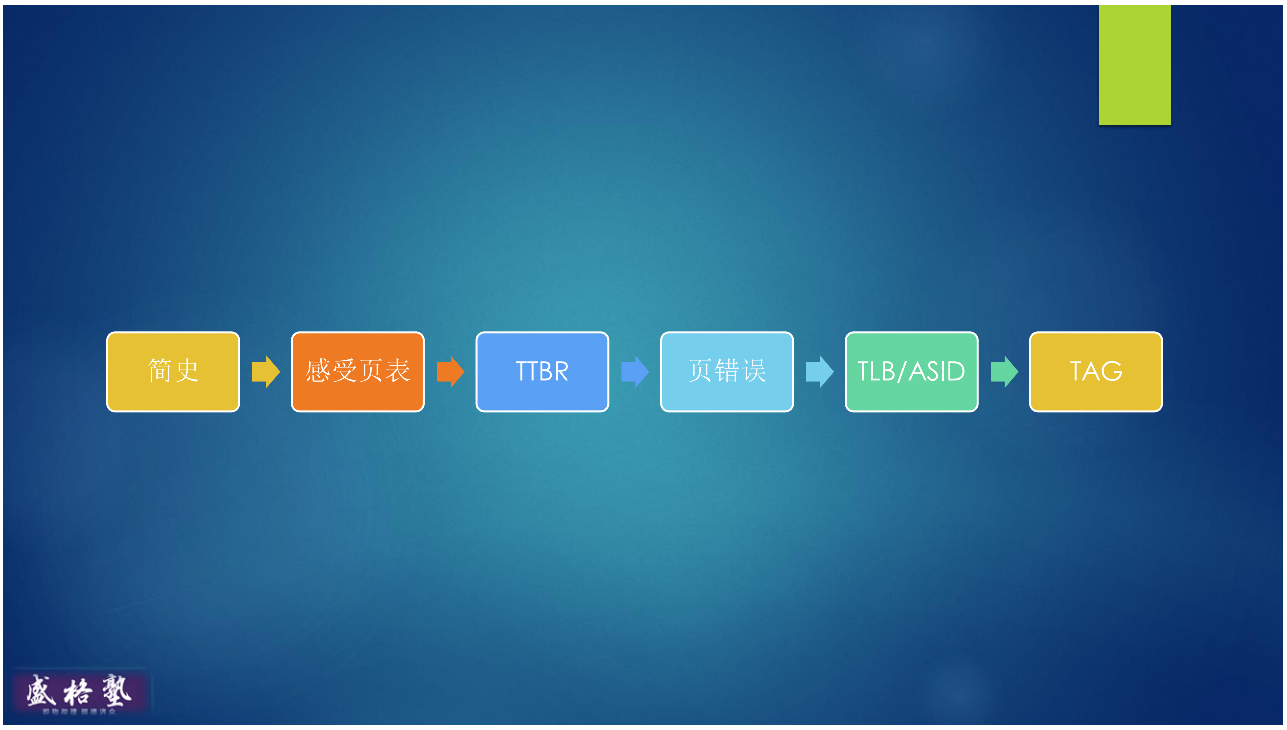


```
[    0.000000] Virtual kernel memory layout:
                    modules : 0xffffff8000000000 - 0xffffff8008000000   (   128 MB)
                    vmalloc : 0xffffff8008000000 - 0xffffffbdbfff0000   (   246 GB)
                      .init : 0xffffff8008e30000 - 0xffffff8008f30000   (  1024 KB)
                      .text : 0xffffff8008080000 - 0xffffff8008a40000   (  9984 KB)
                    .rodata : 0xffffff8008a40000 - 0xffffff8008e30000   (  4032 KB)
                      .data : 0xffffff8008f30000 - 0xffffff80090a9008   (  1509 KB)
                    vmemmap : 0xffffffbdc0000000 - 0xffffffbfc0000000   (     8 GB maximum)
                              0xffffffbdc0008000 - 0xffffffbdc3fc0000   (    63 MB actual)
                    fixed   : 0xffffffbffe7fb000 - 0xffffffbffec00000   (  4116 KB)
                    PCI I/O : 0xffffffbffee00000 - 0xffffffbfffe00000   (    16 MB)
                    memory  : 0xffffffc000200000 - 0xffffffc0ff000000   (  4078 MB)
```

内核启动时打印出的地址空间布局

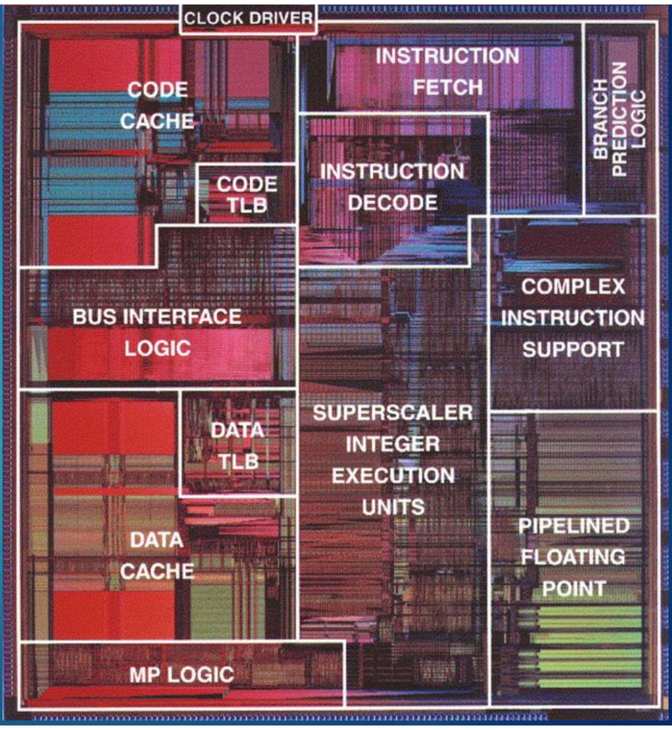


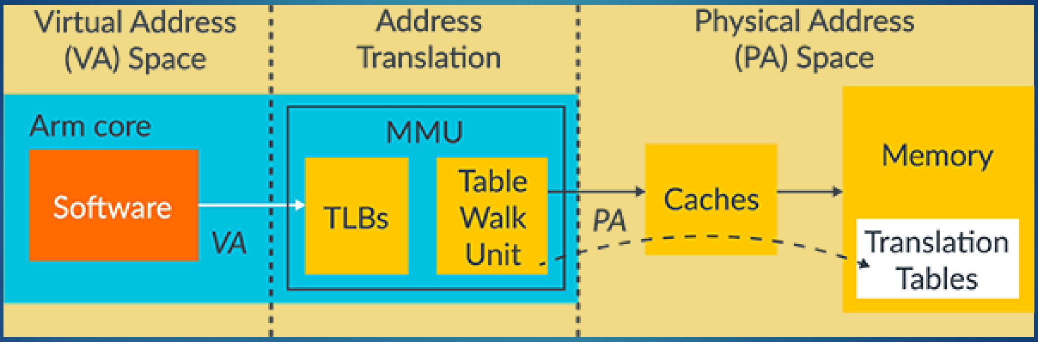
```
==============================
Memory Layout on AArch64 Linux
==============================

Author: Catalin Marinas <catalin.marinas@arm.com>

This document describes the virtual memory layout used by the AArch64
Linux kernel. The architecture allows up to 4 levels of translation
tables with a 4KB page size and up to 3 levels with a 64KB page size.

AArch64 Linux uses either 3 levels or 4 levels of translation tables
with the 4KB page configuration, allowing 39-bit (512GB) or 48-bit
(256TB) virtual addresses, respectively, for both user and kernel. With
64KB pages, only 2 levels of translation tables, allowing 42-bit (4TB)
virtual address, are used but the memory layout is the same.

ARMv8.2 adds optional support for Large Virtual Address space. This is
only available when running with a 64KB page size and expands the
number of descriptors in the first level of translation.

User addresses have bits 63:48 set to 0 while the kernel addresses have
the same bits set to 1. TTBRx selection is given by bit 63 of the
virtual address. The swapper_pg_dir contains only kernel (global)
mappings while the user pgd contains only user (non-global) mappings.
The swapper_pg_dir address is written to TTBR1 and never written to
TTBR0.


AArch64 Linux memory layout with 4KB pages + 4 levels (48-bit)::
```

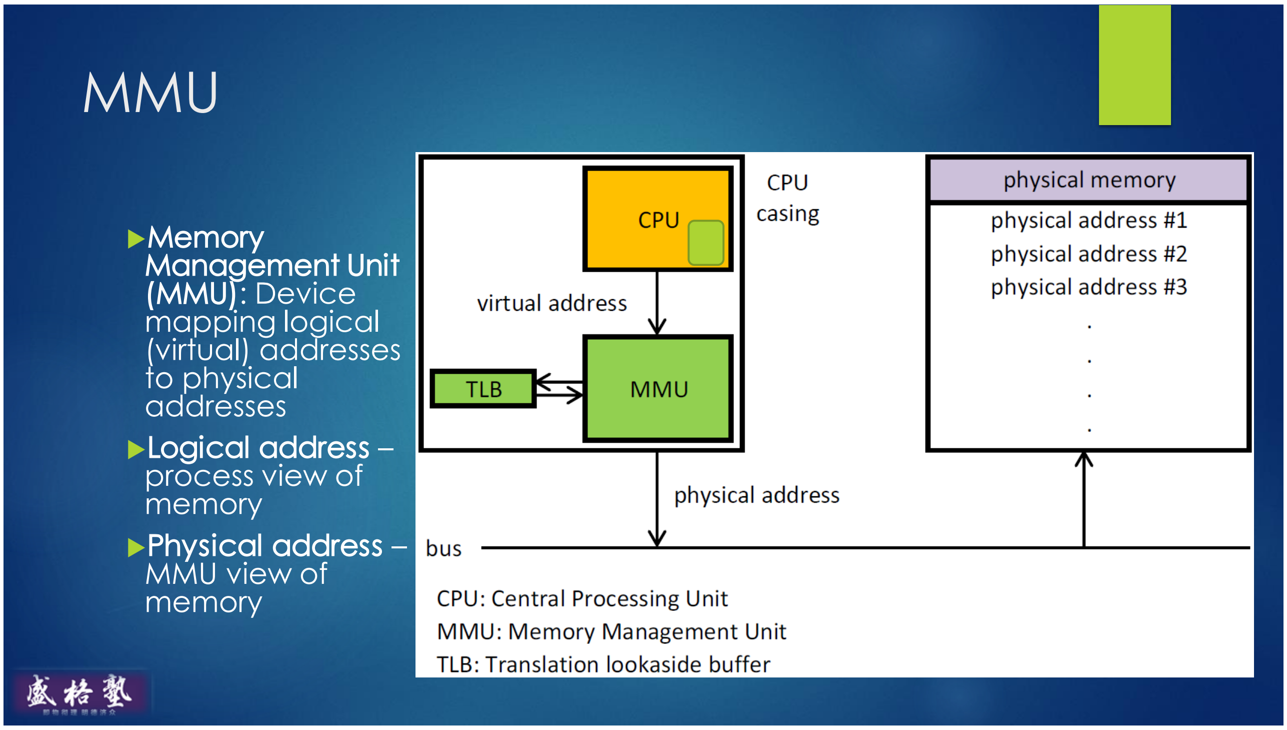


TLB

- Translation Lookaside Buffer
- a small associative memory that caches virtual to physical page table resolutions

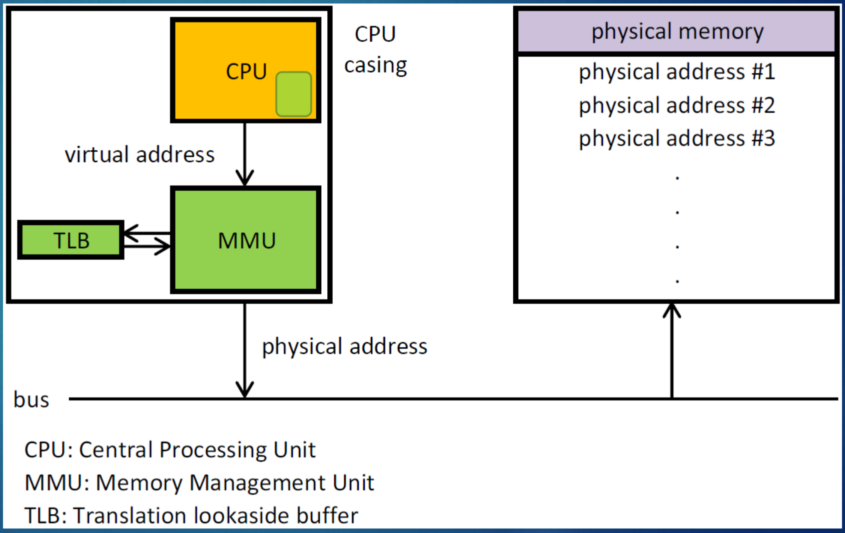




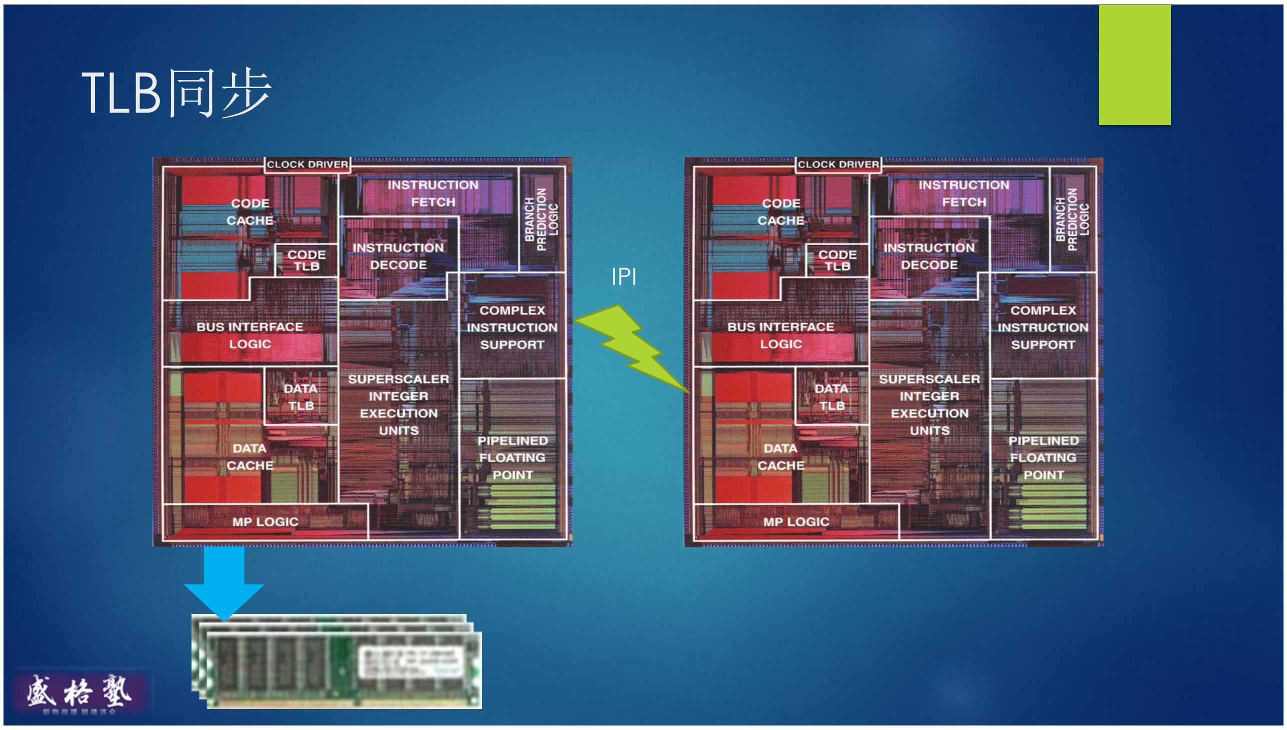


MMU

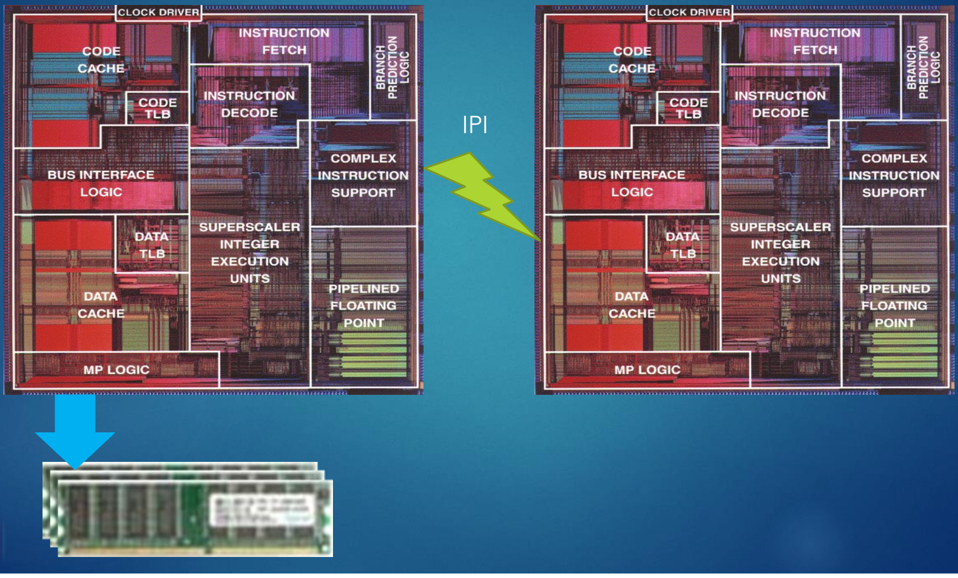
- Memory Management Unit (MMU): Device mapping logical (virtual) addresses to physical addresses
- Logical address – process view of memory
- Physical address – MMU view of memory

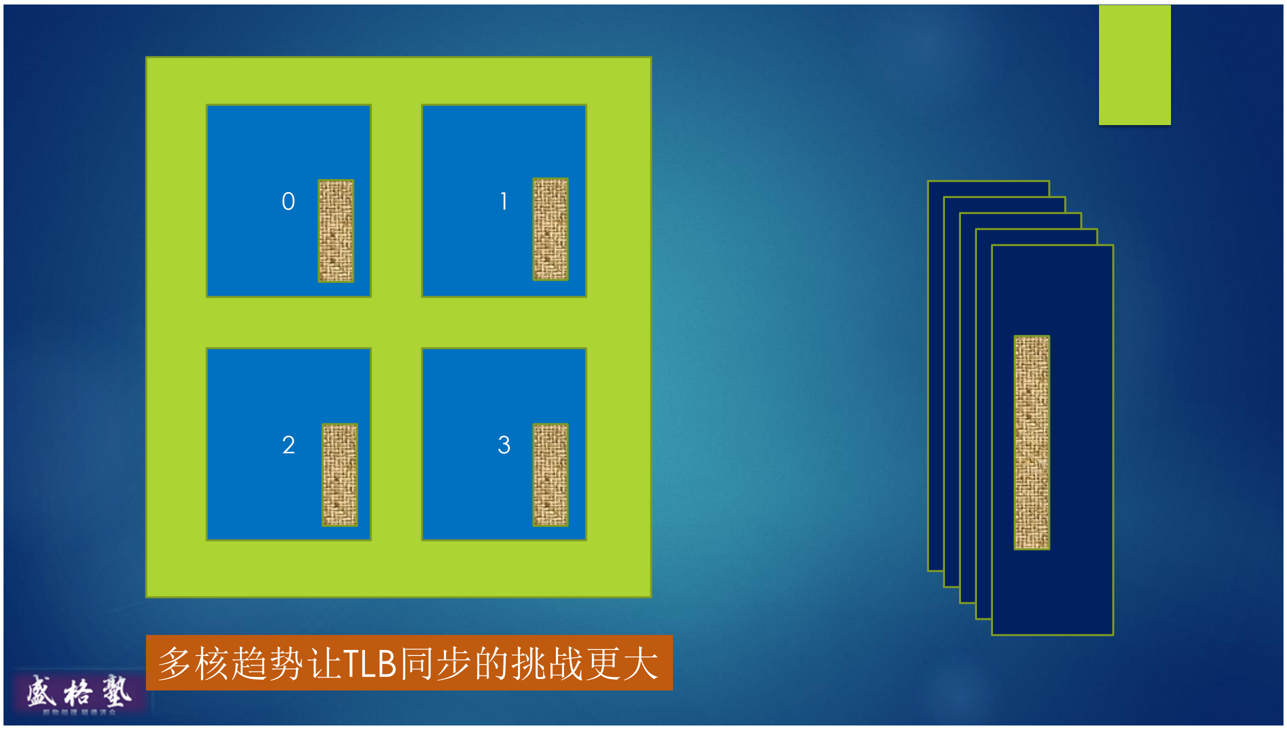


CPU: Central Processing Unit
MMU: Memory Management Unit
TLB: Translation lookaside buffer

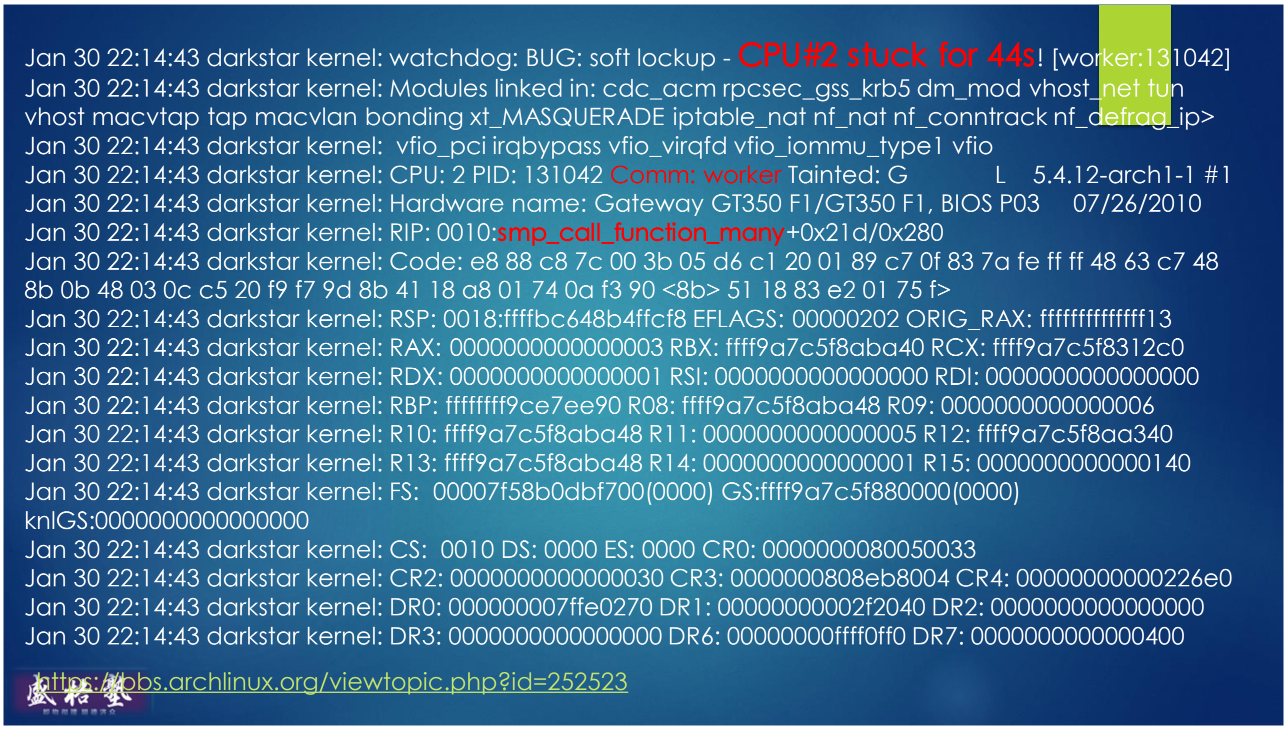


TLB同步



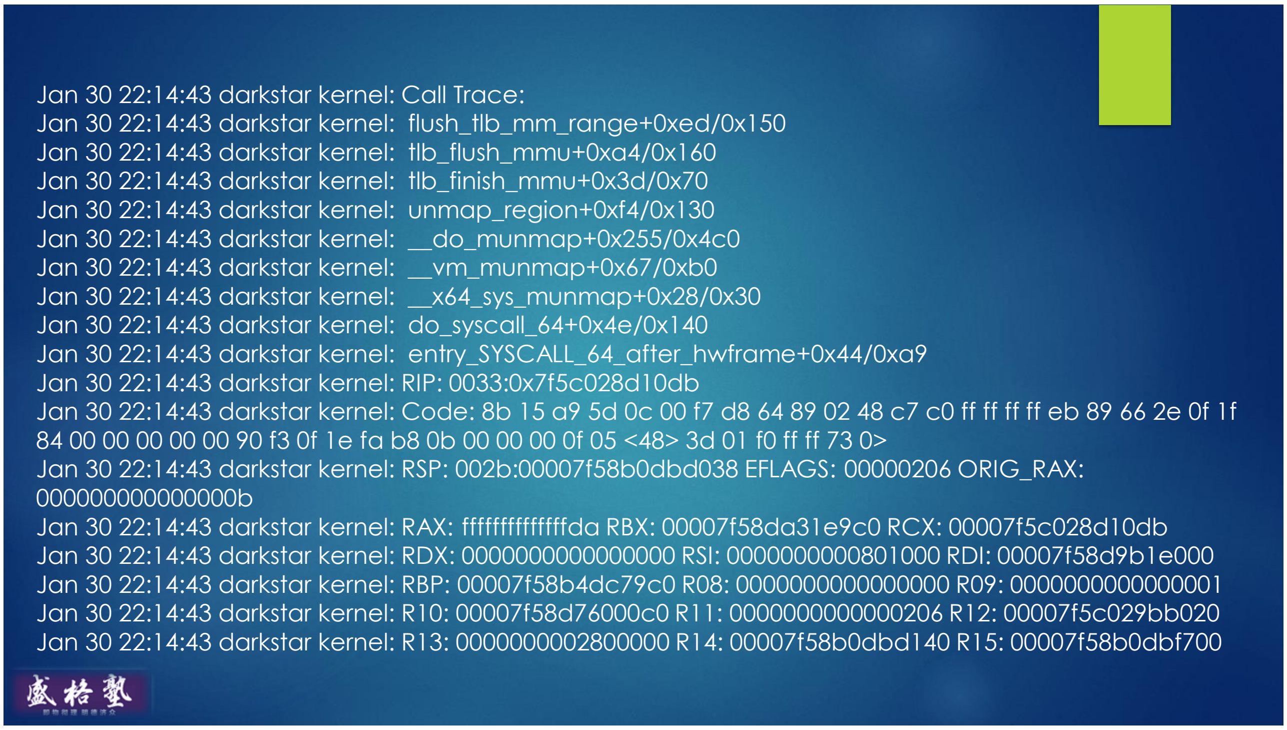


多核趋势让TLB同步的挑战更大

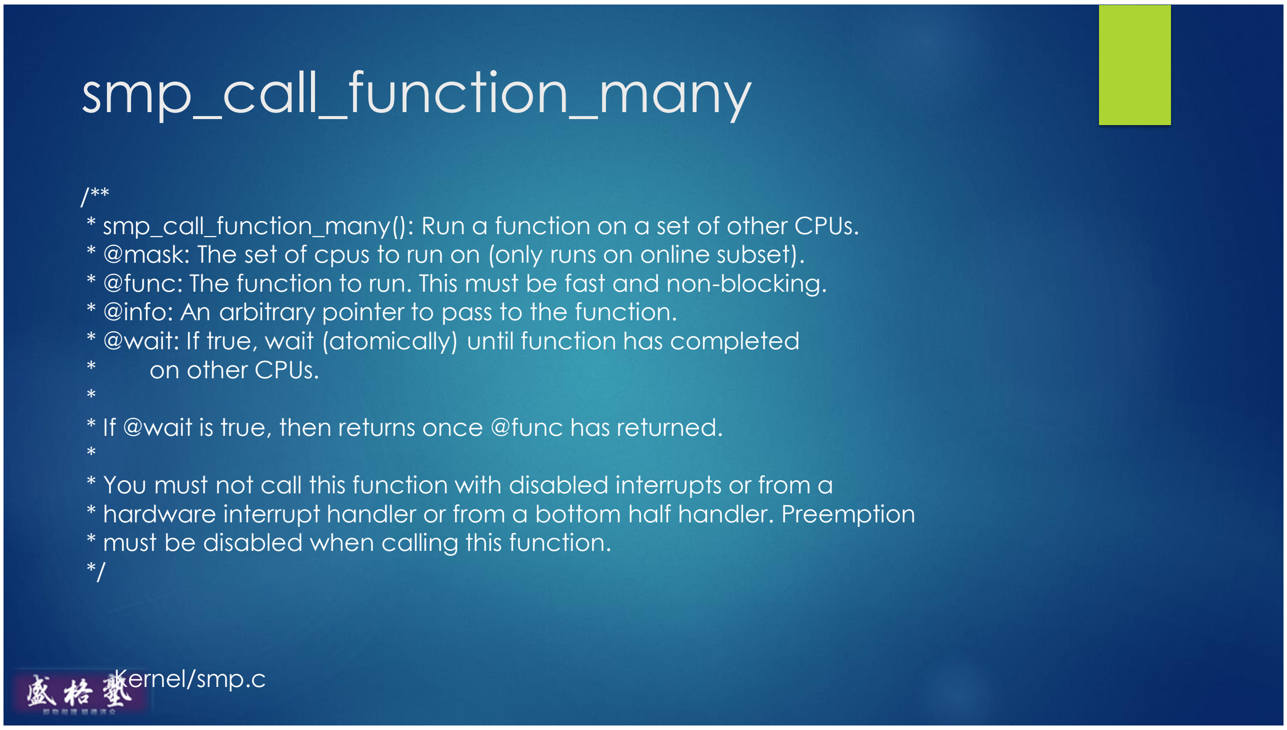


```
Jan 30 22:14:43 darkstar kernel: watchdog: BUG: soft lockup - CPU#2 stuck for 44s! [worker:131042]
Jan 30 22:14:43 darkstar kernel: Modules linked in: cdc_acm rpcsec_gss_krb5 dm_mod vhost_net tun
vhost macvtap tap macvlan bonding xt_MASQUERADE iptable_nat nf_nat nf_conntrack nf_defrag_ip>
Jan 30 22:14:43 darkstar kernel:  vfio_pci irqbypass vfio_virqfd vfio_iommu_type1 vfio
Jan 30 22:14:43 darkstar kernel: CPU: 2 PID: 131042 Comm: worker Tainted: G            L    5.4.12-arch1-1 #1
Jan 30 22:14:43 darkstar kernel: Hardware name: Gateway GT350 F1/GT350 F1, BIOS P03     07/26/2010
Jan 30 22:14:43 darkstar kernel: RIP: 0010:smp_call_function_many+0x21d/0x280
Jan 30 22:14:43 darkstar kernel: Code: e8 88 c8 7c 00 3b 05 d6 c1 20 01 89 c7 0f 83 7a fe ff ff 48 63 c7 48
8b 0b 48 03 0c c5 20 f9 f7 9d 8b 41 18 a8 01 74 0a f3 90 <8b> 51 18 83 e2 01 75 f>
Jan 30 22:14:43 darkstar kernel: RSP: 0018:ffffbc648b4ffcf8 EFLAGS: 00000202 ORIG_RAX: ffffffffffffff13
Jan 30 22:14:43 darkstar kernel: RAX: 0000000000000003 RBX: ffff9a7c5f8aba40 RCX: ffff9a7c5f8312c0
Jan 30 22:14:43 darkstar kernel: RDX: 0000000000000001 RSI: 0000000000000000 RDI: 0000000000000000
Jan 30 22:14:43 darkstar kernel: RBP: ffffffff9ce7ee90 R08: ffff9a7c5f8aba48 R09: 0000000000000006
Jan 30 22:14:43 darkstar kernel: R10: ffff9a7c5f8aba48 R11: 0000000000000005 R12: ffff9a7c5f8aa340
Jan 30 22:14:43 darkstar kernel: R13: ffff9a7c5f8aba48 R14: 0000000000000001 R15: 0000000000000140
Jan 30 22:14:43 darkstar kernel: FS:  00007f58b0dbf700(0000) GS:ffff9a7c5f880000(0000)
knlGS:0000000000000000
Jan 30 22:14:43 darkstar kernel: CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
Jan 30 22:14:43 darkstar kernel: CR2: 0000000000000030 CR3: 0000000808eb8004 CR4: 00000000000226e0
Jan 30 22:14:43 darkstar kernel: DR0: 000000007ffe0270 DR1: 00000000002f2040 DR2: 0000000000000000
Jan 30 22:14:43 darkstar kernel: DR3: 0000000000000000 DR6: 00000000ffff0ff0 DR7: 0000000000000400
```

https://bbs.archlinux.org/viewtopic.php?id=252523



```
Jan 30 22:14:43 darkstar kernel: Call Trace:
Jan 30 22:14:43 darkstar kernel:  flush_tlb_mm_range+0xed/0x150
Jan 30 22:14:43 darkstar kernel:  tlb_flush_mmu+0xa4/0x160
Jan 30 22:14:43 darkstar kernel:  tlb_finish_mmu+0x3d/0x70
Jan 30 22:14:43 darkstar kernel:  unmap_region+0xf4/0x130
Jan 30 22:14:43 darkstar kernel:  __do_munmap+0x255/0x4c0
Jan 30 22:14:43 darkstar kernel:  __vm_munmap+0x67/0xb0
Jan 30 22:14:43 darkstar kernel:  __x64_sys_munmap+0x28/0x30
Jan 30 22:14:43 darkstar kernel:  do_syscall_64+0x4e/0x140
Jan 30 22:14:43 darkstar kernel:  entry_SYSCALL_64_after_hwframe+0x44/0xa9
Jan 30 22:14:43 darkstar kernel: RIP: 0033:0x7f5c028d10db
Jan 30 22:14:43 darkstar kernel: Code: 8b 15 a9 5d 0c 00 f7 d8 64 89 02 48 c7 c0 ff ff ff ff eb 89 66 2e 0f 1f
84 00 00 00 00 00 90 f3 0f 1e fa b8 0b 00 00 00 0f 05 <48> 3d 01 f0 ff ff 73 0>
Jan 30 22:14:43 darkstar kernel: RSP: 002b:00007f58b0dbd038 EFLAGS: 00000206 ORIG_RAX:
000000000000000b
Jan 30 22:14:43 darkstar kernel: RAX: ffffffffffffffda RBX: 00007f58da31e9c0 RCX: 00007f5c028d10db
Jan 30 22:14:43 darkstar kernel: RDX: 0000000000000000 RSI: 0000000000801000 RDI: 00007f58d9b1e000
Jan 30 22:14:43 darkstar kernel: RBP: 00007f58b4dc79c0 R08: 0000000000000000 R09: 0000000000000001
Jan 30 22:14:43 darkstar kernel: R10: 00007f58d76000c0 R11: 0000000000000206 R12: 00007f5c029bb020
Jan 30 22:14:43 darkstar kernel: R13: 0000000002800000 R14: 00007f58b0dbd140 R15: 00007f58b0dbf700
```



smp_call_function_many

```
/**
 * smp_call_function_many(): Run a function on a set of other CPUs.
 * @mask: The set of cpus to run on (only runs on online subset).
 * @func: The function to run. This must be fast and non-blocking.
 * @info: An arbitrary pointer to pass to the function.
 * @wait: If true, wait (atomically) until function has completed
 *        on other CPUs.
 *
 * If @wait is true, then returns once @func has returned.
 *
 * You must not call this function with disabled interrupts or from a
 * hardware interrupt handler or from a bottom half handler. Preemption
 * must be disabled when calling this function.
 */
```

kernel/smp.c

循环等待

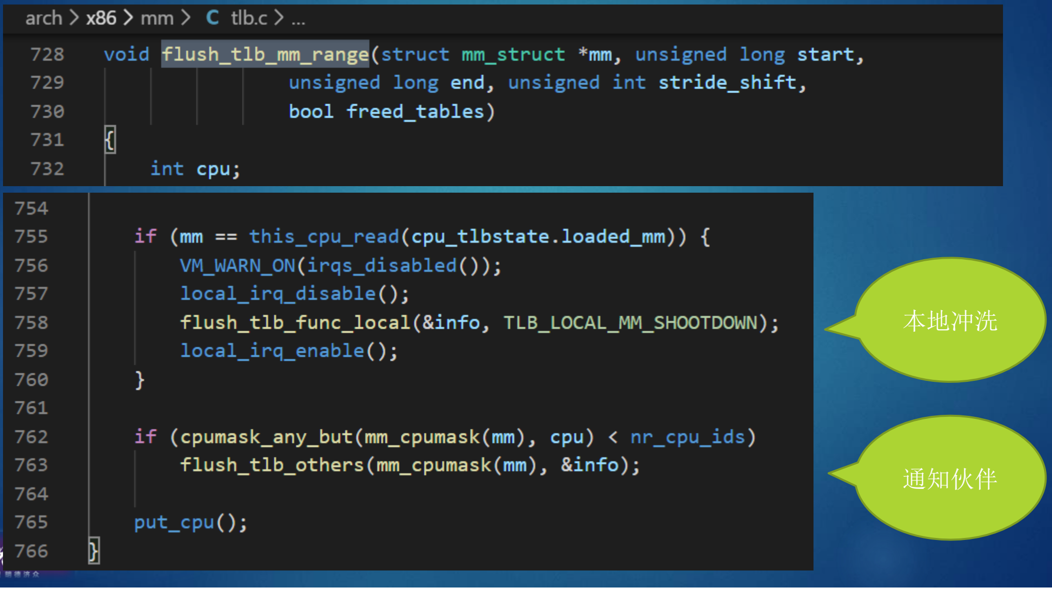
```
/* Send a message to all CPUs in the map */
arch_send_call_function_ipi_mask(cfd->cpumask_ipi);

if (wait) {
        for_each_cpu(cpu, cfd->cpumask) {
                call_single_data_t *csd;

                csd = per_cpu_ptr(cfd->csd, cpu);
                csd_lock_wait(csd);
        }
}
```

call_single_data_t

```
/**
 * smp_cond_load_relaxed() - (Spin) wait for cond with no ordering guarantees
 * @ptr: pointer to the variable to wait on
 * @cond: boolean expression to wait for
 *
 * Equivalent to using READ_ONCE() on the condition variable.
 *
 * Due to C lacking lambda expressions we load the value of *ptr into a
 * pre-named variable @VAL to be used in @cond.
 */
#ifndef smp_cond_load_relaxed
#define smp_cond_load_relaxed(ptr, cond_expr) ({          \
    typeof(ptr) __PTR = (ptr);                   \
    typeof(*ptr) VAL;                        \
    for (;;) {                           \
        VAL = READ_ONCE(*__PTR);                 \
        if (cond_expr)                       \
            break;                       \
        cpu_relax();                         \
    }                                    \
    VAL;                                 \
})
#endif
```

忙等

flush_tlb_mm_range



```c
void flush_tlb_mm_range(struct mm_struct *mm, unsigned long start,
                        unsigned long end, unsigned int stride_shift,
                        bool freed_tables)
{
    int cpu;
```

```c

    if (mm == this_cpu_read(cpu_tlbstate.loaded_mm)) {
        VM_WARN_ON(irqs_disabled());
        local_irq_disable();
        flush_tlb_func_local(&info, TLB_LOCAL_MM_SHOOTDOWN);
        local_irq_enable();
    }

    if (cpumask_any_but(mm_cpumask(mm), cpu) < nr_cpu_ids)
        flush_tlb_others(mm_cpumask(mm), &info);

    put_cpu();
}
```

2020-01-30 20:26:00
Gruntz
Member From: Haskovo, Bulgaria
Registered: 2007-08-31 Posts: 285
Hello all,

I have a supemicro motherboard and two xeon x5650. I have 64GB of ram and several VMs on it.

I have one windows10, that i use for gaming (with gpu pass-through)
I have another, with archlinux for work (soft dev, testings... i use this one as my main desktop.GPU pass-through here too.)
I have 3-4-5 more, for database server, gitlab server, stuff like that.

From time to time my host machine crashes. It has a lot of "watchdog: BUG: soft lockup - CPU#4 stuck for 45s! [worker:131043]" messages.
Each time different processor. After it appears, the computer slowly becomes unresponsive, and eventually hangs completely.

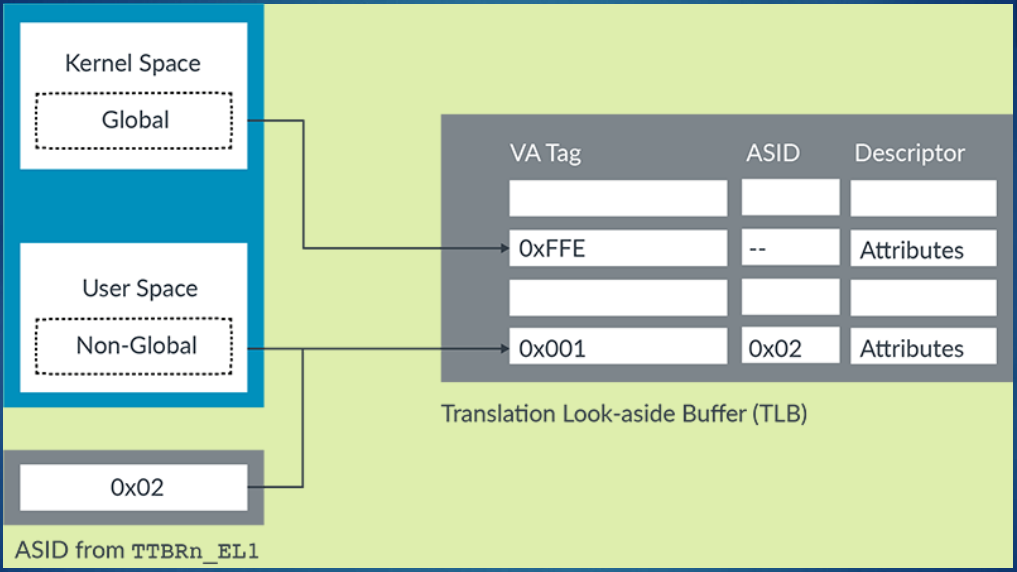
Do you have any clue what this could be? I read about the problem. It appears when a cpu is stuck ot task for a long time or something like that.
But that is normal for VMs. Can I work around it?

Best regards.

https://bbs.archlinux.org/viewtopic.php?id=252523

ASID

Address Space Identifiers



* https://developer.arm.com/documentation/101811/0101/Address-spaces-in-AArch64

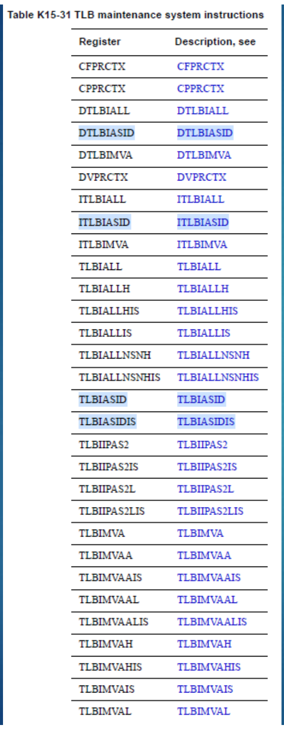


Table K15-31 TLB maintenance system instructions

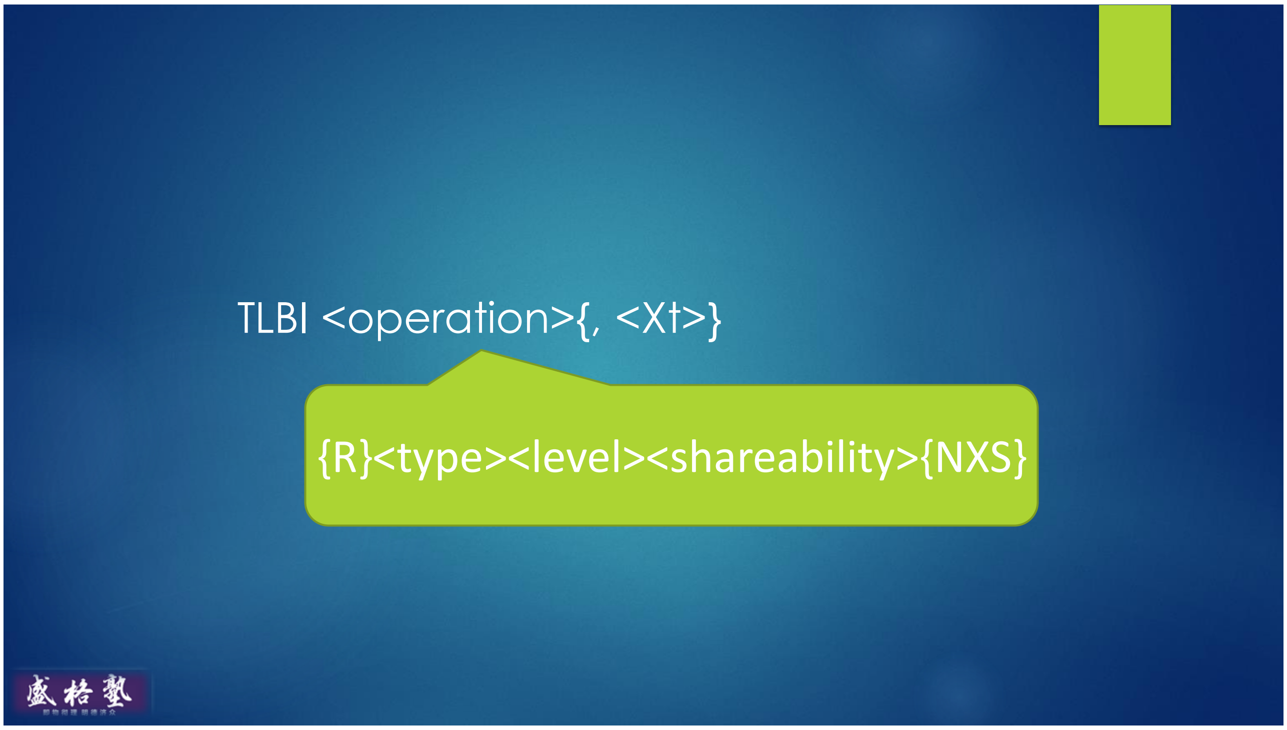
Register	Description, see
CFPRCTX	CFPRCTX
CPPRCTX	CPPRCTX
DTLBIALL	DTLBIALL
DTLBIASID	DTLBIASID
DTLBIMVA	DTLBIMVA
DVPRCTX	DVPRCTX
ITLBIALL	ITLBIALL
ITLBIASID	ITLBIASID
ITLBIMVA	ITLBIMVA
TLBIALL	TLBIALL
TLBIALLH	TLBIALLH
TLBIALLHIS	TLBIALLHIS
TLBIALLIS	TLBIALLIS
TLBIALLNSNH	TLBIALLNSNH
TLBIALLNSNHIS	TLBIALLNSNHIS
TLBIASID	TLBIASID
TLBIASIDIS	TLBIASIDIS
TLBIIPAS2	TLBIIPAS2
TLBIIPAS2IS	TLBIIPAS2IS
TLBIIPAS2L	TLBIIPAS2L
TLBIIPAS2LIS	TLBIIPAS2LIS
TLBIMVA	TLBIMVA
TLBIMVAA	TLBIMVAA
TLBIMVAAIS	TLBIMVAAIS
TLBIMVAAL	TLBIMVAAL
TLBIMVAALIS	TLBIMVAALIS
TLBIMVAH	TLBIMVAH
TLBIMVAHIS	TLBIMVAHIS
TLBIMVAIS	TLBIMVAIS
TLBIMVAL	TLBIMVAL

TLB maintenance system instructions

DTLBIASID, Data TLB Invalidate by ASID match

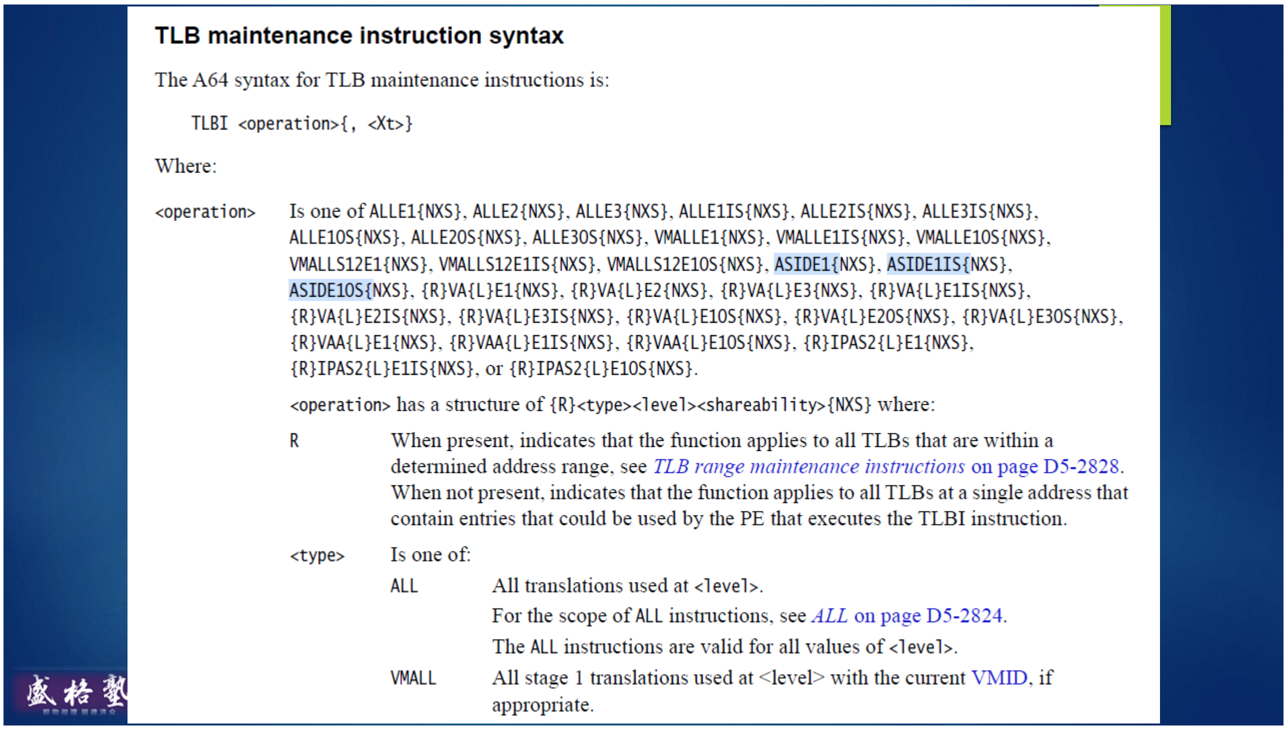
ITLBIASID, Instruction TLB Invalidate by ASID match

TLBIASID, TLB Invalidate by ASID match



TLBI <operation>{, <Xt>}

{R}<type><level><shareability>{NXS}



TLB maintenance instruction syntax

The A64 syntax for TLB maintenance instructions is:

```
TLBI <operation>{, <Xt>}
```

Where:

`<operation>` Is one of ALLE1{NXS}, ALLE2{NXS}, ALLE3{NXS}, ALLE1IS{NXS}, ALLE2IS{NXS}, ALLE3IS{NXS}, ALLE1OS{NXS}, ALLE2OS{NXS}, ALLE3OS{NXS}, VMALLE1{NXS}, VMALLE1IS{NXS}, VMALLE1OS{NXS}, VMALLS12E1{NXS}, VMALLS12E1IS{NXS}, VMALLS12E1OS{NXS}, ASIDE1{NXS}, ASIDE1IS{NXS}, ASIDE1OS{NXS}, {R}VA{L}E1{NXS}, {R}VA{L}E2{NXS}, {R}VA{L}E3{NXS}, {R}VA{L}E1IS{NXS}, {R}VA{L}E2IS{NXS}, {R}VA{L}E3IS{NXS}, {R}VA{L}E1OS{NXS}, {R}VA{L}E2OS{NXS}, {R}VA{L}E3OS{NXS}, {R}VAA{L}E1{NXS}, {R}VAA{L}E1IS{NXS}, {R}VAA{L}E1OS{NXS}, {R}IPAS2{L}E1{NXS}, {R}IPAS2{L}E1IS{NXS}, or {R}IPAS2{L}E1OS{NXS}.

`<operation>` has a structure of {R}<type><level><shareability>{NXS} where:

R — When present, indicates that the function applies to all TLBs that are within a determined address range, see *TLB range maintenance instructions* on page D5-2828. When not present, indicates that the function applies to all TLBs at a single address that contain entries that could be used by the PE that executes the TLBI instruction.

`<type>` Is one of:

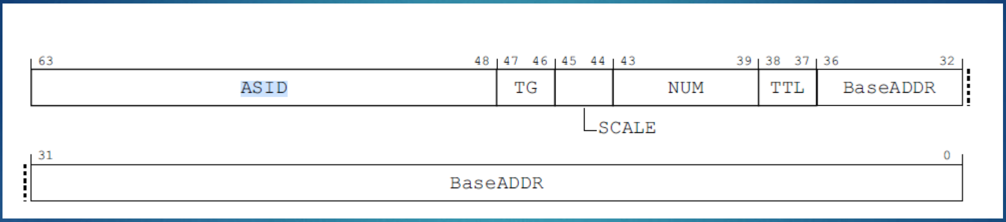
ALL — All translations used at <level>.
For the scope of ALL instructions, see *ALL* on page D5-2824.
The ALL instructions are valid for all values of <level>.

VMALL — All stage 1 translations used at <level> with the current VMID, if appropriate.

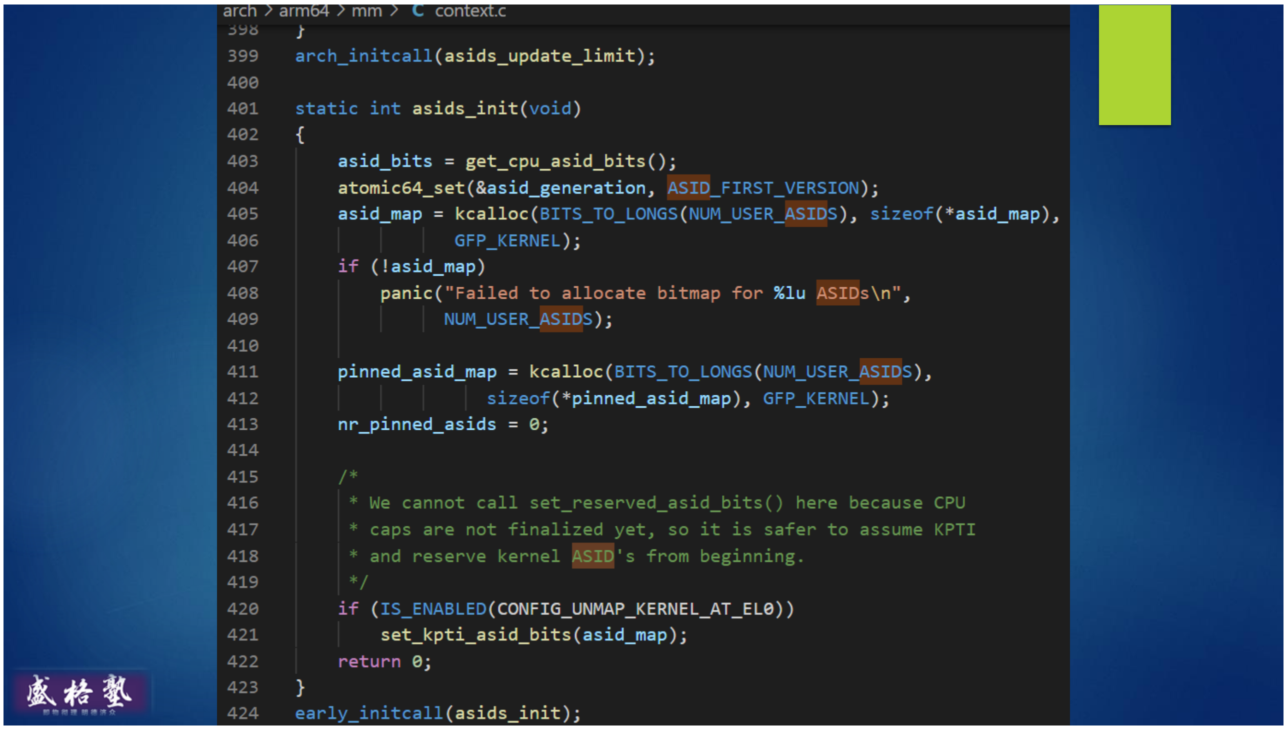
C tlbflush.h ×

arch > arm64 > include > asm > C tlbflush.h

```
static inline void flush_tlb_kernel_range(unsigned long start, unsigned long end)
{
        unsigned long addr;

        if ((end - start) > (MAX_TLBI_OPS * PAGE_SIZE)) {
                flush_tlb_all();
                return;
        }

        start = __TLBI_VADDR(start, 0);
        end = __TLBI_VADDR(end, 0);

        dsb(ishst);
        for (addr = start; addr < end; addr += 1 << (PAGE_SHIFT - 12))
                __tlbi(vaale1is, addr);
        dsb(ish);
        isb();
}
```

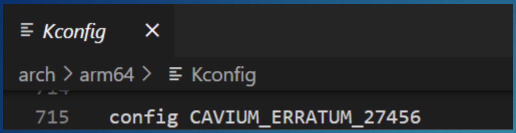


TLB表项示例



arch > arm64 > mm > context.c

```c
}
arch_initcall(asids_update_limit);

static int asids_init(void)
{
	asid_bits = get_cpu_asid_bits();
	atomic64_set(&asid_generation, ASID_FIRST_VERSION);
	asid_map = kcalloc(BITS_TO_LONGS(NUM_USER_ASIDS), sizeof(*asid_map),
			   GFP_KERNEL);
	if (!asid_map)
		panic("Failed to allocate bitmap for %lu ASIDs\n",
		      NUM_USER_ASIDS);

	pinned_asid_map = kcalloc(BITS_TO_LONGS(NUM_USER_ASIDS),
				  sizeof(*pinned_asid_map), GFP_KERNEL);
	nr_pinned_asids = 0;

	/*
	 * We cannot call set_reserved_asid_bits() here because CPU
	 * caps are not finalized yet, so it is safer to assume KPTI
	 * and reserve kernel ASID's from beginning.
	 */
	if (IS_ENABLED(CONFIG_UNMAP_KERNEL_AT_EL0))
		set_kpti_asid_bits(asid_map);
	return 0;
}
early_initcall(asids_init);
```

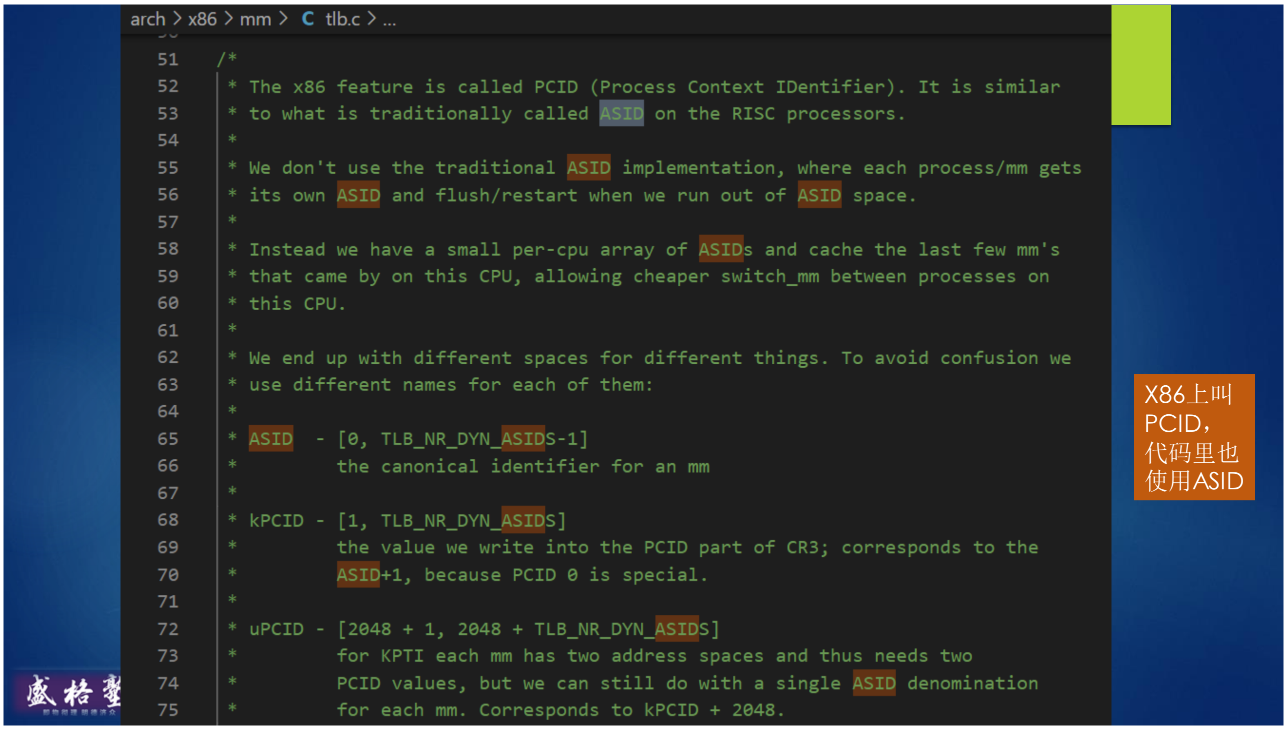


Kconfig ×

arch > arm64 > Kconfig

```
config CAVIUM_ERRATUM_27456
```

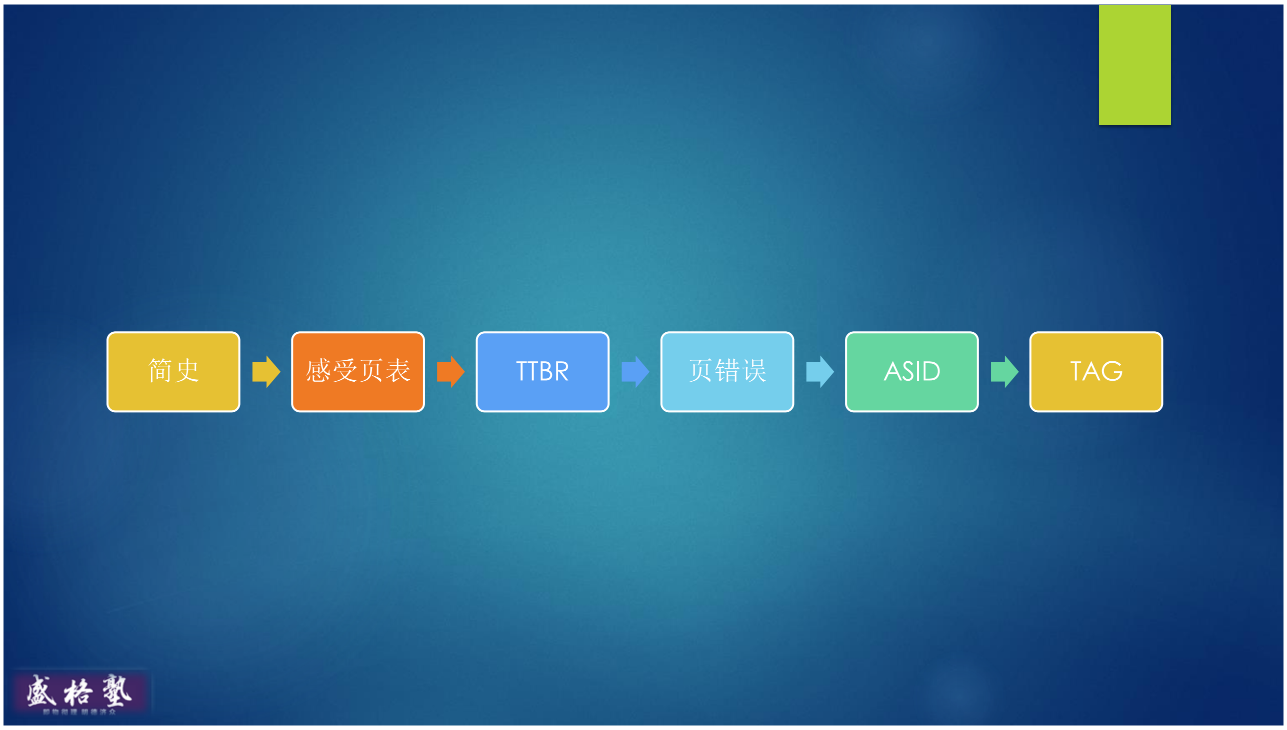
```
config CAVIUM_ERRATUM_27456
	bool "Cavium erratum 27456: Broadcast TLBI instructions may cause icache corruption"
	default y
	help
	  On ThunderX T88 pass 1.x through 2.1 parts, broadcast TLBI
	  instructions may cause the icache to become corrupted if it
	  contains data for a non-current ASID.  The fix is to
	  invalidate the icache when changing the mm context.

	  If unsure, say Y.
```

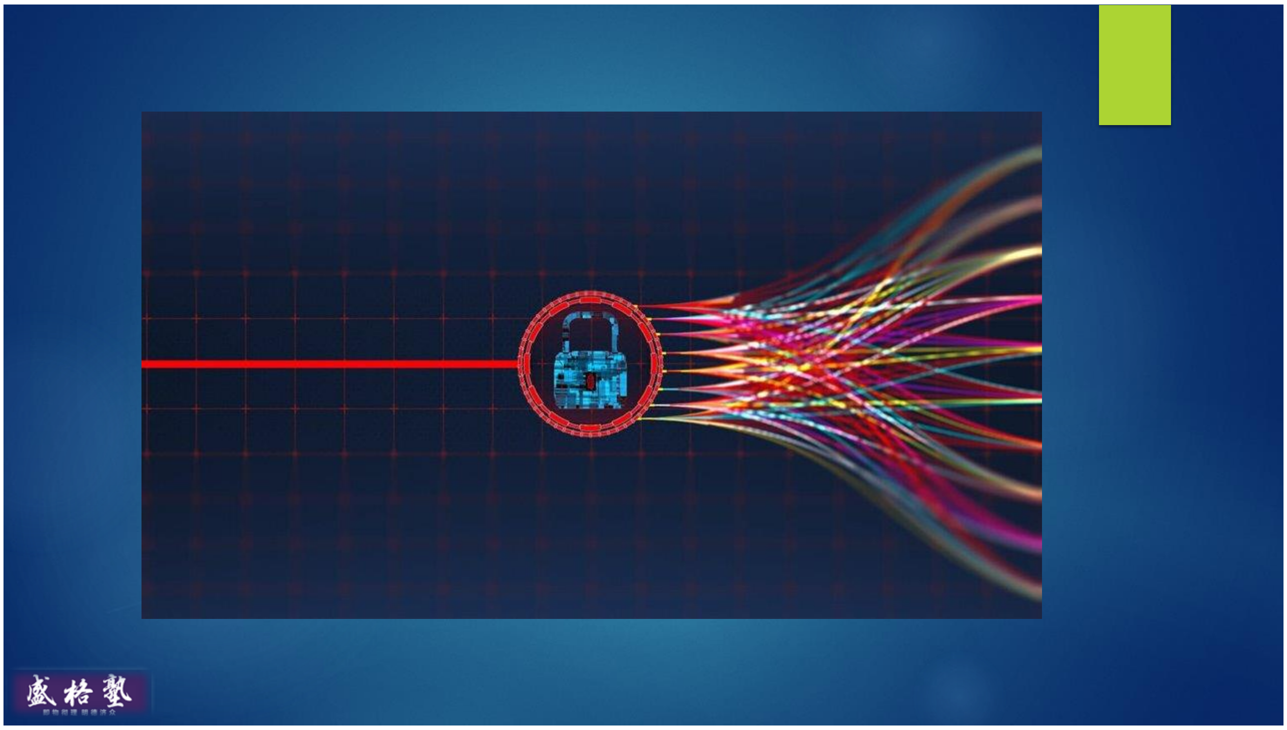


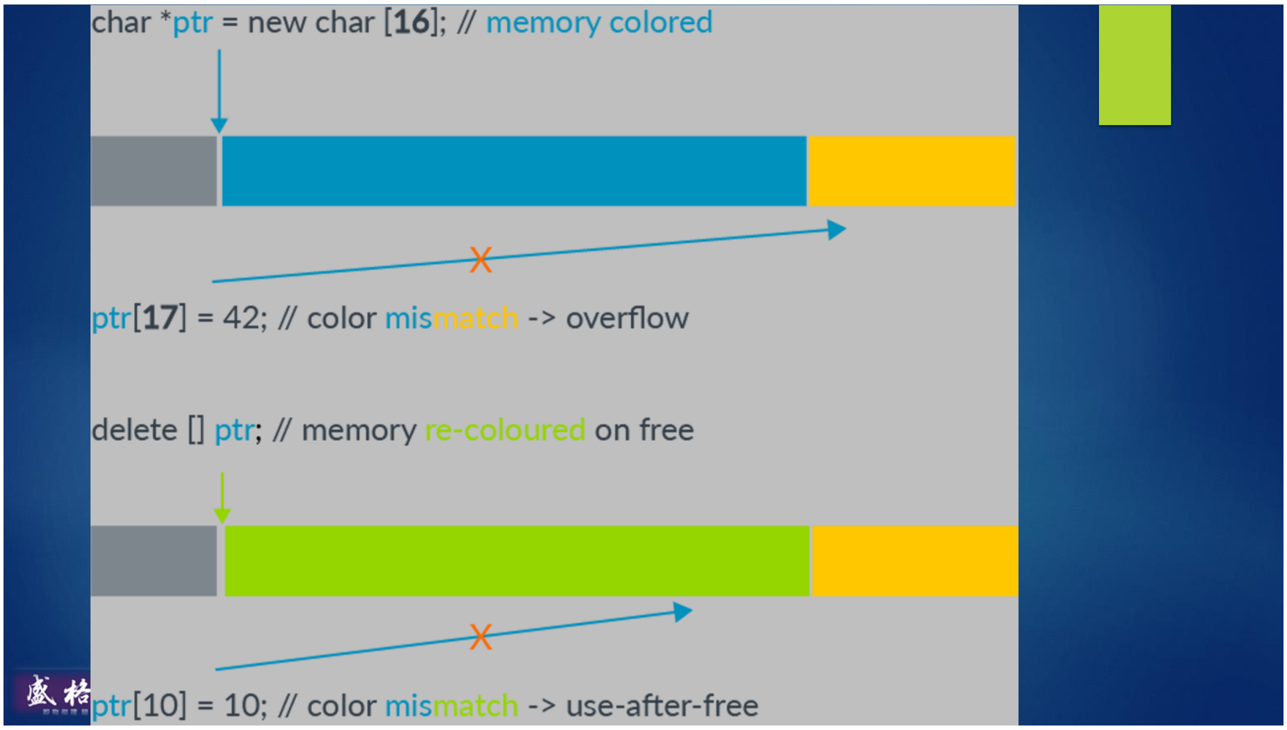
arch > x86 > mm > C tlb.c > ...

```
/*
 * The x86 feature is called PCID (Process Context IDentifier). It is similar
 * to what is traditionally called ASID on the RISC processors.
 *
 * We don't use the traditional ASID implementation, where each process/mm gets
 * its own ASID and flush/restart when we run out of ASID space.
 *
 * Instead we have a small per-cpu array of ASIDs and cache the last few mm's
 * that came by on this CPU, allowing cheaper switch_mm between processes on
 * this CPU.
 *
 * We end up with different spaces for different things. To avoid confusion we
 * use different names for each of them:
 *
 * ASID  - [0, TLB_NR_DYN_ASIDS-1]
 *         the canonical identifier for an mm
 *
 * kPCID - [1, TLB_NR_DYN_ASIDS]
 *         the value we write into the PCID part of CR3; corresponds to the
 *         ASID+1, because PCID 0 is special.
 *
 * uPCID - [2048 + 1, 2048 + TLB_NR_DYN_ASIDS]
 *         for KPTI each mm has two address spaces and thus needs two
 *         PCID values, but we can still do with a single ASID denomination
 *         for each mm. Corresponds to kPCID + 2048.
```

X86上叫PCID，代码里也使用ASID





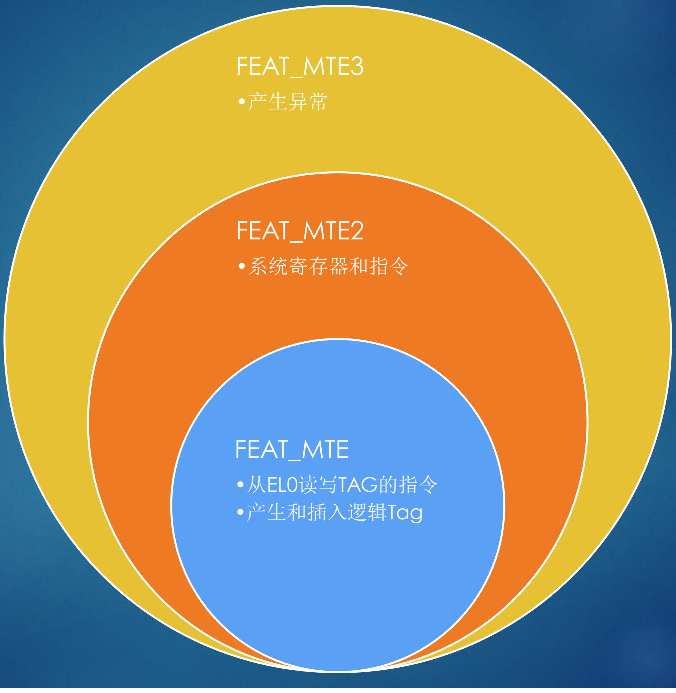


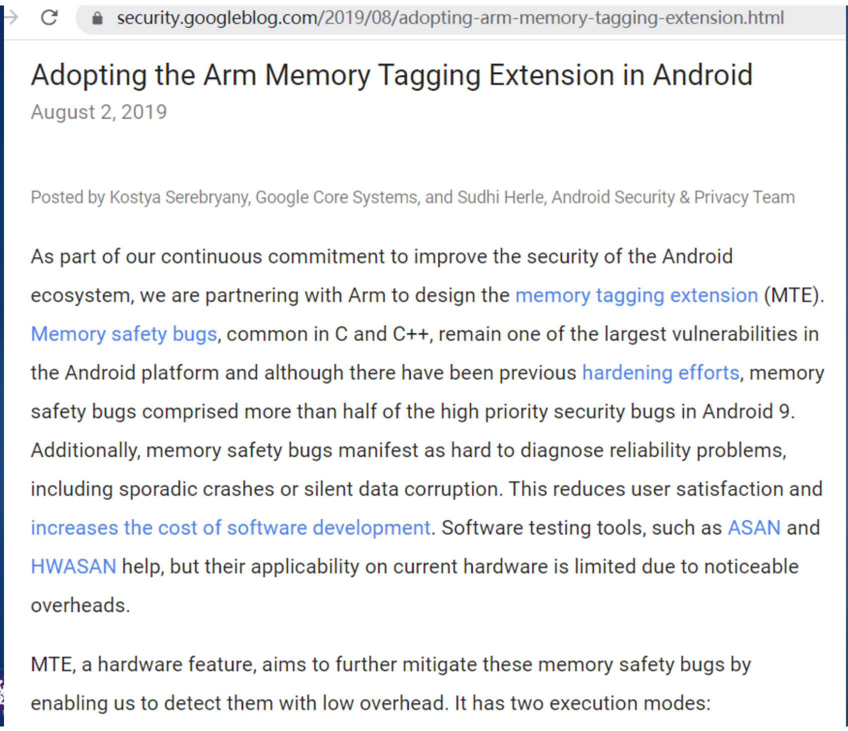
char *ptr = new char [16]; // memory colored

ptr[17] = 42; // color mismatch -> overflow

delete [] ptr; // memory re-coloured on free

ptr[10] = 10; // color mismatch -> use-after-free





security.googleblog.com/2019/08/adopting-arm-memory-tagging-extension.html

Adopting the Arm Memory Tagging Extension in Android

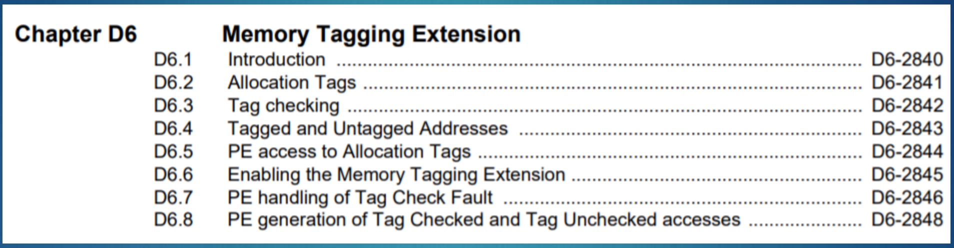
August 2, 2019

Posted by Kostya Serebryany, Google Core Systems, and Sudhi Herle, Android Security & Privacy Team

As part of our continuous commitment to improve the security of the Android ecosystem, we are partnering with Arm to design the memory tagging extension (MTE). Memory safety bugs, common in C and C++, remain one of the largest vulnerabilities in the Android platform and although there have been previous hardening efforts, memory safety bugs comprised more than half of the high priority security bugs in Android 9. Additionally, memory safety bugs manifest as hard to diagnose reliability problems, including sporadic crashes or silent data corruption. This reduces user satisfaction and increases the cost of software development. Software testing tools, such as ASAN and HWASAN help, but their applicability on current hardware is limited due to noticeable overheads.

MTE, a hardware feature, aims to further mitigate these memory safety bugs by enabling us to detect them with low overhead. It has two execution modes:

https://security.googleblog.com/2019/08/adopting-arm-memory-tagging-extension.html



Chapter D6 **Memory Tagging Extension**

切问而近思

欢迎关注格友公众号