**实验八 分页内存管理**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab8/index.html#id1)

**Armv8的地址转换**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab8/index.html#armv8)

[ARM Cortex-A Series Programmer’s Guide for ARMv8-A](https://developer.arm.com/documentation/den0024/a/The-Memory-Management-Unit/Context-switching) 中提到：For EL0 and EL1, there are two translation tables. TTBR0\_EL1 provides translations for the bottom of Virtual Address space, which is typically application space and TTBR1\_EL1 covers the top of Virtual Address space, typically kernel space. This split means that the OS mappings do not have to be replicated in the translation tables of each task. 即TTBR0指向整个虚拟空间下半部分通常用于应用程序的空间，TTBR1指向虚拟空间的上半部分通常用于内核的空间。其中TTBR0除了在EL1中存在外，也在EL2 and EL3中存在，但TTBR1只在EL1中存在。

TTBR0\_ELn 和 TTBR1\_ELn 是页表基地址寄存器，地址转换的过程如下所示。

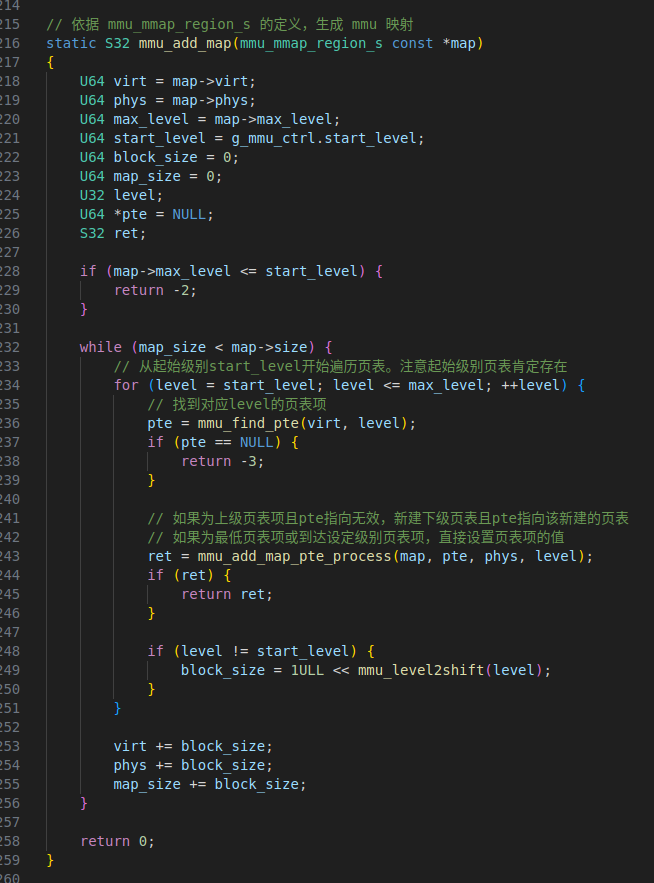
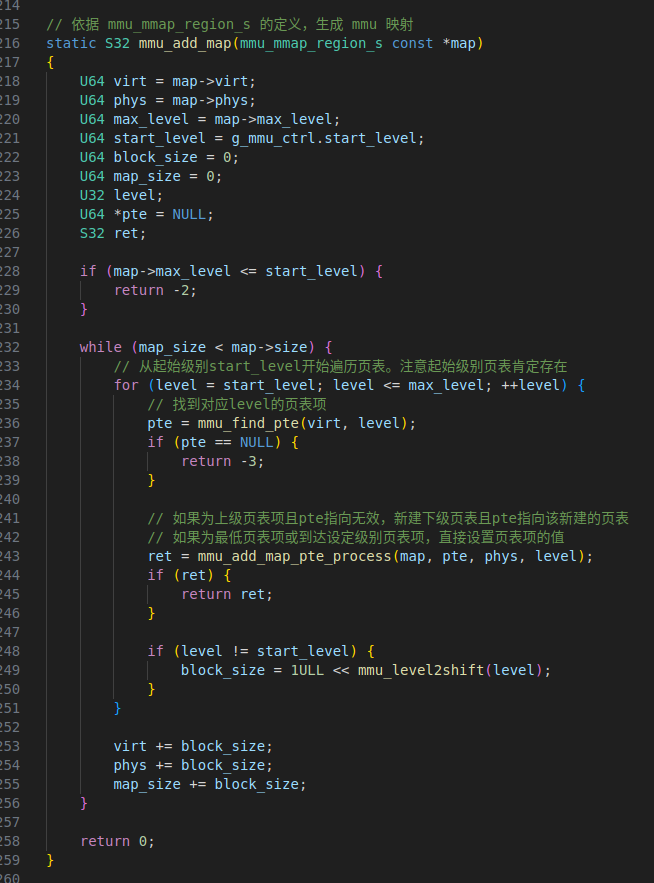
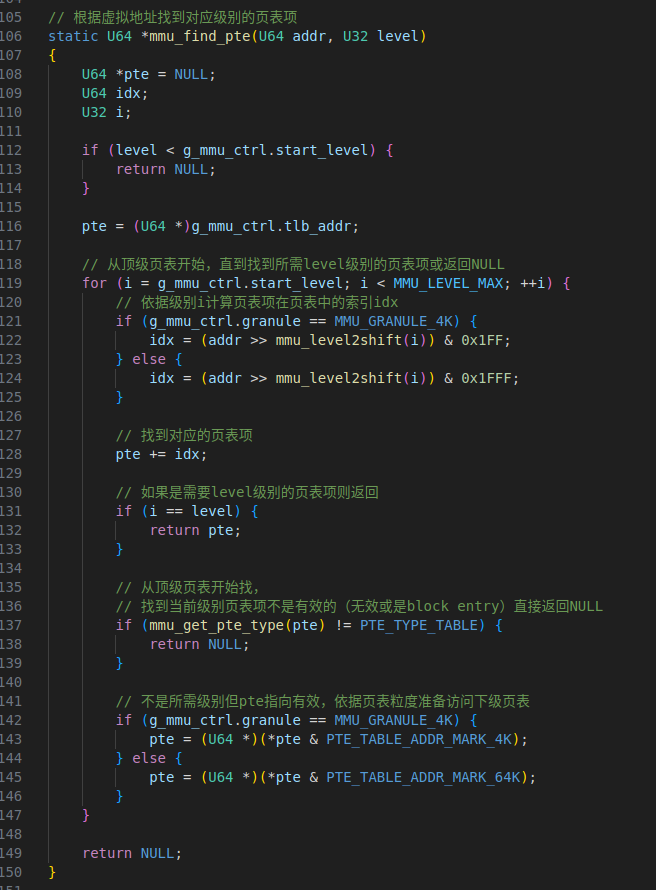
In a simple address translation involving only one level of look-up. It assumes we are using a 64KB granule with a 42-bit Virtual Address. The MMU translates a Virtual Address as follows:

1. If VA[63:42] = 1 then TTBR1 is used for the base address for the first page table. When VA[63:42] = 0, TTBR0 is used for the base address for the first page table.
2. The page table contains 8192 64-bit page table entries, and is indexed using VA[41:29]. The MMU reads the pertinent level 2 page table entry from the table.
3. The MMU checks the page table entry for validity and whether or not the requested memory access is allowed. Assuming it is valid, the memory access is allowed.
4. In the above Figure, the page table entry refers to a 512MB page (it is a block descriptor).
5. Bits [47:29] are taken from this page table entry and form bits [47:29] of the Physical Address.
6. Because we have a 512MB page, bits [28:0] of the VA are taken to form PA[28:0]. See Effect of granule sizes on translation tables
7. The full PA[47:0] is returned, along with additional information from the page table entry.

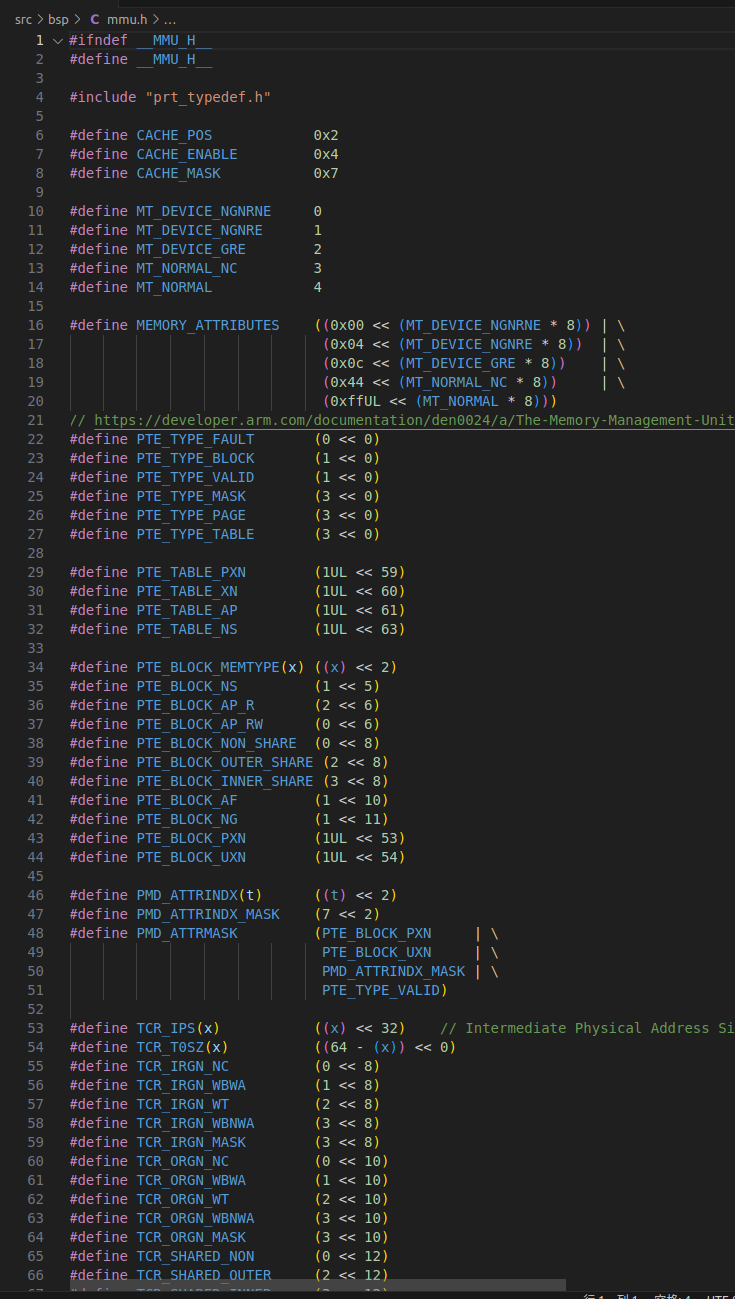
In practice, such a simple translation process severely limits how finely you can divide up your address space. Instead of using only this first-level translation table, a first-level table entry can also point to a second-level page table.

**mmu管理**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab8/index.html#mmu)

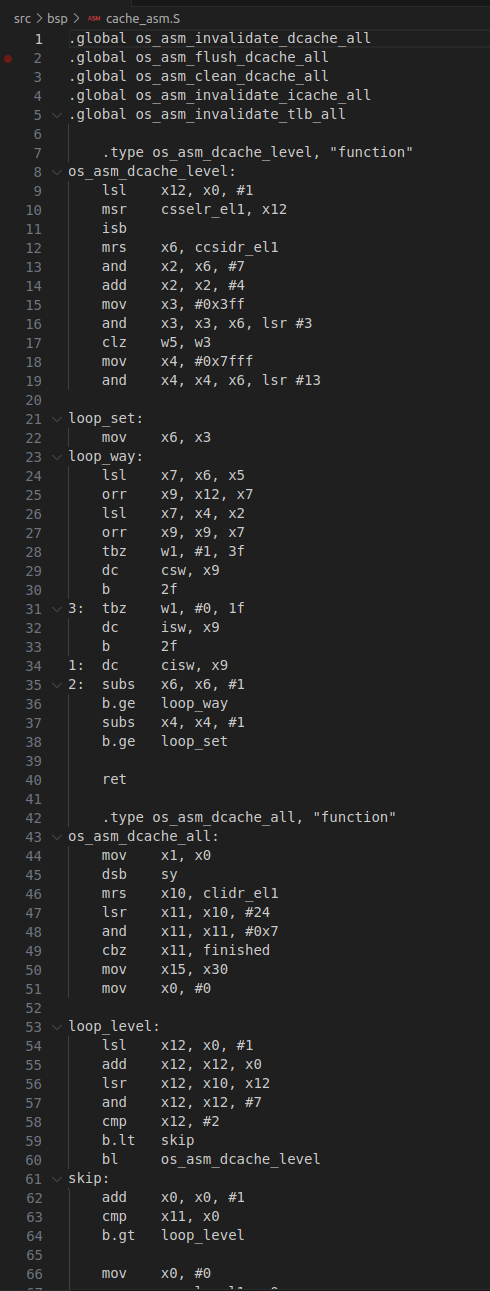
新建 src/bsp/mmu.c 文件



新建 src/bsp/mmu.h，

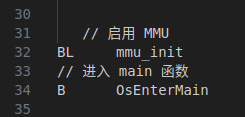


新建 src/bsp/cache\_asm.S，

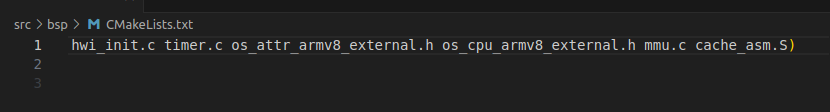


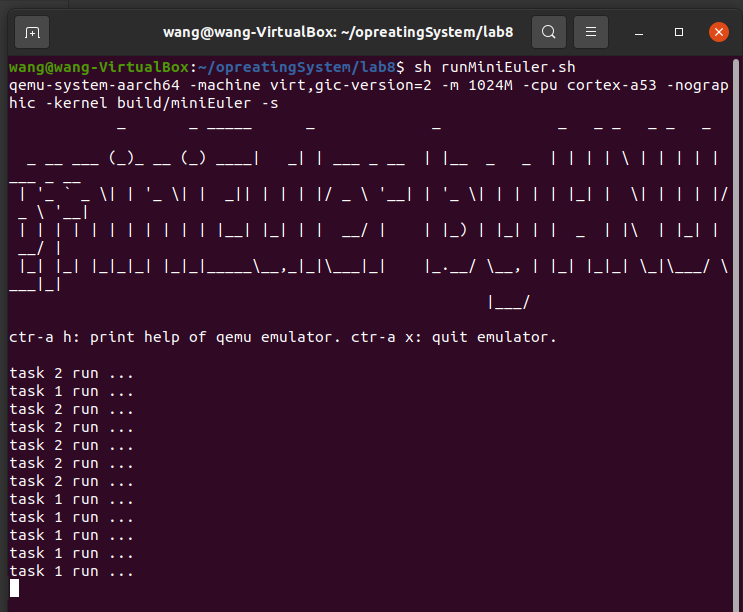
**启用 mmu**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab8/index.html#id4)

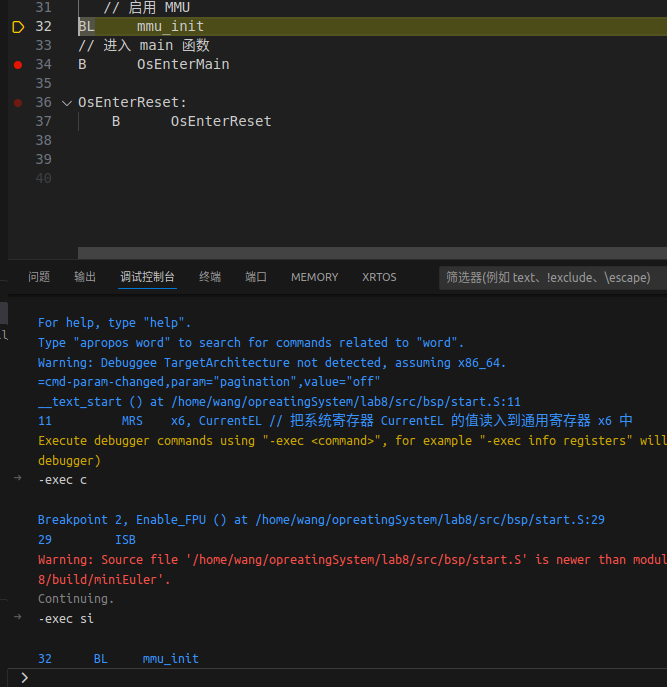
start.S 中在 B OsEnterMain 之前启用 MMU

**提示**

将新增文件加入构建系统







**提示**

通过调试确保你真的启动了 MMU

**lab8 作业**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab8/index.html#lab8)

**作业1**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab8/index.html#id5)

启用 TTBR1 ，将地址映射到虚拟地址的高半部分，使用高地址访问串口 修改后：（1）src/bsp/print.c中

#define UART\_0\_REG\_BASE (0xffffffff00000000 + 0x09000000)

(2)src/bsp/hwi\_init.c 中

#define GIC\_DIST\_BASE (0xffffffff00000000 + 0x08000000)

#define GIC\_CPU\_BASE (0xffffffff00000000 + 0x08010000)

程序可以正常运行。（GIC\_DIST\_BASE 和 GIC\_CPU\_BASE 的高位多少个f与你对MMU的配置有关）

