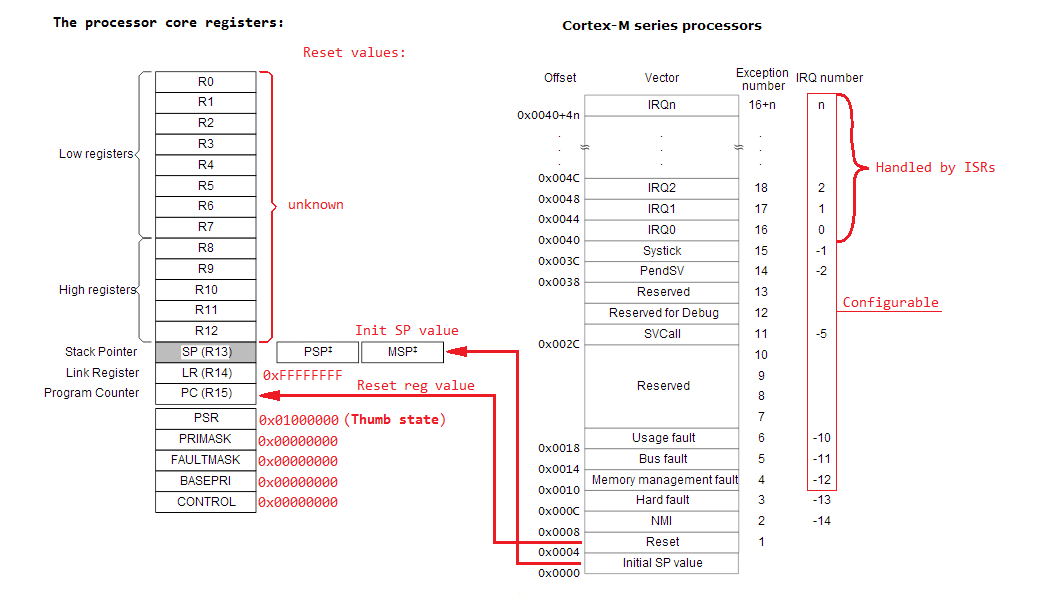
# BOOTING PROCESS

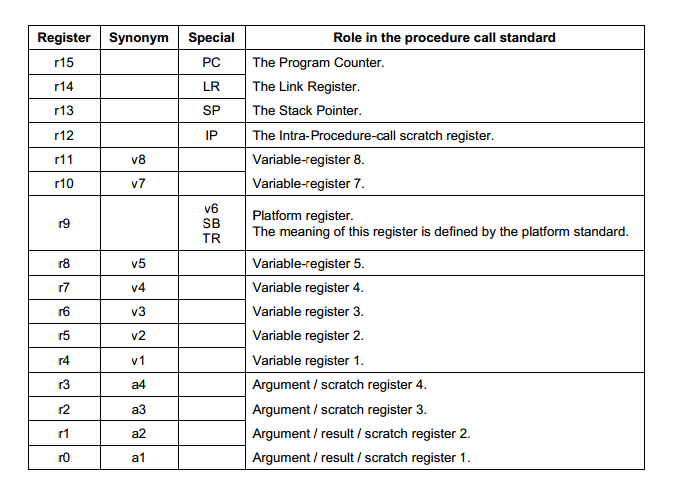
## CORE REGISTER

Here's what happens after the processor is reset:

1. General-purpose registers have unknown values in them
2. Stack pointer register is loaded with the value of the Initial SP value register located at 0x0000 (Vector Table)
3. Link register will have 0xFFFFFFFF
4. Program counter is loaded with the value of the Reset register located at 0x0004 (Vector Table)
5. In the case of STM32F4, Reset register contains the address of the Reset\_Handler which is defined in startup\_stm32f40xx.s
6. Reset\_Handler performs microcontroller system setup, configures the system clock, eventually branches to \_\_main (entry point of a program) in the C library.
7. Vector Table is mapped to address 0 at reset (can be relocated in this range: 0x00000080 - 0x3FFFFF80)

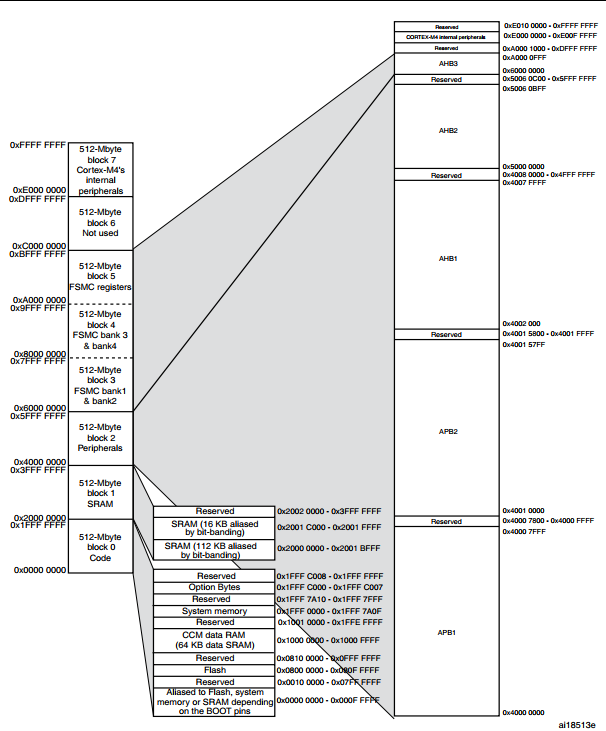


Summary of registers and their roles in the procedure call standard (AAPCS).



## MEMORY ORGANIZATION

The Cortex-M processors have 32-bit memory addressing and therefore have 4GB memory space.



|  |  |  |  |
| --- | --- | --- | --- |
| ****Region**** | ****Address Range**** | ****Size**** | ****Function**** |
| Code | 0x00000000 – 0x1FFFFFFF | 512MB | – Store program code, including default vector table. |
| SRAM | 0x20000000 – 0x3FFFFFFF | 512MB | -Primarily for connecting SRAM, mostly on-chip SRAM – Program code can be executed from this region – The first 1MB of the SRAM region is bit addressable if the optional bit-band feature is included |
| Peripherals | 0x40000000 – 0x5FFFFFFF | 512MB | – Use mostly for on-chip peripherals – The first 1MB of the SRAM region is bit addressable if the optional bit-band feature is included |
| RAM | 0x60000000- 0x9FFFFFFF | 1GB | – Contain two slots on 512MB memory space (total 1GB) for other RAM such as off chip memories – Can be used for program code as well as data |
| Devices | 0xA0000000 – 0xDFFFFFFF | 1GB | – Contain two slots on 512MB memory space (toal 1GB) for other peripheral |
| System | 0xE0000000 – 0xFFFFFFFF | 512MB | – Contain several part: Internal Private Peripheral Bus; External Private Peripheral Bus; Vendor-specific area |

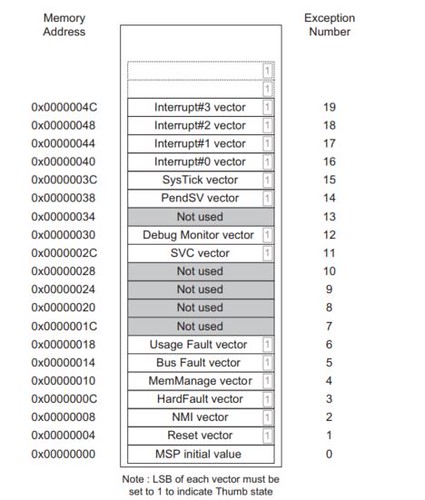
## CODE REGION IN DETAIL

### Vector table

The vector table holds the starting address of interrupts and system exception.

By default the vector table is located at the beginning of the memory space(address 0x00), and the vector address is arranged according to the exception number time four.

The vector table is normally defined in the startup codes provided by the microcontroller vendors.



The vector table used in startup code also contains the initial value of the main stack pointer (MSP).

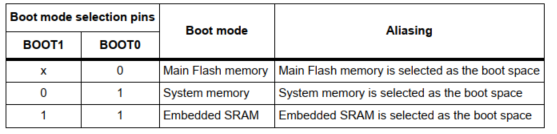
One thing you need to note that the vector tables in the Cortex-M processors are different form the vector tables in traditional ARM processor. In traditional ARM processors, the vector tables contain instructions such as branch instruction to branch to appropriate handlers, whereas in Cortex-M, the vector table contains the starting addresses of exception handlers.

### Physical remap

#### Boot mode configuration

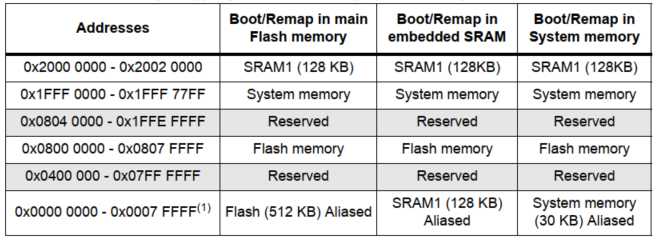
Due to its fixed memory map, the code area starts from address 0x00000000 (accessed through the ICode/DCode buses) while the data area (SRAM) starts from address 0x20000000 (accessed through system bus).

The Cortex-M4 with FPU CPU always fetches the reset vector on the ICode bus, which implies to have the boot space available only in code area. STM32F4xE microcontrollers implement a special mechanism to boot from other memory.



#### Physical remap in STM32F411xE

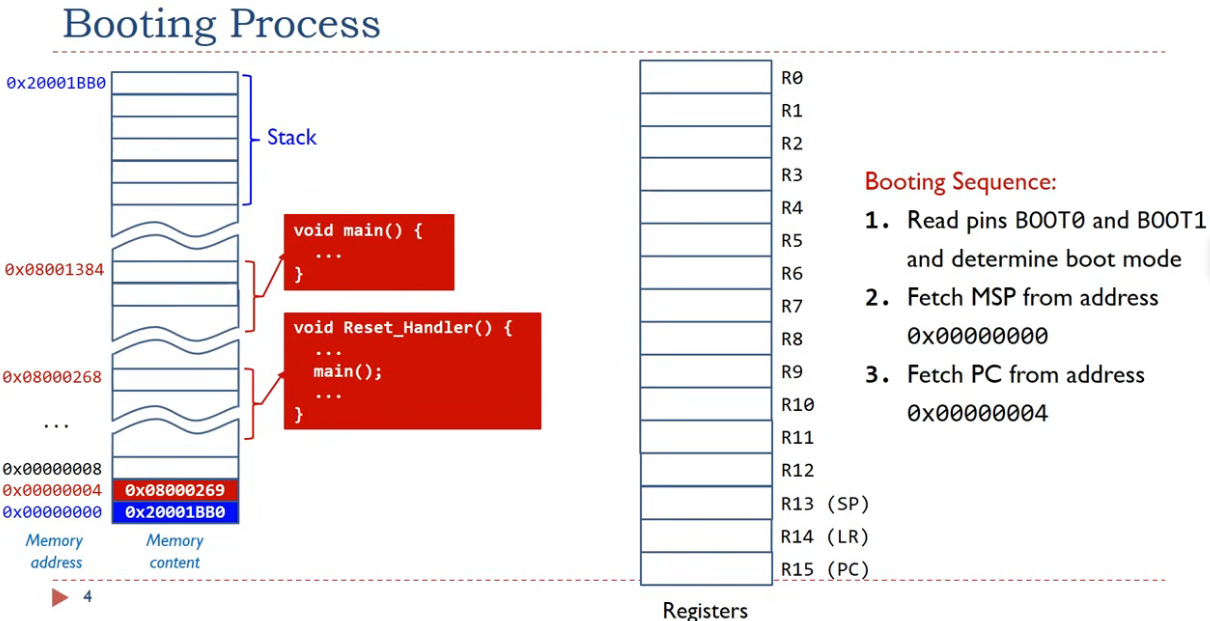
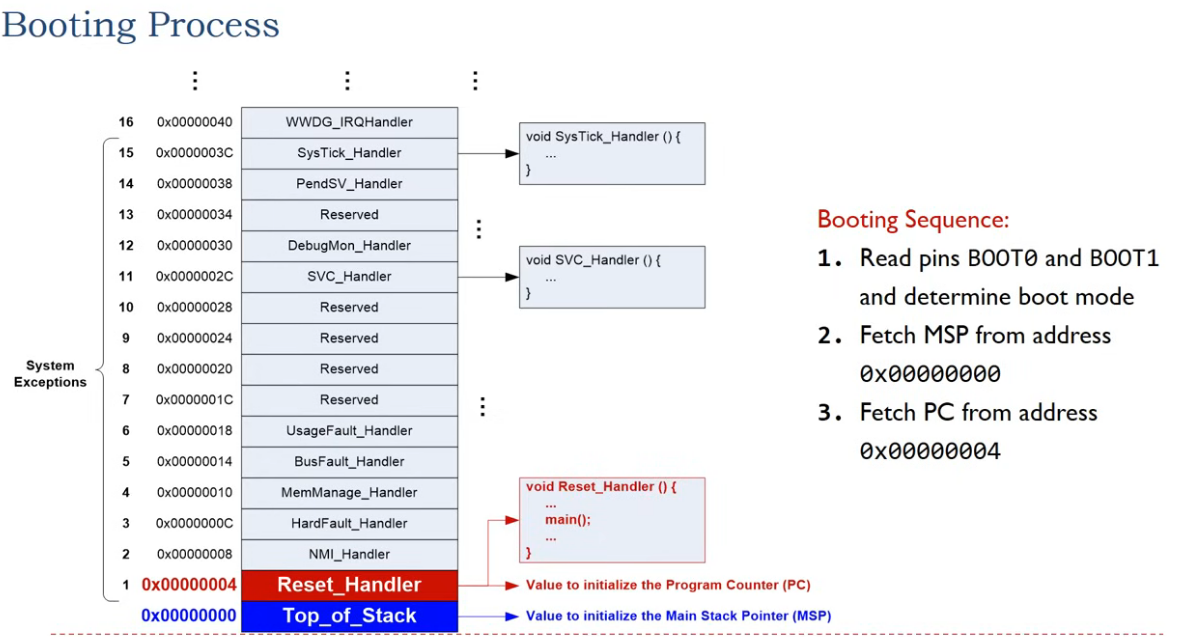
Now look code CODE region area in detail you can see the memory mapping and boot mode remap in STM32F411xE

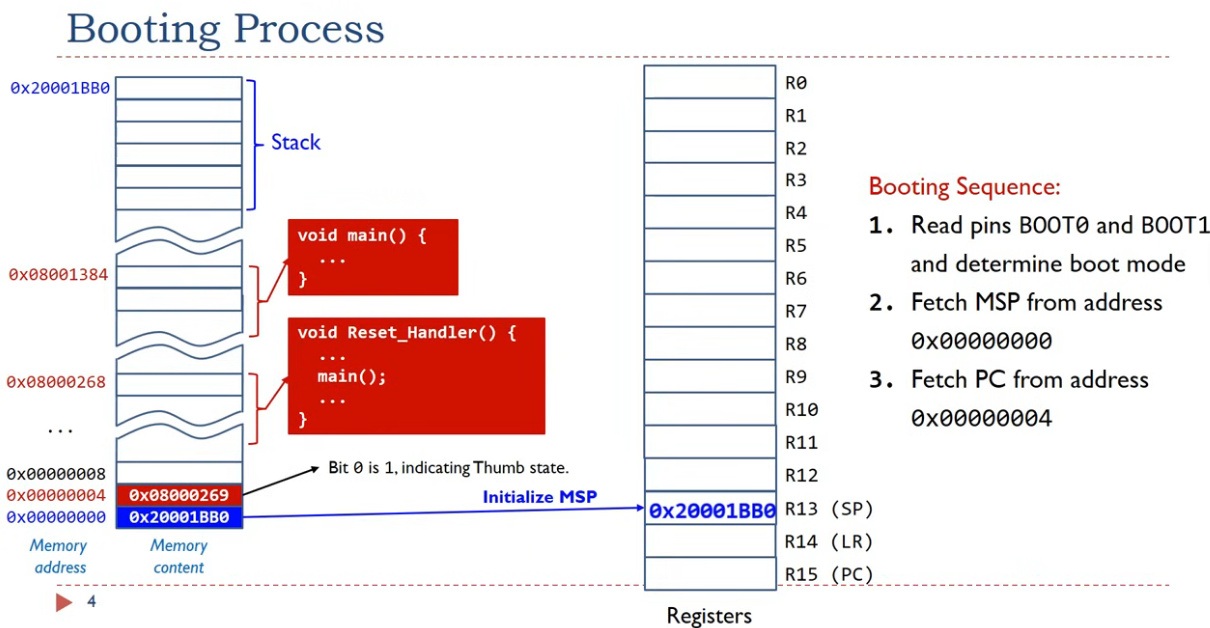


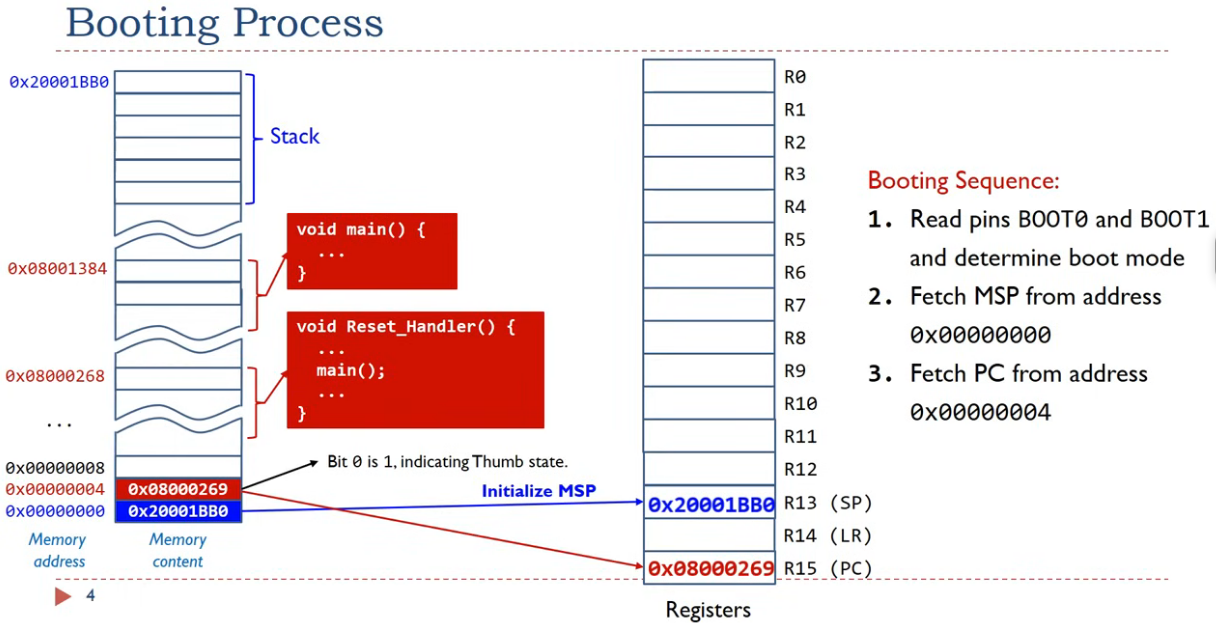
Based on the BOOT pin configuration, the corresponding memories can be remapped to Alias Area.

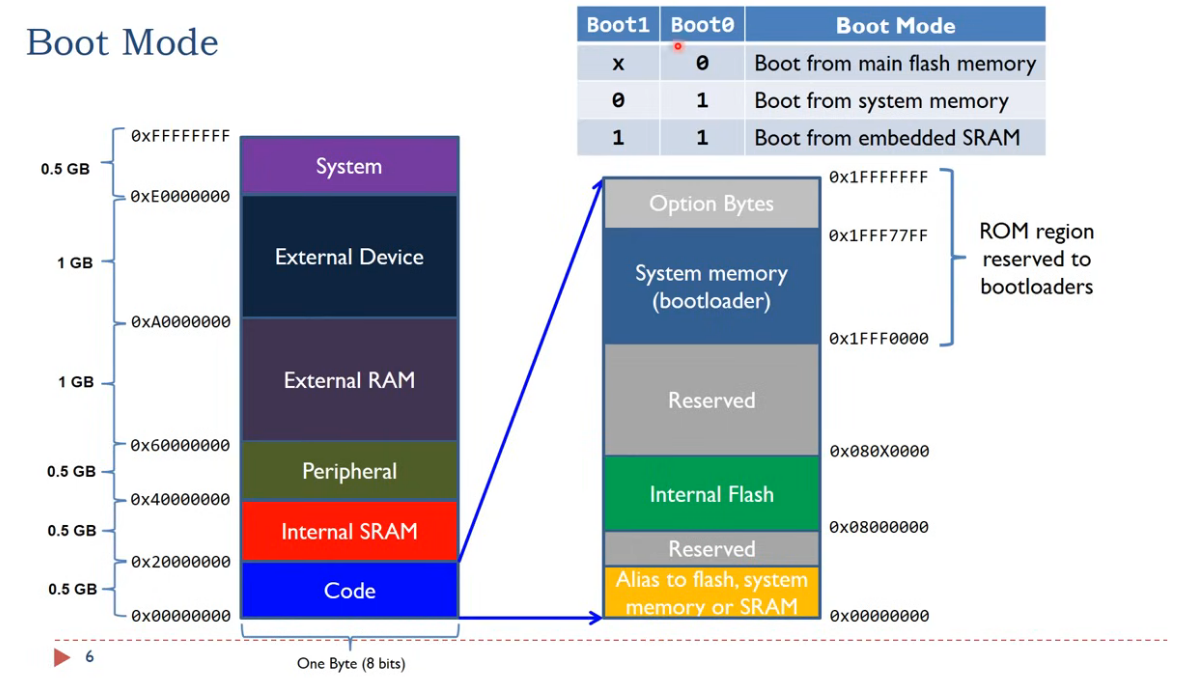
## BOOTING SEQUENCE

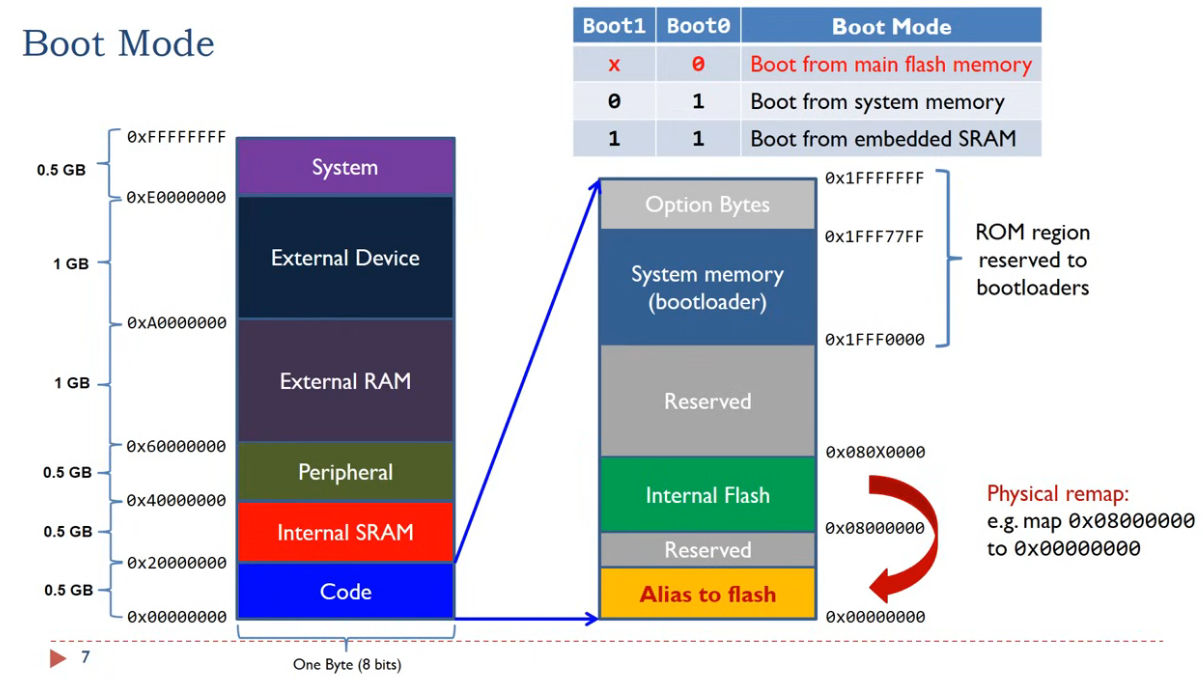
Now we understand the basic concepts, lets see how they work in booting time.

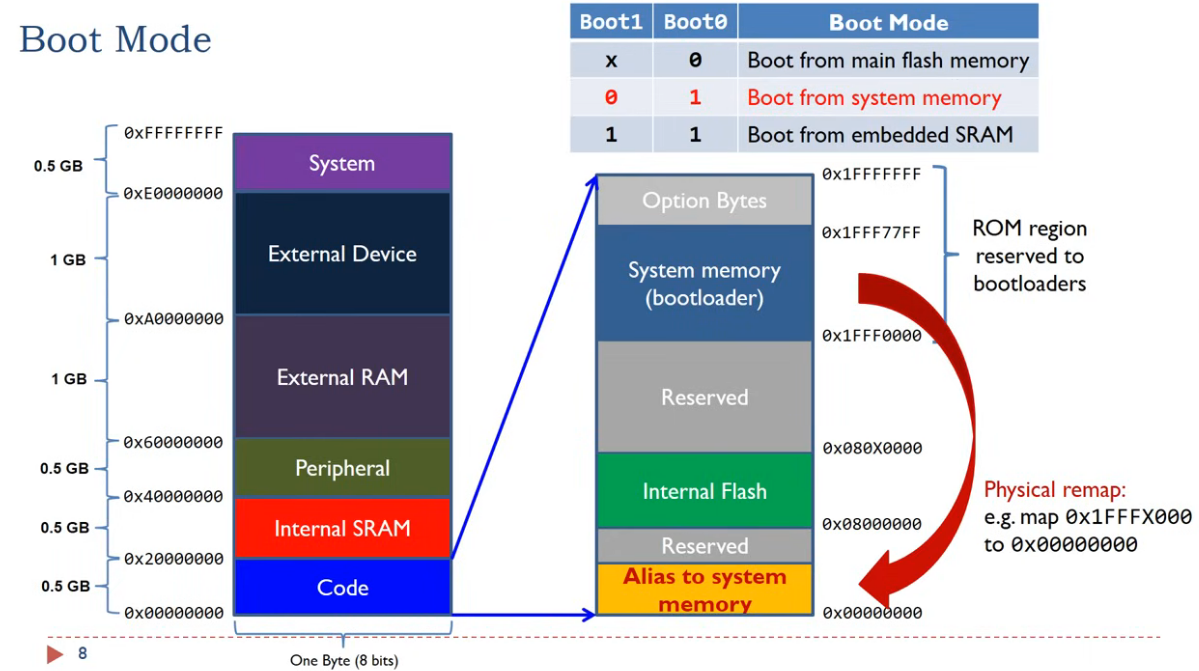
1. Power On Reset
2. Execute boot loader in boot ROM
3. Processor read BOOT pins to determine the booting mode. For example if BOOT0 is 0, Main Flash memory is mapped into Alias space from address 0x00000000
4. Next processor fetch the first word from memory space. As the beginning of the memory space contains the vector table, and the first word in the vector table is the initial value for the Main Stack Pointer (MSP). Processor set up the MSP after that.
5. Processor continues read the next word in vector table, now is reset vector, which contains the Reset\_Handler() function address.
6. Reset\_Handler() function address is moved into Program Counter (PC) and processor starts running Reset\_Handler()
7. Inside Reset\_Handler(), System\_Init( ) function will be run first, follow is our application (main())

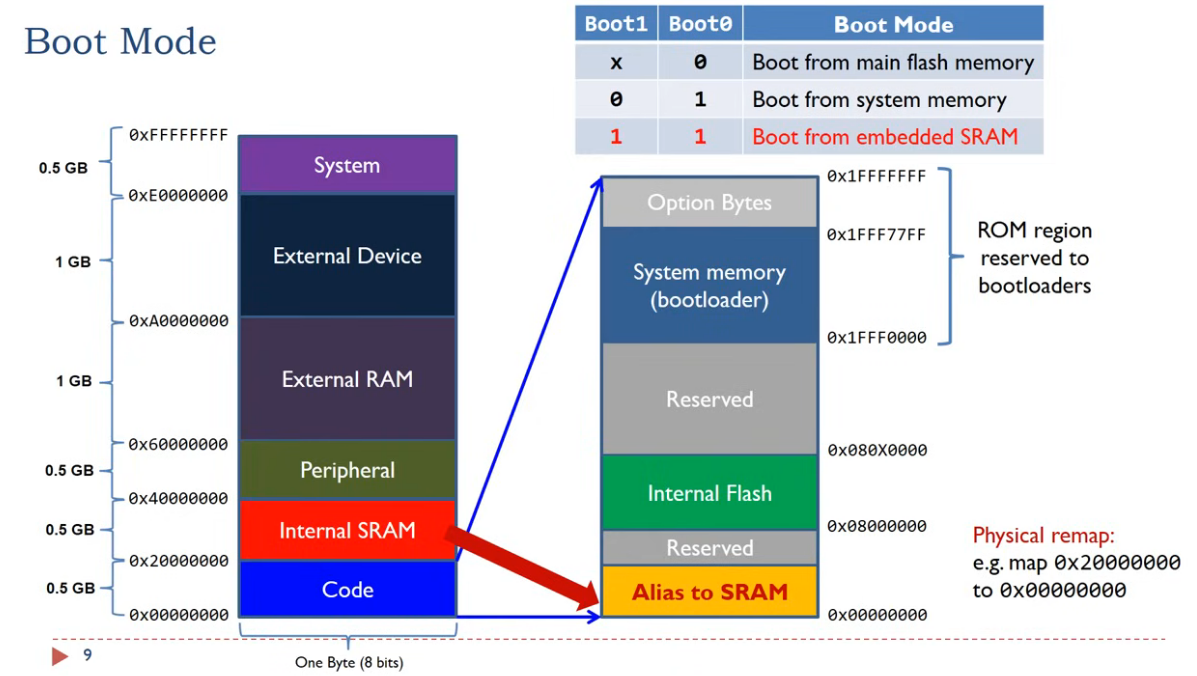










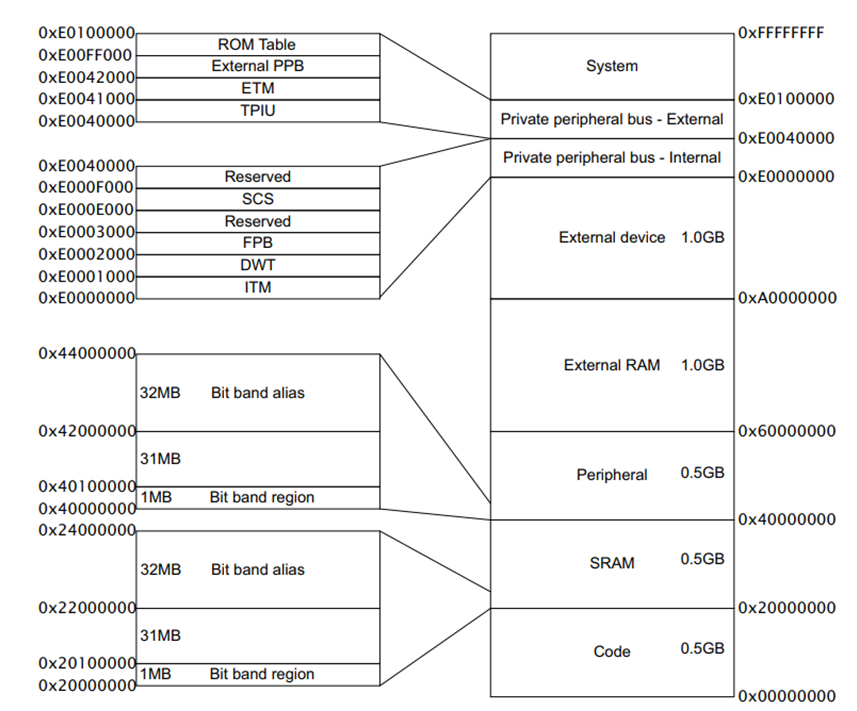


# Chức năng, tổ chức và bản đồ không gian nhớ của Vi xử lý ARM Cortex – M

****I. Chức năng và tổ chức bộ nhớ****

* Tất cả các vi xử lý ARM Cortex – M đều có 32bit địa chỉ bộ nhớ, điều này nghĩa là không gian bộ nhớ tối đa có thể lên đến 4GB
* Toàn bộ 4GB không gian bộ nhớ được đánh địa chỉ liên tục từ 0-4GB và được chia thành các phần khác nhau như vùng nhớ lưu trữ mã code (code memory), vùng nhớ lưu trữ dữ liệu (data memory), vùng nhớ ngoại vi (peripheral memories)…
* Vi xử lý ARM Cortex – M sử dụng kiến trúc bus Harvard, điều này có nghĩa là nó có bus chuyên dụng dành cho việc lấy lệnh (instruction fetch) và lấy dữ liệu (data fetch). Ưu điểm của kiến trúc này là trong khi fetch instruction thì vi xử lý có thể fetch data một cách đồng thời, điều này làm tăng hiệu năng của vi xử lý, vi xử lý không cần phải đợi cho đến khi data được fetch trước khi nó muốn lấy instruction tiếp theo và ngược lại
* Vi xử lý ARM Cortex – M hỗ trợ cả hệ thống bộ nhớ theo kiểu đánh địa chỉ little endian và big endian. Có 1 thanh ghi ở vi xử lý cho phép cấu hình để lựa chọn kiểu đánh địa chỉ. Mặc định và phổ biến thì vi xử lý làm việc với kiểu little endian
* Hỗ trợ unaligned data transfer
* Một số phần của bộ nhớ có địa chỉ dành cho 1 bit dữ liệu (bit-banding). Nghĩa là vi xử lý có khả năng đánh địa chỉ cho 1 bit riêng lẻ trong vùng nhớ
* Vi xử lý cũng có thể hỗ trợ MPU (Memory Protection Unit). Sử dụng tính năng này, người phát triển có thể bảo vệ dữ liệu giữa các vùng nhớ khác nhau. Nó thường được sử dụng trong thiết kế hệ điều hành thời gian thực RTOS và các ứng dụng có yêu cầu an toàn cao.

****II. Bản đồ không gian bộ  nhớ (Memory Map)****

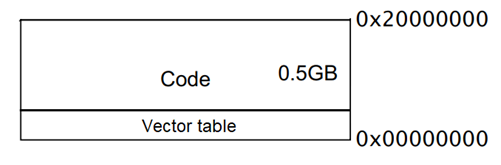


****– Tổng quan****

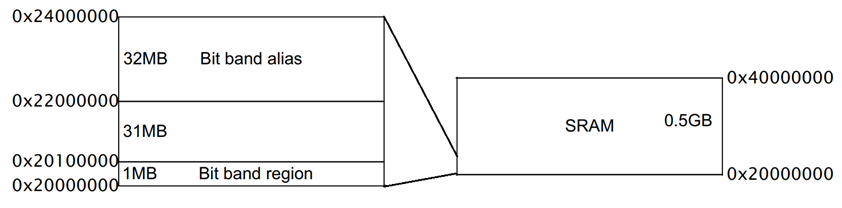
* 4GB không gian bộ nhớ được thiết kế chia thành một số vùng khác nhau. Code, SRAM, Peripherals, External RAM, External Device, Pravite peripheral bus internal and external, Vendor Specific như hình trên.
* Không gian bộ nhớ luôn bắt đầu với vùng code ở địa chỉ 0x0000 0000. Sau đó là vùng nhớ SRAM và các vùng khác như Peripheral và External RAM theo sau đó. Memory Map sẽ nói cho chúng ta biết được tại những vị trí nào của bộ nhớ chứa code và vị trí nào của bộ nhớ chứa dữ liệu hay địa chỉ nào của bộ nhớ bạn có thể truy cập vào các ngoại vi của vi điều khiển.
* Đội ngũ phát triển phần cứng của vi xử lý ARM chịu trách nhiệm cho việc định nghĩa và phân bổ các vùng nhớ khác nhau trong 1 không gian bộ nhớ của vi xử lý. Sau đó, đội ngũ phát triển phần cứng của các hãng vi xử lý như NXP, ST hay TI sẽ chịu trách nhiệm cho việc thêm các bộ nhớ khác nhau của họ như FLASH, ROM, SRAM và các ngoại vi khác nhau như TIMER, ADC, USB… vào các vùng nhớ.

****– Chi tiết một số vùng nhớ:****

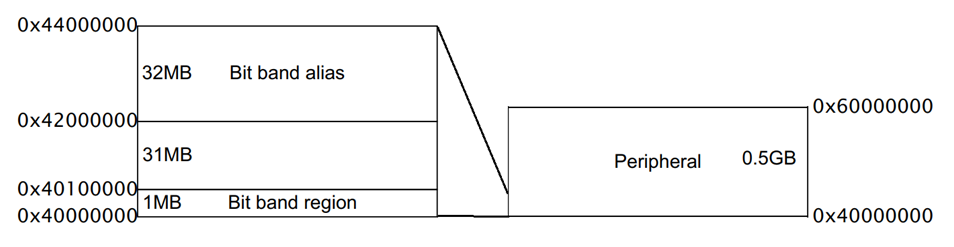
* Code region là vùng bắt đầu từ địa chỉ 0x0000 0000 đến 0x1FFF FFFF, không gian này chứa code của chương trình cùng với bản vector. Tổng cộng, vùng code có kích thước 512 MB. Một cách đơn giản thì bất cứ nhà sản xuất vi điều khiển nào cũng có thể đặt 1 bộ nhớ flash hoặc bộ nhớ ROM với kích cỡ tối đa là 512MB vào vùng nhớ này. Tuy nhiên dung lượng như vậy thì rất lớn, thường thường thì các nhà sản xuất vi điều khiển sẽ giới hạn bộ nhớ code của họ lại khoảng bé hơn 2MB. Một khoảng nhỏ đầu tiên của vùng nhớ từ vị trí 0x0000 0000 sẽ chứa vector table. Vector table này sẽ chứa giá trị khởi tạo stack và địa chỉ của các Exception handler.



* SRAM region hay có thể gọi đơn giản là vùng RAM, vùng này cũng có kích cỡ 512MB ngay sau vùng code. RAM hay SRAM nội (được tích hợp trên vi điều khiển) sẽ được kết nối với vùng này. SRAM được sử dụng để truy xuất data và bộ nhớ stack. Không có giới hạn nào về loại bộ nhớ mà bạn sử dụng để kết nối vào vùng nhớ này, có thể là SRAM hay EEPROM…Trong vùng nhớ này, 1MB đầu tiên được chia thành vùng Bit-band. Mỗi bit trong vùng Bit-band được đánh địa chỉ độc lập. Bạn cũng có thể thực thi code từ vùng nhớ SRAM, Vi xử lý không giới hạn việc thực thi chương trình từ đây.



* Peripherals Region, nhà thiết kế vi điều khiển có toàn quyền quyết định những ngoại vi nào được sử dụng và dùng vùng nhớ này cho những ngoại vi đó. Vùng nhớ này cũng có kích cỡ 512MB. Những ngoại vi thường được sử dụng nhất cho vi điều khiển như GPIO, ADC, TIMER, RTC, các bộ giao tiếp nối tiếp…vv. Để rõ hơn thì bạn bạn nên mở reference manual của vi điều khiển bạn đang sử dụng lên và kiểm tra Memory Map, bạn sẽ thấy được nhà cung cấp sẽ đưa rất nhiều ngoại vi khác nhau vào vùng nhớ này. Vi xử lý ngăn cản việc thực thi code từ vùng này, vì vậy bạn sẽ không thể thực thi bất cứ mã lệnh nào từ vùng Peripheral.



* External RAM Region: Nếu như bộ nhớ RAM được tích hợp sẵn trên vi điều khiển không đủ và bạn muốn kết nối với 1 bộ nhớ RAM ngoài thì bạn có thể khai thác được chức năng của vùng nhớ này*.*



* External Device Region: Vùng nhớ này có thể được sử dụng để kết nối với các thiết bị bên ngoài như EEPROM, bộ nhớ chia sẻ của các IC chức năng. Vi xử lý cũng không cho phép thực thi code từ vùng nhớ này… Và một số vùng nhớ khác các bạn có thể tham khảo thêm chi tiết từ tài liệu của ARM và hãng sản xuất Vi điều khiển.