< BACK Make Note | Bookmark CONTINUE >

Memory Organization

IOS maps the entire physical memory into one large flat virtual address space. The CPU's MMU is used when available to create the virtual address space even though IOS doesn't employ a full virtual memory scheme. To reduce overhead, the kernel does not perform any memory paging or swapping, so virtual address space is limited to the bounds of the physical memory available.

IOS divides this address space into areas of memory called *regions,*which mostly correspond to the various types of physical memory. For example, SRAM might be present for storing packets and DRAM might be present for storing software and data on a given type of router. Classifying memory into regions allows IOS to group various types of memory so software needn't know about the specifics of memory on every platform.

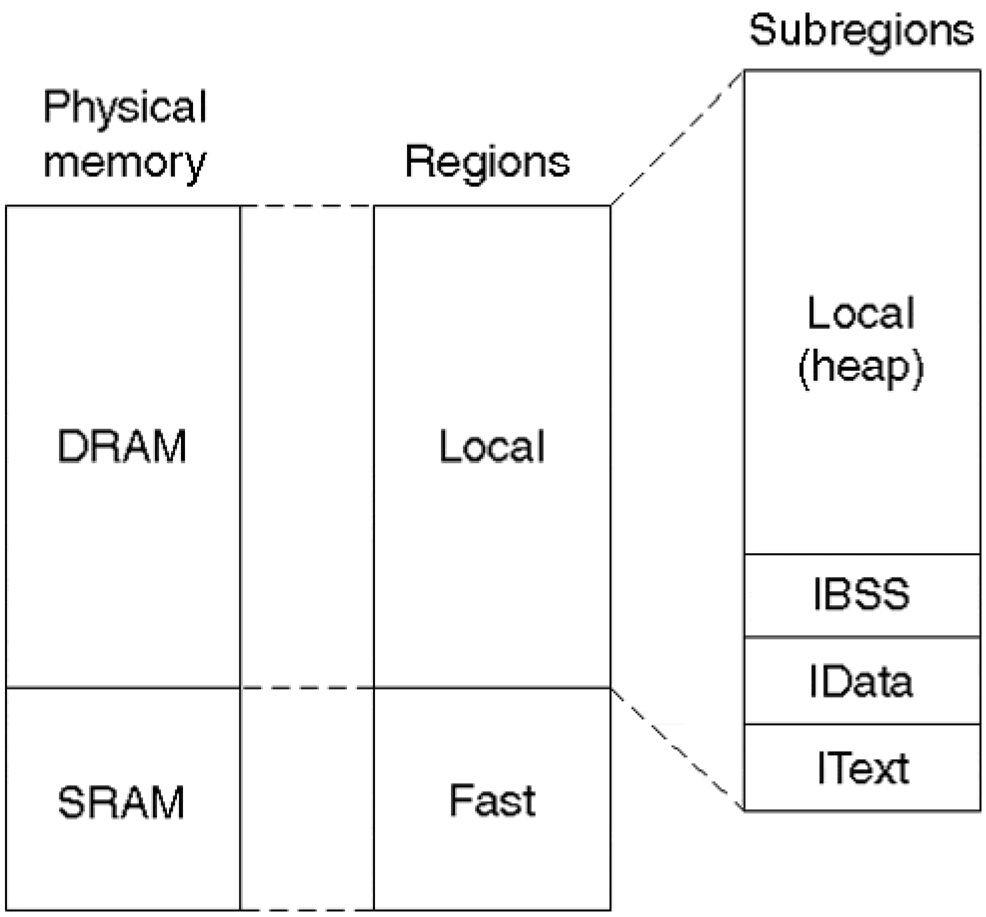
Memory regions are classified into one of eight categories, which are listed in Table 1-1.

Table 1-1. Memory Region Classes

Memory Region Class	Characteristics
Local	Normal run-time data structures and local heaps; often DRAM.
Iomem	Shared memory that is visible to both the CPU and the network media controllers over a data bus. Often is SRAM.
Fast	Fast memory, such as SRAM, used for special-purpose and speed-critical tasks.
IText	Executable IOS code.
IData	Initialized variables.
IBss	Uninitialized variables.
PCI	PCI bus memory; visible to all devices on the PCI buses.
Flash	Flash memory. This region class can be used to store run-from-Flash or run-from-RAM IOS images. It often also can be used to store backups of the router configuration and other data, such as crash data. Typically, a file system is built in the Flash memory region.

Memory regions also can be nested in a parent-child relationship. Although there is no imposed limit on the depth of nesting, only one level is really used. Regions nested in this manner form *subregions* of the parent region. Figure 1-2 shows a typical platform virtual memory layout and the regions and subregions IOS might create.

Figure 1-2. Memory Regions



The IOS EXEC command **show region** can be used to display the regions defined on a particular system as demonstrated in Example 1-1 (taken from a Cisco 7206 router).

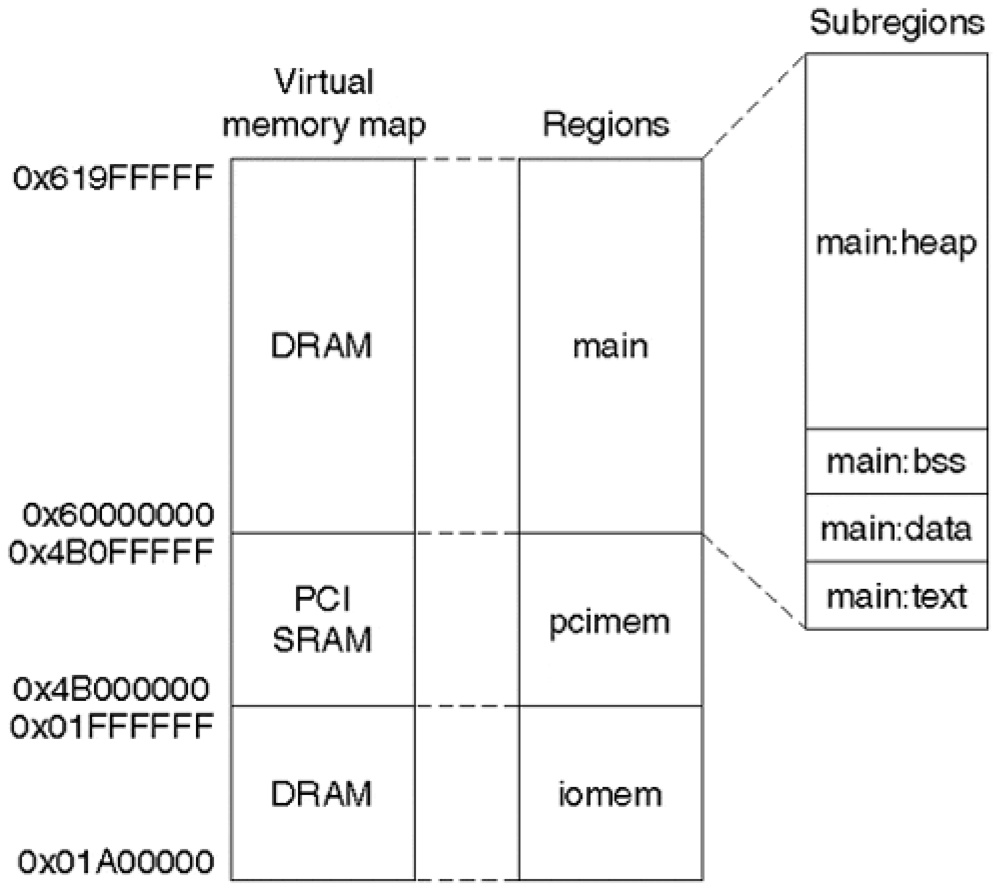
Example 1-1. *show region* Command Output

```
router#show region Region Manager: Start End Size(b) Class Media Name 0x01A00000 0x01FFFFFF
6291456 Iomem R/W iomem 0x31A00000 0x31FFFFFF 6291456 Iomem R/W iomem:(iomem_cwt)
0x4B000000 0x4B0FFFFF 1048576 PCI R/W pcimem 0x60000000 0x619FFFFF 27262976 Local R/W main
0x600088F8 0x61073609 17214738 IText R/O main:text 0x61074000 0x611000FF 573696 idata R/W
main:data 0x61100100 0x6128153F 1578048 IBss R/W main:bss 0x61281540 0x619FFFFF 7858880 Local
R/W main:heap 0x7B000000 0x7B0FFFFF 1048576 PCI R/W pcimem:(pcimem_cwt) 0x80000000
0x819FFFFF 27262976 Local R/W main:(main_k0) 0xA0000000 0xA19FFFFF 27262976 Local R/W main:
(main_k1)
```

On the left, the **Start** and **End** addresses correspond to parts of the platform virtual memory map. On the right are the regions and subregions. Subregions are denoted by a name with the **:** separator and no parentheses.

Figure 1-3 illustrates this memory map and its regions.

Figure 1-3. Memory Map and Regions



The gaps between the address ranges—for example, **pcimem** ends at **0x4B0FFFFF** and **main** begins at **0x60000000**—are intentional. These gaps allow for expansion of the regions and provide a measure of protection against errant threads. If a runaway thread is advancing through memory writing garbage, it is forced to stop when it hits a gap.

From the output in Example 1-1 and Figure 1-3, you see that the entire DRAM area from 0x60000000 to 0x619FFFFF has been classified as a **local** region and further divided into subregions. These subregions correspond to the various parts of the IOS image itself (text, BSS, and data) and the heap. The heap fills all the local memory area left over after the image is loaded.

Some regions appear to be duplicates, only with a different address range, such as **iomem:(iomem cwt)**:

```
     Start          End       Size(b)   Class   Media   Name
0x01A00000   0x01FFFFFF     6291456   Iomem   R/W     iomem
0x31A00000   0x31FFFFFF     6291456   Iomem   R/W     iomem:(iomem_cwt)
....
```

These duplicate regions are called *aliases*. Some Cisco platforms have multiple *physical* address ranges that point to the same block of physical memory. These different ranges are used to provide alternate data access methods or automatic data translation in hardware. For example, one address range might provide cached access to an area of physical memory while another might provide uncached access to the same memory.

The duplicate ranges are mapped as alternate views during system initialization and IOS creates alias regions for them. Aliased regions don't count toward the memory total on a platform (because they aren't really separate memory), so they allow IOS to provide a separate region for alternate memory views without artificially inflating the total memory calculation.

Memory Pools

IOS manages available free memory via a series of *memory pools*, which are essentially heaps in the generic sense; each pool is a collection of memory blocks that can be allocated and deallocated as needed. Memory pools are built out of regions and are managed by the kernel. Often, the pools correspond one-to-one to particular regions, but they are not required to. A memory pool can be built from memory spanning several regions, allowing memory to be allocated and reclaimed from various areas for maximum efficiency.

You can obtain information on IOS memory pools by using the **show memory** command as demonstrated by Example 1-2.

Example 1-2. IOS Memory Pool Information in *show memory* Command Output

router#**show memory** Head Total(b) Used(b) Free(b) Lowest(b) Largest(b) Processor 61281540 7858880 3314128 4544752 4377808 4485428 I/O 1A00000 6291456 1326936 4964520 4951276 4964476 PCI 4B000000 1048576 407320 641256 641256 641212 ...

NOTE

The output from **show memory** can be very long. It should *not* be issued while console output paging is disabled (the terminal length set to 0) because there is no way to stop or pause the output until it completes.

In Example 1-2, there are three memory pools: **Processor**, **I/O**, and **PCI**. Comparing the **Head** column in this output with the **Start** column from the output of **show region** in Example 1-3 allows you to see which regions are contained within each memory pool.

Example 1-3. *show region* Command Output

router#**show region** Region Manager: Start End Size(b) Class Media Name 0x01A00000 0x01FFFFFF 6291456 Iomem R/W iomem 0x31A00000 0x31FFFFFF 6291456 Iomem R/W iomem:(iomem_cwt) 0x4B000000 0x4B0FFFFF 1048576 PCI R/W pcimem 0x60000000 0x619FFFFF 27262976 Local R/W main 0x600088F8 0x61073609 17214738 IText R/O main:text 0x61074000 0x611000FF 573696 idata R/W main:data 0x61100100 0x6128153F 1578048 IBss R/W main:bss 0x61281540 0x619FFFFF 7858880 Local R/W main:heap ....

From Example 1-3, you can see the Processor memory pool contains memory from the subregion of **main**, called **heap**, which is in class **Local**. The Processor memory pool is common to all IOS systems and always resides in local memory. This is the general memory pool from which data is allocated (such as routing tables).

The **I/O** pool is managing memory from the **iomem** region and the **PCI** pool is managing memory from the **pcimem** region.

The remaining fields in the **show memory** output in Example 1-2 provide useful statistics about the pools as documented in the following list. All units are in bytes.

- **Total—**

  Total size of the pool.

- **Used—**

  Current amount of memory allocated.

- **Free—**

Current amount of memory available.

- **Lowest—**

  The least amount of memory ever available since the pool was created.

- **Largest—**

  The size of the largest contiguous block of memory currently available.

The **show memory** command can also display the blocks within each memory pool, as you'll see later in this chapter.

< BACK Make Note | Bookmark CONTINUE >

Index terms contained in this section