ESP Memory usage

Memory

Location	Description
Bootloader	internal ROM, size 64KB, contains bootloader to access the external flash.
Flash	external flash memory, size depending on the board variant between 512 KB and 4 MB.
IRAM	Instruction RAM, static RAM, total size 64 KB. 32KB contains persistent code that needs to execute quickly or compete with other Flash read routines (such as Interrupt Service Routines (ISR)), this area is populated from Flash at boot time. 32 KB, divided into two 18 KB blocks, serves as an instruction cache which is loaded from the Flash at runtime with the respective current code.
DRAM	Data RAM, static RAM, size 96 KB, contains the program data. 16 KB are already consumed by the Sytems RAM block, so that about 80 KB are still available for the application. In the Arduino environment, the SDK already consumes about 32 KB. For application data (static data, stack and heap) thus remain about 48 KB.

Sections

The following sections are relevant:

Sector	Description
.data	This section contains “initialized” data, i.e. which needs copying from FLASH to DRAM on powerup.
.rodata	This section contains “read only” data, like constant data, e.g. strings: Serial.print (“Hello”); ⟩ “Hello” ⇒ rodata. When booting, this area is copied from FLASH to DRAM. Which is kind of strange, because it is read only and could be in flash as well. But maybe this is done for speed.
.bbs	Section used for uninitialized variables, e.g. int i ;. When booting, this area is filled with 0 (zero).
.irom0.text	This section is used for uncached code and is located in FLASH. The currently required parts are copied to one of the two instruction cache blocks (16 KB each) at runtime and executed there.
.text	Section used for code, which is transferred to IRAM at boot time and executed there. The code remains permanently in the IRAM. It is code that needs to be executed quickly or in competition with the other reading routines of Flash (e.g., interrupt service routines (ISR)). The maximum size is 32 KB. Due to this size limitation we usually end up using a directive to put code into FLASH to save space – but REALLY time critical code should be left in RAM. Of course it does get copied into RAM (cacheing) but that could mean execution times which are different first time around.

The section sizes can be obtained by running:

```
xtensa-lx106-elf-objdump -h -j .data -j .rodata -j .bss -j .text -j
.irom0.text controller
```

Which results in something like this:

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .data         0000087e  3ffe8000  3ffe8000  000000e0  2**4
                  CONTENTS, ALLOC, LOAD, DATA
  1 .rodata       00005494  3ffe8880  3ffe8880  00000960  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .bss          000066d8  3ffedd18  3ffedd18  00005df8  2**4
                  ALLOC
  3 .irom0.text   0003a96e  40201010  40201010  0000cc40  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  4 .text         00006e3d  40100000  40100000  00005df4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
```

Heap

There are no individual limits for the sizes of the sections. The ESP8266 has 80K of RAM which is occupied by **.data**, **.rodata**, **.bss**, the **stack** and the **heap**. Basically, the space represented by (80K - sizeof(.data) - sizeof(.rodata) - sizeof(.bss)) is used by the heap growing.