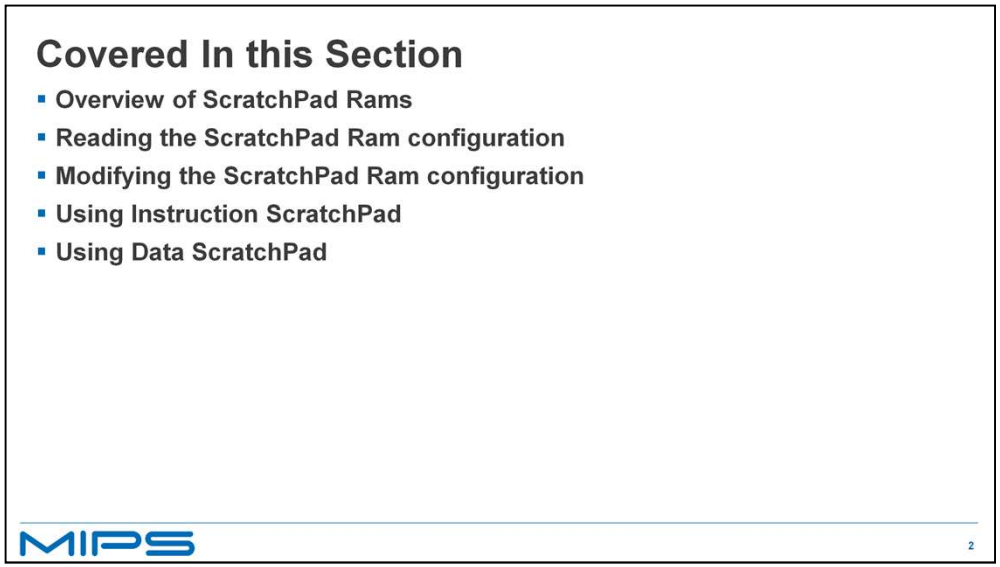

MIPS
MIPS Software Training
Scratch Pad Memory
www.mips.com



This ScratchPad section will cover

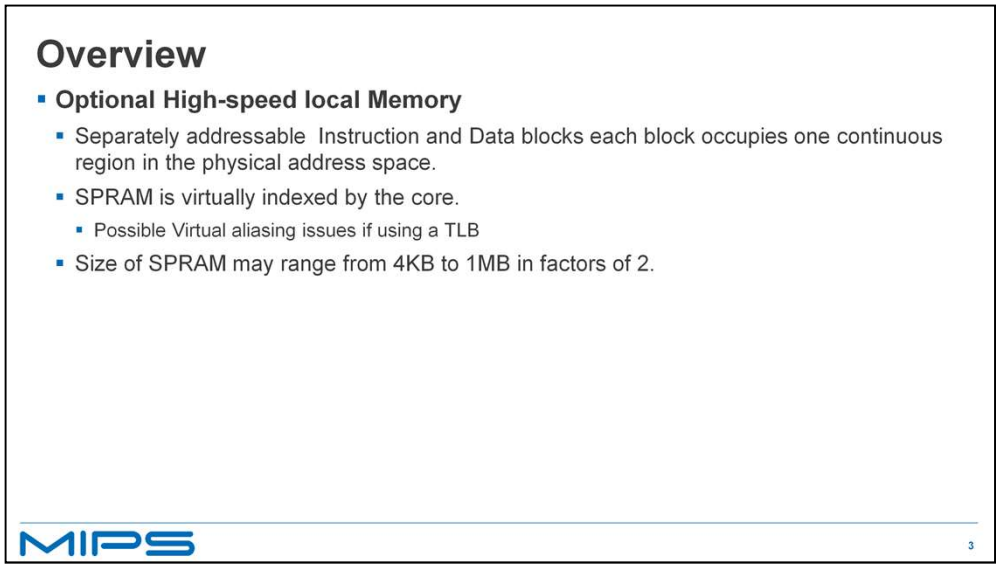
+ An overview of what ScratchPad ram is are and some issues to consider

+ How to get the current configuration of the ScratchPad blocks. This will show you how to get the current address the block starts from, how big the block is and if it is enabled or not.

+ Then I'll go into how you can change the starting address of the ScratchPad blocks and enable or disable them.

+ I'll detail how you would load code into the instruction ScratchPad

+ and last how you would use the Data ScratchPad



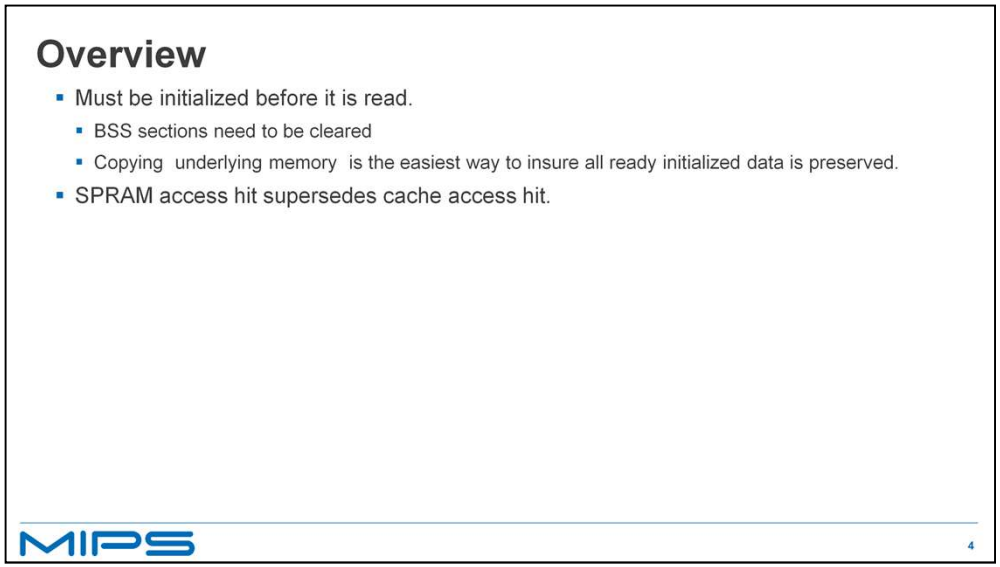
Most processor Cores support the option of adding high-speed local memory blocks during the building of the core. These blocks are referred to as ScratchPad RAM (or SPRAM). They provide low-latency storage for critical code or data. SPRAM access speed is similar to that of locked cache lines, but without impact on cache performance or maintenance.

+ The Instruction and Data ScratchPad Ram blocks are addressed separately from each other and from main memory, so it is possible to for the ScratchPad addresses to overlap each other and main memory. There can be only one continuous physical address range for each Instruction or Data ScratchPad block.

+ The SPRAM array, like the cache arrays, is indexed with a virtual address and the “tag comparison” is performed using a physical address. Since the SPRAM size can be larger than the 4KB minimum page size, it is possible to have virtual aliasing in the SPRAM if using a TLB. Virtual aliasing occurs when a single physical address is accessed via two different virtual addresses that can simultaneously reside in memory. This is not handled by hardware and programmers must be aware of it. One way around this would the to make one TLB entry that covers the whole

ScratchPad memory by making the page size for the TLB entry the same size as the scratchpad RAM.

+ ScratchPad blocks can be as little as 4K and as large as 1 megabyte.



Software must ensure a SPRAM entry has been initialized the same as regular memory before it is read, to avoid reading spurious data.

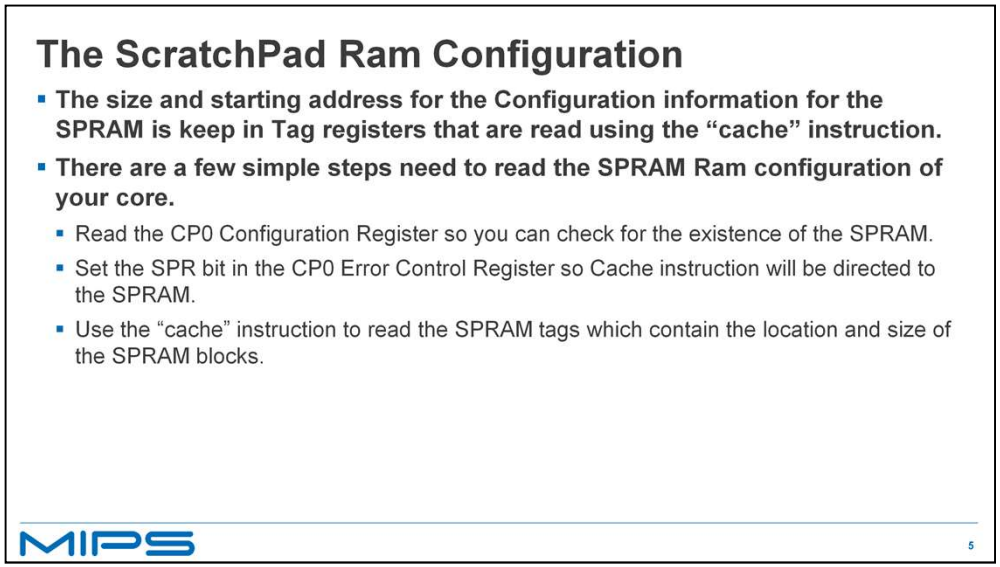
+ In most systems the BSS section is cleared when the processor is being initialized or a process is first brought into memory so if you configure your data SPRAM after this clear step to an addresses that covers the BSS section you will need to clear the area again.

+ The easiest thing to do if you are overlapping main memory with ScratchPad memory is to copy the underlying main memory to the SPRAM memory, that way you would be copying already initialized values in main memory to your SPRAM memory.

+ One thing to note: if the area that the Data SPRAM memory is replacing was already in use and it was cached you should first flush and invalidate any cache lines that correspond to the physical memory being overlaid. Since SPRAM access supersedes the data cache hit it will be impossible

to reference the data cache at the overlaid addresses.

It is also possible if you are using a TLB to have a virtual address mapped to a cached area and another virtual address mapped through the SPRAM but both using the same physical memory. In which case you could have updated values in either the cache or SPRAM memory one not seen by the other so you need to avoid this situation.



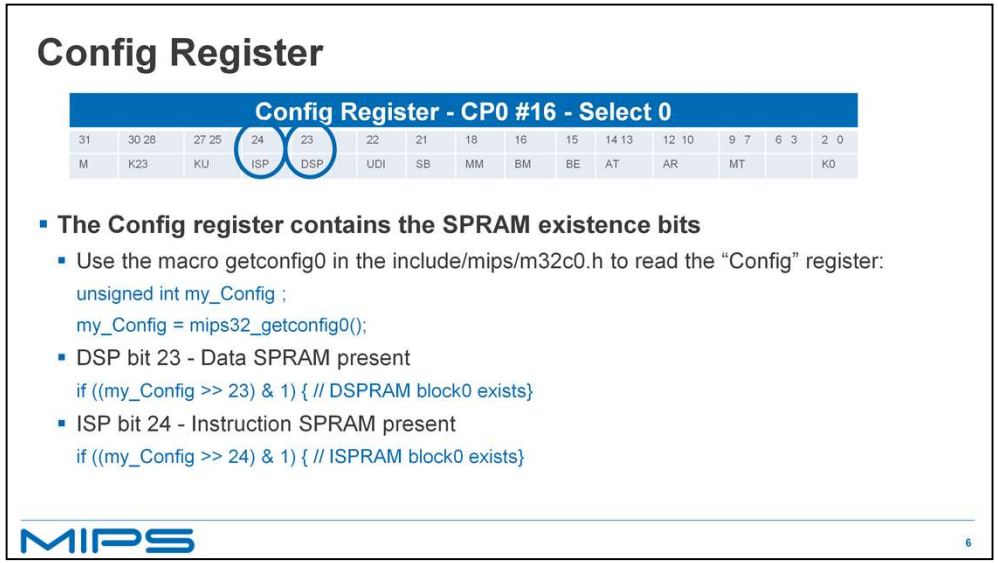
Configuration information for the ScratchPad blocks are keep in the SPRAM tag register.

+ I’ll go into details on how you would go about reading these from your C code but first I’ll give you an overview of the process.

+ First the code should read the CP0 Configuration register to see if the Data or Instruction SPRAM blocks exist on your core.

+ Then to prepare to issue cache instructions to the SPRAM interface you need to set the SPR bit in the CP0 Error Control Register so CACHE instructions operate on the SPRAM instead of the Caches.

+ last you need to read the SPRAM tags to get the location and size for the SPRAM blocks.

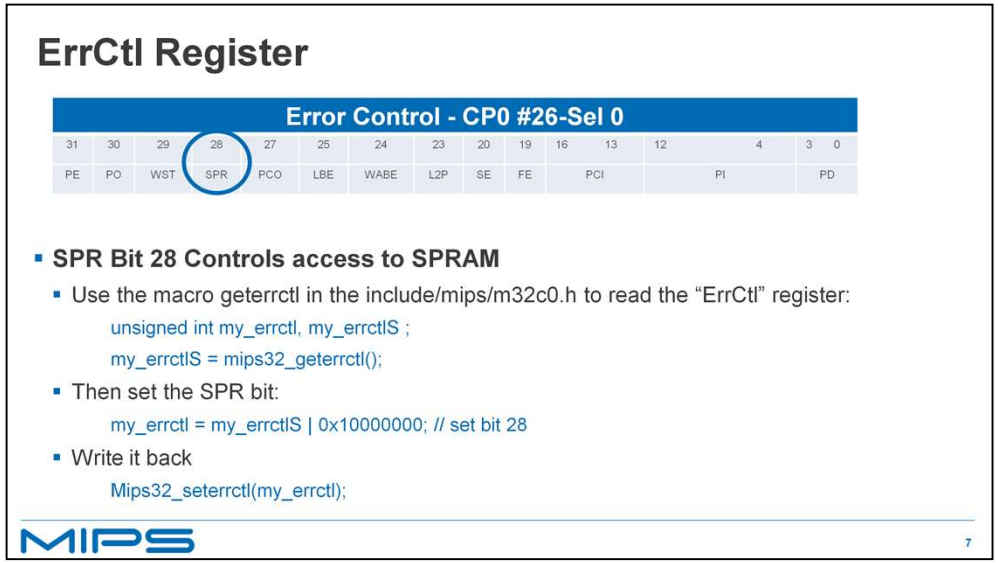


The CP0 Config register contains the SPRAM configuration bits

+ Use the macro called getconfig0 in the include/mips/m32c0.h file to read the "Config" register from C code

+ Bit 23, the DSP Bit tells you there is a Data SPRAM block.

+ Bit 24, the ISP bit tells you there is a Instruction SPRAM block.

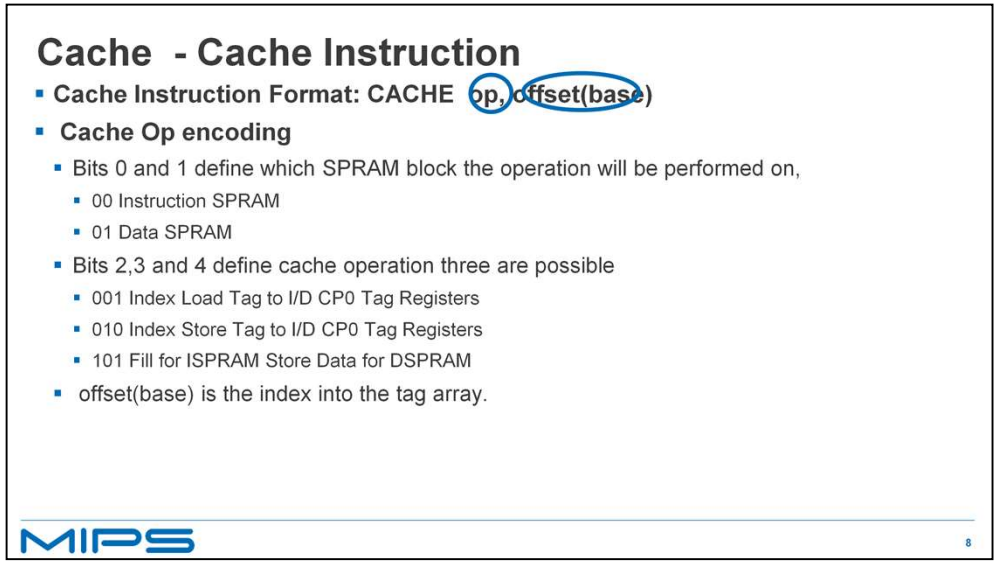


Next you need to Set the SPR bit in the CP0 Error Control Register so Cache instruction will be directed to the SPRAM.

+ Use the macro called geterrctl in the include/mips/m32c0.h file to read the "Error Control" register from C code. Usually you want to save this value off so it can be restored latter.

+ Set bit 28 the SPR bit

+ and write it back



Use the cache instruction to read the SPRAM tags to get the size and starting address information.

+ Here is the instruction format

+ the cache op is encoded with 2 pieces of information

+ bits zero and one tell determine which SPRAM block the operation will be performed on

+ zero zero sets it for the Instruction SPRAM
+ zero one sets it for the Data SPRAM

+ Bits two, three and four tell the instruction which operation to perform
+ zero zero one will load a tag

+ zero one zero will store a tag

+ and one zero one will fill data into the SPRAM blocks memory

+ The offset and base register control which of the 2 possible tags the load or store operation will be performed on or which address within the SPRAM block will be filled.

TagLo Tag Registers for microAptiv Cores

TagLo, CP0 #28-Sel 0
I or D Tags

tag	31	10	9	8	7	6	1	0
0	Physical Base Address		Unused		E	0		P
1	Size		Unused					

- **Tag 0 (offset 0) - Enable bit 7 and Base Address**

```
unsigned int my_TagLo ;
index_load_tag_i (0) ;                  // read the SPRAM tag 0 into the I tag lo register
my_TagLo = getitaglo() ;                // read the I tag lo register into my_TagLo
base_paddr = my_TagLo & 0xfffffc00 ;    // extract the SPRAM base address
*enabled = (my_TagLo >> 7) & 1 ;        // set the enable bit
```

- **Tag 1 (offset 4) - Size in cache lines sections10 – 31**

```
index_load_tag_i(4) ;                   // read the SPRAM tag 1 into the I tag lo register
my_TagLo = getitaglo() ;                // read the I tag lo register into my_TagLo
block_size = ((my_TagLo >> 10) * 16);   // extract the size of the SPRAM
```

MIPS 9

For a microAptivCores the cache instruction will use the TagLo register to read or write both the instruction and data tags.

Each SPRAM block contains two tag Registers

+ Tag 0 contains the SPRAM Enable bit which must be set before you write or read data from the SPRAM block and the physical address bits 10 through 31 of the SPRAM block. These address bits will be set on bring up to the default values the core was configured to at build time. You may change this address to any that is aligned on a page size address which is 16 bytes.

+ To read the Tag 0 use the getitaglo macro from m32c0.h.

+ Mask out the address field

+ Shift the enable bit into bit 0 and mask it out

+ Tag 1 contains the number of cache line size sections of the SPRAM block in bits 10 through 31

+ To get the block size in bytes shift the size to the left by 10 bits and multiply by the line size of 16 bytes.

For all other cores the cache instruction will use the DTaglo register for data tags and the ITagLo register for instruction tags

The SPRAM Data tag registers are read or written using the CP0 DTagLo register.

And the SPRAM Instruction Tag registers are read or written using the CP0 ITagLo register

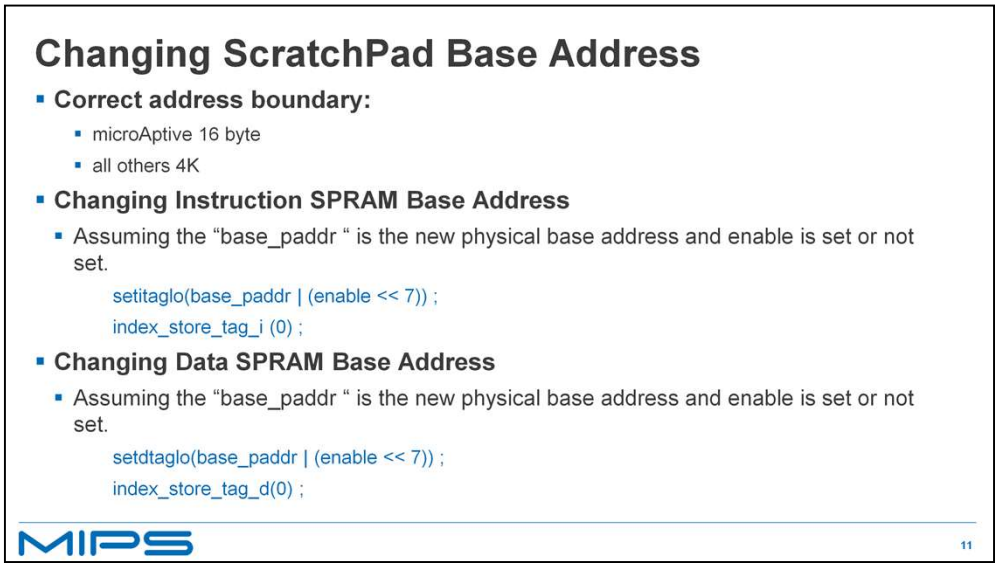
Each ScratchPad block contains two tag Registers

+ Tag 0 contains the SPRAM Enable bit which must be set before you write or read data from the SPRAM block and the physical address bits 12 through 31 of the ScratchPad block. These address bits will be set on bring up to the default values the core was configured to at build time. You may change this address to any aligned 4K address.

+ Write the tag to the tag lo register using the **index_load_tag_i or d with a offset of 0 and copy the register to a variable using** getitaglo or getdtaglo macro both macros are from m32c0.h.

+ Mask out the address field
+ Shift the enable bit into bit 0 and mask it out

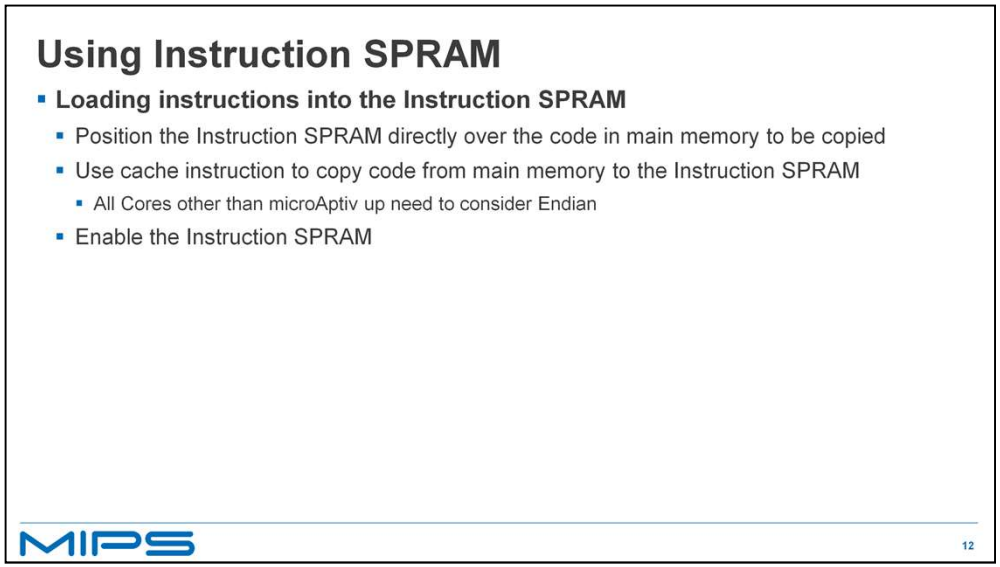
+ Tag 1 contains the number of 4K sections of the SPRAM block in bits 12 through 19
+ Get the tag the same as before but this time use offset 8
+ use the value for the block size in bytes



You don't need to stick with the default physical base address that was set at core build time. You can change it to anywhere in memory on correct boundary for your core

+ Using the assemble macros set the new address in the tag register

+ and the store the tag.



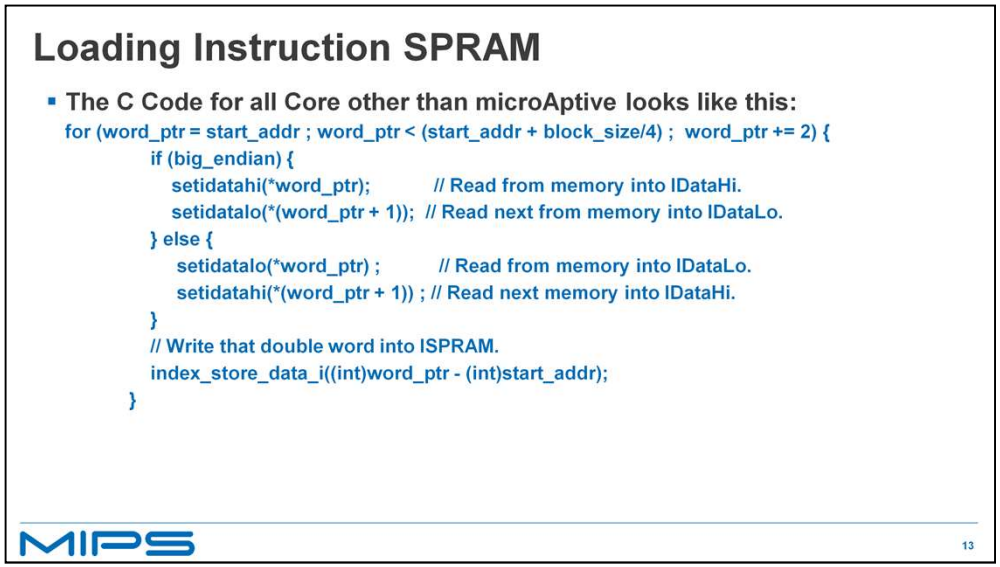
To get your code into the scratchpad, the instruction SPRAM block needs to be loaded from main memory using the cache instruction, index store Data.

+ The best way I have found to do that is to load the code into main memory then position the Instruction SPRAM Block directly over it.

+ While the Instruction SPRAM Block is disabled use the cache instruction to copy the code from main memory to it.

+ for 24K core and up the is Endian to consider because of the use of 2 data tag register so the code will switch the order of the reads into these registers depending on the Endian. For a M14Kc and 4KE there is only one tag register so there are no Endian issues.

+ then enable the Instruction SPRAM and its ready to use

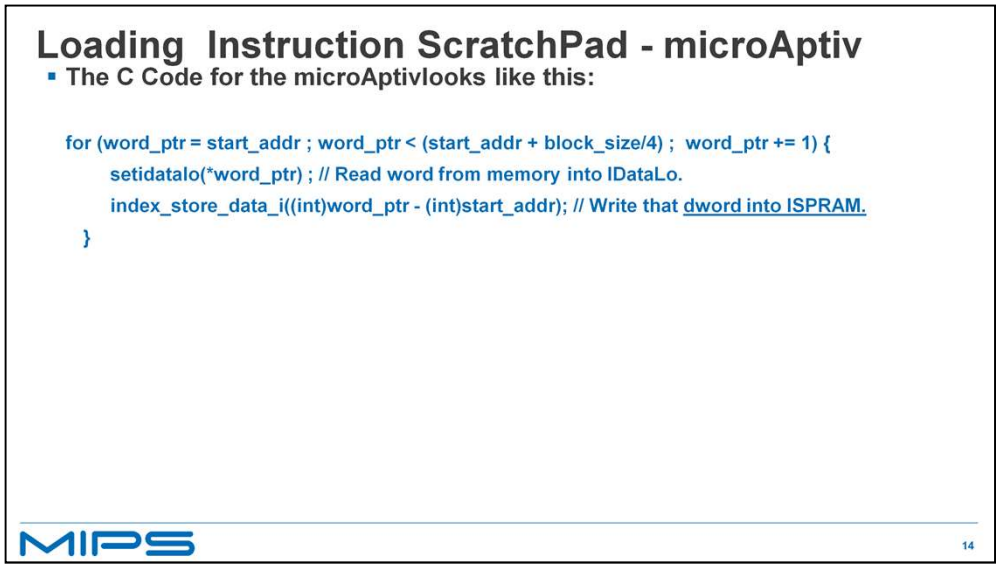


This code assumes you have already positioned the Instruction SPRAM directly over the code in main memory so that the starting address variable start_addr is the starting address of the code to be copied in main memory and also the starting address of the Instruction SPRAM Block. The starting address is also assumed to be a kseg0 cacheable address. The block size is the size of the Instruction SPRAM block.

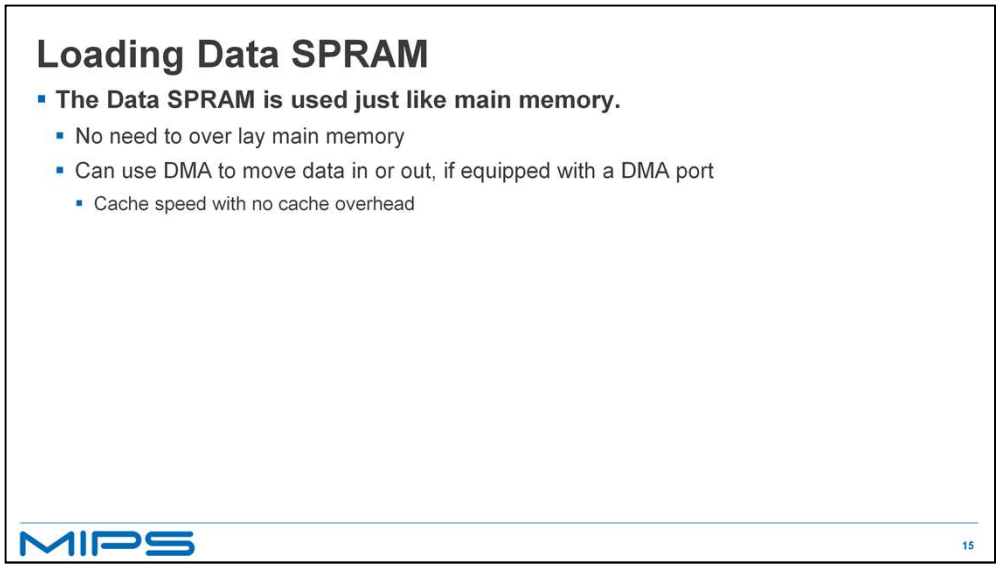
+ As you can see the code needs to make necessary adjustment for endianness. This is due to the bus size of 64bits. The macros setidatahi and setidatalo are used to load the CP0 C0_TAGHI and C0_TAGLO registers

+ then the indexe_stor_data_i fills the instruction SPRAM 2 full words at a time.

The Instruction SPRAM could also be loaded using DMA requests if equipped with a DMA port.



The code for the microAptiv cores it is simpler because there is no need to account for Endian because only 32 bits are written at a time.



The Data SPRAM RAM block appears to your program as normal physical memory so there is nothing special you need to do once the Block is enabled.

+ there is no need to have underlying main memory when you are using the Data SPRAM. You can configure the block anywhere in the physical memory map including a non cached memory addresses.

+ Configuring the Data SPRAM RAM at core build time with a DMA interface makes it especially efficient as a staging area for communicating with complex I/O devices. It's a great way to implement a "push" style I/O, where the device writes incoming data close to the CPU.

+ One other advantage in using Data SPRAM for DMA buffers is you don't have any cache management issues such as flushing and invalidating cache lines.