Registers

Embedded Software Essentials

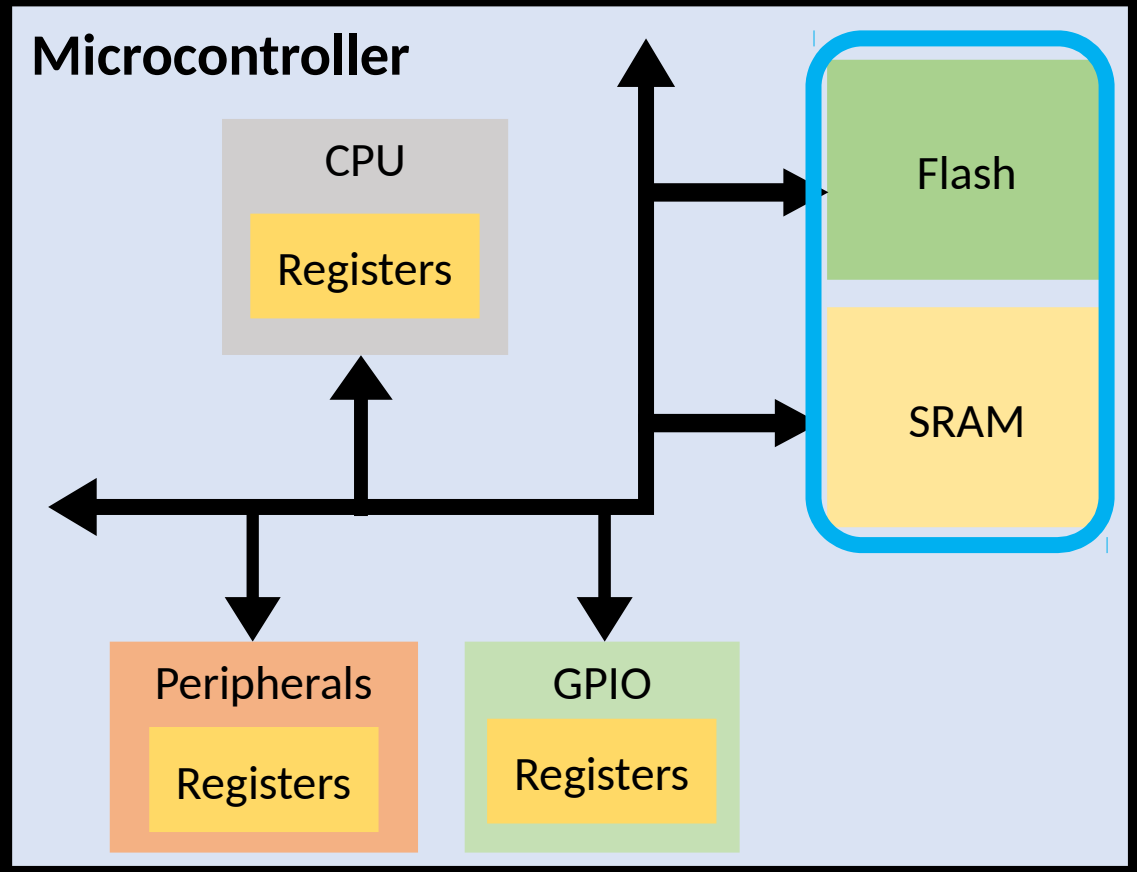
C2M1V8

Embedded System Memories [S1]

- Memories of an Embedded Systems
  - Code Memory (Flash)
  - Data Memory (SRAM)
  - Register Memory (internal to chip)
  - External Memory (if applicable)

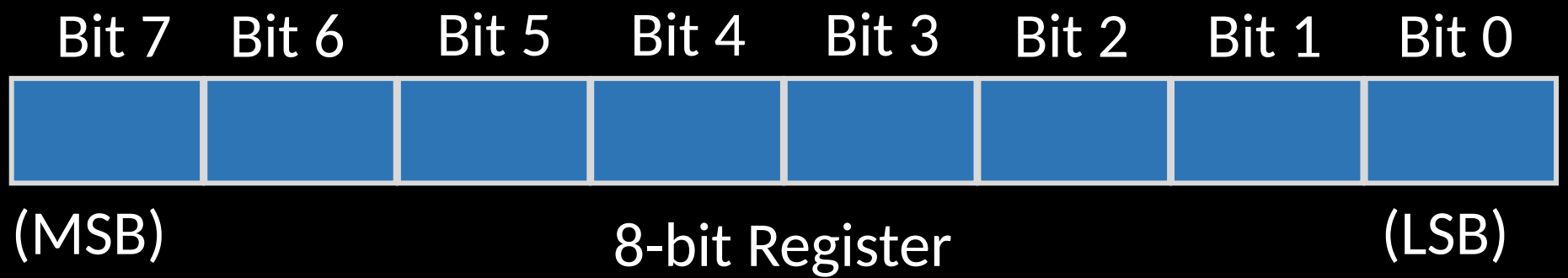
- Compilation tracks and maps memory from a program into segments

Specified in the Linker File



Registers [S1a]

- Registers store temporary data
  - Can Be Read/Write or Read-Only
- Typically divided into multiple bit fields



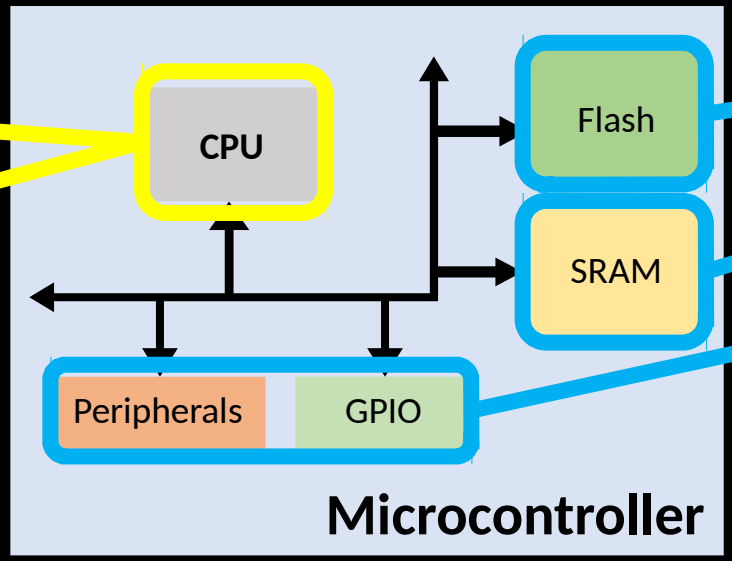
Registers [S1b]

- Registers store temporary data
  - Can Be Read/Write or Read-Only
- Typically divided into multiple bit fields

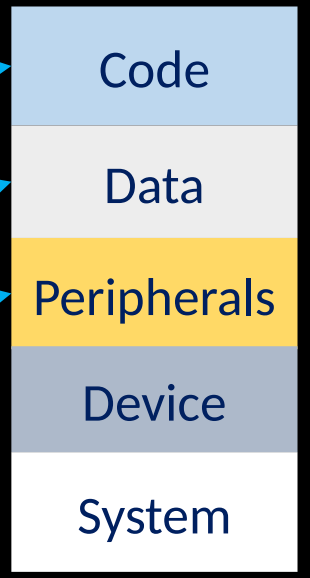
Core CPU Registers

R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
SP (R13)
LR (R14)
PC (R15)

PSR
PRIMASK
FAULTMASK
BASEPRI
CONTROL



Memory Map



Memory Map [S2]

- **Memory Map**: Provides a memory address to physical device mapping within an address space for use in programming

Mapped Components (Memory, Peripherals, System Config, etc.)

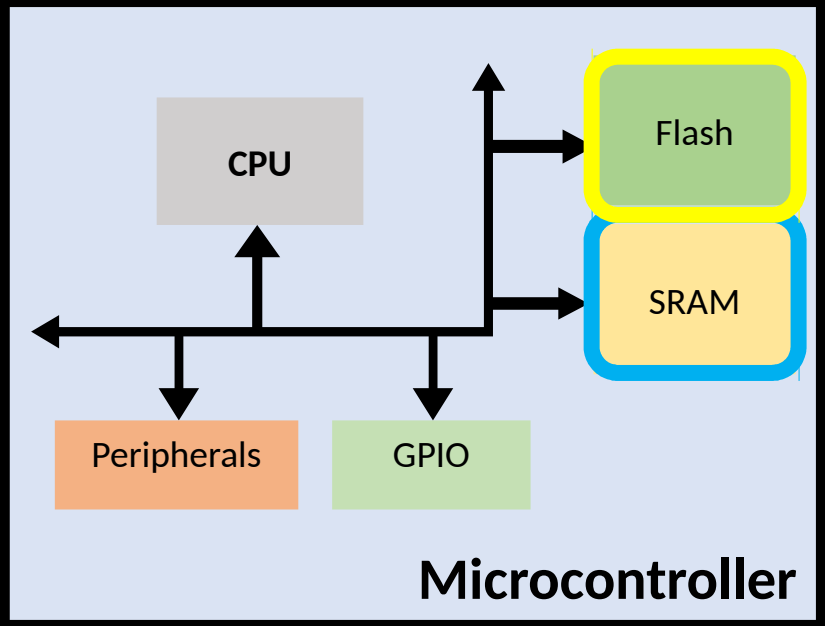
Memory region	Description	Access via	Address range
Code	Normally flash SRAM or ROM	ICode and Dcode bus	0x00000000-0x1FFFFFFF
SRAM	On-chip SRAM, with bit-banding feature	System bus	0x20000000-0x3FFFFFFF
Peripheral	Normal peripherals, with bit-banding feature	System bus	0x40000000-0x5FFFFFFF
External RAM	External memory	System bus	0x60000000-0x9FFFFFFF
External device	External peripherals or shared memory	System bus	0xA0000000-0xDFFFFFFF
Private peripheral bus	System devices, see Table 2-3 on page 2-25	System bus	0xE0000000-0xE00FFFFF
Vendor specific	-	-	0xE0100000-0xFFFFFFFF

Address Space (Ranges for each device)

Embedded Memory [S3]

- Compilation tracks and maps memory from program code and program data into segments
  - Code Segment (Flash)
  - Data Segment (SRAM)

Specified in the Linker File



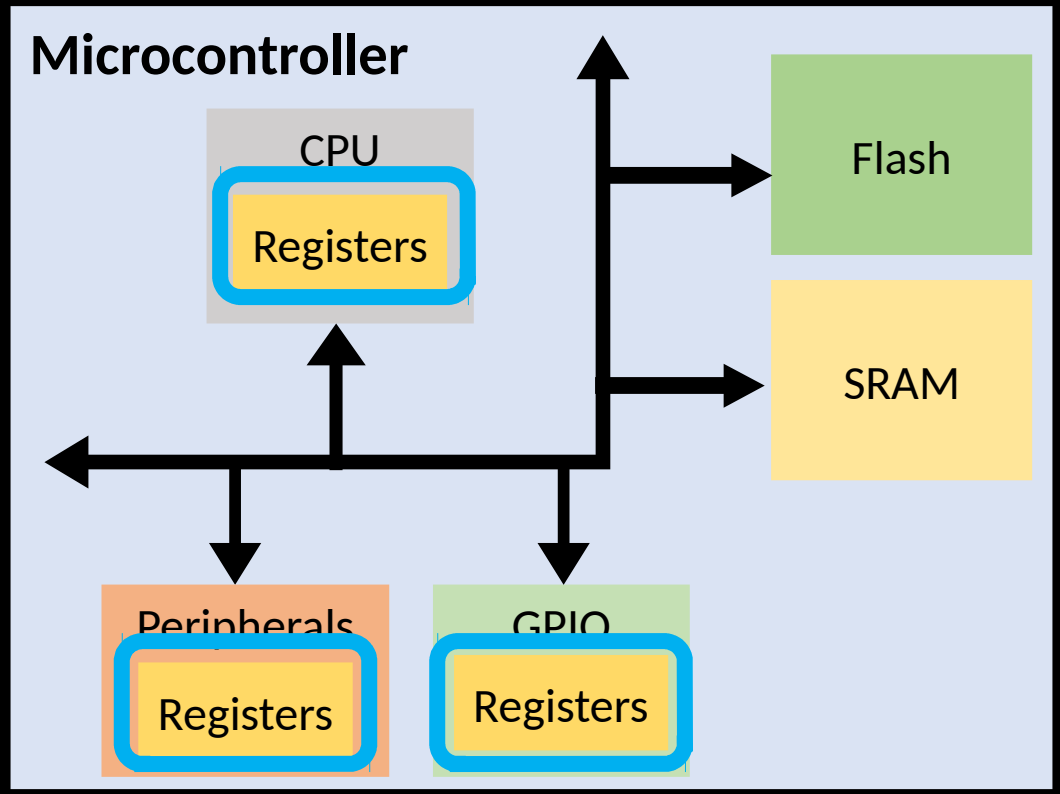
- Memory allocation is application dependent
  - Exact address allocation changes at every compile

```
MEMORY
{
   MAIN  (RX) : origin = 0x00000000, length = 0x00040000
   DATA  (RW) : origin = 0x20000000, length = 0x00010000
}
```

Register Memory [S4a]

- Registers from Peripherals, System and Vendor specific regions have immutable addresses

- Typical Register Memories
  - Internal Core CPU
  - Internal Private Peripherals
  - External Private Peripherals
  - General Peripheral Memory



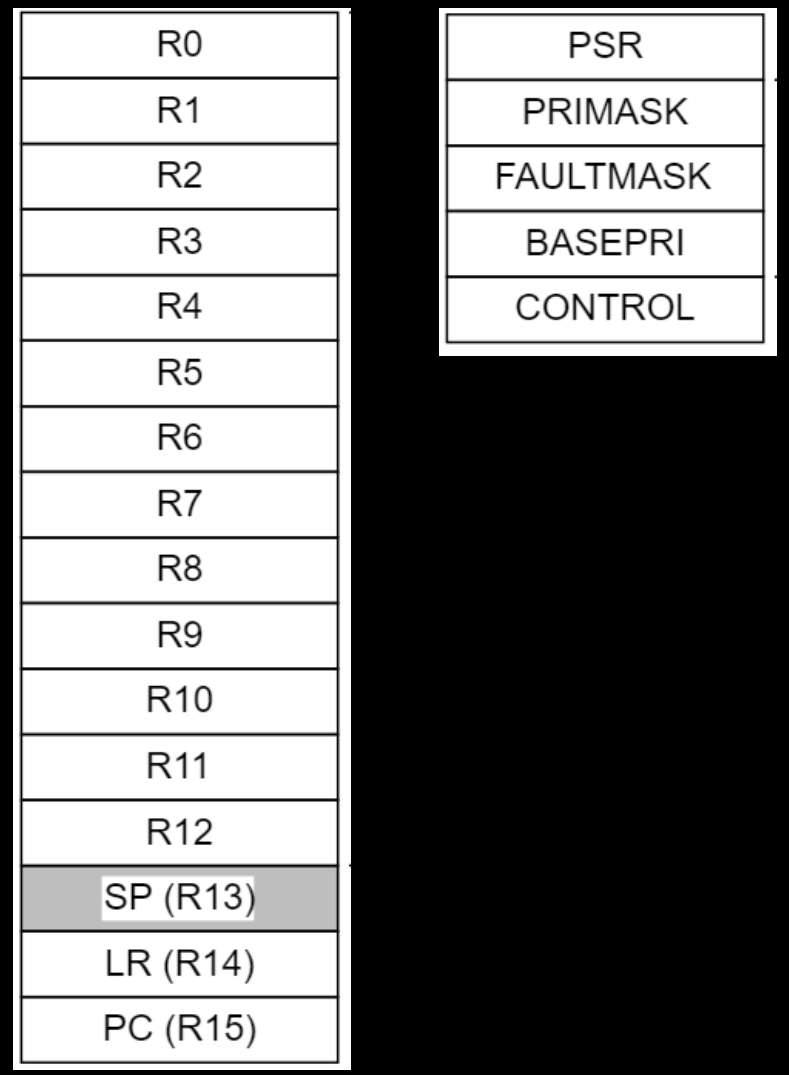
Examples:

- **CPU Core Register Identifiers: r0-r16**
- **Timer A0 Control Register Address: 0x40000000**

Core CPU Registers [S4b]

- Cortex-M CPU Core registers
  - r0-r15 General Purpose
  - r13-r15 Special Role (lr, sp, pc)
  - Program Status Registers
  - Exception Mask Registers
  - Control Register

- General Purpose Registers
  - Used on every clock cycle
  - Always updating



Core CPU Registers [S5]

- Cortex-M CPU Special Function Registers
  - Program Status Registers
    - APSR
    - IPSR
    - EPSR
  - Exception Mask Registers
    - PRIMASK
    - FAULTMASK
    - BASEPRI
  - Control Register

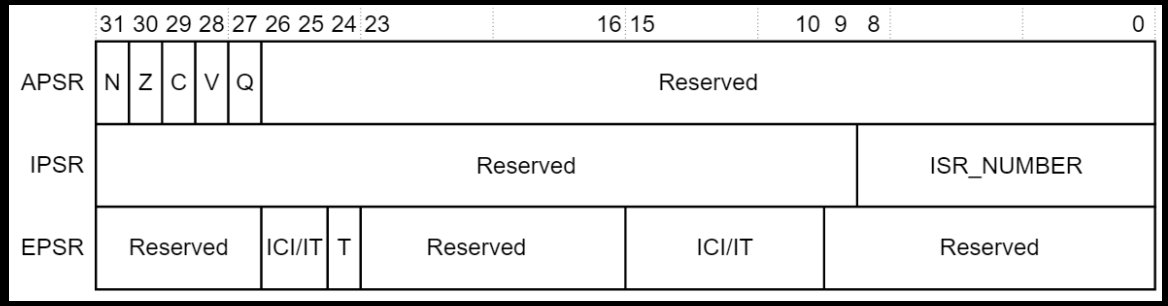
Not part of the memory map. Needs special instructions to read/write

```
MRS     r1, BASEPRI
MSR     FAULTMASK, r0
```

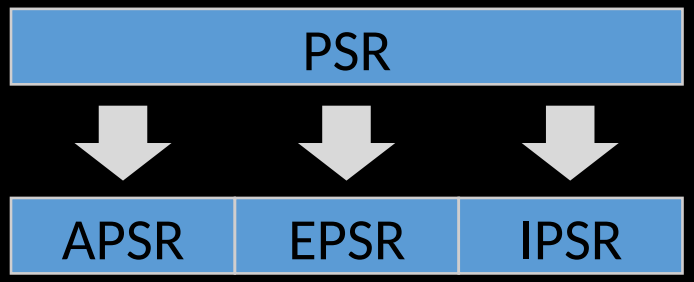
Cortex-M4 Program Status Registers [S6a]

PSR

The Program Status Register (PSR) captures the current CPU and program application state.

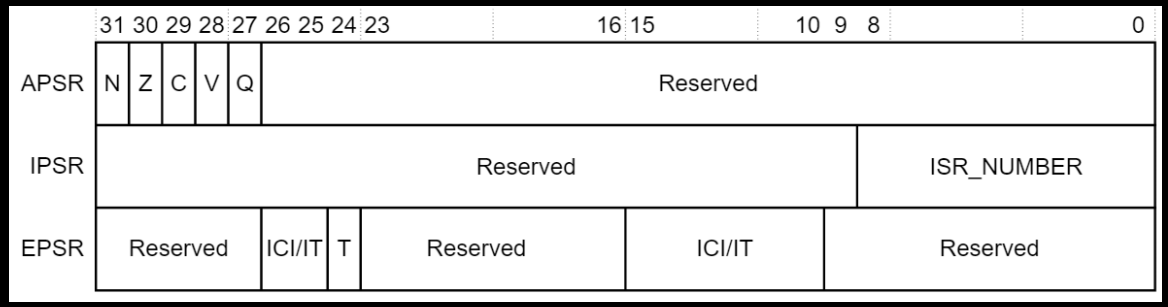


Cortex-M4 Program Status Registers [S6b]



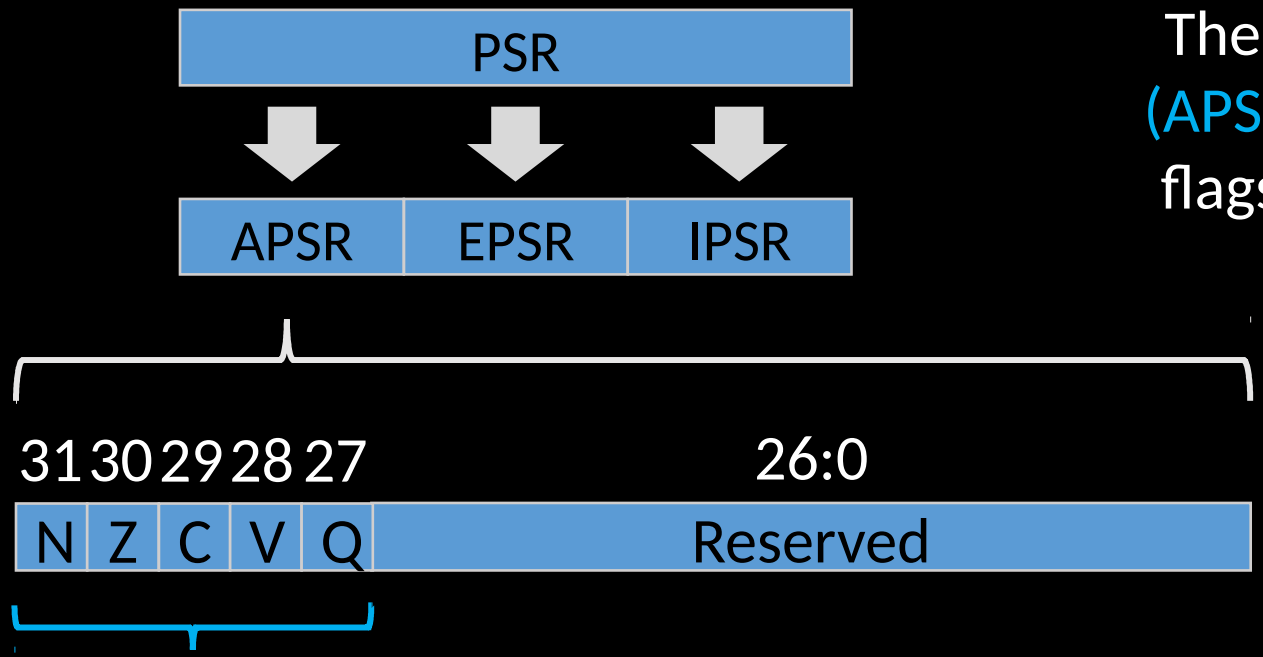
The Program Status Register (PSR) captures the current CPU and program application state.

The PSR combines the APSR, EPSR, IPSR



These status registers are *mutually exclusive* bit fields in the 32-bit PSR.

Cortex-M4 Program Status Registers [S7]



The Application Program Status Register (APSR) contains current state of condition flags from *previous instruction execution*

N = Negative Flag → Was last result negative?

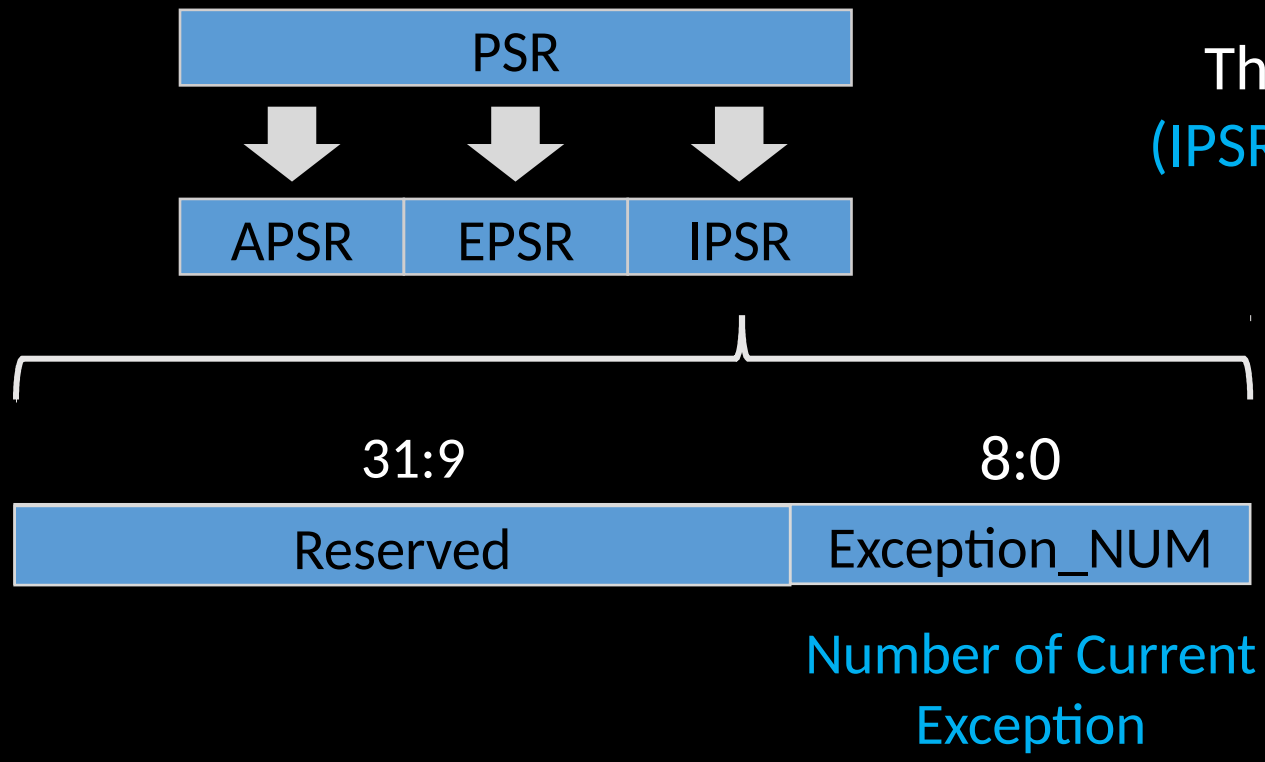
Z = Zero Flag → Was last result zero?

C = Carry Flag → Did last result have a carry?

V = Overflow Flag → Did last result have an overflow?

Q = Saturation Flag → Did last result saturate?

Cortex-M4 Program Status Registers [S8]



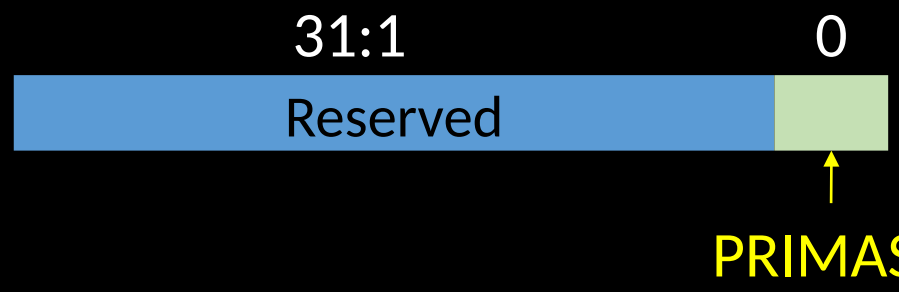
The Interrupt Program Status Register (IPSR) contains the *exception type number* of the current ISR.

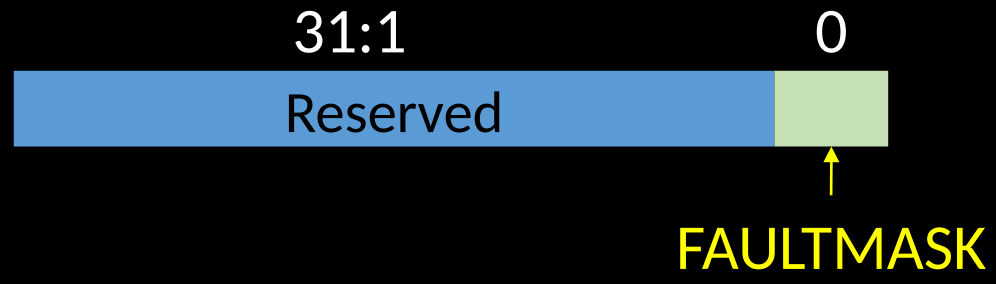
Exception Numbers:

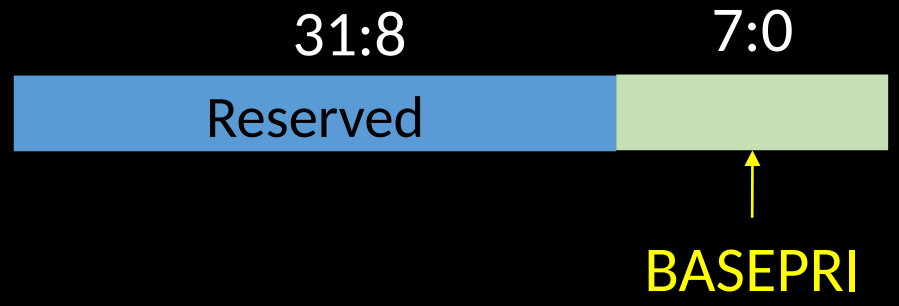
- 0 = Reset
- 1 = Non-Maskable Interrupt (NMI)
- 2 = Hard Fault
- 3 = Memory Management Fault
- …
- 16 = IRQ0
- 17 = IRQ1
- …
- N+15 = IRQ(n-1)

Exception Mask Registers [S9a]

- PRIMASK
  - Prevents activation of all exceptions with configurable priority (If set to 1)
- FAULTMASK
  - Prevents activation of all exceptions with configurable priority EXCEPT the Non-Maskable Interrupt (NMI) (If set to 1)
- BASEPRI
  - Defines minimum priority for exception processing

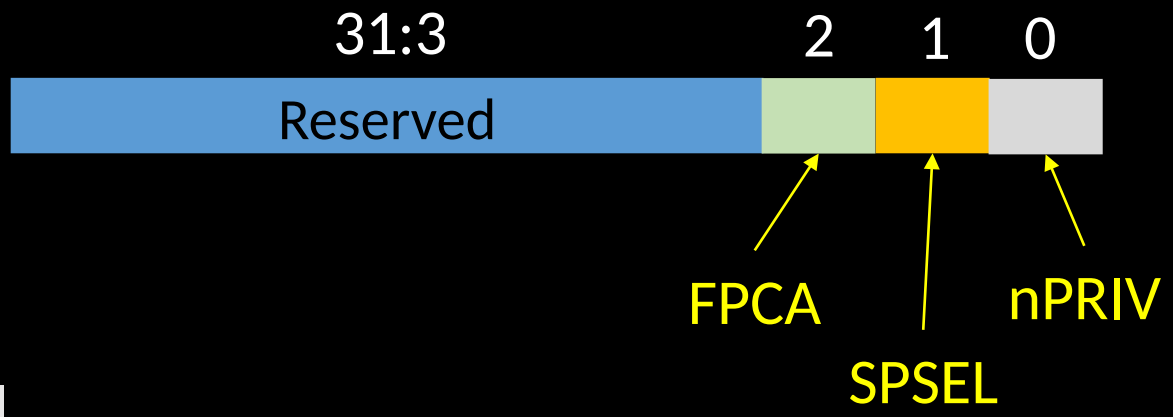




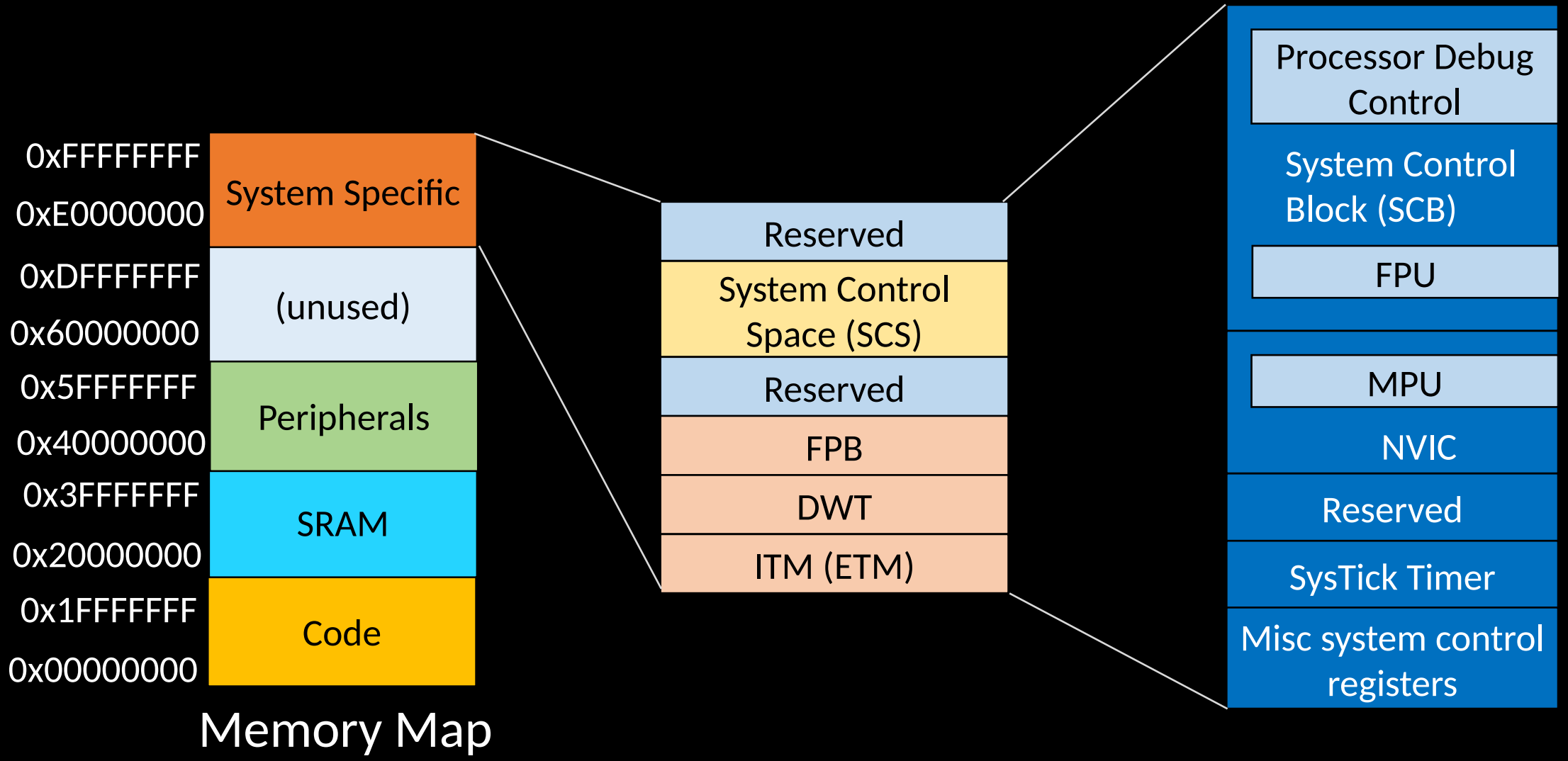


Control Register [S9b]

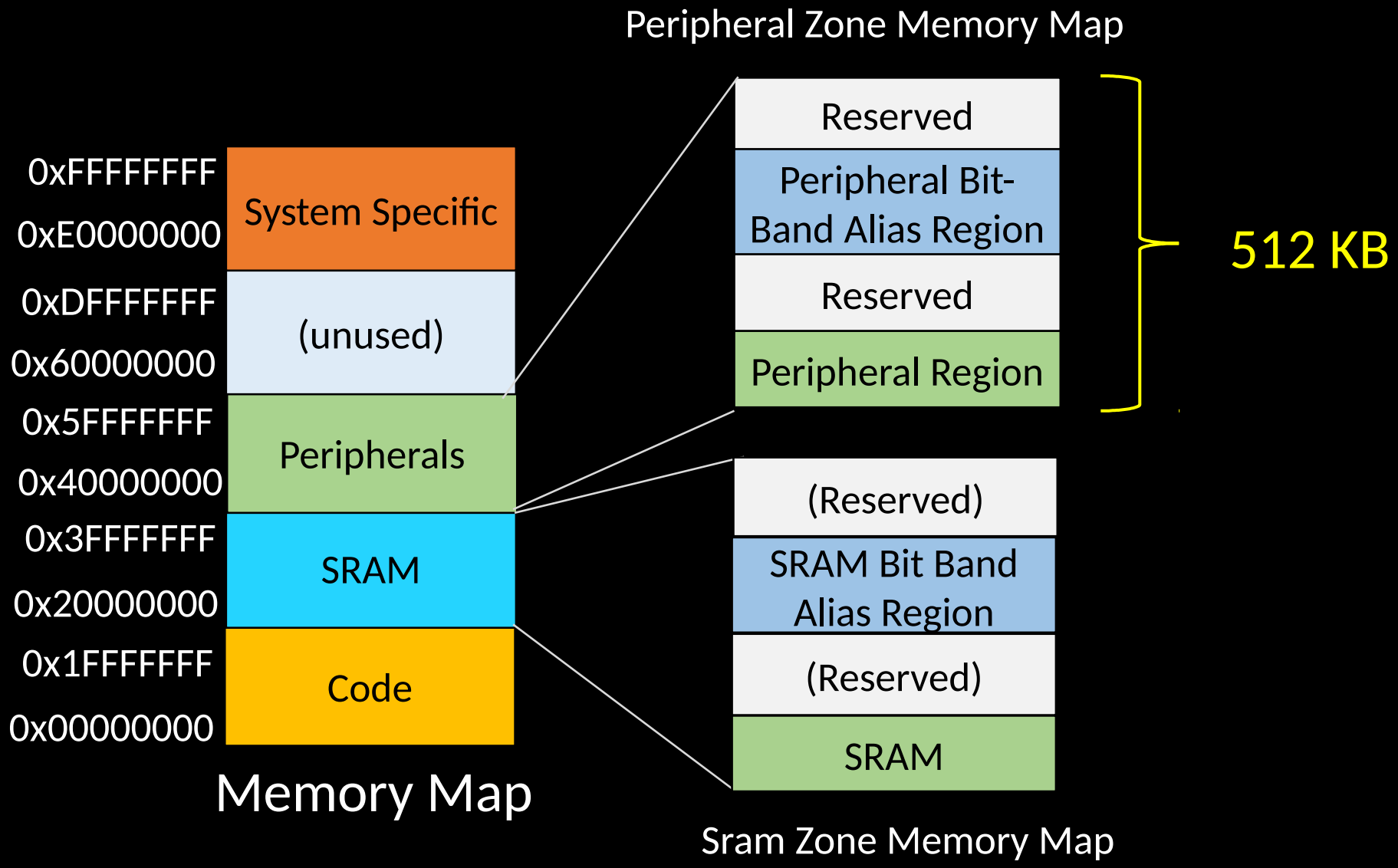
- FPCA – Floating Point State Active
  - 0 = No Floating Point Context Active
  - 1 = Floating Point Context Active
- SPSEL – Selects Current Stack Pointer
  - 0 = MSP – Main Stack Pointer
  - 1 = PSP – Process Stack Pointer
- nPRIV – Defines Privilege Level
  - 0 = Privileged
  - 1 = Unprivileged



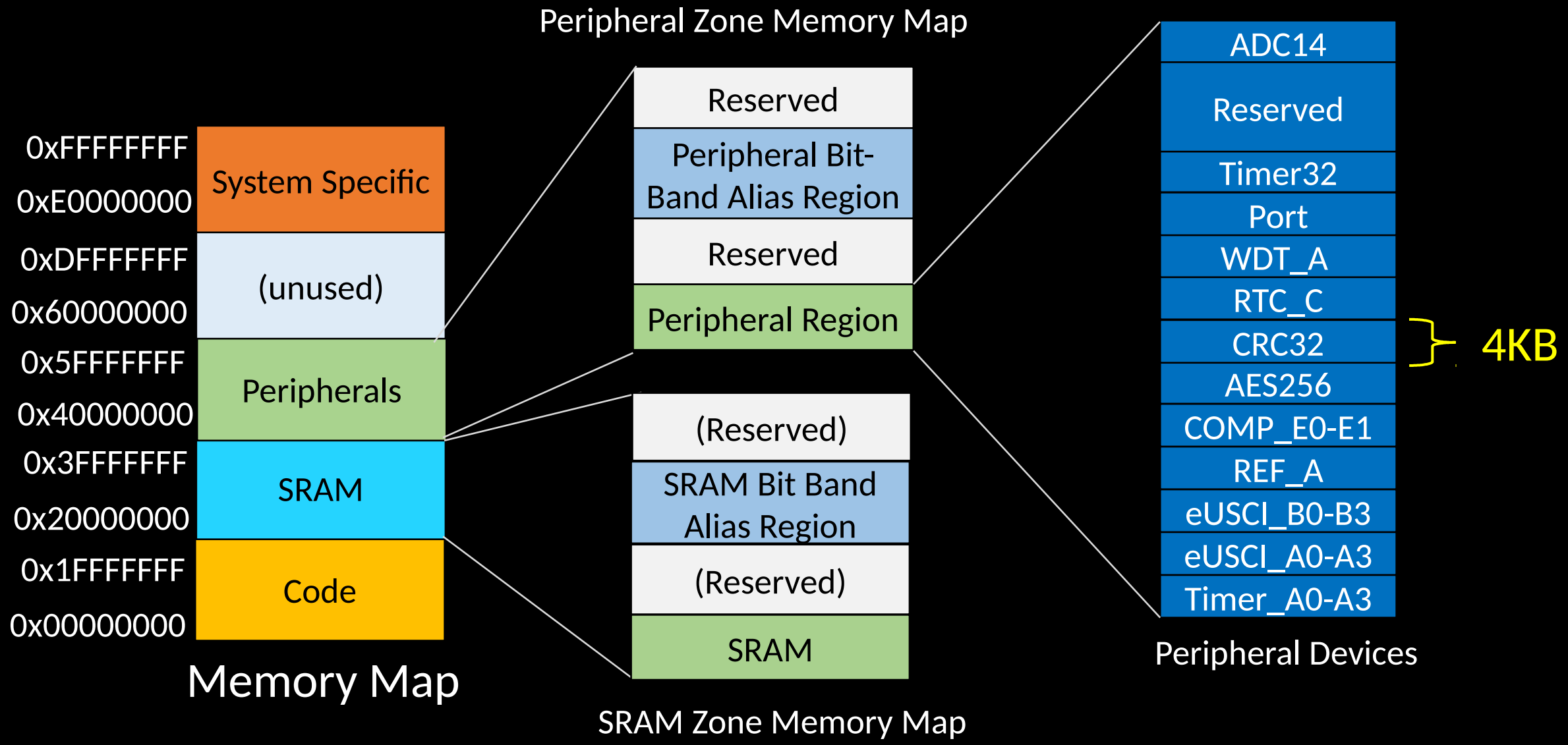
Private Peripherals [S10]



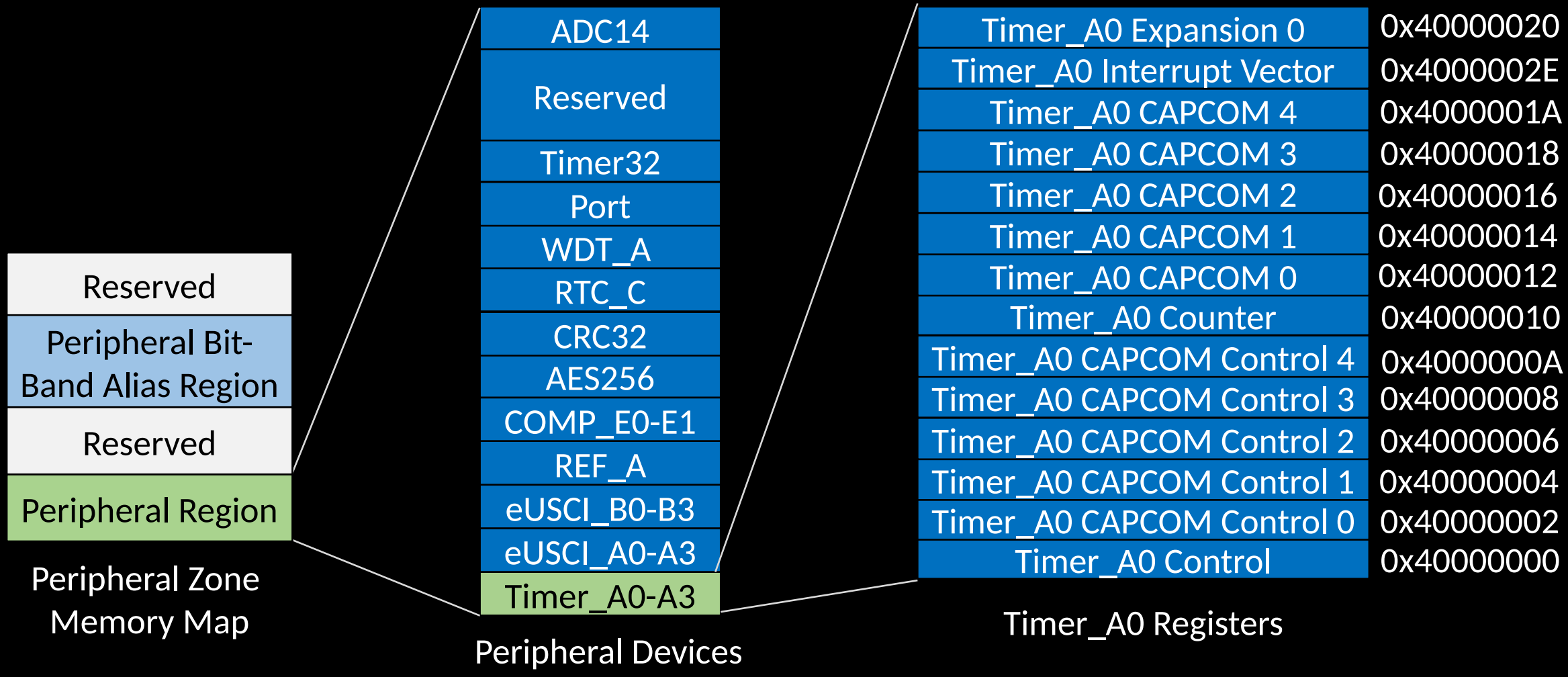
General Peripherals [S11a]



General Peripherals [S11b]



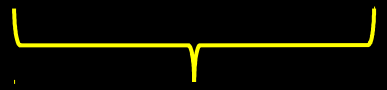
Timer A0 Peripheral [S12]



Timer A0 Control Register [S13a]

Timer_A0 Control Register

15:10	9	8	7	6	5	4	3	2	1	0
Reserved	TASSEL		ID		MC		Reserved	TACLR	TAIE	TAIFG

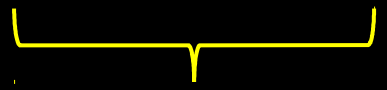


Bit fields are variables with *predefined widths* (i.e. non-standard sizes) within the register

Timer A0 Control Register [S13b]

Timer_A0 Control Register

15:10	9	8	7	6	5	4	3	2	1	0
Reserved	TASSEL		ID		MC		Reserved	TACLR	TAIE	TAIFG



Bit fields are variables with *predefined widths* (i.e. non-standard sizes) within the register

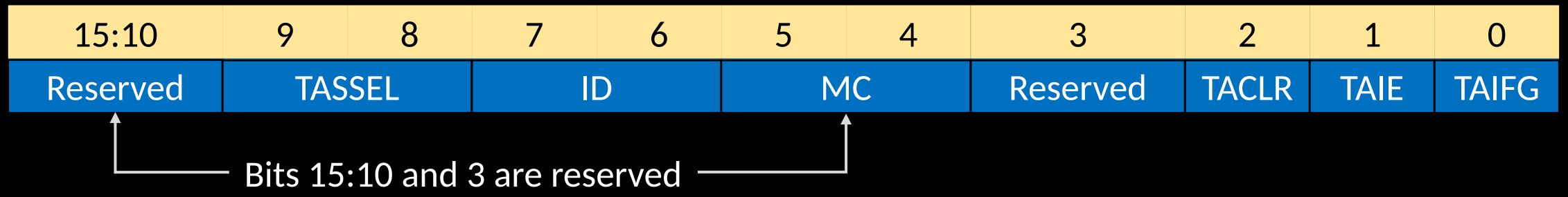
Bit Fields help represent operations/status/configurations.

Bit fields of Timer_A0 Control Register

TASSEL	= Select input clock source for Timer_A0
ID	= Select input clock divider for Timer_A0
MC	= Select count-mode for Timer_A0
TACLR	= Resets Timer_A0
TAIE	= Enable Timer_A0 interrupt
TAIFG	= Timer_A0 interrupt flag

Timer A0 Control Register [S13c]

Timer_A0 Control Register



Reserved bits can be:

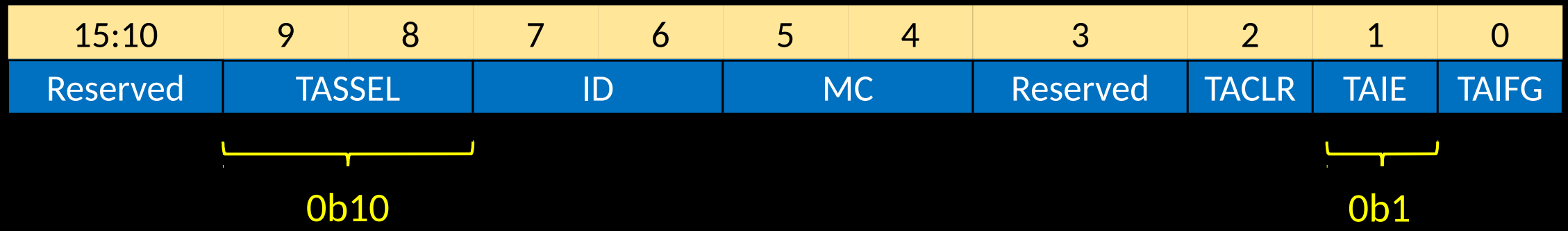
- Unused
- Reserved for advanced sub-family models
- For internal use only

Bit fields of Timer_A0 Control Register

TASSEL	= Select input clock source for Timer_A0
ID	= Select input clock divider for Timer_A0
MC	= Select count-mode for Timer_A0
TACLR	= Resets Timer_A0
TAIE	= Enable Timer_A0 interrupt
TAIFG	= Timer_A0 interrupt flag

Timer A0 Control Pointer [S14]

Timer_A0 Control Register

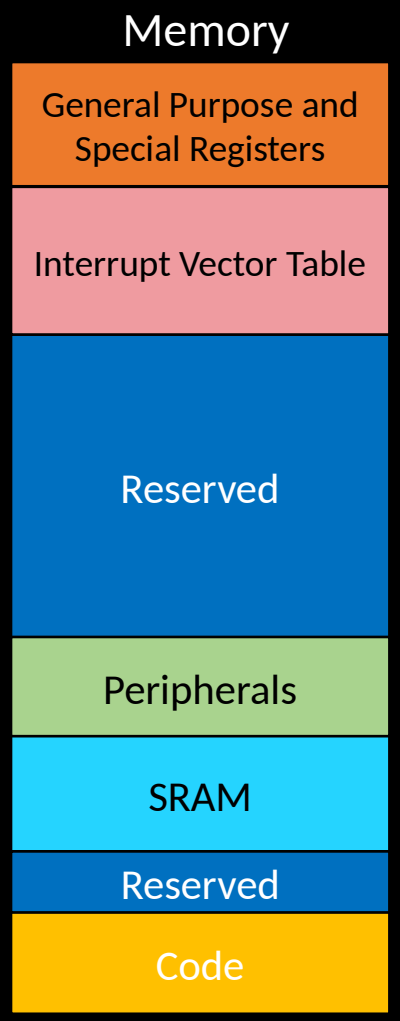


- 16-Bit Register (Address = 0x40000000)
- Configure for:
  - TASSEL = 2 = 0b10
  - TAIE = 1 = 0b1

```
volatile uint16_t * ta0_ctrl = (uint16_t*)0x40000000;
*ta0_ctrl = 0x0202;
```

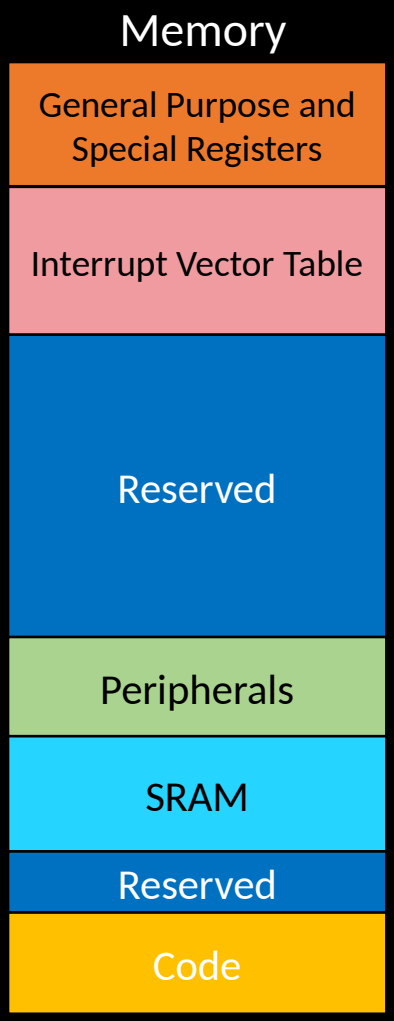
Unused Slides

Memory Map [S1.3.5.a]



Process of dividing the Address Space into logical sections

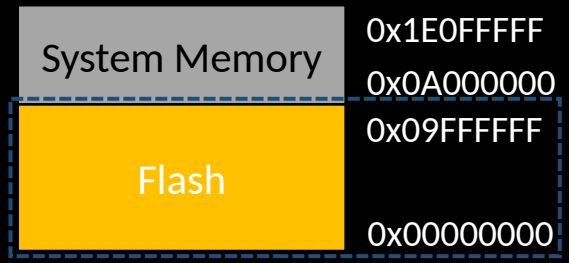
Memory Map [S1.3.5.b]



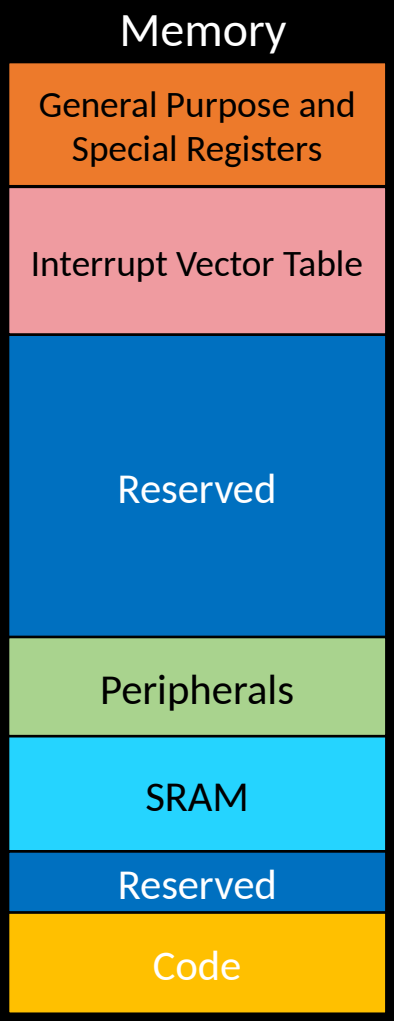
Process of dividing the Address Space into logical sections

➡ Address ranges are mapped to particular functions

e.g. 0x00000000-0x09FFFFFF range mapped to flash

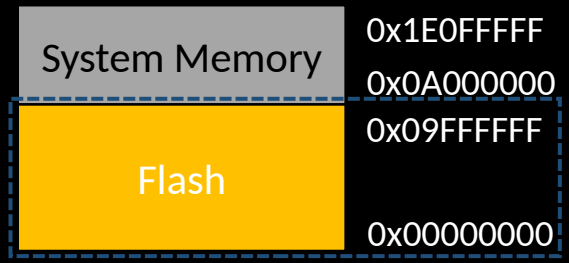


Memory Map [S1.3.5.c]

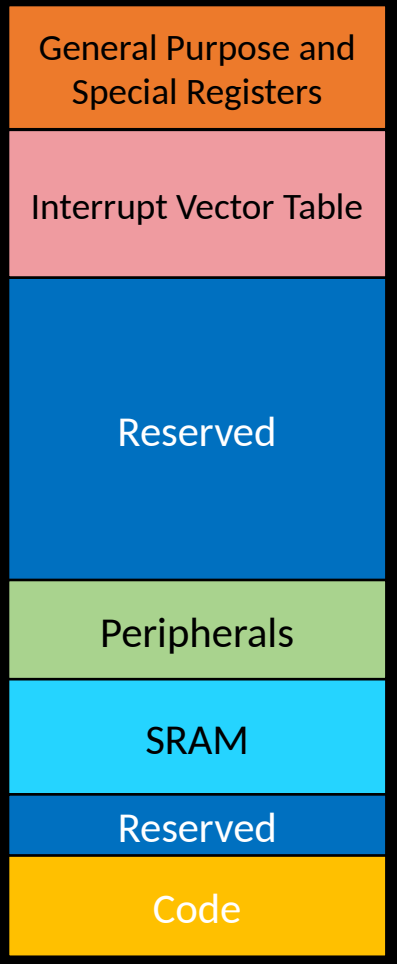


Process of dividing the Address Space into logical sections

- Address ranges are mapped to particular functions
  - e.g. 0x00000000-0x09FFFFFF range mapped to flash
- Addresses act as a physical/virtual access of a memory location

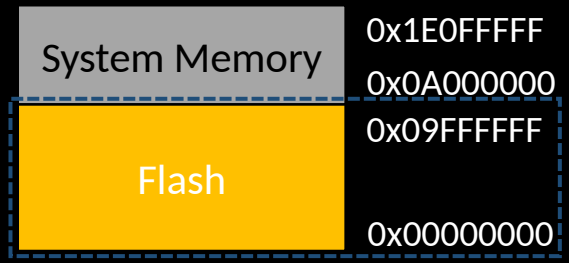


Memory Map [S1.3.5.d]



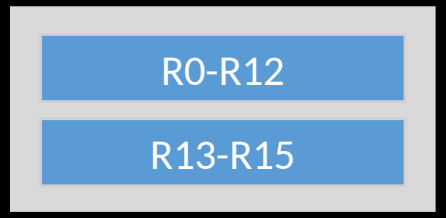
Process of dividing the Address Space into logical sections

- Address ranges are mapped to particular functions
  - e.g. 0x00000000-0x09FFFFFF range mapped to flash
- Addresses act as a physical/virtual access of a memory location
- Typically includes: Flash, SRAM, Peripherals, Debug, General Purpose and Special Registers



Types of Registers [S1.3.6.b]

CPU Registers

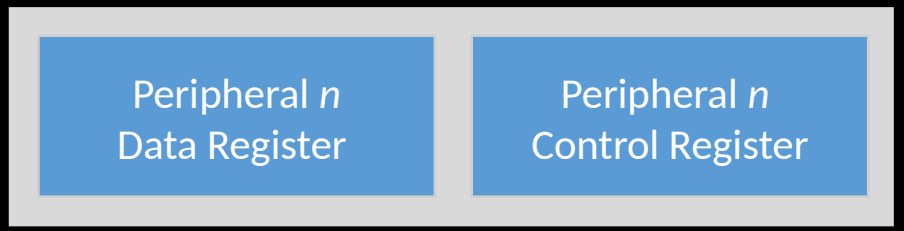


General-Purpose Registers

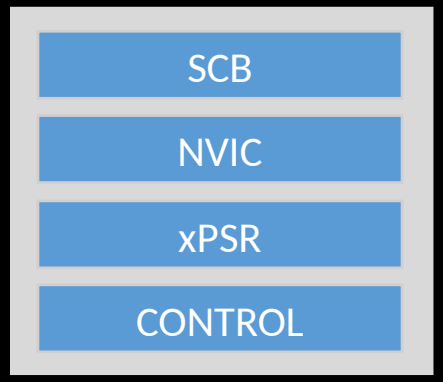
CPU-Special Registers

- Stack Pointer (SP)
- Link Register (LR)
- Program Counter (PC)

Peripherals



Special Registers



System Control Block

Nested Vector Interrupt Controller

Program Status Registers

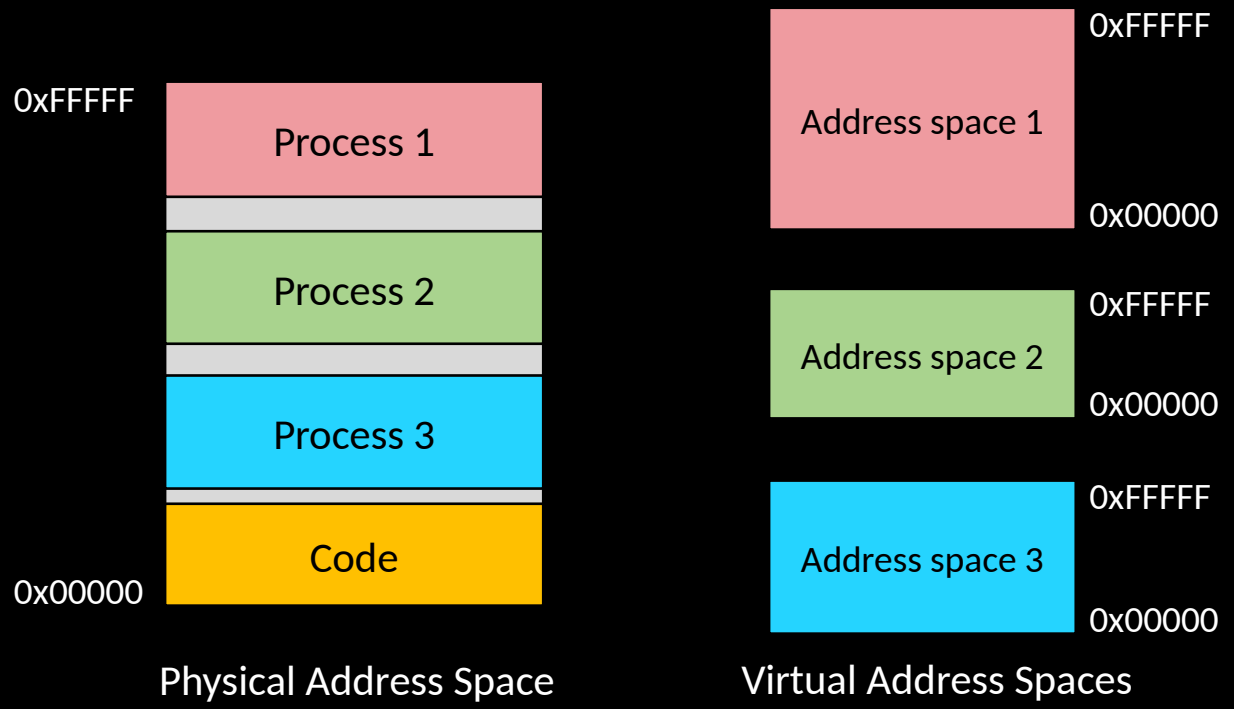
Processor's control

Address Space [S1.3.5.3]

The address space is the range of valid addresses in memory that a program/process can *access*.

Peripherals located (i.e. *mapped*) to address space of physical memory (memory space) are called *Memory Mapped Peripherals* (or more commonly *memory mapped I/O*).

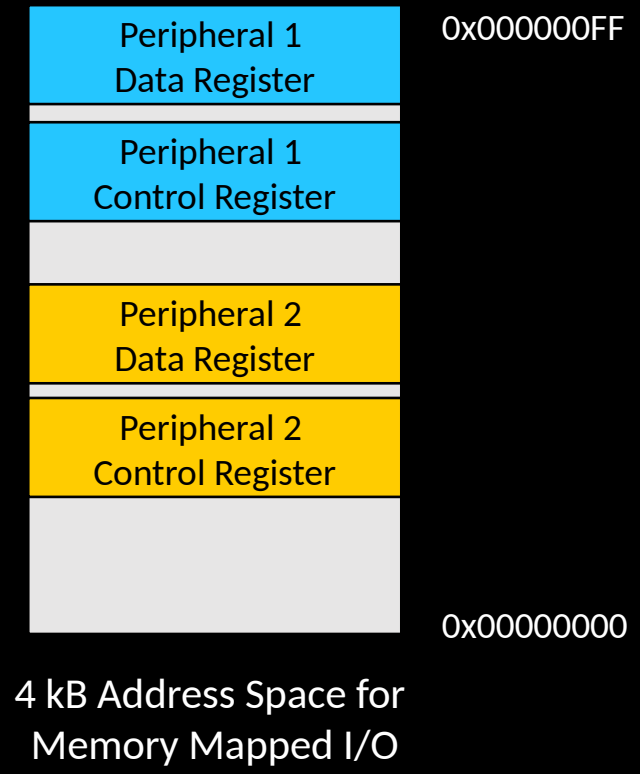
Easy for programmer to interface address space due to bus decision making. However, different address spaces require different read/write mechanisms.



I/O Blocks and Peripheral Limitations [S1.3.5.4]

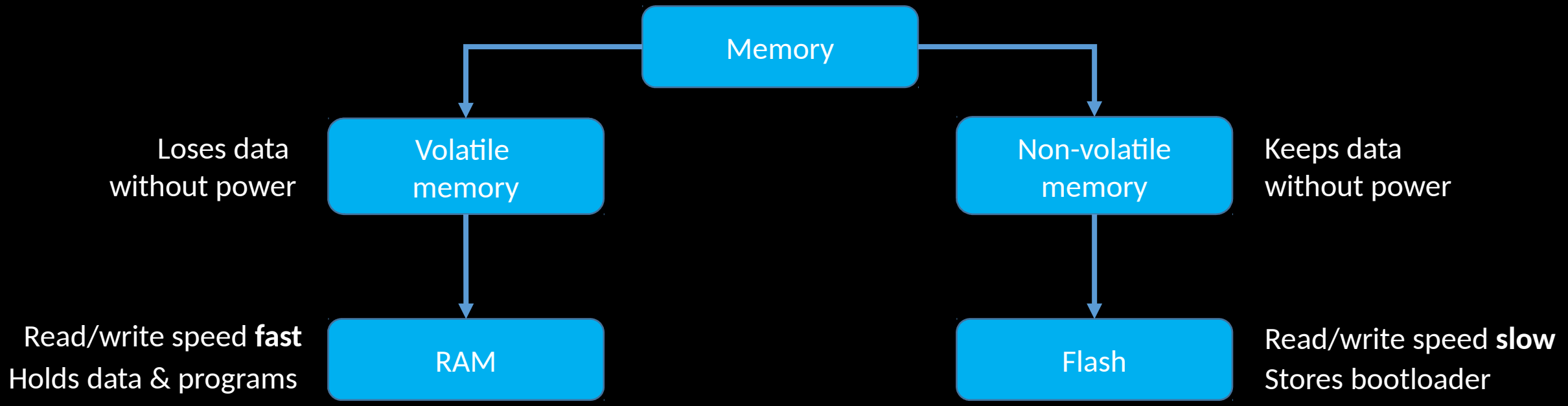
Typical size of address spaces are 4kB, though can be larger.

Because of memory mapped I/O, peripherals are therefore limited by size of address space.

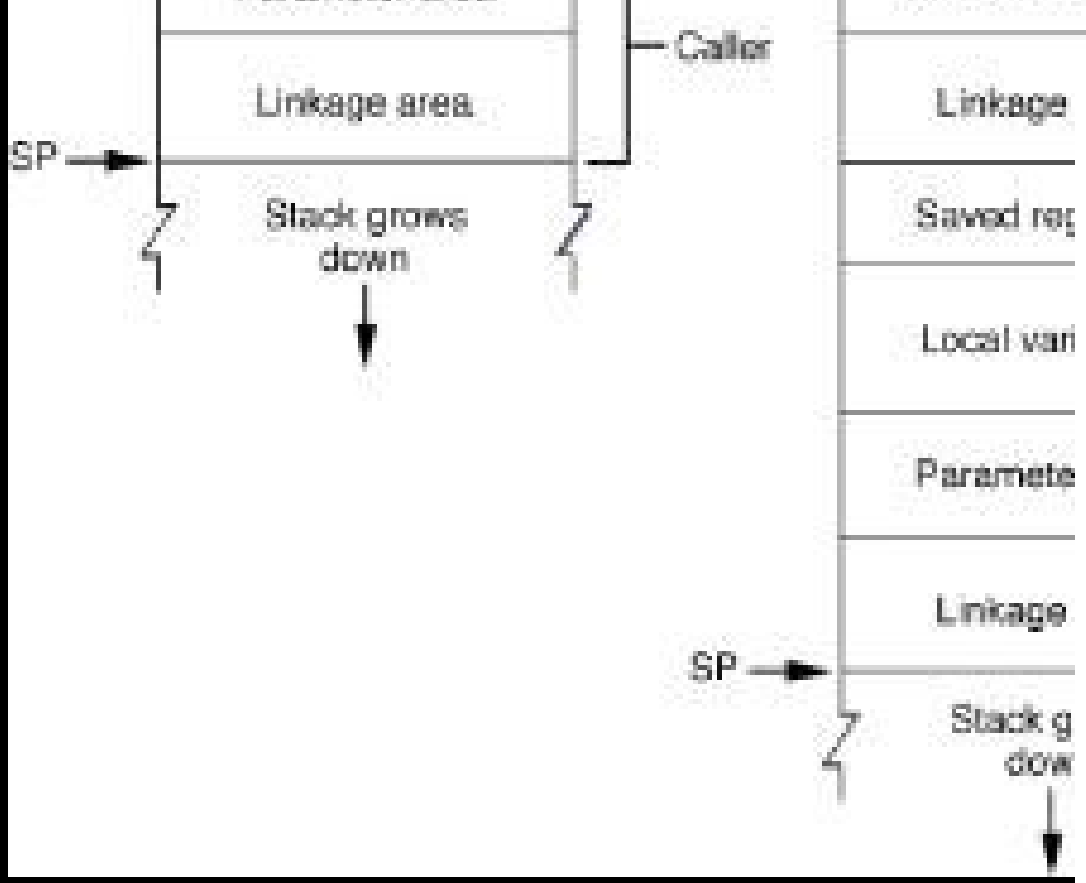


4 kB Address Space for Memory Mapped I/O

Types of Memory [S1.3.3.1a]



Reading/writing to memory:
Expensive, Takes Time,
Architecture designed for memory cache hits



(stack grows towards zero)

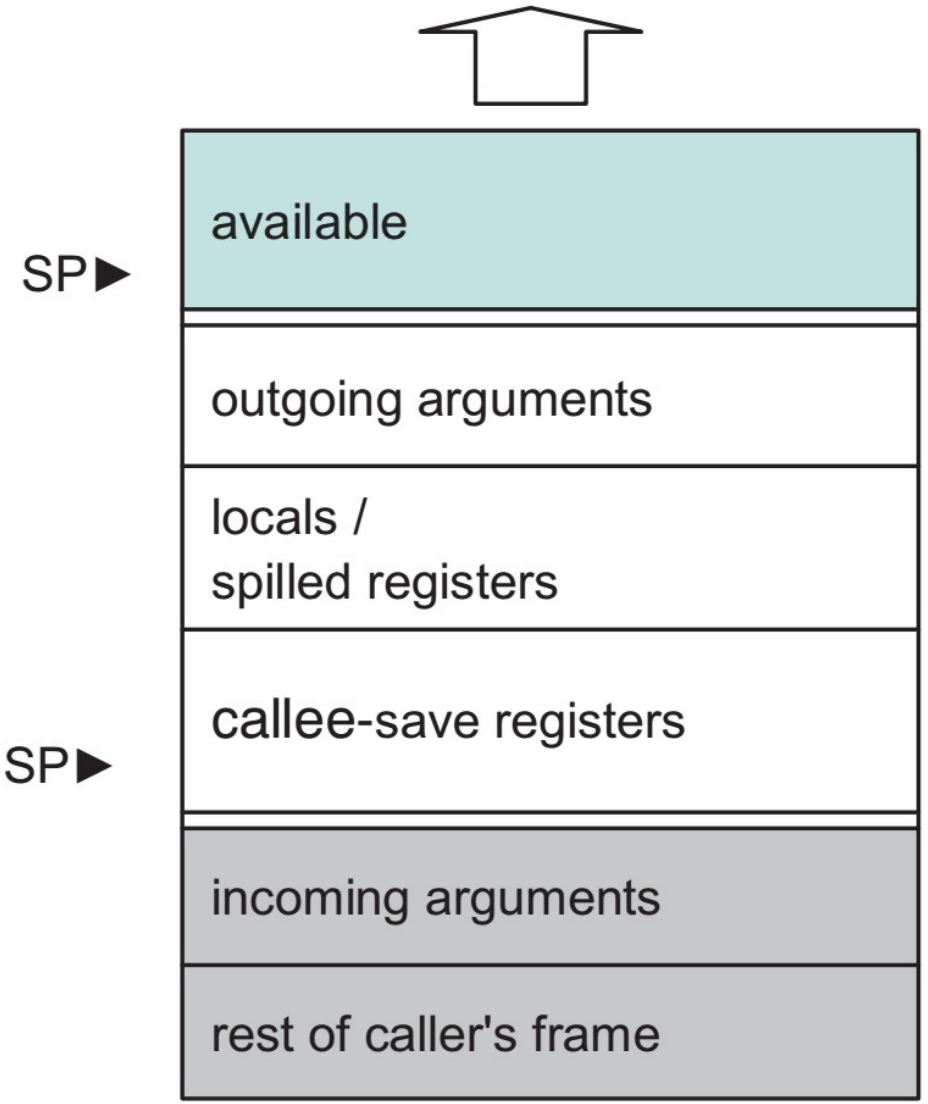
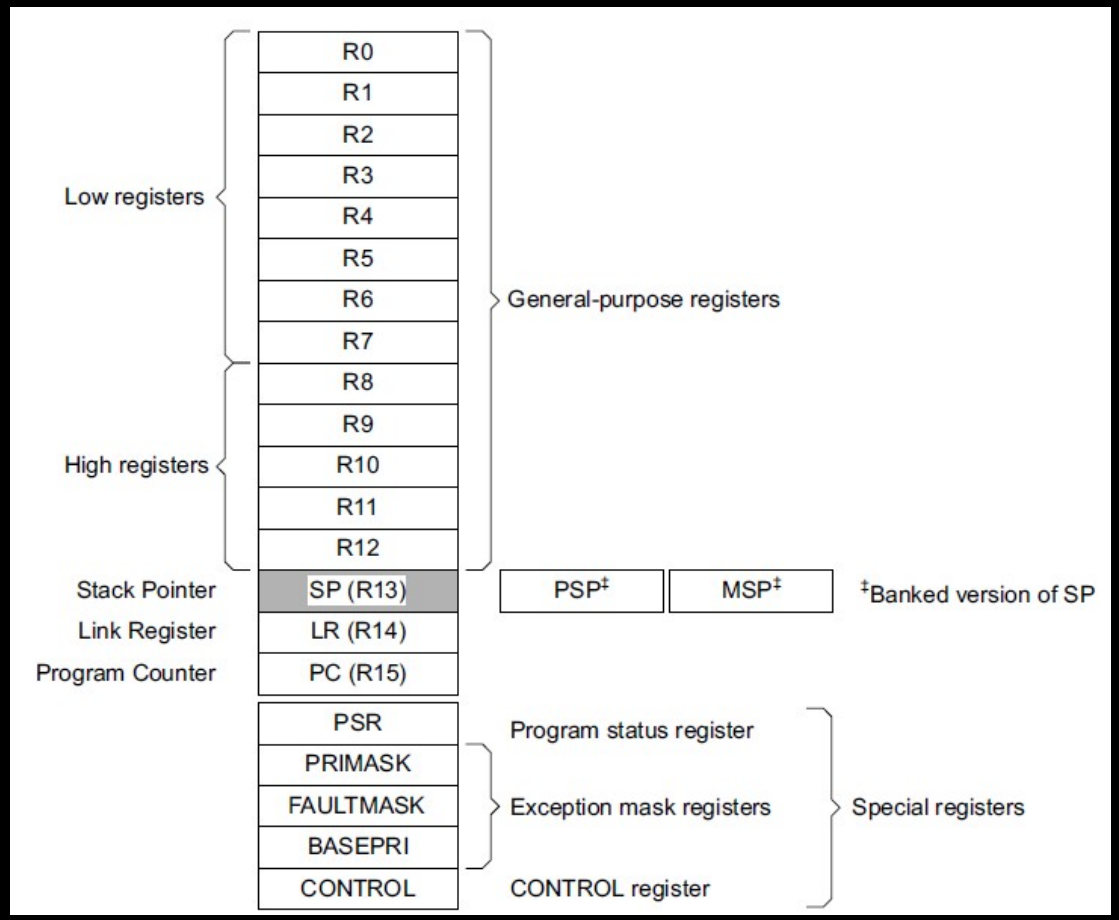


Figure 4. Local Frame Layout

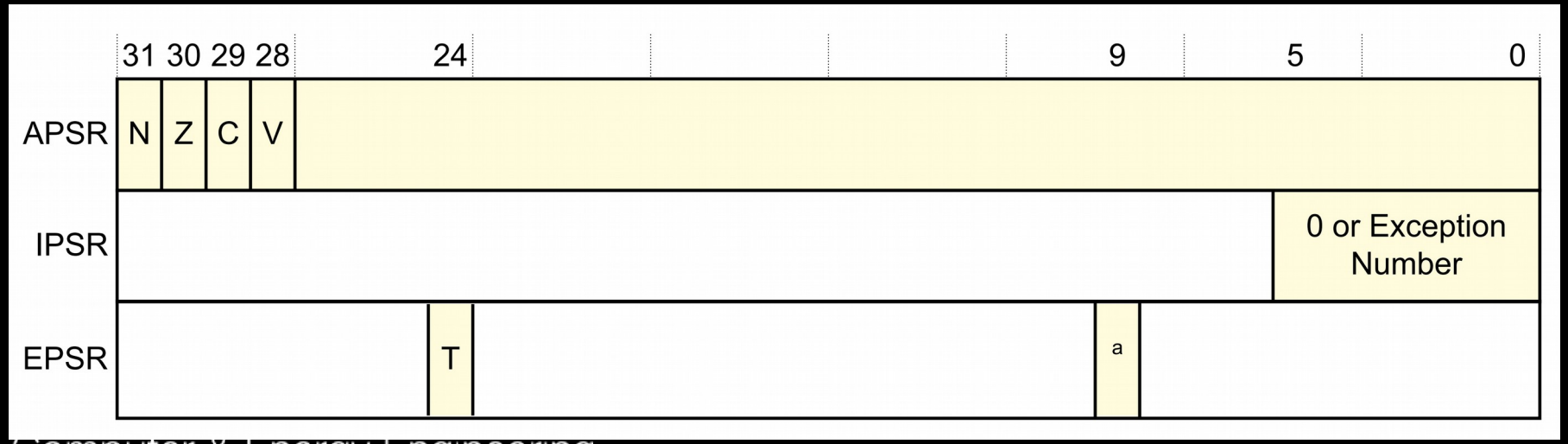
CPU Registers [S6]



Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter.
r14		LR	The Link Register.
r13		SP	The Stack Pointer.
r12		IP	The Intra-Procedure-call scratch register.
r11	v8		Variable-register 8.
r10	v7		Variable-register 7.
r9		v6 SB TR	Platform register. The meaning of this register is defined by the platform standard.
r8	v5		Variable-register 5.
r7	v4		Variable register 4.
r6	v3		Variable register 3.
r5	v2		Variable register 2.
r4	v1		Variable register 1.
r3	a4		Argument / scratch register 4.
r2	a3		Argument / scratch register 3.
r1	a2		Argument / result / scratch register 2.
r0	a1		Argument / result / scratch register 1.

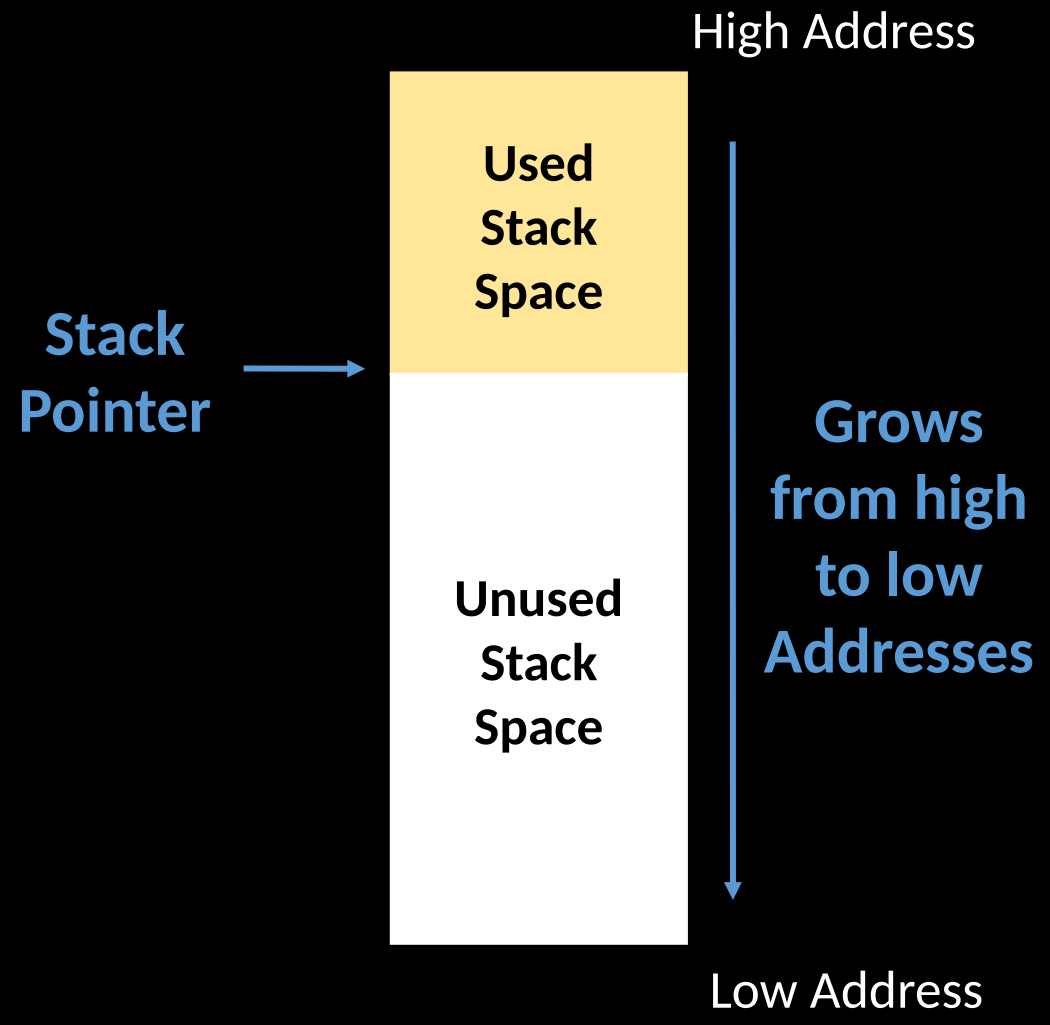
Special Purpose Registers

- APSR – Application Program Status Register
  - NZCV = Flags that evaluate conditional execution
- IPSR – Interrupt Program Status Register
- EPSR – Execution Program Status Register



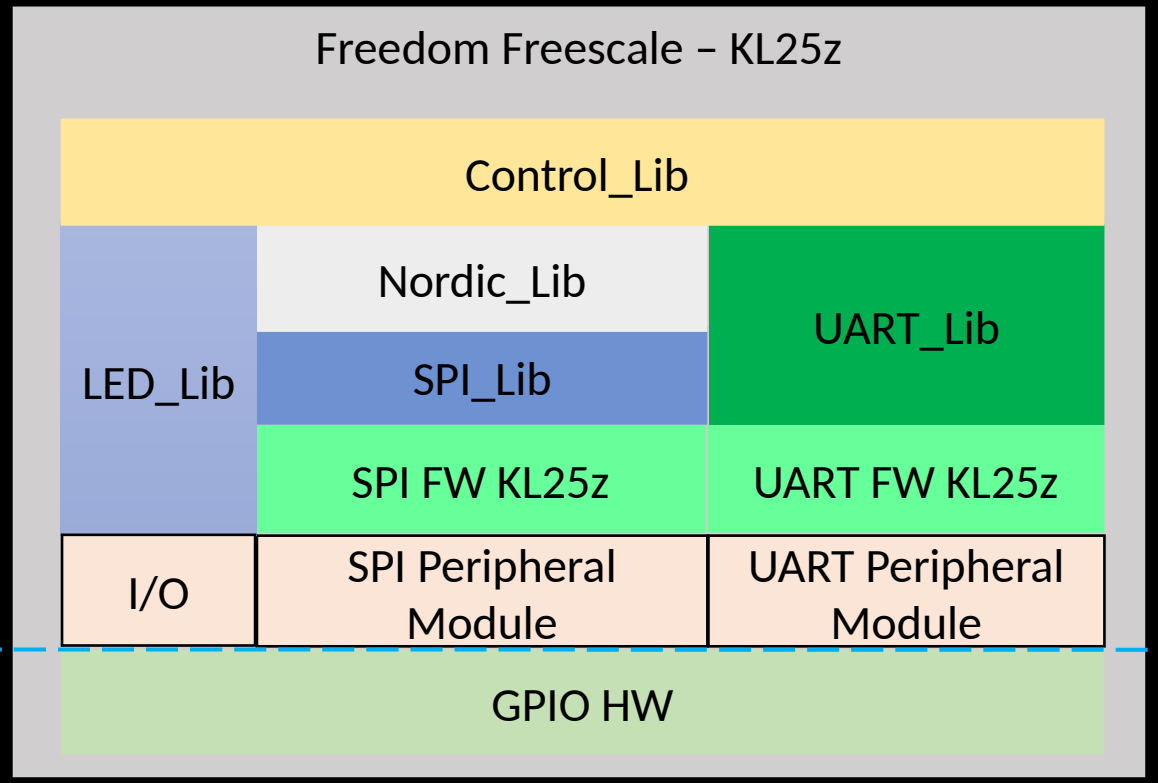
ARM Stack Calling Conventions

- In ARM, the stack is a Full Descending Stack
  - Starts at a High Address, and grows toward low addresses
- LIFO – Last-in-First-Out



Architecture Independence [S2]

- Put a picture in of application code on top of hardware features or hardware specific code



```
811c:       b580            push    {r7, lr}
811e:       af00            add     r7, sp, #0
8120:       4b0a            ldr     r3, [pc, #40]
8122:       210a            movs    r1, #10
8124:       0018            movs    r0, r3
8126:       f000 f85f       bl      81e8 <clear_all>
812a:       4b08            ldr     r3, [pc, #32]
812c:       2200            movs    r2, #0
812e:       21aa            movs    r1, #170
8130:       0018            movs    r0, r3
8132:       f000 f80d       bl      8150 <set_value>
```