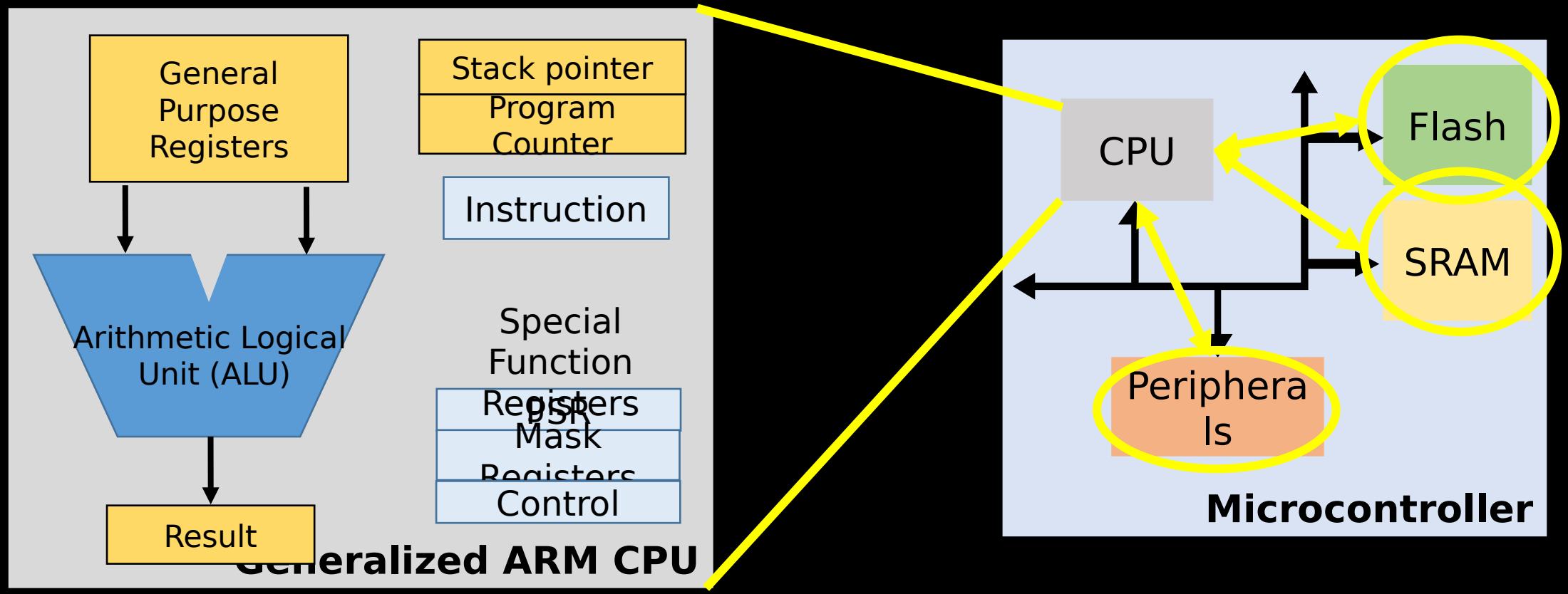
Interacting With Memory

Embedded Software Essentials

C2M1V4

Microcontroller Memory [S1]

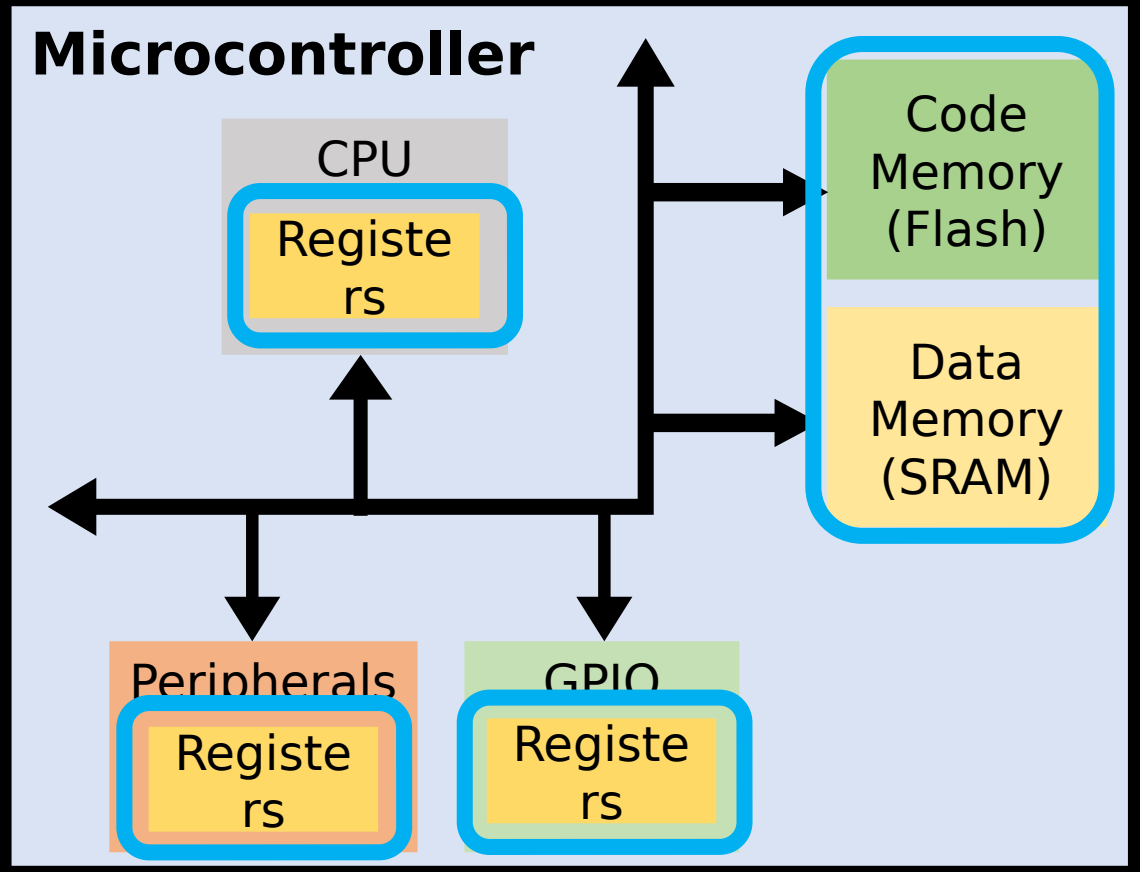
- Memory is used during every single instruction



Embedded Memory [S2]

- Memories of an Embedded Systems
  - Code Memory (Flash)
  - Data Memory (SRAM)
  - Register Memory

- The CPU and peripherals contain register memory
  - CPU Registers
    - General Purpose
    - Special Purpose
  - General Peripheral Registers

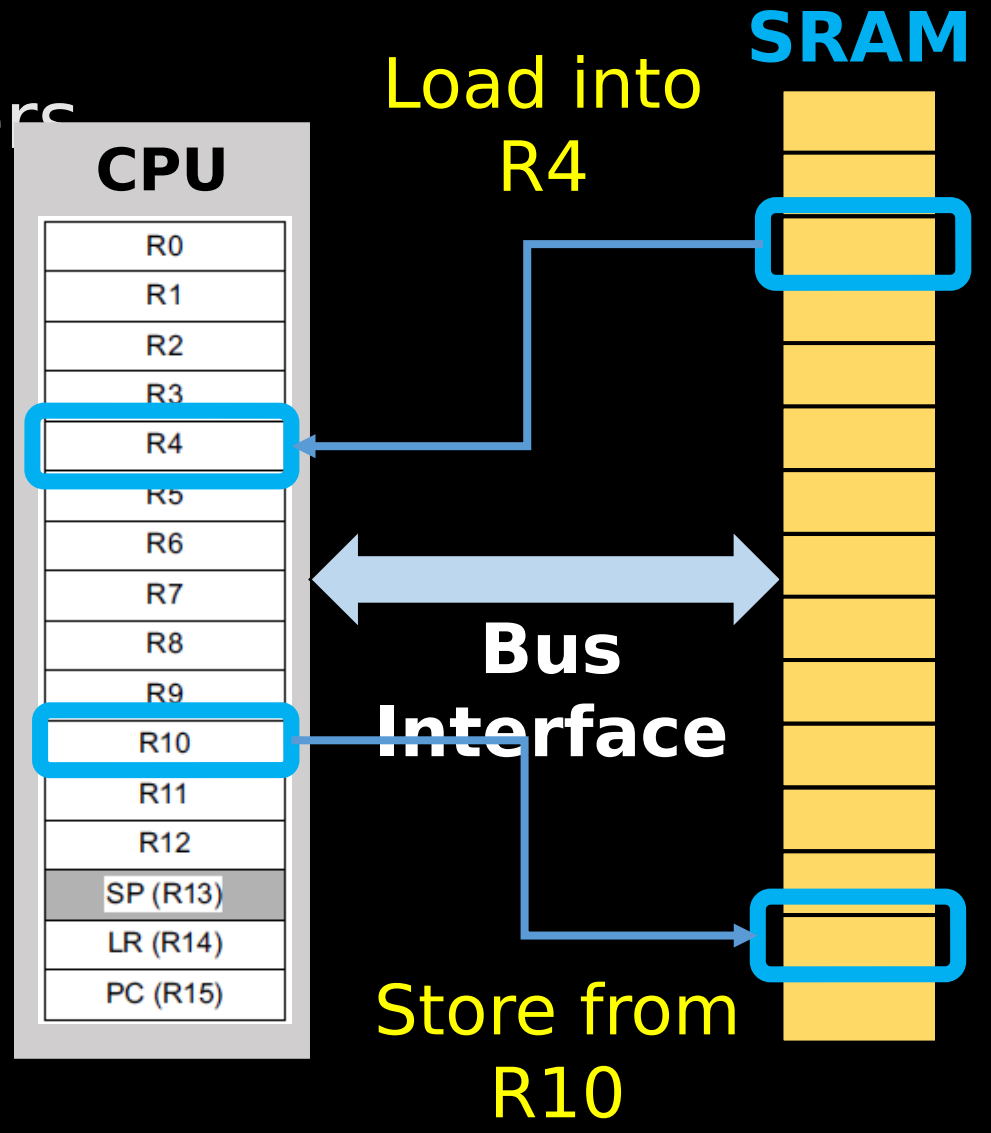


Core CPU Registers [S3]

- CPU Registers have unique identifiers
  - r0-r15, sp, pc, lr
  - apsr, ipsr, epsr
  - primask, faultmask, control, etc

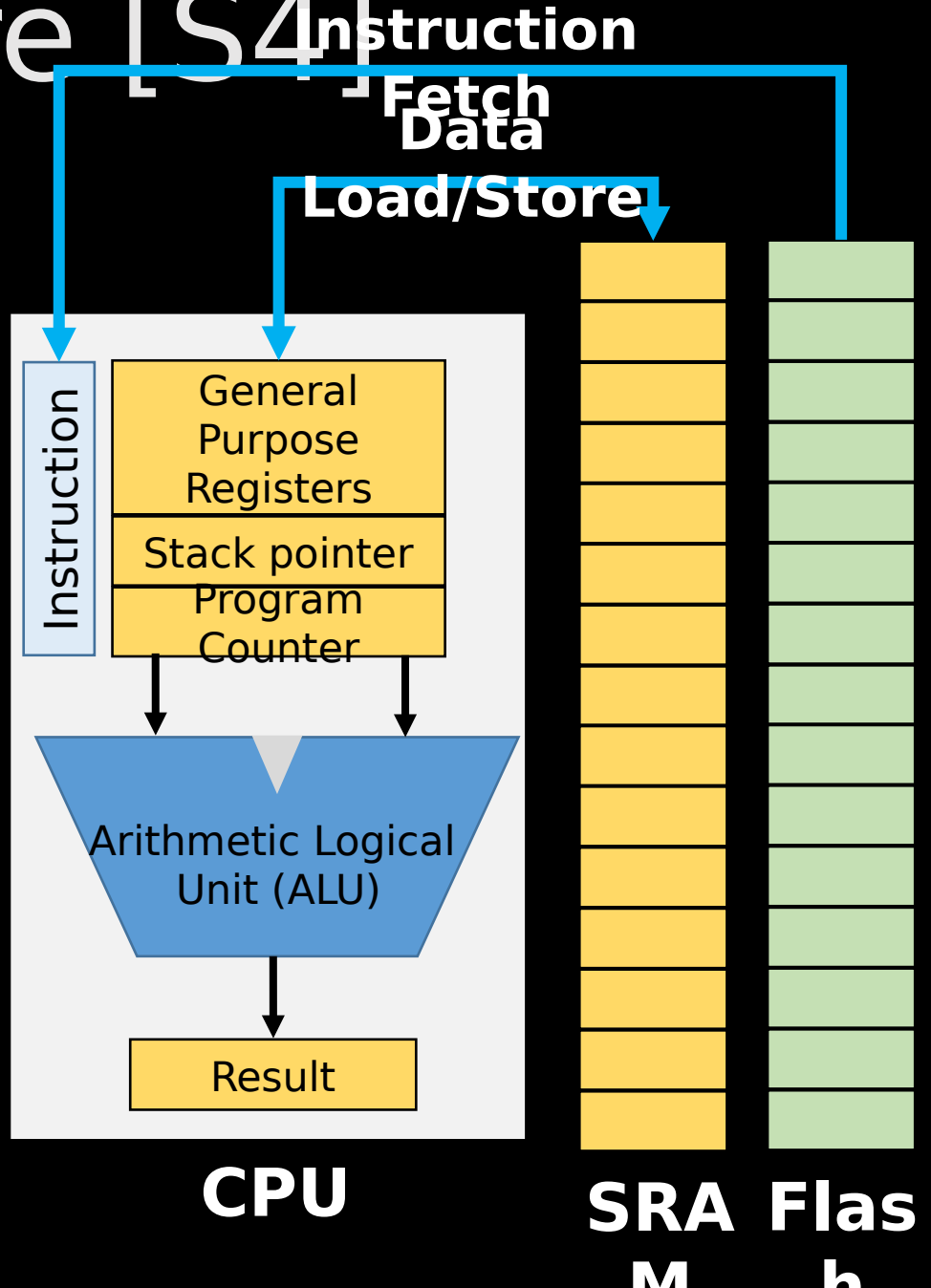
```
811c:       b580            push    {r7, lr}
811e:       af00            add     r7, sp, #0
8120:       4b0a            ldr     r3, [pc, #40]
8122:       210a            movs    r1, #10
8124:       0018            movs    r0, r3
8126:       f000 f85f       bl      81e8 <clear_all>
812a:       4b08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

Example Assembly Code



Load-Store Architecture [S4]

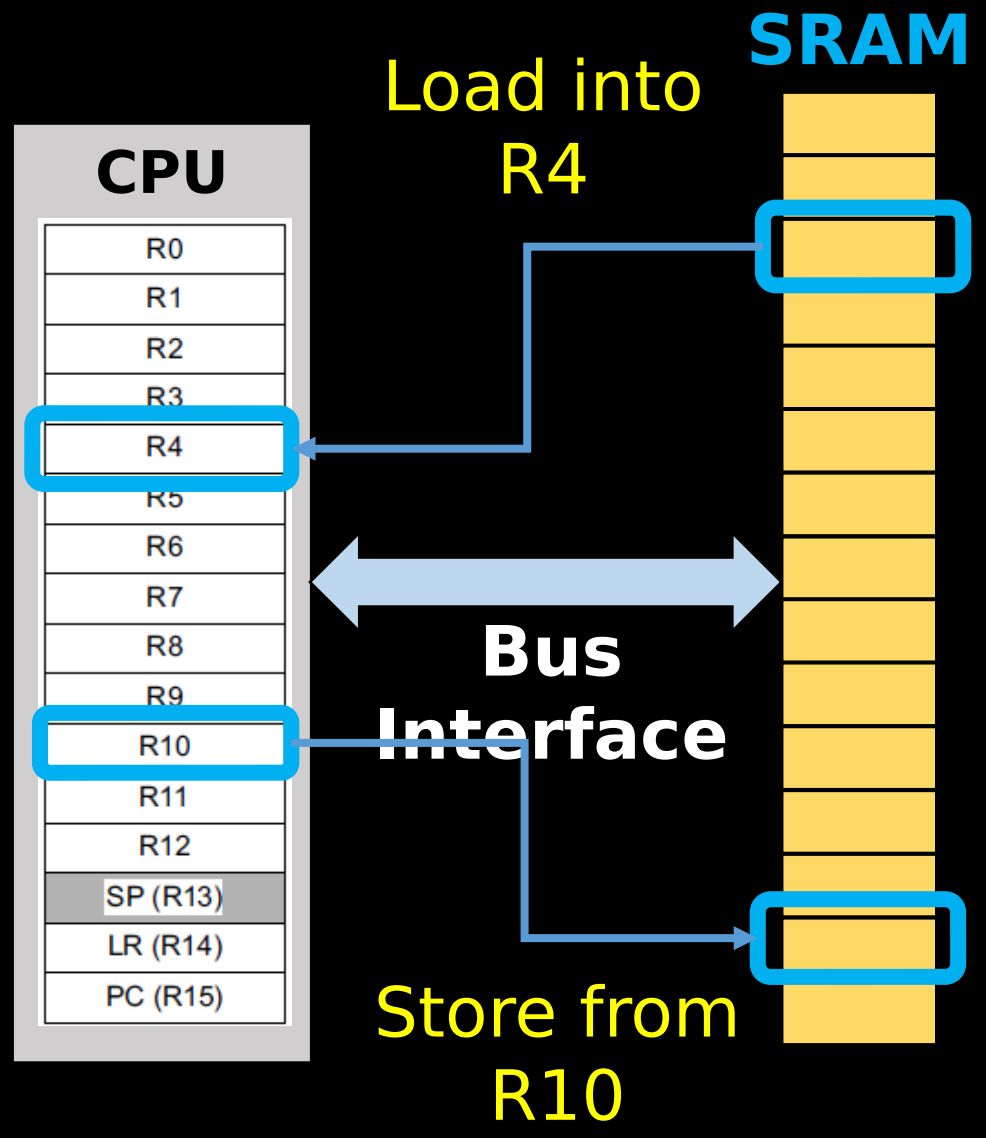
- CPU Register memory is small
  - 16 Registers (R0-R12, SP, PC, LR)
  - 5 Special Function Registers
- All processing occurs in CPU
- CPU Register data constantly changes
  - Data is **loaded** in from memory
  - Results are **stored** back to memory



Read-Modify-Write [S5]

- **Read**: Data is loaded into CPU
- **Modify**: Data is Operated on
- **Write**: Result is stored back

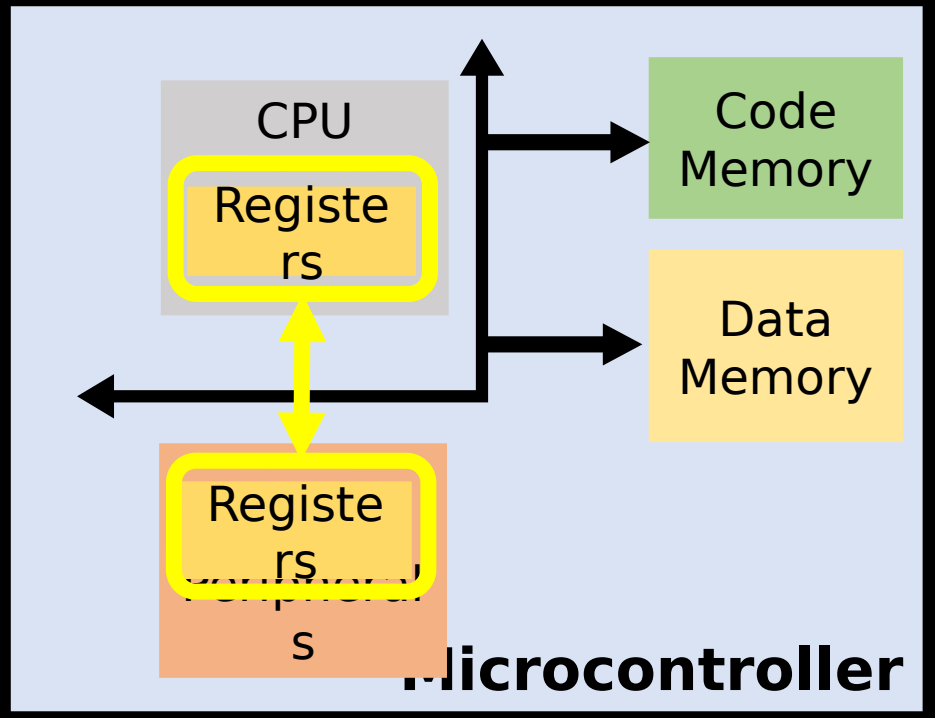
- Compiler tries to reduce number of loads and stores for execution efficiency → Optimization



Peripheral Registers [S6]

- Peripheral register require some contents (bit-fields) to be **preserved**

- Use **bit manipulation** to change certain bits of a register (not all contents)




```
uint8_t * ptr = (uint8_t *) 0x1000;
```

Set 4th bit without changing other bits:

```
*ptr |= 0x10;
```

Clear 4th bit without changing other bits:

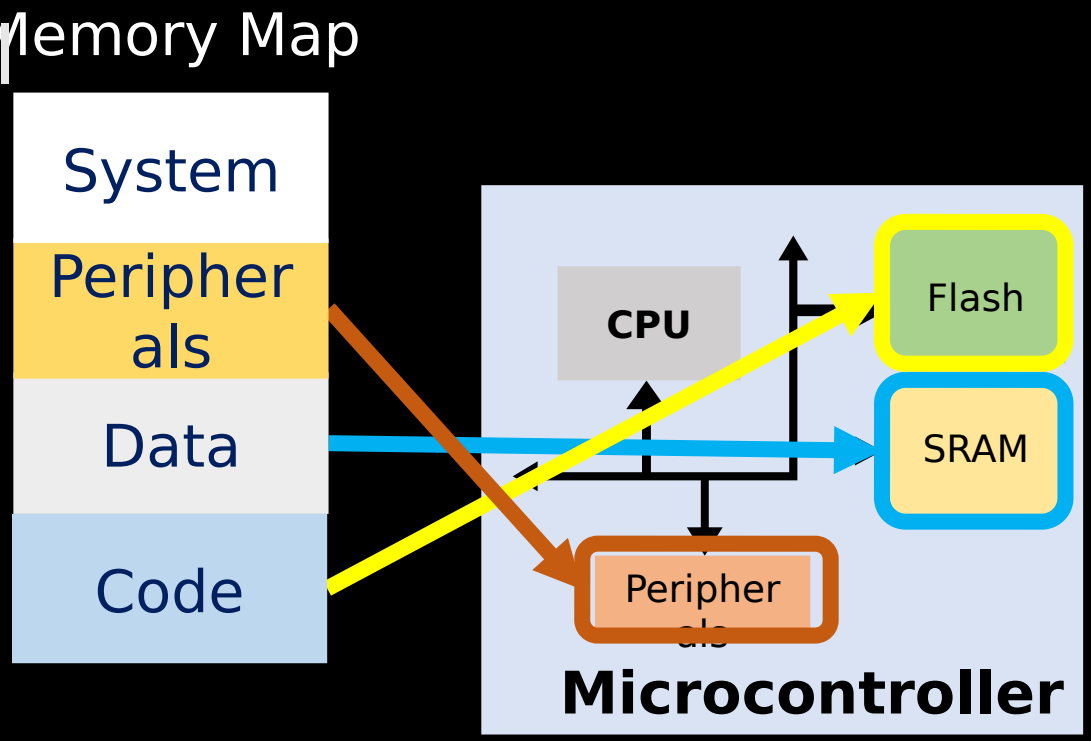
```
*ptr &= ~(0x10);
```

Bit-Banded Regions [S7a]

- Bit-Banded Regions allow programmer to perform bit-level loads and stores

- Bit-Banded operations are atomic
  - Handled in hardware
  - No CPU Load/Store

Reduces Read-Modify-Write overhead!

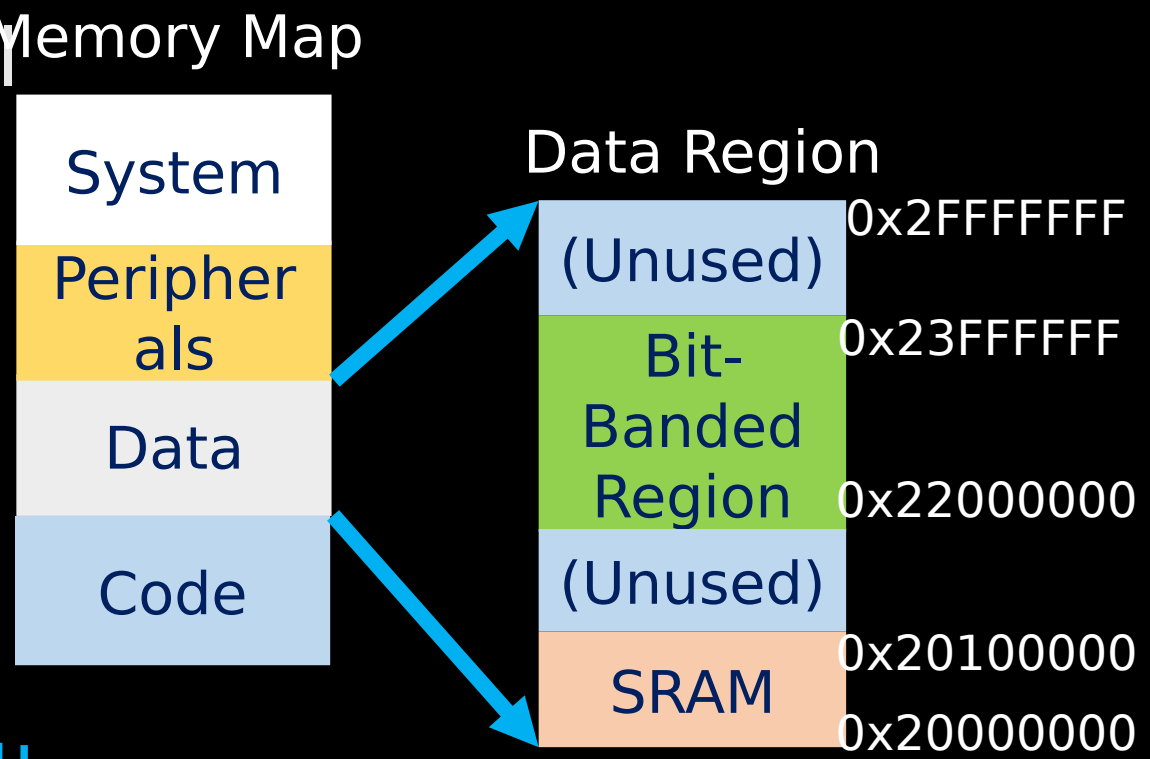


Bit-Banded Regions [S7b]

- Bit-Banded Regions allow programmer to perform bit-level loads and stores

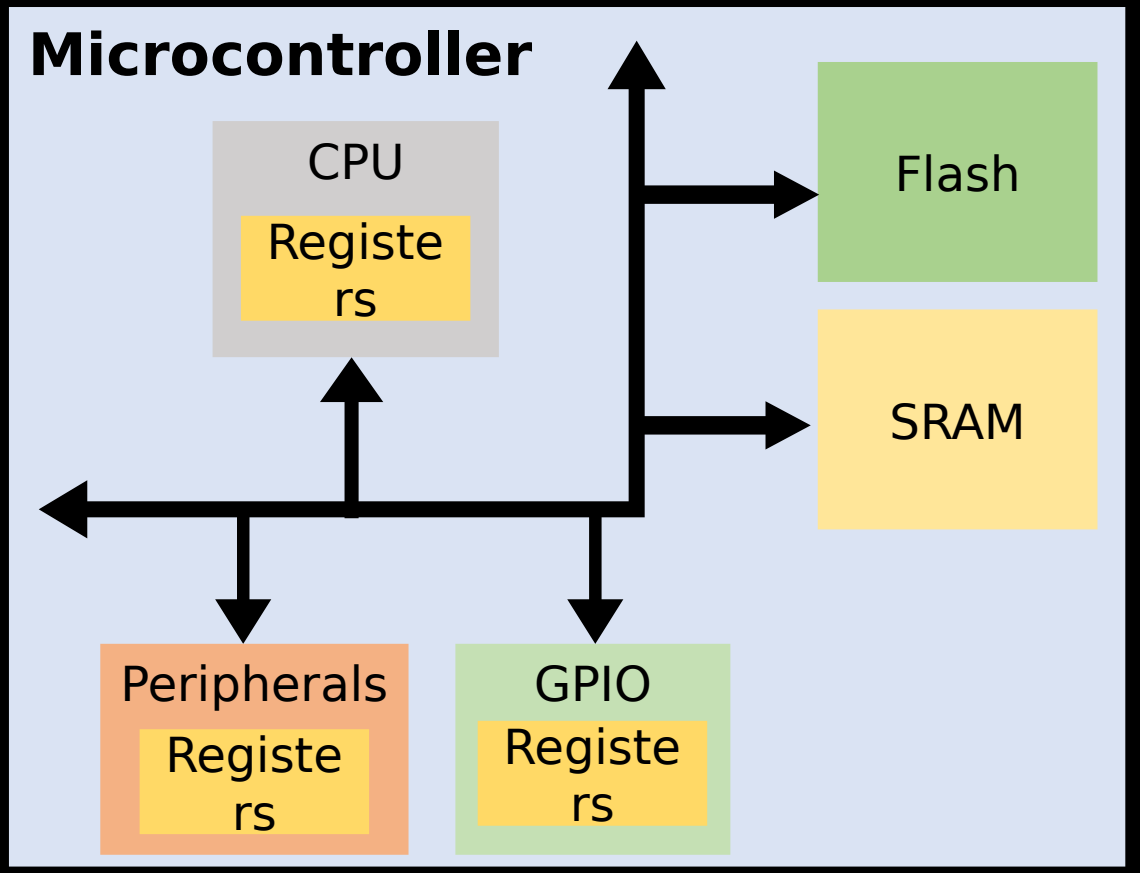
- Bit-Banded operations are atomic
  - Handled in hardware
  - No CPU Load/Store

Reduces Read-Modify-Write overhead!

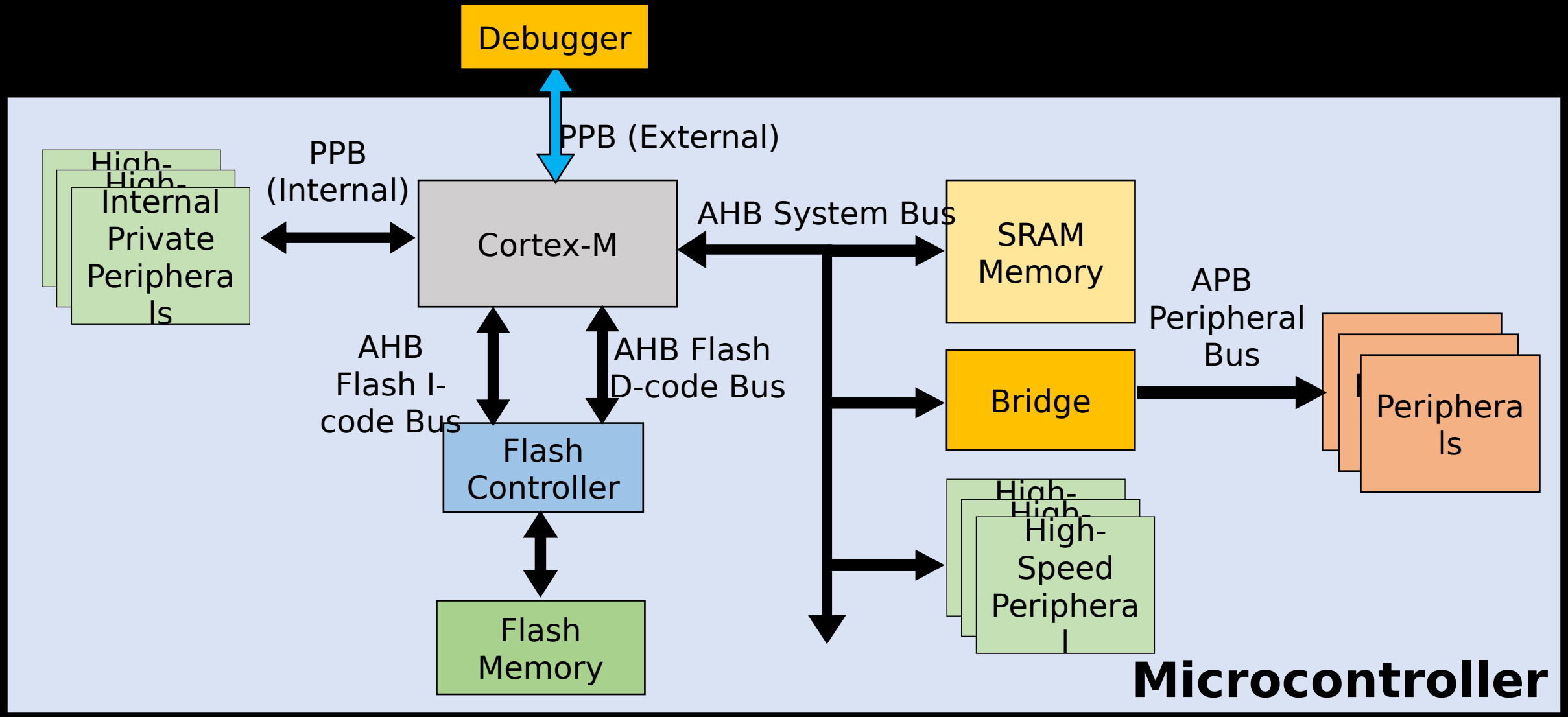


Data/Instructions Busses [S8a]

Simplistic model shows only 1 bus interface to all components of the microcontroller



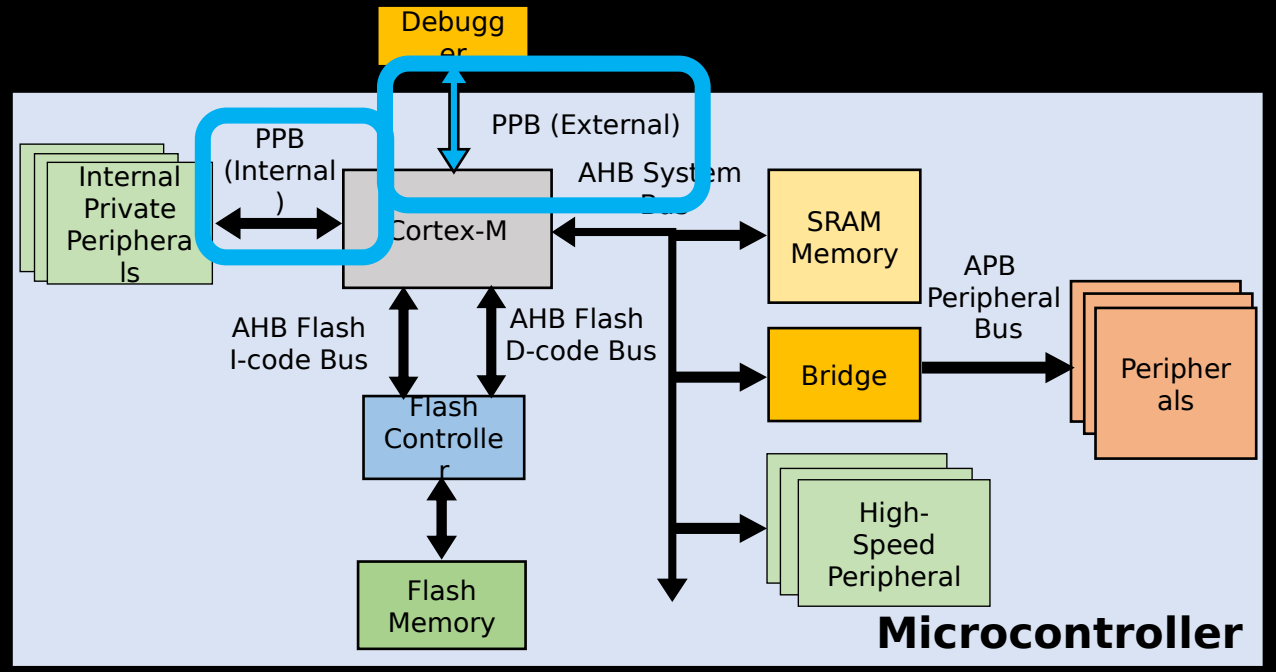
Microcontroller Busses [S8b]



Private Bus Interfaces [S9]

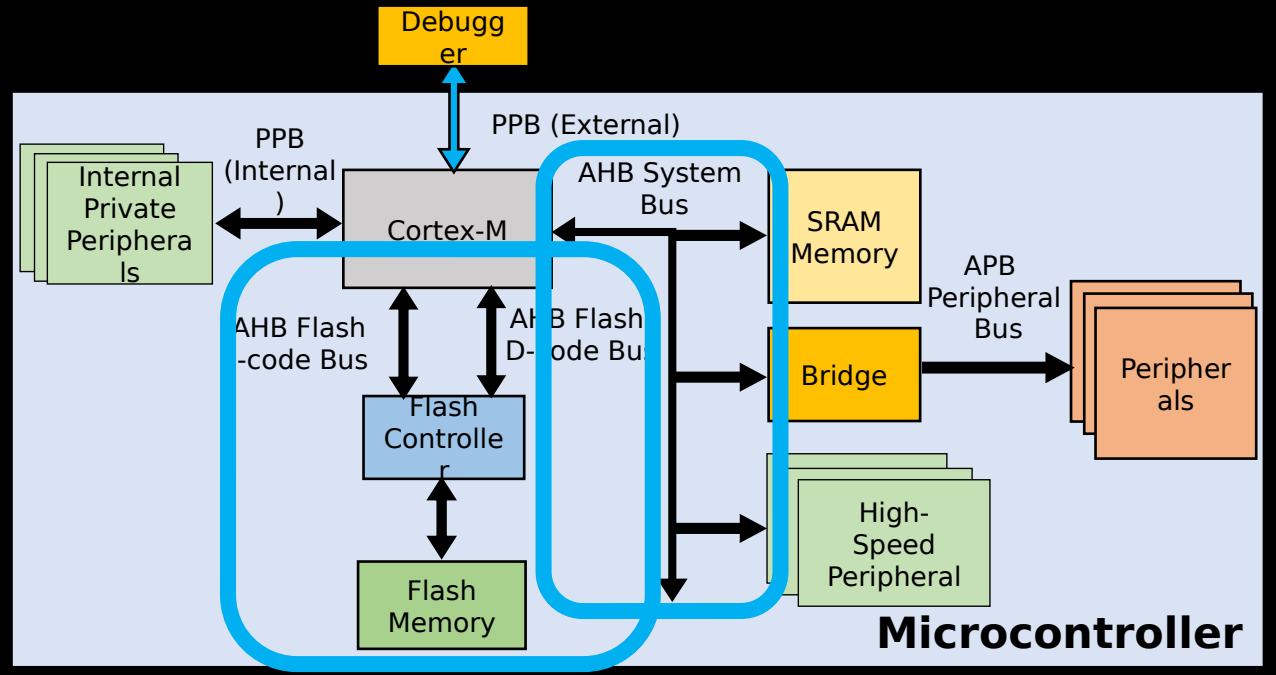
- AMBA High-Performance Bus (AHB)
  - Fast/High-Bandwidth Interfaces

- Private Cortex-M4 Busses
  - Internal Private Peripherals Bus (PPB)
    - NVIC, SCS, MPU
  - External Private Peripherals Bus (PPB)
    - External Debugging



Code and Data Busses [S10]

- AHB External Core AHB busses
  - System Bus (AHB-Lite)
    - SRAM
    - High Speed Peripherals
    - Peripheral Bridge
  - Flash Bus
    - I-Code Bus
    - D-Code Bus



Flash Busses allow for fetching of Code and Flash Data simultaneously

Peripheral Bus [S11]

- Advanced Peripheral Bus (APB)
  - Low Bandwidth Bus
  - Connected to CPU via Bridge to System Bus
- Interfaces with Low Bandwidth
  - Communications
    - UART
    - SPI
    - I2C
  - Timers