Memory Access and Manipulation

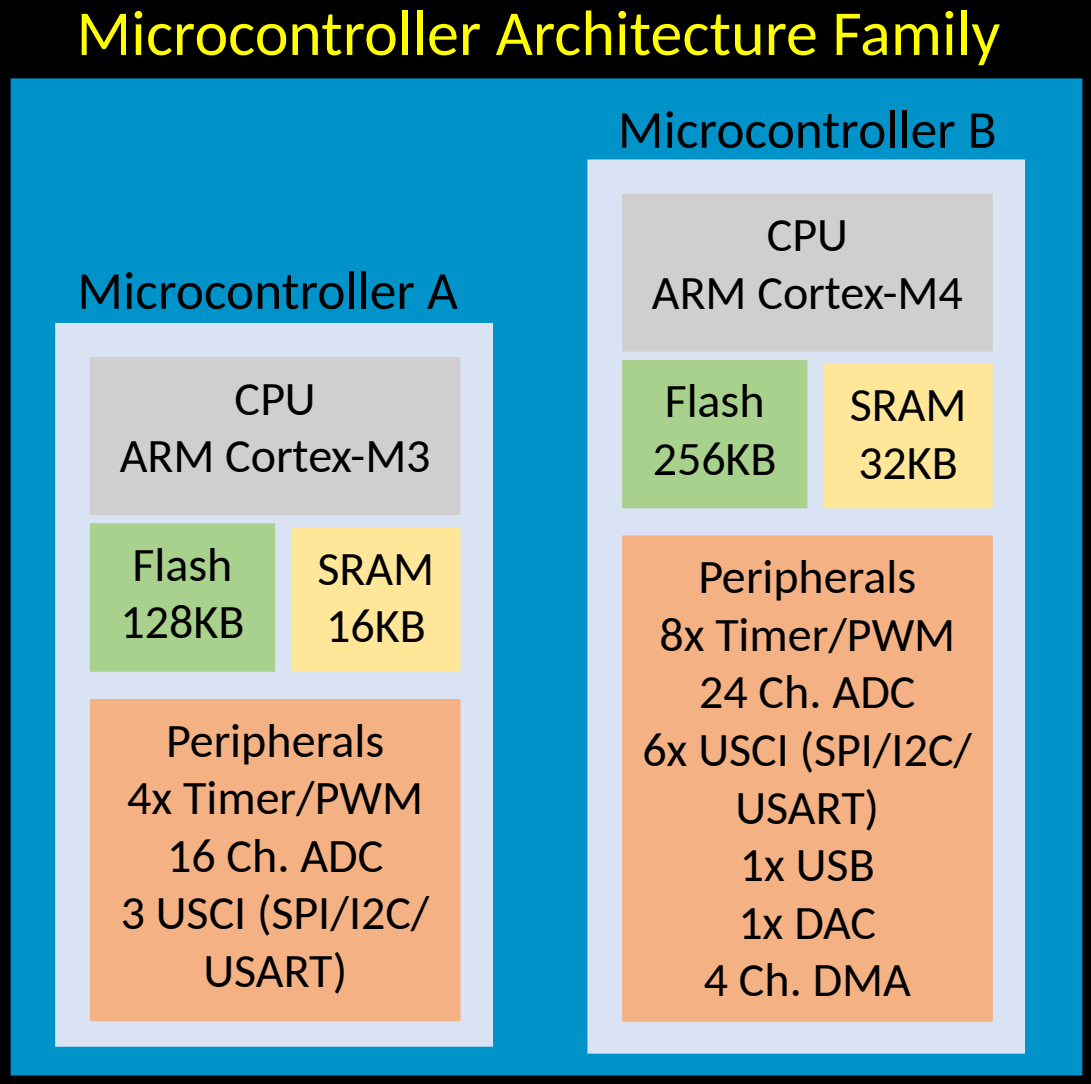
Embedded Software Essentials

Microcontroller Varieties [S1a]

- Manufacturers have multiple chips in a family with varying features
  - Flash
  - SRAM
  - Peripherals

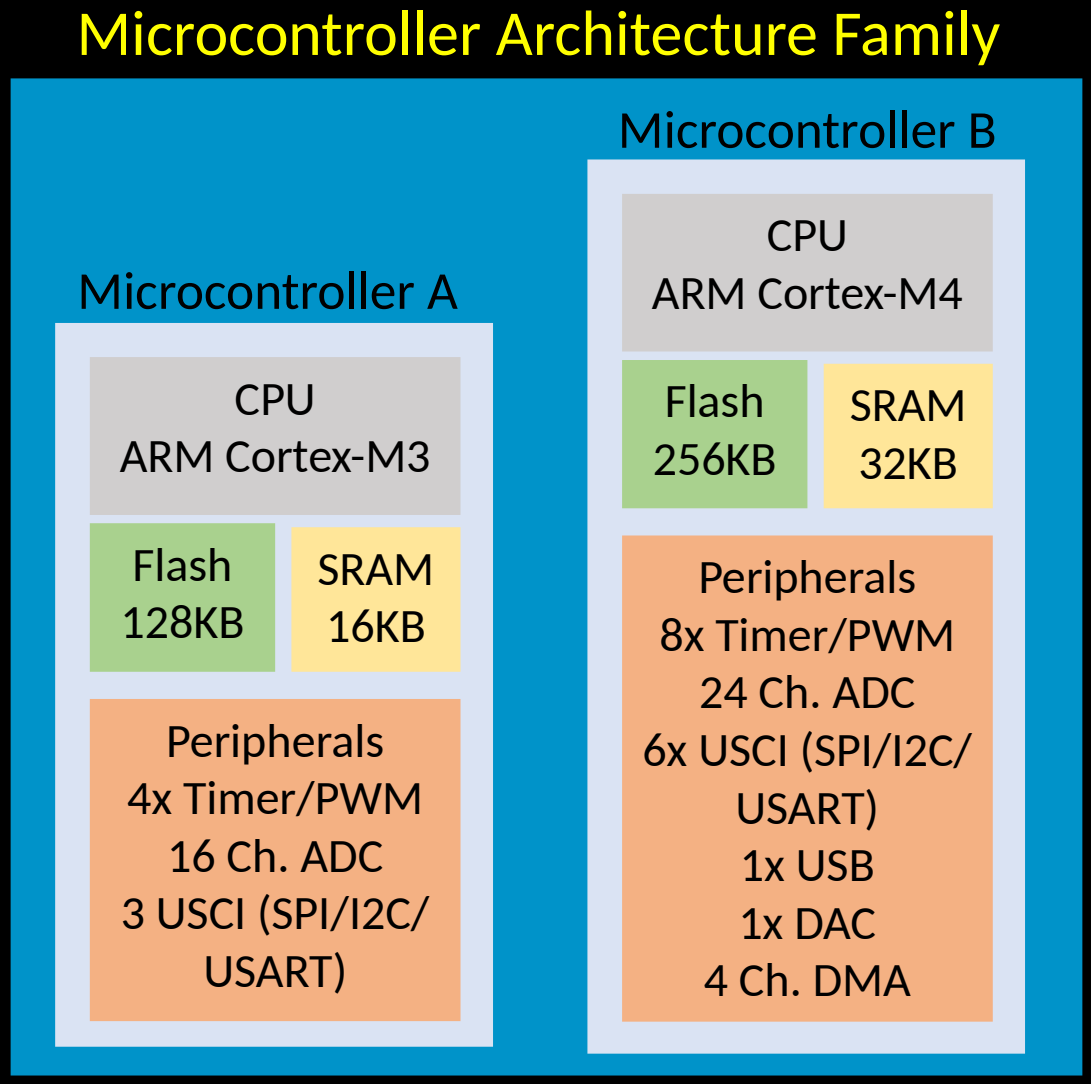
Microcontroller Varieties [S1b]

- Manufacturers have multiple chips in a family with varying features
  - Flash
  - SRAM
  - Peripherals



Microcontroller Varieties [S1c]

- Manufacturers have multiple chips in a family with varying features
  - Flash
  - SRAM
  - Peripherals
- Important to track platform differences in your software

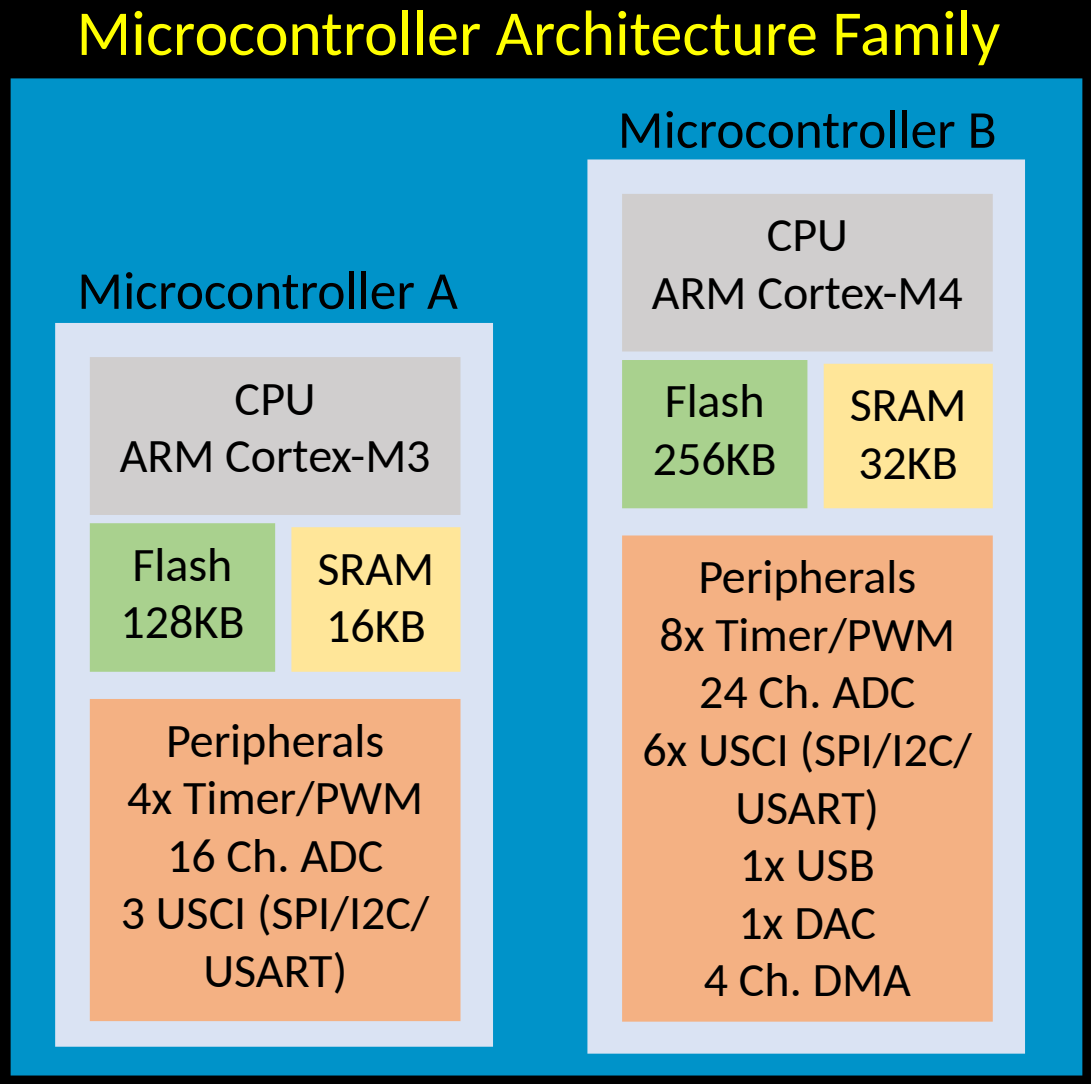


Microcontroller Varieties [S1d]

- Manufacturers have multiple chips in a family with varying features
  - Flash
  - SRAM
  - Peripherals

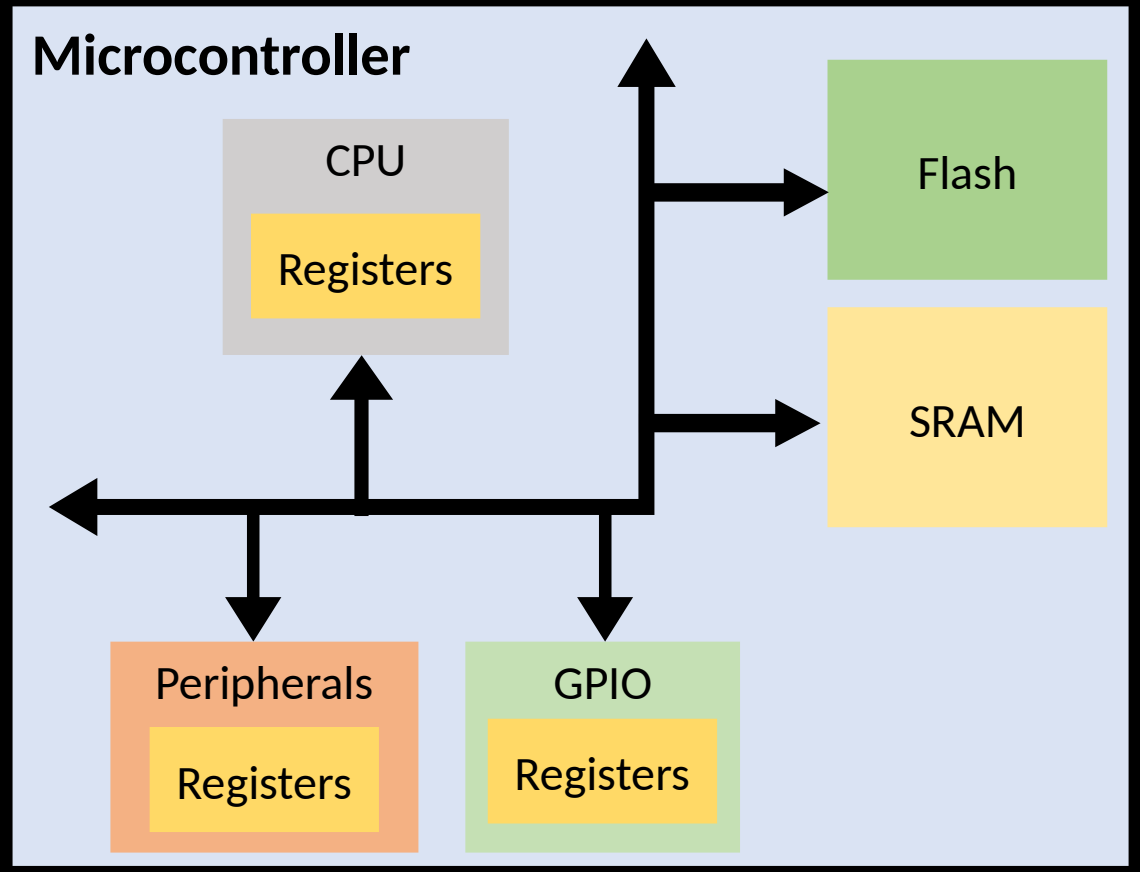
- Important to track platform differences in your software

Register Definition File
Linker File
Compiler Toolchain



Embedded System Memories [S2a]

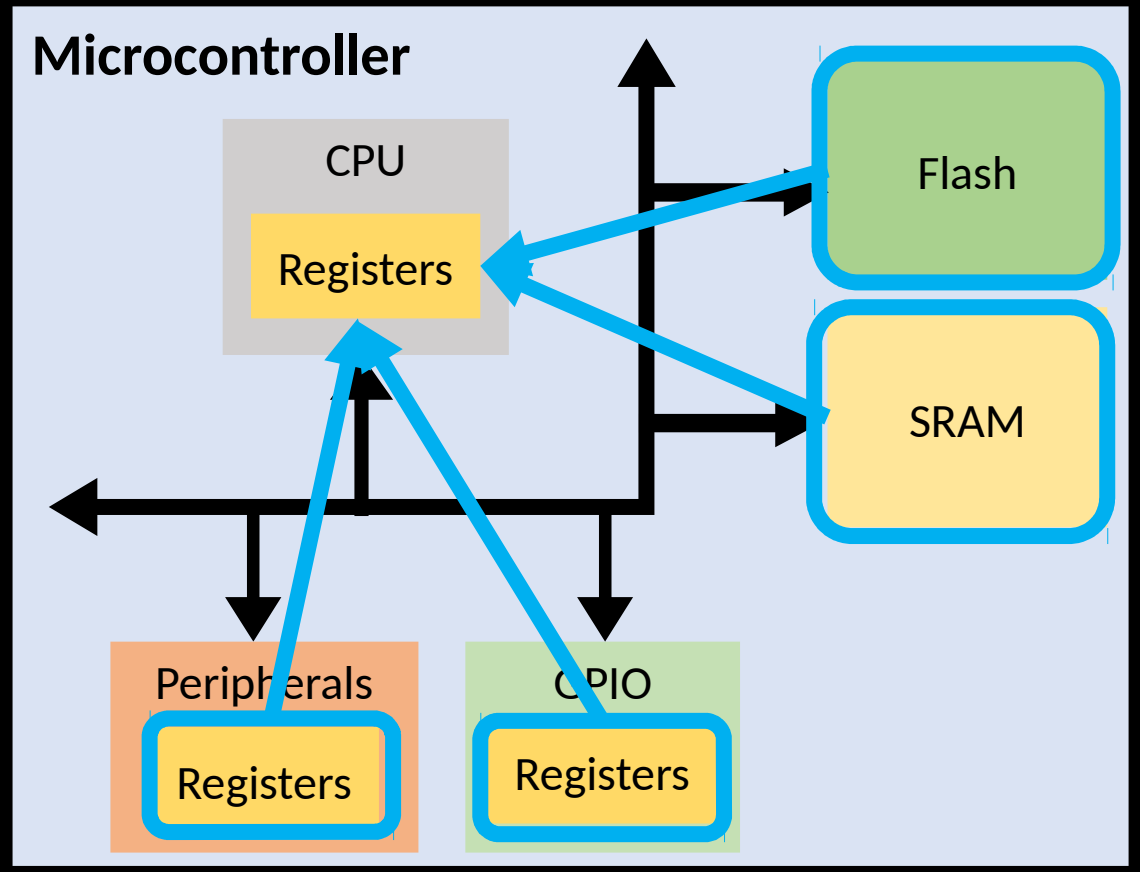
- Memories of an Embedded System
  - Code Memory (Flash)
  - Data Memory (SRAM)
  - Register Memory



Embedded System Memories [S2b]

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

Pointers used to interact with memory

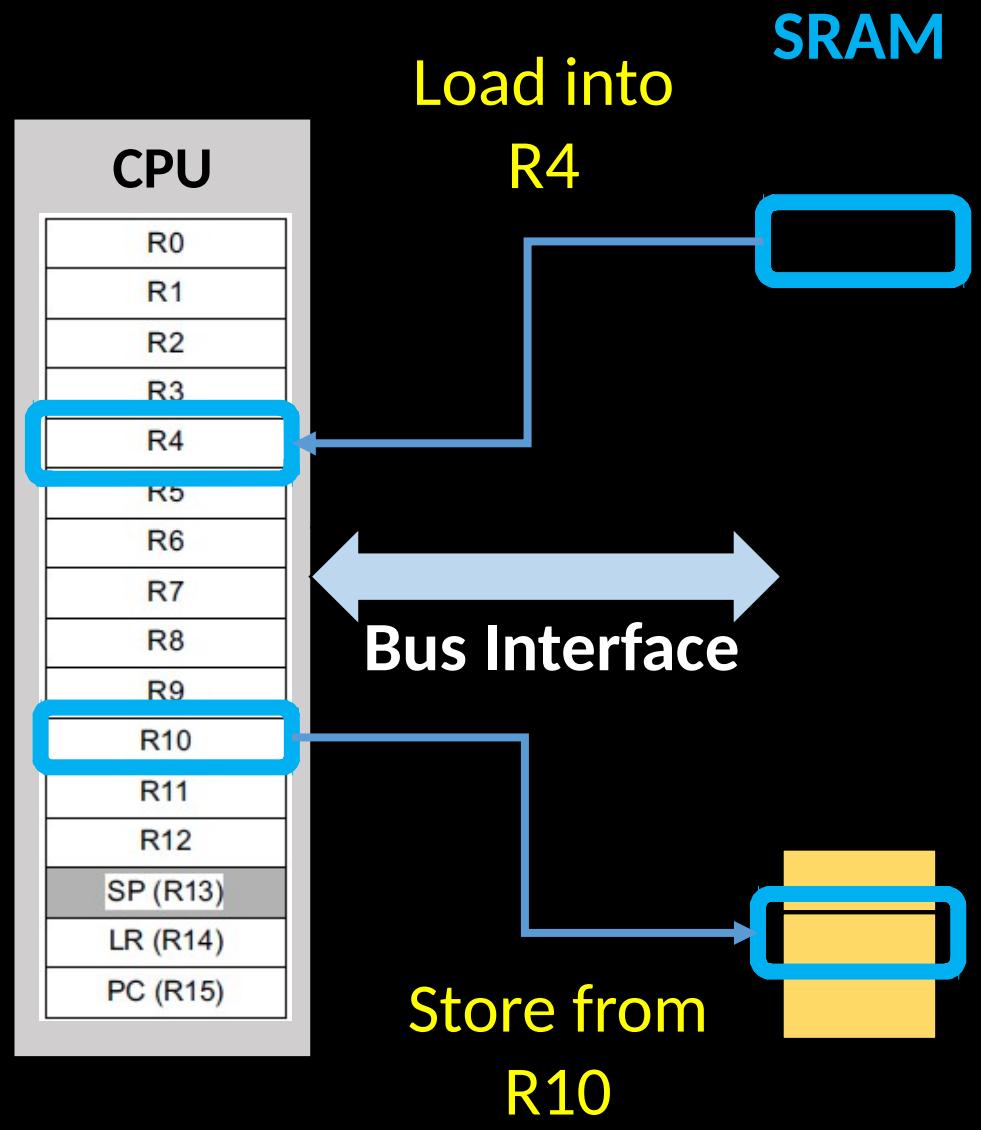


- All processing occurs in CPU
  - Data is loaded into register and then stored back to memory

Embedded System Memories [S3a]

- CPU registers constantly change
  - Data is **loaded** in from memory
  - Data is **stored** back to memory

Data manipulation needed at bit level

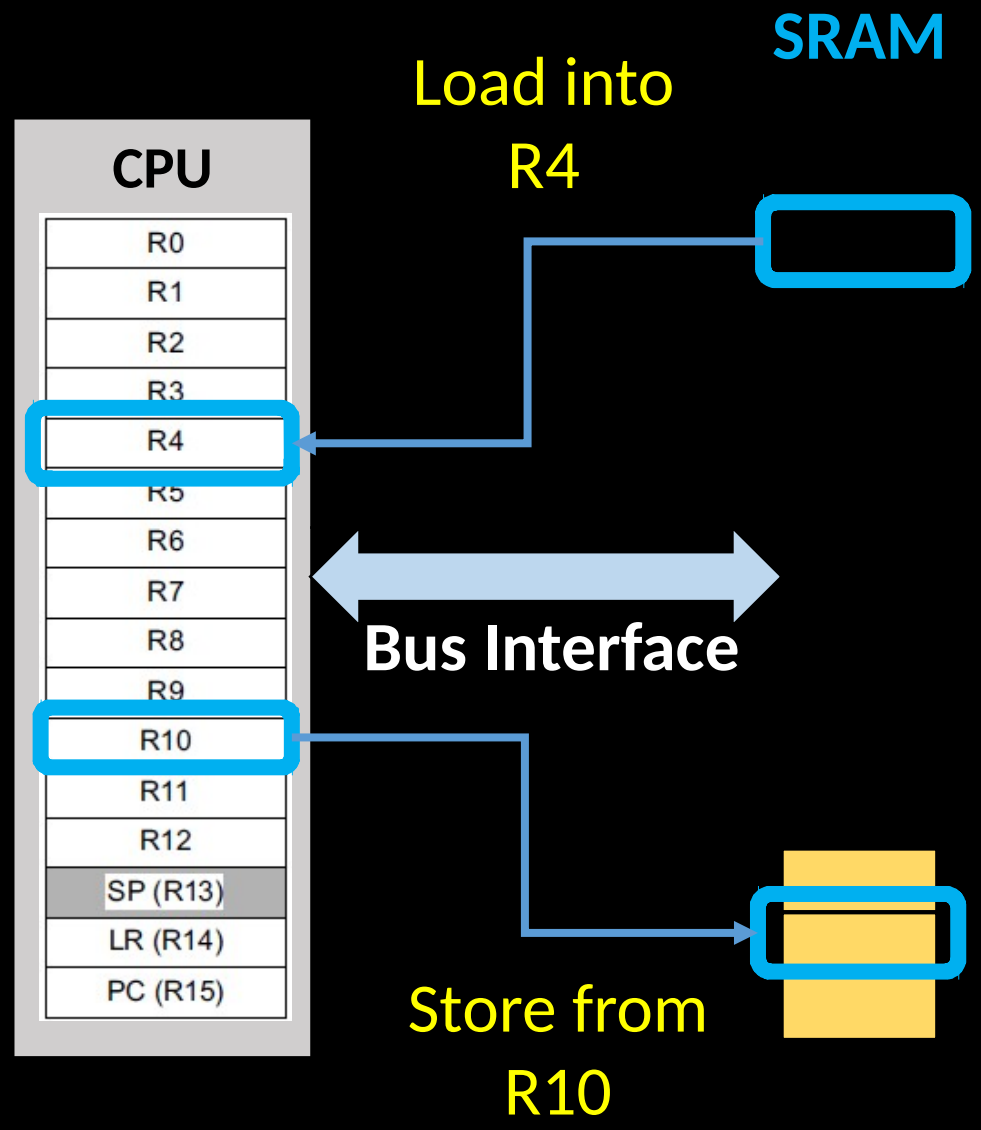


Embedded System Memories [S3b]

- CPU registers constantly change
  - Data is **loaded** in from memory
  - Data is **stored** back to memory

Data manipulation needed at bit level

- Bit Manipulation used to configure microcontrollers



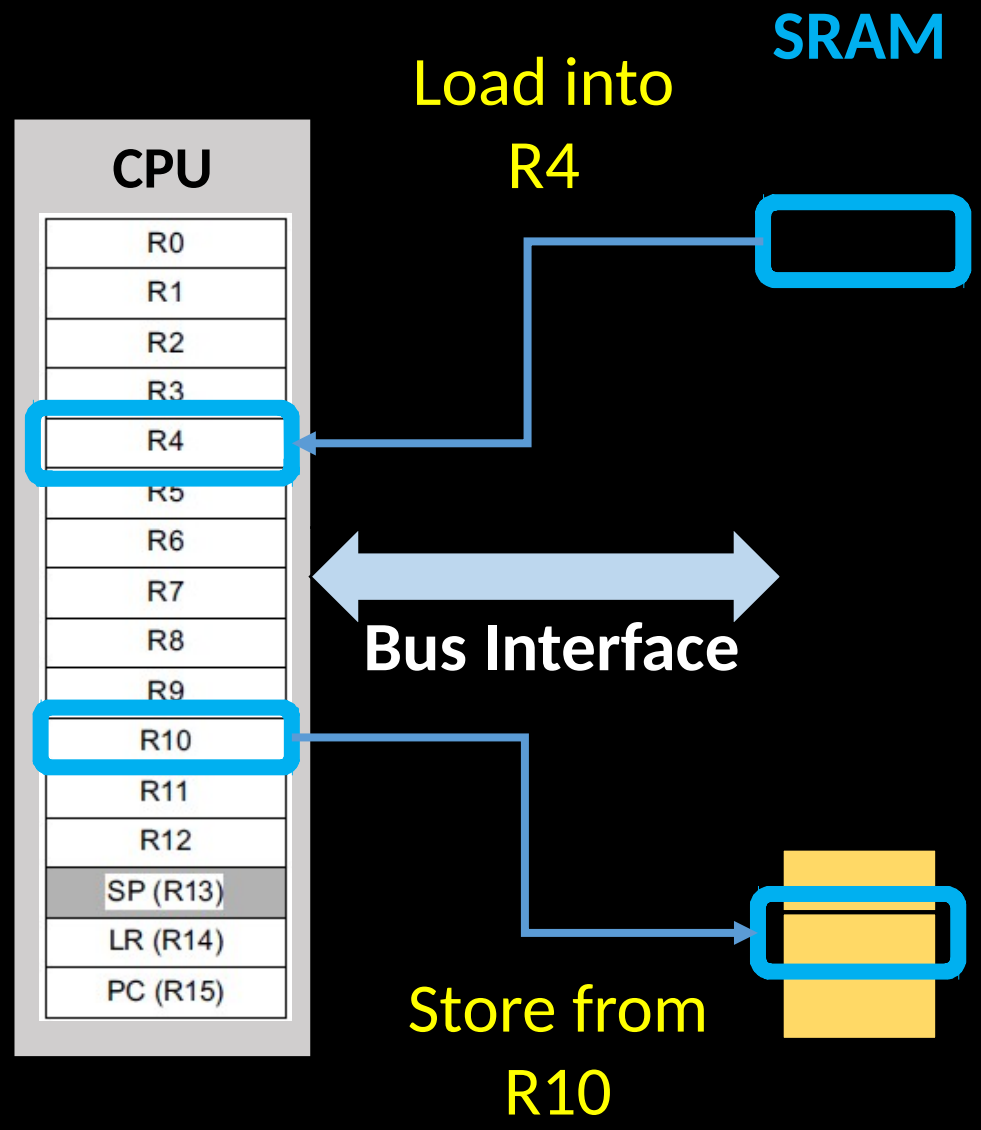
Embedded System Memories [S3c]

- CPU registers constantly change
  - Data is **loaded** in from memory
  - Data is **stored** back to memory

Data manipulation needed at bit level

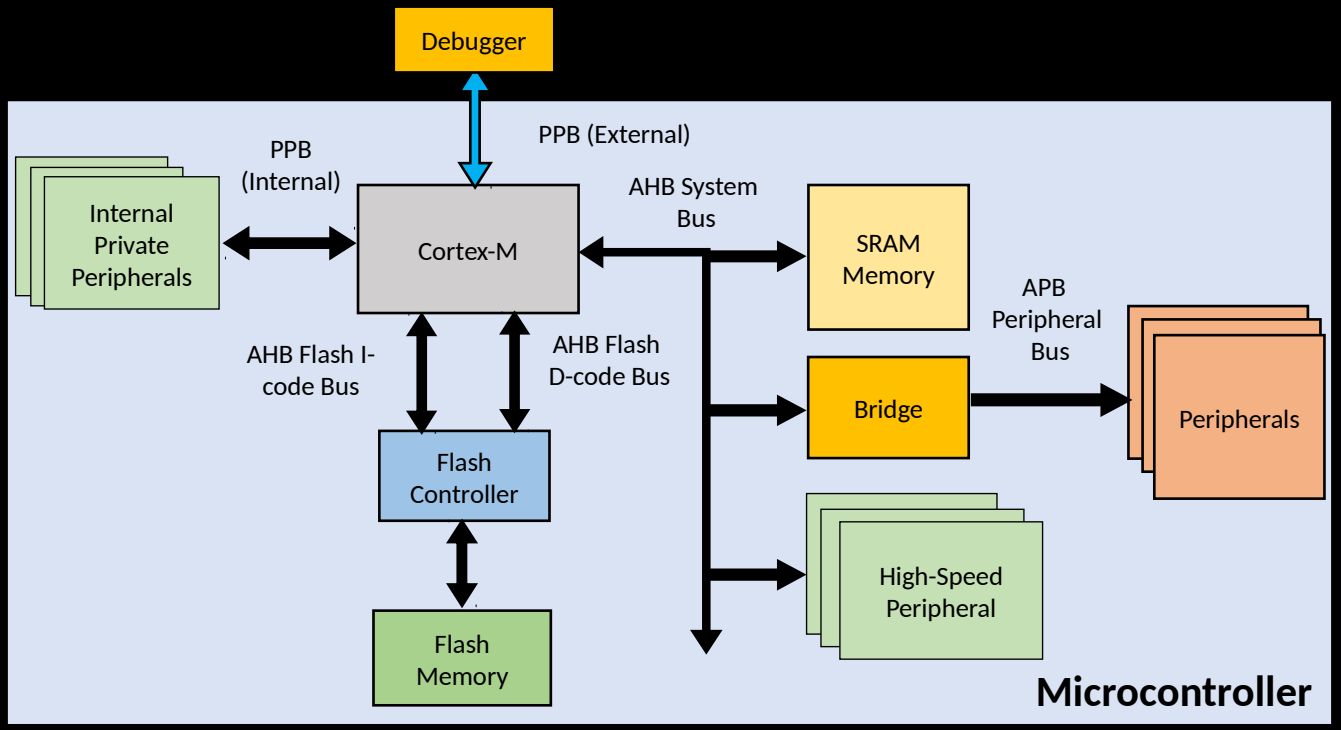
- Bit Manipulation used to configure microcontrollers

  Use of Bitwise and Arithmetic Operators!



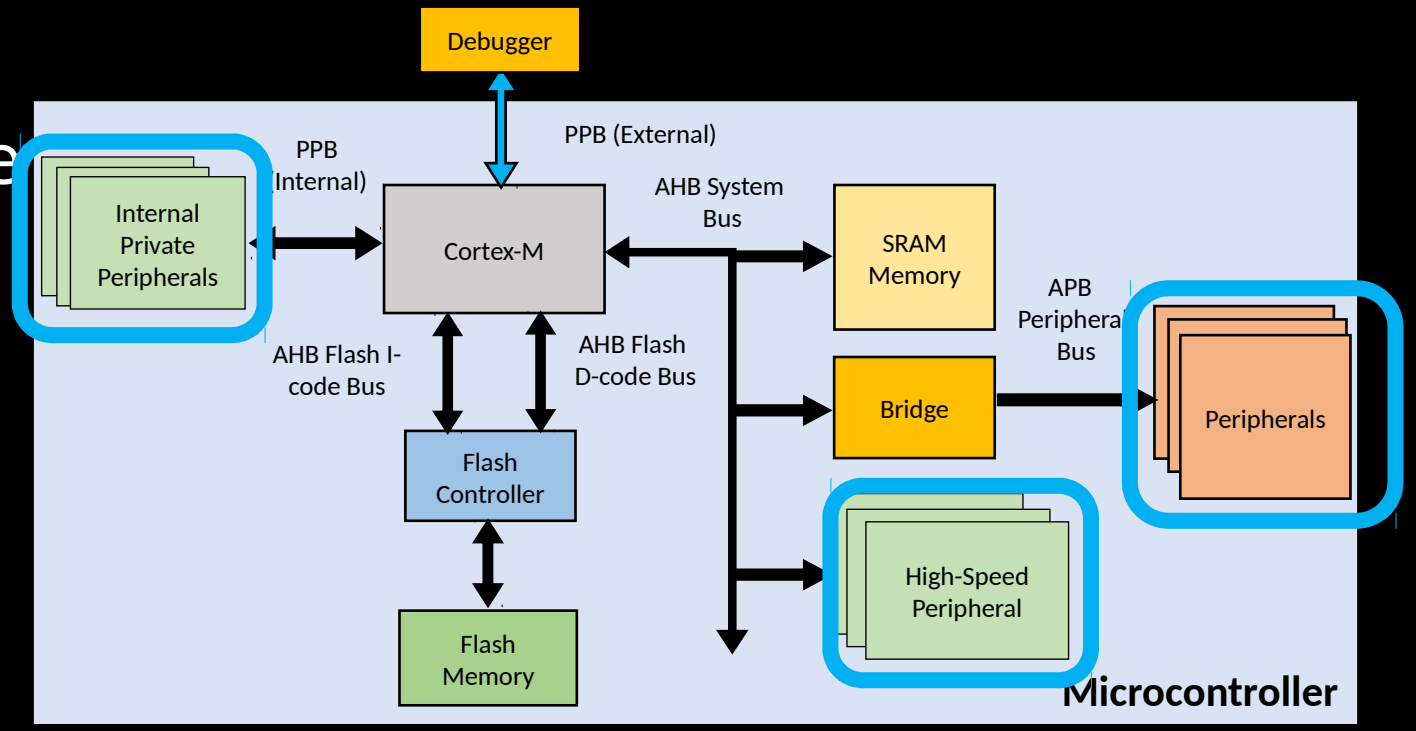
Peripherals [S4a]

- Peripherals: External to CPU specialized support hardware



Peripherals [S4b]

- Peripherals: External to CPU specialized support hardware

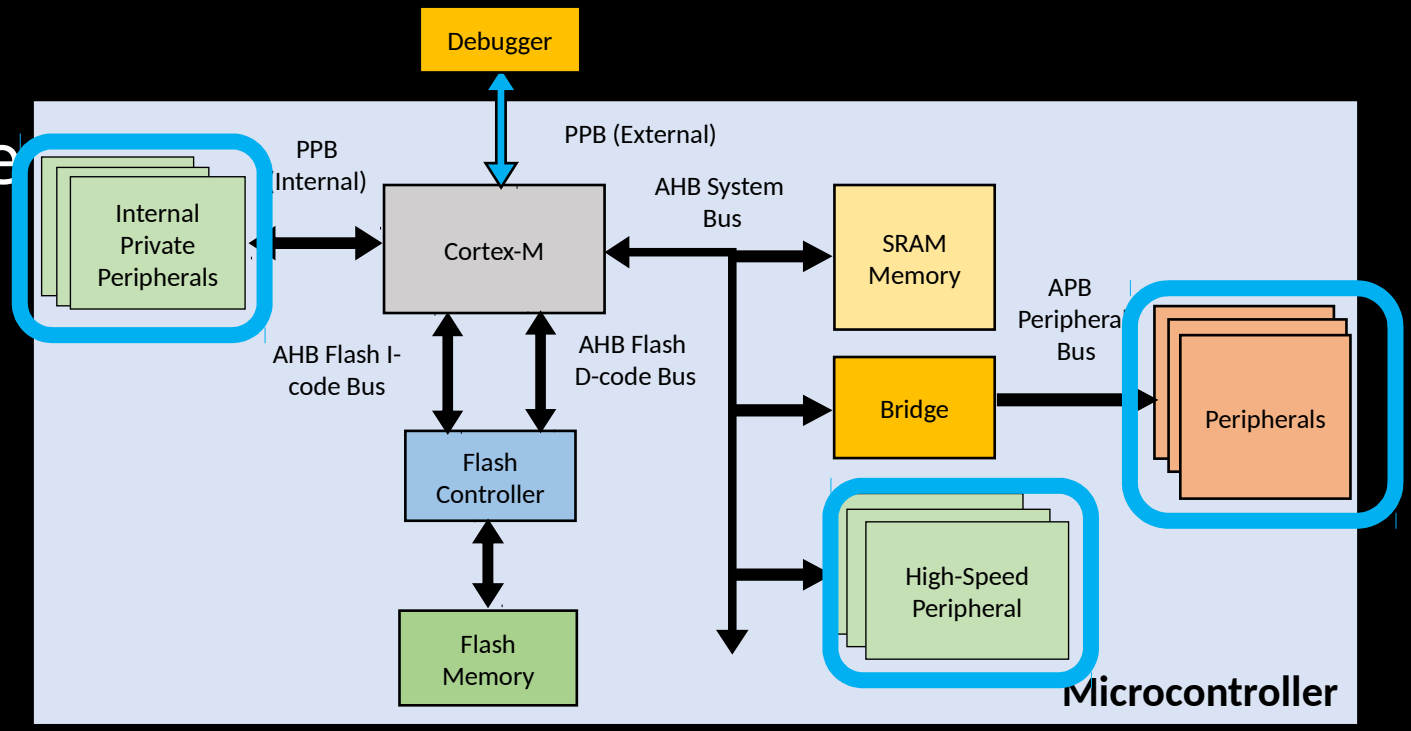


Many Different Types:

- High / Low Speed
- Internal / External to CPU

Peripherals [S4c]

- Peripherals: External to CPU specialized support hardware
- Technical Reference Manual provides configuration guide
  - Functional Description
  - Register Description
  - Addresses
  - Order of Operations

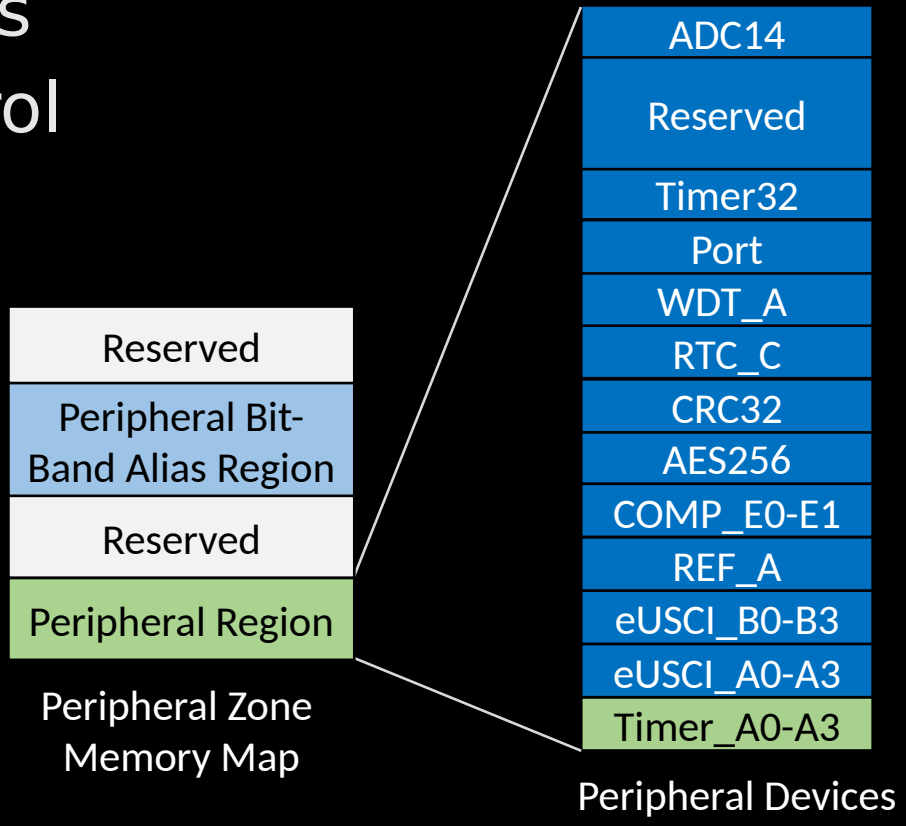


Many Different Types:

- High / Low Speed
- Internal / External to CPU

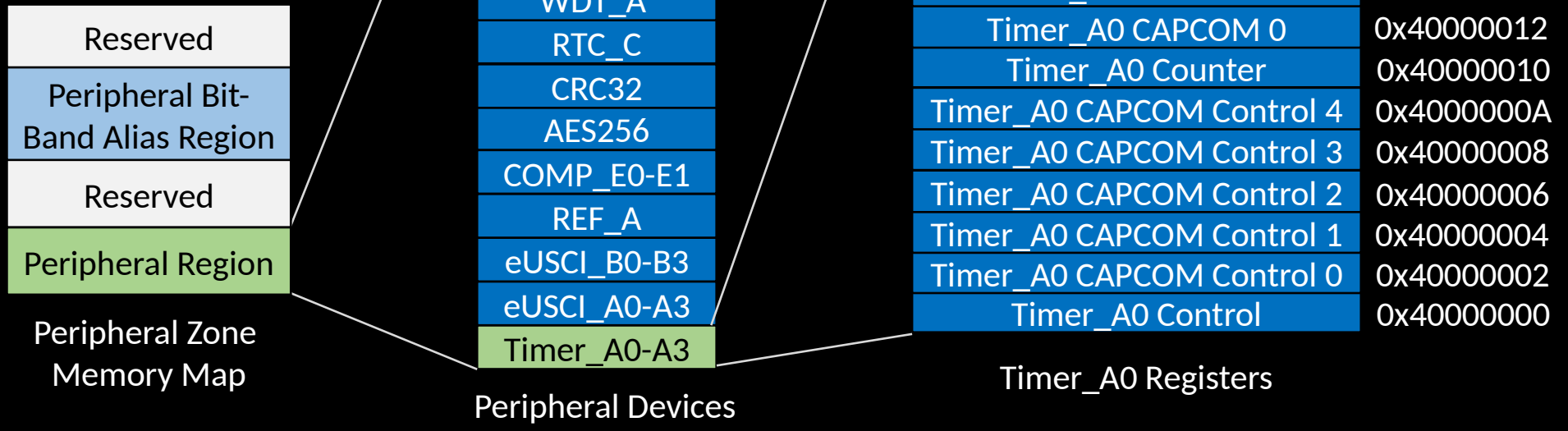
Timer A0 Peripheral [S5a]

- Peripherals have multiple registers
  - Status
  - Control
  - Data



Timer A0 Peripheral [S5b]

- Peripherals have multiple registers
  - Status
  - Control
  - Data



Timer A0 Peripheral [S5c]

- Peripherals have multiple registers
  - Status
  - Control
  - Data

Timer_A0 Expansion 0	0x40000020
Timer_A0 Interrupt Vector	0x4000002E
Timer_A0 CAPCOM 4	0x4000001A
Timer_A0 CAPCOM 3	0x40000018
Timer_A0 CAPCOM 2	0x40000016
Timer_A0 CAPCOM 1	0x40000014
Timer_A0 CAPCOM 0	0x40000012
Timer_A0 Counter	0x40000010
Timer_A0 CAPCOM Control 4	0x4000000A
Timer_A0 CAPCOM Control 3	0x40000008
Timer_A0 CAPCOM Control 2	0x40000006
Timer_A0 CAPCOM Control 1	0x40000004
Timer_A0 CAPCOM Control 0	0x40000002
Timer_A0 Control	0x40000000

Timer_A0 Registers

Register Definition File (msp.h)

```
/* Timer Peripheral Device Structure Overlay */
typedef struct {
  __IO uint16_t CTL;
  __IO uint16_t CCTL[7];
  __IO uint16_t R;
  __IO uint16_t CCR[7];
  __IO uint16_t EX0;
  uint16_t RESERVED0[6];
  __I  uint16_t IV;
} Timer_A_Type;

/* Define the Base Address of Peripheral Regions */
#define PERIPH_BASE       ((uint32_t) 0x40000000)
#define TIMER_A0_BASE  (PERIPH_BASE+0x00000000)

/ * Multiple Timer Modules, Different Addresses */
#define TIMER_A0  ((Timer_A_Type *) TIMER_A0_BASE)
...
```

Code Improvements [S6a]

- General improvements can include
  - Runtime Optimizations
  - Architectural Improvements
  - Dynamic Function Calling

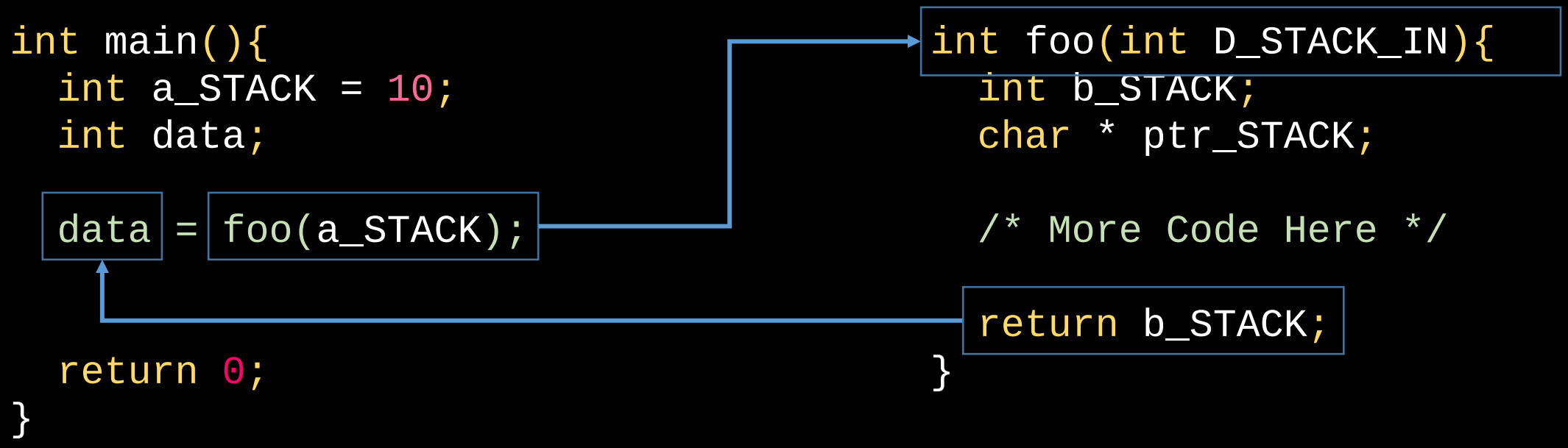
```
int main(){
  int a_STACK = 10;
  int data;

  data = foo(a_STACK);


  return 0;
}
```

```
int foo(int D_STACK_IN){
  int b_STACK;
  char * ptr_STACK;

  /* More Code Here */

  return b_STACK;
}
```

Code Improvements [S6b]

- General improvements can include
  - Runtime Optimizations
  - Architectural Improvements
  - Dynamic Function Calling

Example: Reducing function call overhead due branch and register/state saving



Hardware Abstraction [S7]

- Abstract hardware layers from higher level software
  - Example: Input Output Library
- Higher level software should be independent of low level firmware