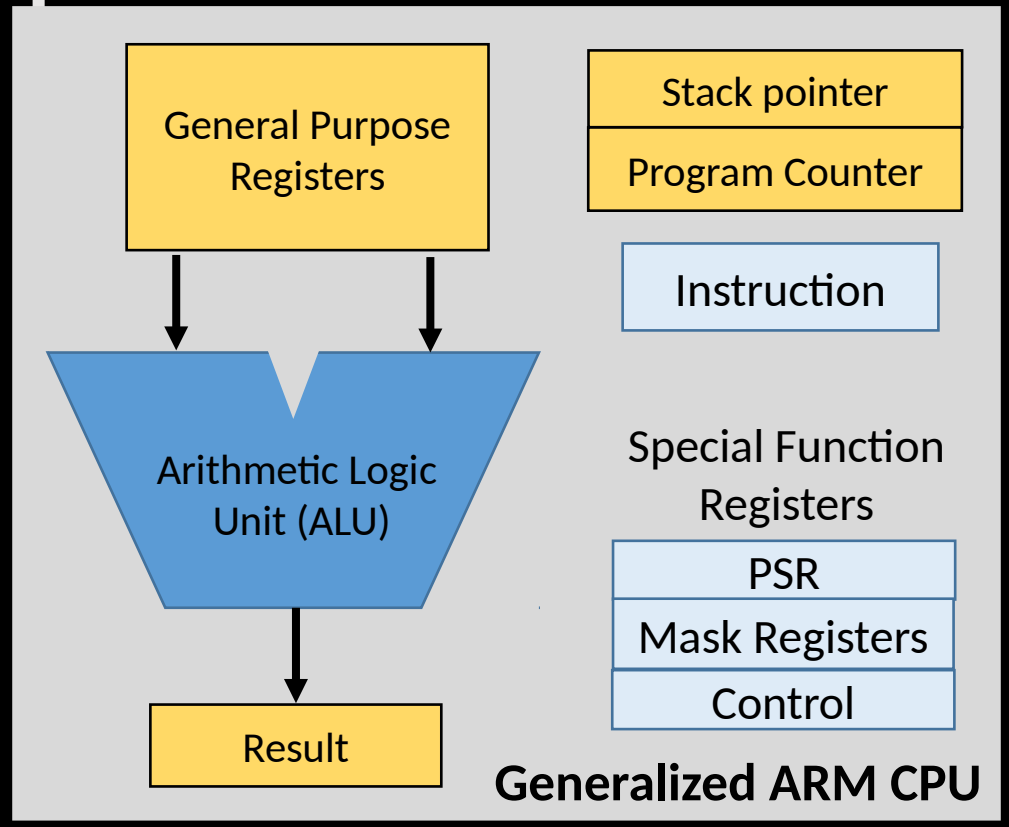
Bit Manipulation

Embedded Software Essentials

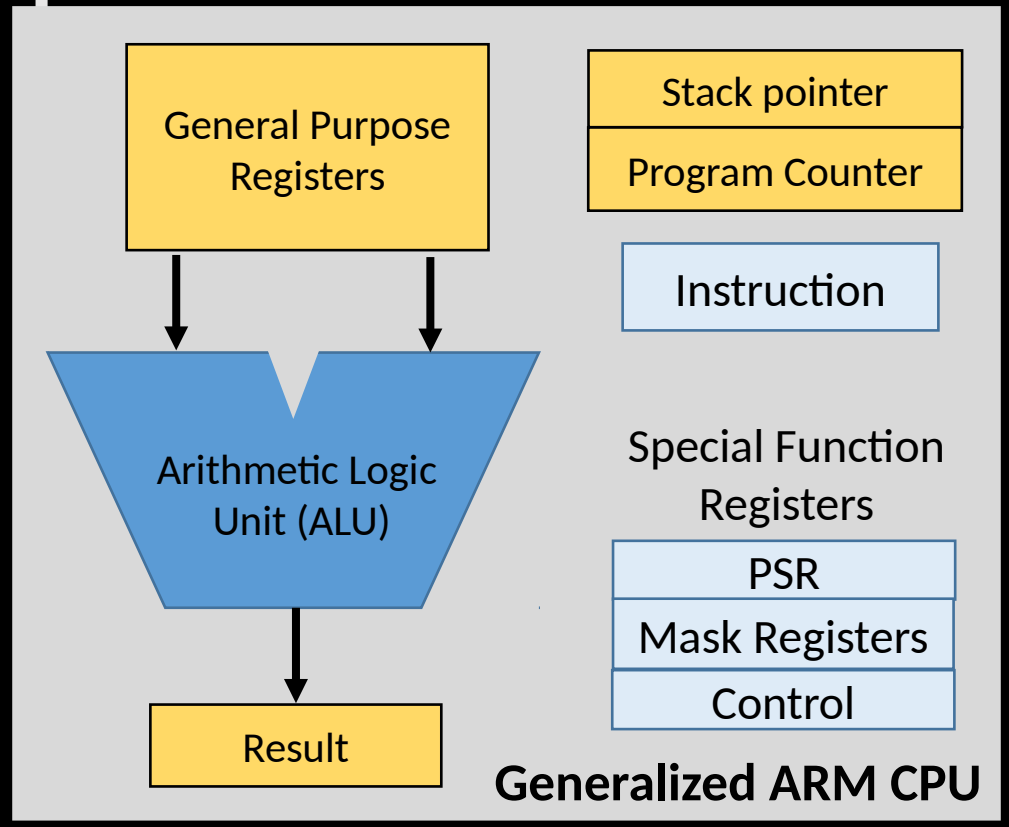
Bit Manipulation [S1a]

- Bit Manipulation used to configure microcontrollers



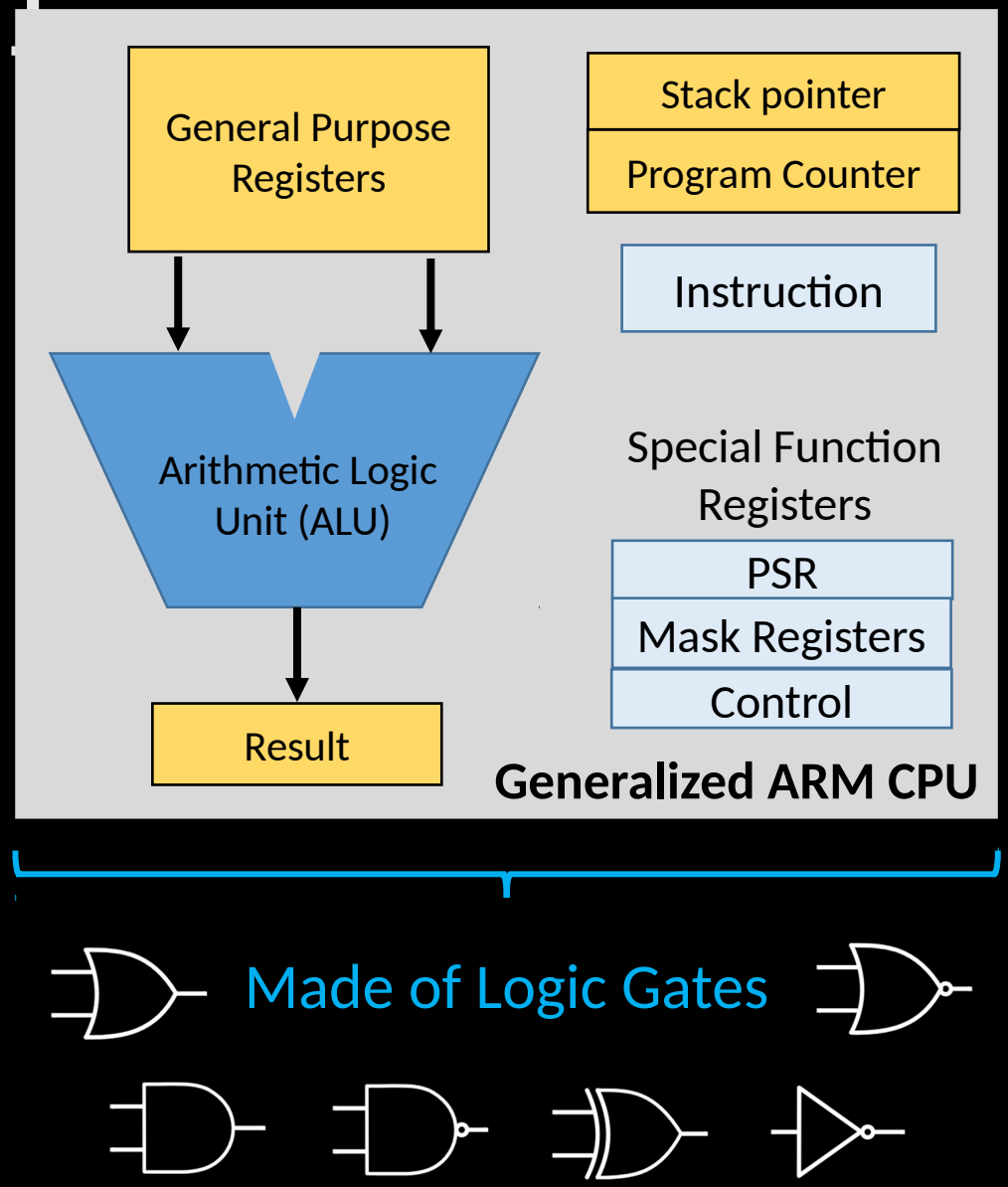
Bit Manipulation [S1b]

- Bit Manipulation used to configure microcontrollers
- All arithmetic operations can be done with bitwise operations



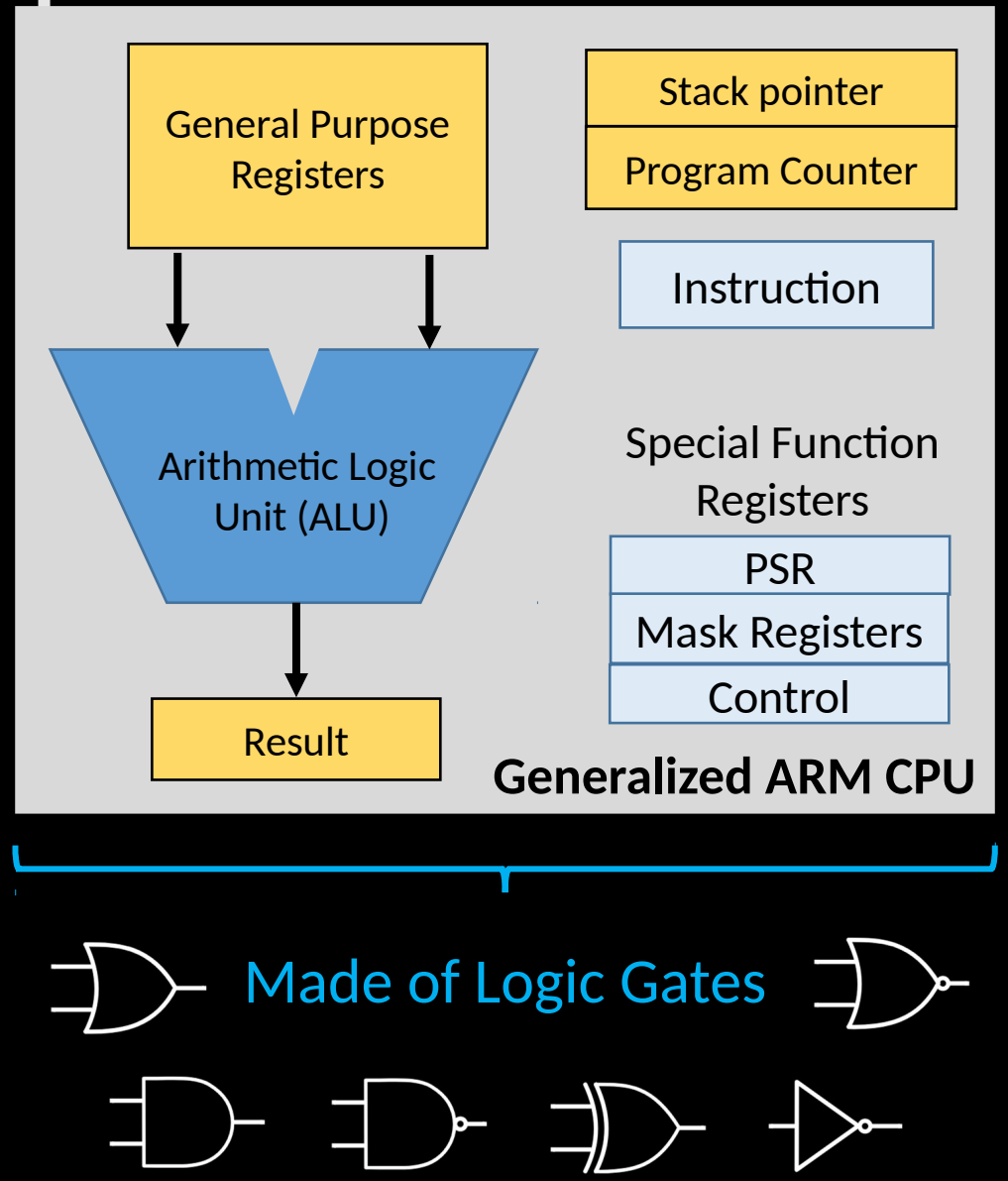
Bit Manipulation [S1c]

- Bit Manipulation used to configure microcontrollers
- All arithmetic operations can be done with bitwise operations



Bit Manipulation [S1d]

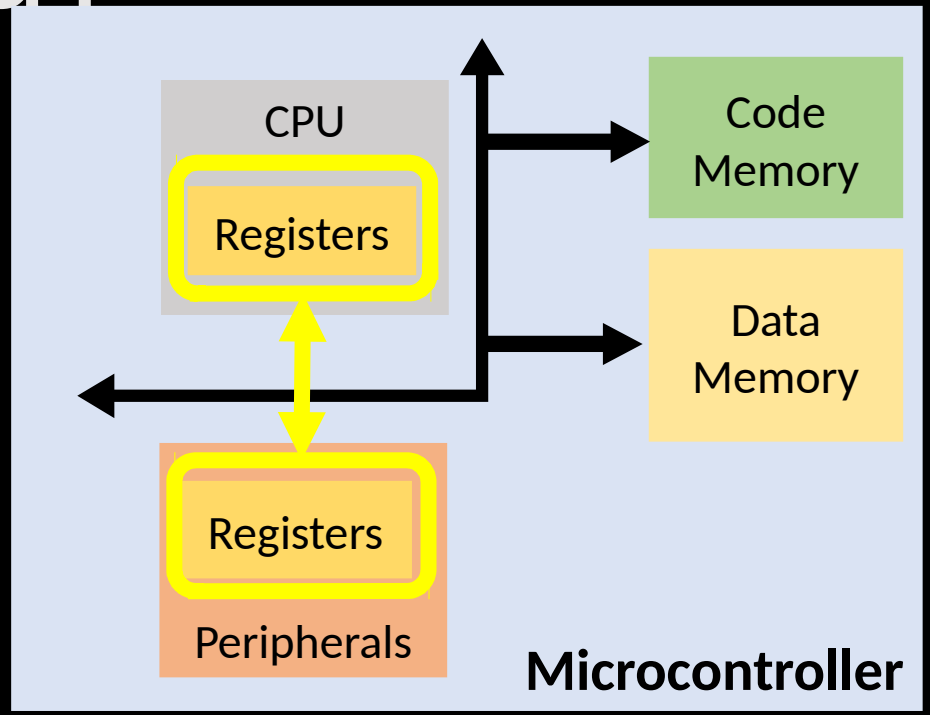
- Bit Manipulation used to configure microcontrollers
- All arithmetic operations can be done with bitwise operations
- Bitwise operators are needed to configure peripherals



Bitwise Operators [S2a]

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

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

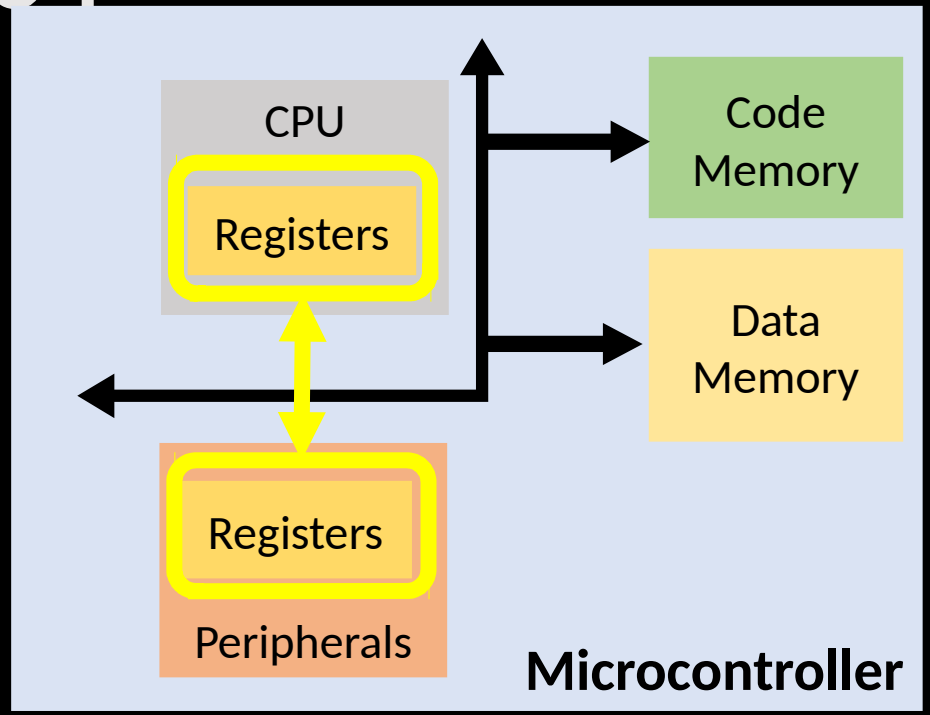


Bitwise Operators [S2b]

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

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

- C-programming provides bitwise operators
  - << >> & | ^ ~

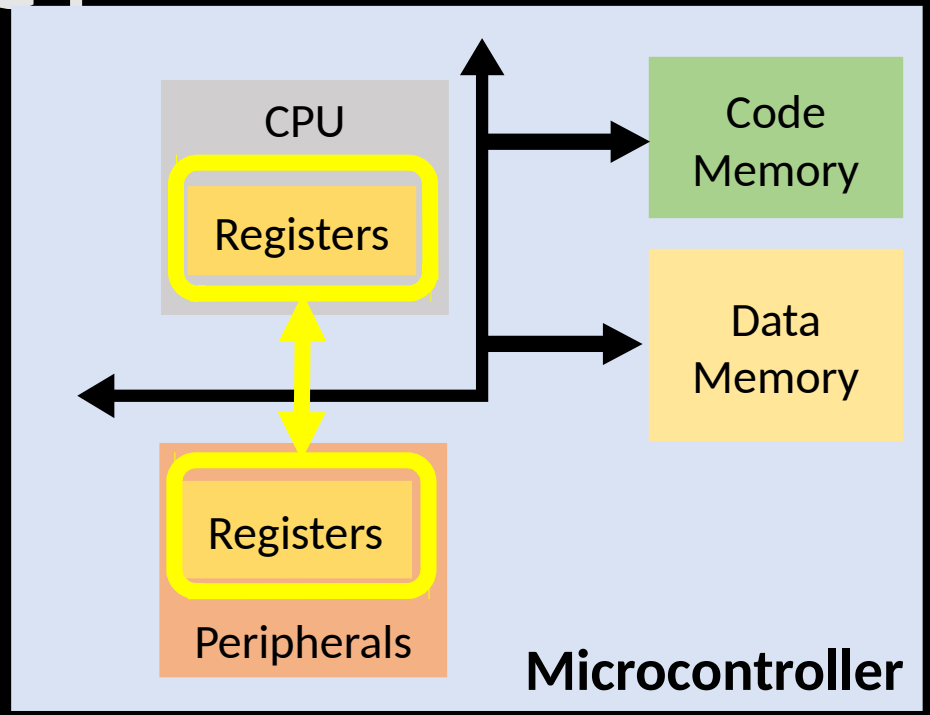


Bitwise Operators [S2c]

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

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

- C-programming provides bitwise operators
  - `<<` `>>` `&` `|` `^` `~`




```
foo = foo | 0x10;
```

foo	?	?	?	?	?	?	?	?
0x10	0	0	0	1	0	0	0	0
foo	?	?	?	1	?	?	?	?

Bitwise Operators [S3a]

- C-programming provides bitwise operators
  - `<<  >>  & | ^ ~`

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

Set 4th bit: `*ptr |= 0x10;`

Clear 4th bit: `*ptr &= ~(0x10);`

Toggle 4th bit: `*ptr ^= 0x10;`

Bitwise Operators [S3b]

- C-programming provides bitwise operators
  - `<<  >>  & | ^ ~`

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

Set 4th bit: `*ptr |= 0x10;`

Clear 4th bit: `*ptr &= ~(0x10);`

Toggle 4th bit: `*ptr ^= 0x10;`

All bits preserved **except** bit 4 using logical assignment combination

Bitwise Operators [S3c]

- C-programming provides bitwise operators
  - `<<  >>  & | ^ ~`

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

Set 4th bit: `*ptr |= 0x10;`

Clear 4th bit: `*ptr &= ~(0x10);`

Toggle 4th bit: `*ptr ^= 0x10;`

All bits preserved **except** bit 4 using logical assignment combination

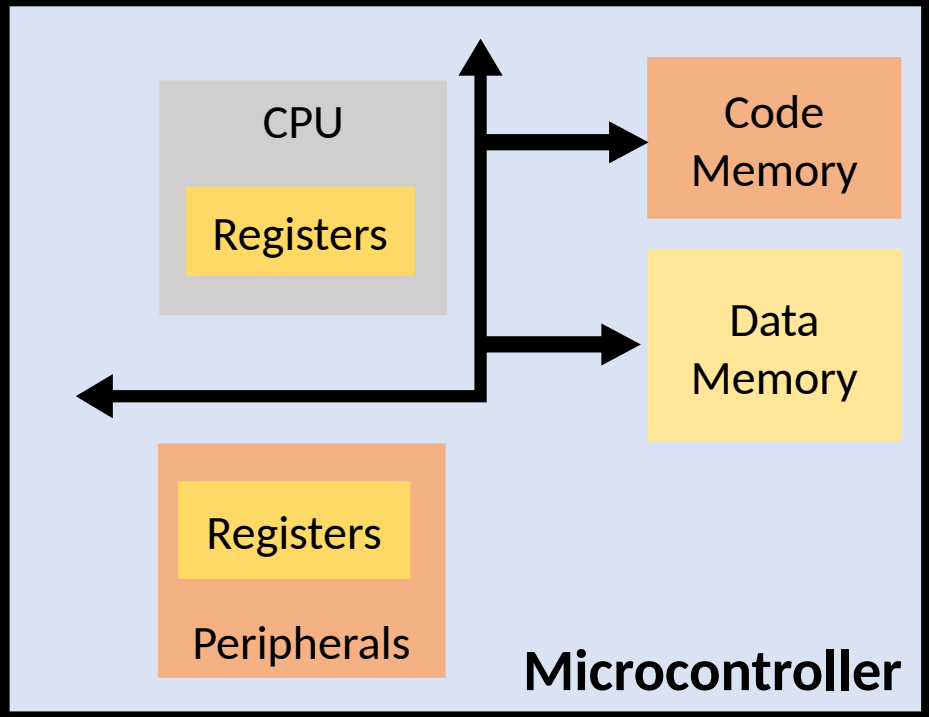
Combine logic with assignment, performs a read, modify, write

Bitwise Example: OR [S4a]

Set Bits 4 & 5:

```
uint8_t foo = 0x84;

foo = foo | 0x30;
```



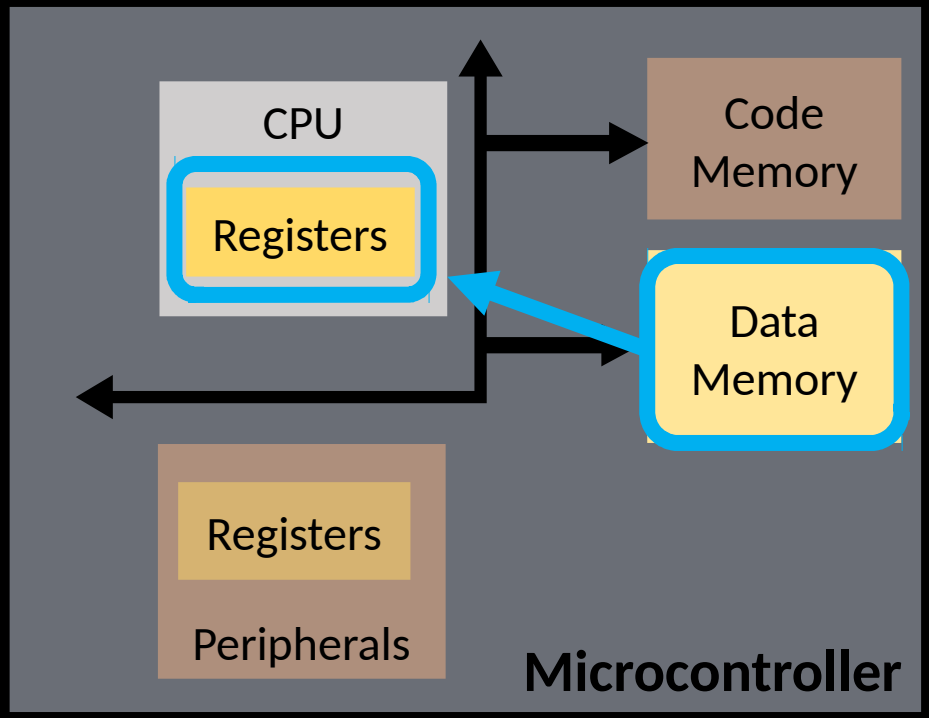
Bitwise Example: OR [S4b]

Set Bits 4 & 5:

```
uint8_t foo = 0x84;

foo = foo | 0x30;
```

This performs a READ to load foo into CPU registers



Bitwise Example: OR [S4c]

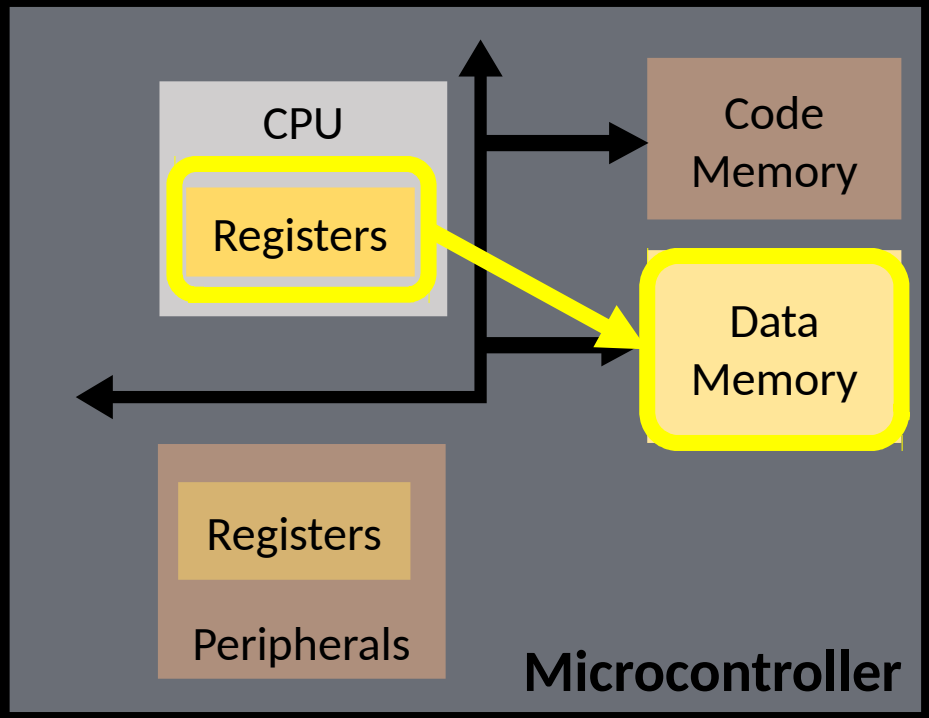
Set Bits 4 & 5:

```
uint8_t foo = 0x84;

foo = foo | 0x30;
```

Performs a WRITE to update memory

This performs a READ to load foo into CPU registers



Bitwise Example: OR [S4d]

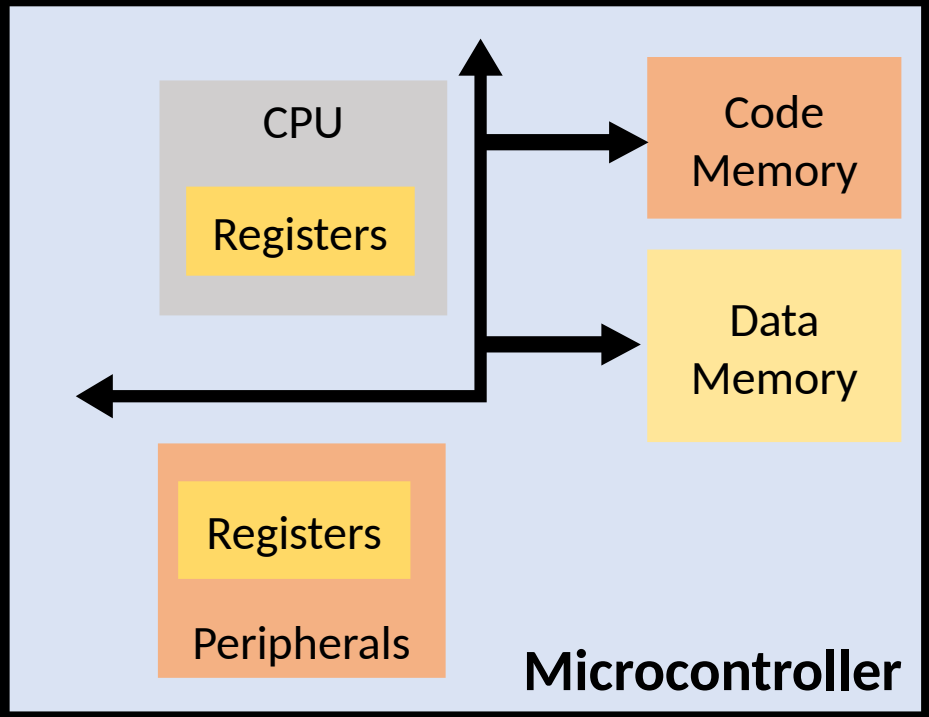
Set Bits 4 & 5:

```
uint8_t foo = 0x84;

foo |= 0x30;
```

Still performs a Read, Modify, Write

Provides a cleaner shorthand for same expression



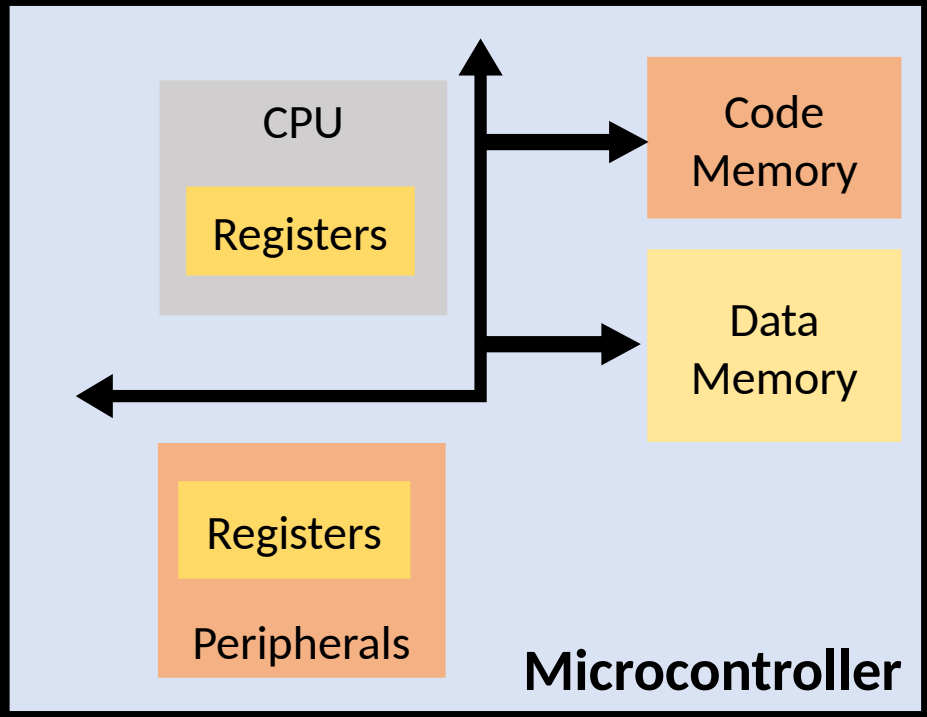
Alternatively:

```
foo |= (0x03 << 4);
```

Bitwise Example: & [S5a]

Clear Bits 6 & 7:

```
uint8_t foo = 0xFF;
```



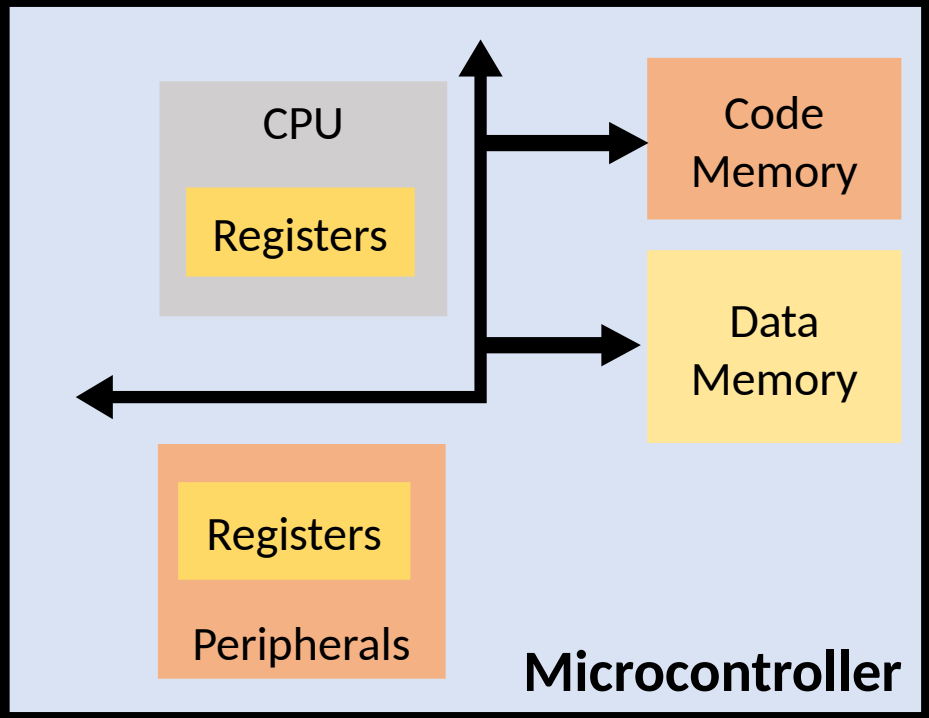
Bitwise Example: & [S5b]

Clear Bits 6 & 7:

```
uint8_t foo = 0xFF;

foo = foo & 0x3F;
```

Results in clearing bits 6 & 7

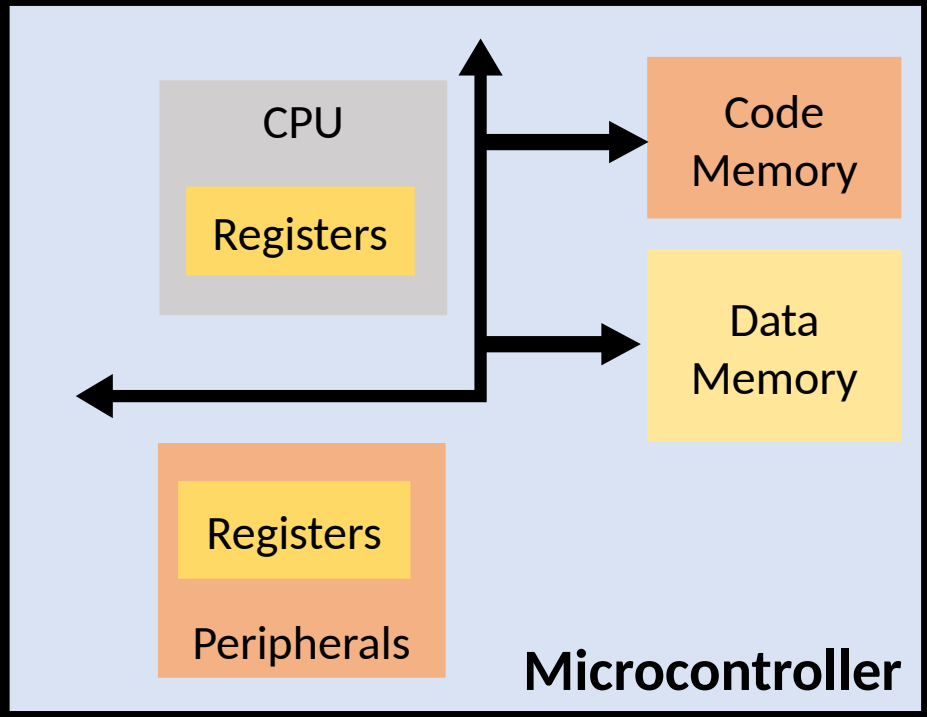


Bitwise Example: & [S5c]

Clear Bits 6 & 7:

```
uint8_t foo = 0xFF;

foo &= 0x3F;
```



Bitwise Example: & [S5d]

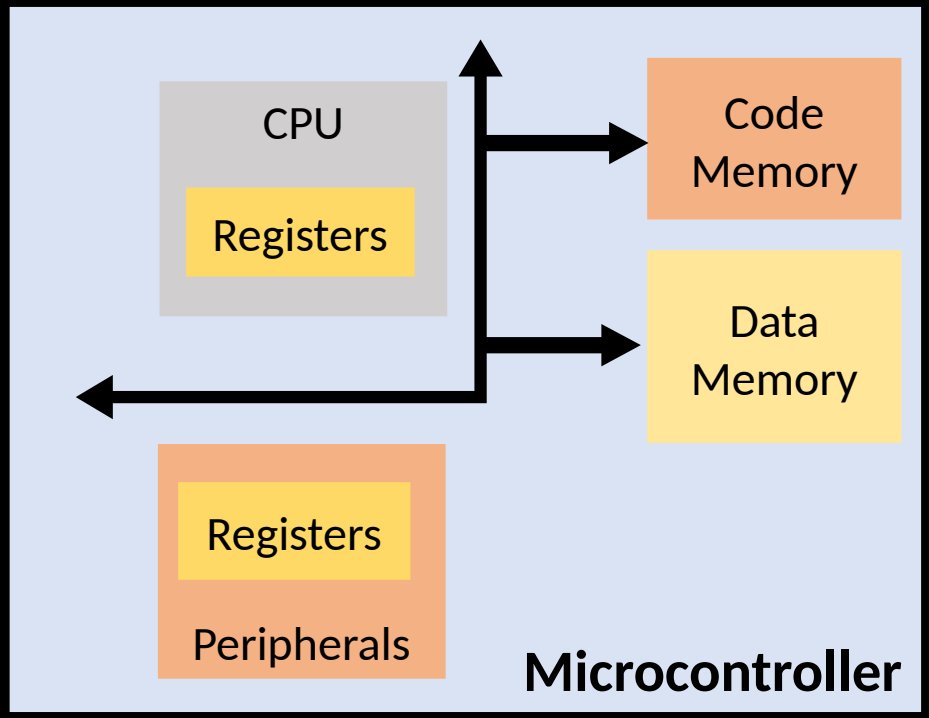
Clear Bits 6 & 7:

```
uint8_t foo = 0xFF;

foo &= ~(0xC0);
```

Specifying bits you wish to clear is more readable

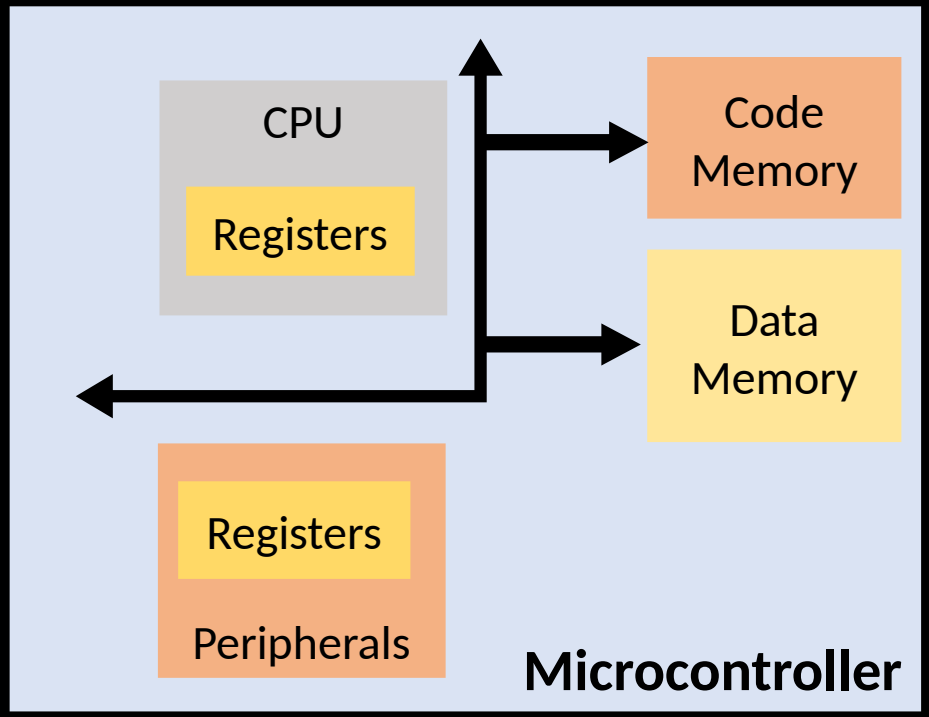
~(0xC0) ▭ 0x3F



Bitwise Example: TOGGLE [S6a]

Toggle Bits 1, 2, & 3:

```
uint8_t foo = 0x0C;
```



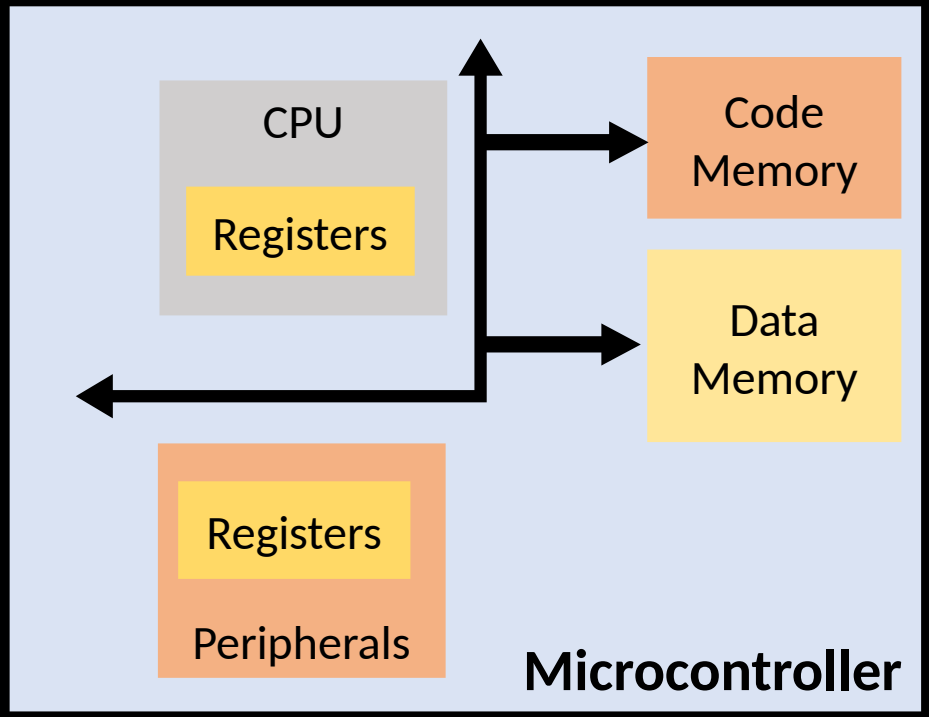
Bitwise Example: TOGGLE [S6b]

Toggle Bits 1, 2, & 3:

```
uint8_t foo = 0x0C;

foo = foo ^ 0x0E;
```

Results in 0x02



Bitwise Example: TOGGLE [S6c]

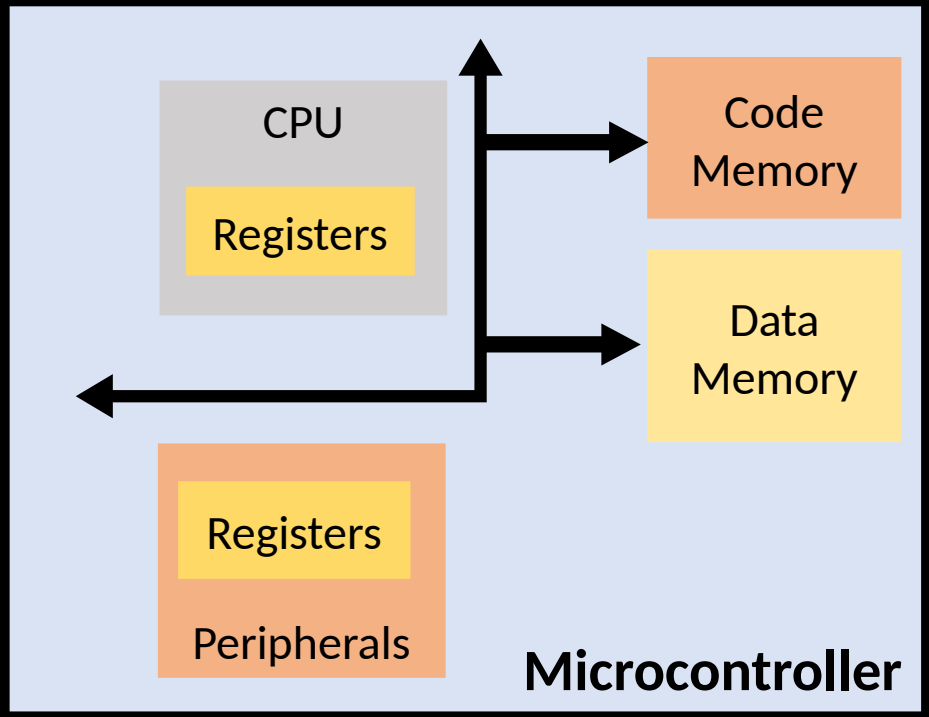
Toggle Bits 1, 2, & 3:

```
uint8_t foo = 0x0C;

foo = foo ^ 0x0E;
```

Results in 0x02

foo	0	0	0	0	1	1	0	0
0x0E	0	0	0	0	1	1	1	0
foo	0	0	0	0	0	0	1	0

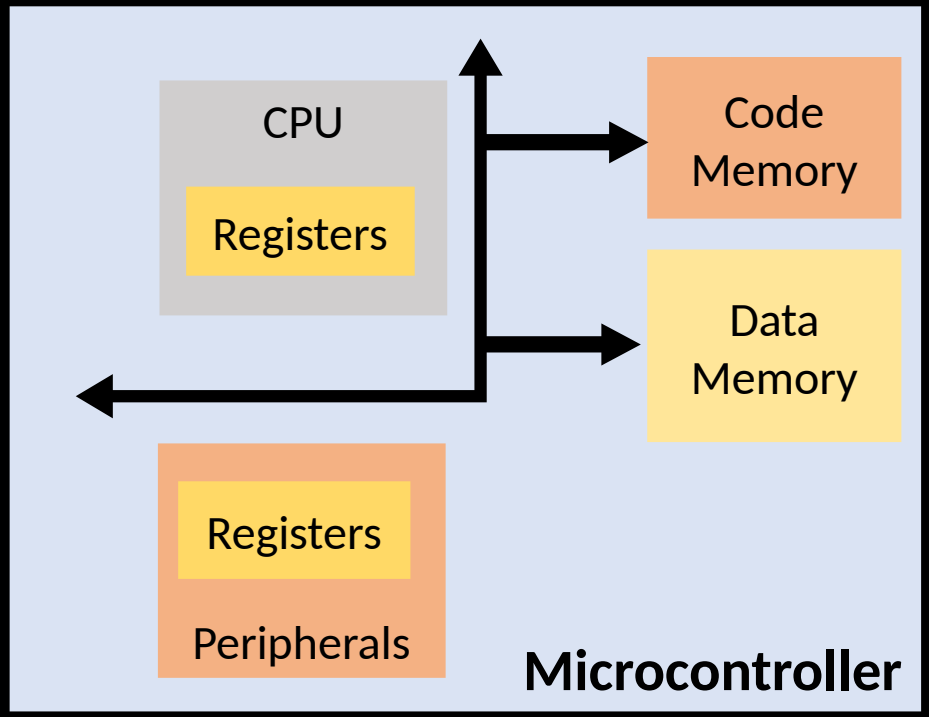


Bitwise Example: TOGGLE [S6d]

Toggle Bits 1, 2, & 3:

```
uint8_t foo = 0x0C;

foo ^= 0x0E;
```



foo	0	0	0	0	1	1	0	0
0x0E	0	0	0	0	1	1	1	0
foo	0	0	0	0	0	0	1	0

Bit Masks [S7a]

- Bit Masks are constant expressions used to set, clear, or toggle a specific set of bits

```
foo |=    0x30 ;
foo &= ~( 0xC0 );
foo ^=    0x0E ;
```

Bit Masks [S7b]

- Bit Masks are constant expressions used to set, clear, or toggle a specific set of bits

```
foo |=     0x30 ;
foo &= ~( 0xC0 );
foo ^=     0x0E ;
```

Use Preprocessors to make code more readable

```
#define MASK1 (0x30)
#define MASK2 (0xC0)
#define MASK3 (0x0E)
```

Bit Masks [S7c]

- Bit Masks are constant expressions used to set, clear, or toggle a specific set of bits

```
foo |=     0x30 ;
foo &= ~(  0xC0 );
foo ^=     0x0E ;
```

Use Preprocessors to make code more readable

```
#define MASK1 (0x30)
#define MASK2 (0xC0)
#define MASK3 (0x0E)

foo |=  MASK1 ;
foo &=  MASK2 ;
foo ^=  MASK3 ;
```

Bit Masks [S8a]

- Bit Masks are constant expressions used to set, clear, or toggle a specific set of bits

```
foo |=    0x30 ;
foo &= ~( 0xC0 );
foo ^=    0x0E ;
```

Example Bit Defines from Texas Instruments msp.h Header Files

msp.h

```
#define BIT0    (uint16_t)(0x0001)
#define BIT1    (uint16_t)(0x0002)
#define BIT2    (uint16_t)(0x0004)
#define BIT3    (uint16_t)(0x0008)
#define BIT4    (uint16_t)(0x0010)
#define BIT5    (uint16_t)(0x0020)
#define BIT6    (uint16_t)(0x0040)
#define BIT7    (uint16_t)(0x0080)
#define BIT8    (uint16_t)(0x0100)
#define BIT9    (uint16_t)(0x0200)
#define BITA    (uint16_t)(0x0400)
#define BITB    (uint16_t)(0x0800)
#define BITC    (uint16_t)(0x1000)
#define BITD    (uint16_t)(0x2000)
#define BITE    (uint16_t)(0x4000)
#define BITF    (uint16_t)(0x8000)
```

Bit Masks [S7d]

- Bit Masks are constant expressions used to set, clear, or toggle a specific set of bits

```
foo |= (BIT4 | BIT5);
foo &= ~( BIT7 | BIT6 );
foo ^= (BIT3 | BIT2 | BIT1);
```

Example Bit Defines from Texas Instruments msp.h Header Files

msp.h

```
#define BIT0    (uint16_t)(0x0001)
#define BIT1    (uint16_t)(0x0002)
#define BIT2    (uint16_t)(0x0004)
#define BIT3    (uint16_t)(0x0008)
#define BIT4    (uint16_t)(0x0010)
#define BIT5    (uint16_t)(0x0020)
#define BIT6    (uint16_t)(0x0040)
#define BIT7    (uint16_t)(0x0080)
#define BIT8    (uint16_t)(0x0100)
#define BIT9    (uint16_t)(0x0200)
#define BITA    (uint16_t)(0x0400)
#define BITB    (uint16_t)(0x0800)
#define BITC    (uint16_t)(0x1000)
#define BITD    (uint16_t)(0x2000)
#define BITE    (uint16_t)(0x4000)
#define BITF    (uint16_t)(0x8000)
```

Peripheral Configuration [S9a]

- Often need to combine set and clear to create desired effect without destroying other bit values

Peripheral Configuration [S9b]

- Often need to combine set and clear to create desired effect without destroying other bit values

- Example
  - Set Bits: 4 & 5 → Set with | (OR)
  - Clear Bits: 6 & 7 → Clear with & / ~ (AND/Complement)
  - Preserve Other Bit Values → Combine logic and assignment

Peripheral Configuration [S10a]

- Often need to combine set and clear to create desired effect without destroying other bit values

- Example
  - Set Bits: 4 & 5
  - Clear Bits: 6 & 7
  - Preserve Other Bit Values

*ptr	?	?	?	?	?	?	?	?
	C	C	S	S				
*ptr (after)	0	0	1	1	?	?	?	?

Peripheral Configuration [S10b]

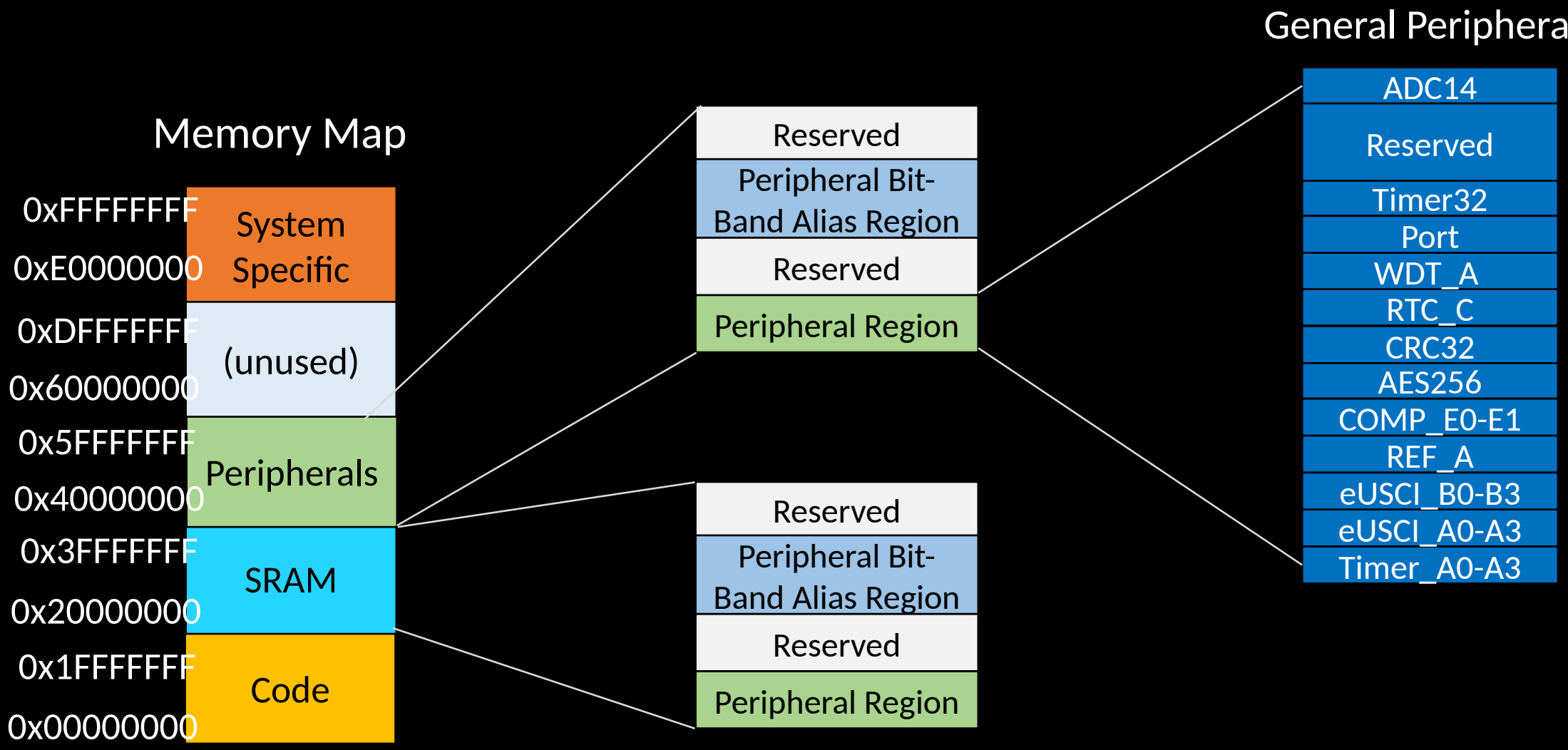
- Often need to combine set and clear to create desired effect without destroying other bit values

- Example
  - Set Bits: 4 & 5
  - Clear Bits: 6 & 7
  - Preserve Other Bit Values

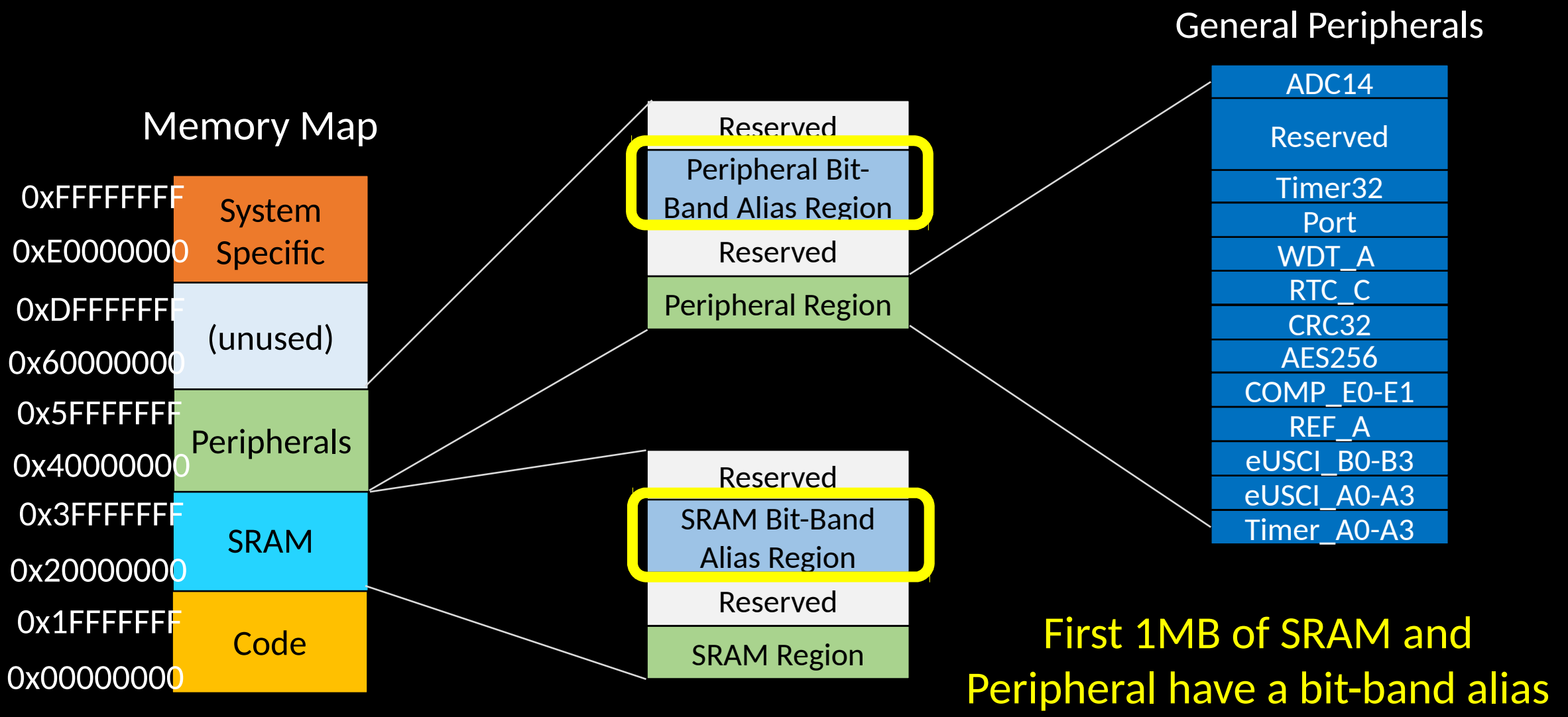
*ptr	?	?	?	?	?	?	?	?
	C	C	S	S				
*ptr (after)	0	0	1	1	?	?	?	?

```
uint8_t * ptr = (uint8_t *)0x40004C02;
*ptr &= ~(BIT6 | BIT7) ;
*ptr |= (BIT4 | BIT5);
```

Bit Banded Memory [S11a]



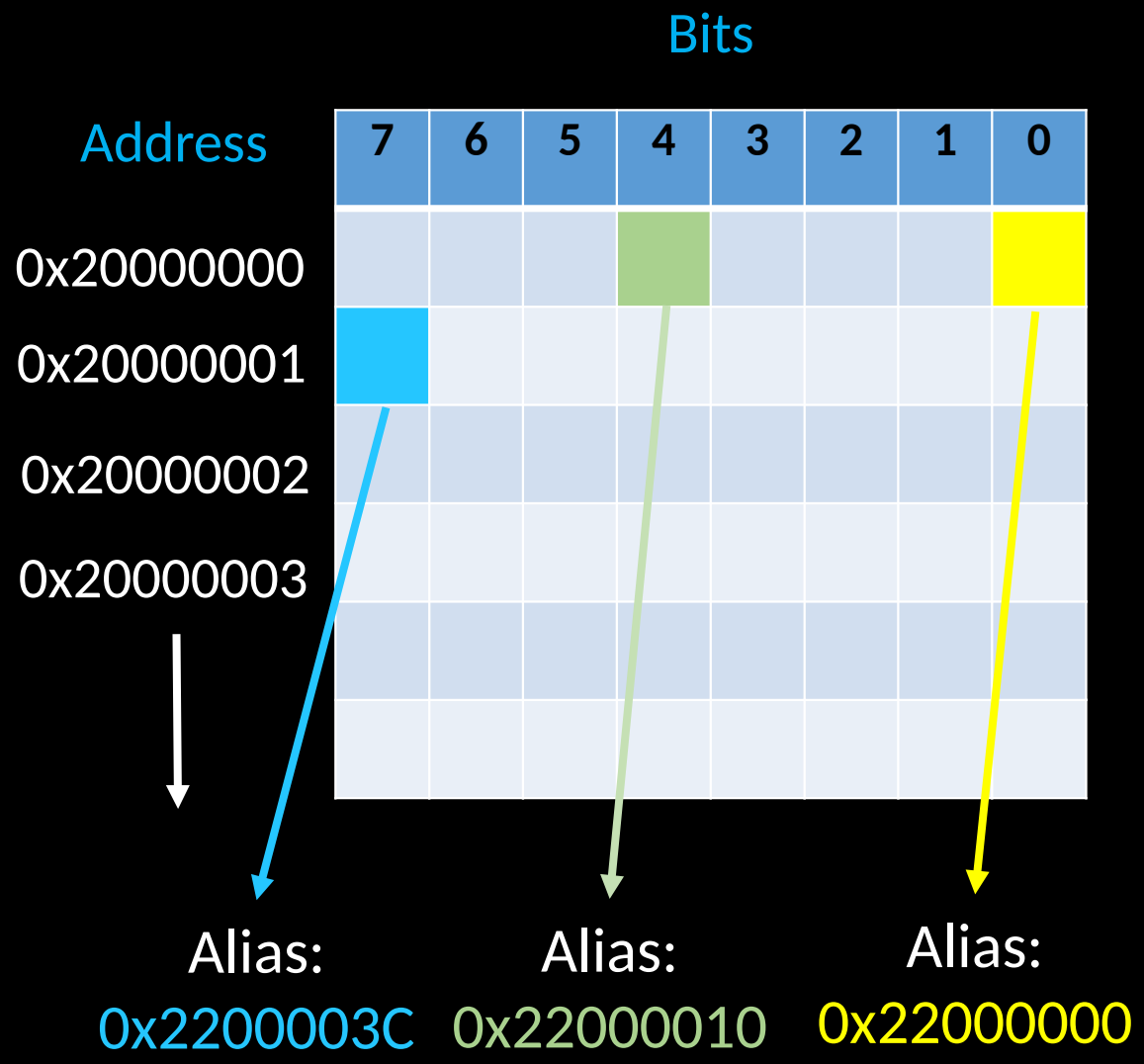
Bit Banded Memory [S11b]



First 1MB of SRAM and Peripheral have a bit-band alias

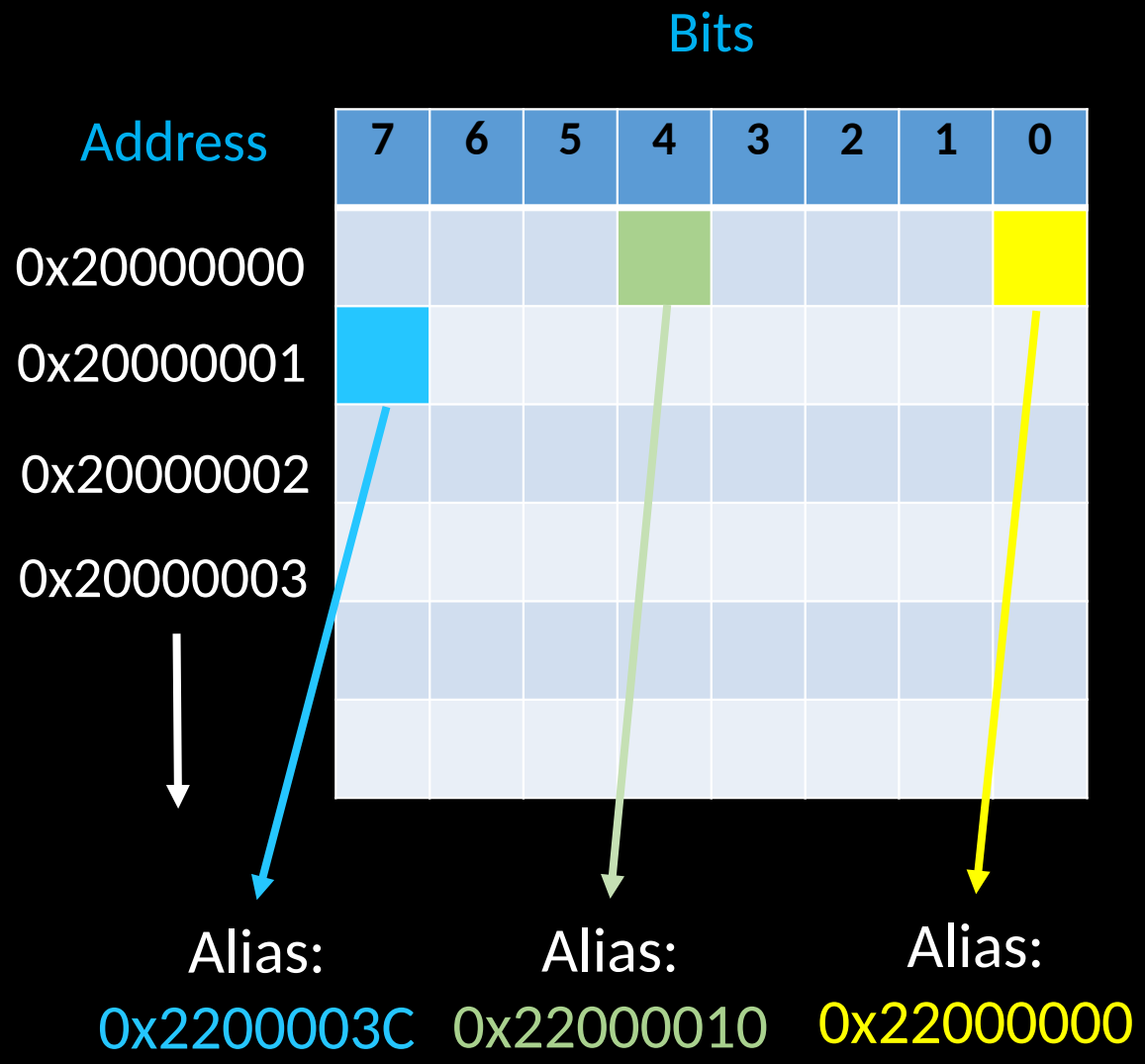
Bit Band Alias [S12a]

- Each bit in the Peripheral & SRAM region is bit addressable
  - Bits are word aligned



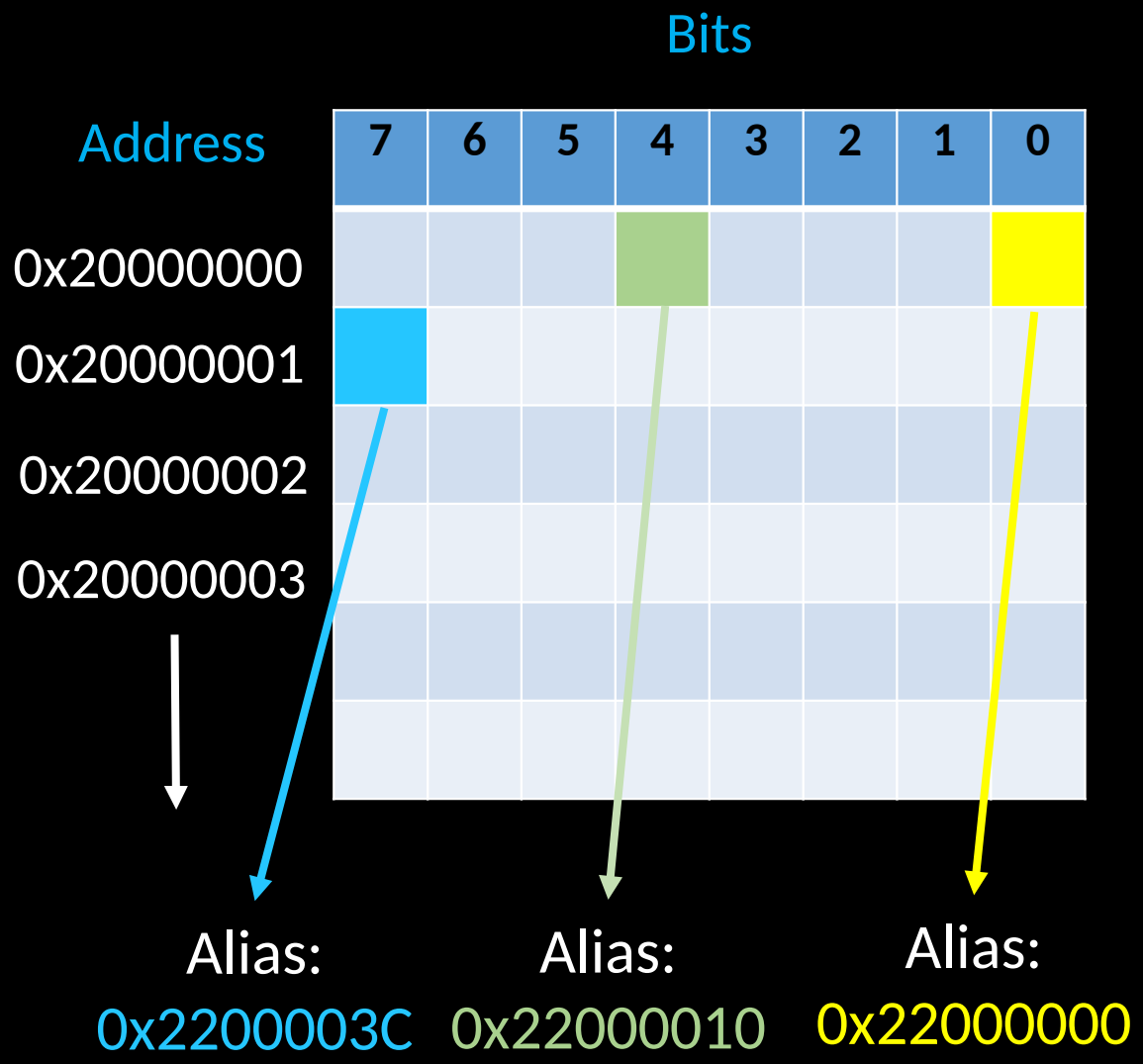
Bit Band Alias [S12b]

- Each bit in the Peripheral & SRAM region is bit addressable
  - Bits are word aligned
- Alias region is offset 0x02000000
  - Peripheral Bit Band: 0x42000000
  - SRAM Bit Band: 0x22000000



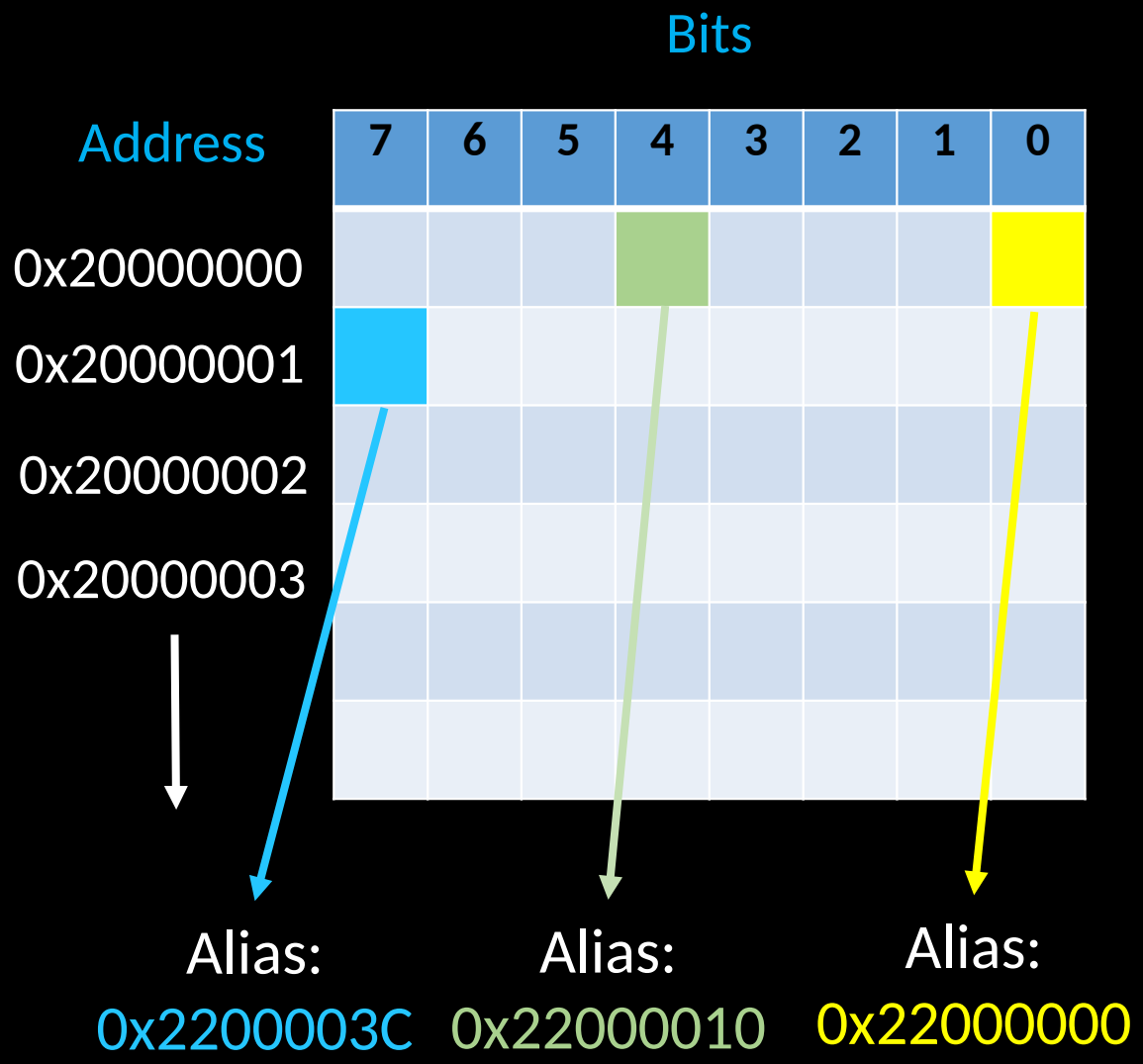
Bit Band Effects [S13a]

- Negatives
  - Reduces the overall available memory for other hardware



Bit Band Effects [S13b]

- Negatives
  - Reduces the overall available memory for other hardware
- Positives
  - Reduces number of instructions needed for read, modify, write



Bit Band Effects [S13c]

- Negatives
  - Reduces the overall available memory for other hardware
- Positives
  - Reduces number of instructions needed for read, modify, write
    - Operation is atomic

Only one instruction is needed and it cannot be interrupted