ENSC 3213

Midterm Exam Review Laboratories

Professor: Dr. Yanmin Gong
TAs: Francisco Fernandes
Khuong Nguyen

Spring 2019

Bitwise Operators

- `A = 0xA2;   B = 0x34;`

AND

A	10100010
B	00110100
A & B	00100000

OR

A	10100010
B	00110100
A | B	10110110

EXCLUSIVE OR

A	10100010
B	00110100
A ^ B	10010110

NOT

A	10100010
~ A	01011101

SHIFT RIGHT

A	10100010
A>>2	00101000

SHIFT LEFT

A	10100010
A<<2	10001000

Masking

- Check a bit:

  ```
  bit = a & (mask)
  ```

- Set a bit:

  ```
  a  |=  (mask)
  ```

- Clear (reset) a bit:

  ```
  a &= ~(mask)
  ```

- Toggle a bit:

  ```
  a ^= mask
  ```

GPIO

```
#define __IO volatile //allows read and write

Typedef struct
{
  __IO uint32_t MODER;   // Mode register
  __IO uint16_t OTYPER;  // Output type register
       uint16_t rev0;    // Padding two bytes
  __IO uint32_t OSPEEDR; // Output speed register
  __IO uint32_t PUPDR;   // Pull-up/pull-down register
  __IO uint16_t IDR;     // Input data register
       uint16_t rev1;    // Padding two bytes
  __IO uint16_t ODR;     // Output data register
       uint16_t rev2;    // Padding two bytes
  __IO uint16_t BSRRL;   // Bit set/reset register (low)
  __IO uint16_t BSRRH;   // Bit set/reset register (high)
  __IO uint32_t LCKR;    // Configuration lock register
  __IO uint32_t AFR[2];  // Alternate function registers
  __IO uint32_t BRR;     // Bit reset register
  __IO uint32_t ASCR;    // Analog switch control register
} GPIO_TypeDef;
#define GPIOB ((GPIO_TypeDef *) 0x48000400)
```

GPIO

```
#define __IO volatile //allows read and write

Typedef struct
{
  __IO uint32_t MOD
  __IO uint16_t OT
       uint16_t r
  __IO uint32_t
  __IO uint32
  __IO uint1
       uint1
  __IO uint16_
       uint16
  __IO uint16
  __IO uint16_
  __IO uint32_t
  __IO uint32_t AFR[2                         gi
  __IO uint32_t BRR;          e
  __IO uint32_t ASCR;     // Analog sw        trol register
} GPIO_TypeDef;
#define GPIOB ((GPIO_TypeDef *) 0x48000400)
```

Remember to study the GPIO Register map!

GPIO

- Enable the clock of GPIO Port A (for joystick), Port B (for Red LED) and Port E (for Green LED)

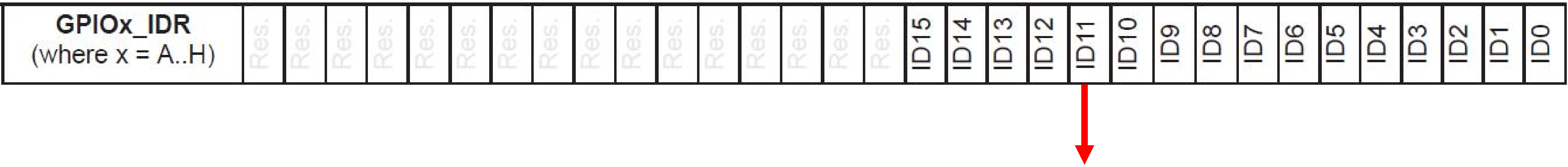
Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AHB2ENR														RNGEN		AESEN			ADCEN	OTGFSEN					GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
Mask	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1
Desired register output																												1			1	1

```
uint32_t MASK = 0x00000013;
RCC->AHB2ENR |= MASK;

Alternative solution (using constants macros in C):
RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOAEN |
                 RCC_AHB2ENR_GPIOBEN | RCC_AHB2ENR_GPIOEEN);
```

Reading inputs

- Suppose we want to verify if only **pin 11** on **GPIO port A** has an input, what would be the if-statement we need to write?



GPIOx_IDR (where x = A..H)	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0

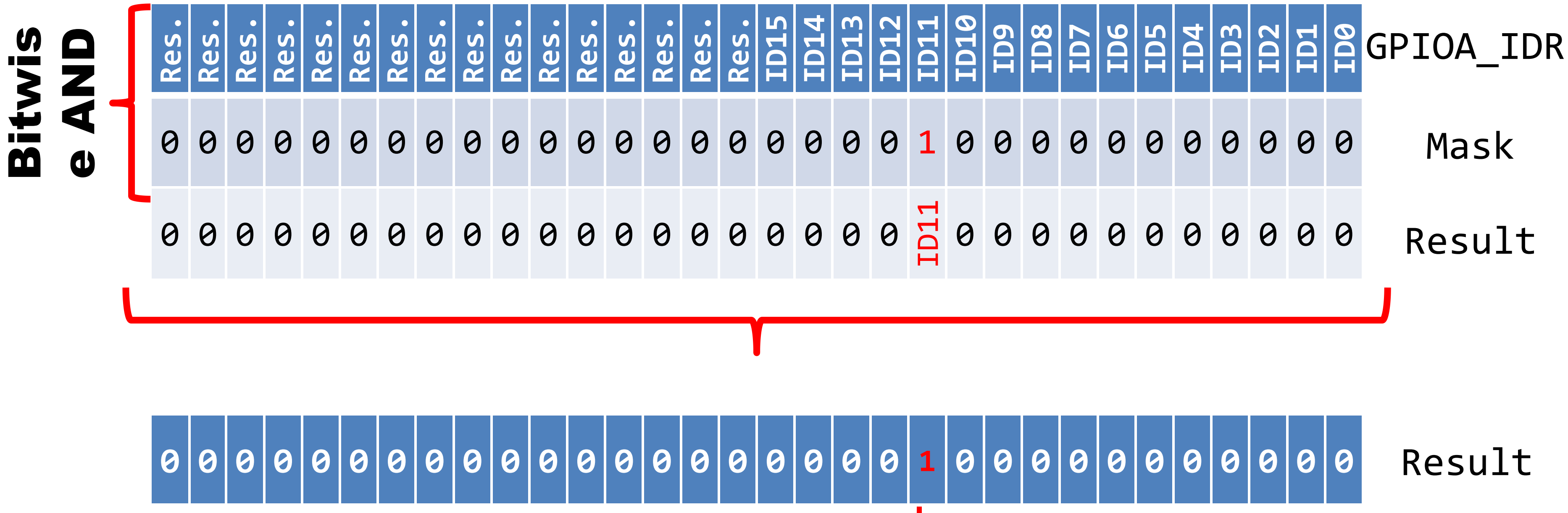
Pin 11

Therefore, if ONLY pin 11 has an input, the GPIOA_IDR register will have the following value:

In binary: 0b0000 0000 0000 0000 0000 1000 0000 0000
In hexadecimal: 0x00000800

Reading inputs

- **Suppose we want to verify if only pin 11 on GPIO port A has an input, what would be the if-statement we need to write?**



Bitwise AND

Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	GPIOA_IDR
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	Mask
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ID11	0	0	0	0	0	0	0	0	0	0	0	Result

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	Result

If we want to verify if `ID11` is equal to `1`.

Reading inputs

- **Suppose we want to verify if only pin 11 on GPIO port A has an input, what would be the if-statement we need to write?**

Bitwise AND

Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	GPIOA_IDR
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	Mask
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ID11	0	0	0	0	0	0	0	0	0	0	0	Result

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Result

If we want to verify if ID11 is equal to 0.

Reading inputs

- **Suppose we want to verify if only pin 11 on GPIO port A has an input, what would be the if-statement we need to write?**

Bitwise AND

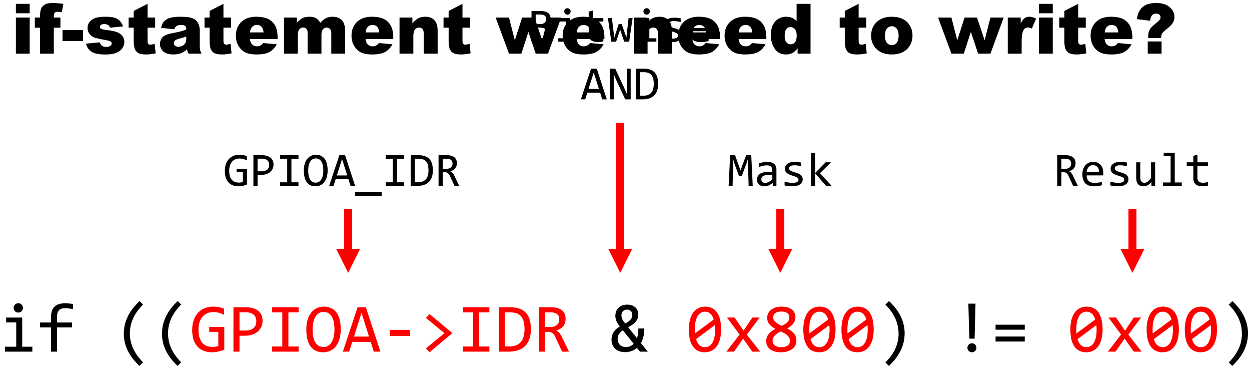
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	GPIOA_IDR
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	Mask
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ID11	0	0	0	0	0	0	0	0	0	0	0	Result

To sum up:

- **To verify any bit of a register, you just need to put a 1 in the bit you want in your mask.**
- **The** `result` **will depend if you want to verify if the bit is 0 or 1.**

Reading inputs

- **Suppose we want to verify if only pin 11 on GPIO port A has an input, what would be the if-statement we need to write?**



Bitwise AND

GPIOA_IDR

Mask

Result

```
if ((GPIOA->IDR & 0x800) != 0x00)
```

OR

```
if ((GPIOA->IDR & 0x800) == 0x800)
```