Arduino

Tutorial 4
Micro-controller Register Programming

2021 TeakSoon Ding
for STEMKRAF
https://github.com/teaksoon/stemkraf

Register Memory Programming with C-Language

Numbering Systems (Decimal, Binary, Hexadecimal, Octal)

Binary and Hexadecimal Conversion

Memory BITS and BYTES

REGISTER Memory

REGISTER Memory – Replace and Retrieve entire REGISTER

C-Language BIT OPERATOR

REGISTER Memory – Replace and Retrieve individual BIT

REGISTER – Write single BIT to REGISTER to change Voltage on a Pin

REGISTER – Read single BIT from REGISTER for the Voltage state of a Pin

Decimal Numbering System (10 Symbols)

0 1 2 3 4 5 6 7 8 9

This numbering system is used in our daily activities.

Binary Numbering System (2 Symbols)

0 1

This numbering system is used in micro-controllers Memory

Hexadecimal Numbering System (16 Symbols)

0 1 2 3 4 5 6 7 8 9 A B C D E F

This numbering system is used to simplify big binary numbers, to reduce the number of digits.

Octal Numbering System (8 Symbols) – rarely used

0 1 2 3 4 5 6 7 8

This numbering system were used in older Computer Systems to simplify big binary numbers where multiplies of 3-bits were commonly used.

The following represents the same number but in different numbering system. Hex, Decimal, Octal and Binary

```
FF        ( 2 digits in Hex )
255       ( 3 digits in Decimal )
377       ( 3 digits in Octal )
11111111  ( 8 digits in Binary )
```

By looking a the number above we can guess the number FF is from the Hex numbering system. However, for number 255, 377 and 11111111 we cannot tell which numbering system they are using. In C-Language programming, we use a Prefix to indicate the numbering system used by a number

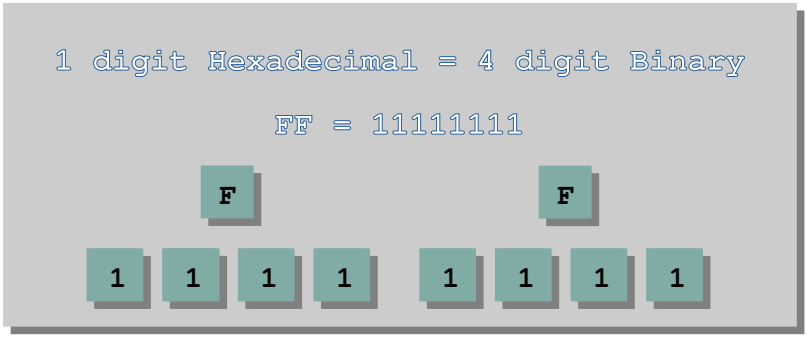
```
0xFF          "0x" as Prefix for Hex
255           No Prefix for Decimal
0o377         "0o" as Prefix for Octal ( 0 is Zero, o is letter 'o')
0b11111111    "0b" as Prefix for Binary
```

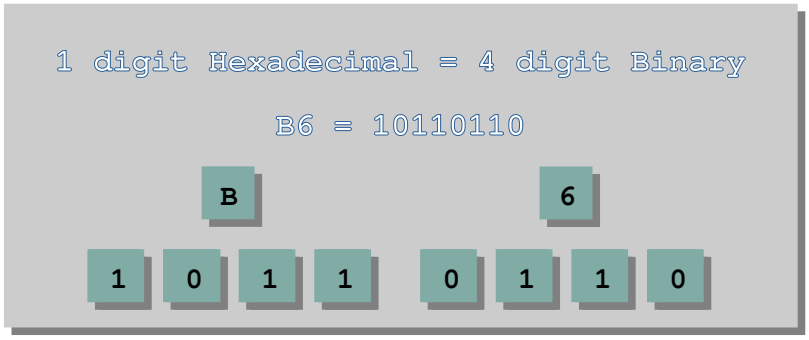
Regardless of what numbering system we use, the number will be stored as binary in the micro-controllers memory.

```
0xFF        will be stored as 11111111 in the micro-controllers Memory
255         will be stored as 11111111 in the micro-controllers Memory
0o377       will be stored as 11111111 in the micro-controllers Memory
0b11111111  will be stored as 11111111 in the micro-controllers Memory
```

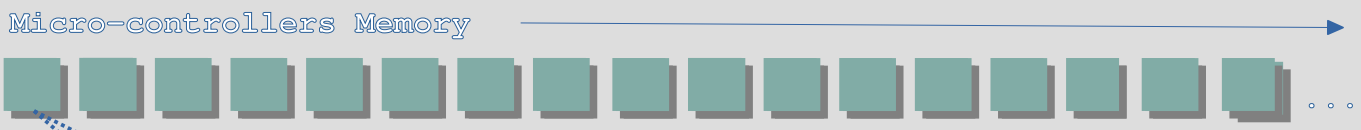
When we are doing programming with the REGISTER Memory, we will see alot of Hexadecimal numbers in our Program Codes to make simplify our codes.

The Hexadecimal will automatically be converted into binary when saved into the REGISTER Memory.



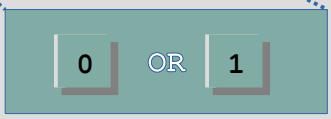


Micro-controllers Memory is consist of many individual physical memory device arranged in a logical sequence.



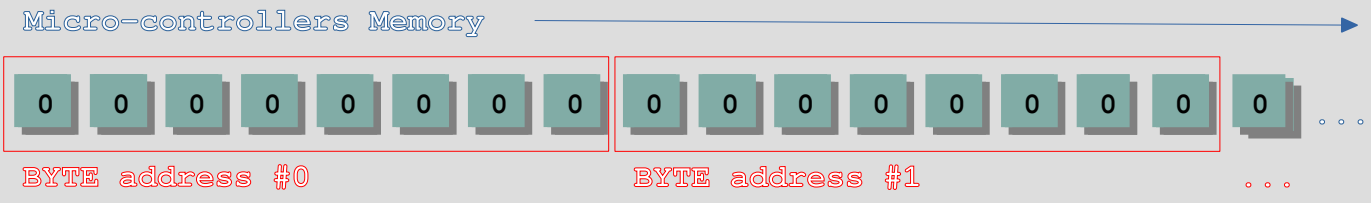
Each physical memory device can have TWO(2) different states where only one state can be active at a time. The current active state in each memory device is represented by either 0 or 1.

Each memory device must be represented by either 0 or 1. This single memory device is known as a **BIT**



The C-Language does not allow us to access to a single BIT from the micro-controller Memory. We can only access/address micro-controllers Working Memory in the smallest **groups of 8-BIT**, also known as a **BYTE**.

ONE(1) BYTE = EIGHT(8) BIT

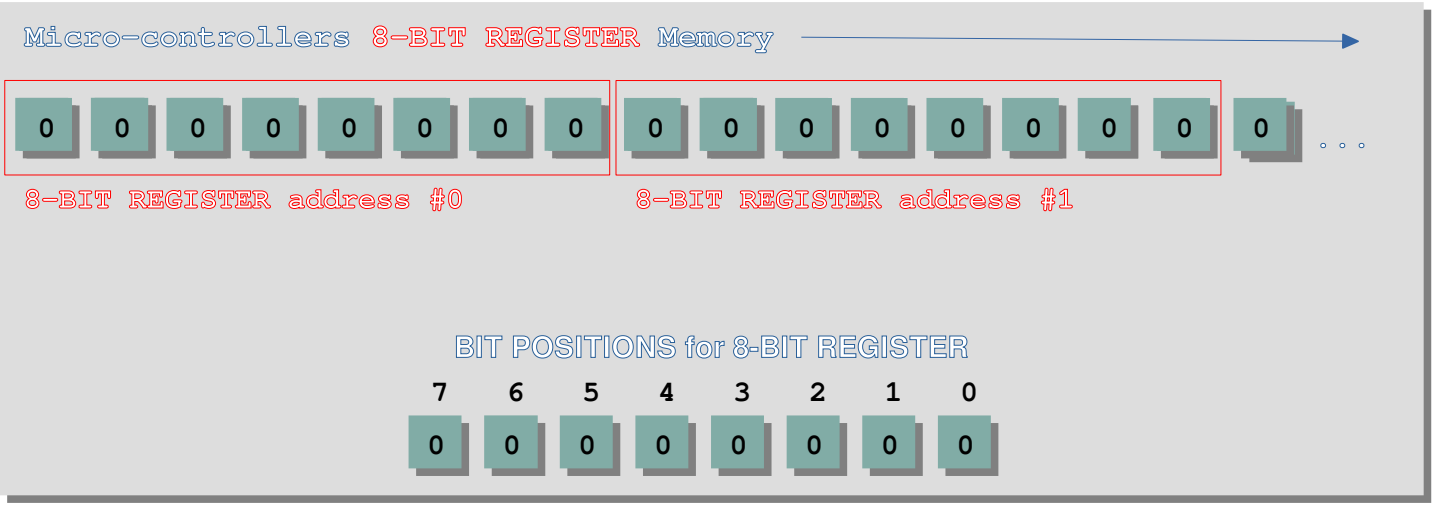


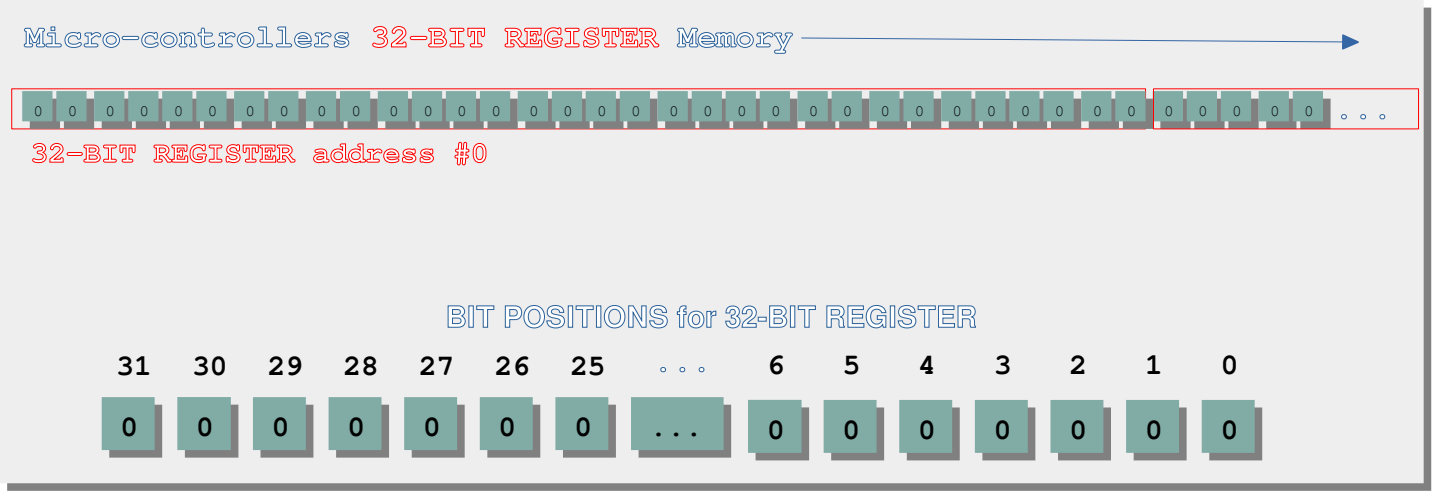
REGISTER is a special type of Memory inside the micro-controller where its usage is already predetermined. There are many REGISTER in the micro-controller Register Memory block, each REGISTER for a specific purpose.

1. The CPU will perform task based on contents of a specific REGISTER
2. The CPU may also change the contents of a specific REGISTER

8-BIT REGISTER = Each REGISTER is a group of 8 BIT from the Memory
32-BIT REGISTER = Each REGISTER is a group of 32 BIT from the Memory

n-BIT REGISTER = Each REGISTER is a group of n BIT from Memory

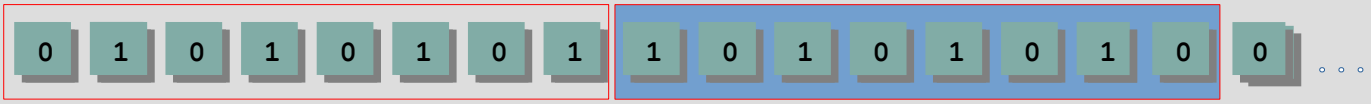




Each REGISTER address are often given a name for easy programming, for example, A REGISTER is named as "PINB", "PORTB", 'GPIOA" and etc...

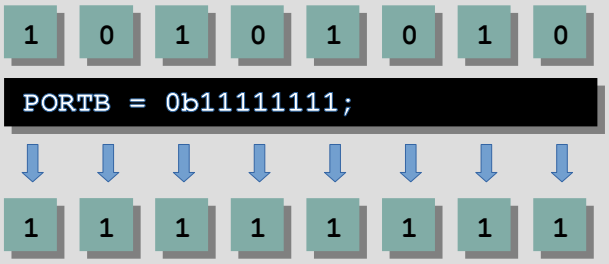
REPLACE ENTIRE 8-BITS WITHOUT RETAINING OLD BIT VALUES

Micro-controllers 8-BIT REGISTER Memory →



8-BIT REGISTER name = "PINB" 8-BIT REGISTER name = "PORTB"

REPLACE all 8 BITS from 8-BIT REGISTER named "PORTB" with 1 1 1 1 1 1 1 1



Instead of coding, `PORTB = 0b11111111;`
wwe can also code in shorter format Hex format, `PORTB = 0xFF;`

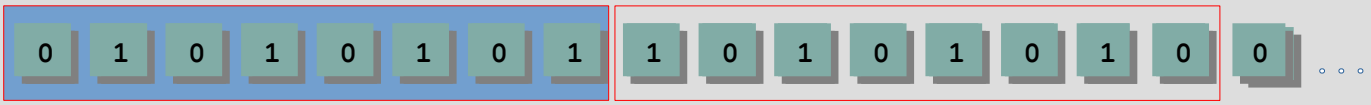
The following in Decimal and Octal will also do the same thing,

```
PORTB = 255;
PORTB = 0o377;
```

0xFF, 255, 0o377 will all be automatically converted the equivalent value of the binary by the C-Language, which is 11111111

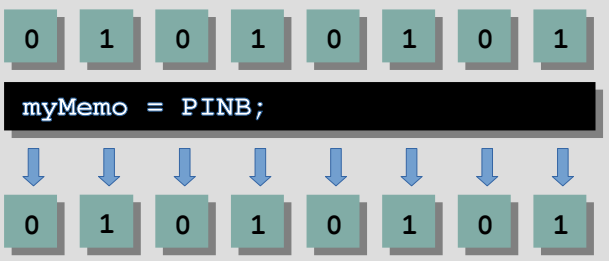
RETRIEVE THE ENTIRE 8-BITS FROM REGISTER

Micro-controllers 8-BIT REGISTER Memory →



8-BIT REGISTER name="PINB" 8-BIT REGISTER name="PORTB"

RETRIEVE from 8-BIT REGISTER named "PINB" and store it to Working Memory named "myMemo"



In order to Replace or Retrieve an individual BITS in micro-controllers Memory, we will need use some of these C-Language BIT Operators.

```
C-Language BIT Operators
&   AND ( Bit Operation between 2 individual bit )
|   OR  ( Bit Operation between 2 individual bit )
^   XOR ( Bit Operation between 2 individual bit )
~   NOT ( Bit Operation on individual bit )

<< LEFT SHIFT  ( Shift bits to the LEFT in Memory )
>> RIGHT SHIFT ( Shift bits to the RIGHT in Memory )
```

```
& Single BIT AND Operation
1 & 1 result = 1
1 & 0 result = 0
0 & 1 result = 0
0 & 0 result = 0

& Multiple BIT AND Operation
          01110111 &
          11000010
Result = 01000010
```

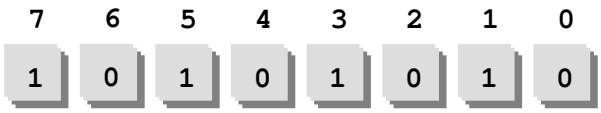
```
| Single BIT OR Operation
1 | 1 result = 1
1 | 0 result = 1
0 | 1 result = 1
0 | 0 result = 0

| Multiple BIT OR Operation
          01110111 |
          11000010
Result = 11110111
```

```
^ Single BIT XOR Operation
1 ^ 1 result = 0
1 ^ 0 result = 1
0 ^ 1 result = 1
0 ^ 0 result = 0

^ Multiple BIT XOR Operation
          01110111 ^
          11000010
Result = 10110101
```

```
~ Single BIT NOT Operation
~1 result = 0
~0 result = 1

~ Multiple BIT NOT Operation
         ~01110111
Result = 10001000
```

```
<< Single BIT LEFT SHIFT Operation
00000001 << 0 result = 00000001
00000001 << 1 result = 00000010
00000001 << 2 result = 00000100
00000001 << 3 result = 00001000
00000001 << 4 result = 00010000
00000001 << 5 result = 00100000
00000001 << 6 result = 01000000
00000001 << 7 result = 10000000

<< Multiple BIT LEFT SHIFT Operation
00000011 << 5 resulted 01100000
```

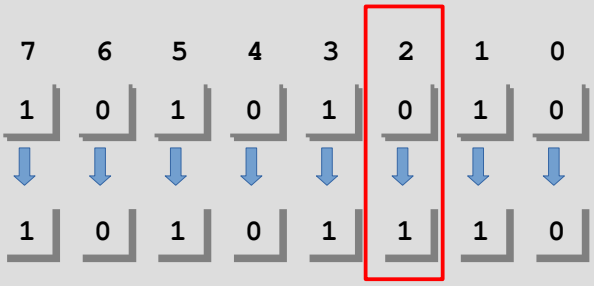
```
>> Single BIT RIGHT SHIFT Operation
10000000 >> 0 result = 10000000
10000000 >> 1 result = 01000000
10000000 >> 2 result = 00100000
10000000 >> 3 result = 00010000
10000000 >> 4 result = 00001000
10000000 >> 5 result = 00000100
10000000 >> 6 result = 00000010
10000000 >> 7 result = 00000001

>> Multiple BIT RIGHT SHIFT Operation
11000000 >> 5 result = 00000110
```



Assuming we have a 8-BIT REGISTER Memory named "PORTB" with 10101010 in its 8-BIT Memory

OPERATION:
CHANGE a single BIT in REGISTER to 1, while RETAINING all the other BITS

Also known as "SET BIT"




```
PORTB = PORTB | ( 1 << 2 );
```

(1 << 2) gives us 00000100

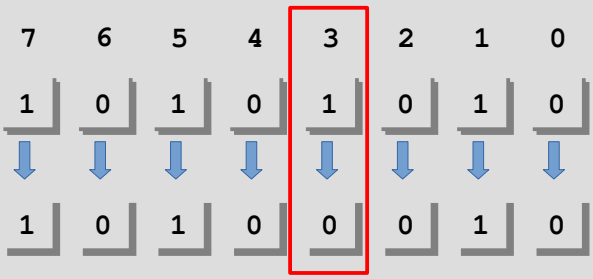
PORTB = PORTB | 00000100

Original BITS in PORTB = 10101010

PORTB = 10101010 |
00000100
PORTB = 10101110
Only BIT #2 from PORTB is changed to 1

OPERATION:
CHANGE a single BIT in REGISTER to 0, while RETAINING all the other BITS

Also known as "CLEAR BIT"




```
PORTB = PORTB & ~( 1 << 3 )
```

(1 << 3) gives us 00001000
~00001000 gives us 11110111

PORTB = PORTB & 11110111

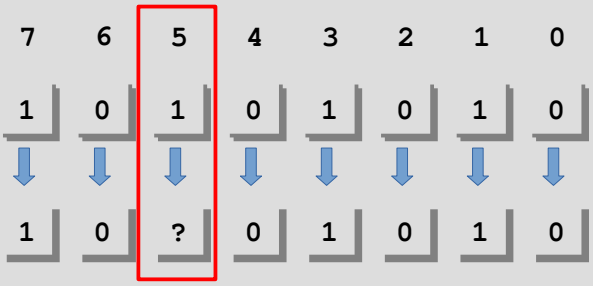
Original BITS in PORTB = 10101010

PORTB = 10101010 |
11110111
PORTB = 10100010

Only BIT #3 from PORT is changed to 0

OPERATION:
CHANGE a single BIT in REGISTER to 0 if previously 1 or to 1 if prevously 0, while RETAINING all the other BITS

Also known as "TOGGLE BIT"




```
PORTB = PORTB ^ ( 1 << 5 )
```

(1 << 5) gives us 00100000

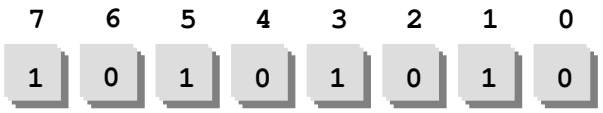
PORTB = PORTB ^ 00100000

Original BITS in PORTB = 10101010

PORTB = 10101010 ^
00100000

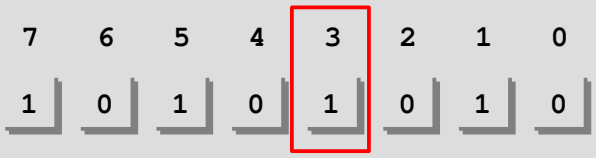
PORTB = 10001010

Only BIT #5 from PORTB is changed to 0
(IF that bit is previously 0 in PORTB, it will become 1 instead)



Assuming we have a 8-BIT REGISTER Memory named "PINB" with 10101010 in its 8-BIT Memory

OPERATION:
RETRIEVE a single BIT from REGISTER (case 1)




```
myBit = (PINB & ( 1 << 3 )) > 0 ? 1:0;
```

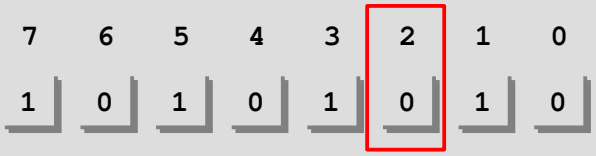
(1 << 3) gives us 00001000

Original BITS in PORTB = 10101010

```
  10101010 &
  00001000
= 00001000
```

00001000 > 0 ? 1:0; is TRUE, myBit is 1

OPERATION:
RETRIEVE a single BIT from REGISTER (case 2)




```
myBit = (PINB & ( 1 << 2 )) > 0 ? 1:0;
```

(1 << 2) gives us 00000100

Original BITS in PORTB = 10101010

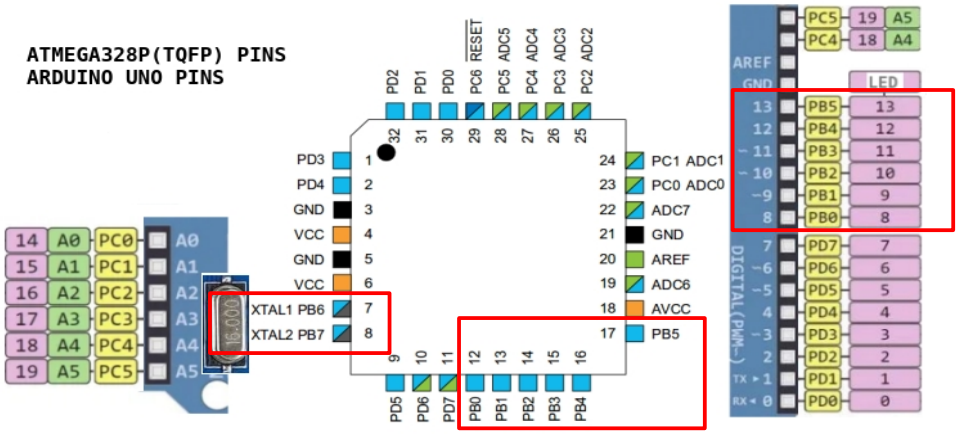
```
  10101010 &
  00000100
= 00000000
```

00000000 > 0 ? 1:0; is FALSE, myBit is 0

ATmega48A/48PA/88A/88PA/168A/168PA/328/328P

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–
0x0 (0x20)	Reserved	–	–	–	–	–	–	–	–

From the Datasheet snapshot, we can see that there are many REGISTER in the Atmega328 micro-controller. Each REGISTER have an Address and a Name. Each of the REGISTER is 8-bits (Bit-0 to Bit-7)where each bits are also labelled.



Lets look at these 3 REGISTER, “DDRB”, “PORTB” and “PINB”

These 3 REGISTER are controls the behaviour of the Pin labeled PB0 to PB7 on the AtMega328. Only PB0 to PB5 of the Atmega328 can be used because the PB6 and PB7 is used by the external clock crystal.

DDRB - to set Pin PB0 to PB5 to be input or output pin
PORTB - to set Pin PB0 to PB5 to either have 5V(bit=1) or 0V(bit=0)
PINB - to read whether Pin PB0 to PB5 is 5V(bit=1) or 0V(bit=0)

To know what is DDRB, PORTB and PINB REGISTER (or any other REGISTER) is used for and how to use them, we need to to refer to the Atmega328 micro-controller DataSheet. This is not just for Atmega328, it is for many other microchips that have REGISTERS.

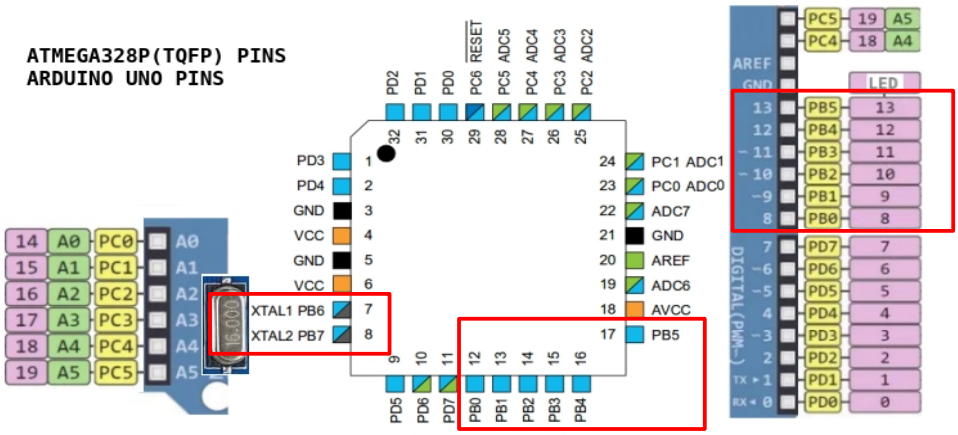
```
// program name = register_digital_write
#define F_CPU 16000000UL  // CPU Speed
#include <avr/io.h>
#include <util/delay.h>

int main() {
  DDRB = DDRB | 0b00100000; // bit-5 DDRB = 1, Output pin. Arduino Uno Pin 13
  while(1) {
    PORTB = PORTB |  0b00100000; // bit-5 of PORTB = 1, 5V at PB5
    _delay_ms(300);
    PORTB = PORTB & ~0b00100000; // bit-5 of PORTB = 0, 0V at PB5
    _delay_ms(300);
  }
  return 0;
}
```

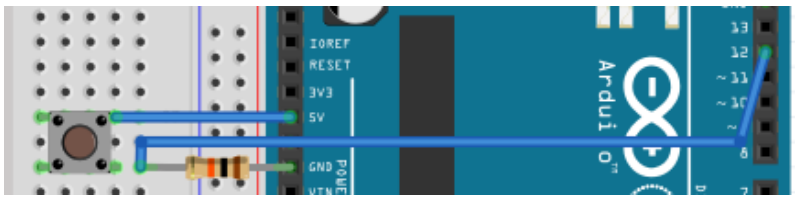
PB5 of the Atmega328 is also connected to the “test LED” on the Arduino Uno, when the PB5 is set to 5V or 0V, the LED will respond accordingly.

ATmega48A/48PA/88A/88PA/168A/168PA/328/328P

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–
0x0 (0x20)	Reserved	–	–	–	–	–	–	–	–

From the Datasheet snapshot, we can see that there are many REGISTER in the Atmega328 micro-controller. Each REGISTER have an Address and a Name. Each of the REGISTER is 8-bits (Bit-0 to Bit-7)where each bits are also labelled.






```
// program name : register_digital_read
#include <avr/io.h>
char buttonDown = 'N';
int main() {

  DDRB = DDRB & ~0b00010000; // Bit-4 = 0, Input pin. Arduino Uno Pin 12
  DDRB = DDRB |  0b00100000; // Bit-5 = 1, Output pin. Arduino Uno Pin 13
  while(1) {
    if ( (PINB & 0b00010000) != 0) {  // Bit-4 = "1"
      if ( buttonDown == 'N') {
        buttonDown = 'Y';
        PORTB = PORTB | 0b00100000;  // Bit-5 = "1"
      }
    } else {
      if ( buttonDown == 'Y') {
        buttonDown = 'N';
        PORTB = PORTB & ~0b00100000; // Bit-5 = "0"
      }
    }
  }
  return 0;
}
```