

Workshop: “Introdução a Microcontroladores”



1. Microprocessador	pag. 2
2. Atmega328p	pag. 2
3. Portas de entrada e saída	pag. 4
4. Ativação dos registos	pag. 5
5. Botões	pag. 6
6. Início de um projeto	pag. 7
7. Timers e interrupções	pag. 9
8. Exemplo	pag. 12
9. Anexos	pag. 14



1-Microprocessador:

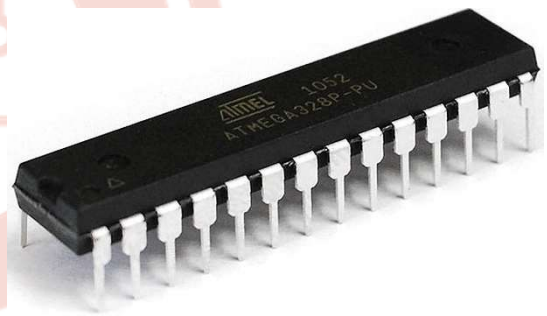
É um circuito sequencial capaz de executar sequências de instruções. É constituído por:

- Unidade aritmética e lógica (faz cálculos)
- Registos (guardam operandos e resultados)
- Sequenciador (encadeia as instruções).

Se juntarmos isto com memória e interfaces de entradas e saídas obtemos um microcontrolador.

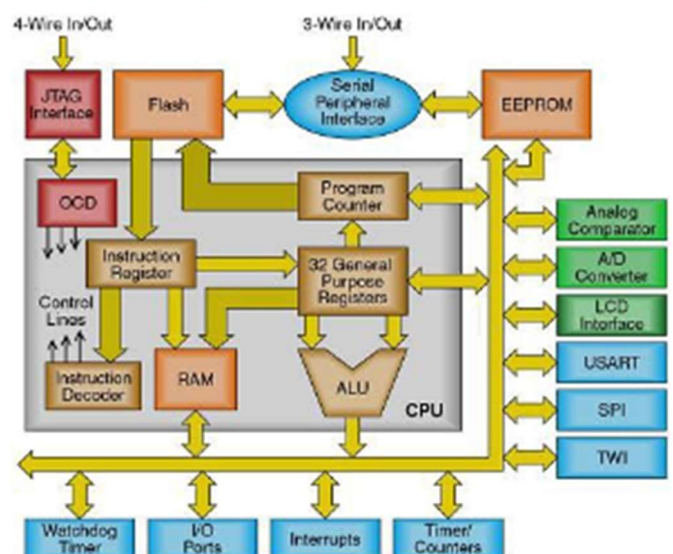
2-Atmega328p:

O Atmega328p faz parte da família AVR de microcontroladores que são desenvolvidos pela empresa Atmel. Esta linha de micros foram dos primeiros a incluir memória flash nos seus chips, para armazenar o programa. São muito frequentes em sistemas embarcados educacionais e recreativos como os Arduinos onde se tornaram bastante populares.



CPU:

- 32 registos
- Arquitetura RISC:
 - 131 instruções
- Memórias
 - Memória do programa
 - Memória para dados



Periféricos:

- Pinos E/S
- Timers/WTD
- ADC 10bits
- Comunicação I2C, SPI, USART

No caso do **Atmega328p** temos memórias de:

- 32kB Flash ROM (usada para o programa)
- 2kB SRAM (Dados/variáveis)
- 1kB EEPROM (Dados/variáveis não voláteis)

3- Portas de entradas e saídas:

Há 3 “portas” multidirecionais que podem ser consideradas como entradas ou saídas (PB, PC, PD). Isto permite ter 23 pinos ao total, com resistências de pull-up, e corrente máximas de 40mA por pino e 200mA ao total.

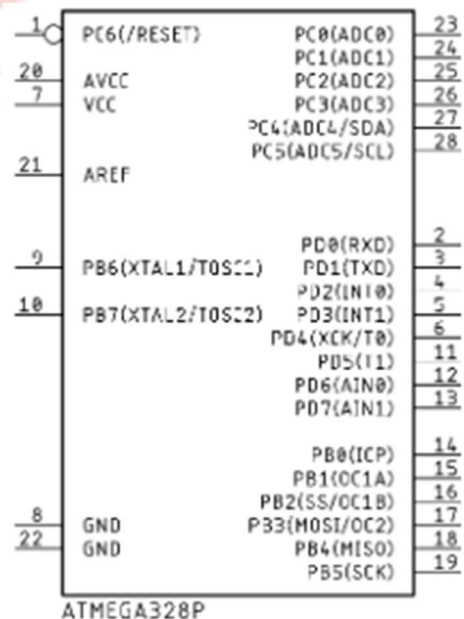
Cada uma destas portas tem 3 registos:

- DDRX**: Configurar o registo (1:Out, 0:In)
- PORTX**: registo de saída
- PINX**: registo de entrada

Isto dá 9 registos ao total:

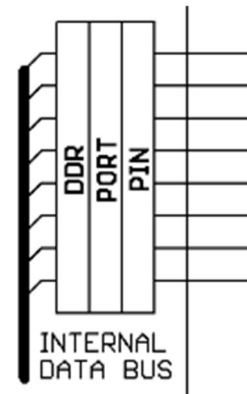
- DDRB, PORTB e PINB.
- DDRC, PORTC e PINC.
- DDRD, PORTD e PIND.

DDRX- Configurar pinos como E/S.



PORTX: Se tiver declarado como saída: 1 liga a saída, 0 desliga a saída. Se tiver declarado como entrada: 1 ativa a resistência de pull-up do pino.

PINX: Ler o sinal na porta configurada como entrada.



4- Ativação dos registros:

Todos os registros DDRX começam com todos os bits a 0, ou seja, estão todos declarados como entradas. Para definirmos um pino como saída temos de colocar o bit do registo correspondente a esse pino a 1.

Exemplo: ligar o bit 4.

b7	b6	b5	b4	b3	b2	b1	b0
OR							
0	0	0	1	0	0	0	0
=							
b7	b6	b5	1	b3	b2	b1	b0

Traduzindo para código:

```
DDRB | 0b00010000;
```

```
⇔ DDRB = DDRB | (1<<4);
```

```
⇔ DDRB |= (1<<4);
```

É possível colocar vários bits a 1 de uma só vez usando uma máscara binária com mais do que um bit a 1 (ex: 0b00010110).

O operador `|` é um OR bit a bit, que é diferente do operador `||` que faz uma comparação em que na saída temos ou zero ou um.

O operador `<<` faz um shift de bits. Neste caso a operação `(1<<4)` desloca o 1 quatro posições para a esquerda resultando em 00010000.

É utilizado um OR binário pois desta maneira independentemente dos valores que tenham os bits dos registos, podemos mudar apenas o bit que desejamos mudar.

Exemplo: por o bit 3 a 0.

b7	b6	b5	b4	b3	b2	b1	b0
AND							
1	1	1	1	0	1	1	1
=							
b7	b6	b5	b4	0	b2	b1	b0

Código:

```
DDRB & 0b11110111;
```

```
⇔ DDRB &= ~(1<<3);
```

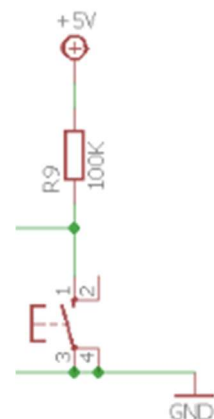
Ao realizar um AND (&) bit a bit, apenas o valor onde a máscara é zero é que é alterado do seu valor original. Todos os outros permanecem os seus valores de origem.

O operador “~” faz a negação da máscara. Uma vez que fazer shift de um zero não é possível.

5- Botões:

O registo PINX é usado para obter o estado que está presente num pino quando este esta declarado como entrada.

d7	d6	d5	d4	d3	d2	d1	d0
AND							
0	0	0	0	0	1	0	0
=							
0	0	0	0	0	d2	0	0



PIND & (1<<2) Permite ler o valor que está presente no pino.

Neste caso quando o botão está solto, na entrada vão estar presentes 5 volts, e quando este está premido a tensão passa a 0 volts.

Para as configurações de botões utilizadas podemos usar o seguinte exemplo de código:

```
If(!(PIND&(1<<2)){
```

```
.....
```

```
}
```

Resumo:

//Escolher o pino como saída ou entrada

DDRD |= (1<<5); //Pino D5(13) -> entrada

DDRD &= (1<<5); //Pino D5(13) -> saída

//Quando saída:

PORTD |= (1<<4); //Pino D4(12) -> ligadar

PORTD &= ~(1<<4); //Pino D4(12) -> desligar

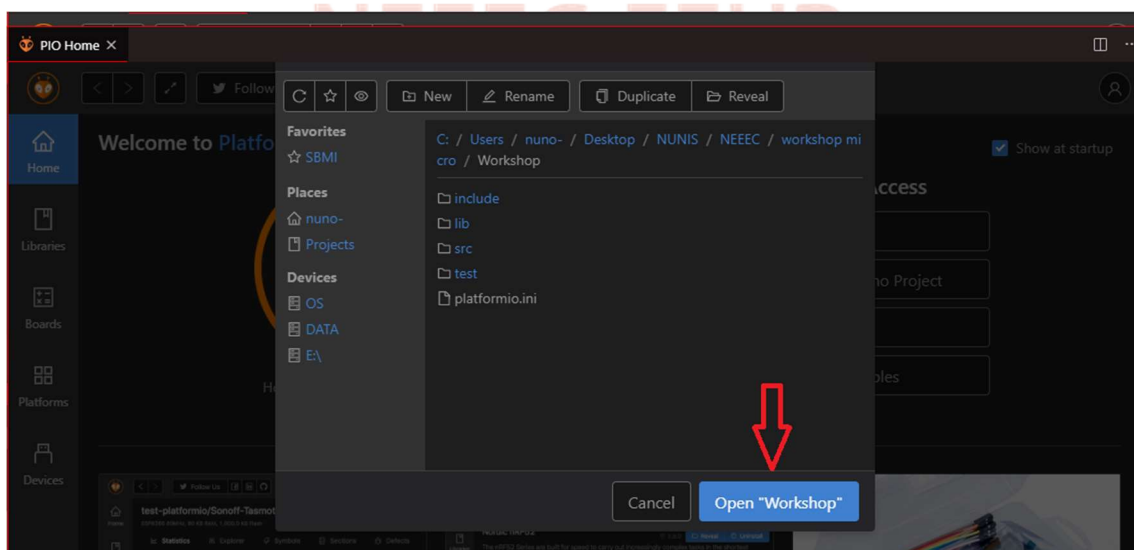
//Quando entrada:

PORTD |= (1<<3); //Ativa o pull-up interno do pino D3(11)

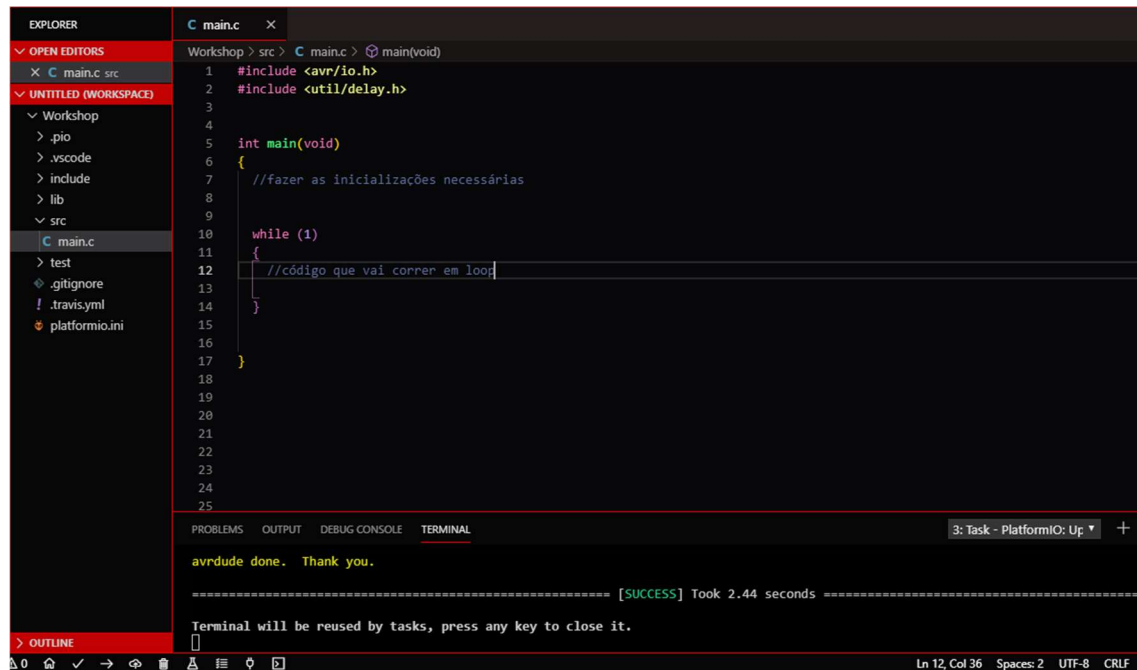
!(PIND & (1<<3)); //Retorna 0 caso o pino não esteja a receber nenhum sinal ou !0 se estiver a receber um

6- Inicio de um projeto:

A extensão platformio permite a criação de projetos para vários tipos de placas. Nós vamos utilizar um projeto já criado anteriormente que vai servir como esqueleto para a criação dos programas.



Depois de o projeto estar aberto, o ficheiro para a escrita de código esta na pasta src sobre o nome main.c.



```
1 #include <avr/io.h>
2 #include <util/delay.h>
3
4
5 int main(void)
6 {
7     //fazer as inicializações necessárias
8
9
10    while (1)
11    {
12        //código que vai correr em loop
13    }
14
15
16 }
17
18
19
20
21
22
23
24
25
```

avrdude done. Thank you.

===== [SUCCESS] Took 2.44 seconds =====

Terminal will be reused by tasks, press any key to close it.

Exercício 1:

Fazer o LED na saída 13 (externo ou interno do arduino) piscar de 1 em 1 segundo.

Para fazer temporizações vamos usar a função `_delay_ms(x)` que introduz um atraso no programa de x milissegundos. Para saber qual o registo correspondente ao pino 13 consultar o esquema de pinos do arduino uno e ver a que registo ta associado o pino.


```

void ISR1_init(void){
    DDRB &= ~(1<<2);      //ativar o PD2 como entrada
    PORTD = PORTD | (1<<2); //ativar a resistência de pull-up
    EICRA = EICRA | (2<<ISC00); //configurar o pedido de interrupção com falling edge
    EIMSK = EIMSK | (1<<INTF0); //habilitar o pino 2 como interrupção
    EIFR |= (1<<INTF0);
    sei();
}

```

```

ISR (INT0_vect){ //interrupção através da ativação do botão

```

```

    //código que se pretende que seja executado quando o botão for premido

```

```

}

```

Apenas os pinos 2 e 3 podem ser configurados como entradas com a funcionalidade de interrupção.

Timers:

Há três timers (TC0, TC1 e TC2) que fazem a contagem dos pulsos do cristal de quartzo. Os timers TC0 e TC2 tem 8 bits e o TC1 16.

Equação do tempo:

$$CP * TP * CNT = 1.600.000$$

Como estamos a usar um arduino, o valor de CP (clock prescaler) é igual a 1.

O valor TP (timer prescaler) pode tomar os valores de:

Valores possíveis para TP							
TC1/TC0	1	8	64	256	1024		
TC2	1	8	32	64	128	256	1024

O valor de CNT tem de ser preferencialmente um valor inteiro.

Para tal, temos duas opções:

- TP=64 e CNT=25000;
- TP=256 e CNT=6250;

Como apenas o TC1 ultrapassa o CNT=256, por ser de 16 bits, vamos usá-lo. Poderíamos usar os outros timers, mas no entanto, por estes serem de 8 bits iríamos ter erro na contagem do tempo.

```
//Configurar o interrupt para o Timer
#define MIN 65536-6250 //CNT = 6250
void tc1_init(void)
{
    TCCR1B = 0; //Parar TC1
    TIFR1 = (7<<TOV1) | (1<<ICF1); //Limpar interrupção pendente
    TCCR1A = 0; // Ativar modo normal
    TCNT1 = MIN; // Carregar o valor do limite inferior
    TIMSK1 = (1<<TOIE1); // Habilitar a interrupção por overflow
    TCCR1B = 4; //Indicar TC1 (TP=256)
    sei(); // habilitar interrupções
}
```

Neste exemplo usamos CP=1, TP=256 e CNT 6250;

Sempre que ocorrer o overflow do timer, ou seja, passarem 100 milissegundos o seguinte excerto de código é executado:

```
ISR (TIMER1_OVF_vect) // interrupção através do overflow do timer
{
    TCNT1 = MIN; //Reset para o valor mínimo
    /*condição para decrementar o timer*/
}
```

*Exemplo para os interrupts no fim do guião.

Exercício 3:

Repetir o exercício 1, mas usando timers em vez da função de delay.

Extra:

Criar um programa que começa com um LED a piscar e outro apagado. Quando o botão for pressionado, o LED para de piscar, ficando apagado, e é ligado outro LED de um pino diferente. Voltando a premir o botão, volta tudo ao início.

Usar o LED da placa (pino 13).

O outro LED tem de ser um pino entre o 4 e 7.

Usar a interrupção por botão para reconhecer a ativação do botão (também é possível usar o primeiro exemplo para ler o estado do botão para os aventureiros)

NÃO usar a função delay, apenas timers 😊.

(Solução na próxima página)



8-Exemplo

```
#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

uint8_t timer=5;

//Configurar o interrupt para o Timer
#define MIN 65536-6250 //CNT = 6250

void tc1_init(void)
{
    TCCR1B = 0; //Parar TC1
    TIFR1 = (7<<TOV1) | (1<<ICF1); //Limpar interrupção pendente
    TCCR1A = 0; // Ativar modo normal
    TCNT1 = MIN; // Carregar o valor do limite inferior
    TIMSK1 = (1<<TOIE1); // Habilitar a interrupção por overflow
    TCCR1B = 4; //Indicar TC1 (TP=256)
    sei(); // habilitar interrupções
}

//configurar o interrupt do botão
void ISR1_init(void){

    DDRB &= ~(1<<2); //ativar o PD2 como entrada
    PORTD = PORTD | (1<<2); //ativar a resistência de pull-up
    EICRA = EICRA | (2<<ISC00); //configurar o pedido de interrupção com falling edge
    EIMSK = EIMSK | (1<<INTF0); //habilitar o pino 2 como interrupção
    EIFR |= (1<<INTF0);
    sei();

}
```

```

int main(void)
{
    tc1_init();    //chamada da função de interrupt do timer
    ISR1_init();   //chamada da função de interrupt do botão

    while (1)
    {

        if(timer==0){    //quando o contador de tempo é igual a 0, ou seja ja passou o tempo desejado.

            //código

        }
    }
}

ISR (TIMER1_OVF_vect) // interrupção através do overflow do timer
{
    TCNT1 = MIN;    //Reset para o valor mínimo
    if(timer>0){

        timer--;

    }

    //  como a interrupção dispara de 100 em 100 ms, o contador timer vai ser decrementado 5 vezes cada uma
    //  destas separada
    //  por 100 ms, resultando numa temporização de 0,5 segundos
}

ISR (INT0_vect){ //interrupção através da ativação do botão

    //código que se pretende que seja executado quando o botão for premido

```

9-Anexos:

Datasheet: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

PinOut Arduino uno: <https://www.circuito.io/blog/arduino-uno-pinout/>

Github workshop: <https://github.com/NEEECFEUP/WS-Introducao-Microcontroladores>

