



MINISTÉRIO DA EDUCAÇÃO

SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA CATARINENSE

CAMPUS LUZERNA

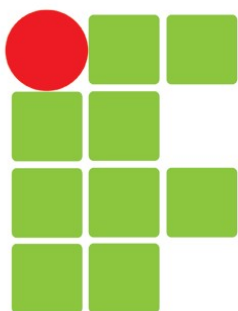
**CURSO DE EDUCAÇÃO PROFISSIONAL TÉCNICO DE
NÍVEL MÉDIO SUBSEQUENTE EM AUTOMAÇÃO
INDUSTRIAL**

Programação Aplicada a Microcontroladores

Aluno: _____

Prof: Ricardo Kerschbaumer

Luzerna, 2018



**INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
CATARINENSE**

Programação Aplicada a Microcontroladores

Este material foi desenvolvido pelo professor Ricardo Kerschbaumer para ser utilizado na componente curricular de programação aplicada a microcontroladores do curso de educação profissional técnico de nível médio subsequente em automação industrial. O objetivo deste material é servir como material de apoio as aulas teóricas e práticas, bem como de material de estudo para que os alunos possam tirar suas dúvidas nos períodos extra classe.

Os assuntos são distribuídos em aulas, cada aula diz respeito a um determinado assunto e não necessariamente a um encontro.

No decorrer do material serão apresentados vários exercícios e vários exemplos. Os exemplos estarão dispostos em quadros com o fundo escurecido, como no exemplo a seguir.

```
void setup() {  
  // Código executado uma vez:  
}  
  
void loop() {  
  // Código executado repetidamente  
}
```

Bons estudos.

Sumário

Aula 1 - Sistemas computacionais.....	8
1.1 Objetivo:.....	8
1.2 Introdução.....	8
1.2.1 Sinais Analógicos e Sinais Digitais.....	8
1.2.2 Portas lógicas.....	9
1.2.3 Registradores.....	9
1.2.4 Memórias.....	11
1.2.5 Memória RAM (Memória de Acesso Aleatório).....	12
1.2.6 Memórias ROM.....	13
1.2.7 Memórias ROM Programáveis (Prom's).....	13
1.2.8 Memórias ROM Programáveis e Apagáveis (EPROM's, EEPROMS's e FLASH).....	13
1.2.9 Associação de memórias.....	13
1.3 Componentes básicos de um sistema computacional.....	14
1.4 Processador.....	15
1.5 Memória.....	17
1.6 Dispositivos de entrada e saída.....	18
1.7 Conclusão.....	18
Aula 2 - Arquitetura Dos Microcontroladores.....	20
2.1 Objetivo:.....	20
2.2 Arquitetura interna dos microprocessadores.....	20
2.3 Registradores de propósito geral.....	21
2.4 Unidade Lógica Aritmética (ULA ou ALU).....	21
2.5 Registrador temporário.....	21
2.6 Acumulador.....	21
2.7 Program Counter (PC).....	21
2.8 Registrador de Instrução.....	21
2.9 Decodificador de Instrução e Unidade de Controle.....	22
2.10 Unidade de Deslocamento.....	22
2.11 Arquitetura C.I.S.C. versus R.I.S.C.....	22
2.12 Arquitetura HARVARD versus VON NEUMANN.....	23
2.13 Sinais de Controle entre microprocessador e Memória.....	24
2.14 Microcontroladores e Microprocessadores.....	25

Aula 3 - Lógica de programação.....	26
3.1 Objetivo:.....	26
3.2 Programas.....	26
3.3 Lógica de programação.....	27
3.4 Linguagem de programação.....	27
3.5 Algoritmos.....	28
3.5.1 Regras para construção do Algoritmo.....	29
3.5.2 Exemplo de Algoritmo.....	29
3.6 Fluxograma.....	30
3.7 Conclusão.....	32
Aula 4 - Fluxogramas parte 1.....	33
4.1 Objetivo:.....	33
4.2 Introdução.....	33
4.2.1 Variáveis e atribuição de valores.....	33
4.2.2 Entradas e saídas.....	34
4.3 Fluxogramas com o software Flowgorithm.....	34
4.3.1 Variáveis e no Flowgorithm.....	35
4.4 Entradas e saídas no Flowgorithm.....	37
4.5 Conclusão.....	43
4.6 Lista de exercícios.....	43
Aula 5 - Fluxogramas parte 2.....	44
5.1 Objetivo:.....	44
5.2 Introdução.....	44
5.2.1 Estruturas de controle.....	44
5.2.2 Estruturas de repetição.....	45
5.3 Estruturas de controle no Flowgorithm.....	45
5.4 Estruturas de repetição no Flowgorithm.....	47
5.4.1 O laço “Enquanto” no Flowgorithm.....	48
5.4.2 O laço “Para” no Flowgorithm.....	49
5.4.3 O laço “Fazer” no Flowgorithm.....	50
5.5 Conclusão.....	51
5.6 Lista de exercícios.....	51
Aula 6 - Introdução ao Arduino.....	52

IFC - Instituto Federal Catarinense, Campus Luzerna - Programação Aplicada a Microcontroladores

6.1	Objetivo:.....	52
6.2	Introdução.....	52
6.3	O microcontrolador utilizado no Arduino.....	54
6.4	Alimentação do Arduino.....	55
6.5	Entradas e saídas do Arduino.....	56
6.6	O ambiente de desenvolvimento do Arduino.....	57
6.7	Utilização da IDE Arduino.....	58
6.8	Conclusão.....	60
6.9	Exercício.....	61
Aula 7 - Programação do Arduino.....		62
7.1	Objetivo:.....	62
7.2	Introdução.....	62
7.3	Programação do Arduino.....	62
7.3.1	Comentários.....	62
7.3.2	Finalização de instruções com ponto e vírgula.....	63
7.3.3	Definição de blocos de código com chaves.....	63
7.3.4	Funções.....	64
7.3.5	A função "void setup()".....	64
7.3.6	A função "void loop()".....	64
7.3.7	Funções de entrada e saída.....	65
7.3.8	Intervalos de tempo.....	67
7.4	Conclusão.....	68
7.5	Exercício.....	68
Aula 8 - Variáveis e tomada de decisões.....		70
8.1	Objetivo:.....	70
8.2	Variáveis no Arduino.....	70
8.3	Operadores.....	71
8.4	Tomadas de decisão no Arduino.....	73
8.5	Exemplo de tomada de decisão.....	76
8.6	Conclusão.....	76
8.7	Exercício.....	77
Aula 9 - Entradas e saídas digitais.....		78
9.1	Objetivo:.....	78

9.2 Entradas digitais no Arduino.....	78
9.3 Saídas digitais no Arduino.....	79
9.3.1 Conexão de reles.....	80
9.3.2 Conexão de um motor utilizando transistor mosfet.....	80
9.3.3 Conexão de lâmpada de sinalização.....	80
9.3.4 Conexão de pequenos motores.....	81
9.3.5 Conexão de motores de passo.....	81
9.3.6 Conexão de motores CC com reversão.....	82
9.3.7 Conexão de displays de LED.....	82
9.3.8 Circuitos especializados de saída.....	83
9.4 Shields de entrada e saída digital para Arduino.....	83
9.5 Conclusão.....	84
9.6 Exercício.....	84
Aula 10 - Funções e Laços de Repetição.....	85
10.1 Objetivo:.....	85
10.2 Funções na programação do Arduino.....	85
10.3 Estruturas de controle.....	88
10.3.1 O laço “Enquanto” ou while.....	88
10.3.2 O laço “Fazer” ou do-while.....	89
10.3.3 O laço “Para” ou for.....	90
10.4 Conclusão.....	92
10.5 Exercício.....	93
Aula 11 - Interrupções e Contadores.....	94
11.1 Objetivo:.....	94
11.2 Interrupções no Arduino.....	94
11.2.1 Configuração das Interrupções externas.....	95
11.3 Contadores e Temporizadores no Arduino.....	98
TCCR1A – Registrador de controle do Timer 1 A.....	99
TCCR1B – Registrador de controle do Timer 1 B.....	99
11.4 Conclusão.....	100
11.5 Exercício.....	100
Aula 12 - Entradas e Saídas Analógicas.....	101
12.1 Objetivo:.....	101

<u>IFC - Instituto Federal Catarinense, Campus Luzerna - Programação Aplicada a Microcontroladores</u>	
12.2	O hardware do conversor Analógico/Digital (A/D).....101
12.3	A utilização do conversor Analógico/Digital do Arduino.....103
12.4	Sinais PWM.....105
12.5	Saídas PWM no Arduino.....105
12.6	Conclusão.....107
12.7	Exercício.....107
Aula 13	- Comunicação Serial no Arduino.....108
13.1	Objetivo:.....108
13.2	Fundamentos da Comunicação Serial.....108
13.3	O padrão RS232.....110
13.4	O padrão RS485.....112
13.5	Comunicação Serial no Arduino.....113
13.5.1	O padrão RS232 no Arduino.....115
13.5.2	O padrão RS485 no Arduino.....116
13.5.3	Programação do Arduino para comunicação serial.....117
13.6	Conversão USB - Serial.....120
13.6.1	Comunicação Serial Através da porta USB do Arduino.....120
13.6.2	Conversores USB-Serial comerciais.....122
13.7	Conclusão.....122
13.8	Exercício.....123
Aula 14	- Circuitos Com Microcontroladores.....124
14.1	Objetivo:.....124
14.2	A fonte de alimentação do microcontrolador.....124
14.3	A conexão do microcontrolador com a fonte.....125
14.4	O circuito de reset.....126
14.5	O circuito de clock.....126
14.6	Conector de programação.....127
14.7	Circuito completo.....127
14.8	O circuito programador.....128
14.9	A programação dos Microcontroladores.....129
14.10	Automatização de Processos utilizando Microcontroladores.....130
14.11	Conclusão.....130

AULA 1 - SISTEMAS COMPUTACIONAIS

Revisão de Eletrônica Digital e introdução aos Sistemas Computacionais.

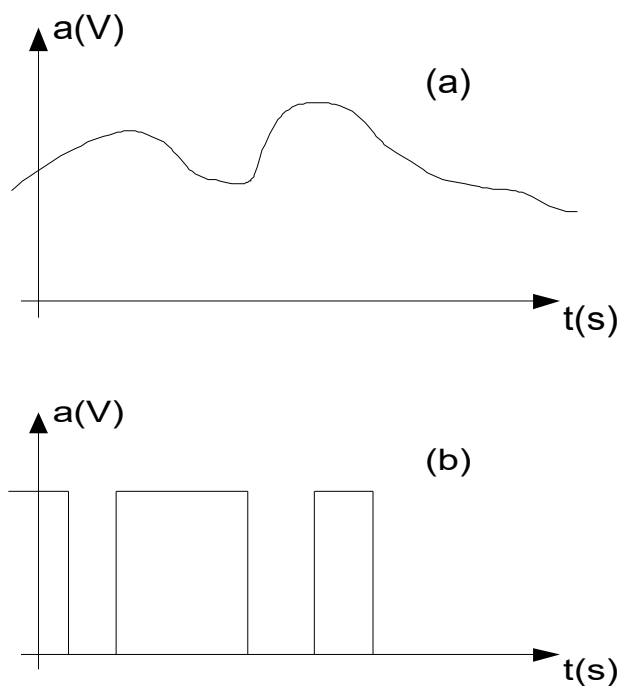
1.1 Objetivo:

O objetivo desta primeira aula é revisar alguns tópicos de eletrônica digital, necessários ao estudo de microcontroladores. Ao final desta aula os alunos serão capazes de compreender o funcionamento dos sistemas computacionais e seus principais elementos.

1.2 Introdução

1.2.1 Sinais Analógicos e Sinais Digitais

Apesar de todos os sinais que circulam nos circuitos serem sinais elétricos, eles podem ser divididos em duas classes, analógicos e digitais. Nas figuras a seguir tem-se um exemplo de cada uma dessas classes de sinais.

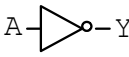
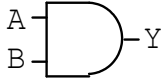
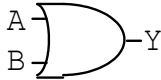
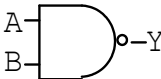




(a) Sinal analógico, (b) Sinal digital.

Observando a figura acima notamos que o sinal analógico pode assumir qualquer valor, enquanto que o sinal digital só pode assumir apenas dois valores, um alto e outro baixo. Assim é costume na eletrônica digital denotar o nível alto por (1) e o nível baixo por (0).

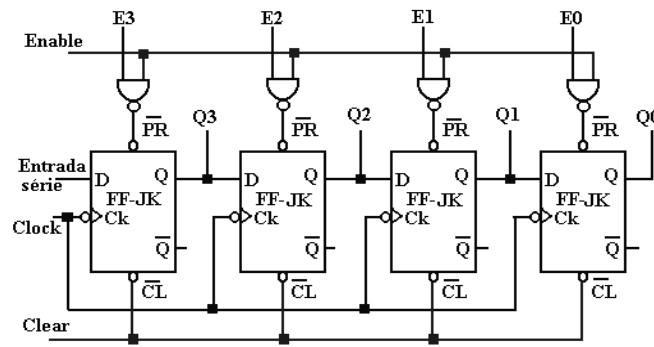
1.2.2 Portas lógicas

Na eletrônica digital a relação entre sinais é chamada de operação lógica, assim as operações mais comuns são realizadas por portas lógicas. A tabela a seguir apresenta as portas lógicas mais comuns e suas tabelas verdade.

NOT		<table border="1" data-bbox="853 470 1141 593"> <thead> <tr> <th>A</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> </tr> </tbody> </table>	A	Y	1	0	0	1									
A	Y																
1	0																
0	1																
AND		<table border="1" data-bbox="837 604 1157 795"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1
A	B	Y															
0	0	0															
0	1	0															
1	0	0															
1	1	1															
OR		<table border="1" data-bbox="837 795 1157 985"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	1
A	B	Y															
0	0	0															
0	1	1															
1	0	1															
1	1	1															
NAND		<table border="1" data-bbox="837 985 1157 1176"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	Y	0	0	1	0	1	1	1	0	1	1	1	0
A	B	Y															
0	0	1															
0	1	1															
1	0	1															
1	1	0															
NOR		<table border="1" data-bbox="837 1176 1157 1366"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	0
A	B	Y															
0	0	1															
0	1	0															
1	0	0															
1	1	0															
XOR		<table border="1" data-bbox="837 1366 1157 1556"> <thead> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	0
A	B	Y															
0	0	0															
0	1	1															
1	0	1															
1	1	0															

1.2.3 Registradores

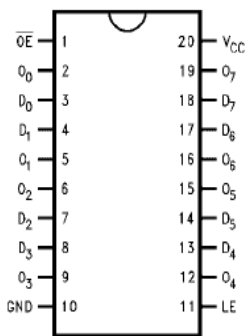
Os registradores são um grupo de elementos (flip-flop's, por ex.) capazes de armazenar uma informação, e que funcionam juntos como uma unidade. Os registradores mais simples armazenam uma palavra binária que pode ter "n" bits. O desenho abaixo apresenta um registrador simples para palavras de quatro bits.



Registrador de quatro bits.

Dentre os inúmeros componentes digitais, os registradores têm uma importância muito grande no que diz respeito aos microcontroladores. O que torna os registradores tão especiais é a capacidade deles de armazenar informações.

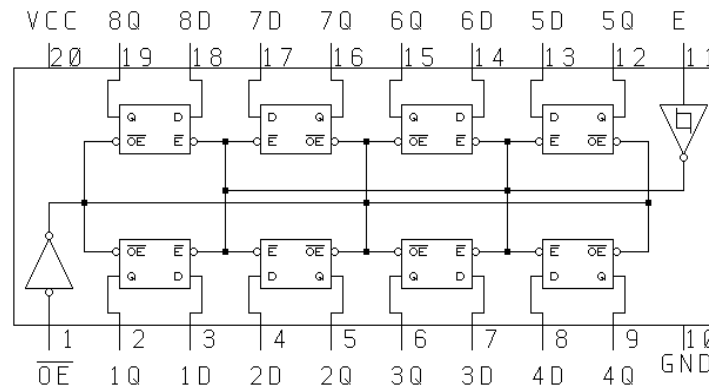
A seguir é apresentado um registrador comercial, o 74373. Este componente é um registrador de oito bits com entrada e saída paralela.



74373

Pinos	Função
D0 – D7	Entradas de dados
O0 – O7	Saída de dados
LE	Habilita a gravação dos dados, é ativo em nível lógico 1
\overline{OE}	Habilita a saída dos dados, é ativo em nível lógico 0

O diagrama lógico a seguir mostra a configuração interna do 74373, e abaixo tem-se a tabela verdade do componente, com o índice n de 0 a 7.



Lógica interna do 74373.

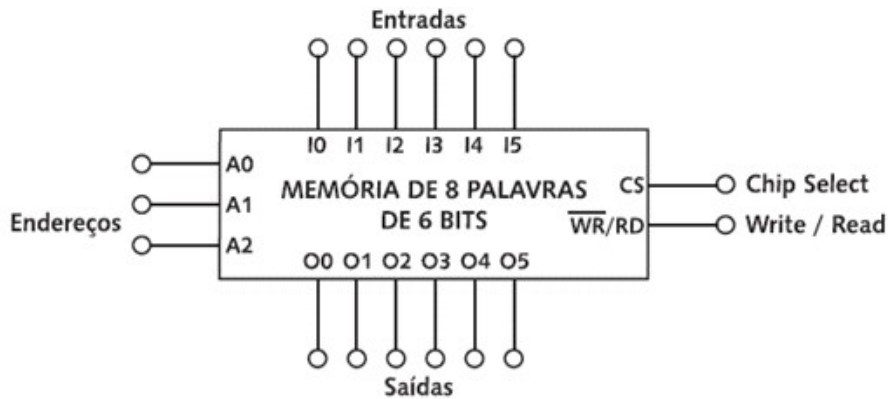
Entradas			Saída
LE	\overline{OE}	Dn	On
1	0	1	1
1	0	0	0
0	0	X	Não Muda
X	1	X	Z

Funcionamento do 74373.

1.2.4 Memórias

Já vimos que através de dispositivos eletrônicos como os registradores, podemos armazenar uma palavra de “n” bits. Memórias são dispositivos utilizados para armazenar palavras binárias na ordem de centenas de milhares. Podem-se utilizar flip-flop’s para o armazenamento em memórias ou outro dispositivo qualquer que sirva para este fim. Os circuitos de memória normalmente têm as seguintes entradas e saídas:

- Algumas vias de entrada para gravação e/ou saídas para leitura (que fisicamente podem ser as mesmas);
- Algumas vias para endereçamento, que selecionará qual registrador será lido/escrito, de acordo com um código (endereço de memória);
- Um pino que habilita o circuito (Chip Select - CS). Se o circuito não estiver habilitado, as saídas permanecem em alta impedância;
- Um pino de leitura/escrita, que habilita uma destas duas operações ou apenas leitura, dependendo do tipo de memória.



Circuito típico de memória.

É possível determinar o número de entradas de endereços para um determinado número de registradores utilizando a seguinte fórmula.

$$n_{\text{registradores}} = 2^{n_{\text{linhas endereço}}}$$

1.2.5 Memória RAM (Memória de Acesso Aleatório)

A memória RAM é uma memória de leitura e escrita, isto é, que pode ser gravada com um determinado valor e este valor pode ser posteriormente lido. Além disso, podemos acessar qualquer registrador desejado aleatoriamente para ler ou escrever uma palavra. A memória RAM comum necessita de alimentação elétrica para manter a integridade de seus dados. É por este motivo, pertencente ao grupo de memórias voláteis.

Quanto à sua construção, as memórias RAM podem ser de dois tipos básicos: estática ou dinâmica.

Na memória RAM estática, os bits são armazenados em flip-flop's individuais e permanecem armazenados indefinidamente enquanto o circuito possuir alimentação.

A memória RAM dinâmica armazena os bits através de carga em diminutos capacitores. Como um capacitor deste tipo ocupa muito menos espaço que um flip-flop em um CI, a memória dinâmica resulta bem mais compacta que a estática. Em compensação, o bit em um capacitor permanece íntegro por apenas uma fração de tempo (aprox. 2 ms), devido às fugas de corrente. Para contornar este problema este tipo de memória deve ter um circuito auxiliar que verifique temporariamente os capacitores e os recarregue se for necessário. Esta operação é denominada *refresh*.

A maioria das memórias tem saídas em coletor aberto ou terceiro estado para permitir a ligação em paralelo e conseqüentemente melhorar a capacidade de manuseio de dados. Assim, quando o “Chip Select” não estiver ativo, o componente ficará em estado de alta impedância, e não se pode nem escrever na memória nem ler os seus conteúdos. Isto significa que a memória estará desconectada dos demais componentes do circuito.

A operação de gravação ou escrita é feita colocando-se os dados nas vias de entrada, habilitando-se o chip, colocando-se os sinais de endereço na posição desejada e habilitando a escrita da memória. Deste modo os dados das vias de entrada serão escritos na posição selecionada. Do mesmo modo, a operação de leitura é feita habilitando-se o chip, colocando-se os sinais de endereço na posição desejada e habilitando a leitura da memória. Deste modo os dados da posição de memória selecionada ficarão disponíveis na saída, para leitura.

1.2.6 Memórias ROM

Uma memória ROM (Read Only Memory) é um tipo de memória no qual podemos ler, mas não escrever. Os conteúdos são fixos e inalterados, sendo estabelecidos na hora da fabricação. Em uma ROM, os conteúdos não precisam ser alterados. Portanto não necessitamos de flip-flop's ou dispositivos semelhantes. Uma ROM na verdade nada mais é do que um conversor de código e pode ser construído a partir de dispositivos mais simples e baratos que as portas normalmente utilizadas.

1.2.7 Memórias ROM Programáveis (Prom's)

Existem circuitos de ROM que permitem que o usuário estabeleça as informações que serão armazenadas, ao invés do fabricante. Estas memórias são chamadas de memórias PROM (Memórias de leitura programáveis). A gravação só pode ser feita uma única vez e não mais alterada. Normalmente a gravação é feita através da queima de elos fusíveis que determinam se a posição de memória conterà “um” ou “zero”.

1.2.8 Memórias ROM Programáveis e Apagáveis (EPROM's, EEPROMS's e FLASH)

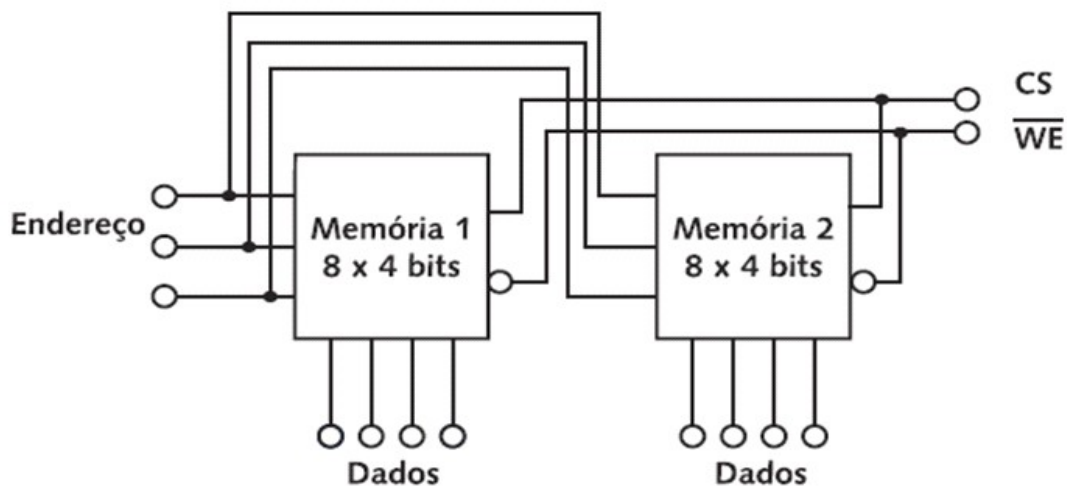
Na EPROM, os dados são armazenados em dispositivos baseados em MOSFET's. Estes dispositivos fazem ou não a conexão (guardam bit “um” ou “zero”) conforme haja ou não carga elétrica na porta do transistor. A programação é feita através de um programador de EPROM's. Uma característica importante é a de que a exposição à luz ultravioleta forte (por aproximadamente 30 min.) permite a fuga das cargas, apagando a memória. O apagamento possibilita uma nova

IFC - Instituto Federal Catarinense, Campus Luzerna - Programação Aplicada a Microcontroladores programação (gravação). Já nas memórias EEPROM e FLASH o processo para apagar pode ser feito eletricamente, facilitando muito assim o processo de alteração das informações armazenadas.

Apesar de estas memórias serem graváveis e apagáveis elas não são iguais as memórias RAM, pois as informações não são perdidas quando a energia é desligada. Outra característica que difere este tipo de memórias das memórias RAM é que o tempo gasto para armazenar a informação nestas é muito maior, e o número de gravações que se pode fazer é limitado.

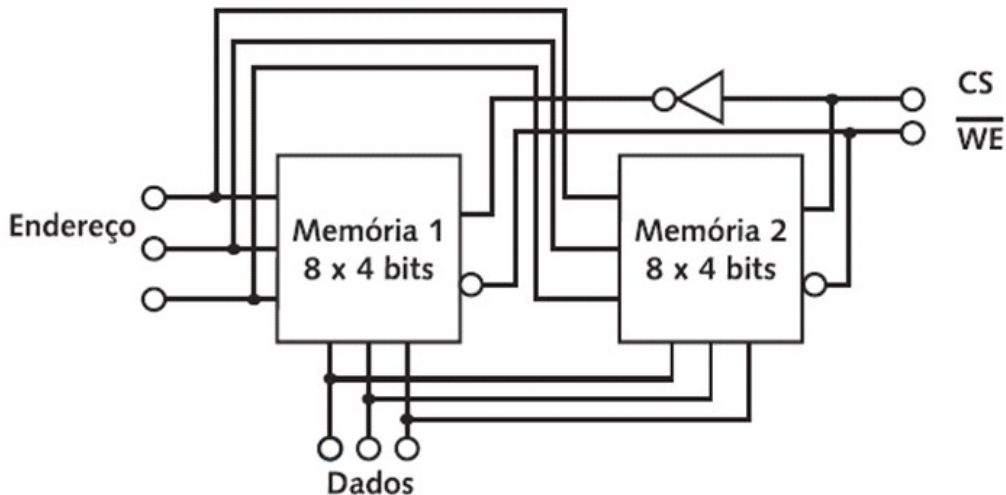
1.2.9 Associação de memórias

A ligação de memórias em paralelo é utilizada quando o número de palavras e/ou o número de bits por palavra disponível em um determinado circuito integrado não é suficiente. Abaixo está representada uma ligação em paralelo para aumentar o número de bits por palavra:



Ligação de Memórias em Paralelo para Aumentar Número de Bits por Palavra

O próximo desenho mostra uma ligação em paralelo para aumentar o número de palavras do conjunto:

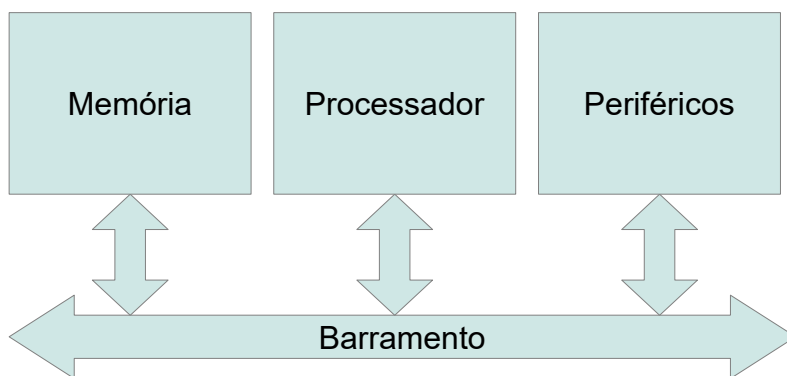


Ligação de Memórias em Paralelo para Aumentar Número de Palavras

1.3 Componentes básicos de um sistema computacional

Apesar da existência de uma grande diversidade de sistemas computacionais em termos de arquiteturas, pode-se enumerar, num ponto de vista mais genérico os componentes básicos desta classe de equipamentos. A Figura a seguir apresenta o esquema de um sistema computacional, destacando os elementos que o compõem.

Apesar da grande evolução ocorrida na área de informática desde o aparecimento dos primeiros computadores, o esquema apresentado na figura pode ser utilizado tanto para descrever um sistema computacional atual como os computadores da década de 40.



Elementos básicos de um sistema computacional

Os principais elementos de um sistema computacional são:

- O processador (ou microprocessador) é responsável pelo tratamento das informações armazenadas em memória (programas em código de máquina e dos dados).

- A memória é responsável pela armazenagem dos programas e dos dados.
- Periféricos, que são os dispositivos responsáveis pelas entradas e saídas de dados do computador, ou seja, pelas interações entre o computador e o mundo externo. Exemplos de periféricos são o monitor, teclados, mouses, impressoras, etc.
- Barramento, que liga todos estes componentes e é uma via de comunicação de alto desempenho por onde circulam os dados tratados pelo computador.

1.4 Processador

Um microprocessador, ou simplesmente processador, é um circuito integrado (ou chip), que é considerado o "cérebro" do computador (Figura a seguir). É ele que executa os programas, faz os cálculos e toma as decisões, de acordo com as instruções armazenadas na memória.



Processador de computador

Os microprocessadores formam uma parte importantíssima do computador, chamada de UCP (Unidade Central de Processamento), ou em inglês, CPU (Central Processing Unit). Ligando-se um microprocessador existem alguns chips de memória e alguns outros chips auxiliares, tornou-se possível construir um computador inteiro em uma única placa de circuito. Esta placa, como visto na Figura a seguir, é comumente chamada de placa-mãe.

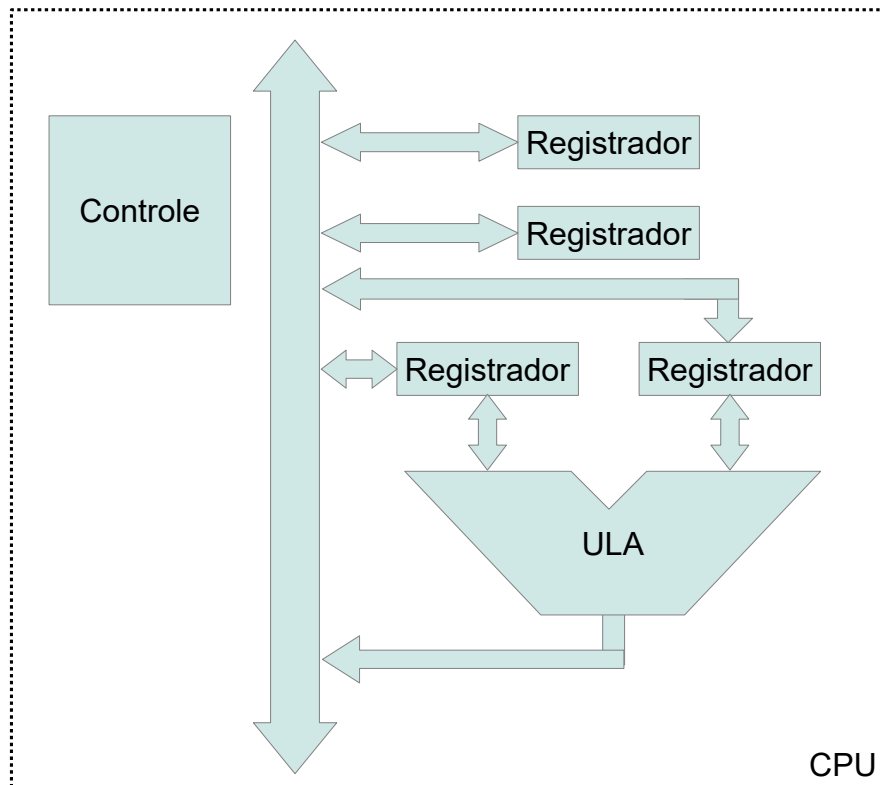


Placa-mãe de um computador

Não importa de que tipo de CPU estamos falando, seja um microcontrolador, microprocessador, ou uma das várias placas que formam a CPU de um computador de grande porte, podemos dizer que a CPU realiza as seguintes tarefas:

- a) Busca e executa as instruções existentes na memória. Os programas e os dados que ficam na memória não volátil, são transferidos para a memória. Uma vez estando na memória, a CPU pode executar os programas e processar os dados.
- b) Comanda todos os outros chips do computador.

A CPU é composta basicamente de três elementos: unidade de controle, unidade lógica e aritmética e registradores, conforme a figura a seguir.

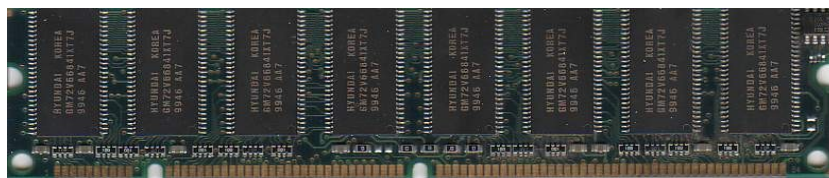


O funcionamento interno das CPUs será abordado na próxima aula.

1.5 Memória

Todo sistema computacional é dotado de uma quantidade de memória (que pode variar de máquina para máquina) a qual se constitui de um conjunto de circuitos capazes de armazenar os dados e os programas a serem executados pela máquina. Nós podemos identificar diferentes categorias de memória:

- A memória principal, ou memória de trabalho, onde normalmente devem estar armazenados os programas e dados a serem manipulados pelo processador, veja a figura a seguir.



Memória RAM

- A memória secundária que permitem armazenar uma maior quantidade de dados e instruções por um período de tempo mais longo; o disco rígido (figura a seguir) é o exemplo mais evidente de memória secundária de um computador, mas podem ser citados outros dispositivos como pendrives e cartões de memória.



Disco rígido de computador

1.6 Dispositivos de entrada e saída

Todos os sistemas computacionais necessitam interagir com o mundo exterior. A entrada e a saída de dados ou informações é crítica para seu funcionamento. Existem muitos dispositivos que podem ser conectados aos sistemas computacionais. Os mais comuns são o teclado, mouse e monitor, porém existem outros não tão comuns, tais como equipamentos científicos, equipamentos de aquisição e processamento de dados, máquinas industriais e assim por diante.

1.7 Conclusão

Esta aula apresentou de forma bastante superficial os fundamentos necessários à compreensão do funcionamento dos sistemas computacionais, bem como a formação do núcleo de um computador. Porém os sistemas computacionais são máquinas complexas, em constante atualização e envolvem muitos outros elementos de software e *hardware* para funcionarem.

AULA 2 - ARQUITETURA DOS MICROCONTROLADORES

Arquiteturas CISC, RISC, Harvard e Von Neumann

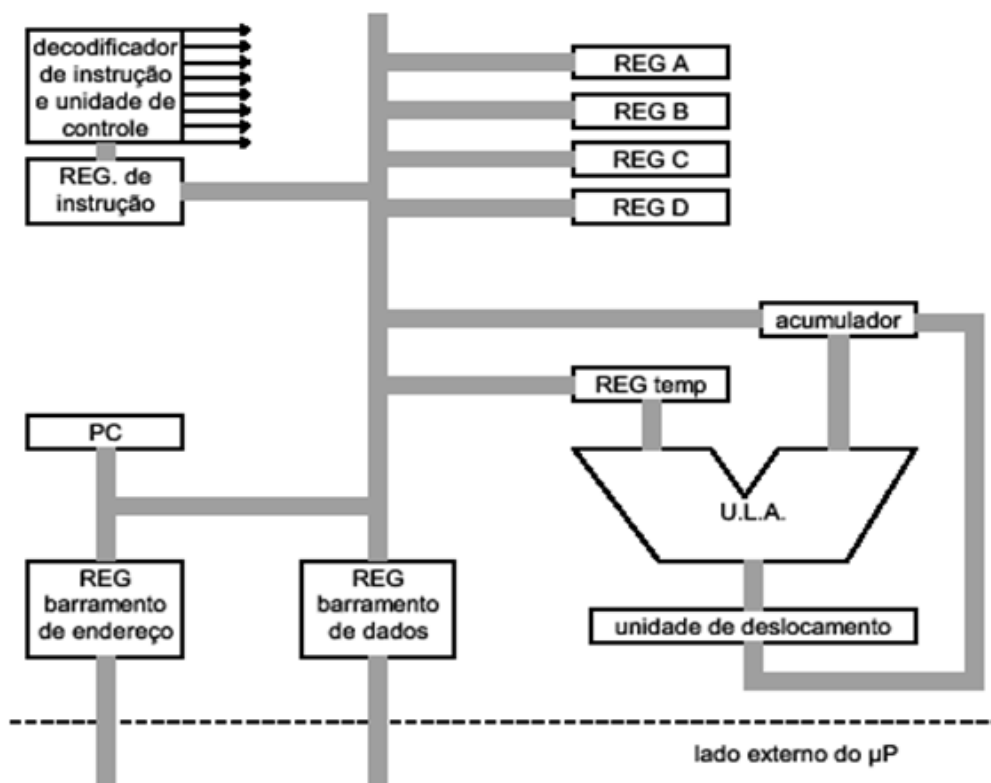
2.1 Objetivo:

Apresentar aos alunos as arquiteturas internas e externas mais comuns aos microcontroladores, ao final desta aula os alunos serão capazes de compreender como um microcontrolador trabalha internamente, como ele executa seus comandos e o que faz cada um de seus componentes.

2.2 Arquitetura interna dos microprocessadores

Os microprocessadores são o coração de um microcontrolador, assim para compreendermos o funcionamento de um microcontrolador é necessário primeiro compreender o funcionamento de um microprocessador.

Embora todos os microprocessadores tenham suas peculiaridades, a maioria deles possui grande semelhança quanto a seu modo geral de funcionamento. A figura a seguir ilustra a arquitetura básica de um microprocessador.



Arquitetura interna genérica de microprocessadores

2.3 Registradores de propósito geral

São registradores nomeados de Reg. A até Reg. D. O número destes registradores varia de um microprocessador para outro. Por exemplo, no AVR, são 32 registradores de 8 bits, no Z80, são 16 de 8 bits, no 8051 são 8 de 8 bits.

2.4 Unidade Lógica Aritmética (ULA ou ALU)

Essa unidade é o centro do microprocessador, ela possui somador, subtrator (em alguns, multiplicador e divisor), operadores AND, OR e XOR bit a bit, incrementador e decrementador, tudo integrado em uma única unidade. Portanto, todas as operações lógicas e aritméticas passam obrigatoriamente por esta unidade.

2.5 Registrador temporário

Serve apenas para armazenar um dos operadores da ULA.

2.6 Acumulador

Trata-se de um registrador especial dedicado às operações envolvendo a ULA. Esse registrador é um dos operandos envolvidos nas operações da ULA e também é o registrador que guarda o resultado dessa operação. Assim como os registradores de propósito geral, admite transferência bidirecional.

2.7 Program Counter (PC)

É nesse registrador que o microprocessador guarda o endereço de memória que aponta para a instrução do programa que está sendo executada. O microprocessador usa esse conteúdo para informar à memória o endereço onde está a instrução, faz a leitura desta instrução e guarda a instrução lida no REGISTRADOR DE INSTRUÇÃO. Logo após ter lido a instrução o conteúdo do registrador PC é automaticamente incrementado para que o microprocessador possa ler a próxima instrução.

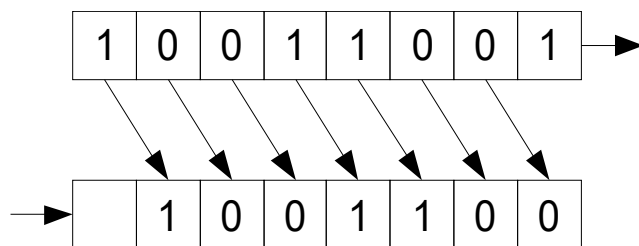
2.8 Registrador de Instrução

É nesse registrador que o microprocessador guarda a instrução lida da memória para que possa ser decodificada e executada.

2.9 Decodificador de Instrução e Unidade de Controle

Quando uma instrução é lida da memória, ela não passa de um byte qualquer. Como saber qual a instrução que corresponde a esse byte e como tomar as devidas providências (micro instruções) para fazer o que a instrução está mandando? Cada registrador do microprocessador precisa ser comandado pelos seus sinais de controle, não só os registradores, mas todo o sistema precisa ser comandado pelos sinais de controle para que todo o sistema possa funcionar. Esses sinais de controle precisam obedecer a uma seqüência adequada para que não ocorram conflitos. A instrução lida passa por uma unidade com um número imenso de portas lógicas que geram os sinais de controle de todo o sistema. Pode-se dizer que esta unidade é realmente o cérebro de todo o sistema.

2.10 Unidade de Deslocamento



Essa unidade contém um registrador de deslocamento série bidirecional e é capaz de realizar um deslocamento dos bits à esquerda ou à direita ou então não realizar deslocamento nenhum.

2.11 Arquitetura C.I.S.C. versus R.I.S.C.

A partir do 4004, com 46 instruções, começava a escalada dos microprocessadores, cada vez mais complexos com maior número de instruções. O 8008 possui 48 instruções. O 8080, 78 instruções. O 8085, aproximadamente 150 instruções, o Z80, mais de 500, o 8086/8088 já possui mais de 700 instruções e o 80386, mais de 1500. Isso nos mostra que, com o aumento do número de instruções, também crescia o número e a complexidade dos circuitos internos do microprocessador.

Alguns microprocessadores de arquitetura CISC atuais possuem um gigantesco volume de transistores incorporados no CHIP. Os microprocessadores 80x86 da Intel utilizados nos PCs são um exemplo típico de processadores CISC. Há algum tempo, a preocupação nos projetos de microprocessadores passou a ser a velocidade de processamento e não a sua complexidade. Por isso, foram criados microprocessadores com um conjunto reduzido de instruções (menos de 250 instruções), mas com alta velocidade de processamento (RISC).

CISC = Complex Instruction Set Code (conjunto de código de instruções complexo).

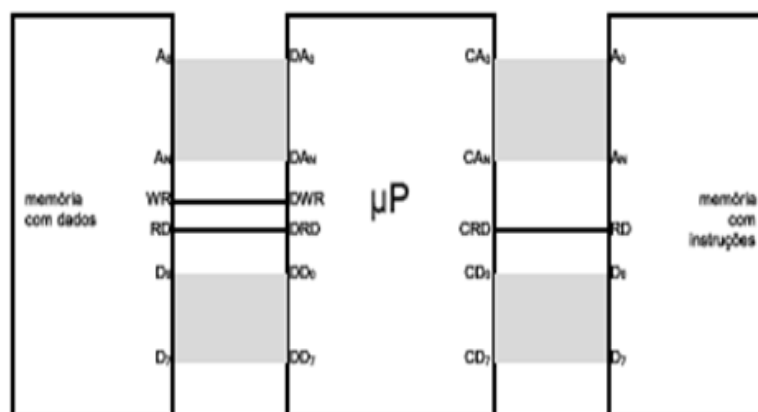
RISC = Reduced Instruction Set Code (conjunto de código de instruções reduzido).

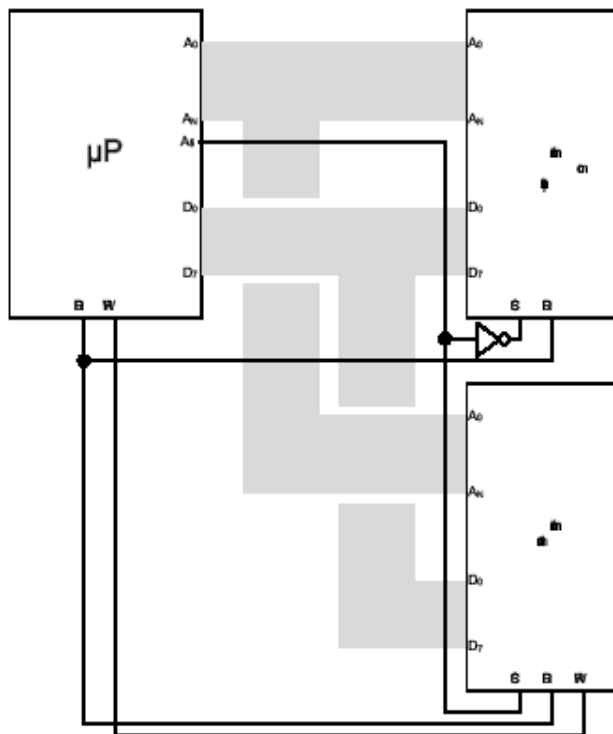
O ciclo de projeto de um microprocessador CISC é muito longo e difícil. O volume de testes para certificação do seu funcionamento é muito grande e a preocupação com a otimização do circuito para ganhar velocidade só vem depois do pleno estabelecimento do novo microprocessador. Em contrapartida, microprocessadores RISC têm um ciclo de projeto bem mais curto. Além disso, o enfoque é dado à elevação das taxas de “clock” e uso de barramentos “largos” (64 bits ou 128 bits). Em geral, o desempenho de microprocessadores RISC costuma ser melhor que microprocessadores CISC.

Outro aspecto importante é o tamanho dos programas. Imagine um programa que faça uma movimentação de dados de 1000 bytes de uma parte da memória para outra. Em um microprocessador CISC esse programa é curto (podemos supor algo em torno de 5 ou 6 bytes). Isso porque utiliza instruções complexas. Em um microprocessador RISC, a mesma tarefa pode ser desempenhada por um programa maior (algo em torno de 10 a 15 bytes). Esse microprocessador possui instruções muito simples e certamente precisará de várias instruções para uma tarefa feita por uma única instrução complexa. Isso permite concluir que os programas dos microprocessadores RISC são maiores.

2.12 Arquitetura HARVARD versus VON NEUMANN

As arquiteturas Harvard e Von Neumann dizem respeito à forma como a memória é conectada ao microprocessador. Na arquitetura Harvard, há dois barramentos de endereços independentes e dois de dados também independentes. Enquanto um desses barramentos serve para a leitura de instruções de um programa, o outro serve para a leitura e escrita de dados. Com isso, é possível operar simultaneamente uma instrução e um byte de dados. Isso garante maior velocidade de processamento. Atualmente, os processadores de sinais digitais (DSP – digital signal processor) utilizam essa arquitetura. DSP’s são processadores especializados no processamento dos sinais em tempo real.





Arquitetura Von Neumann

Na arquitetura Von Neumann, há apenas um barramento de dados e endereços. Neste caso, as instruções estariam em uma faixa de endereços que ative a memória que possui as instruções e os dados estão em outra faixa de endereços que ative outra memória onde se pode ler e escrever os dados. Por exemplo, qualquer endereço entre 0000h e 1FFFh ativa a memória de programas, e entre 8000h e FFFFh ativa a memória de dados. Nessa arquitetura só é possível o acesso a uma memória de cada vez.

Comparando ambas, conclui-se que Harvard é mais veloz, mas exige mais um barramento, Von Neumann utiliza apenas um barramento, mas não pode efetuar acessos simultâneos às memórias.

2.13 Sinais de Controle entre microprocessador e Memória

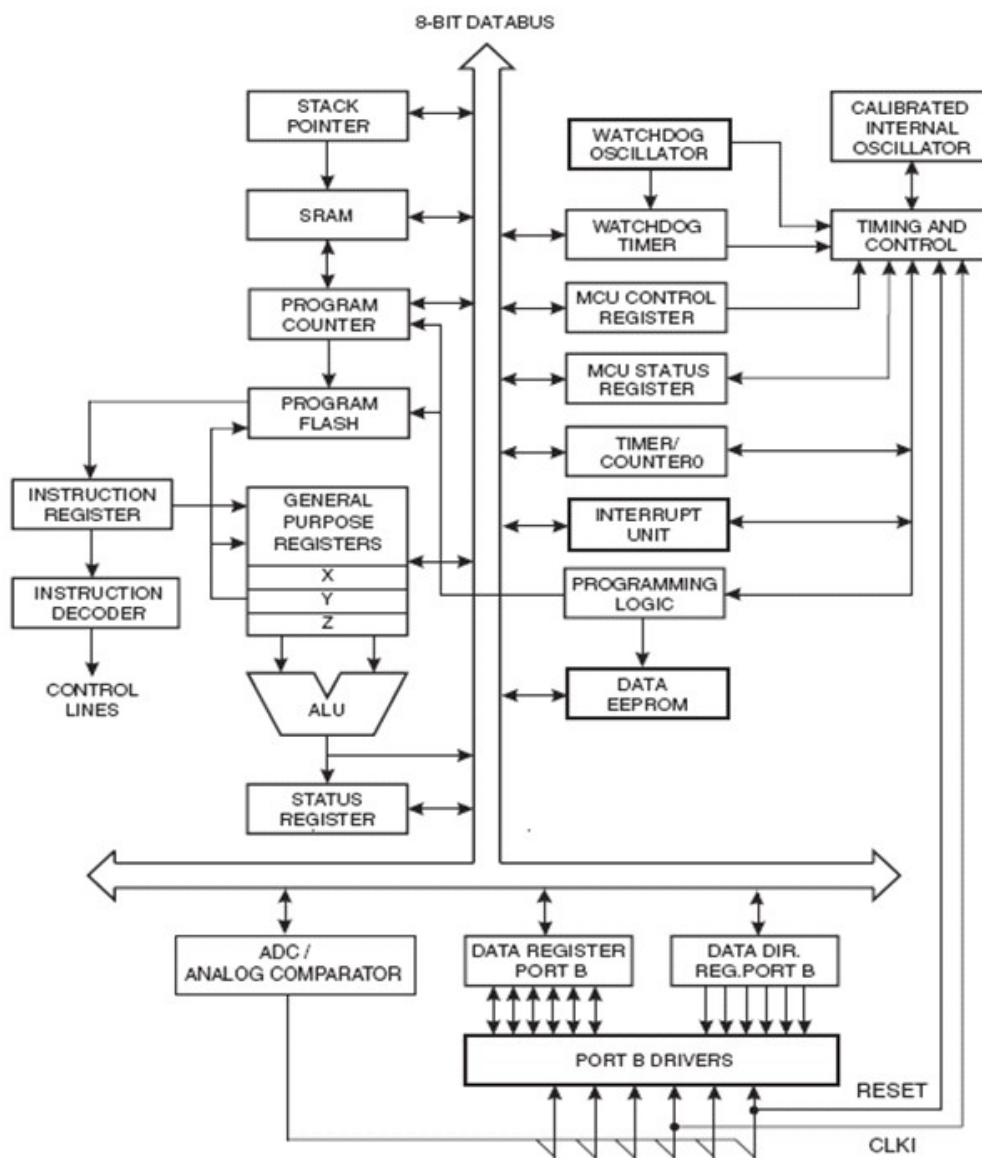
Para que o microprocessador possa se comunicar com as memórias são necessários alguns sinais de controle:

- Endereços: Esse conjunto de sinais serve para localizar a informação dentro da memória.
- Dados: Trata-se de 8 bits que conduzirão o byte lido da memória para o microprocessador ou do microprocessador para a memória.
- RD (Read): Sinal enviado pelo microprocessador requisitando que a memória coloque no barramento de dados o byte previamente endereçado. O microprocessador lê esse byte e logo após desativa o sinal RD.
- WR (write): Sinal enviado pelo microprocessador requisitando que a memória armazene o byte presente no barramento de dados no endereço presente no barramento de endereços. (o byte e o endereço já devem estar preparados antes da ativação deste sinal).

2.14 Microcontroladores e Microprocessadores

Microprocessadores são circuitos integrados que reúnem todos os componentes necessários para a execução dos comandos de um programa, mas não possuem memórias, nem dispositivos de entrada e saída ou circuito de clock.

Já os microcontroladores por sua vez são circuitos integrados que possuem, além de um microprocessador, todos os requisitos para que o sistema possa funcionar sem a necessidade de componentes externos. Os microcontroladores possuem em seu interior o microprocessador chamado de CPU, as memórias de dados e programa e uma infinidade de periféricos como circuito de clock, temporizadores, contadores, interfaces de entrada e saída, etc. A figura a seguir apresenta a arquitetura de um microcontrolador típico.



Arquitetura de um microcontrolador

AULA 3 - LÓGICA DE PROGRAMAÇÃO

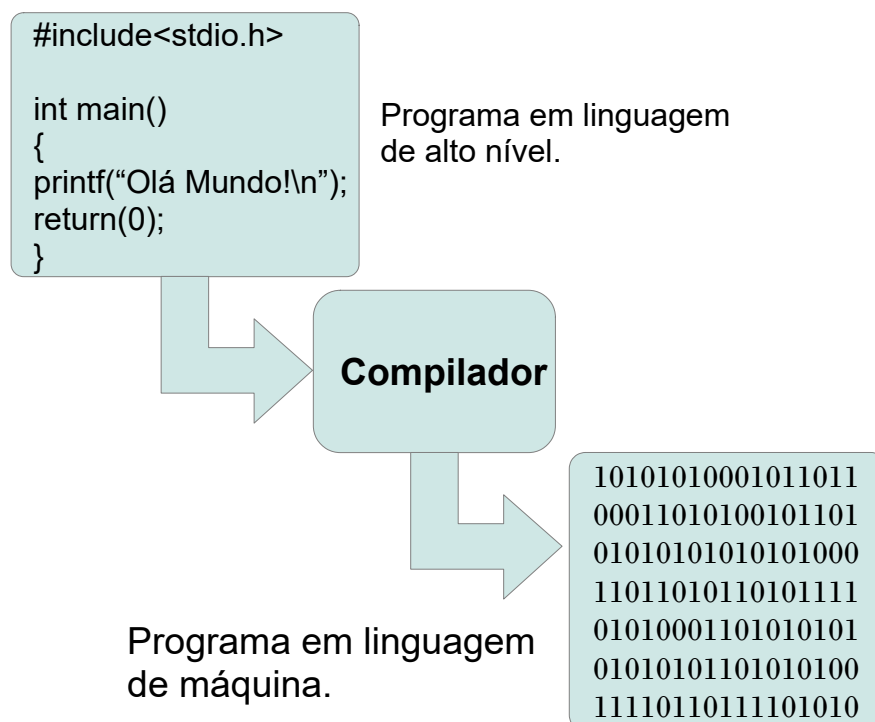
Noções sobre a lógica de programação, algoritmos e fluxogramas

3.1 Objetivo:

O objetivo desta aula é mostrar aos alunos como funciona um programa de computador e como é possível resolver problemas de várias áreas utilizando programação. Inicialmente veremos o que é um programa de computador, na sequência veremos como organizar as ideias de forma a resolver um determinado problema. A partir destes conceitos serão definidos estudados os algoritmos. E finalmente abordaremos a representação gráfica de um algoritmo através de fluxogramas.

3.2 Programas

Para desenvolver um programa é necessário elaborar uma sequência lógica de instruções de modo que estas instruções resolvam o problema em questão. O conjunto de instruções que podem ser utilizadas para resolver os problemas é definido pela linguagem de programação utilizada. Para facilitar a tarefa dos programadores as linguagens de programação atribuem nomes compreensíveis a suas instruções, estas instruções são então convertidas para a linguagem do computador. A figura a seguir mostra este processo.



3.3 Lógica de programação

Lógica de programação é a técnica de encadear pensamentos para atingir determinado objetivo. Estes pensamentos, podem ser descritos como uma sequência de instruções, que devem ser seguidas para se cumprir uma determinada tarefa. Assim são executados passos lógicos até que se possa atingir um objetivo ou que se encontre a solução de um problema.

Nas linguagens de programação as instruções são um conjunto de regras ou normas definidas para a realização das tarefas. Na informática, porém, instrução é a informação que indica a um computador uma ação elementar a ser executada pelo processador.

As instruções isoladamente não resolvem os problemas. É necessário ordenar as instruções de forma correta para que o programa funcione.

Como exemplo podemos imaginar a sequência de atividades necessárias para que se construa uma casa. É claro que não podemos construir o telhado antes de construir as fundações. Na programação de computadores as coisas funcionam da mesma forma.

Um programa de computador é uma coleção de instruções que descrevem uma tarefa a ser realizada por um computador. O termo pode ser uma referência ao código fonte, escrito em alguma linguagem de programação, ou ao arquivo que contém a forma executável deste código fonte. São exemplos de programas de computador os editores de texto, jogos, navegadores de internet etc.

3.4 Linguagem de programação

Para construir os programas de computador são utilizadas linguagens de programação. Uma linguagem de programação é um método padronizado para expressar instruções para um computador. É também um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador. Uma linguagem permite que um programador especifique precisamente sobre quais dados um computador vai atuar, como estes dados serão armazenados ou transmitidos e quais ações devem ser tomadas sob várias circunstâncias.

O conjunto de palavras, compostos de acordo com essas regras, constituem o código fonte de um programa. Esse código fonte é depois traduzido para código de máquina, que é executado pelo processador. O responsável por traduzir o documento de texto que contém o código fonte em um programa é o compilador.

Uma das principais metas das linguagens de programação é permitir que programadores tenham uma maior produtividade, permitindo expressar suas intenções mais facilmente do que quando comparado com a linguagem que um computador entende nativamente (código de máquina). Assim, linguagens de programação são projetadas para adotar uma sintaxe de nível mais alto, que pode ser mais facilmente entendida por programadores humanos. Linguagens de programação são ferramentas importantes para que programadores e engenheiros de software possam escrever programas mais organizados e com maior rapidez.

3.5 Algoritmos

Algoritmos são trechos de programas que tem a finalidade de resolver um problema ou executar uma função. O conceito de algoritmo é frequentemente ilustrado pelo exemplo de uma receita, embora muitos algoritmos sejam mais complexos. Eles podem repetir passos (fazer iterações) ou necessitar de decisões (tais como comparações ou lógica) até que a tarefa seja completada. Formalmente os algoritmos são uma sequência finita de passos que levam a execução de uma tarefa.

Qualquer tarefa, que pode ser descrita através de uma sequência de passos, também pode ser representada através de um algoritmo. A seguir temos um exemplo de um algoritmo para fazer um bolo.

1. Escolher a receita do bolo.
2. Separar os ingredientes.
3. Misturar os ingredientes conforme a receita.
4. Despejar a mistura em uma forma.
5. Ajustar a temperatura do forno.
6. Assar o bolo.
7. Retirar o bolo da forma.

Na programação de computadores é comum escrevermos algoritmos em pseudocódigo. Este nome é uma alusão à posterior implementação em uma linguagem de programação, ou seja, quando formos programar em uma linguagem, por exemplo a linguagem C, geraremos código em C. Por isso os algoritmos são independentes das linguagens de programação. Ao contrário de uma

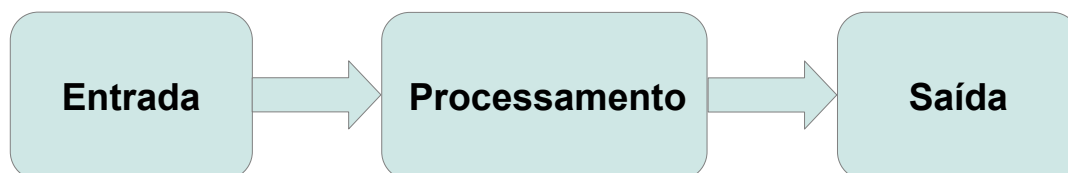
IFC - Instituto Federal Catarinense, Campus Luzerna - Programação Aplicada a Microcontroladores
linguagem de programação, não existe um formalismo rígido de como deve ser escrito o algoritmo. O algoritmo deve ser fácil de se interpretar e fácil de codificar. Ou seja, ele deve ser o intermediário entre a linguagem falada e a linguagem de programação.

3.5.1 Regras para construção do Algoritmo

Para escrever um algoritmo precisamos descrever a sequência de instruções, de maneira simples e objetiva. Para isso utilizaremos algumas técnicas:

- Usar somente um verbo por frase
- Imaginar que você está desenvolvendo um algoritmo para pessoas que não trabalham com informática
- Usar frases curtas e simples
- Ser objetivo
- Procurar usar palavras que não tenham sentido dúbio

Para montarmos um algoritmo, precisamos primeiro dividir o problema apresentado em três fases fundamentais, conforme figura a seguir.



Onde temos:

ENTRADA: São os dados de entrada do algoritmo

PROCESSAMENTO: São os procedimentos utilizados para chegar ao resultado final

SAÍDA: São os dados já processados

3.5.2 Exemplo de Algoritmo

Imagine o seguinte problema: Calcular a média final dos alunos. Os alunos realizarão quatro provas: P1, P2, P3 e P4.

Onde:

$$Média\ Final = \frac{P1 + P2 + P3 + P4}{4}$$

Para montar o algoritmo proposto, faremos três perguntas:

a) Quais são os dados de entrada? R:

Os dados de entrada são P1, P2, P3 e P4

b) Qual será o processamento a ser utilizado?

R: O procedimento será somar todos os dados de entrada e dividi-los por 4 (quatro)

$$\frac{P1 + P2 + P3 + P4}{4}$$

c) Quais serão os dados de saída?

R: O dado de saída será a média final

O algoritmo para realizar esta tarefa é apresentado a seguir.

```
Receba a nota da prova1
Receba a nota de prova2
Receba a nota de prova3
Receba a nota da prova4
Some todas as notas
Divida o resultado por 4
Mostre o resultado da divisão
```

Após elaborarmos um algoritmo é necessário testarmos seu funcionamento. Para isso podemos realizar simulações, utilizando amostras de dados e executando as tarefas manualmente.

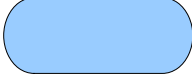


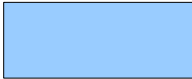
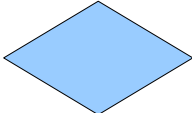
3.6 Fluxograma

Fluxograma é um tipo de diagrama, e pode ser entendido como uma representação esquemática de um processo, muitas vezes feito através de gráficos que ilustram de forma descomplicada a transição de informações entre os elementos que o compõem. Podemos entendê-lo, na prática, como a documentação dos passos necessários para a execução de um processo qualquer.

Na programação os fluxogramas são utilizados para representar graficamente o comportamento dos algoritmos, facilitando assim a compreensão da ordem de execução de cada um dos comandos.

O diagrama de blocos é uma forma padronizada e eficaz para representar os passos lógicos de um determinado processamento.

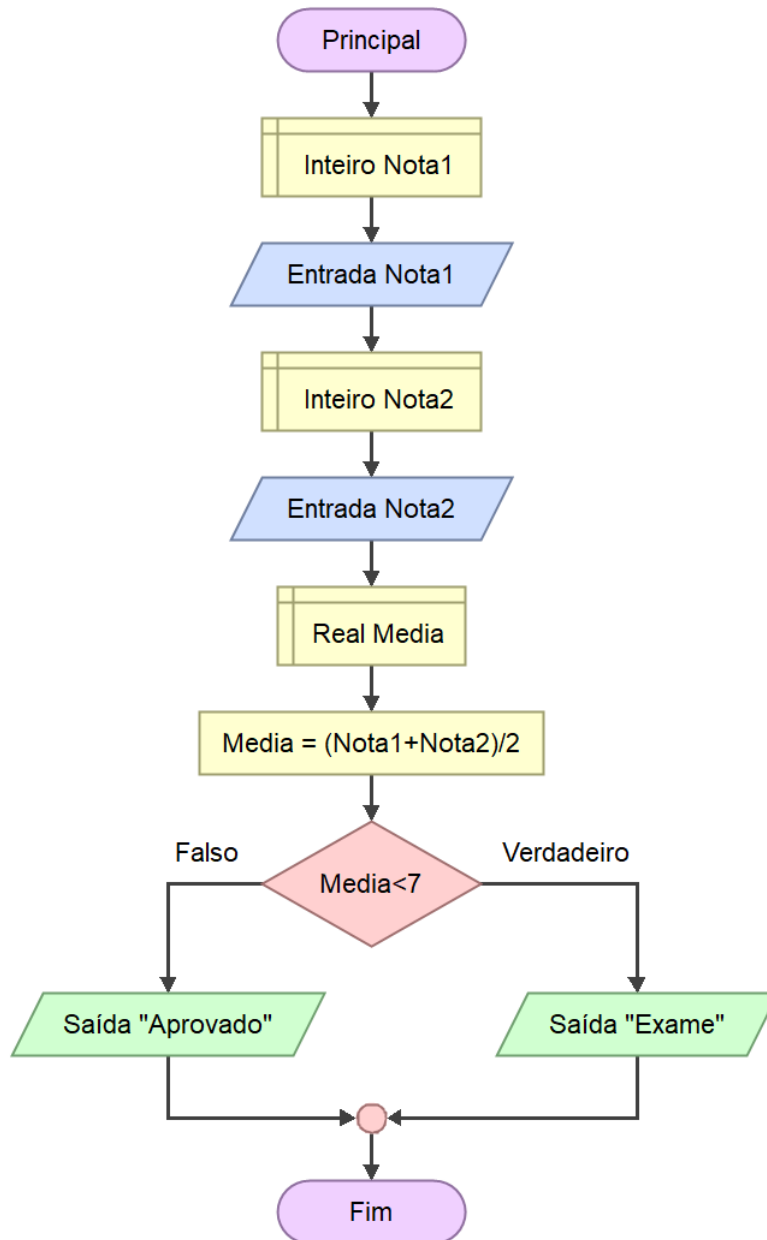
A tabela a seguir apresenta alguns dos principais símbolos usados nos fluxogramas

	Início ou fim
	Entrada ou saída
	Definição
	Atribuição
	Tomada de decisão

Para compreender melhor como funcionam os algoritmos observe o exemplo na página a seguir.

Neste exemplo temos um algoritmo para calcular a média e informar o operador se o aluno está ou não em exame.

Observando o algoritmo notamos que as setas indicam o sentido dos acontecimentos. Inicialmente é realizada a definição das variáveis e a entrada das notas, na sequência é calculada a média e realizado um teste para verificar se a media é menor que 7, se for é apresentada a mensagem “Exame”, se não é apresentada a mensagem “Aprovado”, e o fluxograma chega ao fim.



3.7 Conclusão

A elaboração de programas de computador é a arte de encontrar uma sequência de comandos, capaz de executar a tarefa desejada. Esta sequência de comandos pode ser representada através de algoritmos e fluxogramas, o que facilita seu desenvolvimento e sua compreensão.

As próximas aulas utilizarão os algoritmos e os fluxogramas para explicar os fundamentos da programação.

AULA 4 - FLUXOGRAMAS PARTE 1

Noções sobre a ferramenta Flowgorithm para construção de fluxogramas

4.1 Objetivo:

O objetivo desta aula é exercitar a lógica de programação através do desenvolvimento e simulação de fluxogramas com a ferramenta Flowgorithm.

4.2 Introdução

Para que se possa desenvolver e simular fluxogramas é necessário primeiramente conhecer alguns conceitos fundamentais da programação de computadores. Dentre estes conceitos pode-se destacar o armazenamento de dados através de variáveis, a tomada de decisões através das estruturas de controle, a entrada e saída de dados entre outros.

4.2.1 Variáveis e atribuição de valores

Na programação é normalmente necessário realizar o armazenamento de informações na memória. Estas informações são armazenadas em estruturas chamadas variáveis. Uma variável é uma localizada na memória capaz de armazenar um valor ou expressão. No desenvolvimento dos programas, cada variável é associada a um nome, facilitando assim sua manipulação.

Podemos imaginar uma variável como uma caixa que pode armazenar determinado tipo de informação. Cada uma destas caixas possui um nome para sua identificação e manipulação.

Na computação, quando falamos de variáveis, estamos tratando de uma região da memória previamente identificada cuja finalidade é armazenar os dados ou informações de um programa por um determinado espaço de tempo. A memória do computador se organiza tal qual um armário com várias divisões. Sendo cada divisão identificada por um endereço diferente em uma linguagem que o computador entende.

O computador armazena os dados nessas divisões, sendo que em cada divisão só é possível armazenar um dado e toda vez que o computador armazenar um dado em uma dessas divisões, o dado que antes estava armazenado é eliminado. O conteúdo pode ser alterado, mas somente um dado por vez pode ser armazenado naquela divisão. O processo de armazenar dados em uma variável é chamado de atribuição.

Durante o desenvolvimento de um programa ou algoritmo é necessário definir as variáveis que serão utilizadas. Cada linguagem de programação possui uma forma própria de executar esta tarefa, mas normalmente o procedimento consiste em definir o nome da variável e o tipo de informação que ela deve armazenar.

A tabela a seguir apresenta de forma genérica os principais tipos de dados e qual o tipo de informação eles podem armazenar.

Tipo de dado	Utilização
Booleano	Armazena dois valores, 1 e 0 (verdadeiro ou falso, em inglês <i>True</i> ou <i>False</i>)
Inteiro	Armazena números inteiros (números sem casas decimais)
Real	Armazena números reais (números com casas decimais)
Caractere	Armazena um caractere (uma letra ou símbolo)

A partir destes tipos básicos de dados as linguagens de programação costumam construir inúmeros tipos de dados mais complexos.

4.2.2 Entradas e saídas

Na computação, entradas e saídas são a forma que um programa usa para se comunicar com o mundo exterior. Em um computador tradicional o teclado, por exemplo, é uma entrada e o monitor é uma saída. Em um sistema microcontrolado de automação, por exemplo, os sensores são entradas e os atuadores saídas.

4.3 Fluxogramas com o software Flowgorithm

A utilização de fluxogramas é importante para desenvolvimento e entendimento dos algoritmos e programas de computador. Existem diversas formas de desenvolver fluxogramas, porém a utilização de ferramentas computacionais facilita muito este processo. Uma ferramenta gratuita desenvolvida para este fim é o **Flowgorithm**.

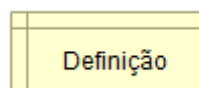
O Flowgorithm é um programa de computador que permite desenvolver e simular fluxogramas de forma rápida e fácil. Esta ferramenta pode ser obtida gratuitamente na internet, no endereço: <http://www.flowgorithm.org/>. A seguir se apresentados os principais fundamentos da utilização desta ferramenta. As explicações a seguir foram fundamentadas na versão 2.16 da ferramenta, com a versão da interface ajustada para português do Brasil.

4.3.1 Variáveis e no Flowgorithm

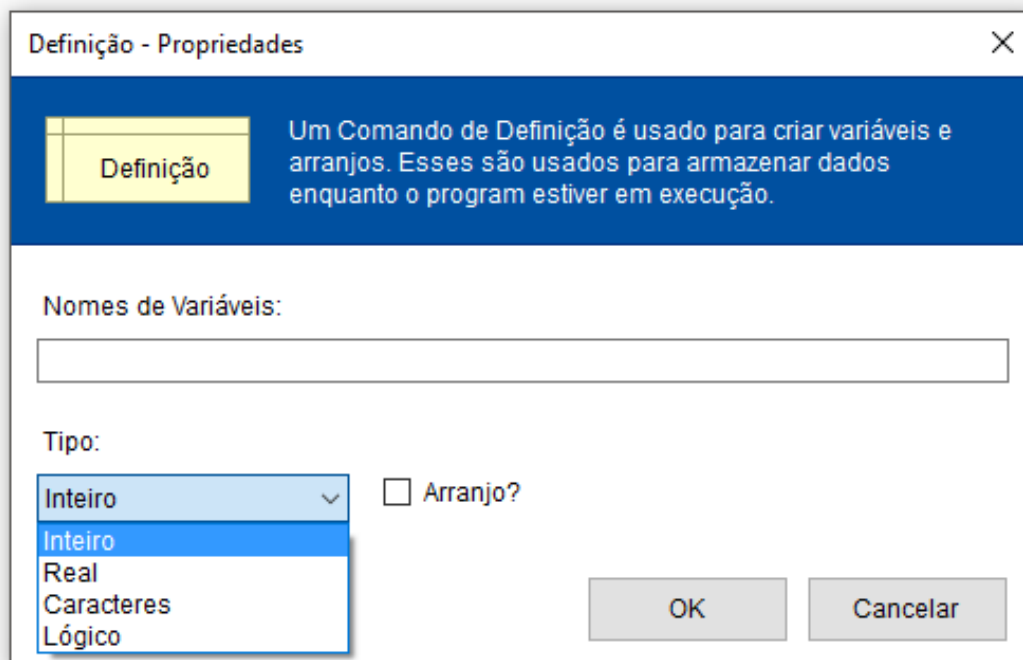
Assim como em qualquer linguagem de programação, no Flowgorithm é possível criar variáveis. A tabela a seguir apresenta os tipos de dados suportados pelas variáveis no Flowgorithm.

Tipo de dado	Utilização
Lógico	Armazena dois valores, 1 e 0 (verdadeiro ou falso, em inglês <i>True</i> ou <i>False</i>)
Inteiro	Armazena números inteiros (números sem casas decimais)
Real	Armazena números reais (números com casas decimais)
Caracteres	Armazena uma cadeia caracteres (texto)

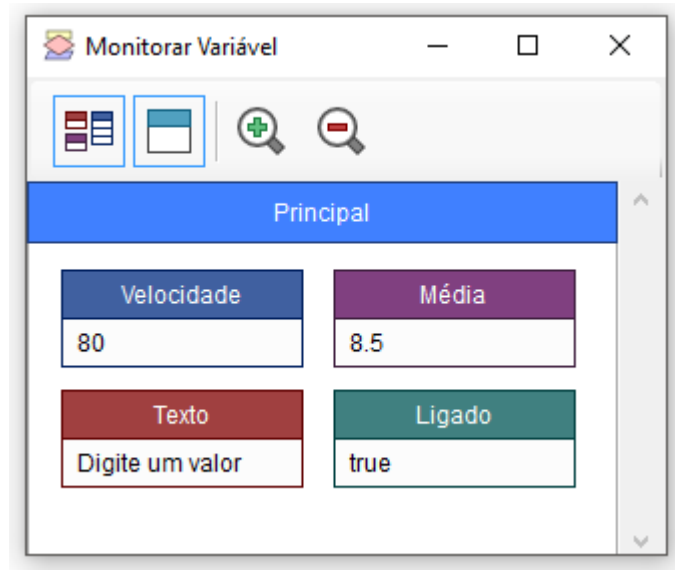
A criação de variáveis é realizada no Flowgorithm através do bloco “Definição”. A figura a seguir apresenta este bloco.



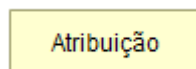
Após adicionar o bloco de definição de variável ao fluxograma é necessário atribuir um nome e um tipo de dado a variável. Para realizar esta tarefa deve-se clicar duas vezes sobre o bloco e preencher os dados da variável. Veja a figura a seguir.



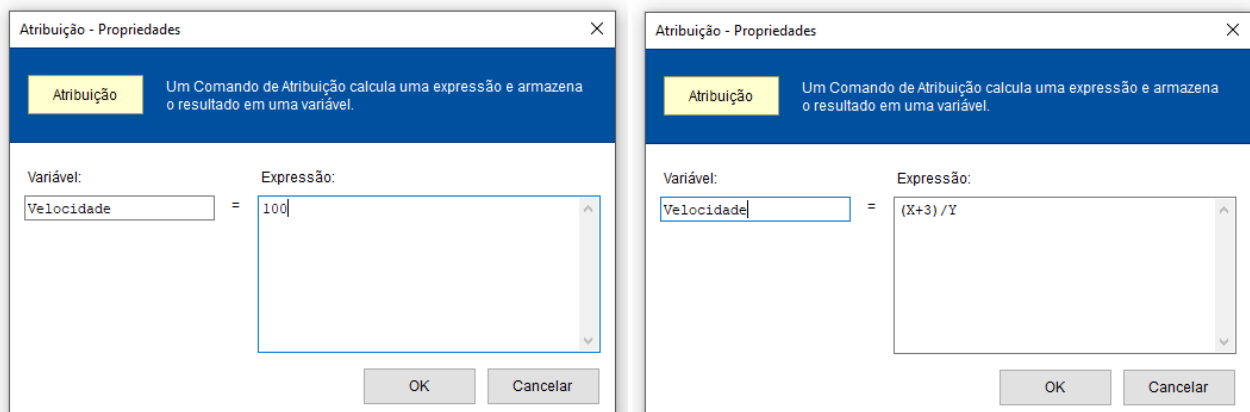
No Flowgorithm também é possível observar o conteúdo e o nome das variáveis com a função “Monitorar Variável”, Veja a figura a seguir,



Após a definição de uma variável é possível atribuir valores a mesma. A figura a seguir apresenta o bloco “Atribuição” que realiza esta tarefa.



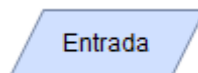
Valores podem ser atribuídos a variáveis de duas formas. Através da atribuição direta dos valores ou através de expressões. A figura a seguir apresenta alguns exemplos.



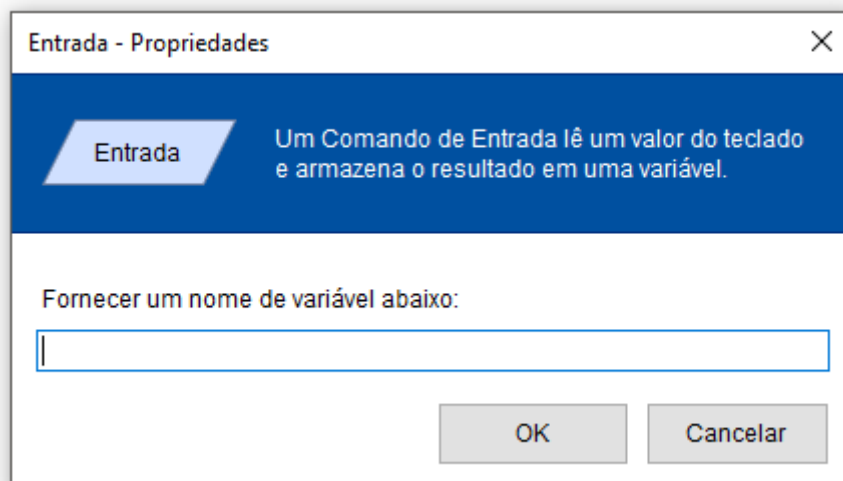
No Flowgorithm as operações são todas realizadas nas atribuições, assim o resultado de qualquer computação é atribuído a uma variável.

4.4 Entradas e saídas no Flowgorithm

O Flowgorithm foi desenvolvido de forma a permitir a entrada de dados através do teclado do computador. O bloco utilizado para realizar a entrada de informações em um fluxograma do Flowgorithm é o bloco “Entrada”. A figura a seguir apresenta este bloco.

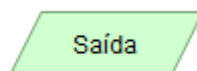


Quando um bloco de entrada é adicionado a um fluxograma é necessário parametrizá-lo. Para isso deve-se clicar duas vezes sobre o bloco. Basta então digitar o nome da variável que vai receber os dados de entrada no campo apropriado. A figura a seguir apresenta a janela de parametrização do bloco de entrada.



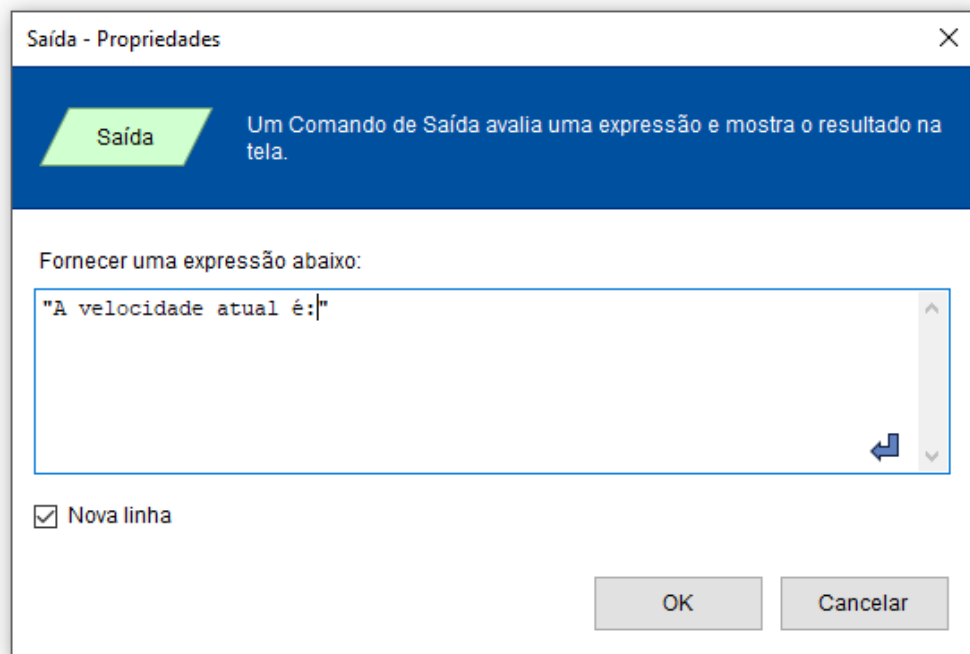
É importante lembrar que a variável deve já ter sido definida anteriormente.

As saídas por sua vez são realizadas nos fluxogramas do Flowgorithm através do bloco “Saída”. A figura a seguir apresenta este bloco.

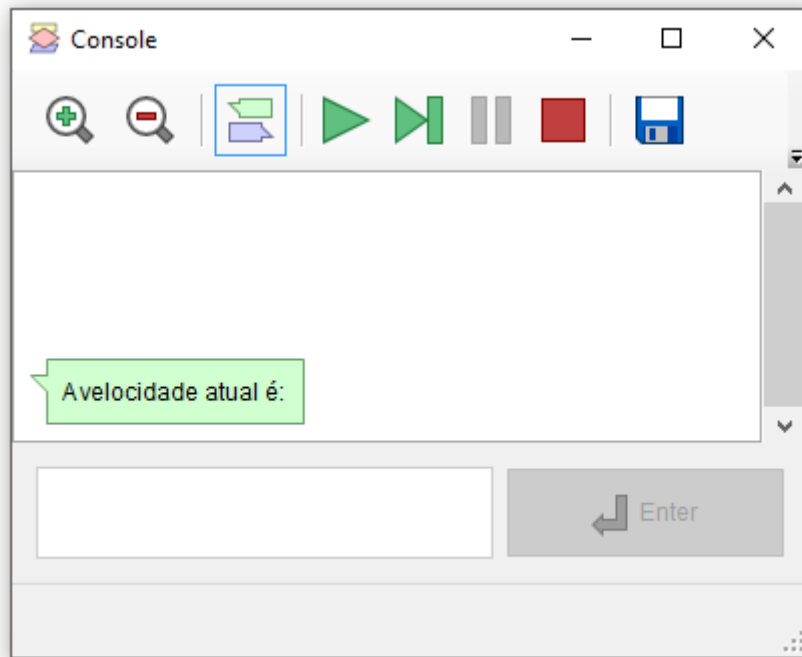


Assim como no caso das entradas este bloco deve ser parametrizado clicando-se duas vezes sobre ele. As saídas podem apresenta na tela do computador dois tipos de informação. Podem ser apresentados textos simples, ou o conteúdo das variáveis.

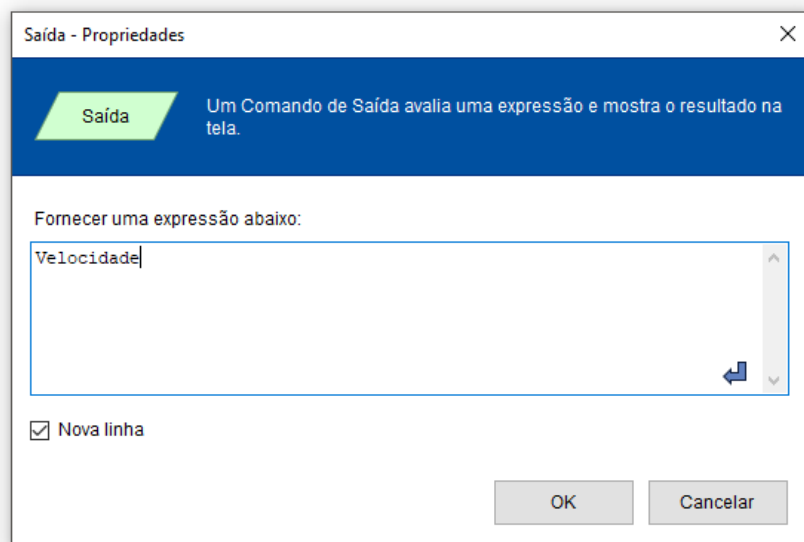
Para apresentar mensagens de texto na tela do computador o bloco de saída deve ser parametrizado com o texto da mensagem entre aspas. Veja um exemplo na figura a seguir.



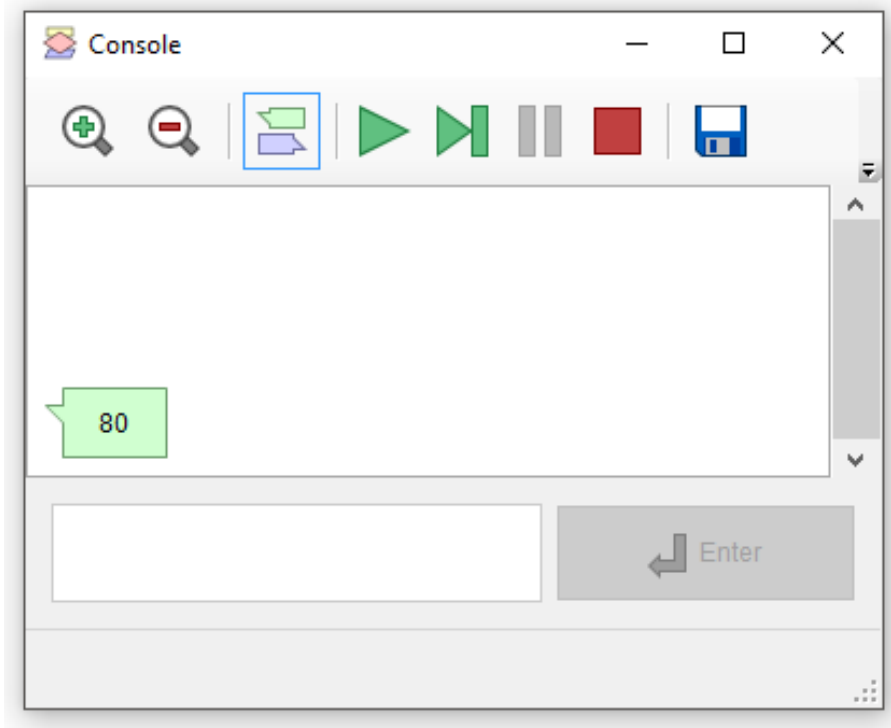
As saídas apresentadas pelos fluxogramas do Flowgorithm são apresentadas em uma janela específica chamada “Console”. O resultado do bloco de saída da figura anterior é apresentado na figura a seguir.



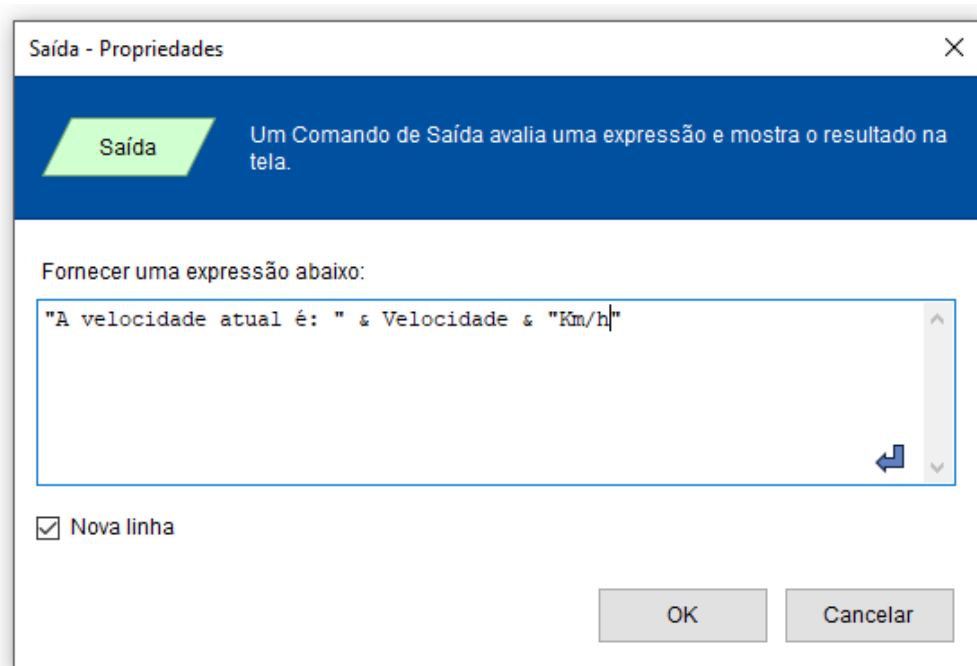
O conteúdo das variáveis pode ser apresentado na saída da mesma forma, basta editar o bloco de saída e parametrizá-lo com o nome da variável que se deseja apresentar. A figura a seguir apresenta a parametrização de um bloco de saída para apresentar o conteúdo da variável Velocidade. Observe que não são usadas aspas no nome da variável.



O resultado na saída para o bloco parametrizado na figura anterior pode ser observado na figura a seguir.

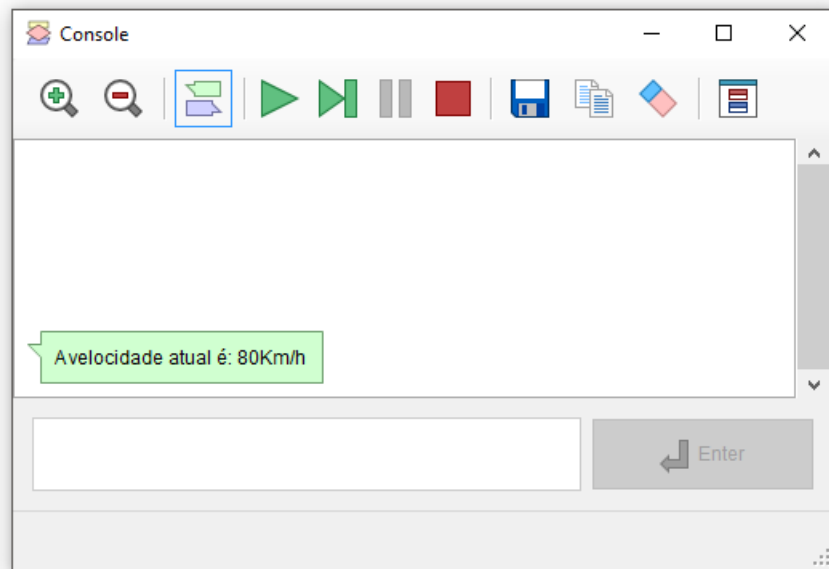


O Flowgorithm permite ainda que se apresenta na saída informações mais complexas, compostas ao mesmo tempo por texto e por conteúdo de variáveis. A figura a seguir apresenta um exemplo deste tipo.

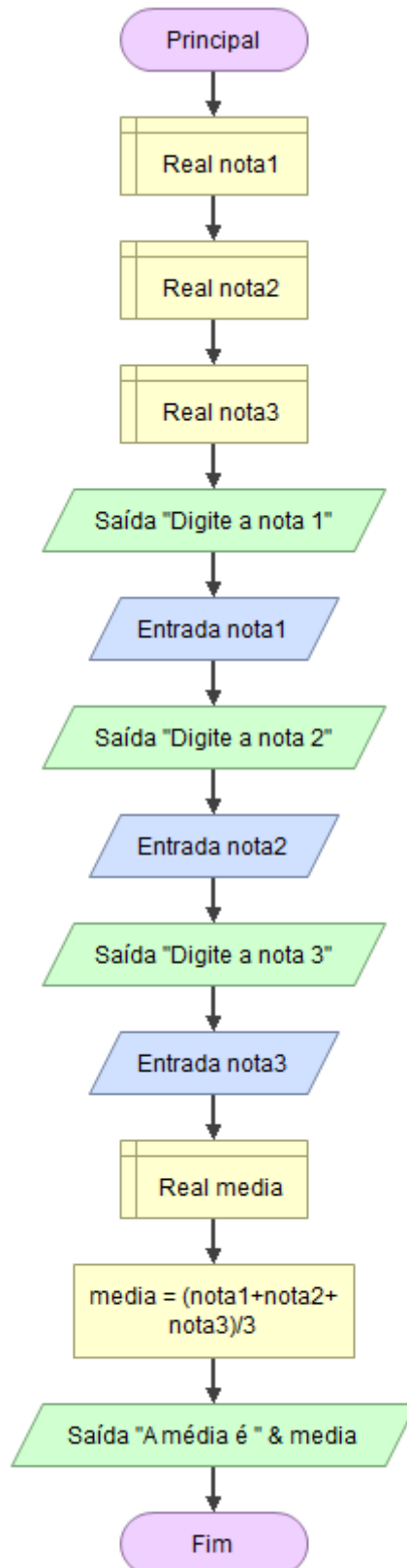


Observem com atenção a forma como o bloco foi parametrizado. Foram usados, ao mesmo tempo blocos de texto e nomes de variáveis, separados pelo caractere &.

A saída para o bloco parametrizado na figura anterior é apresentada na figura a seguir.



A seguir é apresentado um exemplo de algoritmo onde é calculada a média de quatro números digitados pelo usuário.



4.5 Conclusão

Esta aula apresentou uma introdução básica a construção e simulação de fluxogramas com a ferramenta Flowgorithm. Foram apresentados os blocos de definição de variáveis e atribuição de valores a estas variáveis, também foram apresentados os blocos de entrada e saída. O assunto de desenvolvimento e simulação de algoritmos com o Flowgorithm continuará na próxima aula com o estudo com blocos de controle e de repetição.

4.6 Lista de exercícios

- 1) Construa um fluxograma no Flowgorithm que pede para o operador digitar 5 números reais. O fluxograma deve apresentar na tela a soma e o produto destes 5 números.
- 2) Construa um fluxograma no Flowgorithm que pede para o operador digitar o seu nome. O fluxograma deve apresentar na tela a frase: “O nome do operador é” e completar com o nome digitado.
- 3) Construa um fluxograma no Flowgorithm que pergunta um valor em metros e imprime o correspondente em decímetros, centímetros e milímetros.

AULA 5 - FLUXOGRAMAS PARTE 2

Noções sobre a ferramenta Flowgorithm para construção de fluxogramas

5.1 Objetivo:

O objetivo desta aula é continuar a exercitar a lógica de programação através do desenvolvimento e simulação de fluxogramas com a ferramenta Flowgorithm. Serão abordados os blocos de controle e repetição e as subrotinas.

5.2 Introdução

Além dos blocos responsáveis de definição de variáveis, atribuição de valores a variáveis, entrada e saída vistos na aula passada os fluxogramas apresentam também alguns outros blocos importantes. Serão estudados a seguir os blocos de controle e os blocos de repetição disponíveis na ferramenta Flowgorithm.

5.2.1 Estruturas de controle

No desenvolvimento de programas e algoritmos é comum a necessidade de tomar decisões. As decisões estão relacionadas ao processo de verificar se determinada condição é atendida, isso através de um operador de comparação. A tabela a seguir apresenta os principais operadores utilizados na programação.

Operador	Descrição
Igual	Verifica se o conteúdo de duas variáveis ou de uma variável e uma constante é igual e retorna verdadeiro se sim.
Diferente	Verifica se o conteúdo de duas variáveis ou de uma variável e uma constante é diferente e retorna verdadeiro se sim.
Maior	Verifica se o conteúdo de uma variável é maior que o conteúdo de outra variável ou constante e retorna verdadeiro se sim.
Maior ou igual	Verifica se o conteúdo de uma variável é maior ou igual que o conteúdo de outra variável ou constante e retorna verdadeiro se sim.
Menor	Verifica se o conteúdo de uma variável é menor que o conteúdo de outra variável ou constante e retorna verdadeiro se sim.
Menor ou igual	Verifica se o conteúdo de uma variável é menor ou igual que o conteúdo de outra variável ou constante e retorna verdadeiro se sim.

As tomadas de decisão em um algoritmo podem acontecer de duas formas. Em uma tomada de decisão mais simples, um trecho do programa pode ser executado apenas se a condição de teste for verdadeira. Em uma tomada de decisão mais complexa são utilizados dois trechos diferentes do programa, se a condição de teste for verdadeira um dos trechos é executado e se for falsa o outro trecho é executado.

5.2.2 Estruturas de repetição

Nas estruturas de repetição também são tomadas decisões com base em testes sobre variáveis. Porém neste tipo de estrutura os trechos de código costumam ser repetidos um determinado número de vezes. O número de vezes que um trecho de programa deve ser repetido é determinado pelas condições de teste envolvidas no laço de repetição.

5.3 Estruturas de controle no Flowgorithm

No Flowgorithm a tomada de decisões é realizada por um bloco chamado “Alternativa”. A figura a seguir apresenta este bloco.

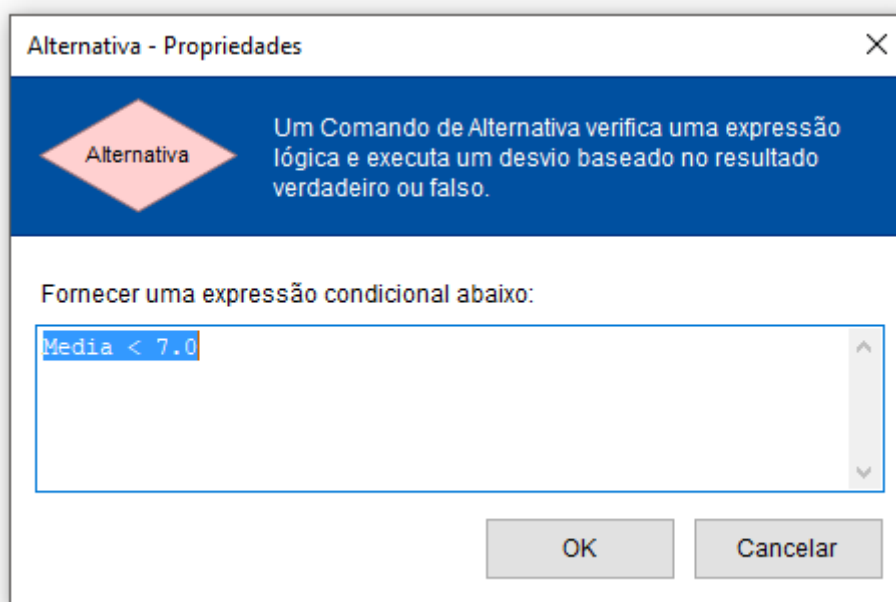


Este bloco deve ser parametrizado com a condição de teste que se deseja verificar. A tabela a seguir apresenta as condições de teste suportadas pelo Flowgorithm e sua grafia. É importante lembrar que os testes são realizados sobre o conteúdo de variáveis ou constantes.

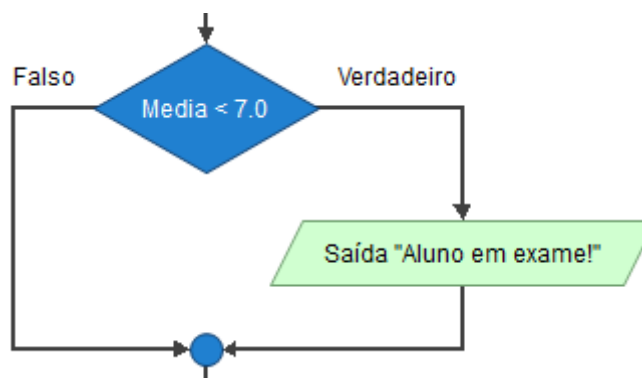
Operador	Grafia	Descrição
Igual	==	Verifica se o conteúdo de duas variáveis ou de uma variável e uma constante é igual e retorna verdadeiro se sim.
Diferente	!=	Verifica se o conteúdo de duas variáveis ou de uma variável e uma constante é diferente e retorna verdadeiro se sim.
Maior	>	Verifica se o conteúdo de uma variável é maior que o conteúdo de outra variável ou constante e retorna verdadeiro se sim.
Maior ou igual	>=	Verifica se o conteúdo de uma variável é maior ou igual que o

		conteúdo de outra variável ou constante e retorna verdadeiro se sim.
Menor	<	Verifica se o conteúdo de uma variável é menor que o conteúdo de outra variável ou constante e retorna verdadeiro se sim.
Menor ou igual	<=	Verifica se o conteúdo de uma variável é menor ou igual que o conteúdo de outra variável ou constante e retorna verdadeiro se sim.

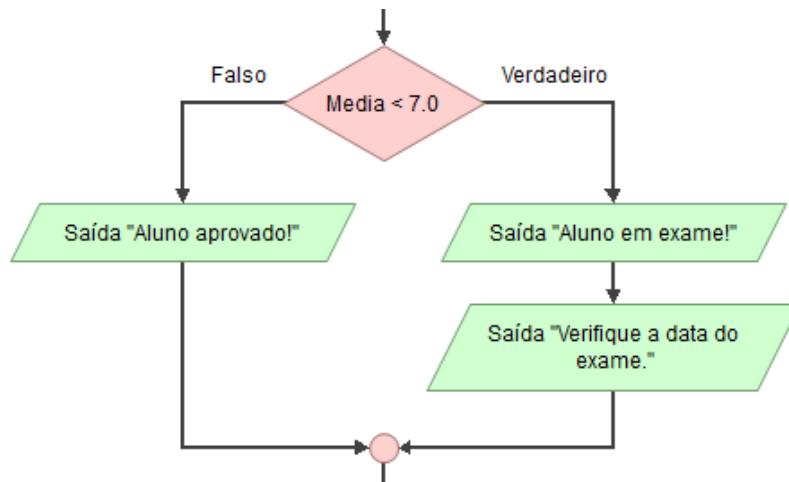
A figura a seguir mostra a janela de parametrização de um bloco “Alternativa”, sendo parametrizado com o teste “Média menor que 7.0”



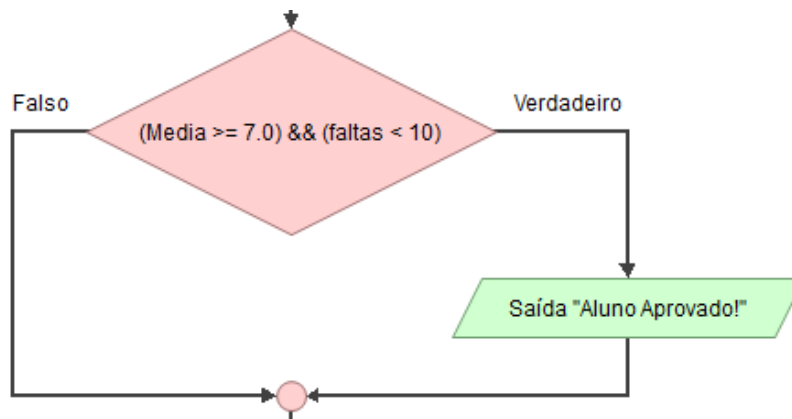
A figura a seguir apresenta como ficou este bloco parametrizado inserido no fluxograma. Nesta estrutura se a média for menor que 7.0 é apresentado na saída o texto “Aluno em exame!”, mas se a média for maior ou igual a 7.0 o programa segue sem apresentar a mensagem.



No flowgorithm as tomadas de decisão podem também realizar tarefas mais complexas, veja o exemplo da figura a seguir.



Em algumas situações é necessário construir estruturas de decisão mais complexas. Para este tipo de decisão é possível associar vários testes com os comandos OU (||) e E (&&). Veja o exemplo da figura a seguir onde para que a condição seja verdadeira a média tem que ser maior ou igual a 7.0 e o número de faltas tem que ser menor que 10.

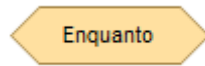


5.4 Estruturas de repetição no Flowgorithm

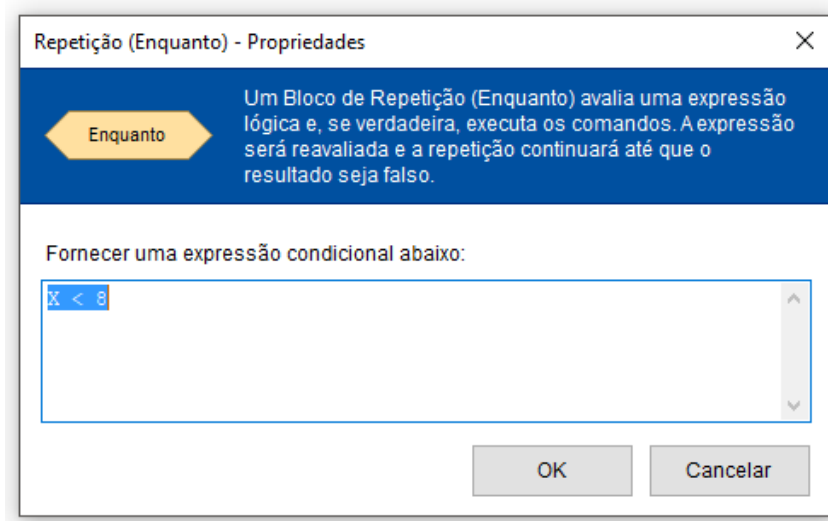
O Flowgorithm possui três tipos de estruturas de controle. O “Enquanto”, o “Para” e o “Fazer”. As seções a seguir detalham cada um deles.

5.4.1 O laço “Enquanto” no Flowgorithm

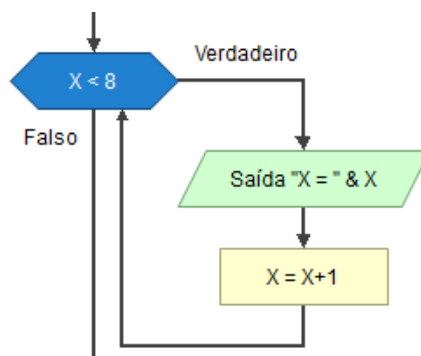
O laço “Enquanto” é utilizado nos fluxogramas do Flowgorithm para repetir um trecho do programa enquanto uma condição de teste é verdadeira. A figura a seguir apresenta o bloco “Enquanto” do Flowgorithm.



Assim como acontece com os outros blocos do Flowgorithm, o bloco “Enquanto” necessita ser parametrizado. Para sua parametrização deve-se adicionar, assim como no bloco “Alternativa”, uma condição de teste. A figura a seguir mostra como parametrizar um bloco “Enquanto” para ficar repetido um trecho do programa enquanto o conteúdo da variável X é menor que 8.



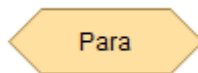
A figura a seguir mostra um trecho de fluxograma onde este bloco é utilizado.



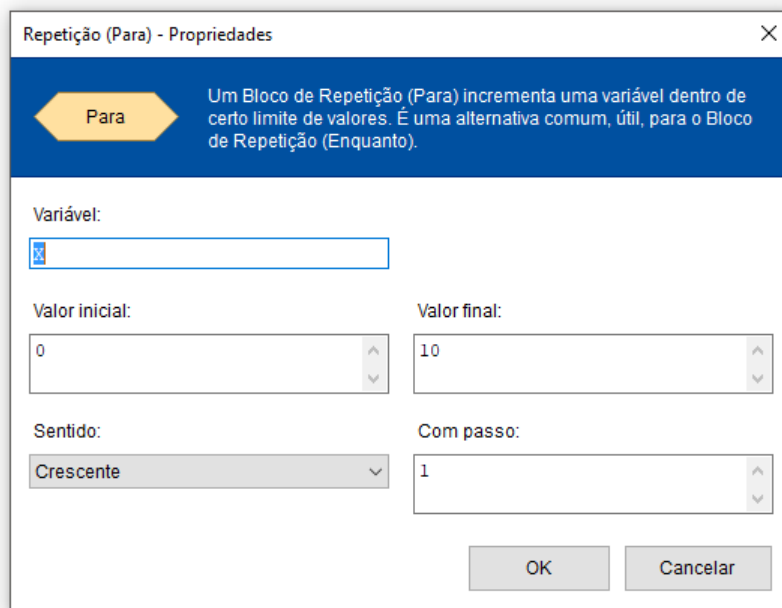
Supondo que a variável X inicia com valor 0, neste bloco “Enquanto” o trecho de programa é repetido até que X deixe de ser menor que 8. Como a variável X é incrementada a cada iteração do laço, na saída serão apresentados os valores de 0 a 7.

5.4.2 O laço “Para” no Flowgorithm

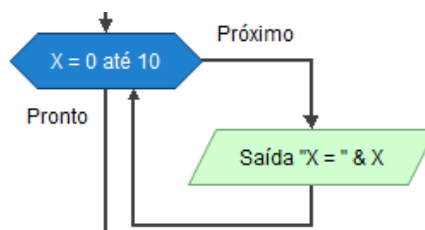
O laço “Para” é parecido com o laço “Enquanto”, porém um pouco mais complexo. Nele é necessário definir uma variável que será utilizada no controle do laço, um valor inicial para esta variável, um sentido de contagem e um incremento. A figura a seguir mostra o bloco “Para”.



Para a parametrização deste bloco todos os elementos necessários devem ser descritos. A figura a seguir mostra a janela de parametrização do bloco “Para”.



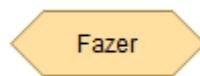
No exemplo da figura a variável utilizada é X, que deve contar no sentido crescente de 0 a 10. Vela um exemplo de utilização do bloco “Para”.



Neste exemplo a variável X inicia com 0 e sendo incrementada de 1 em 1 até 10. Estes valores são apresentados na tela.

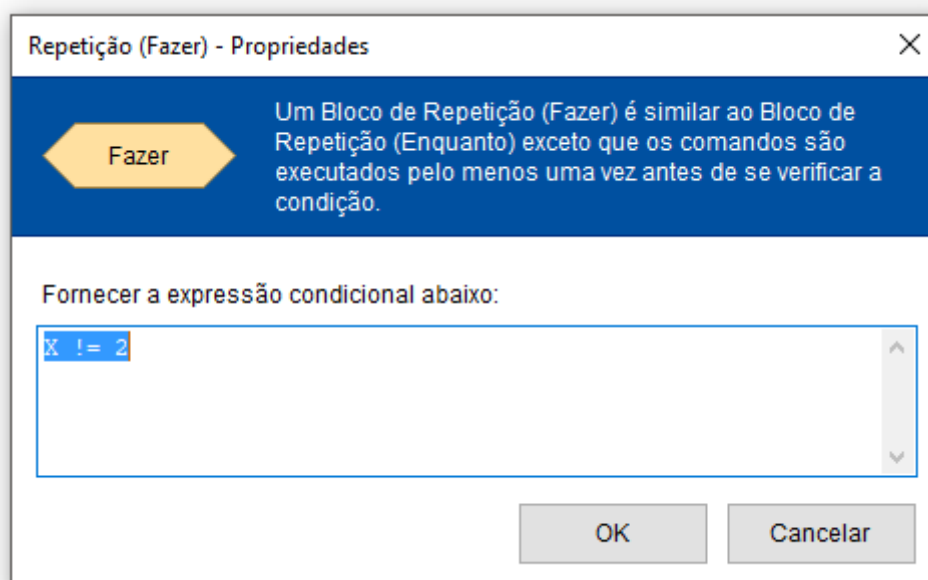
5.4.3 O laço “Fazer” no Flowgorithm

O laço “Fazer” é muito parecido com o laço “Enquanto”, porém, com uma diferença muito importante. No laço “Enquanto” a condição é verificada logo na entrada, e assim se a condição é falsa o laço não é executado nenhuma vez. Já no laço “Fazer” a condição é verificada na saída, e mesmo se a condição for inicialmente falsa, o laço é executado pelo menos uma vez. A imagem a seguir apresenta o bloco “Fazer”.

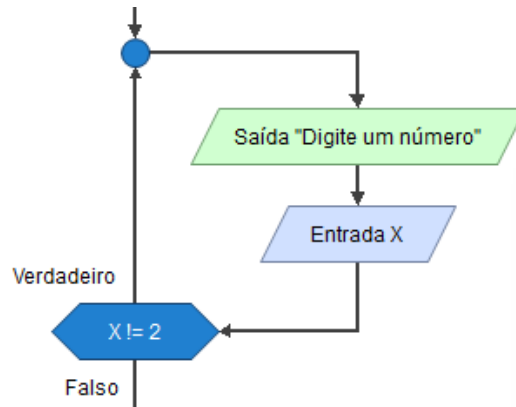


Assim como acontece com os outros blocos do Flowgorithm, o bloco “Fazer” necessita ser parametrizado. Para sua parametrização deve-se adicionar, assim como nos blocos anteriores, uma condição de teste.

A figura a seguir mostra como parametrizar um bloco “Fazer” para ficar repetido um trecho do programa enquanto o conteúdo da variável X é diferente de 2.



A figura a seguir mostra um trecho de fluxograma onde este bloco é utilizado.



Neste bloco “Fazer” o trecho de programa é repetido até que X fique igual a 2. O trecho de programa no interior do laço pede para o operador digitar um número, este número é armazenado em X e se o número não for 2 o processo se repete.

5.5 Conclusão

Esta aula apresentou a segunda parte da introdução a construção e simulação de fluxogramas com a ferramenta Flowgorithm. Foram apresentados os blocos de controle e de repetição. Maiores informações sobre os blocos do Flowgorithm podem ser obtidas em <http://www.flowgorithm.org/documentation/index.htm>

5.6 Lista de exercícios

- 1) Construa um fluxograma no Flowgorithm que lê o preço de um produto e inflaciona esse preço em 10% se ele for menor que 100 e em 20% se ele for maior ou igual a 100.
- 2) Construa um fluxograma no Flowgorithm que funciona como uma calculadora. Ele deve pedir qual a operação desejada (+, -, * ou /) e na sequência deve pedir os dois operandos e realizar a operação. O fluxograma deve ficar se repetindo até que o operador digite uma operação diferente das quatro operações desejadas.
- 3) Construa um fluxograma no Flowgorithm que e imprime uma tabela com a tabuada de 1 a 9.
- 4) Construa um fluxograma no Flowgorithm que lê dois valores e imprime:
 - se o primeiro valor for menor que o segundo, a lista de valores do primeiro até o segundo;
 - se o primeiro valor for maior que o segundo a lista de valores do segundo até o primeiro em ordem decrescente;
 - se ambos forem iguais a mensagem "valores iguais".

AULA 6 - INTRODUÇÃO AO ARDUINO

Fundamentos da plataforma Arduino e de seu ambiente de programação.

6.1 Objetivo:

O objetivo desta aula é mostrar aos alunos o que é e como funciona o Arduino e seus principais componentes, enfatizando os aspectos do *hardware*. O microcontrolador utilizado no Arduino é também apresentado nesta aula, com uma especial atenção a seus componentes internos.

6.2 Introdução

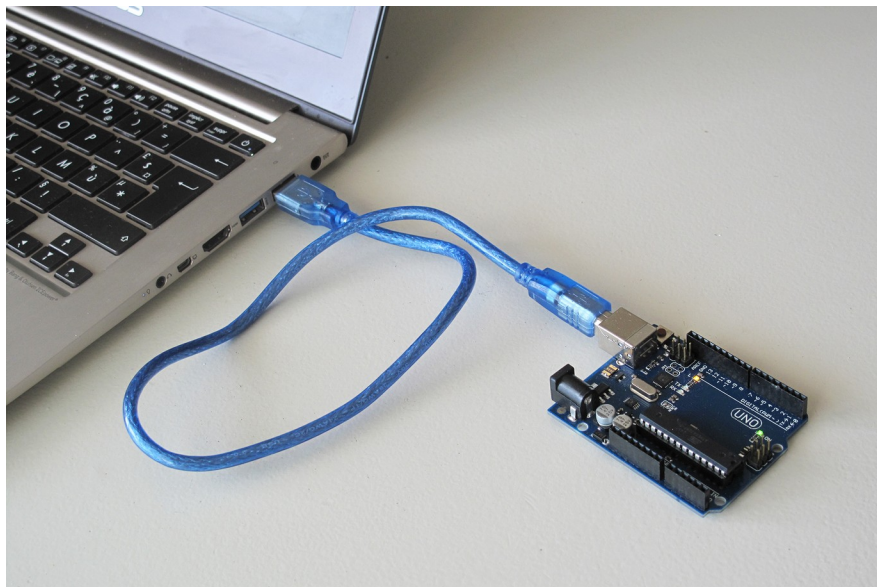
Arduino é uma plataforma de hardware livre, que foi projetada para prototipação. Com base em um microcontrolador Atmel AVR o Arduino possui uma linguagem de programação padrão baseada em C/C++. Por se tratar de um projeto aberto e livre ele possui um conjunto de ferramentas que são acessíveis, com baixo custo, flexíveis e fáceis de se usar. A figura a seguir apresenta um Arduino Uno.



A família de placas que compõe a plataforma Arduino é bastante extensa, assim este material vai concentrar sua atenção no Arduino Uno.

O Arduino é composto principalmente por um microcontrolador, algumas entradas e saídas digitais, algumas entradas e saídas analógicas, além de uma interface serial ou USB que permite interligar o Arduino ao computador.

A programação do Arduino é realizada com a ajuda de um computador. O computador é usado para interagir em tempo real com o Arduino através de comunicação serial ou USB. A figura a seguir mostra esta conexão.

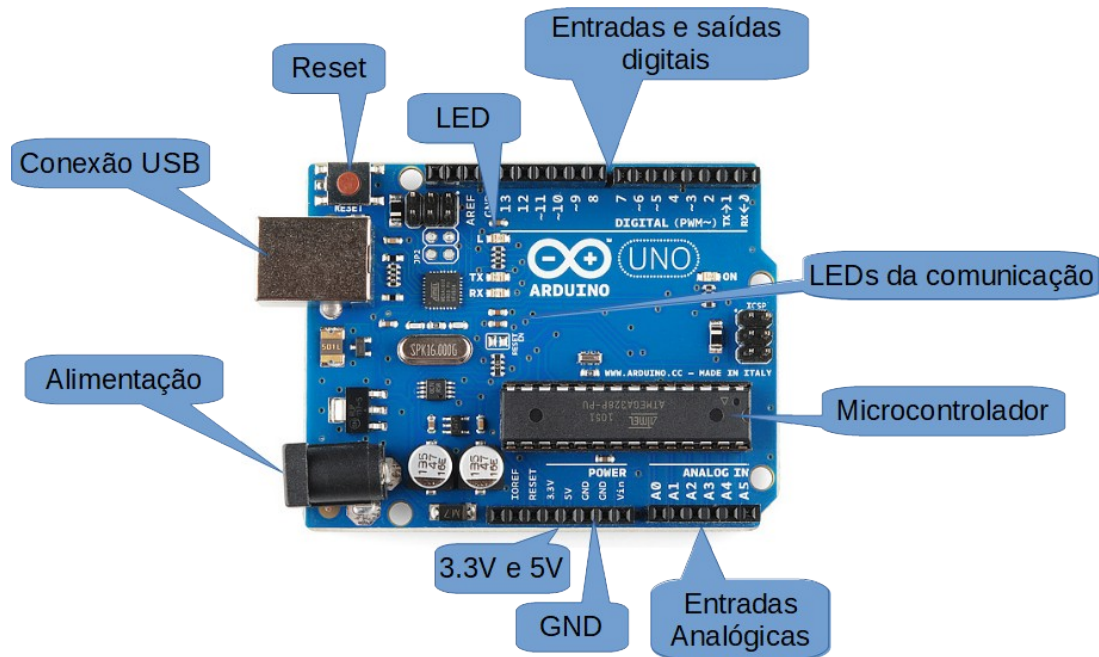


Uma característica interessante da plataforma Arduino é a possibilidade de expansão através de extensões chamadas *shields*. Estes *shields* são placas de circuito que podem ser conectadas ao Arduino e adicionam funcionalidades ao mesmo. A figura a seguir apresenta um *shield* que permite conexão com a rede de computadores ou a internet conectado a um Arduino.



O Arduino é composto por vários componentes, dentre os quais pode-se destacar a porta de comunicação USB, responsável pela comunicação com o computador, o conector de alimentação, as saídas de 3,3V e 5 V, que servem para alimentar periféricos, as entradas analógicas, as antradas e saídas digitais, o sinal de reset, os LED de indicação e o microcontrolador.

A figura a seguir destaca cada um destes componentes.



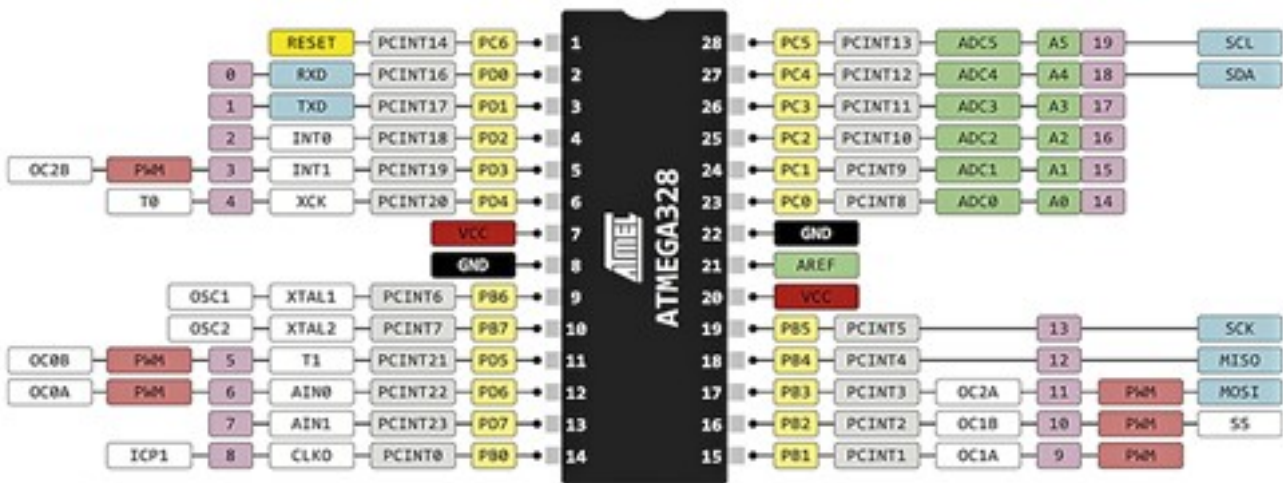
As próximas seções irão tratar mais especificamente de cada um dos componentes do Arduino, bem como de seu ambiente de programação.

6.3 O microcontrolador utilizado no Arduino

Dentre todos os componentes do Arduino o principal é o microcontrolador. O microcontrolador é responsável por todo o processamento do Arduino. Existem vários modelos de Arduino, cada um deles com microcontroladores específicos. O modelo mais comum de Arduino, que é adotado neste material, utiliza um microcontrolador da família AVR da Atmel, o modelo ATmega328. A seguir são apresentadas as principais características deste microcontrolador.

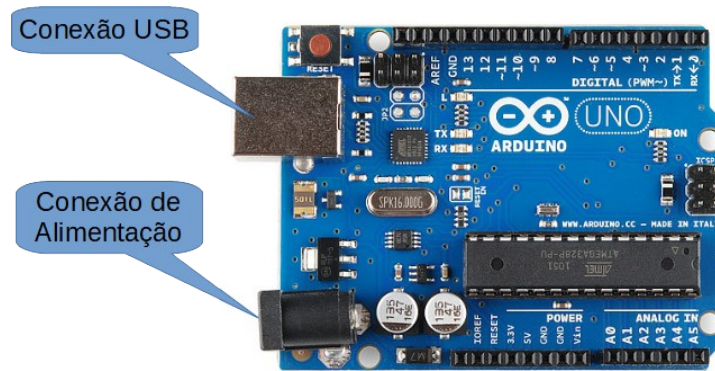
- CPU RISC de 8-Bit
- Capacidade de 20 MIPS em 20MHz
- 32KBytes de memória Flash de programa
- 1KBytes de EEPROM
- 2KBytes de SRAM
- Programável no circuito
- 2 temporizados/contadores de 8 bits
- 1temporizados/contadores de 16 bits
- Contador de tempo real
- 6 canais PWM
- 6 entradas analógicas de 10 bits
- Comunicação UART, SPI e I2C
- 23 entradas ou saídas programáveis
- Operação de 1.8 - 5.5V

A figura a seguir apresenta as funções de cada um dos pinos do microcontrolador ATmega328.



6.4 Alimentação do Arduino

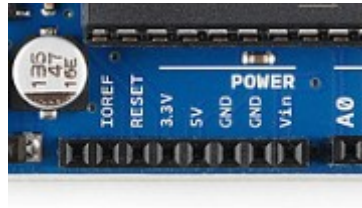
O Arduino pode receber a energia necessária para sua operação através da conexão USB ou através de uma fonte de alimentação externa, conforme exibido na figura abaixo.



Quando a alimentação é realizada pela conexão de alimentação a tensão deve estar entre 7V. a 12V. Neste caso reguladores de tensão presentes no Arduino fazem o ajuste da tensão aos níveis necessários. O consumo total de corrente não deve ultrapassar 500mA.

A comutação da alimentação entre o conector de alimentação e a porta USB é realizada automaticamente pelo Arduino.

O Arduino também fornece conexões de alimentação para circuitos periféricos ou *Shield* como pode ser observado na figura a seguir.



Neste conector o sinal IOREF - Fornece uma tensão de referência para os *shields* e periféricos, assim os *shields* podem reconhecer se a tensão de entrada e saída do arduino é 3,3V ou 5V.

O sinal de RESET é conectado ao reset do microcontrolador e pode ser utilizado para um reset externo do Arduino.

O sinal 3,3 V fornece tensão de 3,3V para os *shields* ou periféricos.

O sinal 5 V fornece tensão de 5V para os *shields* ou periféricos.

Os sinais GND são os pinos de referência ou terra.

O sinal VIN permite alimentar o Arduino através de algum circuito externo, sem a necessidade de utilizar o conector de alimentação.

6.5 Entradas e saídas do Arduino

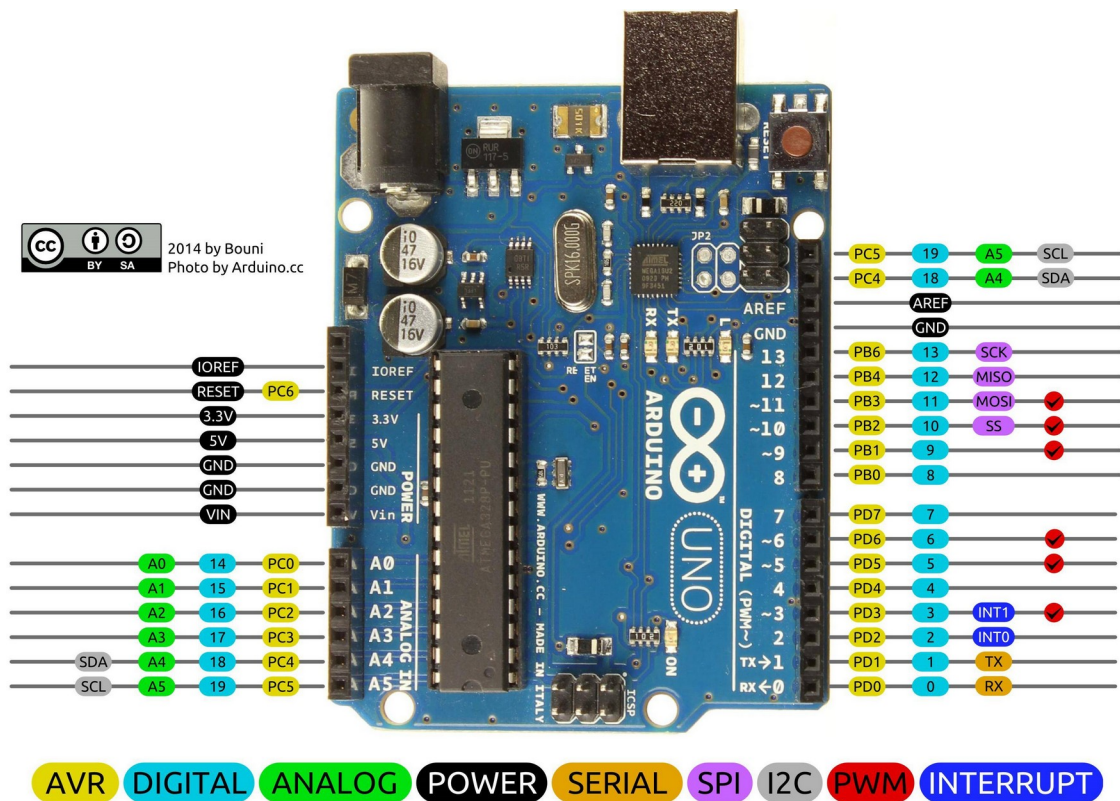
O Arduino permite sinais de entrada e saída digitais bem como sinais de entrada e saída analógicos. O Arduino possui 14 pinos de entrada ou saída digital. Cada um destes pinos pode receber um enviar um sinal digital de 5 V. É possível habilitar por software um resistor interno de *pull-up* para cada um destes pinos. A capacidade máxima de corrente de cada um dos pinos é de 40 mA.

Alguns destes pinos tem também funções especiais, como por exemplo a função PWM, disponível nos pinos 3,5,6,9,10 e 11, a comunicação serial, disponível nos pinos 0 e 1 e as interrupções externas disponíveis nos pinos 2 e 3.

As entradas analógicas, por sua vez, são contempladas no Arduino através de um conversor Analógico/Digital. São disponibilizadas 6 entradas analógicas de 10 bits. Na configuração padrão as entradas analógicas suportam tensões entre 0 e 5V, fornecendo resultados numéricos entre 0 e 1023.

É também possível emular saídas analógicas através das saídas digitais em modo PWM. Neste modo é possível definir uma razão cíclica de 0 a 255 em cada uma das 6 saídas que suportam o modo de operação PWM.

A figura a seguir mostra as estradas e saídas do Arduino com suas principais funções.



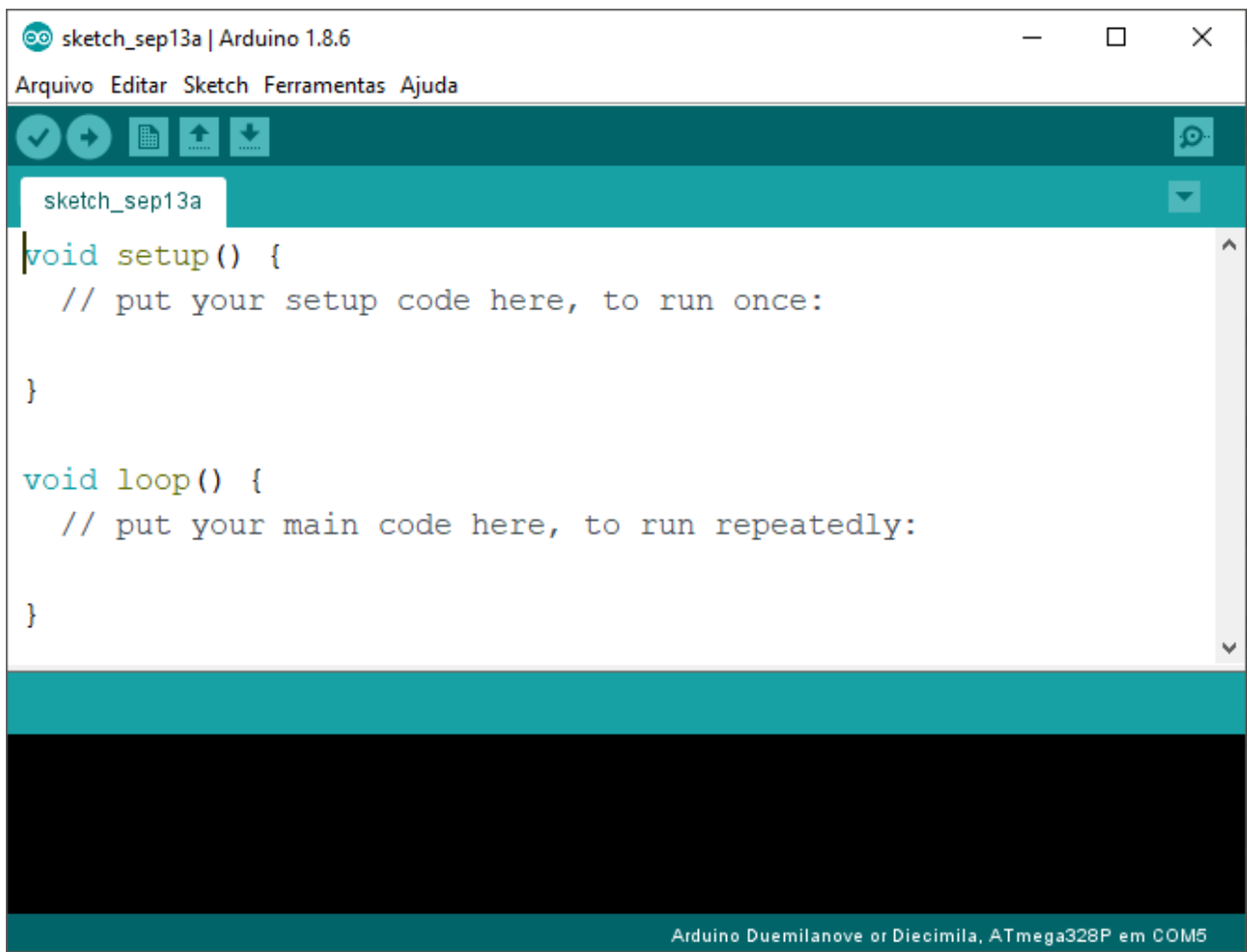
6.6 O ambiente de desenvolvimento do Arduino

Para que se possa desenvolver a programação do Arduino é necessário um ambiente de desenvolvimento apropriado. Este ambiente de desenvolvimento é chamado comumente de IDE (Integrated Development Environment) Arduino. A IDE Arduino pode ser obtida gratuitamente no site do Arduino na internet, cujo endereço é: <https://www.arduino.cc/en/Main/Software>.

Este ambiente de desenvolvimento é um editor de texto onde são editados os programas e onde a compilação dos mesmos é realizada. A IDE Arduino também permite transferir o programa compilado para a placa do Arduino e monitorar seu funcionamento através de comunicação serial, realizada através do cabo USB que conecta o Arduino ao computador.

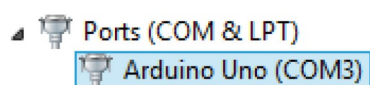
Através desta comunicação serial é possível enviar comandos para o Arduino, na forma de texto obtido do teclado do computador. Também é possível mostrar na tela do computador mensagens de texto enviadas pelo Arduino. É possível ainda plotar na forma de gráficos informações numéricas que o Arduino manda para o computador.

A figura a seguir mostra a interface da IDE Arduino.

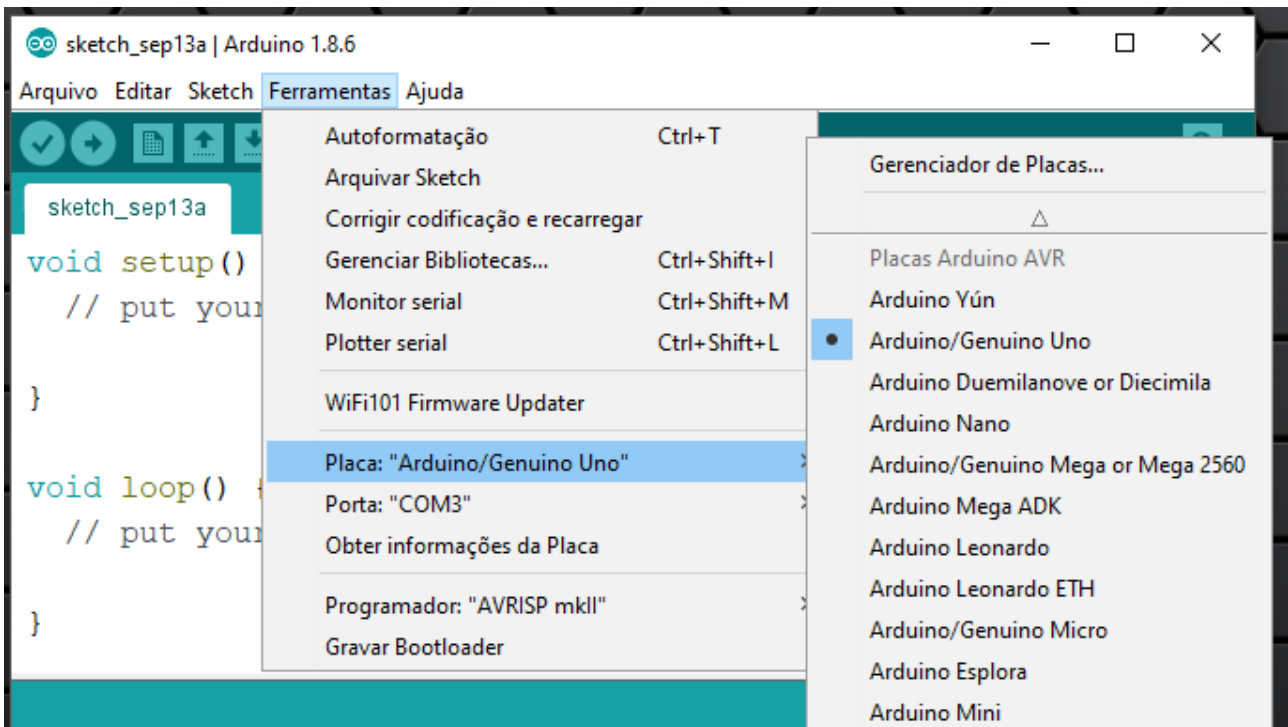


6.7 Utilização da IDE Arduino

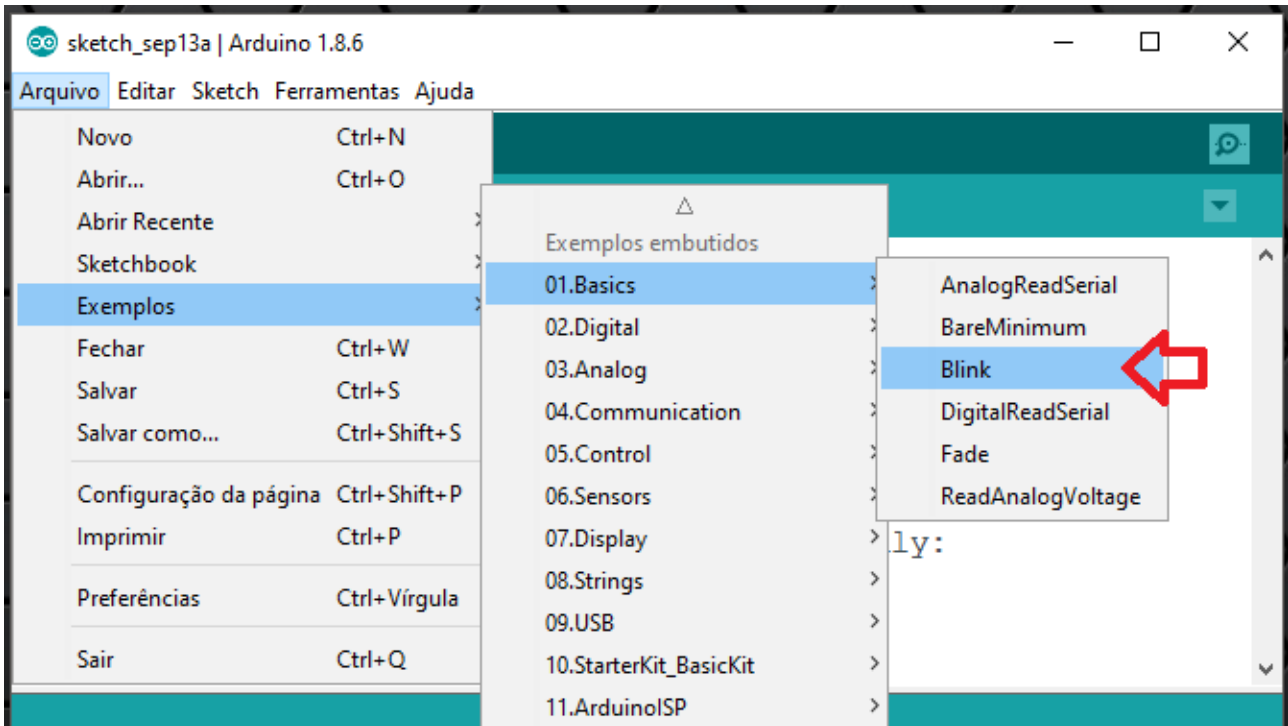
Para que se possa utilizar a IDE Arduino no desenvolvimento de programas para esta plataforma é primeiro necessário configurá-la. Para isso devemos definir qual o modelo de Arduino estamos utilizando e a qual porta do computador ele está conectado. O primeiro passo é conectar o Arduino ao computador, e após o reconhecimento da placa pelo sistema operacional descobrir em qual porta ele se conectou. No caso do Windows basta abrir o “Gerenciados de Dispositivos” e procurar por “Portas (COM e LPT)”. Neste local deve aparecer a porta onde o Arduino está conectado. Veja um exemplo na figura a seguir.



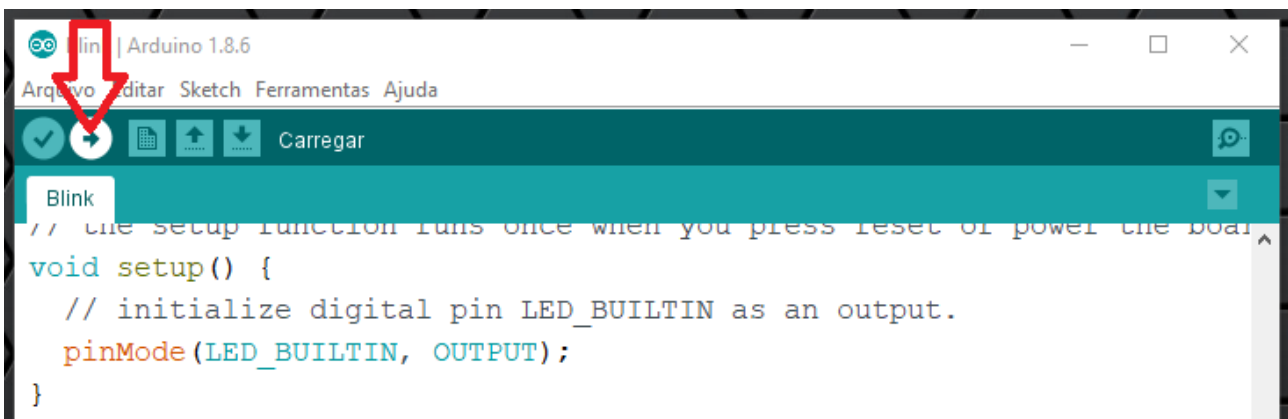
De posse desta informação deve-se acessar o menu “Ferramentas” na IDE Arduino e definir o modelo da placa e a porta de comunicação conforme apresentado na figura a seguir.



Realizada a configuração a IDE Arduino está pronta para ser utilizada. Para verificar seu funcionamento pode-se utilizar um dos exemplos de programas que vem com a IDE. Um exemplo bastante simples e funcional é o “Blink” que serve apenas para piscar um LED na placa do Arduino. A figura a seguir mostra como abrir este exemplo.



O exemplo já está pronto para funcionar no Arduino, assim basta verificar se o Arduino está conectado ao computador e clicar no botão para compilar e carregar o programa na placa. A figura a seguir destaca o botão com esta função.



Se tudo funcionar como esperado o LED da placa do Arduino deve começar a piscar.

6.8 Conclusão

Esta aula apresentou o Arduino a sua plataforma de programação. Maiores informações sobre estes assuntos podem ser obtidas diretamente no site do Arduino, no endereço: <https://www.arduino.cc/>, este site é bem completo e possui muito material importante para a utilização desta plataforma.

Na próxima aula trataremos da programação do Arduino, mais especificamente dos detalhes da linguagem de programação utilizada nesta plataforma.

6.9 Exercício

Como exercício realize o procedimento de configuração da IDE Arduino e execute o exemplo de programa “Blink”, como descrito anteriormente.

AULA 7 - PROGRAMAÇÃO DO ARDUINO

Fundamentos da programação da plataforma Arduino.

7.1 Objetivo:

O objetivo desta aula é apresentar aos alunos os fundamentos da programação da plataforma Arduino. Não são necessários conhecimentos anteriores de linguagens de programação, uma vez que todos os fundamentos necessários serão apresentados neste material.

7.2 Introdução

A linguagem de programação utilizada na plataforma Arduino é baseada em C/C++ e desta forma é fácil encontrar materiais de estudo sobre o assunto, porém existem algumas pequenas diferenças entre as linguagens C e C++ tradicionais e a variante utilizada no Arduino.

Na programação do Arduino os algoritmos são escritos em uma linguagem legível para humanos. Uma das tarefas da IDE Arduino é compilar estes códigos, ou seja, converter o código escrito nesta linguagem legível para humanos em um código legível para o microcontrolador.

Para que se possa desenvolver programas para o Arduino as seções a seguir apresentarão todos os detalhes de sua linguagem de programação.

7.3 Programação do Arduino

Os programas para Arduino são escritos na forma de texto, sendo que cada comando tem uma sintaxe específica que deve obrigatoriamente ser observada pelo programador.

7.3.1 Comentários

A primeira característica da programação do Arduino que iremos observar é a introdução de comentários no programa. Os comentários são importantes para documentar o programa e facilitam a manutenção do mesmo. Na linguagem de programação do Arduino, os comentários de uma linha devem iniciar com "//", tudo que vem após as duas barras é ignorado pelo compilador, veja o exemplo a seguir.

```
void setup() {  
  // exemplo de comentário  
}
```

Também é possível incluir comentários de várias linhas, para isso o trecho de texto deve iniciar com o código "/*" e finalizar com o código "*/", veja o exemplo a seguir,

```
/*  
Exemplo de comentário  
com várias linhas  
*/
```

Tudo que está entre o "/*" e o "*/" é ignorado pelo compilador.

7.3.2 Finalização de instruções com ponto e vírgula

A linguagem de programação do Arduino também exige que todas as instruções sejam finalizadas com ponto e vírgula (;), veja o exemplo a seguir.

```
void setup() {  
  pinMode(LED_BUILTIN, OUTPUT);  
}
```

Neste exemplo a função "pinMode" necessita ser finalizada com ";".

7.3.3 Definição de blocos de código com chaves

No Arduino todos os blocos de código devem ser iniciados e finalizados com chaves ({ }). Isso serve tanto para sub-rotinas como para instruções que integrem blocos de código. Veja o exemplo a seguir.

```
void loop() {  
  if(valor > 5)  
  {  
    digitalWrite(LED_BUILTIN, HIGH);  
    delay(1000);  
    digitalWrite(LED_BUILTIN, LOW);  
    delay(1000);  
  }  
}
```

Neste exemplo temos dois blocos de código definidos entre chaves, o bloco da sub-rotina ou função "loop()" e o bloco da instrução "if".

É uma boa prática deslocar com a tecla "TAB" cada bloco de código, pois assim é mais fácil visualizar onde cada bloco inicia e termina.

7.3.4 Funções

Funções são trechos de código que por serem usados repetidamente são emcapsulados e podem ser utilizados quando necessário no programa. Algumas funções já estão disponíveis no programa, mas para algumas outras é necessário adicionar ao programa bibliotecas externas de funções. As principais funções serão apresentadas durante as próximas aulas. O exemplo a seguir apresenta dias funções comumente utilizadas no Arduino.

```
digitalWrite(LED_BUILTIN, HIGH); // liga o LED
delay(1000); // Aguarda 1 segundo
```

Neste exemplo temos a função "digitalWrite" que escreve em uma saída digital do Arduino e a função "delay", que para o programa por um determinado tempo.

7.3.5 A função "void setup()"

Esta função é uma função especial na programação do Arduino, ela é executada uma única vez toda vez que o Arduino é ligado ou reiniciado. No corpo desta função devem ser colocadas todas as instruções que devem ser executadas na inicialização do Arduino, como a definição dos pinos de saída por exemplo. Veja o exemplo a seguir.

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}
```

7.3.6 A função "void loop()"

Assim como a função "void setup()", a função "void loop()" é uma função especial na programação do Arduino. Esta função fica se repetindo constantemente durante o funcionamento do Arduino. Toda a lógica do programa que deve ficar se repetindo é inserida nesta função. Pode-se afirmar que esta função é a principal função em um programa para Arduino. Veja o exemplo a seguir.

```
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```


Neste exemplo o programa fica repetidamente ligando e desligando um LED em intervalos de 1 segundo.

Todo programa para Arduino deve possuir as funções “void setup()” e “void loop()”. A seguir temos um trecho de código com o esqueleto mínimo de um programa para Arduino.

```
void setup() {  
  
}  
  
void loop() {  
  
}
```

7.3.7 Funções de entrada e saída

Além das funções apresentadas anteriormente a programação do Arduino disponibiliza um número muito grande de funções. Dentre estas funções pode-se destacar as funções de entrada e saída de valores digitais. Todos os programas desenvolvidos para o Arduino devem possuir algum tipo de comunicação com o mundo exterior. O modo mais comum de comunicação é através de entradas e saídas digitais.

As entradas digitais são responsáveis por verificar o estado de uma entrada do microcontrolador, ou seja, um pino do Arduino, retornando o estado deste pino. O valor de retorno é alto ou baixo, ou do inglês **HIGH** ou **LOW**, representando respectivamente 1 ou 0.

O nome da função que realiza a operação de entrada digital no Arduino é:

digitalRead(pino)

Esta função recebe entre os parênteses um parâmetro que é o número do pino do Arduino que se deseja ler. Veja um exemplo a seguir.

```
val = digitalRead(7);
```

Neste exemplo a variável de nome “val” vai receber o estado do pino 7 do Arduino.

As saídas digitais por sua vez são responsáveis por setar o estado de um pino de saída do Arduino. O valor de um pino do Arduino pode ser setado para alto ou baixo, ou do inglês **HIGH** ou **LOW**, representando respectivamente 1 ou 0.

Quando o Arduino é energizado todos os pinos estão configurados como entrada, assim para que se possa utilizar um pino como saída é necessário configurar o Arduino para isso. A função que instrui o Arduino a configurar um pino como saída é a função:

pinMode(pino, modo);

Esta função recebe dois parâmetros, o número do pino que se deseja configurar e o modo de operação para este pino. Este modo de operação pode ser entrada, do inglês, **INPUT** e é a configuração padrão para todos os pinos ou saída, do inglês **OUTPUT**. Veja um exemplo a seguir.

```
void setup()
{
  pinMode(13, OUTPUT);
}
```

Neste exemplo o pino 13 do Arduino é configurado como saída. A função pinMode() é normalmente utilizada dentro da função void setup();

Quando um pino está configurado como saída é possível alterar o estado deste pino, e o nome da função que realiza a operação de saída digital no Arduino é:

digitalWrite(pino, valor);

Esta função recebe dois parâmetros, o número do pino do Arduino que se deseja alterar e o valor que se deseja gravar neste pino. Os valores podem ser alto ou baixo, ou do inglês **HIGH** ou **LOW**, representando respectivamente 1 ou 0. Veja um exemplo a seguir.

```
digitalWrite(13, HIGH);
```

Neste exemplo o pino 13 do Arduino recebe o valor lógico alto.

As funções de entrada e saída digital permitem que o Arduino receba e envie sinais a um número muito grande de periféricos, permitindo assim que se realize o controle ou a automação de uma variedade de aplicações. O programa a seguir é um pequeno exemplo que utiliza entradas e saídas digitais.

```
int ledPin = 13; // LED conectado ao pino 13
int inPin = 7; // entrada conectada ao pin 7
int val = 0; // variável que armazena o valor da entrada

void setup()
{
```

```
pinMode(ledPin, OUTPUT); // Ajusta o pino 13 como saída
}

void loop()
{
  val = digitalRead(inPin); // lê o pino de entrada
  digitalWrite(ledPin, val); // seta o pino de saída
}
```

Neste exemplo o pino 13 do Arduino é configurado como saída, na função void setup(). Na função void loop() o sinal do pino 7 é lido e atribuído ao pino 13 repetidamente.

7.3.8 Intervalos de tempo

Em muitas aplicações com o Arduino é necessário parar ou pausar o programa por um período de tempo. A IDE do Arduino disponibiliza uma função para este fim. A função que pausa o programa no Arduino é:

delay(ms)

Esta função recebe como parâmetro o tempo que se deseja pausar o programa em milissegundos. A seguir temos um exemplo de programa que utiliza esta função.

```
int ledPin = 13; // LED conectado ao pino 13

void setup()
{
  pinMode(ledPin, OUTPUT); // configura o pino do LED como saída
}

void loop()
{
  digitalWrite(ledPin, HIGH); // Liga o LED
  delay(1000); // Pausa por 1 segundo
  digitalWrite(ledPin, LOW); // Desliga o LED
  delay(1000); // // Pausa por 1 segundo
}
```

Neste exemplo o LED conectado ao pino 13 do Arduino fica piscando, permanecendo 1 segundo ligado e 1 segundo desligado.

7.4 Conclusão

Esta aula apresentou os conceitos básicos sobre a programação do Arduino, apresentando algumas das funções essenciais desta plataforma. Maiores informações sobre estas funções podem ser encontradas no site do Arduino, no endereço <https://www.arduino.cc/reference/en/#functions>.

Na próxima aula trataremos de aspectos mais complexos da programação do Arduino, como tipos de dados, variáveis e tomadas de decisão.

7.5 Exercício

1) Implemente o programa a seguir no Arduino e verifique se funcionamento fornecendo entradas no pino 7 e verificando o comportamento do LED.

```
int ledPin = 13; // LED conectado ao pino 13
int inPin = 7;   // entrada conectada ao pin 7
int val = 0;    // variável que armazena o valor da entrada

void setup()
{
  pinMode(ledPin, OUTPUT); // Ajusta o pino 13 como saída
}

void loop()
{
  val = digitalRead(inPin); // lê o pino de entrada
  digitalWrite(ledPin, val); // seta o pino de saída
}
```

2) Implemente o programa a seguir no Arduino e verifique se funcionamento. Fala alterações nos tempos de LED ligado e desligado e verifique o que acontece.

```
int ledPin = 13; // LED conectado ao pino 13

void setup()
{
  pinMode(ledPin, OUTPUT); // configura o pino do LED como saída
}

void loop()
{
  digitalWrite(ledPin, HIGH); // Liga o LED
  delay(1000); // Pausa por 1 segundo
  digitalWrite(ledPin, LOW); // Desliga o LED
}
```

```
delay(1000);           // // Pausa por 1 segundo  
}
```

- 3) Realize uma pesquisa e descubra como a plataforma online <https://www.tinkercad.com/> pode ser utilizada para simular os programas da plataforma Arduino.
- 4) Simule os dois últimos exemplos apresentados nesta aula usando a plataforma online <https://www.tinkercad.com/>.

AULA 8 - VARIÁVEIS E TOMADA DE DECISÕES

Tipos de dados, criação de variáveis e tomada de decisões no Arduino

8.1 Objetivo:

Nesta aula é apresentada a continuação dos fundamentos da programação do Arduino. O primeiro assunto a ser abordado é a criação de variáveis no Arduino. Neste contexto será apresentada a sintaxe de criação de variáveis nesta plataforma bem como os tipos de dados suportados. Na sequência será apresentada a estrutura e a sintaxe das tomadas de decisão na plataforma Arduino. Para facilitar a compreensão destes novos conhecimentos serão apresentados alguns exemplos e realizados alguns exercícios.

8.2 Variáveis no Arduino

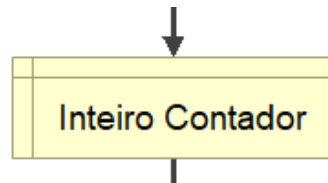
Durante a programação da plataforma Arduino é normalmente necessário armazenar informações na memória deste dispositivo. A linguagem de programação do Arduino permite que se armazene informações na memória através das variáveis.

Cada plataforma computacional e cada linguagem de programação possui características específicas com relação a forma de armazenamento de dados. No Arduino existem vários tipos de dados que podem ser usados para armazenar os dados na memória. A tabela a seguir apresenta estes tipos de dados.

Tipos de Dados	Tamanho	Capacidade	Utilização
void	-	-	Utilizado em sub-rotinas
boolean	8b (1 byte)	true ou false	Valor verdadeiro ou falso
char	8b (1 byte)	1 caractere	Caracteres
unsigned char	8b (1 byte)	0 a 255	Números inteiros
byte	8b (1 byte)	0 a 255	Números inteiros
int	16b (2 byte)	-32768 a 32767	Números inteiros
unsigned int	16b (2 byte)	0 a 65535	Números inteiros
word	16b (2 byte)	0 a 65535	Números inteiros
short	16b (2 byte)	-32768 a 32767	Números inteiros
long	32b (4 byte)	-2147483648 a 2147483647	Números inteiros
unsigned long	32b (4 byte)	0 to 4294967295	Números inteiros
Float	32b (4 byte)	3,4028235E-38 -3,4028235E+38	Números reais

Durante o desenvolvimento dos programas é importante escolher o tipo de dados apropriado ao tipo da informação que se deseja armazenar.

Durante o desenvolvimento de fluxogramas, realizado nas aulas passadas a criação de variáveis era realizada através do bloco de definição. Veja a figura a seguir.



Na plataforma Arduino a sintaxe de criação de variáveis é a seguinte. Primeiro deve-se informar o tipo de dados utilizado, na sequência o nome da variável. O nome da variável deve iniciar com uma letra e não deve possuir espaços. É possível declarar mais de uma variável do mesmo tipo por vez, basta separar seus nomes com vírgula. Ao final da linha de declaração de variáveis deve-se adicionar um ponto e vírgula. Veja a seguir alguns exemplos de declaração de variáveis.

```
int contador, valor_1;  
long velocidade_do_motor;  
float temperatura1;
```

As variáveis criadas desta forma não possuem nenhum valor armazenado. No Arduino é possível atribuir um valor a uma variável assim que ela é criada, basta adicionar o sinal de igual (=) e o valor a ser atribuído logo após o nome da variável. Veja alguns exemplos a seguir.

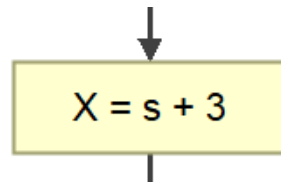
```
int numero_de_pecas = 0;  
float valorMaximo = 1500;
```

É importante salientar que não são permitidos caracteres especiais nos nomes de variáveis, assim letras acentuadas e o “ç” não podem ser usados.

8.3 Operadores

A linguagem de programação do Arduino assim como suas predecessoras, o C e o C++, apresenta um amplo conjunto de operadores que podem ser utilizados na construção dos programas. Estes operadores podem ser utilizados em várias situações, porém, seu uso mais comum é em

IFC - Instituto Federal Catarinense, Campus Luzerna - Programação Aplicada a Microcontroladores operações de atribuição. A figura a seguir apresenta o bloco utilizado nos fluxogramas para a operação de atribuição.



Assim como nos fluxogramas, na programação do Arduino as atribuições são realizadas com o sinal de igual “=”. Nas atribuições é que são realizadas as principais operações lógicas e aritméticas. A tabela a seguir apresenta os principais operadores aritméticos utilizados na programação do Arduino.

Operador	Função
=	Atribuição (ex: $x = y * 3$);
+	Soma (ex: $t = a + b$);
-	Subtração (ex: resultado = valor - resto);
*	Multipliação (ex: total = $x * 10$);
/	Divisão (ex: temp = $ad / 5$);
%	Resto (ex: $r = 10 \% 3$); Neste caso r recebe 1 que é o resto da divisão de 10 por 3.
++	Incremento (ex: contador++); Soma 1 a variável contador.
--	Decremento (ex: contador--); Subtrai 1 da variável contador.

Estes operadores poder operar sobre valores numéricos ou sobre variáveis, porém é importante respeitar o tipo de dados utilizado nas variáveis que compõe a operação.

Também é possível realizar operações mais complexas utilizando vários operadores e, se necessário, usando parênteses “()”. A figura a seguir apresenta um exemplo.

```
Valor = 45 * X / (a + b);
```

A precedência deste operadores segue a lógica da matemática. Primeiro são avaliadas as expressões dentro dos parênteses mais internos, depois os incrementos e decrementos, na sequência as multiplicações e divisões e finalmente as somas e subtrações.

Além dos operadores aritméticos também são disponibilizados vários operadores lógicos. A tabela a seguir apresenta estes operadores, com uma breve explicação de seu funcionamento e com um pequeno exemplo.

Operador	Sintaxe	Descrição
Igual	==	Verifica se o conteúdo de duas variáveis ou de uma variável e uma constante é igual e retorna verdadeiro se sim.
Diferente	!=	Verifica se o conteúdo de duas variáveis ou de uma variável e uma constante é diferente e retorna verdadeiro se sim.
Maior	>	Verifica se o conteúdo de uma variável é maior que o conteúdo de outra variável ou constante e retorna verdadeiro se sim.
Maior ou igual	>=	Verifica se o conteúdo de uma variável é maior ou igual que o conteúdo de outra variável ou constante e retorna verdadeiro se sim.
Menor	<	Verifica se o conteúdo de uma variável é menor que o conteúdo de outra variável ou constante e retorna verdadeiro se sim.
Menor ou igual	<=	Verifica se o conteúdo de uma variável é menor ou igual que o conteúdo de outra variável ou constante e retorna verdadeiro se sim.

Com relação ao bloco de código da função if, é possível utilizar vários formatos para este bloco de código, veja alguns exemplos a seguir.

```

if (x > 120) digitalWrite(LEDpin, HIGH);

if (x > 120)
  digitalWrite(LEDpin, HIGH);

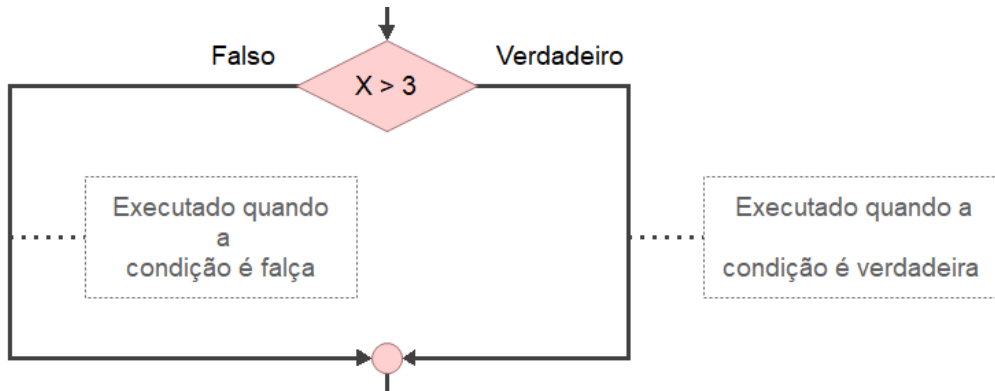
if (x > 120) { digitalWrite(LEDpin, HIGH); }

if (x > 120) {
  digitalWrite(LEDpin1, HIGH);
  digitalWrite(LEDpin2, HIGH);
}

if (x > 120)
{
  digitalWrite(LEDpin, HIGH);
}

```

As tomadas de decisão podem também fornecer dois caminhos alternativos de forma a executar um de dois blocos de código dependendo do resultado de uma condição de teste. Veja um exemplo de fluxograma que apresenta esta condição na figura a seguir.



A sintaxe da função if, neste caso, é semelhante a sintaxe anterior, só que agora é necessário adicionar após o primeiro bloco de código a função “else” e na sequência, um segundo bloco de código que será executado se a condição de teste for falsa. Veja um exemplo na figura a seguir.

```

if(X > 3)
{
  // Executado quando a condição é verdadeira
}
else
{
  // Executado quando a condição é falsa
}
  
```

É possíveis utilizar vários formatos para os blocos de código associados as funções if e else, veja alguns exemplos a seguir.

```

if (x > 120) digitalWrite(LEDpin, HIGH);
else digitalWrite(LEDpin, LOW);

if (x > 120)
  digitalWrite(LEDpin, HIGH);
else
  digitalWrite(LEDpin, LOW);

if (x > 120)
{
  digitalWrite(LEDpin, HIGH);
}
else
{
  digitalWrite(LEDpin, LOW);
}
  
```

8.5 Exemplo de tomada de decisão

A seguir é apresentado um programa que utiliza a função if para tomar uma decisão. O programa verifica o estado do pino 3 do Arduino e se este pino estiver em nível alto o programa pisca o LED conectado ao pino 13.

```
int ledPin = 13;           // LED conectado ao pino 13

void setup()
{
  pinMode(ledPin, OUTPUT); // configura o pino do LED como saída
}

void loop()
{
  if(digitalRead(3)==HIGH) // verifica se o pino 3 está em nível alto
  {
    digitalWrite(ledPin, HIGH); // Liga o LED
    delay(1000);                // Pausa por 1 segundo
    digitalWrite(ledPin, LOW);  // Desliga o LED
    delay(1000);                // Pausa por 1 segundo
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

8.6 Conclusão

Esta aula apresentou conceitos importantes com relação a programação do Arduino. Os tipos de dados e a sintaxe para criação de variáveis é muito importante para o desenvolvimento de qualquer programa no Arduino. As tomadas de decisão através da função if também tem grande relevância na programação desta plataforma. Maiores informações sobre a função if podem ser encontrados no site do Arduino, no endereço:

<https://www.arduino.cc/reference/en/language/structure/control-structure/if/>

Na próxima aula serão apresentados circuitos importantes que ajudam a conectar periféricos às entradas e saídas digitais do Arduino.

8.7 Exercício

- 1) Implemente o programa do exemplo anterior no TinkerCad e no Arduino, e verifique seu funcionamento.
- 2) Altere o programa do exercício anterior de forma que quando o pino 3 do Arduino estiver em nível alto o LED pisca 5 vezes por segundo e quando o pino 3 do Arduino estiver em nível baixo o LED pisca 1 veze por segundo. Repita os testes no TinkerCad e no Arduino.

AULA 9 - ENTRADAS E SAÍDAS DIGITAIS

Estruturas de *hardware* para entradas e saídas digitais no Arduino

9.1 Objetivo:

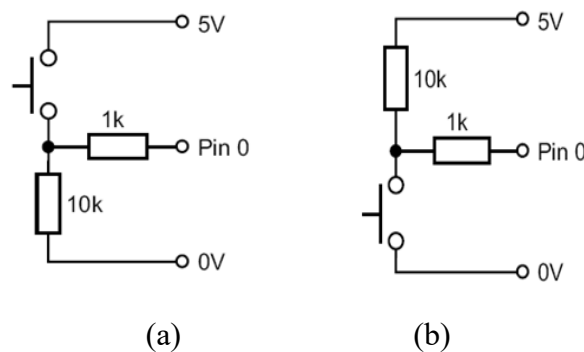
Nesta aula serão apresentados alguns conceitos relacionados aos circuitos externos ao Arduino, necessários a implementação de entradas e saídas digitais.

9.2 Entradas digitais no Arduino

As entradas digitais do Arduino estão preparadas para receber sinais digitais, ou seja, sinais com 0 V ou 5 V. Nem todos os circuitos que se deseja conectar nestas entradas fornecem este tipo de sinal, assim são necessárias algumas adequações para seu correto funcionamento.

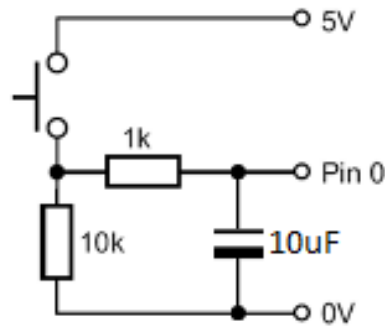
Existem inúmeros circuitos de entrada digital, abordaremos apenas os mais comuns, mas o conhecimento aqui adquirido pode ser facilmente extrapolando para entradas mais complexas.

Entradas para botões, ou chaves, a figura a seguir apresenta dois modelos de conexão para botões ou chaves.



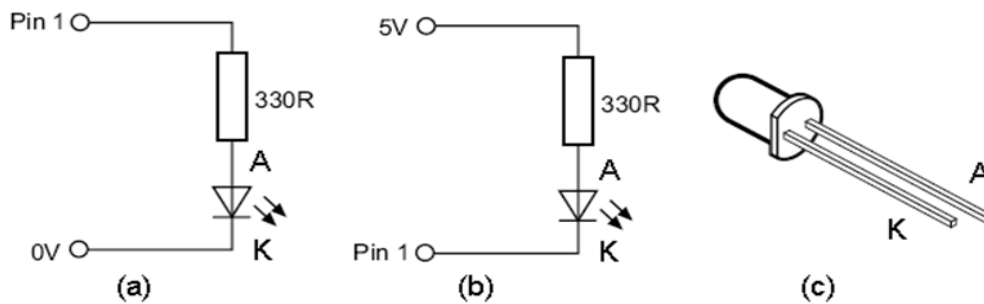
Na figura (a) o Arduino recebe nível lógico alto quando a chave fecha, e na figura (b) o microcontrolador recebe nível lógico baixo quando a chave fecha. Na maioria dos casos o resistor de 1K pode ser suprimido, mas ele ajuda a proteger o circuito.

É importante lembrar que na comutação de chaves mecânicas pode ocorrer a geração de ruídos, para evitar que o microcontrolador interprete erroneamente o sinal é comum utilizar um capacitor para a filtragem deste sinal. Também é possível utilizar no programa um pequeno intervalo de tempo após a detecção da mudança de estado da entrada para a efetiva leitura do sinal. A figura a seguir mostra um circuito de entrada com capacitor de filtro.



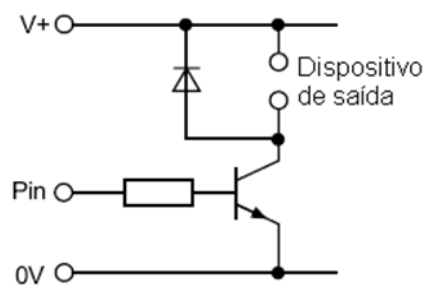
9.3 Saídas digitais no Arduino

As saídas digitais podem comandar uma grande variedade de dispositivos, cada um deles tem suas particularidades. Se a corrente necessária for pequena o microcontrolador pode alimentar diretamente a carga, como nas figuras a seguir.



Na figura (a) o LED acende quando o pino está em nível lógico alto, na figura (b) o LED acende quando o pino está em nível lógico baixo e na figura (c) é possível ver como se identifica a polaridade do LED.

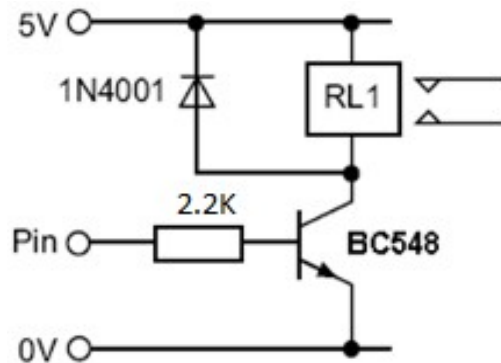
Já para cargas maiores que necessitam mais corrente é necessário utilizar um circuito auxiliar. A figura a seguir apresenta um circuito genérico para este fim.



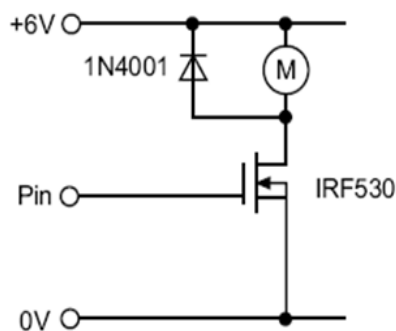
O circuito da figura aciona cargas com uma corrente mais elevada. O tipo de transistor e o valor do resistor dependem da corrente da carga. O diodo serve para proteger o circuito contra descargas.

A seguir serão apresentados alguns exemplos que utilizam o modelo de circuito anterior.

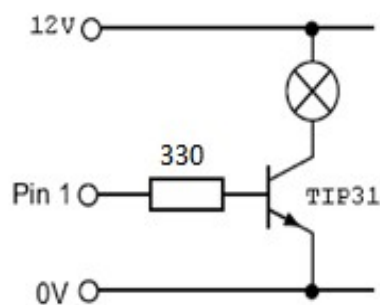
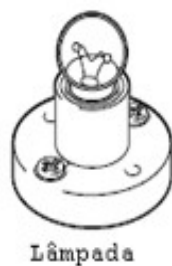
9.3.1 Conexão de reles.



9.3.2 Conexão de um motor utilizando transistor mosfet.



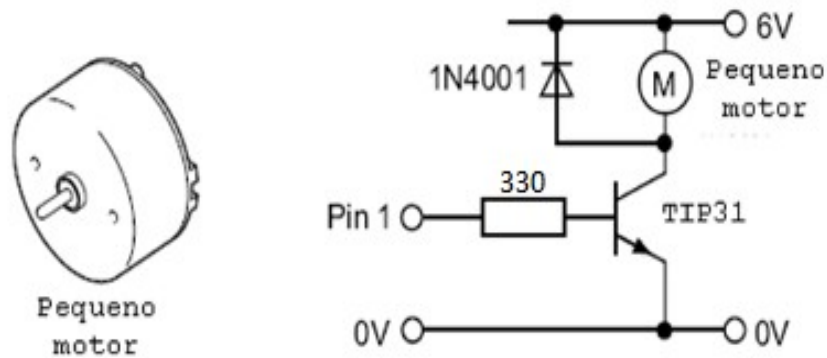
9.3.3 Conexão de lâmpada de sinalização



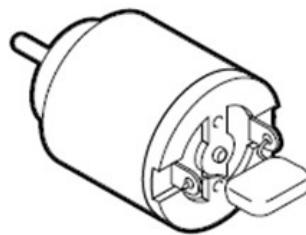
Se observarmos com atenção os exemplos anteriores veremos que a tensão que alimenta a carga é diferente da tensão do microcontrolador (5V), isso nos ajuda a interconectar cargas com tensões diferentes.

9.3.4 Conexão de pequenos motores

A conexão de pequenos motores como na figura a seguir exige alguns cuidados especiais.

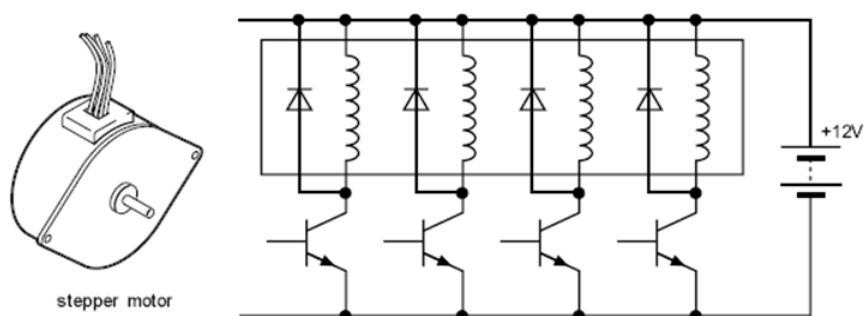


Alguns motores geram muitos ruídos que podem interferir no resto do circuito, assim é comum conectarmos em paralelo com o motor um capacitor cerâmico de 220nF, como na figura a seguir.



9.3.5 Conexão de motores de passo

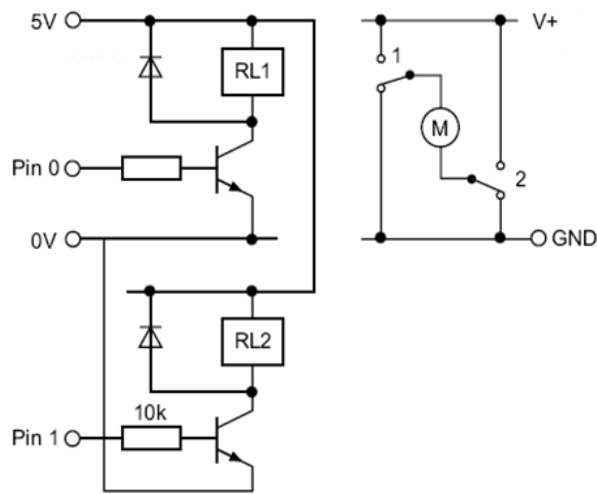
Existem configurações mais complexas, como por exemplo, a conexão de um motor de passo mostrada a seguir.



Os motores de passo têm quatro bobinas e assim necessitam de quatro transistores para seu comando. Para que se possa comandar um motor de passo é necessário gerar uma sequência de pulsos com tempo de duração proporcional a velocidade. Para mais detalhes, procure o manual do fabricante do motor.

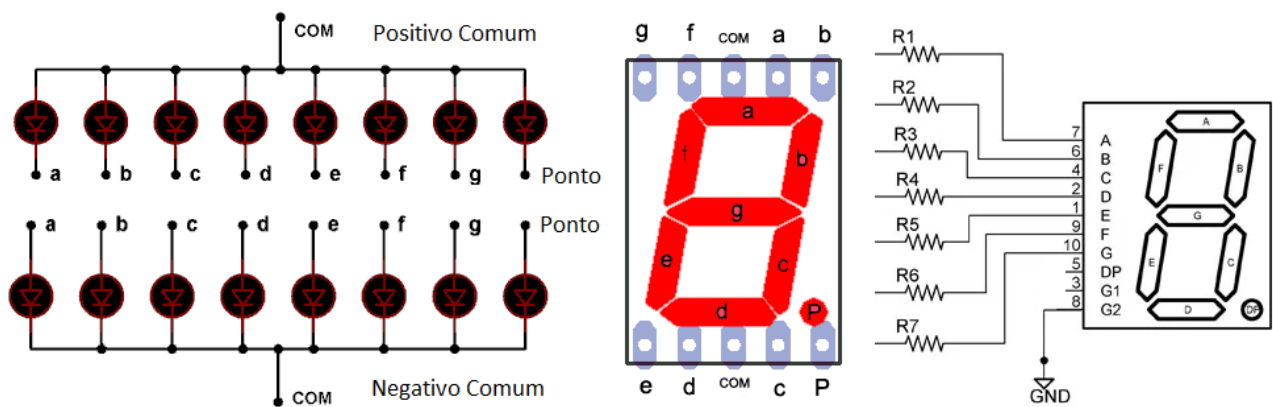
9.3.6 Conexão de motores CC com reversão

Outra situação comum é utilizarmos motores de corrente contínua com reversão, a figura a seguir mostra um exemplo de interligação para este caso.



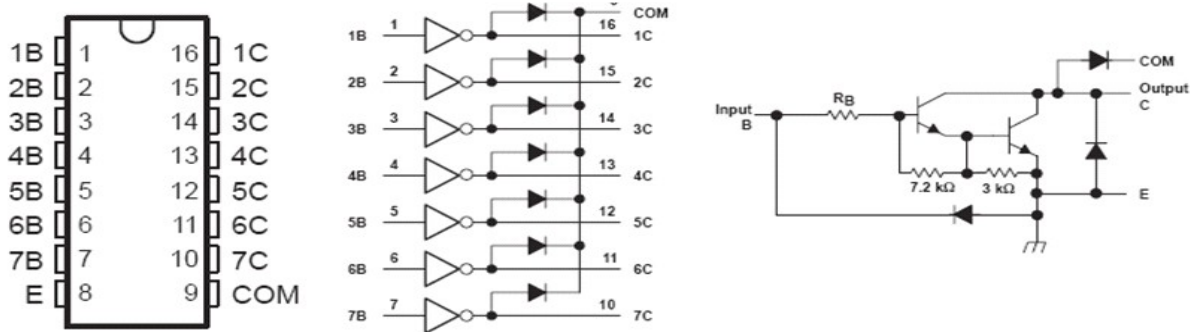
9.3.7 Conexão de displays de LED

Também é comum conectarmos ao microcontrolador os displays de sete segmentos. Na verdade estes displays nada mais são do que sete LEDs e a figura a seguir mostra esta ligação.



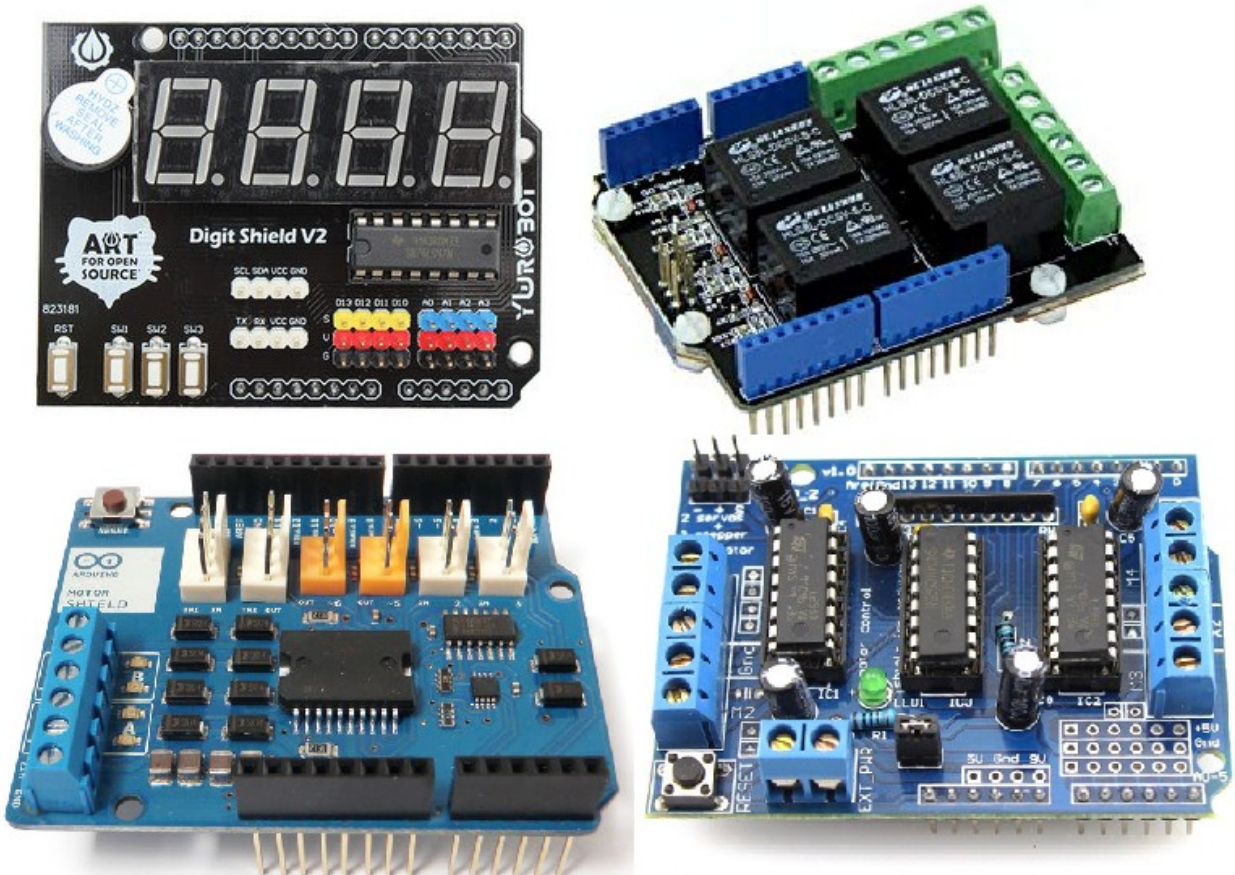
9.3.8 Circuitos especializados de saída

Pensando em facilitar nossa vida os fabricantes de circuito integrados criaram componentes para ajudar na conexão de sinais digitais o ULN2003 mostrado na figura a seguir é um exemplo.



9.4 Shields de entrada e saída digital para Arduino

A plataforma Arduino, como já foi mencionado, possibilita a conexão de placas de expansão. Estas placas chamadas Shields podem ser utilizadas para facilitar a conexão de entradas e saídas digitais ao Arduino. A seguir temos alguns exemplos de Shields para entradas e saídas digitais.



9.5 Conclusão

Esta aula apresentou alguns circuitos básicos importantes para a conexão de periféricos ao Arduino. As entradas e saídas digitais são muito importantes nos projetos que envolvem esta plataforma, assim escolher a forma correta de conexão é fundamental para o correto funcionamento do projeto.

Na próxima aula serão apresentadas mais algumas funções importantes na programação do Arduino, assim como os laços de repetição por exemplo.

9.6 Exercício

- 1) Projete um circuito com Arduino conectando a ele dois botões, um LED vermelho e um LED verde. Em seguida faça um programa em que quando um botão é pressionado o LED vermelho acende e o verde fica apagado. Quando o outro botão é pressionado os LEDs se alternam.
- 2) Projete um circuito com o Arduino e um display de 7 segmentos, utilizando os pino digitais de 0 a 7 para os segmentos de “a” a “g”, respectivamente. Faça um programa que apresenta o número 9 neste display.
- 3) Adicione 2 botões ao circuito do exercício anterior e faça um programa onde uma contagem de 0 a 9 é apresentada no display. O número apresentado deve aumentar ou diminuir conforme os botões são pressionados, um botão aumenta e outro diminui o número apresentado no display.

AULA 10 - FUNÇÕES E LAÇOS DE REPETIÇÃO

Estudo das funções e dos laços de repetição utilizados no Arduino

10.1 Objetivo:

Nesta aula serão estudadas funções, onde os programados pode implementar sub-rotinas ou criar suas próprias bibliotecas. Também serão apresentados os principais laços de repetição utilizados na programação do Arduino.

10.2 Funções na programação do Arduino

Tudo nos programas em C está associado a funções, no Arduino é também assim. Existem duas funções que são obrigatórias na programação do Arduino, `setup ()` e `loop ()`. Outras funções podem ser criadas fora dessas duas funções para executar alguma tarefa do programa.

Uma função é um trecho de programa independente que executa um conjunto de comandos para realizar determinada tarefa. A estrutura básica de uma função é a seguinte.

- As funções iniciam com a identificação de seu tipo (void, int etc).
- Em seguida vem seu nome.
- Seguido de um conjunto de parênteses, que pode ou não conter um conjunto de parâmetros.
- Na sequência temos o início do corpo da função, “{”.
- No corpo são introduzidos os comandos que executam a tarefa delegada a função.
- A função pode então retornar um valor, se necessário.
- Para encerrar a função temos “}”.

A seguir temos um exemplo de função, é uma função simples cuja tarefa é multiplicar dois números e retornar o resultado.

```
int multiplica(int x, int y)
{
    int resultado;
    resultado = x * y;
    return resultado;
}
```

Nesta função pode-se identificar a estrutura conforme explicado anteriormente. A função inicia pela declaração de seu tipo de dado de retorno, no caso “int”. O nome da função é “multiplica”. A função recebe dois parâmetros inteiros, “x” e “y”. No corpo da função é declarada uma variável inteira chamada “resultado”. A esta variável é atribuído o resultado da multiplicação de “x” por “y”. A função finaliza retornando o valor da variável resultado.

A seguir temos um exemplo de programa que utiliza a função do exemplo anterior para multiplicar dois números.

```
int multiplica(int x, int y)
{
    int resultado;
    resultado = x * y;
    return resultado;
}

void setup()
{
}

void loop()
{
    int a = 3;
    int b = 4;
    int c;
    c = multiplica(a,b); // utiliza a função multiplica, c recebe o resultado de 3 * 4.
}
```

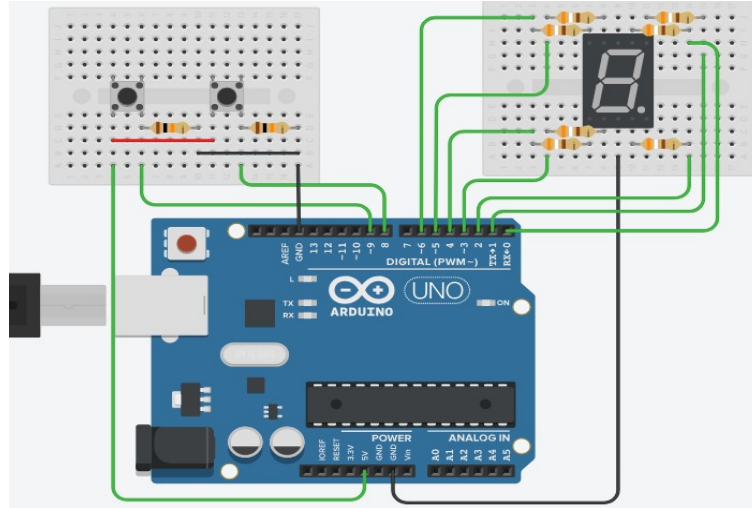
É importante destacar que as funções devem ser implementadas fora do “setup” e do “loop”, e devem ser declaradas antes de ser utilizadas na ordem de execução do código. Para utilizarmos as funções em nossos programas basta “chamá-las” por seu nome, fornecendo os parâmetros necessários, conforme este exemplo.

A divisão do programa em funções permite que um programador crie partes modulares de código que executam uma determinada tarefa e, em seguida, retornam ao programa principal. É comum utilizarmos funções quando é necessário executar a mesma ações repetidas vezes.

A utilização de funções tem várias vantagens: funções ajudam na organização do programa, funções centralizam uma ação em um lugar, de modo que a função só precisa ser pensada e depurada uma única vez, reduz as chances de erros na modificação do código, tornam o código menor e mais compacto porque trechos de código são reutilizados muitas vezes, facilitam a

IFC - Instituto Federal Catarinense, Campus Luzerna - Programação Aplicada a Microcontroladores
reutilização de código em outros programas, tornando-o mais modular e, também torna o código mais legível.

Um exemplo de programa que utiliza função é apresentado na sequência. Inicialmente observe o circuito da figura a seguir.



Neste circuito não conectados dois botões e um display de 7 segmentos ao Arduino. Os botões estão conectados aos pinos 8 e 9. O display está conectado aos pinos de 0 a 6, que também correspondem a porta D do microcontrolador do Arduino. No programa a seguir é criada uma função chamada display que recebe como parâmetro um número inteiro e apresenta este número no display utilizando a porta D.

```
int cont=0;

void display(int n)
{
  int leds[]={0b0111111,0b0000110,0b1011011,0b1001111,0b1100110,
              0b1101101,0b1111101,0b0000111,0b1111111,0b1101111};
  if(n>=0 && n<=9)
  {
    PORTD=leds[n];
  }
}

void setup()
{
  pinMode(0, OUTPUT);
  pinMode(1, OUTPUT);
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
}
```

```

pinMode(6, OUTPUT);
}

void loop()
{
  if(digitalRead(8))
  {
    cont=cont+1;
    delay(500); // Wait for 500 millisecond(s)
  }
  if(digitalRead(9))
  {
    cont=cont-1;
    delay(500); // Wait for 500 millisecond(s)
  }
  display(cont);
}

```

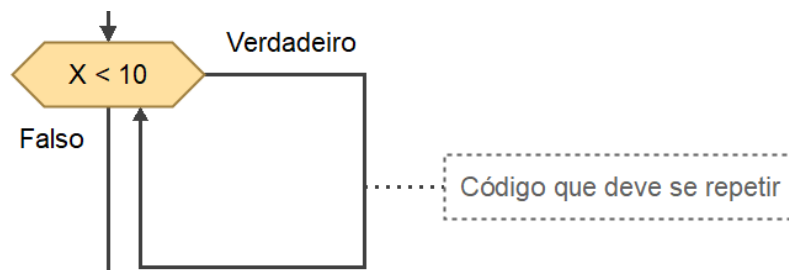
Pode-se observar neste programa que a função `display` é utilizada no “loop”, recebendo como parâmetro a variável “cont”, assim o valor de “cont” é apresentado no display de 7 segmentos.

10.3 Estruturas de controle

A linguagem de programação do Arduino disponibiliza várias estruturas de controle, já estudamos as tomadas de decisão através do comando “if”, agora serão estudadas mais algumas estruturas de controle, começando pelo laço de repetição “Enquanto” ou `while`.

10.3.1 O laço “Enquanto” ou `while`

Como já foi visto o laço “Enquanto” é utilizado para repetir um trecho do programa enquanto uma condição de teste é verdadeira. Na programação do Arduino o comando utilizado para implementar um laço “Enquanto” é o **while**. O `while` é um laço contínuo, que fica se repetindo até que a expressão dentro dos parênteses se torne falsa. A figura a seguir apresenta a topologia do laço “Enquanto” na forma de fluxograma, para que se possa compará-lo com a estrutura do `while` empregado no Arduino.



Na linguagem de programação do Arduino a estrutura do while é a seguinte.

```
while(condição)
{
  // Código que deve se repetir
}
```

Assim o “Enquanto” inicia com a palavra “while”, seguido da condição entre parêntesis. As condições que podem ser usadas são as mesmas já apresentadas para a instrução “if”. Em seguida vem o bloco de código que deve se repetir, colocado entre um par de chaves. Se este bloco de código é composto por apenas uma instrução as chaves podem ser suprimidos.

Veja um exemplo de laço while a seguir.

```
int var;

void setup()
{
  pinMode(13, OUTPUT);
}

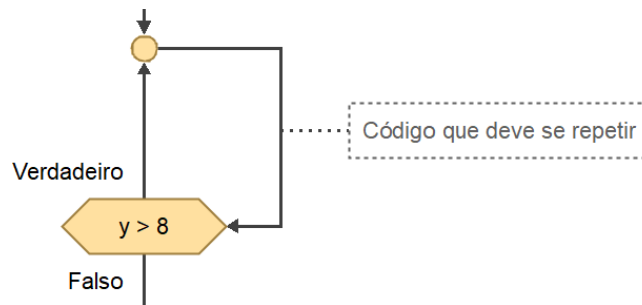
void loop()
{
  var = 0;
  while(var < 20)
  {
    digitalWrite(13, HIGH);
    delay(500);
    digitalWrite(13, LOW);
    delay(500);
    var++;
  }
  delay(4000);
}
```

Neste exemplo o LED conectado ao pino 13 do Arduino pisca 20 vezes, depois o programa aguarda 4 segundos e prossegue normalmente repetindo o loop.

10.3.2 O laço “Fazer” ou do-while

O laço “Fazer” que na linguagem de programação do Arduino é o “do - while” é também utilizado para repetir um trecho do programa enquanto uma condição for verdadeira. A diferença entre o do – while e o while é que no do – while o código é executado pelo menos uma vez, independente da condição ser verdadeira ou não. A figura a seguir apresenta a topologia do laço

IFC - Instituto Federal Catarinense, Campus Luzerna - Programação Aplicada a Microcontroladores
“Fazer” na forma de fluxograma, para que se possa compará-lo com a estrutura do comando do - while empregado no Arduino.



Observando o fluxograma da figura pode-se notar que o código é executado uma vez antes que a condição de teste seja verificada.

Na linguagem de programação do Arduino a estrutura do comando do - while é a seguinte.

```
do
{
  // Código que deve se repetir
} while (condição);
```

Assim o “Fazer” inicia com a palavra “do”, seguido pelo bloco de código que deve se repetir, colocado entre um par de chaves. Na sequência vem a palavra “while” seguida da condição, entre parêntesis. As condições que podem ser usadas são as mesmas já apresentadas para a instrução “if”.

Veja um exemplo de laço do - while a seguir.

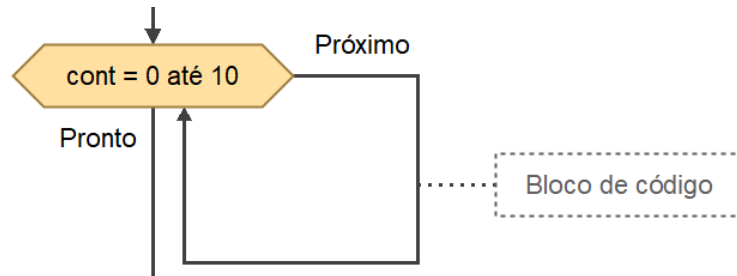
```
int x = 0;
do
{
  delay(50);      // Aguarda 50 ms
  x = leSensor(); // verifica o sensor
} while (x < 100);
```

Neste exemplo um sensor é lido repetidas vezes até que seu valor seja maior ou igual a 100.

10.3.3 O laço “Para” ou for

O laço “para” que na linguagem de programação do Arduino é o “for” é também utilizado para repetir um trecho do programa enquanto uma condição for verdadeira, porém com a ajuda de uma variável de controle que tem seu valor atualizado a cada iteração. A figura a seguir apresenta a

IFC - Instituto Federal Catarinense, Campus Luzerna - Programação Aplicada a Microcontroladores
topologia do laço “Para” na forma de fluxograma, para que se possa compará-lo com a estrutura do comando “for” empregado no Arduino.



Na linguagem de programação do Arduino a estrutura do comando for é a seguinte.

```
for (inicialização; condição; incremento)
{
  //Bloco de código
}
```

Assim o “Para” na linguagem de programação do Arduino inicia com a palavra “for”, seguido pelo conjunto de condições entre parêntesis. Estas condições são divididas em três partes, que são: inicialização, que é onde atribui-se o valor inicial a variável de controle do for; condição, que é a condição que deve ser verdadeira para que o for continue se repetindo; e incremento, que é a atualização da variável de controle que deve ser executada após cada iteração do laço. Em seguida vem o bloco de código que deve se repetir, colocado entre um par de chaves.

Veja um exemplo de laço for a seguir.

```
int i;
for (i=0; i <= 255; i++)
{
  analogWrite(6, i);
  delay(10);
}
```

Neste exemplo uma variável inteira chamada “i” é criada para ser utilizada com o laço for. O for começa atribuindo o valor 0 a variável i. A condição, que deve ser verdadeira, para que o for continue executando é $i \leq 255$. O incremento é feito com a instrução $i++$, que soma 1 a variável i a cada iteração do laço. No corpo do laço o código atribui o valor da variável i a saída analógica do pino 6 do Arduino e aguarda 10 ms.

Um exemplo mais complexo de programa que utiliza o laço for no Arduino é apresentado a seguir.

```
void display(int n)
{
  int leds[]={0b0111111,0b0000110,0b1011011,0b1001111,0b1100110,
              0b1101101,0b1111101,0b0000111,0b1111111,0b1101111};
  if(n>=0 && n<=9)
  {
    PORTD=leds[n];
  }
}

void setup()
{
  pinMode(0, OUTPUT);
  pinMode(1, OUTPUT);
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
}

void loop()
{
  int var;
  for(var = 0; var < 10; var++)
  {
    display(var);
    delay(1000);
  }
}
```

O programa deste exemplo apresenta os números de 0 a 9 em um display de 7 segmentos conectado as saídas digitais de 0 a 6, conforme o circuito da figura apresentada no início desta aula.

10.4 Conclusão

Esta aula apresentou como são implementadas as funções na programação do Arduino. Também foram apresentados os laços de repetição e sua forma de implementação nesta plataforma. Maiores informações sobre estes assuntos podem ser encontradas nos seguintes sites.

<https://www.arduino.cc/en/Reference/FunctionDeclaration>

<https://www.arduino.cc/reference/en/language/structure/control-structure/while/>

<https://www.arduino.cc/reference/en/language/structure/control-structure/dowhile/>

<https://www.arduino.cc/reference/en/language/structure/control-structure/for/>

Na próxima aula serão apresentados alguns dos periféricos internos do Arduino, como interrupções, temporizadores e contadores.

10.5 Exercício

- 1) Implemente o exemplo de programa apresentado junto a instrução “while” no TinkerCad e verifique seu funcionamento utilizando o LED interno do próprio Arduino.
- 2) Implemente o exemplo de programa apresentado junto a instrução “while” no Arduino e verifique seu funcionamento utilizando o LED interno do próprio Arduino.
- 3) Implemente o exemplo de programa apresentado junto a instrução “for” no TinkerCad, conectando o display de 7 segmentos aos pinos de 0 a 6 do Arduino, e verifique seu funcionamento.
- 4) Implemente o exemplo de programa apresentado junto a instrução “for” no Arduino conectando o display de 7 segmentos aos pinos de 0 a 6, e verifique seu funcionamento.
- 5) Altere o programa do exercício anterior adicionando a ele um botão. Quando este botão não estiver pressionado a contagem deve ser crescente, ou seja, de 0 a 9. Quando este botão for pressionado a contagem deve ser decrescente, ou seja, de 9 a 0. Implemente este programa no Tinkercad e no Arduino.

AULA 11 - INTERRUPÇÕES E CONTADORES

Estudo das interrupções, dos contadores e dos temporizadores

11.1 Objetivo:

Nesta aula serão estudadas funções internas do hardware do microcontrolador utilizado no Arduino. O primeiro assunto abordado são as interrupções, seguido pelos contadores e temporizadores. O objetivo do estudo destes conteúdos é permitir aos alunos que utilizem estes recursos de hardware para desenvolver seus programas.

11.2 Interrupções no Arduino

Interrupções são desvios realizados no programa pelo hardware do microcontrolador, ou seja, uma função do programa é chamada quando algum evento acontece no hardware do Arduino. As interrupções são instrumentos muito úteis na programação do Arduino, pois é através delas é possível executar várias tarefas ao mesmo tempo, de forma paralela.

Estas funções, que são chamadas pelas interrupções são chamadas de ISRs e possuem algumas limitações únicas que as outras funções não possuem. Uma ISR não recebe argumentos, e não pode retornar nada. Geralmente, uma ISR deve ser o mais curta e rápida possível. Se o seu programa usa múltiplas ISRs, apenas uma pode ser executada de cada vez, outras interrupções serão executadas após a atual terminar, em uma ordem que depende de sua prioridade.

A função `millis()` depende de interrupções para contar, então essa nunca deverá ser utilizada dentro de uma ISR. O mesmo acontece com a função `delay()`. A função `micros()` funciona inicialmente, mas começará a se comportar erráticamente após 1-2 ms. A função `delayMicroseconds()`, por sua vez, não usa nenhum contador, então funciona normalmente.

Outra questão que deve ser levada em conta quando se fala de ISRs são as variáveis. Tipicamente variáveis globais são usadas para passar dados entre uma ISR e o programa principal, porém, para que isso funcione estas variáveis devem ser declaradas usando o modificador **volatile**.

Existem várias fontes de interrupção no Arduino, como por exemplo, os pinos de interrupção externa, os temporizadores, o conversor A/D, a comunicação serial etc.

O número de entradas que suportam interrupções externas varia conforme o modelo do Arduino. No caso do Arduino uno pode-se utilizar os pinos 2 e 3.

11.2.1 Configuração das Interrupções externas

O Arduino fornece uma função específica para configurar as interrupções externas. Esta função é a `attachInterrupt()`. A sintaxe desta função é a seguinte.

`attachInterrupt(digitalPinToInterrupt(pino), ISR, modo);`

Onde:

pino: é o pino do Arduino (2 ou 3).

ISR: é o nome da função que deve ser executada pela interrupção.

modo: é o modo de ativação da interrupção, que pode ser,

- LOW acionar a interrupção quando o estado do pino for LOW,
- CHANGE acionar a interrupção quando o sempre estado do pino mudar
- RISING acionar a interrupção quando o estado do pino for de LOW para HIGH apenas,
- FALLING acionar a interrupção quando o estado do pino for de HIGH para LOW apenas.

A seguir temos um exemplo de programa que utiliza a interrupção externa no Arduino.

```
const int pinoLed = 9;
const int pinoInterrupcao = 2; // só podem ser os pinos 2 ou 3

volatile int estado = LOW;

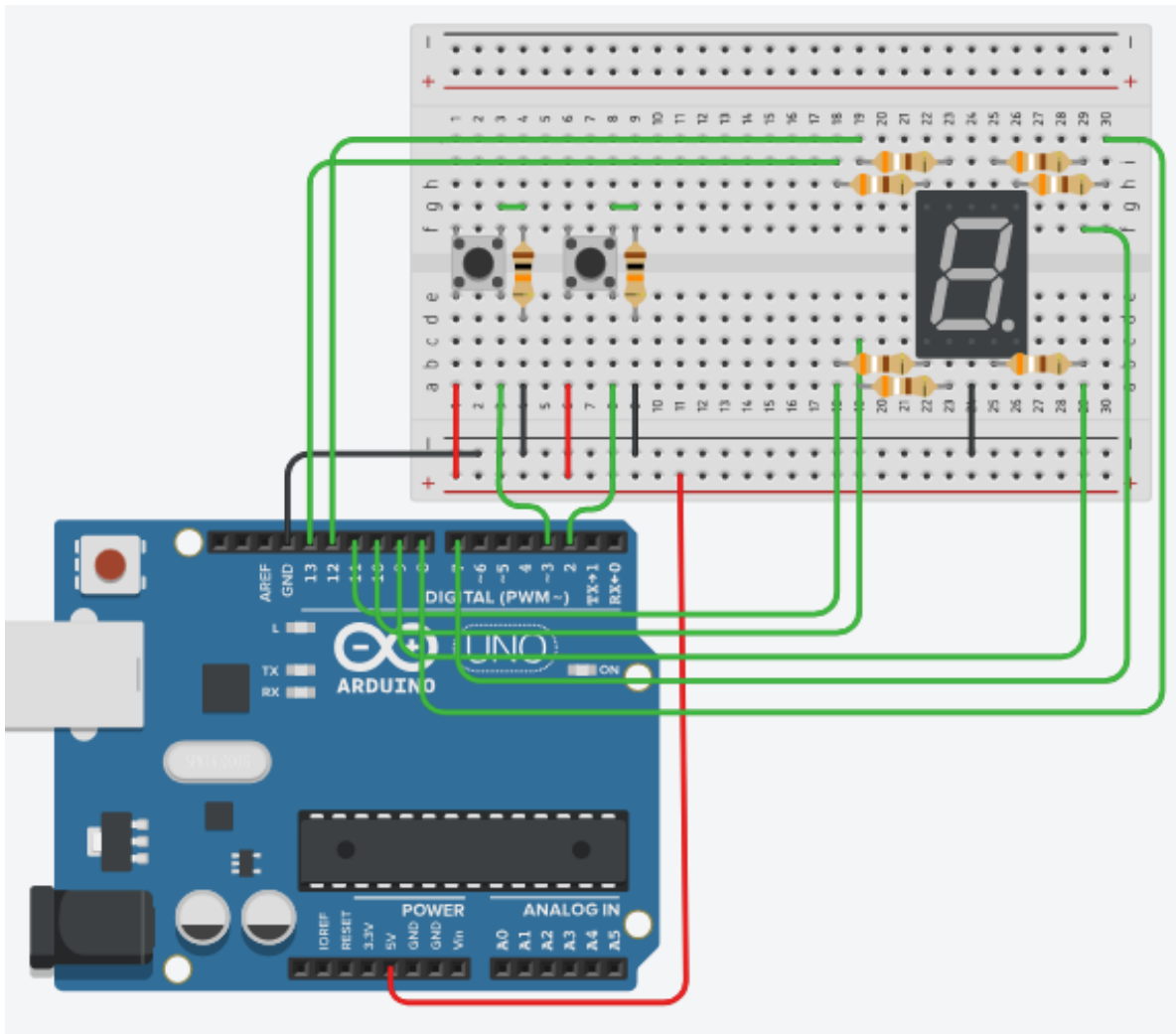
void pisca() // função de interrupção
{
  estado = !estado; // inverte o valor do estado
}

void setup()
{
  pinMode(pinoLed, OUTPUT);
  // ativa a interrupção
  attachInterrupt(digitalPinToInterrupt(pinoInterrupcao), pisca, RISING);
}

void loop()
{
  digitalWrite(pinoLed, estado);
}
```

Neste exemplo, um LED conectado ao pino 9 do Arduino é ligado e desligado através de um botão conectado ao pino 2 do Arduino, com o auxílio da interrupção externa.

Observe que neste exemplo não é necessário verificar o estado do pino 2 para verificar se o botão foi pressionado, o hardware o microcontrolador utilizado no Arduino faz isso automaticamente.



No circuito da figura temos as seguintes conexões.

Pino 2 – Botão de incremento.

Pino 3 – Botão de decremento.

Pino 7 – Segmento A do display.

Pino 8 – Segmento B do display.

Pino 9 – Segmento C do display.

Pino 10 – Segmento D do display.

Pino 11 – Segmento E do display.

Pino 12 – Segmento F do display.

Pino 13 – Segmento G do display.

O programa a seguir utiliza interrupções para incrementar e decrementar os números apresentados no display.


```

volatile int cont=0;
void display(int n)
{
  if(n==0)
  {
    digitalWrite(7,HIGH);
    digitalWrite(8,HIGH);
    digitalWrite(9,HIGH);
    digitalWrite(10,HIGH);
    digitalWrite(11,HIGH);
    digitalWrite(12,HIGH);
    digitalWrite(13,LOW);
  }
  if(n==1)
  {
    digitalWrite(7,LOW);
    digitalWrite(8,HIGH);
    digitalWrite(9,HIGH);
    digitalWrite(10,LOW);
    digitalWrite(11,LOW);
    digitalWrite(12,LOW);
    digitalWrite(13,LOW);
  }
  if(n==2)
  {
    digitalWrite(7,HIGH);
    digitalWrite(8,HIGH);
    digitalWrite(9,LOW);
    digitalWrite(10,HIGH);
    digitalWrite(11,HIGH);
    digitalWrite(12,LOW);
    digitalWrite(13,HIGH);
  }
  if(n==3)
  {
    digitalWrite(7,HIGH);
    digitalWrite(8,HIGH);
    digitalWrite(9,HIGH);
    digitalWrite(10,HIGH);
    digitalWrite(11,LOW);
    digitalWrite(12,LOW);
    digitalWrite(13,HIGH);
  }
  if(n==4)
  {
    digitalWrite(7,LOW);
    digitalWrite(8,HIGH);
    digitalWrite(9,HIGH);
    digitalWrite(10,LOW);
    digitalWrite(11,LOW);
    digitalWrite(12,HIGH);
    digitalWrite(13,HIGH);
  }
  if(n==5)
  {
    digitalWrite(7,HIGH);
    digitalWrite(8,LOW);
    digitalWrite(9,HIGH);
    digitalWrite(10,HIGH);
    digitalWrite(11,LOW);
    digitalWrite(12,HIGH);
    digitalWrite(13,HIGH);
  }
  if(n==6)
  {
    digitalWrite(7,HIGH);
    digitalWrite(8,LOW);
    digitalWrite(9,HIGH);
    digitalWrite(10,HIGH);
    digitalWrite(11,HIGH);
    digitalWrite(12,HIGH);
    digitalWrite(13,HIGH);
  }
  if(n==7)
  {
    digitalWrite(7,HIGH);
    digitalWrite(8,HIGH);
    digitalWrite(9,HIGH);
    digitalWrite(10,LOW);
    digitalWrite(11,LOW);
    digitalWrite(12,LOW);
    digitalWrite(13,LOW);
  }
  if(n==8)
  {
    digitalWrite(7,HIGH);
    digitalWrite(8,HIGH);
    digitalWrite(9,HIGH);
    digitalWrite(10,HIGH);
    digitalWrite(11,HIGH);
    digitalWrite(12,HIGH);
    digitalWrite(13,HIGH);
  }
  if(n==9)
  {
    digitalWrite(7,HIGH);
    digitalWrite(8,HIGH);
    digitalWrite(9,HIGH);
    digitalWrite(10,HIGH);
    digitalWrite(11,LOW);
    digitalWrite(12,HIGH);
    digitalWrite(13,HIGH);
  }
}

void aumenta() // função de interrupção
{
  cont++;
}

void diminui() // função de interrupção
{
  cont--;
}

void setup()
{
  pinMode(7, OUTPUT); //A
  pinMode(8, OUTPUT); //B
  pinMode(9, OUTPUT); //C
  pinMode(10, OUTPUT); //D
  pinMode(11, OUTPUT); //E
  pinMode(12, OUTPUT); //F
  pinMode(13, OUTPUT); //G
  attachInterrupt(digitalPinToInterrupt(2), aumenta, RISING);
  attachInterrupt(digitalPinToInterrupt(3), diminui, RISING);
}

void loop()
{
  display(cont);
}

```

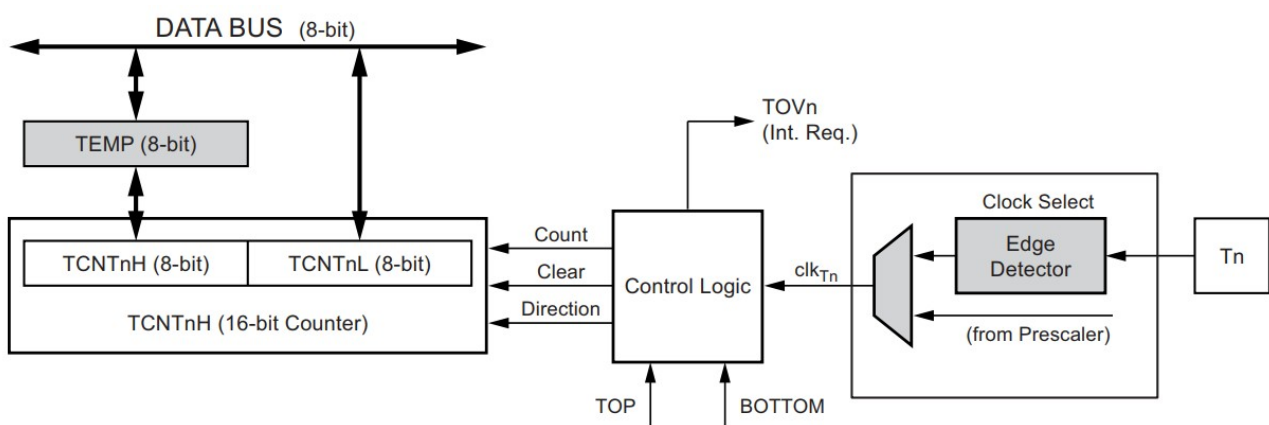
É importante observar que neste exemplo a interrupções incrementam e decrementam a variável “cont”, assim foi necessário declarar esta variável com o modificador “volatile”.

11.3 Contadores e Temporizadores no Arduino

Os temporizadores e os contadores são elementos de hardware presentes no microcontrolador utilizado no Arduino. Na realidade trata-se de um único elemento que realiza ambas as tarefas. Isso se deve ao fato de um contador poder contar eventos e assim trabalhar como um contador ou então contar tempo e trabalhar como um temporizador. Assim sendo um temporizador nada mais é do que um contador que conta o tempo. No microcontrolador utilizado no Arduino existem 3 destes contadores ou temporizadores.

Utilizaremos como exemplo o temporizador ou contador 1, que é composto por um contador de 16 bits que pode contar eventos ou pulsos de clock, operando assim como temporizador. Ele possui um divisor de frequência que pode ser utilizado para dividir o clock por um determinado valor antes que este sinal seja aplicado ao contador. Este circuito é conhecido como prescaler.

A figura a seguir apresenta um diagrama de blocos do temporizador ou contador 1.



O temporizador ou contador 1 pode receber os pulsos para sua contagem tanto do circuito interno como de um sinal externo. A fonte dos pulsos de contagem é definida nos registradores de controle. A seguir são apresentados estes registradores.

TCCR1A – Registrador de controle do Timer 1 A

7	6	5	4	3	2	1	0	
COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
R/W	R/W	R/W	R/W	R	R	R/W	R/W	
0	0	0	0	0	0	0	0	

TCCR1B – Registrador de controle do Timer 1 B

7	6	5	4	3	2	1	0	
ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Cada um destes registradores possui grupos de bits que devem ser configurados para ajustar o modo de operação do contador ou temporizados. A configuração destes registradores não é trivial, e requer uma leitura atenciosa do manual do microcontrolador utilizado no Arduino.

Além destes registradores existem ainda outros registradores que são utilizados na contagem e na comparação dos valores dos temporizadores ou contadores.

A seguir é apresentado um exemplo de programa que utiliza o hardware do microcontrolador como temporizador.

```
ISR(TIMER1_OVF_vect)
{
  TCNT1 = 49911;
  if(digitalRead(13)==LOW) {
    digitalWrite(13,HIGH);
  }
  else {
    digitalWrite(13,LOW);
  }
}

void setup()
{ // configuração do temporizador
  TCNT1 = 49911; // para 1 segundo
  PRR &= ~(1 << PRTIM1);
  TCCR1A = 0;
  TCCR1B=(0<<WGM13)|(0<<WGM12)|0<<ICNC1|0<<ICES1|(1<<CS12)|(0<<CS11)|(1<<CS10);
  TIMSK1 = 0 << OCIE1B | 0 << OCIE1A | 0 << ICIE1 | 1 << TOIE1;
  sei();
  pinMode(13,OUTPUT);
}

void loop()
{
}
```

Neste exemplo o temporizador 1 é configurado para executar uma interrupção de 1 em 1 segundo, esta interrupção é utilizada para ligar ou desligar o LED conectado ao pino 13 do Arduino.

Os temporizadores e contadores são assunto bastante amplo e complexo, o próprio Arduino não possui suporte nativo a todas as funções destes componentes de hardware.

Este material apresentou apenas um pequeno exemplo de temporizadores ou contadores, para mais informações o manual do microcontrolador deve ser consultado.

11.4 Conclusão

Nesta aula foram apresentados temas relacionados a funções mais específicas do microcontrolador utilizado no Arduino. O assunto é bastante amplo, e esta aula serve apenas para demonstrar superficialmente o funcionamento destes componentes. Maiores informações sobre o microcontrolador utilizado no Arduino podem ser encontrada em:

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

Na próxima aula serão apresentados as entradas analógicas e as saídas PWM do Arduino.

11.5 Exercício

- 1) Implemente o exemplo de programa que utiliza as interrupções externas para alterar os números do display no TinkerCad e verifique seu funcionamento.
- 2) Implemente o exemplo de programa que utiliza as interrupções externas para alterar os números do display no Arduino e verifique seu funcionamento.
- 3) Implemente o exemplo de programa do temporizador de 1 segundo no TinkerCad e verifique seu funcionamento.
- 4) Implemente o exemplo de programa do temporizador de 1 segundo no Arduino e verifique seu funcionamento.

AULA 12 - ENTRADAS E SAÍDAS ANALÓGICAS

Estudo do conversor Analógico/Digital e das saídas PWM

12.1 Objetivo:

O objetivo desta aula é discutir o funcionamento e a utilização do conversor Analógico/Digital presente no Arduino, bem como sua utilização na leitura de sinais analógicos. Também é objetivo desta aula estudar as saídas de Modulação por Lagura de Pulso, ou em inglês Pulse Width Modulation (PWM). Os sinais PWM são responsáveis pelas saídas analógicas do Arduino.

12.2 O hardware do conversor Analógico/Digital (A/D)

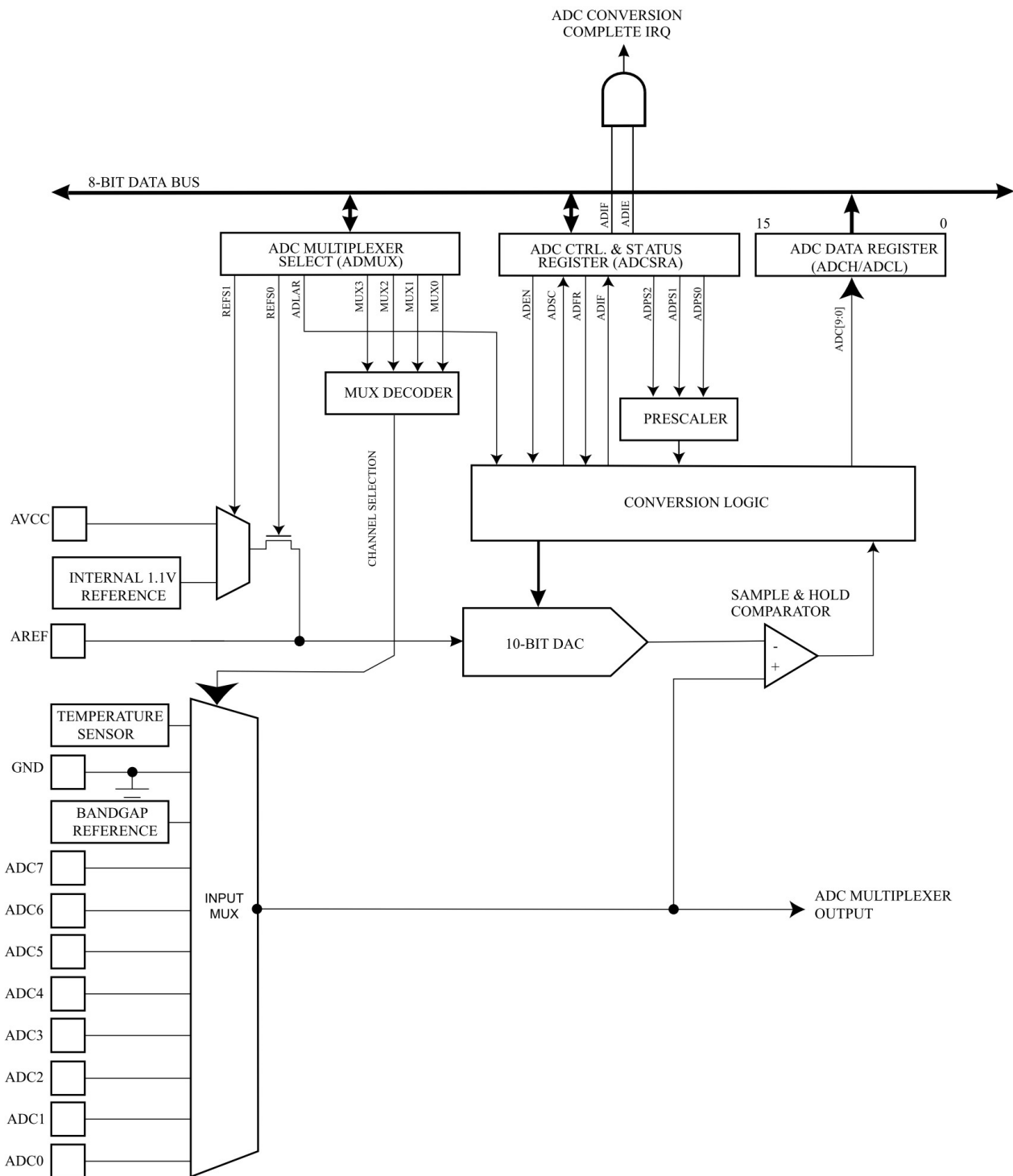
O microcontrolador utilizado no Arduino possui em seu interior um circuito conversor Analógico/Digital. Este circuito, como o próprio nome sugere, converte sinais analógicos em sinais digitais que podem ser processados pelo Arduino.

O conversor A/D é muito útil nos projetos que envolvem o Arduino porque muitos dos sinais enviados pelos sensores utilizados nestes projetos são analógicos e não podem ser interpretados diretamente pelo Arduino. Assim para que o Arduino possa “medir” os sinais enviados por este tipo de sensores é necessário que estes sinais sejam digitalizados através do conversor A/D.

O conversor A/D utilizado na família de microcontroladores AVR tem as seguintes características.

- 10 bits de resolução
- Linearidade de 0,5 LSB
- Precisão absoluta de ± 2 LSB
- Tempo de conversão de 13 - 260 μ s
- Até 15K amostras por segundo em resolução máxima
- 6 canais multiplexados
- Opção de trabalhar em 8 bits
- Tensão de entrada de 0 - VCC
- Referência interna opcional de 1.1V
- Interrupção de final de conversão
- Leitura interna de temperatura

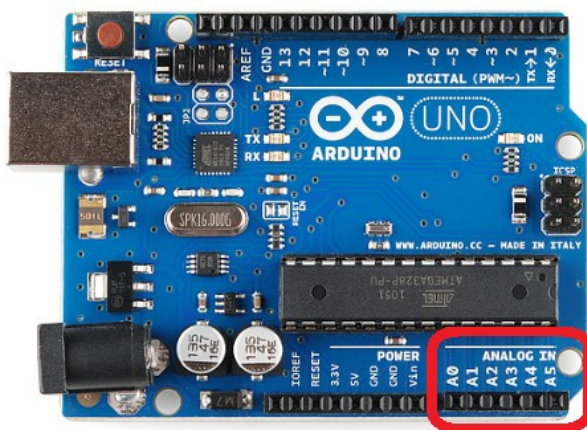
A figura a seguir mostra a topologia interna do conversor A/D utilizado no Arduino.



O Arduino contém 6 entradas analógicas (A0 à A5), como pode ser observado na figura a seguir. Como o conversor Analógico/Digital é de 10 bits, isso significa que ele converte tensões de

IFC - Instituto Federal Catarinense, Campus Luzerna - Programação Aplicada a Microcontroladores
entrada entre 0 e 5 volts em valores inteiros entre 0 e 1023. Assim tem-se uma resolução entre leituras de: 5 volts / 1024 ou, 0.0049 volts (4,9 mV). O intervalo de entrada e a resolução podem ser alterados usando o comando `analogReference()`.

Cada conversão, na configuração normal do Arduino, demora cerca de 100 microssegundos (0,0001 s), portanto, a taxa máxima de leitura é de cerca de 10.000 vezes por segundo. A figura a seguir destaca as entradas analógicas do Arduino.



12.3 A utilização do conversor Analógico/Digital do Arduino

A linguagem de programação do Arduino fornece algumas funções específicas para a utilização do conversor Analógico/Digital. A principal delas é a `analogRead()`, cuja sintaxe é apresentada a seguir.

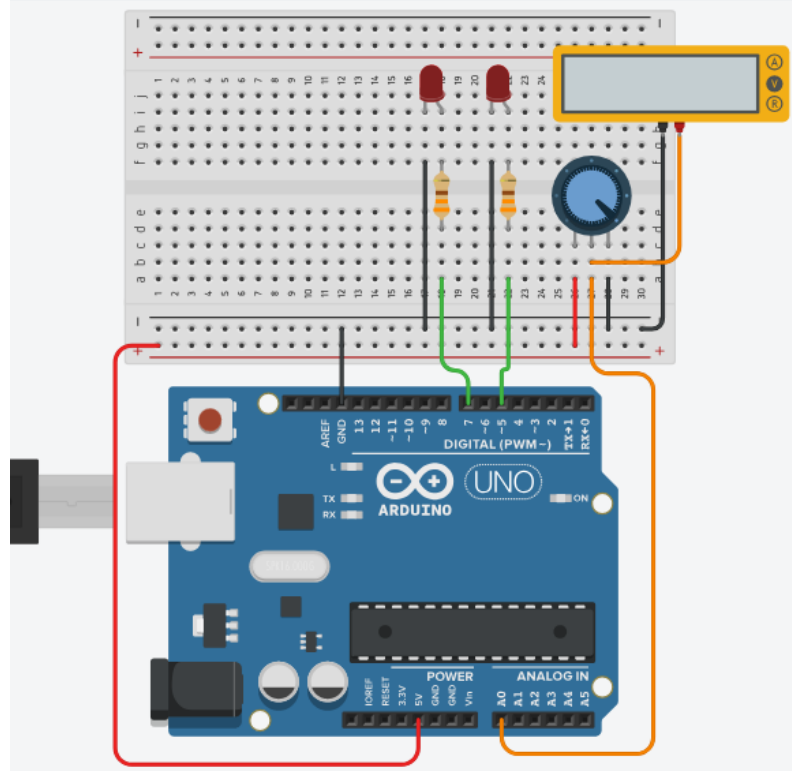
```
analogRead(pino);
```

onde “pino” é um dos pinos analógicos (A0 à A5).

A seguir é apresentado um exemplo que utiliza a entrada analógica A0 na leitura da tensão de um potenciômetro. Este potenciômetro tem seus terminais conectados ao 0V e ao 5V, assim o potenciômetro pode enviar ao Arduino uma tensão entre 0 e 5 V, conforme o seu ajuste.

O programa utilizado neste exemplo tem a função de ligar o primeiro LED quando a tensão da entrada analógica A0 for maior que 2V e ligar o segundo LED quando a a tensão da entrada analógica A0 for maior que 4V.

A figura a seguir apresenta o circuito utilizado neste exemplo. Nesta figura foi também adicionado um voltímetro, de modo a permitir visualizar a tensão na entrada analógica A0.



A seguir é apresentado o programa utilizado neste exemplo.

```
int vin;
void setup()
{
  pinMode(5, OUTPUT);
  pinMode(7, OUTPUT);
}
void loop()
{
  vin=analogRead(A0);

  if(vin>409) // 2V
  {
    digitalWrite(5,HIGH);
  }
  else
  {
    digitalWrite(5,LOW);
  }

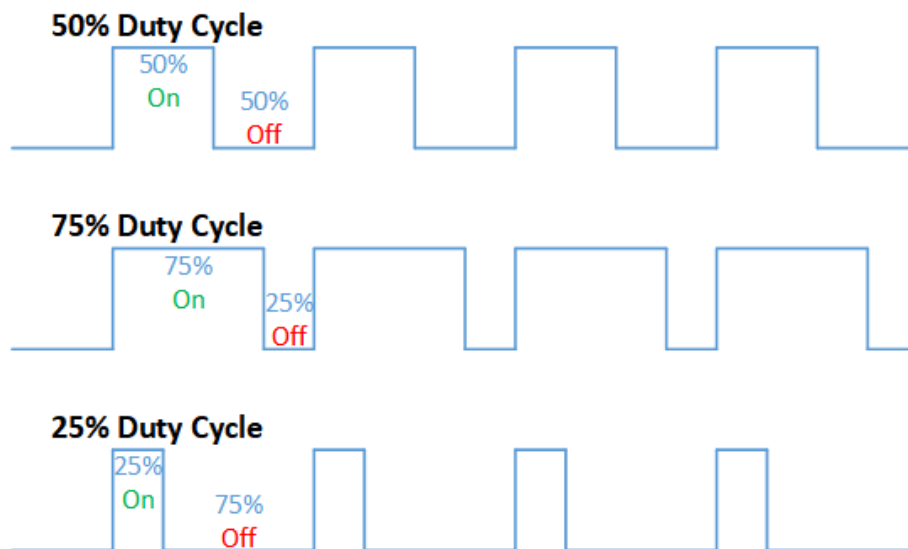
  if(vin>818) // 4V
  {
    digitalWrite(7,HIGH);
  }
  else
  {
    digitalWrite(7,LOW);
  }
}
```


Existe ainda outra função disponibilizada para a programação do Arduino e que tem relação com o conversor Analógico/Digital. É a função `analogReference()`, cuja função é escolher a referência de tensão do conversor Analógico/Digital.

12.4 Sinais PWM

A modulação por largura de pulso (ou Pulse Width modulation - PWM) é uma forma de simular um sinal analógico através de uma saída digital. Sua principal utilização é no controle de dispositivos como lâmpadas e motores. Neste tipo de sinal o valor médio de tensão é enviado a carga, controlando a potência média aplicada. Quanto mais tempo o interruptor estiver ligado em comparação com os períodos desligados, maior será a energia total fornecida à carga.

A figura a seguir apresenta algumas formas de onda de sinais PWM.



O importante é saber que a tensão média do sinal PWM é igual à relação entre o tempo ligado e o tempo desligado do sinal modulado.

12.5 Saídas PWM no Arduino

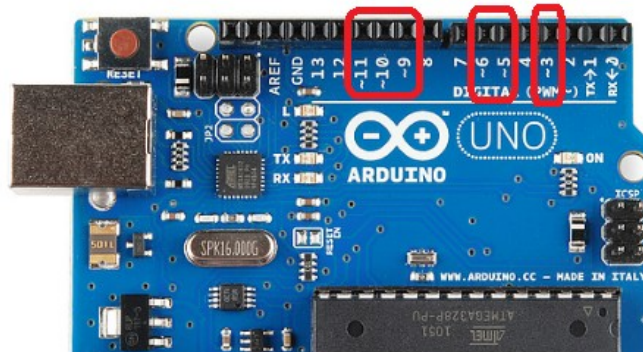
O microcontrolador utilizado no Arduino possui 6 saídas digitais com capacidade de gerar automaticamente sinais PWM. Estas saídas estão conectadas aos pinos 3, 5, 6, 9, 10 e 11. Não é possível utilizar outros pinos com a função PWM.

Na programação do Arduino a função responsável pelo controle das saídas PWM é a função `analogWrite()`. A seguir é apresentada sua sintaxe.

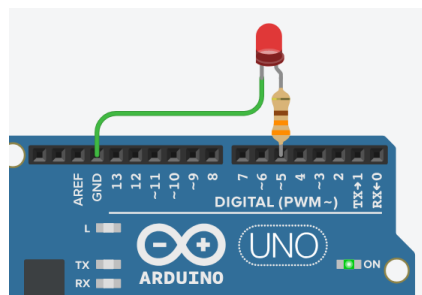
IFC - Instituto Federal Catarinense, Campus Luzerna - Programação Aplicada a Microcontroladores
analogWrite(pino, valor);

Onde: “pino” é um dos pinos que suportam a função PWM e “valor” é uma número entre 0 e 255, onde 0 representa uma razão cíclica de 0% e 255 representa uma razão cíclica de 100%.

A figura a seguir destaca as saídas PWM do Arduino.



A seguir temos um exemplo de programa que utiliza a saída PWM do Arduino para variar o brilho de um LED conectado ao pino 5. A figura a seguir apresenta o circuito utilizado neste exemplo.



O programa a seguir utiliza a função “analogWrite” para alterar o brilho entre LED de 0 a 100%.

```
int valor=0;

void setup()
{
  pinMode(5, OUTPUT);
}

void loop()
{
  analogWrite(5,valor);
  valor++;
  if(valor>255)
  {
    valor=0;
  }
  delay(100);
}
```

12.6 Conclusão

Nesta aula foram apresentadas as entradas analógicas e as saídas PWM do Arduino. Estes assuntos são bastante importantes, pois são através destas entradas e saídas que o Arduino consegue interagir com sinais que não são digitalizados.

Maiores informações sobre as entradas analógicas do Arduino podem ser obtidas em: <https://www.arduino.cc/en/Tutorial/AnalogInputPins>.

Maiores informações sobre as saídas PWM do Arduino podem ser obtidas em: <https://www.arduino.cc/en/Tutorial/PWM>.

Na próxima aula será tratada a comunicação serial no Arduino.

12.7 Exercício

- 1) Implemente o exemplo de programa que utiliza a entrada analógica para a leitura da tensão de um potenciômetro no TinkerCad e verifique seu funcionamento.
- 2) Implemente o exemplo de programa que utiliza a entrada analógica para a leitura da tensão de um potenciômetro no Arduino e verifique seu funcionamento.
- 3) Implemente o exemplo de programa que utiliza a saída PWM para variar o brilho do LED no TinkerCad e verifique seu funcionamento.
- 4) Implemente o exemplo de programa que utiliza a saída PWM para variar o brilho do LED no Arduino e verifique seu funcionamento.

AULA 13 - COMUNICAÇÃO SERIAL NO ARDUINO

Estudo da porta de comunicação serial do Arduino

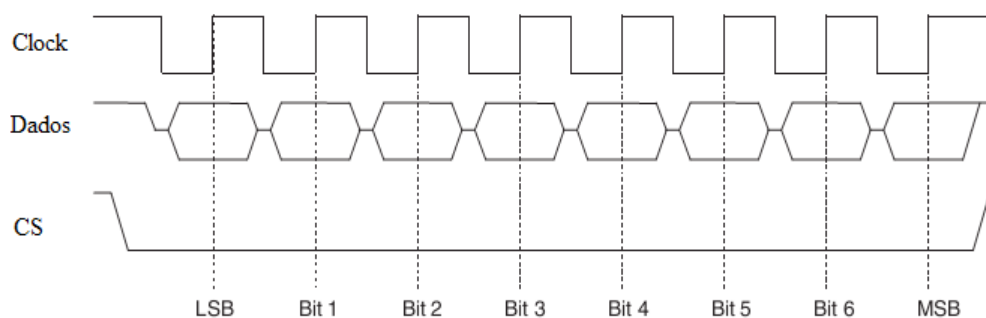
13.1 Objetivo:

O objetivo desta aula é estudar a porta de comunicação serial presente no Arduino. Será apresentada uma fundamentação sobre comunicação serial e sobre o hardware envolvido. Também serão estudados os comandos envolvidos na programação do Arduino, de modo a utilizar a comunicação serial nesta plataforma.

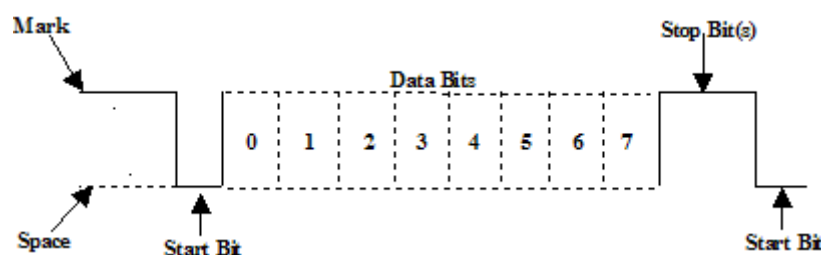
13.2 Fundamentos da Comunicação Serial

Comunicação serial é o processo de enviar dados um bit de cada vez, sequencialmente, num canal de comunicação ou barramento. A comunicação serial é usada em toda comunicação de longo alcance e na maioria das redes de computadores e redes industriais.

Existem diversos tipos de comunicação serial, dentre elas podemos destacar a comunicação serial síncrona, onde junto com os dados é enviado um sinal de clock para sincronização e a comunicação serial assíncrona, onde a sincronização é feita através da temporização dos sinais. A figura a seguir mostra a forma de onda de um sinal serial síncrono.



A figura a seguir mostra a forma de onda de um sinal serial assíncrono.



A comunicação serial síncrona é mais utilizada para enviar sinais em alta velocidade e curta distância, não sendo apropriada a maioria das aplicações industriais. Já a comunicação serial assíncrona é utilizada em comunicações com distâncias maiores, conseqüentemente perdendo em velocidade.

As velocidades de transmissão dos dados são padronizadas para facilitar a configuração dos equipamentos, a seguir são mostradas as principais velocidades de transmissão.

300 bps	600 bps	1200 bps
2400 bps	4800 bps	9600 bps
14400 bps	19200 bps	28800 bps
38400 bps	56000 bps	57600 bps
115200 bps	128000 bps	256000 bps

Alguns equipamentos permitem que se configure velocidades de transmissão personalizadas, permitindo assim outros valores.

Também existem padrões no que diz respeito ao número de bits de dados e de parada que são transmitidos em cada pacote. É possível ainda configurar um bit extra de verificação de erros, chamado de bit de paridade. A tabela a seguir mostra os valores comuns para cada um destes parâmetros.

Parâmetro	Valores Comuns
Número de bits por pacote	5, 6, 7 e 8.
Bits de parada	1, 1,5 e 2.
Bit de paridade	Par, Impar, 0 e 1.

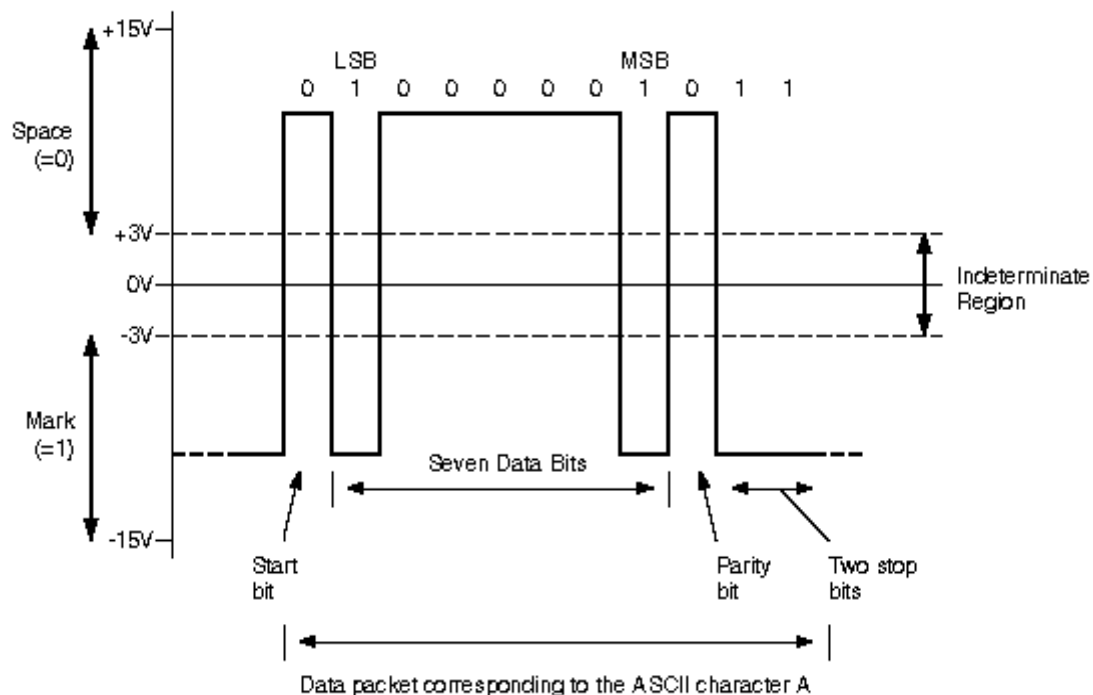
Eletricamente existem vários padrões, que definem níveis de tensão para cada nível lógico. No interior das placas de circuito normalmente utilizamos 0 e 5 V, assim como nos circuitos digitais normais, porém estes níveis de tensão não são apropriados para comunicações com distâncias maiores que alguns centímetros.

Para comunicações em distâncias maiores são utilizados vários padrões, dentre os quais estudaremos o RS232 e o RS485.

13.3 O padrão RS232

O padrão RS232 é o padrão utilizado nos computadores pessoais, e assim é amplamente difundido em uma grande variedade de equipamentos. Este padrão foi desenvolvido para comunicação entre 2 dispositivos, não permitindo um número maior de equipamentos.

Neste padrão os sinais definidos como níveis de tensão, o nível lógico 0 é representado por uma tensão positiva entre 3 e 15 V, já o nível lógico 1 é representado por uma tensão negativa entre -3 e -15 V. A figura a seguir mostra esta situação.



Utilizando níveis de tensão mais elevados é possível alcançar distâncias maiores, na casa de alguns metros. Um fator importante nesta distância é a blindagem do cabo, cabos blindados fornecem maior imunidade a ruído permitindo maior alcance.

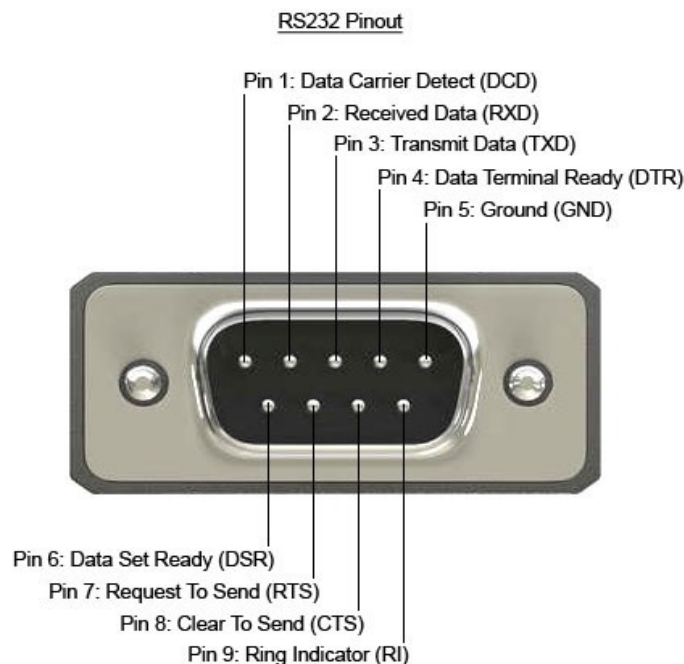
O padrão RS232 também define alguns sinais de controle, que ajudam a sincronizar o envio e o recebimento dos dados. Estes sinais são:

- DTR: (Data Terminal Ready) Usado para avisar que está pronto para comunicar
- RTS: (Request To Send) Usado para avisar que deseja enviar dados agora.
- DSR: (Data Set Ready) Usado para avisar que está pronto para operar.
- CTS: (Clear To Send) Usado para avisar que está pronto para aceitar transmissão de dados.

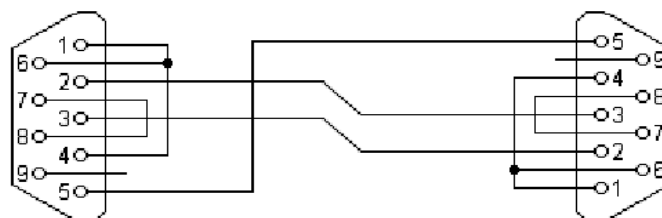
- CD: (Data Carrier Detect) Usado para avisar que estabeleceu uma conexão.
- RI: (Ring Indicator) Usado para avisar que a linha está chamando (usado com modems).

Estes sinais não são obrigatórios, mas podem ser usados para facilitar a comunicação e estão presentes nas portas seriais dos computadores.

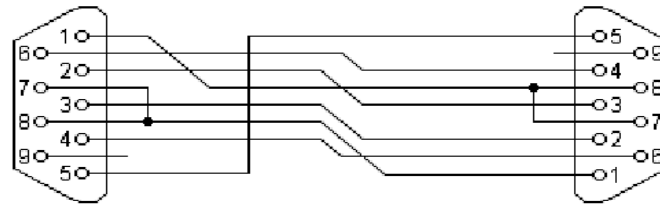
A transmissão dos dados é realizada pela saída TX, e a recepção é realizada pela entrada RX. O conector padrão nos computadores para a porta serial é o DB9 macho, e a distribuição dos pinos é mostrada na figura a seguir.



Para que se possa realizar a comunicação são apenas necessários os sinais de transmissão TXD, de recepção RXD e de referência GND. A figura a seguir mostra a ligação entre dois dispositivos em RS232.



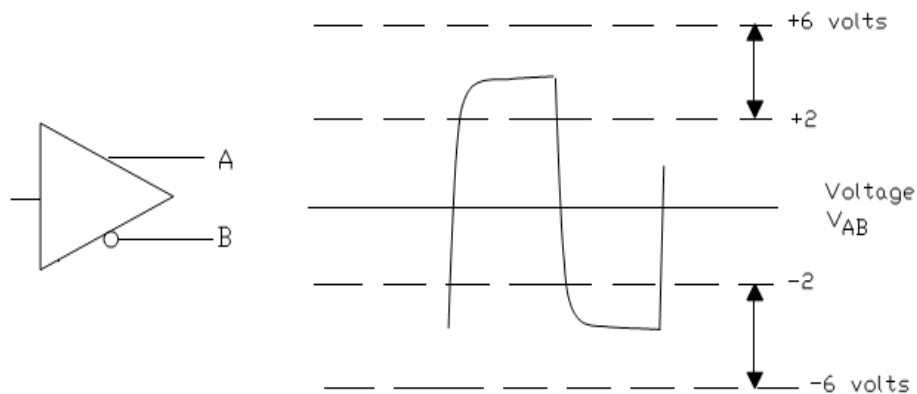
Caso seja necessário um controle do fluxo de dados através do hardware todos os sinais devem ser interligados. A figura a seguir mostra esta ligação.



Analisando estas informações é possível observar que o padrão RS232 permite comunicação em ambos os sentidos ao mesmo tempo.

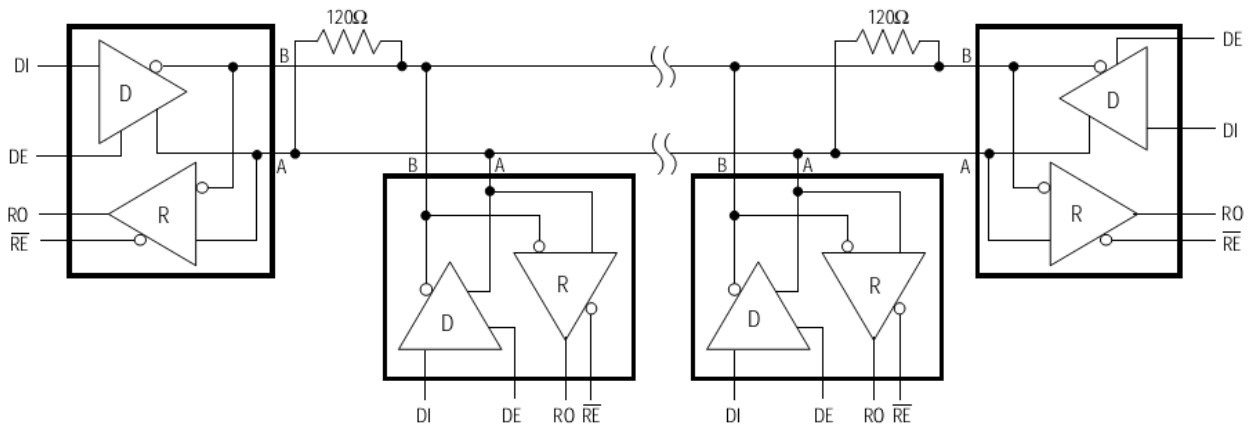
13.4 O padrão RS485

O padrão RS485 é o padrão utilizado nas redes industriais, diferente do padrão RS232 cada sinal é transmitido através de 2 fios. Este arranjo é mais imune a ruído, permitindo comunicações com distâncias superiores a 1 Km.



Eletricamente falando o sinal que transmite os dados na comunicação RS485 é um sinal diferencial. A figura a seguir mostra como este sinal diferencial funciona. É importante salientar que os níveis de tensão indicados na figura dizem respeito a tensão entre os terminais A e B, e não com relação a referência.

O padrão de comunicação RS485 é projetado para interligar 2 ou mais dispositivos, permitindo assim redes com vários dispositivos. Porém, tanto a transmissão como a recepção são feitas no mesmo par de fios, o que não permite que um dispositivo envie e receba dados ao mesmo tempo. A figura a seguir mostra a ligação de alguns dispositivos no padrão RS485.

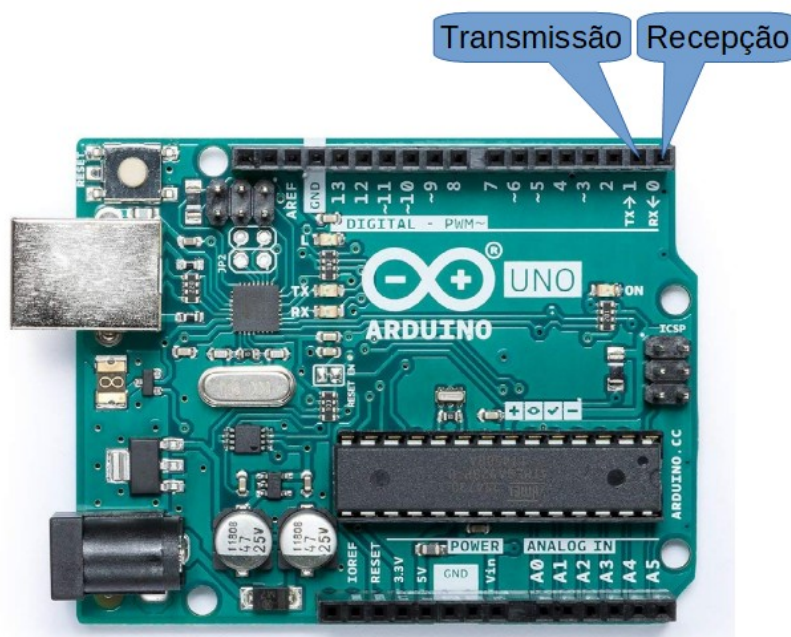


Observando a figura podemos notar a presença de 2 resistores de 120Ω , estes resistores são necessários para casar a impedância dos equipamentos com a rede, tornando a mesma mais imune a ruídos e a erros de comunicação.

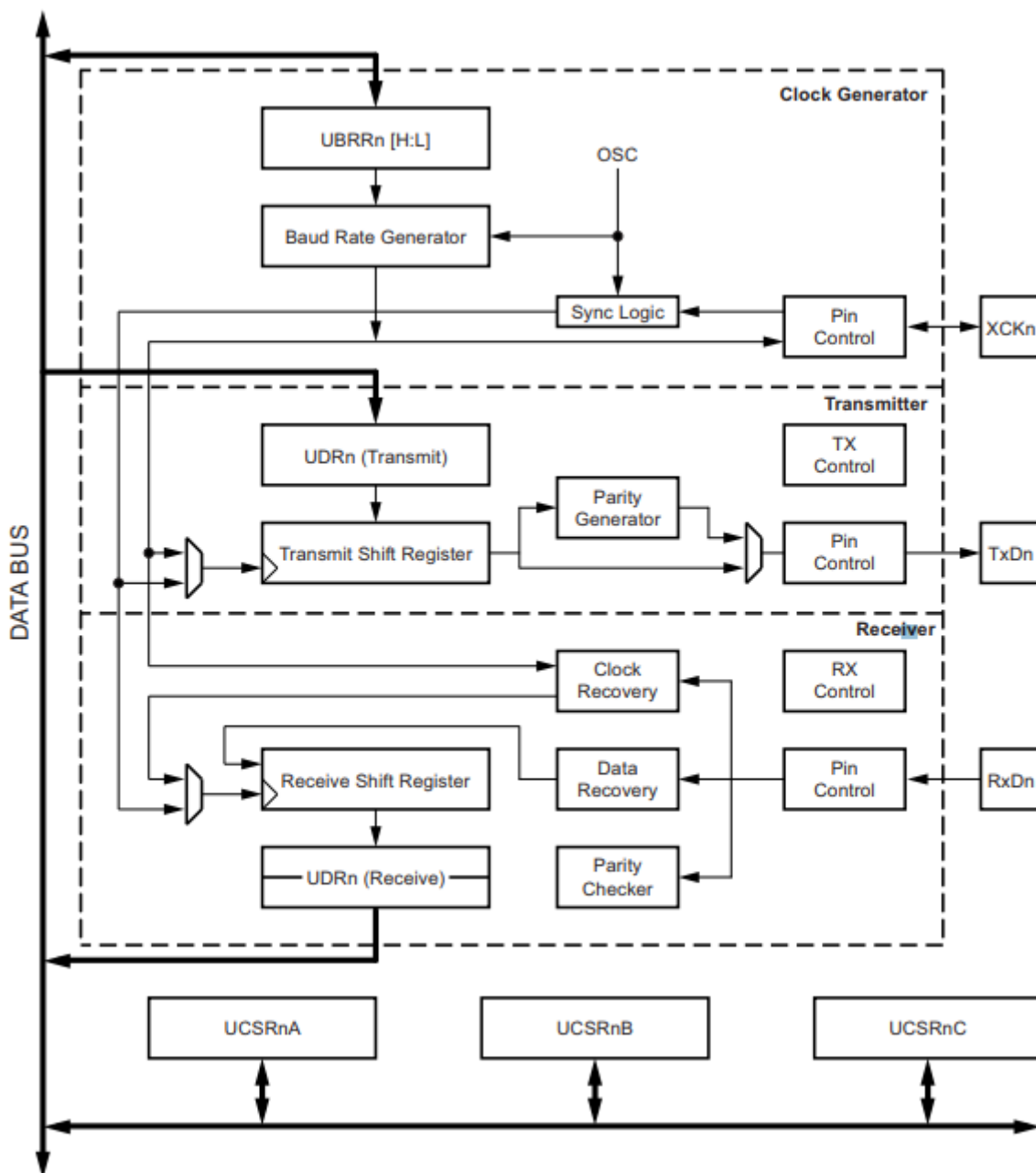
13.5 Comunicação Serial no Arduino

O Arduino tem em seu interior todo o hardware necessário a realização de comunicação serial, tanto síncrona como assíncrona. É possível utilizá-los tanto no padrão RS232 como no padrão RS485, porém, nestes casos, devemos adicionar dispositivos externos para adequar os sinais transmitidos e recebidos.

A figura a seguir destaca os pinos utilizados para a comunicação serial no Arduino.

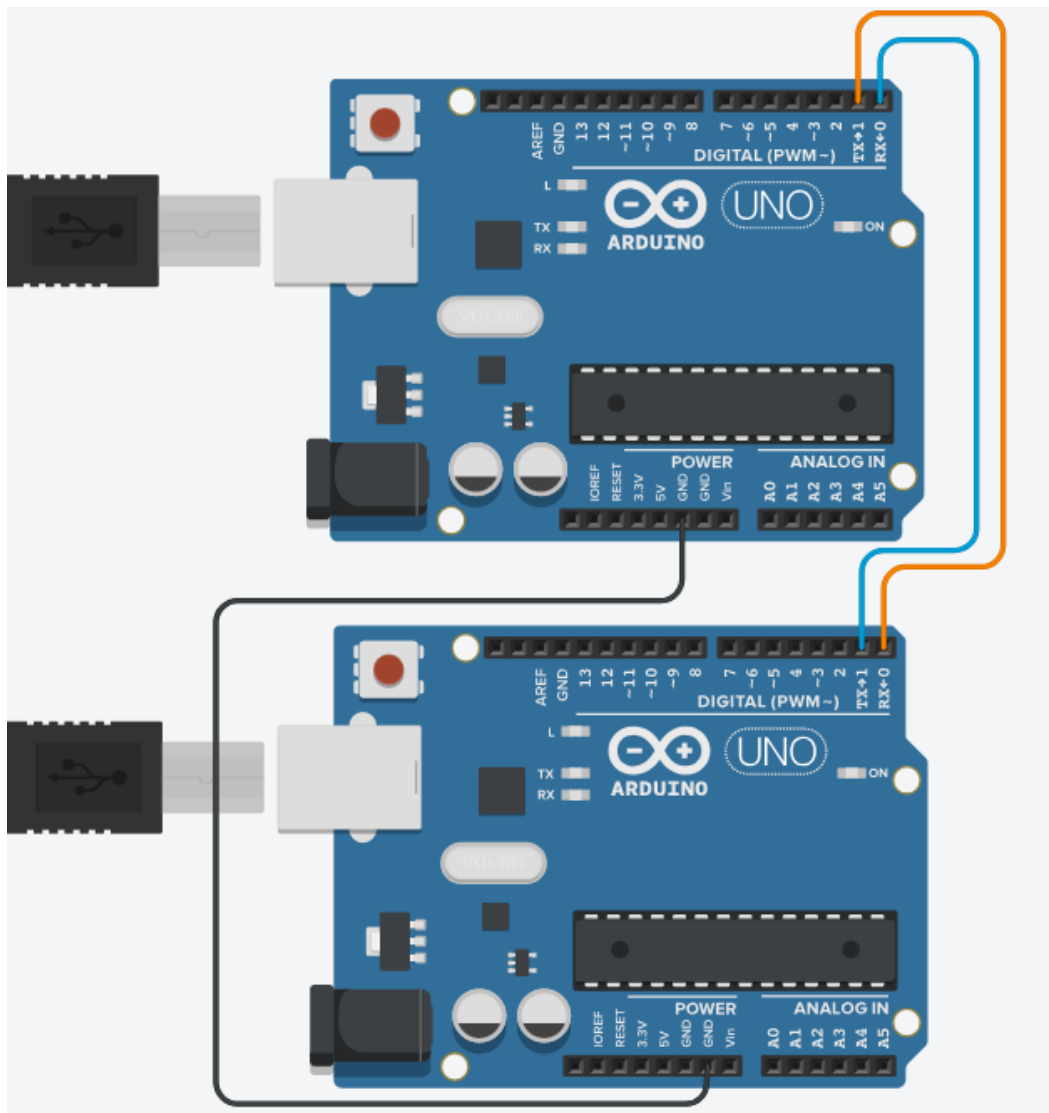


O microcontrolador utilizado no Arduino possui um circuito bastante completo para tratar da comunicação serial. A figura a seguir apresenta este circuito.



Não é o objetivo deste material estudar o funcionamento interno do circuito de comunicação serial do Arduino, maiores informações podem ser obtidas diretamente do manual do microcontrolador utilizado no Arduino.

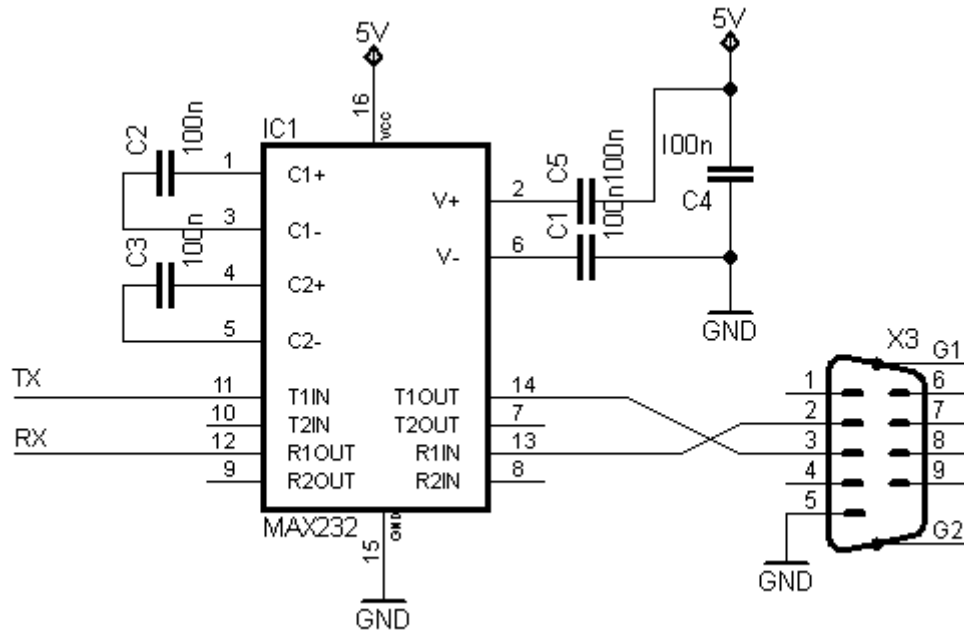
Se desejarmos conectar diretamente dois Arduinos, não é necessário utilizar nenhum padrão complexo de comunicação, basta apenas conectá-los como mostrado na figura a seguir.



É importante lembrar que a distância não pode ultrapassar alguns centímetros, pois este tipo de sinal é muito susceptível a ruídos. Para que a comunicação aconteça, é necessário que os dois microcontroladores estejam trabalhando na mesma velocidade de transmissão.

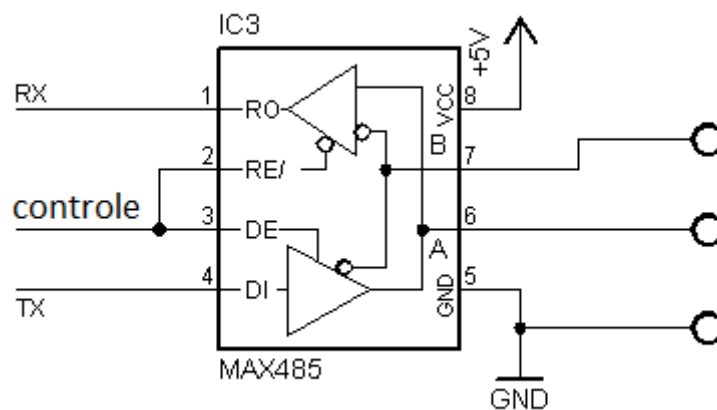
13.5.1 O padrão RS232 no Arduino

Para que o Arduino possa se comunicar através do padrão RS232 é necessário que se adicione a ele um circuito conversor de sinais. Existem vários circuitos integrados que desenvolvem esta função, o mais comum é o MAX232. A figura a seguir mostra o circuito necessário para que o microcontrolador possa se comunicar no padrão RS232 utilizando o circuito integrado MAX232.

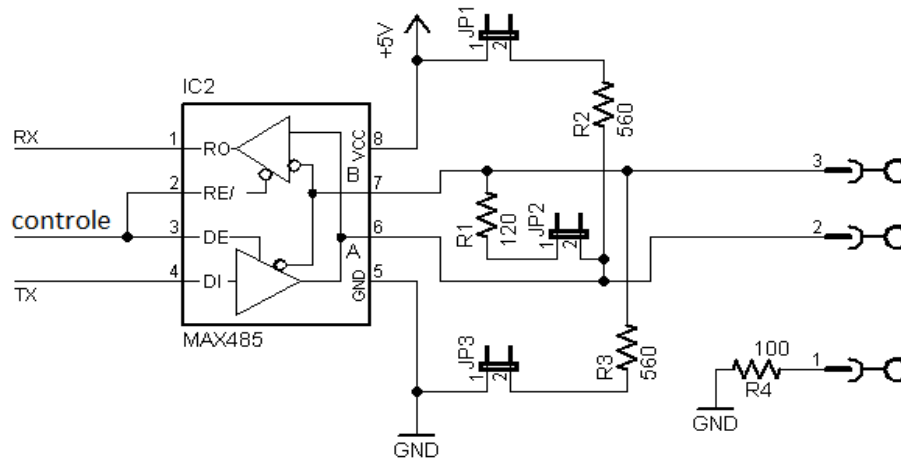


13.5.2 O padrão RS485 no Arduino

Para que o Arduino possa se comunicar através do padrão RS485 é necessário que se adicione a ele um circuito conversor de sinais. Existem vários circuitos integrados que desenvolvem esta função, o mais comum é o MAX485. A figura a seguir mostra o circuito necessário para que o microcontrolador possa se comunicar no padrão RS485 utilizando o circuito integrado MAX485.



Este circuito é utilizado para conectar o microcontrolador a uma rede onde já existam os resistores de terminação e polarização. A figura a seguir mostra um circuito mais completo onde estão presentes resistores de polarização da linha e o resistor de terminação.



Também foi incluído neste circuito um resistor no GND, com o objetivo de absorver diferenças de potencial. Esta configuração mais completa deve ser utilizada apenas em uma das extremidades da rede, pois é ela que mantém a rede estável quando ninguém está transmitindo. No circuito foram adicionados jumpers que permitem ligar ou desligar cada um dos resistores independentemente. Estas figuras tratam apenas do meio físico da rede, é necessário que se estabeleça um protocolo de comunicação para que os dados façam sentido tanto para quem transmite como para quem recebe.

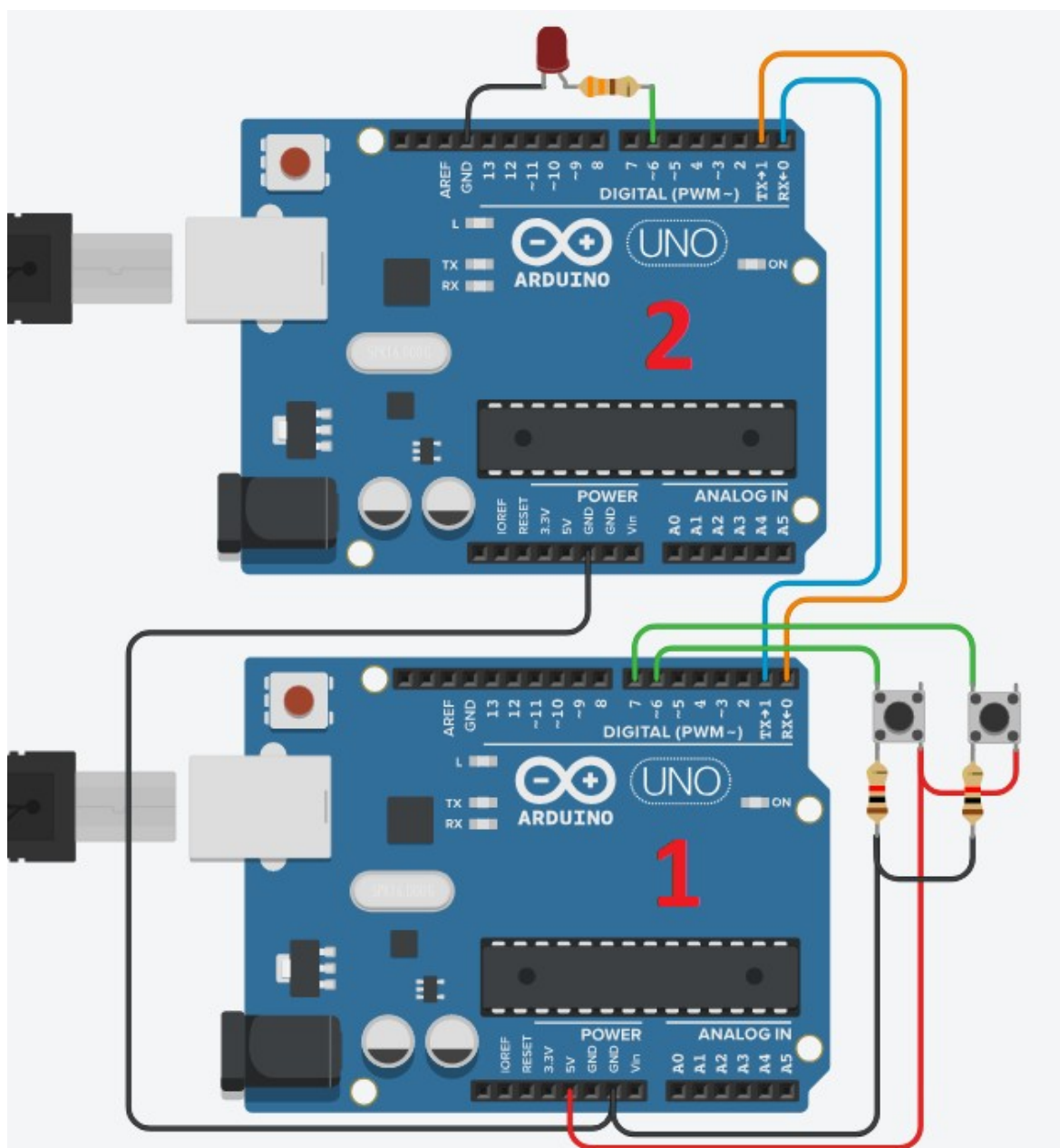
13.5.3 Programação do Arduino para comunicação serial

O ambiente de programação do Arduino um conjunto bastante completo de funções para gerenciar a comunicação serial. A tabela a seguir mostra as principais funções.

Função	Descrição
Serial.available()	Retorna o número de bytes (caracteres) disponíveis para leitura da porta serial.
Serial.availableForWrite()	Retorna o número de bytes (caracteres) livres no buffer de transmissão serial.
Serial.begin()	Configura a taxa de transferência para transmissão serial.
Serial.end()	Desativa a comunicação serial.
Serial.find()	Serial.find() lê dados da serial até a string especificada ser encontrada.
Serial.findUntil()	Lê dados até uma string especificada ou um terminador serem encontrados.
Serial.flush()	Espera a transmissão de dados seriais enviados terminar.
Serial.parseFloat()	Retorna o primeiro número válido de ponto flutuante da serial.
Serial.parseInt()	Procura o próximo inteiro válido no buffer de recebimento serial.
Serial.peek()	Retorna o próximo byte de dados seriais recebidos sem o remover da serial.
Serial.print()	Imprime dados na porta serial em como texto ASCII.
Serial.println()	Imprime dados na porta serial como texto ASCII e pula uma linha.
Serial.read()	Lê dados recebidos na porta serial.

Serial.readBytes()	Lê caracteres da porta serial e os move para um buffer.
Serial.readBytesUntil()	Lê caracteres da porta serial e os move para um buffer de forma controlada.
Serial.readString()	Lê caracteres do buffer serial e os move para uma String.
Serial.readStringUntil()	Lê caracteres do buffer serial e os move para uma String de forma controlada.
Serial.setTimeout()	Configura o número máximo de milissegundos a se esperar por dados seriais.
Serial.write()	Escreve dados binários na porta serial.
Serial.serialEvent()	Chamada quando dados estão disponíveis no buffer serial.

A seguir é apresentado um exemplo de comunicação serial entre dois Arduinos. A figura a seguir apresenta o circuito utilizado.



Neste circuito temos 2 botões conectados ao Arduino 1 e um LED conectado ao Arduino 2. O objetivo é ligar e desligar o LED do Arduino 2 através dos botões do Arduino 1, usando comunicação serial.

A seguir é apresentado o programa para o Arduino 1.

```
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  if (digitalRead(6)==HIGH)
  {
    Serial.write('L');
    delay(50);
  }
  if (digitalRead(7)==HIGH)
  {
    Serial.write('D');
    delay(50);
  }
}
```

Neste programa, a comunicação serial é iniciada com uma taxa de transmissão de 9600 bits por segundo. O funcionamento do programa é simples, quando o botão conectado ao pino 6 é pressionado, a letra “L” é enviada pela serial, indicando que o LED do Arduino 2 deve ser ligado. Quando o botão conectado ao pino 7 é pressionado, a letra “D” é enviada pela serial, indicando que o LED do Arduino 2 deve ser desligado.

A seguir é apresentado o programa do Arduino 2.

```
int recebido = 0;

void setup()
{
  pinMode(6, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  if (Serial.available() > 0)
  {
    // lê o dado recebido:
    recebido = Serial.read();

    if(recebido=='L')
    {
```

```
// liga o LED
digitalWrite(6,HIGH);
}
else
{
//desliga o LED
digitalWrite(6,LOW);
}
}
}
```

Este programa também inicia a comunicação serial com uma taxa de transmissão de 9600 bits por segundo. O funcionamento do programa é o seguinte, quando um dado é recebido pela serial, a função `Serial.available()` se torna verdadeira. Então o dado recebido é lido e comparado com a letra “L”, se a comparação for verdadeira o LED conectado ao pino 6 é ligado, senão o LED é desligado.

13.6 Conversão USB - Serial

Está cada vez mais difícil encontrar computadores com portas seriais instaladas de fábrica. Atualmente o padrão para portas de comunicação nos computadores é o USB. Para contornar esta dificuldade é cada vez mais comum utilizarmos conversores USB-Serial. Estes dispositivos servem como uma ponte, interligando dispositivos com comunicação serial a dispositivos com comunicação USB. O próprio Arduino possui um conversor deste tipo, ou seja, a comunicação realizada entre o Arduino e o computador é USB, mas utiliza-se de um conversor USB-Serial.

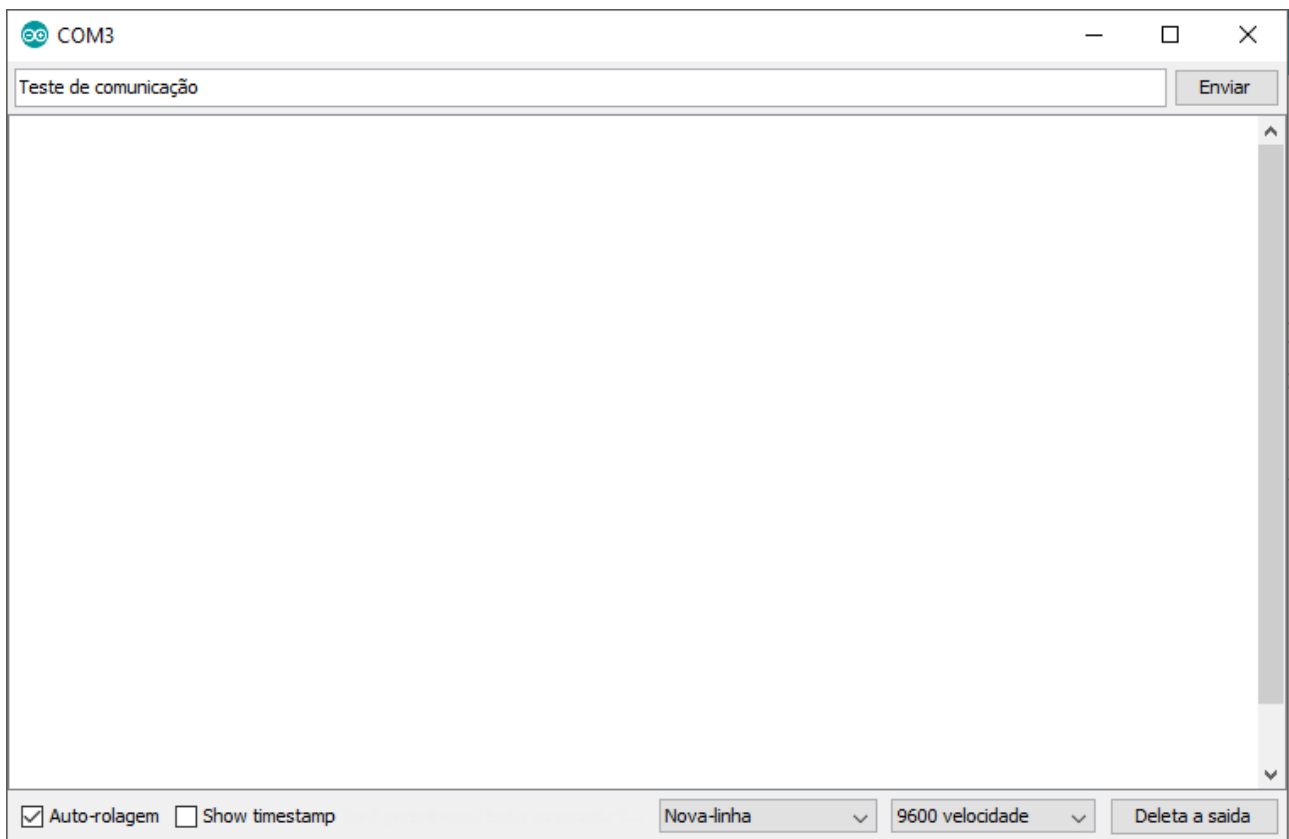
13.6.1 Comunicação Serial Através da porta USB do Arduino

A porta USB do Arduino não serve apenas para a programação do mesmo, ela também serve para comunicar o Arduino com o computador. A porta serial do Arduino está conectada a um conversor USB-Serial, e assim, quando o Arduino está conectado ao computador, todas as informações que são transmitidas pela porta serial do Arduino são enviadas ao computador. O caminho contrário também funciona.

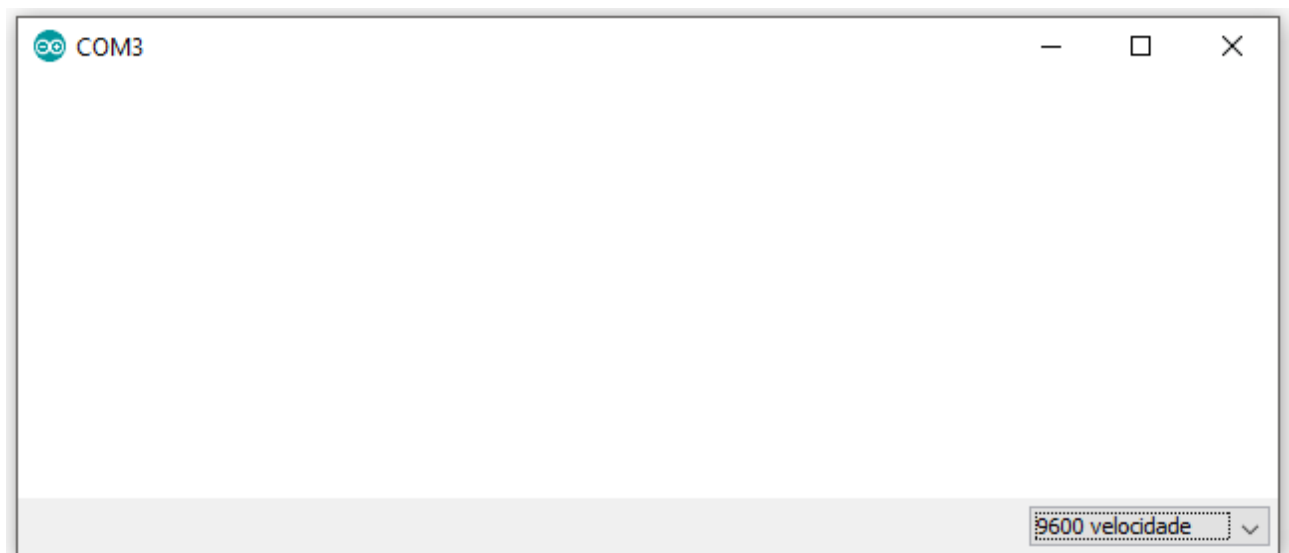
Esta característica do Arduino é muito útil, pois assim é possível trocar dados com o computador sem nenhum componente adicional.

O próprio ambiente de programação do Arduino fornece duas ferramentas muito úteis para este tipo de comunicação. A primeira é o Monitor Serial, que serve para mandar e receber mensagens na forma de texto para e do Arduino.

A figura a seguir mostra a interface desta ferramenta.



A segunda ferramenta é o Plotter Serial, que é responsável por desenhar gráficos com os dados recebidos pela serial. A figura a seguir apresenta esta ferramenta.



A seguir é apresentado um exemplo de programa que envia uma sequência de números pela serial. Este programa pode ser utilizado para verificar o funcionamento das ferramentas apresentadas.

```
int cont=0;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.println(cont);
  delay(500);
  cont++;
}
```

13.6.2 Conversores USB-Serial comerciais

Uma alternativa para estabelecer uma comunicação serial com um computador que não possui porta serial é a utilização de um conversor USB-Serial comercial. Existem inúmeros modelos no mercado, a figura a seguir apresenta alguns deles.



13.7 Conclusão

Nesta aula foi tratado sobre comunicação serial, tanto seus fundamentos como sua implementação no Arduino. Maiores informações sobre a comunicação serial no Arduino podem ser obtidas em: <https://www.arduino.cc/reference/en/language/functions/communication/serial/>.

Na próxima aula trataremos do projeto de circuitos e da automatização de processos utilizando microcontroladores.

13.8 Exercício

- 1) Implemente o exemplo de programa que utiliza 2 arduinos para fazer a comunicação serial no TinkerCad e verifique seu funcionamento.
- 2) Implemente junto com um colega o exemplo de programa que utiliza dois Arduinos para realizar a comunicação serial. Um aluno deve implementar o transmissor e o outro aluno deve implementar o receptor. Os dois circuitos devem ser interligados para verificar seu funcionamento.
- 3) Implemente o exemplo de programa que envia uma sequência de números pela serial no TinkerCad e verifique seu funcionamento utilizando as ferramentas Monitor Serial e Plotter Serial.
- 4) Implemente o exemplo de programa que envia uma sequência de números pela serial no Arduino e verifique seu funcionamento utilizando as ferramentas Monitor Serial e Plotter Serial.

AULA 14 - CIRCUITOS COM MICROCONTROLADORES

Estudo de circuitos com microcontroladores na automação de processos

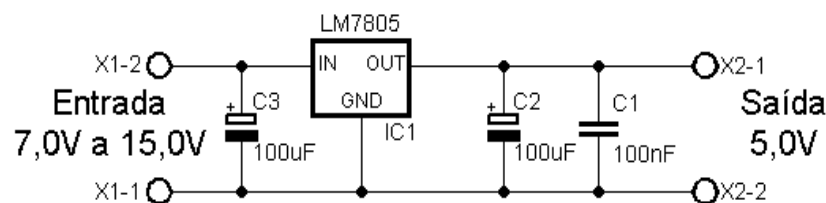
14.1 Objetivo:

Nas aulas anteriores o hardware utilizado foi o Arduino, porém, para projetos profissionais é necessário desenvolvermos nossos próprios circuitos utilizando diretamente os microcontroladores. Assim, o objetivo desta aula é apresentar alguns circuitos utilizando microcontroladores, de forma que se possa compreender como os microcontroladores são conectados ao restante do circuito. As principais conexões são entre o microcontrolador e a fonte de alimentação e entre o microcontrolador e o circuito clock.

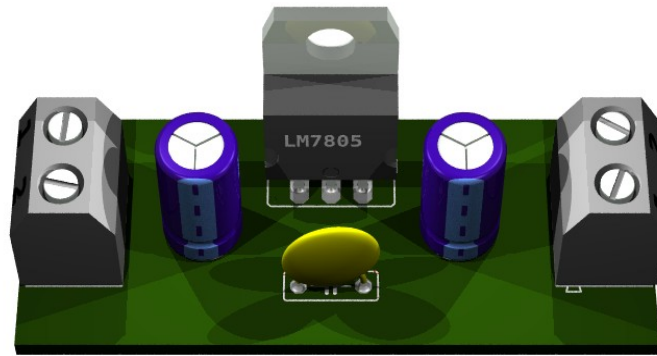
14.2 A fonte de alimentação do microcontrolador

Para que um microcontrolador possa operar corretamente ele deve ser conectado a uma fonte de alimentação adequada. Esta fonte de alimentação deve ter uma tensão apropriada ao microcontrolador e aos periféricos utilizados. A família de microcontroladores utilizada como exemplo neste material é a família AVR. Dentre os microcontroladores desta família utilizaremos o ATmega8, que aceita tensões de alimentação de 2,8 a 5,0V, porém na grande maioria das aplicações a tensão utilizada é de 5,0V. Isso quando a alimentação do circuito não é feita através de baterias, que normalmente é um caso a parte e exige circuitos apropriados.

Na grande maioria das aplicações de microcontroladores a tensão de operação é de 5,0V, e para que esta tensão permaneça estável e livre de ruídos é apropriado que se utilize um circuito regulador de tensão protegido por filtros capacitivos. A figura a seguir apresenta um circuito para esta função onde o regulador de tensão é o LM7805.

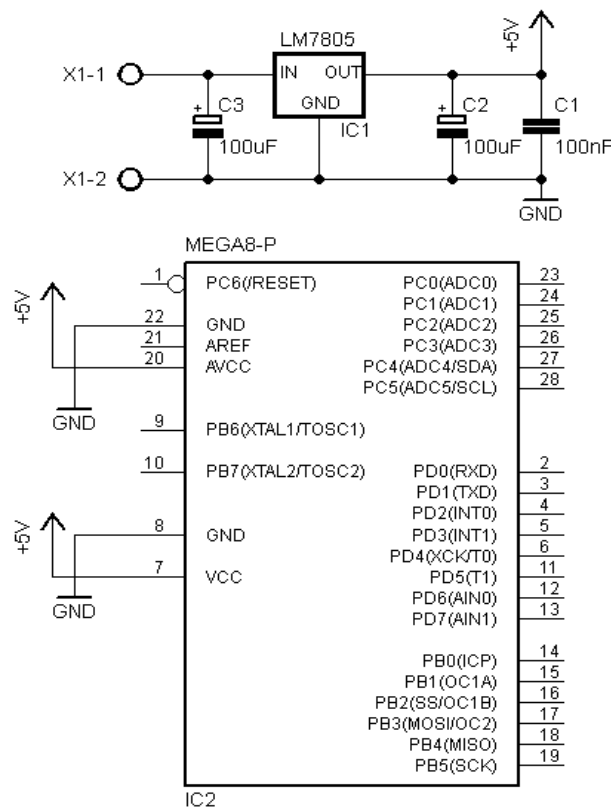


A figura a seguir apresenta este circuito montado em uma placa de circuitos.



14.3 A conexão do microcontrolador com a fonte

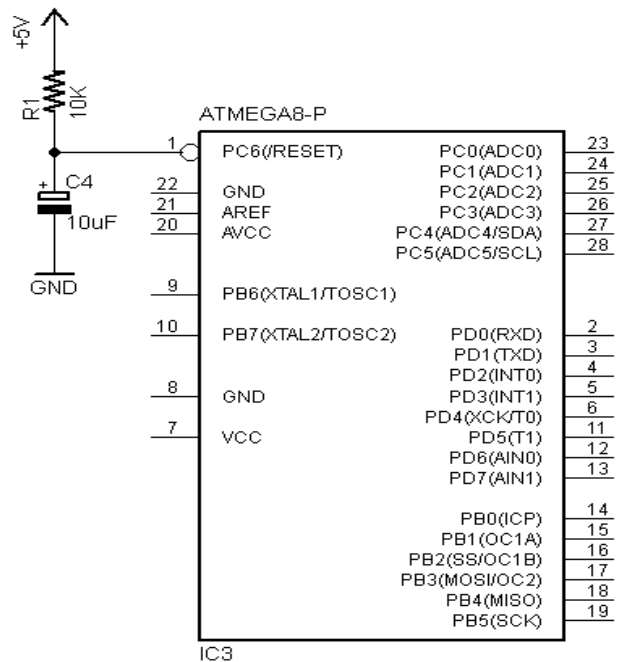
Os microcontroladores da família AVR possuem duas entradas de alimentação distintas, uma para os circuitos digitais e outra para o circuito analógico do conversor analógico digital. A figura a seguir apresenta a conexão dos pinos 7 e 8 do microcontrolador ATmega8 a fonte, correspondendo a alimentação dos circuitos digitais, e dos pinos 20 e 22 correspondentes a parte analógica do conversor A/D.



A alimentação do conversor A/D é separada da alimentação do resto do chip para que se possa construir filtros contra ruído específicos para o conversor A/D, isso é necessário quando se necessita de grande precisão nas leituras analógicas.

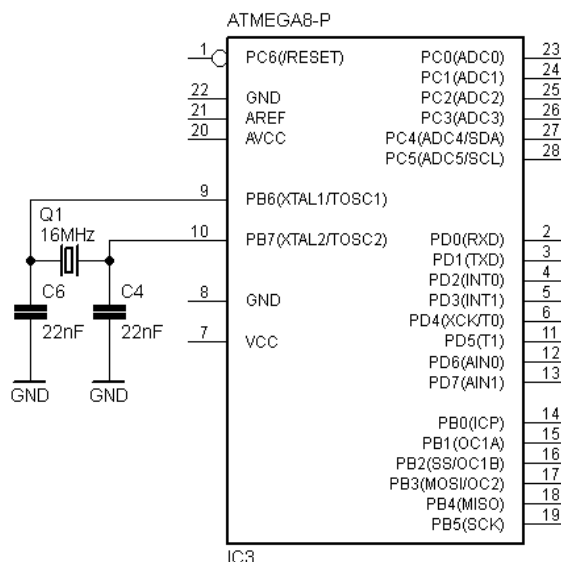
14.4 O circuito de reset

A maioria dos circuitos que utilizam algum tipo de processador tem um sinal de reset, utilizado para reiniciar o sistema. Os microcontroladores não são diferentes. O microcontrolador ATmega8 tem como função principal do pino 1 o reset. Normalmente não é necessário adicionar um botão de reset ao nosso projeto, porém o pino de reset deve ser conectado a um resistor e a um capacitor conforme a figura ao lado.



14.5 O circuito de clock

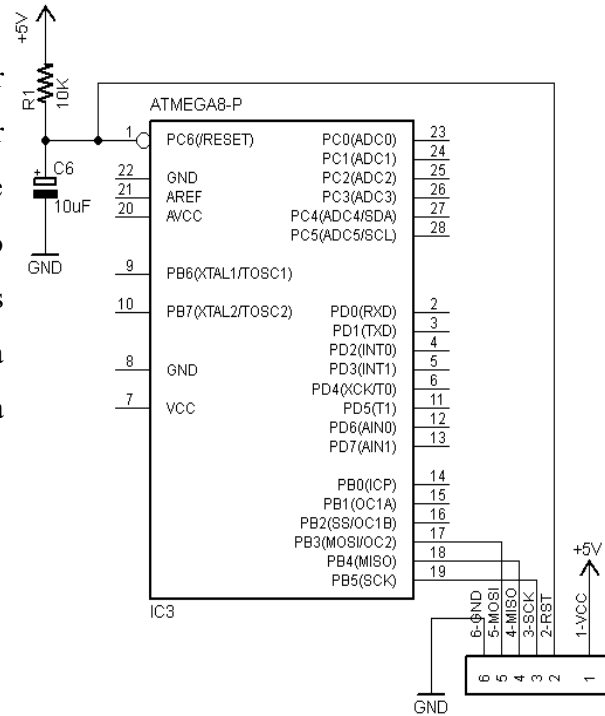
Todos os circuitos microprocessados, inclusive os microcontroladores necessitam de um circuito de relógio, ou circuito de clock. O microcontrolador ATmega8 possui um circuito interno de clock, que pode funcionar de duas formas. A primeira é através de um oscilador RC interno, o que simplifica o circuito porém não oferece precisão. A segunda é através de um cristal externo, o que encarece e complica o circuito, porém oferece grande precisão. A figura a seguir mostra como conectar um cristal ao microcontrolador.



As trilhas entre o microcontrolador, o cristal e os dois capacitores de 22nF devem ser o mais curtas possível.

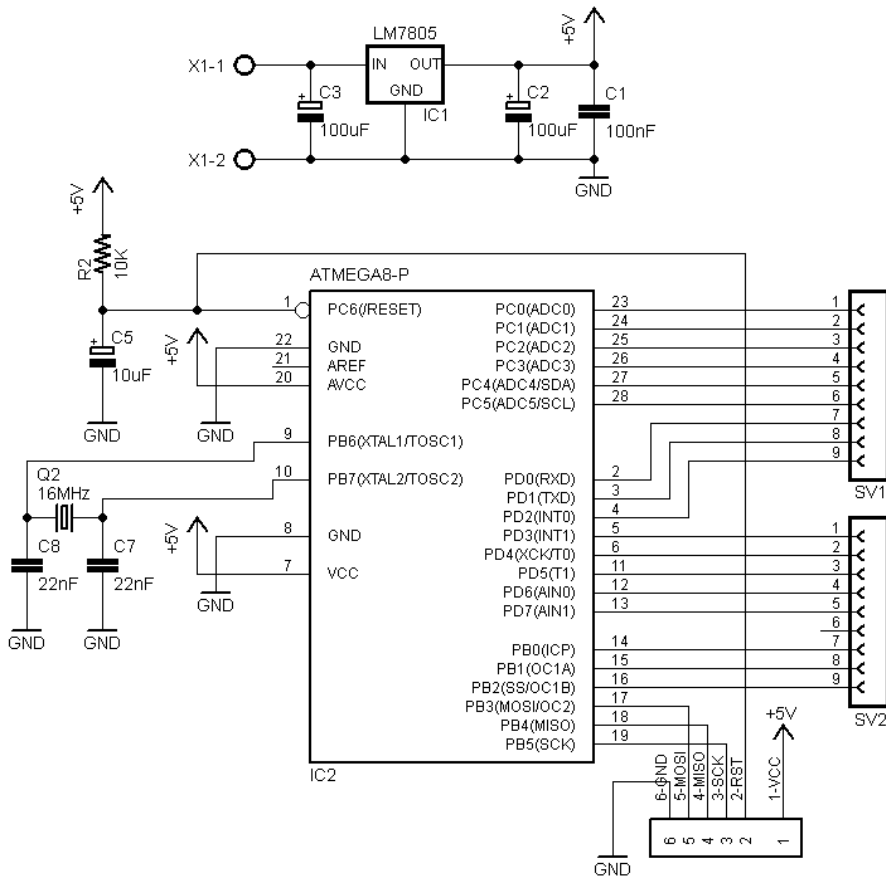
14.6 Conector de programação

Os microcontroladores necessitam ser programados para que possam operar corretamente. Isso pode ser feito fora da placa de circuito ou na própria placa onde o microcontrolador vai operar. Se desejarmos incluir um conector de programação em nossa placa, existem inúmeras possibilidades. A figura ao lado apresenta um exemplo.

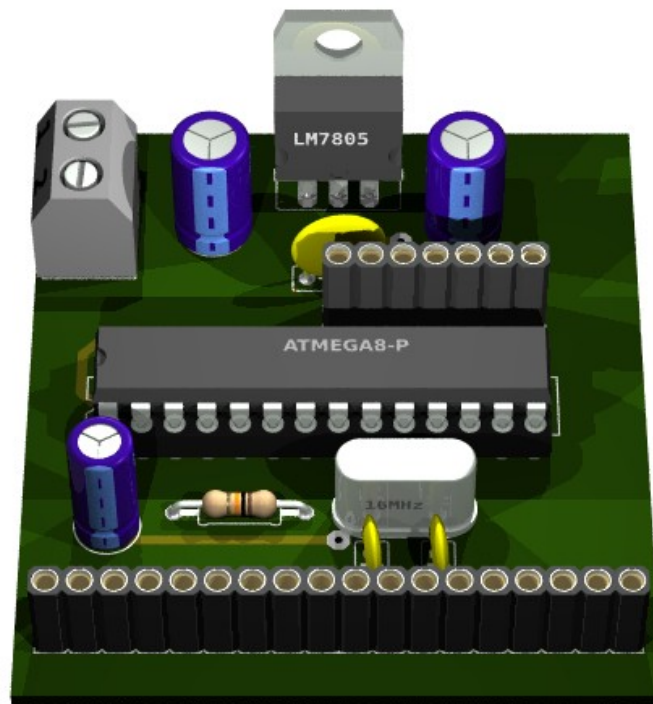


14.7 Circuito completo

Unindo todos os circuitos necessários para o funcionamento de um microcontrolador temos o circuito a seguir.



Foram adicionados dois conectores para interligar os pinos de entrada e saída na figura anterior. A imagem final da placa fica como na figura a seguir.



14.8 O circuito programador

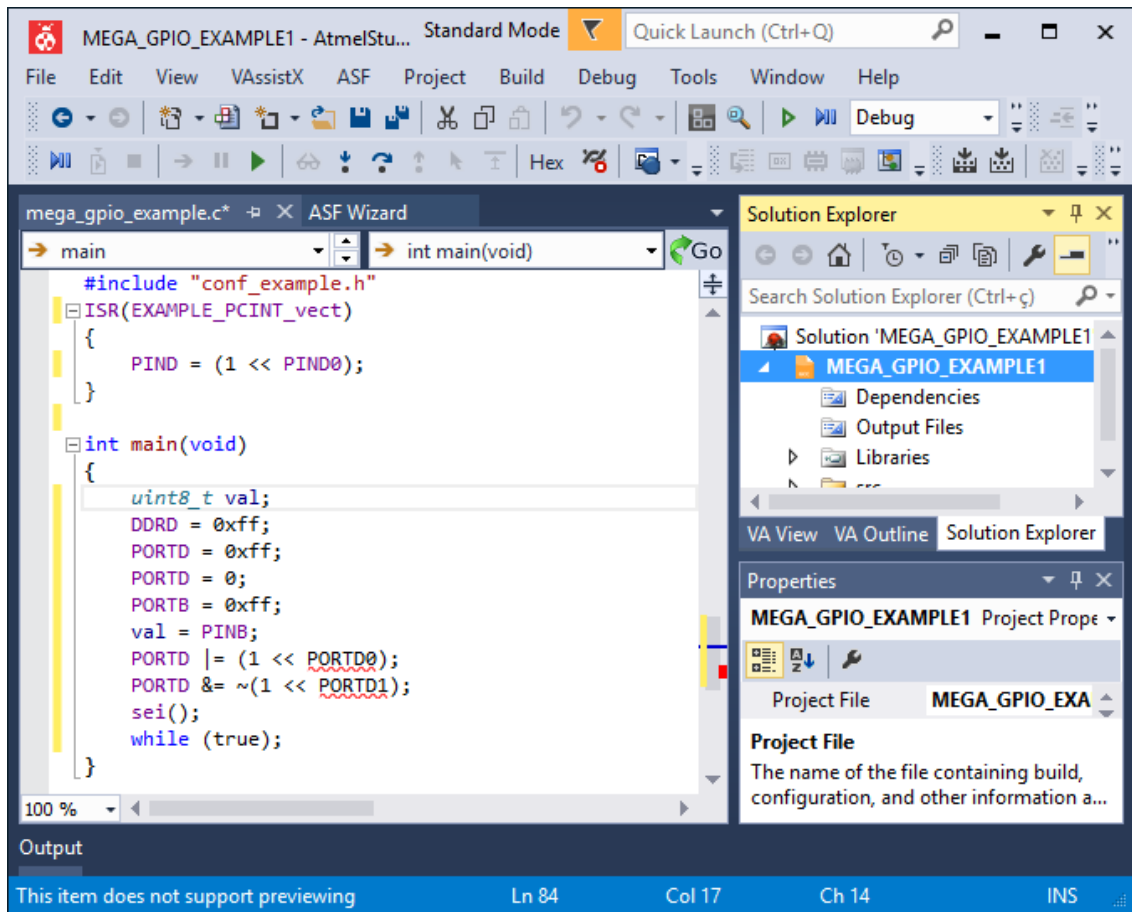
Apesar de não fazer parte do circuito de um microcontrolador o circuito de programação é necessário para transferir o programa do computador para o microcontrolador. Existem vários circuitos para este fim.

Uma alternativa interessante é construirmos nosso próprio gravador de microcontroladores. Um gravador fácil de ser construído é chamado USBASP. Mais informações sobre o USBASP podem ser obtidas no site: <http://www.fischl.de/usbasp/>

Para utilizarmos este circuito é necessário um programa que rode no computador. Um exemplo é o AVRdude, que pode ser baixado em <http://blog.zakkemble.net/avrdude-a-gui-for-avrdude/>

O circuito do programador USBASP é bastante simples, e utiliza um microcontrolador ATmega8 para receber os sinais da USB do computador e gravar o microcontrolador.

A figura a seguir apresenta o circuito eletrônico deste programador.



14.10 Automatização de Processos utilizando Microcontroladores

Levando em consideração o que foi visto nas aulas anteriores, e considerando também o que foi visto nesta aula é possível concluir que os microcontroladores são apropriados para a automatização de processos.

Algumas características dos microcontroladores, como entradas analógicas e saídas PWM facilitam sua utilização no controle e automatização de processos. O baixo custo deste tipo de componente também é um facilitador da sua utilização neste tipo de aplicações.

14.11 Conclusão

Nesta aula foram apresentadas formas alternativas de utilização dos microcontroladores, sem a necessidade de utilizar o Arduino. A plataforma Arduino como um todo é uma ferramenta bastante didática, tem um papel importante no ensino de microcontroladores e em aplicações de hobby. Porém, para aplicações profissionais é necessário um hardware mais robusto, assim o desenvolvimento de circuitos específicos para cada tipo de aplicação, utilizando diretamente os microcontroladores apropriados é necessário.