



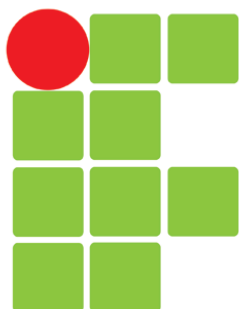
MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA CATARINENSE
CÂMPUS LUZERNA

Engenharia de controle e Automação

Microcontroladores

(RESET) PC6	<input type="checkbox"/>	1	28	<input type="checkbox"/>	PC5 (ADC5/SCL)
(RXD) PD0	<input type="checkbox"/>	2	27	<input type="checkbox"/>	PC4 (ADC4/SDA)
(TXD) PD1	<input type="checkbox"/>	3	26	<input type="checkbox"/>	PC3 (ADC3)
(INT0) PD2	<input type="checkbox"/>	4	25	<input type="checkbox"/>	PC2 (ADC2)
(INT1) PD3	<input type="checkbox"/>	5	24	<input type="checkbox"/>	PC1 (ADC1)
(XCK/T0) PD4	<input type="checkbox"/>	6	23	<input type="checkbox"/>	PC0 (ADC0)
VCC	<input type="checkbox"/>	7	22	<input type="checkbox"/>	GND
GND	<input type="checkbox"/>	8	21	<input type="checkbox"/>	AREF
(XTAL1/TOSC1) PB6	<input type="checkbox"/>	9	20	<input type="checkbox"/>	AVCC
(XTAL2/TOSC2) PB7	<input type="checkbox"/>	10	19	<input type="checkbox"/>	PB5 (SCK)
(T1) PD5	<input type="checkbox"/>	11	18	<input type="checkbox"/>	PB4 (MISO)
(AIN0) PD6	<input type="checkbox"/>	12	17	<input type="checkbox"/>	PB3 (MOSI/OC2)
(AIN1) PD7	<input type="checkbox"/>	13	16	<input type="checkbox"/>	PB2 (SS/OC1B)
(ICP1) PB0	<input type="checkbox"/>	14	15	<input type="checkbox"/>	PB1 (OC1A)

Prof: Ricardo Kerschbaumer



ALUNO: _____

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

Microcontroladores

Este material foi desenvolvido pelo professor Ricardo Kerschbaumer para ser utilizado na componente curricular de Microcontroladores do curso de Engenharia de Controle e Automação. 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.

Sumário

Aula 1 - Microcontroladores e Memórias.....	10
1.1 Objetivo:.....	10
1.2 O que são microcontroladores.....	10
1.3 Aplicações dos microcontroladores.....	11
1.4 Registradores.....	12
1.5 Memórias.....	14
1.6 Memória RAM (Memória de Acesso Aleatório).....	15
1.7 Memórias ROM.....	16
1.8 Memórias ROM Programáveis (Prom's).....	16
1.9 Memórias ROM Programáveis e Apagáveis (EPROM's, EEPROMS's e FLASH).....	16
1.10 Associação de memórias.....	18
1.11 Microcontroladores e memórias.....	19
1.12 Exercícios.....	19
Aula 2 -ARQUITETURA DOS MICROCONTROLADORES.....	20
2.1 Objetivo:.....	20
2.2 Arquitetura interna dos microprocessadores.....	20
2.3 Registradores de propósito geral.....	20
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.....	21
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.....	22
2.13 Sinais de Controle entre microprocessador e Memória.....	24
2.14 Microcontroladores e Microprocessadores.....	24
2.15 Exercícios.....	26
Aula 3 - A FAMÍLIA DE MICROCONTROLADORES AVR	27
3.1 Objetivo:.....	27
3.2 Introdução a família AVR.....	27
3.3 Desempenho.....	27
3.4 Baixo consumo de energia.....	28
3.5 Variedade de dispositivos.....	29
3.6 Variedade de embalagens.....	29
3.7 Aplicações específicas.....	29
3.8 Programação.....	30
3.9 Ferramentas de desenvolvimento	30
3.10 Suporte.....	31
3.11 A CPU AVR.....	31
3.11.1 Arquitetura da CPU AVR.....	31

3.11.2 Unidade Lógica e Aritmética – ULA.....	32
3.11.3 Registrador de Status.....	32
3.11.4 Registradores de uso geral.....	34
3.11.5 Ponteiro da pilha.....	35
3.11.6 Tempo de execução das instruções.....	36
3.11.7 O reset e as interrupções.....	37
3.11.8 Tempo de resposta das interrupções.....	37
3.12 O microcontrolador ATmega8.....	37
3.13 Função dos pinos.....	39
3.14 Características elétricas do ATmega8.....	39
3.15 Consumo de corrente da fonte em relação à frequência de operação:.....	39
3.16 Exercícios.....	41
Aula 4 - A linguagem assembler.....	42
4.1 Objetivo:.....	42
4.2 A linguagem assembler para microcontroladores AVR.....	42
4.3 Características de um programa em linguagem assembler.....	42
4.4 Diretivas do assembler.....	43
4.5 Descrição das diretivas.....	44
4.5.1 BYTE - Reserva um byte para uma variável.....	44
4.5.2 CSEG - Segmento de código.....	44
4.5.3 DB - Define constantes do tipo byte.....	44
4.5.4 DEF - Define um nome simbólico para um registrador.....	45
4.5.5 DEVICE - Define qual o dispositivo usado pelo assembler.....	45
4.5.6 DSEG - Segmento de dados.....	46
4.5.7 DW - Define constantes do tipo word (16 bits).....	46
4.5.8 ENDMACRO - Final de macro.....	47
4.5.9 EQU - Define um símbolo igual a uma expressão.....	47
4.5.10 ESEG - Segmento da EEPROM.....	47
4.5.11 EXIT - Sai do arquivo.....	48
4.5.12 INCLUDE - Lê fonte de outro arquivo (inclui).....	48
4.5.13 LIST - Liga o gerador de arquivo de lista.....	48
4.5.14 LISTMAC - Liga o expensor de macro.....	49
4.5.15 MACRO - Inicia uma macro.....	49
4.5.16 NOLIST - Desliga o gerador de arquivo de lista.....	50
4.5.17 ORG - Define o início do programa na memória.....	50
4.5.18 SET - Define um símbolo para uma expressão.....	51
4.6 Expressões.....	51
4.6.1 Operandos.....	51
4.6.2 Funções.....	51
4.6.3 Operadores.....	52
4.6.4 Operação de negação lógica	52
4.6.5 Operação de negação bit a bit.....	52
4.6.6 Sinal de menos unário.....	52

4.6.7 Multiplicação	52
4.6.8 Divisão.....	53
4.6.9 Adição.....	53
4.6.10 Subtração.....	53
4.6.11 Deslocamento para a esquerda.....	53
4.6.12 Deslocamento para a direita.....	53
4.6.13 Menor que.....	53
4.6.14 Menor ou igual que.....	54
4.6.15 Maior que.....	54
4.6.16 Maior ou igual que.....	54
4.6.17 Igual.....	54
4.6.18 Diferente.....	54
4.6.19 Operação e bit a bit.....	54
4.6.20 Operação ou exclusivo bit a bit.....	55
4.6.21 Operação ou bit a bit.....	55
4.6.22 Operação Logica e.....	55
4.6.23 Operação Logica ou.....	55
4.7 Exercícios.....	56
Aula 5 - O AMBIENTE AVRStudio.....	57
5.1 Objetivo.....	57
5.2 O software AVR Studio®.....	57
5.3 Criando projetos.....	57
5.4 Ambiente de trabalho.....	60
5.5 Exercícios.....	64
Aula 6 -DESENVOLVIMENTO DE PROGRAMAS EM C.....	66
6.1 Objetivo:.....	66
6.2 Características da programação de microcontroladores.....	66
6.3 Modelo de programa.....	66
6.4 Uma pequena revisão.....	67
6.5 Declaração de variáveis.....	67
6.6 Operadores.....	67
6.7 Estruturas de controle.....	68
6.8 if.....	68
6.9 If / else.....	69
6.10 while.....	69
6.11 for.....	70
6.12 Sub-rotinas.....	70
6.13 Exercícios.....	71
Aula 7 - ENTRADAS E SAÍDAS DIGITAIS.....	72
7.1 Objetivo:.....	72
7.2 Estrutura interna das entradas e saídas na família AVR.....	72
7.3 Registradores de controle dos pinos de entrada e saída.....	73
7.4 Alteração de um pino individual de saída.....	73

7.5	Leitura de um pino de entrada.....	74
7.6	Registradores de controle das portas.....	74
7.7	Interface entre o microcontrolador e os circuitos de entrada digital.....	74
7.8	Conexão de reles.....	76
7.9	Conexão de um motor utilizando transistor mosfet.....	76
7.10	Conexão de lâmpada de sinalização.....	76
7.11	Conexão de pequenos motores.....	76
7.12	Conexão de motores de passo	77
7.13	Conexão de motores cc com reversão.....	78
7.14	Conexão de displays de LED.....	78
7.15	Circuitos especializados de saída.....	78
7.16	Exercícios.....	79
Aula 8	- AUTOMAÇÃO USANDO LINGUAGEM C.....	80
8.1	Objetivo:.....	80
8.2	Exemplo de controle de nível.....	80
8.3	Exemplo de controle de um robô.....	81
8.4	Exercícios.....	84
Aula 9	-Interrupções nos Microcontroladores.....	85
9.1	Objetivo:.....	85
9.2	O que são interrupções.....	85
9.3	Exercícios.....	89
Aula 10	- Temporizadores e contadores.....	90
10.1	Objetivo:.....	90
10.2	Introdução.....	90
10.3	Temporizador / contador 0.....	90
10.4	Exercícios.....	95
Aula 11	- Temporizadores Avançados.....	96
11.1	Objetivo:.....	96
11.2	Introdução.....	96
11.3	Temporizador / contador 1.....	96
11.4	Registradores de controle do temporizador / contador 1.....	100
11.5	Exercícios.....	105
Aula 12	- Conversor Analógico Digital.....	106
12.1	Objetivo:.....	106
12.2	Introdução.....	106
12.3	O hardware do conversor analógico digital.....	106
12.4	Prescaler do conversor A/D.....	108
12.5	Resultados das conversões.....	108
12.6	Registradores de controle do conversor A/D.....	109
12.7	Exemplo de programa utilizando o conversor A/D.....	111
12.8	Exercícios.....	113
Aula 13	- Comunicação Serial na Família AVR.....	114
13.1	Objetivo:.....	114

13.2 A comunicação serial.....	114
13.3 O padrão RS232.....	115
13.4 O padrão RS485.....	117
13.5 Comunicação serial nos microcontroladores.....	118
13.5.1 O padrão RS232 nos microcontroladores.....	119
13.5.2 O padrão RS485 nos microcontroladores.....	120
13.5.3 A configuração do microcontrolador.....	121
13.6 Exercícios.....	127
Aula 14 - Ladder nos Microcontroladores.....	128
14.1 Objetivo:.....	128
14.2 O programa LDmicro.....	128
14.3 Iniciando um Programa em Ladder.....	129
14.4 Programando no Ldmicro.....	130
14.5 Instruções do Ldmicro.....	131
14.5.1 Contato Normalmente Aberto.....	131
14.5.2 Contato Normalmente Fechado.....	131
14.5.3 Bobina Normal.....	132
14.5.4 Bobina Negada.....	132
14.5.5 Bobina Ativar.....	132
14.5.6 Bobina Desativar.....	133
14.5.7 Temporizador para Ligar.....	133
14.5.8 Temporizador para Desligar.....	134
14.5.9 Temporizador Retentivo para Ligar.....	134
14.5.10 Reinicia Contador / Temporizador.....	135
14.5.11 Detecta Borda de Subida.....	135
14.5.12 Detecta Borda de Descida.....	135
14.5.13 Circuito Aberto e Curto Circuito.....	136
14.5.14 Rele de Controle Mestre.....	136
14.5.15 Mover.....	136
14.5.16 Operações Aritméticas.....	137
14.5.17 Comparações.....	137
14.5.18 Contadores.....	138
14.5.19 Contador Circular.....	138
14.5.20 Deslocar Registro.....	138
14.5.21 Busca na Tabela.....	139
14.5.22 Linearização por Segmentos.....	140
14.5.23 Ler Conversor A/D.....	140
14.5.24 Valor do PWM.....	140
14.5.25 Fazer Permanente.....	141
14.5.26 UART Receber.....	141
14.5.27 UART Enviar.....	142
14.5.28 UART Enviar.....	142
14.6 Definição dos Pinos.....	143

14.7 Simulação do Programa.....	144
14.8 Compilar o Programa.....	145
14.9 Exercícios.....	146
Práticas.....	147
Aula 1 - Circuitos com microcontroladores	148
1.1 Objetivo:.....	148
1.2 A fonte de alimentação do microcontrolador.....	148
1.3 A conexão do microcontrolador com a fonte.....	149
1.4 O circuito de reset	149
1.5 O circuito de clock.....	150
1.6 Conector de programação.....	150
1.7 Circuito completo.....	151
1.8 O circuito programador	152
1.9 Gravação do programa no microcontrolador.....	152
1.10 Exercícios.....	154
Aula 2 - Entradas e saídas Digitais.....	155
2.1 Objetivo:.....	155
2.2 Fundamentação.....	155
2.3 Exercícios.....	157
Aula 3 - Modulação PWM.....	158
3.1 Objetivo:.....	158
3.2 Fundamentação.....	158
3.3 Exercícios.....	160
Aula 4 - Interrupções no Microcontrolador.....	161
4.1 Objetivo:.....	161
4.2 Fundamentação.....	161
4.3 Exercícios.....	163
Aula 5 - Temporizadores e contadores.....	165
5.1 Objetivo:.....	165
5.2 Temporizador / contador 0.....	165
5.3 Exercícios.....	167
Aula 6 - Temporizadores Avançados.....	168
6.1 Objetivo:.....	168
6.2 Fundamentação.....	168
6.3 Exercícios.....	170
Aula 7 - Conversor Analógico Digital.....	171
7.1 Objetivo:.....	171
7.2 Fundamentação.....	171
7.2.1 O hardware do conversor analógico digital.....	171
7.2.2 Prescaler do conversor A/D.....	172
7.2.3 Resultados das conversões.....	172
7.2.4 Registradores de controle do conversor A/D.....	172
7.3 Exercícios.....	175

IFC – Instituto Federal Catarinense	Câmpus Luzerna	Microcontroladores
Aula 8 - Comunicação Serial na Família AVR.....		176
8.1 Objetivo:.....		176
8.2 Fundamentação.....		176
8.3 Exercícios.....		178
Aula 9 - Ladder nos Microcontroladores.....		179
9.1 Objetivo:.....		179
9.2 Fundamentação.....		179
9.3 Exercícios.....		180
Anexo I – Conjunto de instruções.....		181

AULA 1 - MICROCONTROLADORES E MEMÓRIAS

Introdução aos microcontroladores e revisão sobre registradores e memórias

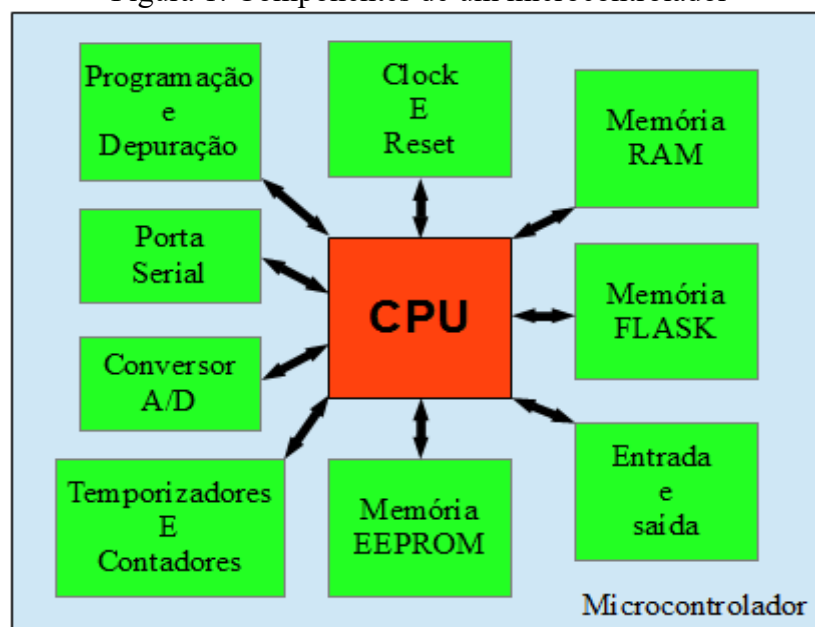
1.1 Objetivo:

O objetivo desta primeira aula é apresentar aos alunos os microcontroladores e revisar alguns tópicos de eletrônica digital, necessários ao estudo de microcontrolador. Ao final desta aula os alunos serão capazes de compreender o funcionamento dos registradores e dos dispositivos eletrônicos de memória, bem como reconhecer os microcontroladores e suas aplicações.

1.2 O que são microcontroladores

Microcontroladores são circuitos integrados que possuem em seu interior todos os componentes necessários ao seu funcionamento dependendo unicamente da fonte de alimentação externa. Pode-se dizer que os microcontroladores são computadores de um único chip. A Figura 1 ilustra os componentes de um microcontrolador típico.

Figura 1: Componentes de um microcontrolador



Um sistema microprocessado é composto por uma unidade central de processamento CPU¹ e um conjunto de periféricos necessários ao seu funcionamento. Dentre estes periféricos podemos destacar a memória de dados, a memória de programa e o circuito de clock². Os microcontroladores diferem dos sistemas tradicionais por já integrarem os seus periféricos dentro do próprio componente.

1 Do inglês Central Processor Unit.

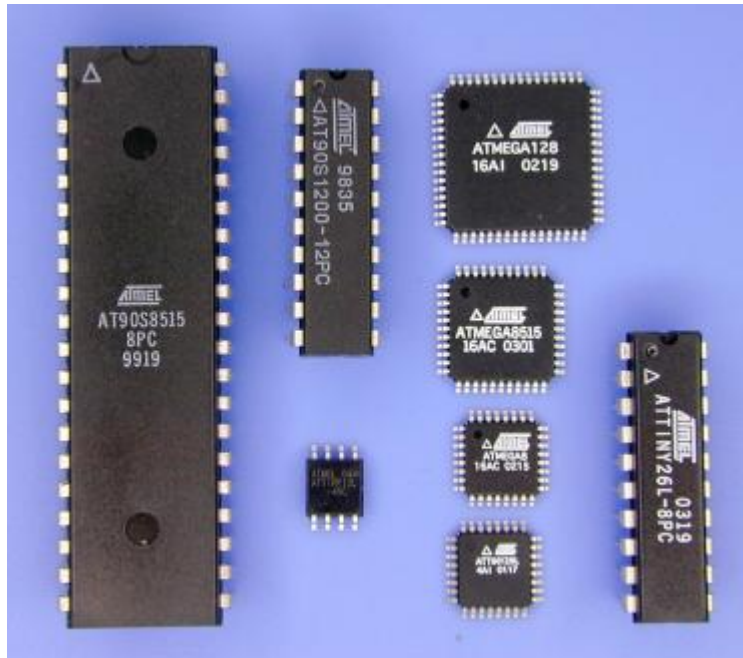
2 Sinal de relógio responsável pela sincronização das operações.

Esta integração é uma das principais vantagens dos microcontroladores, pois contendo todos os periféricos no mesmo componente faz com que sua utilização seja mais fácil e mais barata.

Sistemas microcontrolados não necessitam de muitos componentes, o que torna mais simples a construção das placas de circuito e diminui o custo dos componentes e da produção.

A Figura 2 apresenta algumas embalagens de microcontroladores encontradas no mercado.

Figura 2: Algumas embalagens de microcontroladores



O tipo da embalagem, a velocidade de processamento, a quantidade de memória e os tipos de periféricos variam de modelo para modelo e também entre fabricantes, pois cada microcontrolador é desenvolvido para um tipo de operação.

Os microcontroladores são muito utilizados pela sua versatilidade, pois seu comportamento depende principalmente do software que nele é gravado. Assim um mesmo microcontrolador pode ser utilizado para uma infinidade de aplicações bastando apenas mudar o seu software.

Outra vantagem é a possibilidade de atualização de um produto através da atualização do software do microcontrolador, o que não é possível com circuitos analógicos ou digitais tradicionais.

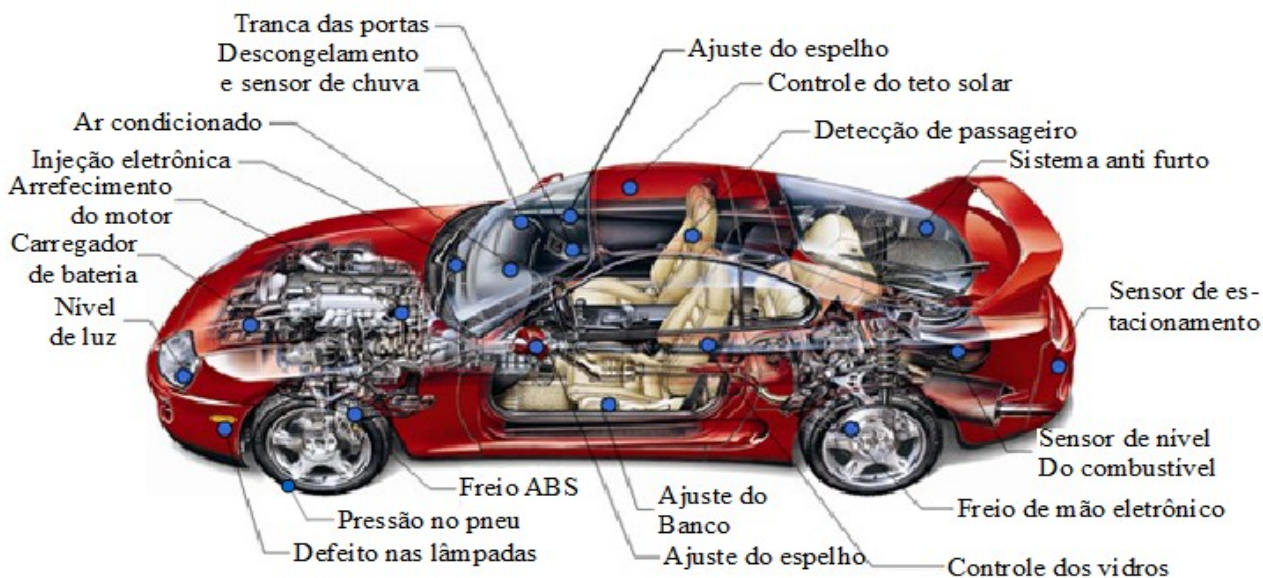
1.3 Aplicações dos microcontroladores

Os microcontroladores são utilizados em praticamente todos os dispositivos eletrônicos digitais que nos cercam, como por exemplo, centrais de alarme, teclados do computador, monitores, discos rígidos de computador, relógio de pulso, máquinas de lavar, forno de micro-ondas, telefones, rádios, televisores, automóveis, aviões, impressoras, marca passos, calculadores, etc. Microcontroladores também são muito utilizados na indústria, como por exemplo nos controladores de processos, sensores inteligentes, inversores, softstarters, interfaces homem máquina,

controladores lógicos programáveis, balanças, indicadores digitais, etc.

Devido a sua grande versatilidade e ao seu baixo custo, praticamente qualquer dispositivo eletrônico pode fazer uso dos microcontroladores. A Figura 3 mostra algumas aplicações dos microcontroladores em um automóvel.

Figura 3: Aplicações do microcontrolador em um automóvel



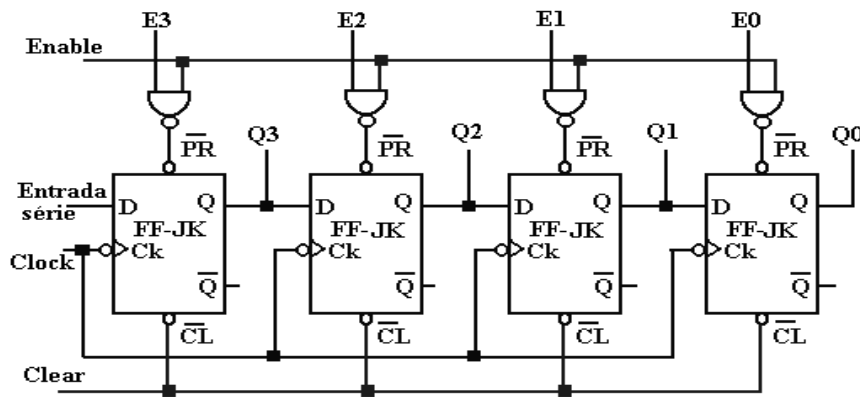
Para que se tenha ideia da importância dos microcontroladores temos, a seguir, uma relação dos principais fabricantes encontrados no mercado. AMCC, Atmel, Cypress MicroSystems, Freescale Semiconductor, Fujitsu, Holtek, Intel, Microchip Technology, National Semiconductor, NXP - Antiga Philips Semiconductors, NEC, Parallax, Inc., Renesas Tech. Corp., STMicroelectronics, Silicon Laboratories, Texas Instruments, Western Design Center, ZiLOG.

É importante salientar que estes são apenas os fabricantes, cada um possui diversas famílias de microcontroladores, cada família com dezenas de componentes diferentes.

1.4 Registradores

Os registradores são a base para o funcionamento de qualquer CPU, e é formado por 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. A Figura 4 apresenta um registrador simples para palavras de quatro bits.

Figura 4: 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 que eles tem de armazenar informações.

A Figura 5 e a Tabela 1 apresentam um registrador comercial, o 74373. Este componente é um registrador de oito bits com entrada e saída paralela.

Figura 5: 74373

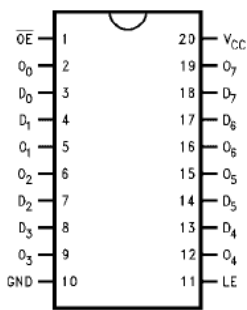


Tabela 1: Características do 74373

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

A Figura 6 mostra a configuração interna do 74373, e a Tabela 2 mostra a tabela verdade do componente, com o índice n de 0 a 7.

Figura 6: Lógica interna do 74373

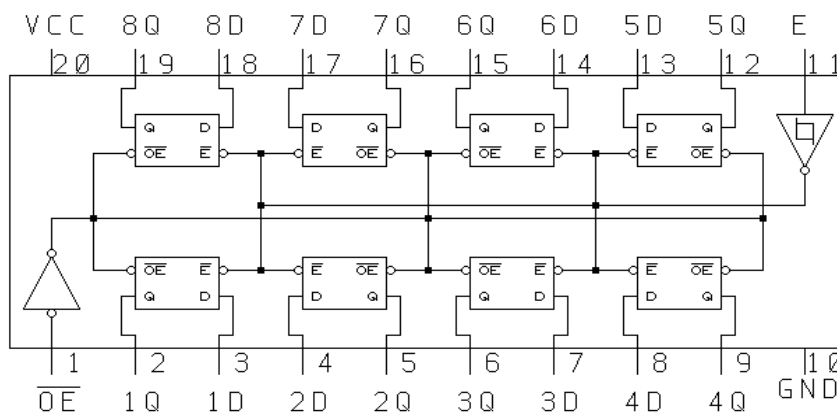


Tabela 2: Tabela verdade do 74373

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

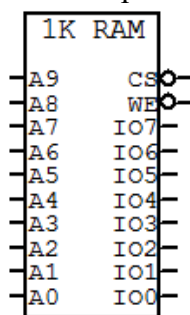
1.5 Memórias

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.

A Figura 7 apresenta um circuito típico de memória, com suas entradas de endereço de A0 a A9, suas entradas e saídas de dados de IO0 a IO7 e seus sinais de controle CS e RW.

Figura 7: Circuito típico de memória



É possível determinar o número de entradas de endereços para um determinado número de registradores a fórmula 1.

$$n_{\text{registradores}} = 2^{n_{\text{linhasendereço}}} \quad (1)$$

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

A Figura 8 apresenta uma memória RAM típica, utilizada em circuitos digitais tradicionais. Seu nome é HM6264, e esta memória é do tipo estática, ou seja, não necessita de *refresh*.

Observando os pinos desta memória notamos a existência de 13 pinos de endereços (A0 a A12) e 8 pinos de dados (I/O0 a I/O7) assim, é possível determinar a capacidade desta memória. Ou seja, 8192 palavras de 8 bits (bytes).

Já a Figura 9 apresenta a embalagem deste componente. Este componente também é encontrado na versão para montagem em superfície (SMD).

Figura 8: Pinagem da memória HM6264

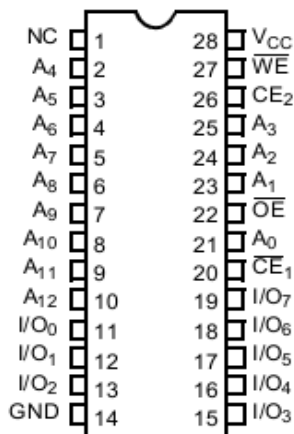
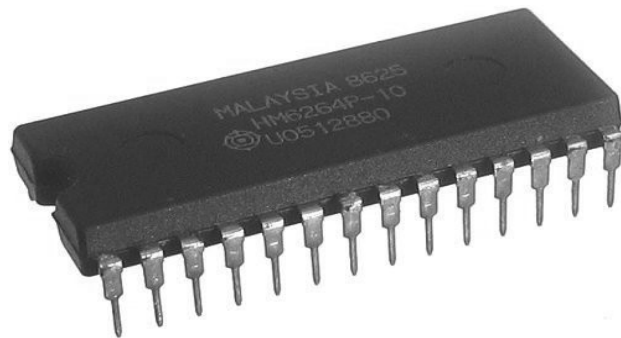


Figura 9: Embalagem da memória HM6264



1.7 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.

1.8 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 conterá “um” ou “zero”.

1.9 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 programação (gravação). A Figura 10 Apresenta a memória EPROM 2764 e a Figura 11 apresenta sua embalagem. É possível observar só centro da embalagem a janela por onde a luz ultravioleta é aplicada.

Figura 10: EPROM 2764

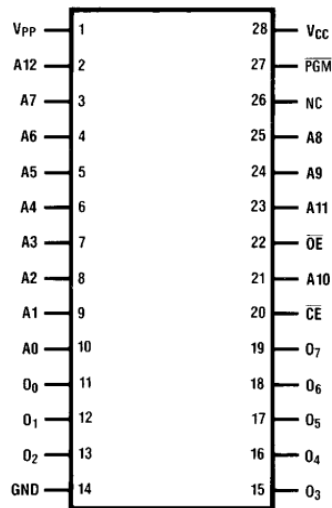


Figura 11: Embalagem da EPROM 2764



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 que é muito maior, e o numero de gravações que se pode fazer que é limitado.

A Figura 12 apresenta a memória 24C64, que é uma memória EEPROM com um tipo de comunicação diferente. Enquanto que nas memórias tradicionais os dados e os endereços são transferidos de forma paralela, nesta memória tudo é transferido de forma serial, através dos pinos SDA e SCL. A transferência é feita através do protocolo I²C. As entradas A0 a A2 servem para determinar o endereço do componente na rede I²C e não o endereço da palavra de memória. A capacidade desta memória é de 8192 palavras de 8 bits, apesar de seu tamanho reduzido. A vantagem é a redução do número de trilhas e pinos utilizados, e a desvantagem é o tempo de acesso que é maior no barramento serial. Já a Figura 13 apresenta a embalagem tradicional deste componente. Também é possível encontrar este componente nas embalagens para montagem em superfície (SMD), o que os torna ainda menores. Estas memórias são muito utilizadas para armazenar as configurações de equipamentos eletrônicos. Por exemplo, quando nosso aparelho de televisão é desligado ele armazena em uma memória deste tipo o volume, o canal, etc. Assim quando o aparelho for novamente ligado as informações podem ser recuperadas.

Figura 12: EEPROM 24C64

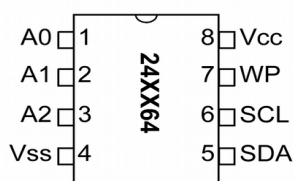


Figura 13: Embalagem da EEPROM 24C64



A Figura 14 apresenta uma memória FLASH de 65536 palavras de 8 bits chamada 29C512. Já a Figura 15 apresenta a sua embalagem. Assim como nos casos anteriores o fabricante fornece este componente para montagem em superfície (SMD).

Figura 14: FLASH 29C512

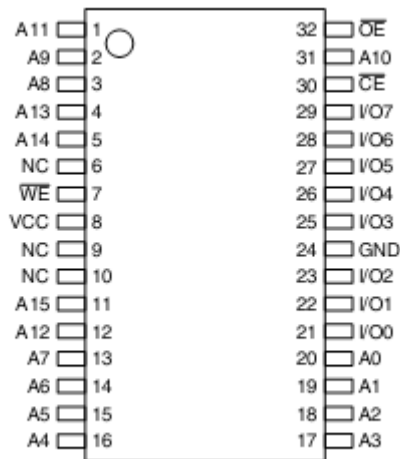


Figura 15: Embalagem FLASH 29C512

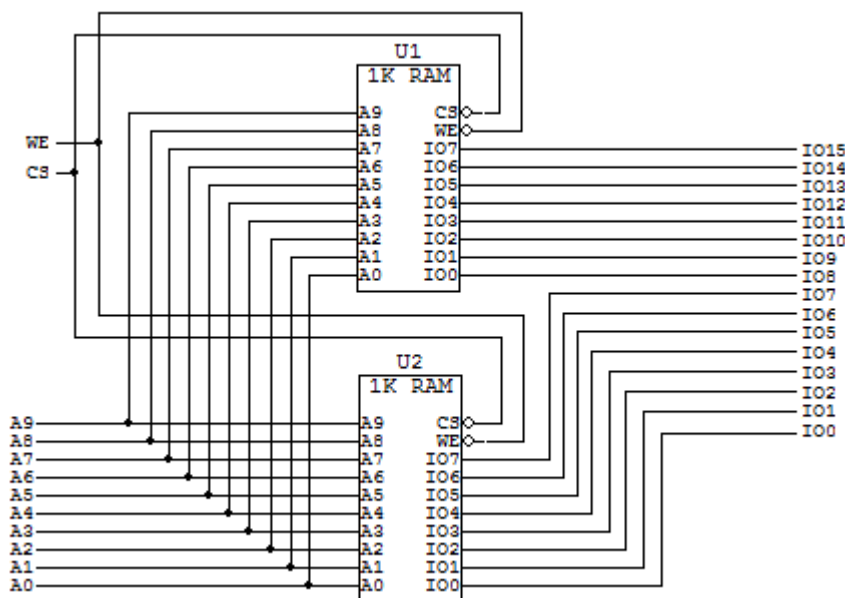


1.10 Associação de memórias

A associação de memórias é utilizada quando o número de palavras 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:

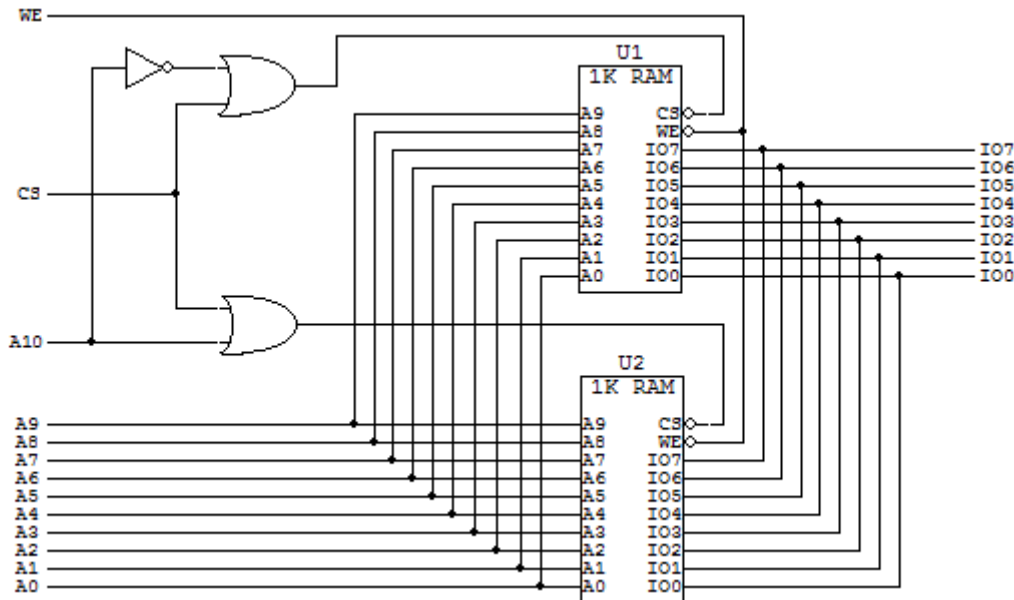
A Figura 16 apresenta a associação de duas memórias de modo a ampliar o número de bits por palavra. É importante observar que o número de entradas de endereço e consequentemente o número de palavras continua o mesmo, o que dobrou foi o número de bits por palavra, criando assim as conexões de IO8 a IO15.

Figura 16: Associação de memórias caso 1



A seguir temos a Figura 16 que apresenta a associação de duas memórias com o objetivo de aumentar o número de palavras e manter o tamanho de cada palavra. É importante observar que o número de conexões de dados não se alterou, porém surgiu mais uma entrada de endereço, chamada A10. Assim o conjunto passa agora a conter o dobro de palavras.

Figura 17: Associação de memórias caso 2



1.11 Microcontroladores e memórias

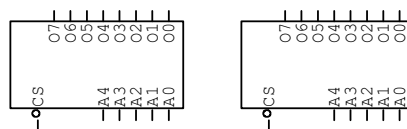
Assim como os registradores as memórias estão intimamente ligadas com os microcontroladores, por este motivo esta aula revisou este assunto.

1.12 Exercícios

1. Para os dois circuitos integrados da figura a seguir determine o número de registradores utilizando a fórmula vista anteriormente.



2. Interconecte as duas memórias a seguir para formar uma única memória com o dobro de registradores. A memória resultante deve ter seis vias de endereços e oito saídas de dados.



AULA 2 -ARQUITETURA DOS MICROCONTROLADORES

Arquiteturas CISC, RISC, Harvard e Von Neumann

2.1 Objetivo:

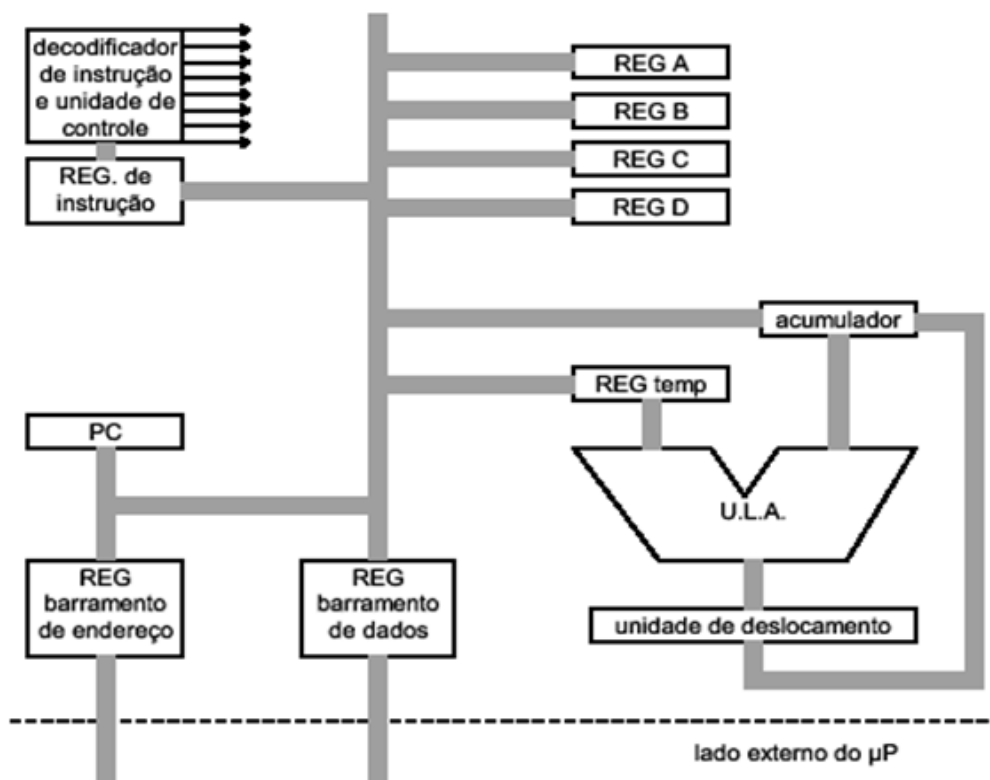
Apresentar aos alunos as arquiteturas internas e externas mais comuns aos microcontroladores. Ao final desta aula os alunos serão capazes de compreender como um microprocessador trabalha internamente, como ele executa seus comandos e o que faz cada um de seus componentes.

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 18 ilustra a arquitetura básica de um microprocessador.

Figura 18: Diagrama de blocos de um microprocessador



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. A função destes registradores é armazenar os dados que estão sendo processados pelo microprocessador.

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 temporariamente 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 recebe o resultado das operações. 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.

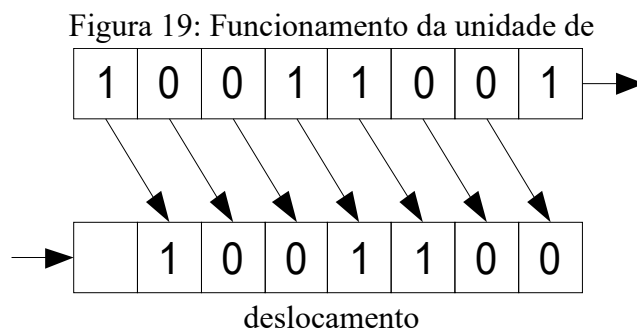
2.8 Registrador de Instrução

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 sequência 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



A Figura 19 mostra o funcionamento da unidade de deslocamento. A unidade de deslocamento 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).

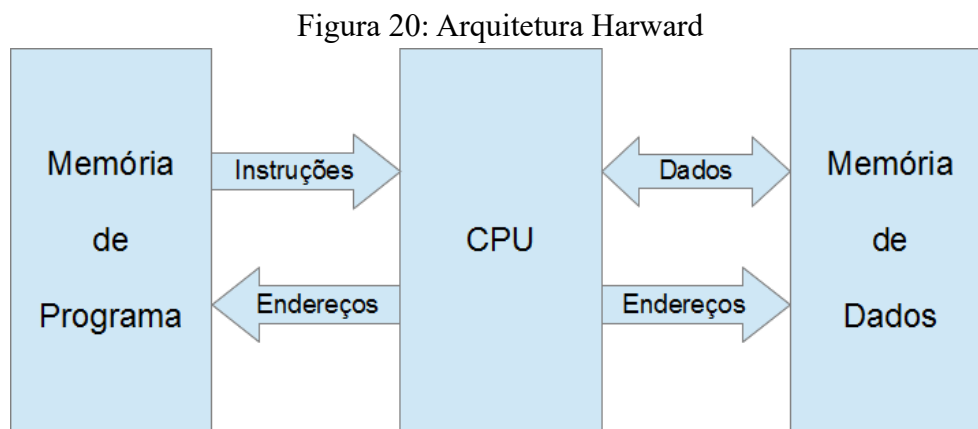
RISC = Reduced Instruction Set Code (conjunto de código de instruções reduzido).

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.

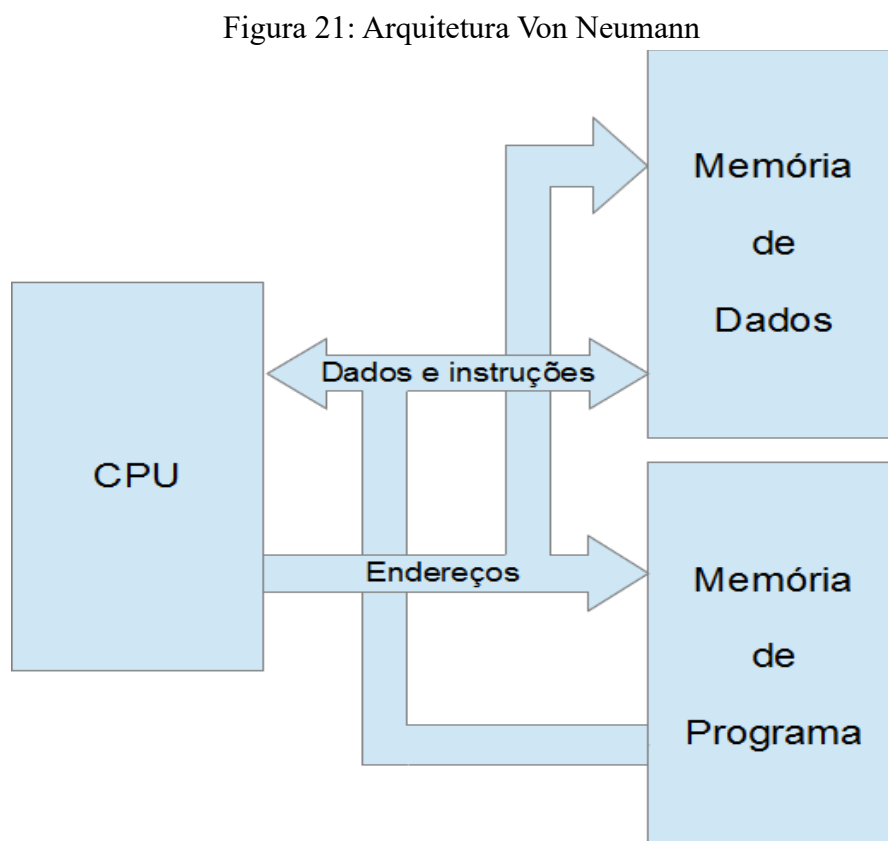
Outro aspecto importante é o tamanho dos programas. Imagine um programa que faça uma movimentação de dados de 1000 bytes de uma parte da memória para outra. Em um microprocessador CISC esse programa é curto (podemos supor algo em torno de 5 ou 6 bytes). Isso porque utiliza instruções complexas. Em um 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. Veja a Figura 20.



Já na arquitetura Von Neumann, há apenas um barramento de dados e endereços, veja a Figura 21.



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. Nesta 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. Assim o custo de produção das placas que utilizam este tipo de processadores é maior. Von Neumann utiliza apenas um barramento, mas não pode efetuar acessos simultâneos às memórias. Assim seu custo de produção é menor, porém seu desempenho também é menor.

Como exemplos podemos citar o computador pessoal que utiliza a arquitetura Von Neumann e os microcontroladores AVR que utilizam a arquitetura Harvard.

A arquitetura Harvard é bastante utilizada nos microcontroladores pelo fato das memórias estarem integradas no próprio componente, o que não acarreta em placas de circuito complexas com várias trilhas de endereçamento e dados.

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 normalmente 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).

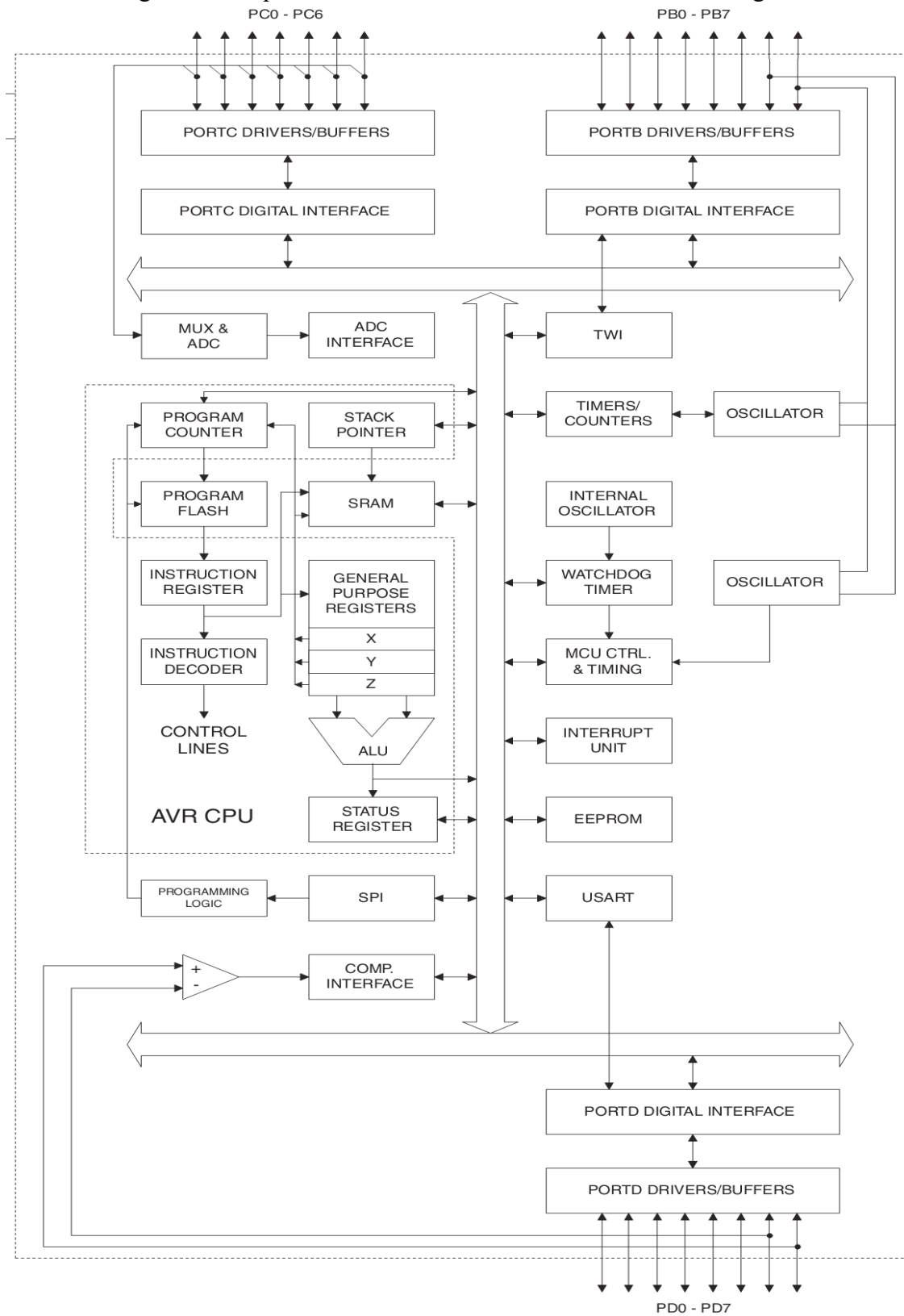
2.14 Microcontroladores e Microprocessadores

Até agora foram estudados microcontroladores e microprocessadores, mas não foi discutida qual a diferença entre eles. 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.

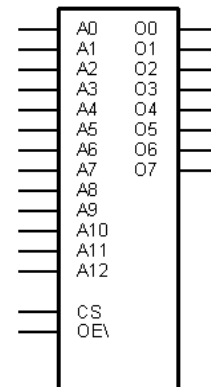
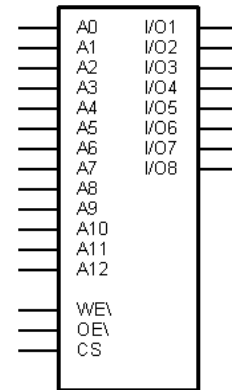
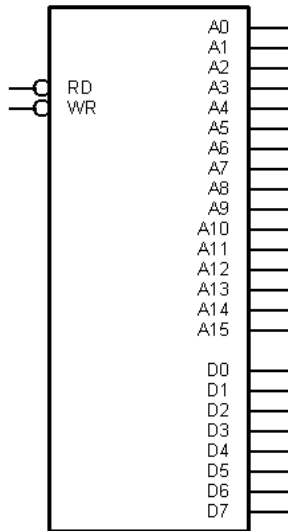
A Figura 22 apresenta a arquitetura interna de um microcontrolador chamado ATmega8 da família AVR fabricado pela ATMEL. Nesta figura é possível observar os periféricos existentes no interior deste componente, principalmente a memória RAM e a memória FLASH.

Figura 22: Arquitetura interna de um microcontrolador ATmega8



2.15 Exercícios

1. Na figura a seguir é apresentado um processador, uma memória RAM para os dados e uma memória ROM para o programa. Faça a ligação dos sinais de dados, de endereços e de controle entre o processador e as memórias. Descubra também qual a capacidade de cada uma das memórias.



AULA 3 - A FAMÍLIA DE MICROCONTROLADORES AVR

Arquitetura interna dos microcontroladores AVR e do Atmega8

3.1 Objetivo:

Mostrar o funcionamento interno da família de microcontroladores AVR, fabricada pela ATMEL. Ao final desta aula os alunos serão capazes de identificar microcontroladores desta família, bem como compreender seu funcionamento interno. O aluno também irá conhecer as características do microcontrolador ATMEGA8, pois este será o microcontrolador utilizado nas aulas práticas.

3.2 Introdução a família AVR

AVR é o nome dado a uma linha de microcontroladores fabricada pela ATMEL. A família de microcontroladores de 8 bits AVR é composta por dispositivos com baixo consumo de energia e alta velocidade. Estas características são garantidas por uma CPU RISC com instruções de um único ciclo de clock.

As instruções do AVR são projetadas para reduzir o tamanho do programa, tanto em linguagem assembler como em C. A estrutura dos pinos de entrada e saída ajuda a reduzir a necessidade de componentes externos, diminuindo assim o custo do projeto.

Possui internamente uma variedade de osciladores, temporizadores, portas seriais, moduladores PWM, resistores de “PULL UP³”, Conversores A/D e comparadores, facilitando em muito a vida dos projetistas e reduzindo o tempo de desenvolvimento.

3.3 Desempenho

Os microcontroladores AVR têm características que garantem um alto desempenho, tais como:

- Arquitetura RISC.
- Instruções realmente em um único ciclo.
- Um MIPS⁴ por MHz.
- 32 Registradores de uso geral
- Arquitetura Harvard
- Baixo consumo de energia

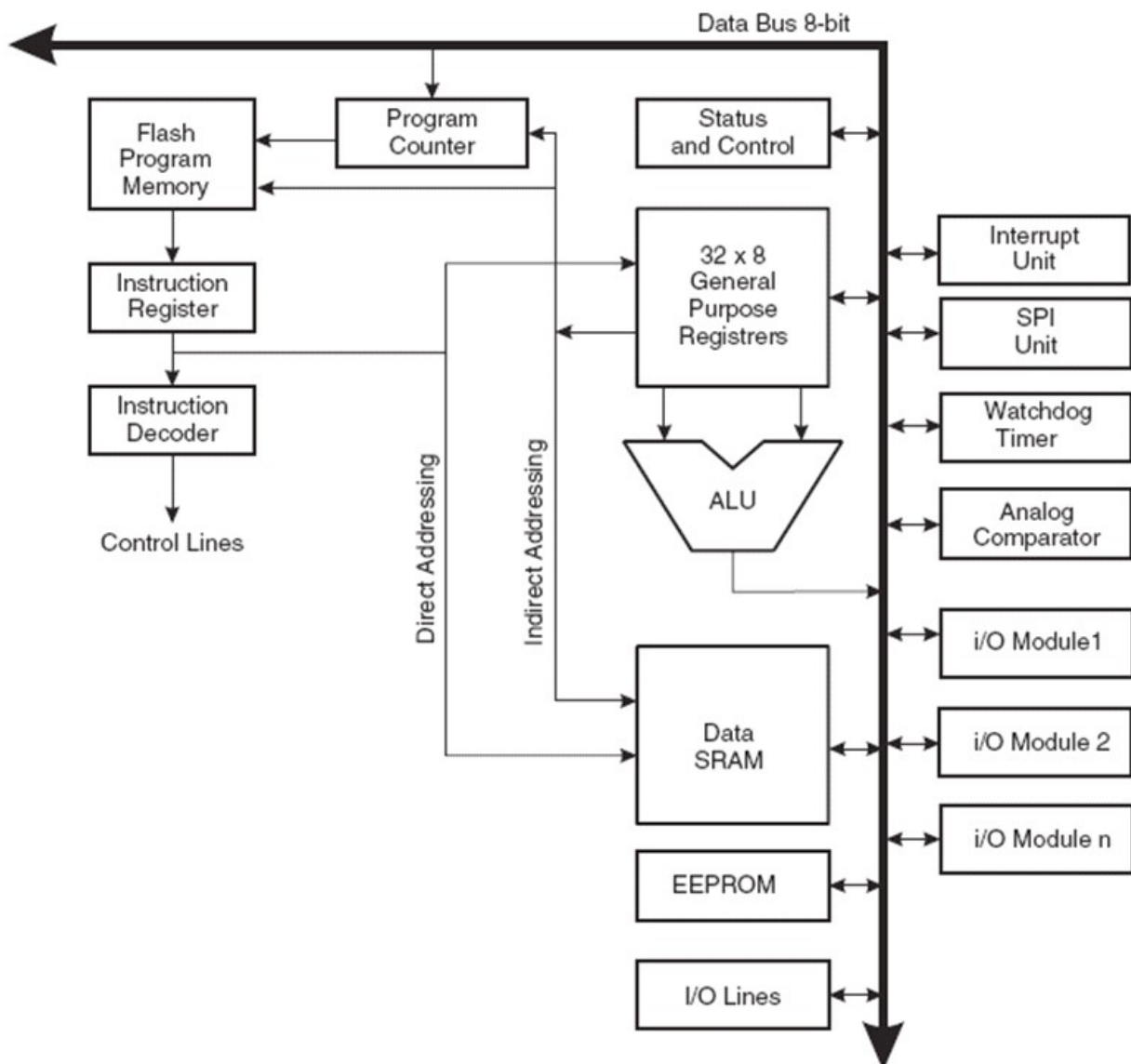
A família AVR pode operar com velocidades de até 20 MHz, processando até 20 MIPS. Com seus 32 registradores de uso geral os microcontroladores AVR têm um ganho ainda maior de desempenho, especialmente quando programado com linguagens de alto nível, como C, Pascal ou Basic.

3 Resistor que polariza um pino sem sinal definido.

4 Milhões de instruções por segundo.

A Figura 23 mostra a arquitetura interna geral dos microcontroladores AVR.

Figura 23: Arquitetura Interna dos microcontroladores AVR



3.4 Baixo consumo de energia

As características que garantem o baixo consumo da família AVR é garantido por:

- Operação de 1.8 a 5.5V;
- Grande variedade de modos de espera;
- Frequência de operação controlada por software;
- Alta densidade de código.

A grande variedade de modos de funcionamento e a flexibilidade no ajuste de velocidade permitem controlar o consumo de energia eficientemente. Para aplicações com baterias, os microcontroladores AVR são capazes de operar com baterias completamente cheias ou quase vazias, pois a tensão de operação é extremamente flexível.

Os microcontroladores AVR têm seis modos de espera, isto assegura baixo consumo de energia e ao mesmo tempo alta velocidade no retorno ao modo normal de operação. O ajuste de velocidade por software assegura alta performance quando for necessário, e baixo consumo no restante do tempo.

A característica de alta densidade de código garante que menos instruções sejam necessárias para executar uma determinada tarefa, o que diminui muito o consumo de energia.

3.5 Variedade de dispositivos

A família AVR possui uma grande variedade de componentes, com as seguintes características:

- Grande variedade de número de pinos;
- Compatibilidade total de código;
- Compatibilidade de funções e periféricos;
- Conjunto único de ferramentas de desenvolvimento.

Com a variedade de dispositivos é possível começar com um dispositivo de pequeno porte e se no decorrer do projeto for necessário mais capacidade do que foi previsto migrar para outro sem perder o trabalho já feito.

Todos os microcontroladores da família AVR utilizam o mesmo núcleo (CPU), o que torna possível reutilizar o código de um projeto em outro, mesmo com periféricos diferentes. Com dispositivo variando de 1 Kbytes a 256 Kbytes de memória de programa e embalagens com 8 a 100 pinos, os projetos terão sempre um microcontrolador sob medida.

Outra vantagem é que é possível utilizar sempre a mesma ferramenta de desenvolvimento, para todos os membros da família.

3.6 Variedade de embalagens

A família AVR de microcontroladores é fornecida em uma grande variedade de embalagens, se adequando as mais variadas aplicações.

A Figura 24 mostra a variedade de embalagens da família AVR, é possível observar que existem embalagens do tipo DIP⁵ que podem ser soldadas a mão e embalagens SMD⁶ que devem ser soldadas por máquinas.

3.7 Aplicações específicas

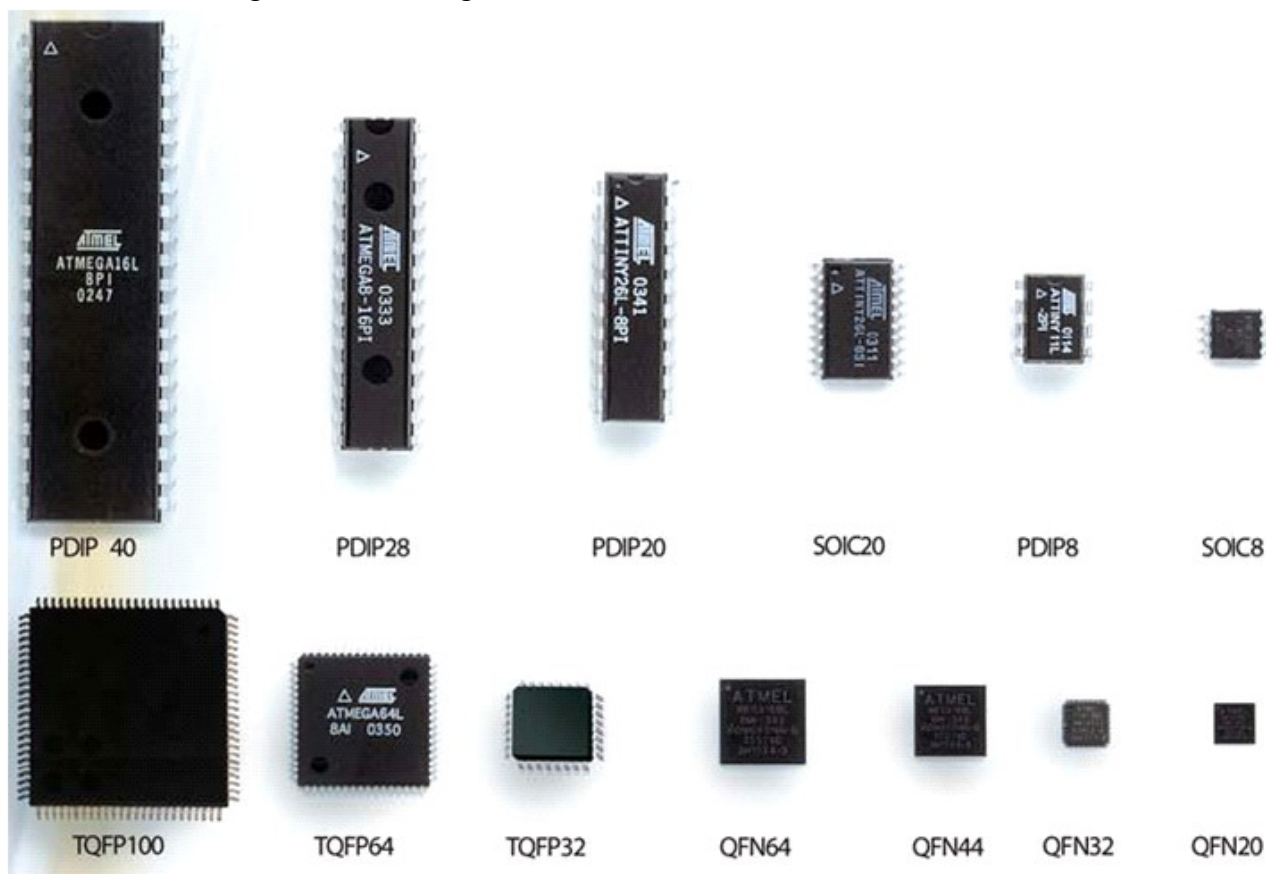
Existem microcontroladores da família AVR criados para ter características únicas, estes

5 Dual inline package

6 Surface mounted devices

dispositivos tem aplicações específicas, como automotivas, LCDs, redes CAM, USB, controle de motores, controle de telas de LCD, etc.

Figura 24: Embalagens mais comuns dos microcontroladores AVR



3.8 Programação

Os microcontroladores AVR possuem características de programação que os tornam muito flexíveis, tais como:

- Programação na própria placa;
- Debug no próprio chip;
- Verificação no próprio circuito;
- Memória Flash com 10.000 ciclos de gravação.

Com estes microcontroladores o tempo de desenvolvimento é reduzido, pois o programa pode ser testado na própria placa, sem a necessidade de remover o componente para programá-lo.

A memória de programa é do tipo Flash, garantindo assim mais de 10.000 ciclos de gravação, facilitando o desenvolvimento do programa e também futuras atualizações em equipamentos já prontos.

Lock-bits protegem o programa contra cópias não autorizadas, evitando assim a pirataria.

3.9 Ferramentas de desenvolvimento

Existe uma variedade de ferramentas de desenvolvimento para os microcontroladores da família AVR:

- AVR Studio®;
- WinAVR™;
- Kits de desenvolvimento;
- Emuladores;

O fabricante fornece uma ferramenta gratuita de desenvolvimento chamada AVR Studio®, esta ferramenta serve para programação e simulação de todos os componentes desta família. O ambiente pode ser baixado no site www.atmel.com.

3.10 Suporte

A ATMEL disponibiliza em seu site vasta documentação que nos ajuda a resolver nossos problemas. Dentre estes documentos existem as notas de aplicação (*Application notes*), que são exemplos prontos que ilustram o funcionamento e as aplicações de cada um dos periféricos destes componentes. Também existem sites dedicados a esta família, como por exemplo, o AVRfreaks (www.avrfreaks.net), onde existem muitos exemplos de hardware e software.

3.11 A CPU AVR

A CPU é o núcleo (core) do microcontrolador, ela é responsável pela correta execução do programa. A CPU também é responsável por acessar as memórias, realizar os cálculos, controlar os periféricos e manipular as interrupções.

3.11.1 Arquitetura da CPU AVR

A Figura 23 mostra a arquitetura interna da CPU dos microcontroladores AVR, bem como sua conexão com as memórias e módulos de entrada e saída.

Para garantir a máxima performance e o paralelismo, o AVR utiliza a arquitetura Harvard, com memórias e barramentos separados para programa e dados. As instruções da memória de programa são executadas através de um *pipeline* simples, ou seja, enquanto uma instrução é executada outra é lida da memória de programa e decodificada. Este conceito permite que a CPU AVR execute uma instrução a cada ciclo de *clock*.

Esta CPU contém 32 registradores de 8 bits, estes registradores são de propósito geral e tem um tempo de acesso de um ciclo de *clock*. Isto permite que a unidade lógica aritmética execute suas instruções em um único ciclo de *clock*.

Em uma operação típica da unidade lógica e aritmética, dois operadores são carregados a partir dos registradores, a operação é executada e o resultado é armazenado novamente nos registradores. Na CPU AVR este processo é executado em um ciclo de *clock*. Seis dos 32 registradores podem ser utilizados como três registradores de 16 bits para endereçamento indireto. Estes registradores são utilizados como ponteiros, para endereçar dados. Possibilitando assim, um eficiente cálculo de endereços. Um dos três registradores, pode também ser usado como ponteiro para apontar dados na memória de programa.

Estes registradores de 16 bits são chamados de registrador - X, registrador -Y e registrador - Z, e serão estudados mais a frente. A ULA suporta operações lógicas e aritméticas entre registradores ou entre uma constante e um registrador. Também existem instruções que utilizam apenas um

operador.

Após a execução de uma instrução o registrador de Status é atualizado, refletindo informações sobre o resultado da operação. O fluxo do programa é providenciado por instruções de salto condicional e salto incondicional. Também existem instruções de chamada de sub-rotinas. Todas estas instruções são capazes de endereçar todo o espaço de memória diretamente.

A maioria das instruções da CPU AVR são formadas por uma palavra (*word*) de 16 bits. Cada endereço de memória de programa possui uma instrução de 16 bits ou de 32 bits.

A memória de programa é dividida em duas sessões, a sessão de *boot* e a sessão de programa. Cada setor possui *Lock Bits* independentes, que protegem contra a escrita e leitura não autorizada do programa. A instrução SPU que é responsável por gravar dados na memória e programa só pode ser executada na sessão de *boot* da memória de programa.

Durante interrupções e chamadas de sub-rotinas, o endereço de retorno é armazenado na pilha. A pilha é eficientemente alocada na memória de dados SRAM, assim o tamanho da pilha é limitado apenas pela memória de dados livre no microcontrolador.

Todos os programas de usuário devem iniciar o ponteiro da pilha (*stack pointer SP*) antes de executar sub-rotinas ou interrupções. O ponteiro da pilha é um registrador que indica o endereço de memória onde os dados da pilha devem ser gravados. Este registrador pode ser lido e gravado através de operações de entrada e saída.

A memória de dados, SRAM, pode ser acessada facilmente através de cinco modos diferentes de endereçamento. Os espaços de memória da CPU AVR são todos lineares e regulares.

A CPU AVR possui um módulo de interrupções que pode ser acessado através de operações de entrada e saída. Este módulo possui um bit de acionamento no registrador de *status*. Todas as interrupções possuem um vetor de interrupção em uma tabela específica. A prioridade de cada interrupção é dada pela posição de seu vetor nesta tabela. Quanto mais baixo o endereço do vetor de interrupção, maior é a prioridade.

O espaço de memória de entrada e saída possui 64 endereços para controle dos periféricos, através dos registradores de controle. O espaço de memória de entrada e saída pode ser acessado diretamente ou no espaço de dados que se seguem aos registradores, 0x20 – 0x5F.

3.11.2 Unidade Lógica e Aritmética – ULA

A unidade lógica e aritmética da CPU AVR opera em conexão direta com os 32 registradores de uso geral. Executando em um único ciclo de *clock* as operações lógicas e aritméticas entre registradores de uso geral ou entre uma constante e um destes registradores.

As operações da ULA são divididas em três grandes categorias: Aritméticas, lógicas e funções de bits. Para maiores detalhes sobre as instruções executadas pela ULA consulte o resumo das instruções nos anexos deste material.

3.11.3 Registrador de Status

O registrador de status contém informações sobre o resultado da última operação executada pela ULA. Estas informações podem ser usadas para alterar o fluxo do programa executando operações condicionais. Observe que o registrador de status é atualizado após todas as instruções, conforme indicado no resumo das instruções nos anexos.

Isso faz com que em muitos casos não seja necessário utilizar instruções de comparação,

resultando em código mais compacto e mais rápido.

O registrador de status não é armazenado automaticamente quando uma interrupção é executada, assim como não é restaurado automaticamente ao final da execução da rotina de interrupção. Estas operações devem ser realizadas por software.

O registrador de status é chamado de SREG, e definido como na Tabela 4.

Tabela 3: Registrador de status - SREG

BIT	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Leitura / Escrita	L / E	L / E	L / E	L / E	L / E	L / E	L / E	L / E	
Valor inicial	0	0	0	0	0	0	0	0	

Bit 7 – I: Global Interrupt Enable

Este bit deve ser setado para que as interrupções estejam habilitadas. O controle da habilitação de interrupções individuais é realizado em registradores específicos. Se este bit estiver em zero nenhuma das interrupções estará habilitada, independente do controle individual de cada uma delas. O bit I é zerado pelo hardware toda vez que uma interrupção acontece, e é setado toda vez que a instrução RETI é executada no final da interrupção.

O bit I também pode ser controlado pelo software, utilizando as instruções SEI e CLI, como pode ser observado no resumo das instruções nos anexos.

Bit 6 – T: Bit Copy Storage

As instruções de cópia de bit, BLD (Bit LoaD) e BST (Bit Store) usam o bit T como fonte ou destino da operação com bits. Um bit de um registrador dos registradores de uso geral pode ser copiado no bit T pela instrução BST e um bit no registrador T pode ser copiado para um registrador de uso geral pela instrução BLD.

Bit 5 – H: Half Carry Flag

O bit Half Carry Flag indica a situação de Half Carry, que pode ocorrer em algumas operações aritméticas. A indicação de Half Carry é útil em operações aritméticas que utilizam a notação BCD.

Maiores detalhes podem ser encontrados no resumo das instruções nos anexos.

Bit 4 – S: Sign Bit, $S = N \oplus V$

O bit S é o resultado da operação ou exclusivo entre o bit Negative Flag N e o Two's Complement Overflow Flag V.

Maiores detalhes podem ser encontrados no resumo das instruções nos anexos.

Bit 3 – V: Two's Complement Overflow Flag

Este bit da suporte a aritmética em complemento de dois. Maiores detalhes podem ser encontrados no resumo das instruções nos anexos.

Bit 2 – N: Negative Flag

Este bit indica um resultado negativo em uma operação lógica ou aritmética. Maiores detalhes podem ser encontrados no resumo das instruções nos anexos.

Bit 1 – Z: Zero Flag

Este bit indica um zero como resultado em uma operação lógica ou aritmética. Maiores detalhes podem ser encontrados no resumo das instruções nos anexos.

Bit 0 – C: Carry Flag

Este bit indica a existência de resto em operações lógicas ou aritméticas. Maiores detalhes podem ser encontrados no resumo das instruções nos anexos.

3.11.4 Registradores de uso geral

O conjunto de registradores de uso geral é otimizado para ser usado com o conjunto avançado de instruções RISC da CPU AVR. Para obter a performance e a flexibilidade exigidas, os seguintes senários de entrada e saída são suportados pelos registradores de uso geral.

- Um operando de 8 bits e um resultado de 8 bits
- Dois operandos de 8 bits e um resultado de 8 bits
- Dois operandos de 8 bits e um resultado de 16 bits
- Um operando de 16 bits e um resultado de 16 bits

A Tabela 4 apresenta os registradores de uso geral da CPU AVR.

Tabela 4: Registradores de uso geral

	7	0	Endereço	
Registradores de uso geral	R0		0x00	
	R1		0x01	
	...			
	R25		0x19	
	R26		0x1A	Byte baixo do registrador X
	R27		0x1B	Byte alto do registrador X
	R28		0x1C	Byte baixo do registrador Y
	R29		0x1D	Byte alto do registrador Y
	R30		0x1E	Byte baixo do registrador Z
	R31		0x1F	Byte alto do registrador Z

A maioria das instruções que operam com os registradores de uso geral tem acesso direto a todos os registradores e em sua maioria, as instruções são executadas em um único ciclo de *clock*.

Como pode ser visto na Tabela 4, cada registrador possui um endereço de memória, assim os

primeiros 32 bytes da memória de dados ocupados pelos registradores de uso geral. Apesar de não estar fisicamente implementado como memória SRAM, esta arquitetura proporciona grande flexibilidade no acesso aos registradores, pois os ponteiros X, Y e Z podem ser ajustados para apontar qualquer registros de uso geral.

O registrador X, o registrador Y e o registrador Z

Os registradores R26 a R31 tem funções especiais. Estes registradores são ponteiros de endereço de 16 bits, usados para acessar indiretamente a memória de dados. Estes três registradores são definidos como pode ser visto na Tabela 5.

Tabela 5: Registradores X, Y e Z

	15	XH		XL	0
Registrador X	7		0	7	0
	R27 (0x1B)			R26 (0x1A)	
	15	YH		YL	0
Registrador Y	7		0	7	0
	R29 (0x1D)			R28 (0x1C)	
	15	ZH		ZL	0
Registrador Z	7		0	7	0
	R31 (0x1F)			R30 (0x1E)	

Nos diferentes modos de endereçamento, estes registradores tem funções como, deslocamento fixo, incremento automático e decremento automático. Maiores detalhes podem ser encontrados no resumo das instruções nos anexos.

3.11.5 Ponteiro da pilha

A pilha é comumente usada para armazenar dados temporários, variáveis locais e endereços de retorno após a chamada de sub-rotinas e interrupções. O ponteiro da pilha sempre aponta para o topo da pilha. É importante lembrar que a pilha esta alocada no final da memória de dados e cresce na direção dos endereços mais baixos da memória. Isto implica em que uma instrução PUSH (inserir dados na pilha) irá decrementar o ponteiro da pilha. A Tabela 6 demonstra este processo.

O espaço para a pilha e o endereço inicial para a mesma deve ser ajustado pelo programa antes de ativar as interrupções ou chamar sub-rotinas. O ponteiro da pilha deve ser inicializado com um valor acima de 0x60.

Como já foi dito, o ponteiro da pilha é decrementado de um quando uma instrução de PUSH é executada, ele é decrementado de dois quando o endereço de retorno de uma interrupção ou sub-rotina é armazenado. Isto acontece pois os endereços tem 16 bits.

Tabela 6: Comportamento do ponteiro da pilha

Antes da instrução PUSH			Após da instrução PUSH		
Endereço	Memória		Endereço	Memória	
0x70			0x70		
0x71			0x71		
0x72		Ponteiro da pilha	0x72	0xF8	← Ponteiro da pilha
0x73	0x33	← 0x73	0x73	0x33	
0x74	0xAB		0x74	0xAB	

O ponteiro da pilha é incrementado em um quando um dado é retirado da pilha através de uma instrução POP, e incrementado em dois quando um endereço de retorno é retirado através de uma instrução RET ou RETI (retorno de sub-rotina ou interrupção). O ponteiro da pilha da CPU AVR é implementada como dois registradores de 8 bits no espaço de entrada e saída. Na programação o ponteiro da pilha é chamado SP. A Tabela 7 apresenta esta arquitetura.

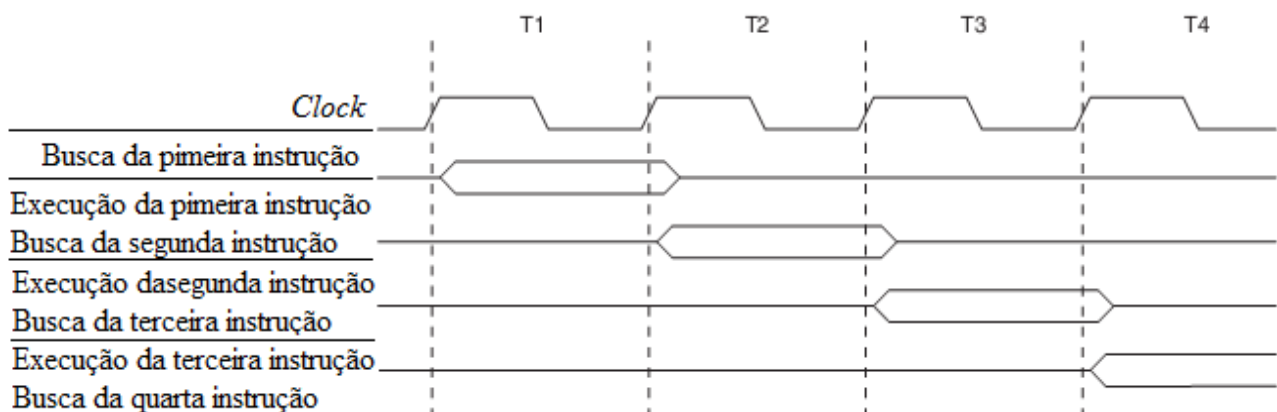
Tabela 7: O ponteiro da pilha

BIT	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Leitura / Escrita	L / E	L / E	L / E	L / E	L / E	L / E	L / E	L / E	
Valor inicial	0	0	0	0	0	0	0	0	

3.11.6 Tempo de execução das instruções

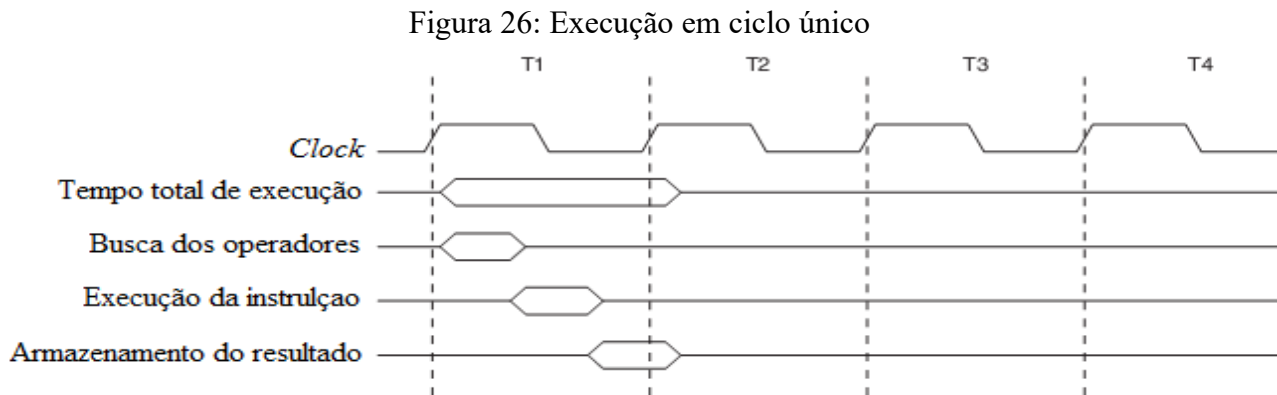
As instruções da CPU AVR são impulsionados pelo *clock* da CPU. Este *clock* é gerado pela fonte de *clock* selecionada para o chip. Não é usado nenhum tipo de divisão interna de *clock*, a Figura 25 apresenta a busca paralela e execução das instruções.

Figura 25: Busca e execução paralela de instruções



A CPU AVR utiliza a arquitetura Harward, que junto a arquitetura dos registradores de uso geral permite a execução das instruções e apenas um ciclo de *clock*. Este conceito é o que permite que a CPU execute 1 MIPS por MHz.

A Figura 26 apresenta a temporização interna envolvendo os registradores de uso geral e a ULA. Em um único ciclo de *clock* uma operação da ULA, envolvendo dois operadores, é executada e o resultado é armazenado no registrador de destino.



3.11.7 O reset e as interrupções

A CPU AVR fornece várias fontes de interrupções, estas interrupções e o reset possuem um vetor de interrupção independente, na memória de programa. Cada interrupção possui um bit de ativação individual. Para habilitar as interrupções é necessário ativar o bit I no registrador SREG.

Os endereços mais baixos na memória de programa são reservados para os vetores de interrupção e o reset. Quando uma interrupção acontece o bit I do registrador de status SREG é zerado e as interrupções são desabilitadas. O software do usuário pode escrever um neste bit para habilitar interrupções aninhadas. Assim qualquer interrupção pode interromper a interrupção ativa. O bit I é automaticamente passado para um quando a instrução RETI é executada.

3.11.8 Tempo de resposta das interrupções

O tempo de resposta para qualquer interrupção ativa na CPU AVR é de quatro ciclos de clock. Após estes quatro ciclos de clock o endereço do vetor de interrupção é executado. O vetor de interrupção é normalmente um salto para a sub-rotina de interrupção, o que leva mais três ciclos de clock. Este tempo pode ainda aumentar se a instrução que esta sendo executada é uma instrução de múltiplos ciclos, pois a instrução atual deve terminar para a interrupção ser atendida.

O retorno da interrupção também demora quatro ciclos de clock. Neste tempo o endereço de retorno é puxado da pilha, e movido para o PC (program counter), o ponteiro da pilha é incrementado em dois e o bit I é aticado.

3.12 O microcontrolador ATmega8

Como a família de microcontroladores AVR é muito vasta, não será possível o aprendizado com todos os componentes, assim sendo escolheremos o ATmega8. A seguir estão as principais características deste componente.

- **Arquitetura RISC avançada**

- 130 Instruções, a maioria de um único ciclo;
- 32 registradores de 8 bits de uso geral;
- Até 16 MIPS em 16 MHz.

- **Memória não volátil de alta durabilidade**

- 8K Bytes de memória Flash programável no circuito, Ciclos de leitura e escrita: 10.000.
- 512 Bytes EEPROM, Ciclos de leitura e escrita: 100.000.
- 1K Byte SRAM interna;

- **Periféricos**

- Dois temporizadores/contadores de 8 bits e um temporizador/contador de 16 bits;
- Contador de tempo real com oscilador independente;
- Três canais de PWM;
- 6-canais ADC de 10-bits;
- Interface serial assíncrona programável;
- Interface SPI mestre/escravo;
- Temporizador “Watchdog”;
- Comparador analógico

- **Funções especiais**

- Oscilador RC interno calibrado.

- **Entradas e saídas**

- 23 linhas de entrada e saída programáveis.

- **Tensão de operação**

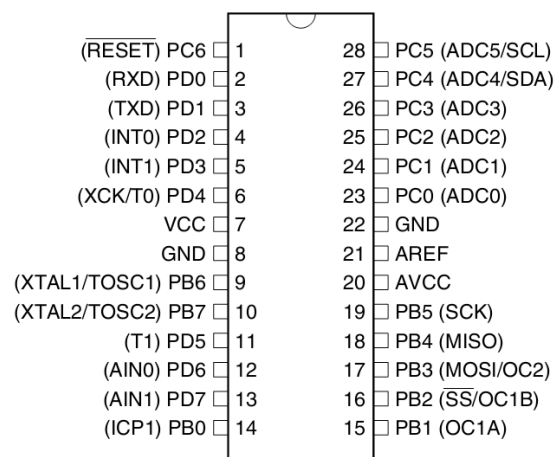
- 2.7 - 5.5V (ATmega8L);
- 4.5 – 5.5V (Atmega8).

- **Velocidades**

- 0 - 8 MHz (ATmega8L) e 0 - 16 MHz (ATmega8).

O ATmega8 é vendido em vários encapsulamentos, mas para nossos estudos usaremos a configuração PDIP. A Figura 27 mostra a configuração dos pinos deste componente.

Figura 27: Pinagem do ATmega8



3.13 Função dos pinos

A Tabela 8 Apresenta a função dos pinos do ATmega8.

Tabela 8: Funções dos pinos do ATmega8

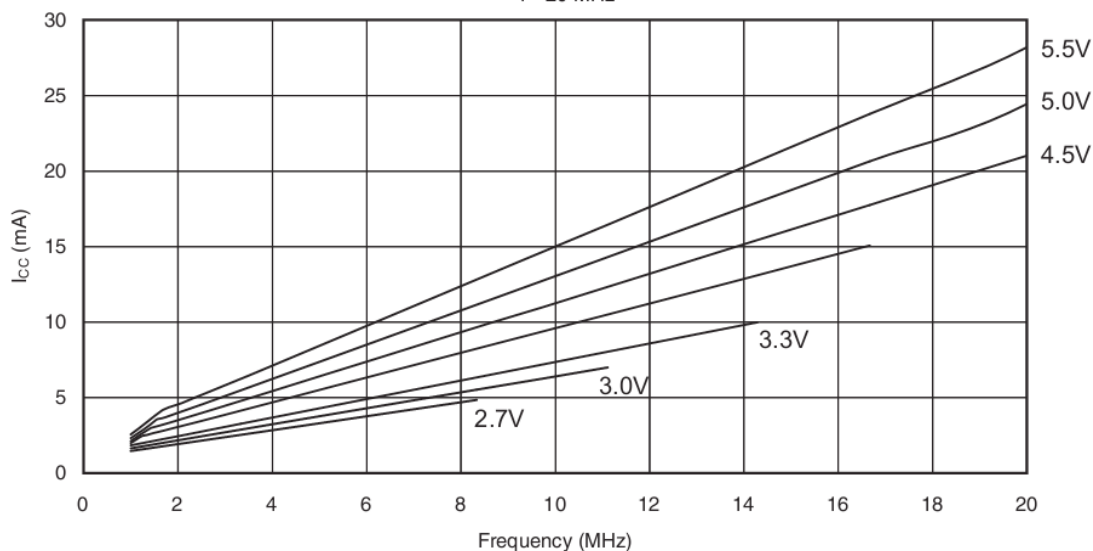
VCC	Alimentação positiva do circuito.
GND	Alimentação negativa do circuito.
Port B, C e d	As portas B, C e D são portas de 8 bits, bidirecionais e cada pino pode tanto fornecer como drenar corrente. Estas portas também tem funções especiais.
AVcc	Alimentação do conversor A/D.
Aref	Referência para o conversor A/D.

3.14 Características elétricas do ATmega8

- Temperatura de operação: -55°C to $+125^{\circ}\text{C}$;
- Temperatura de armazenamento: -65°C to $+150^{\circ}\text{C}$;
- Tensão em qualquer pino com relação ao GND: 0.5V to $\text{VCC}+0.5\text{V}$;
- Tensão no reset com relação ao GND: 0.5V to $+13.0\text{V}$;
- Tensão máxima de operação: 6.0V ;
- Corrente máxima por pino de entrada e saída: 40.0 mA ;
- Corrente máxima no GND e no Vcc: 300.0 mA .

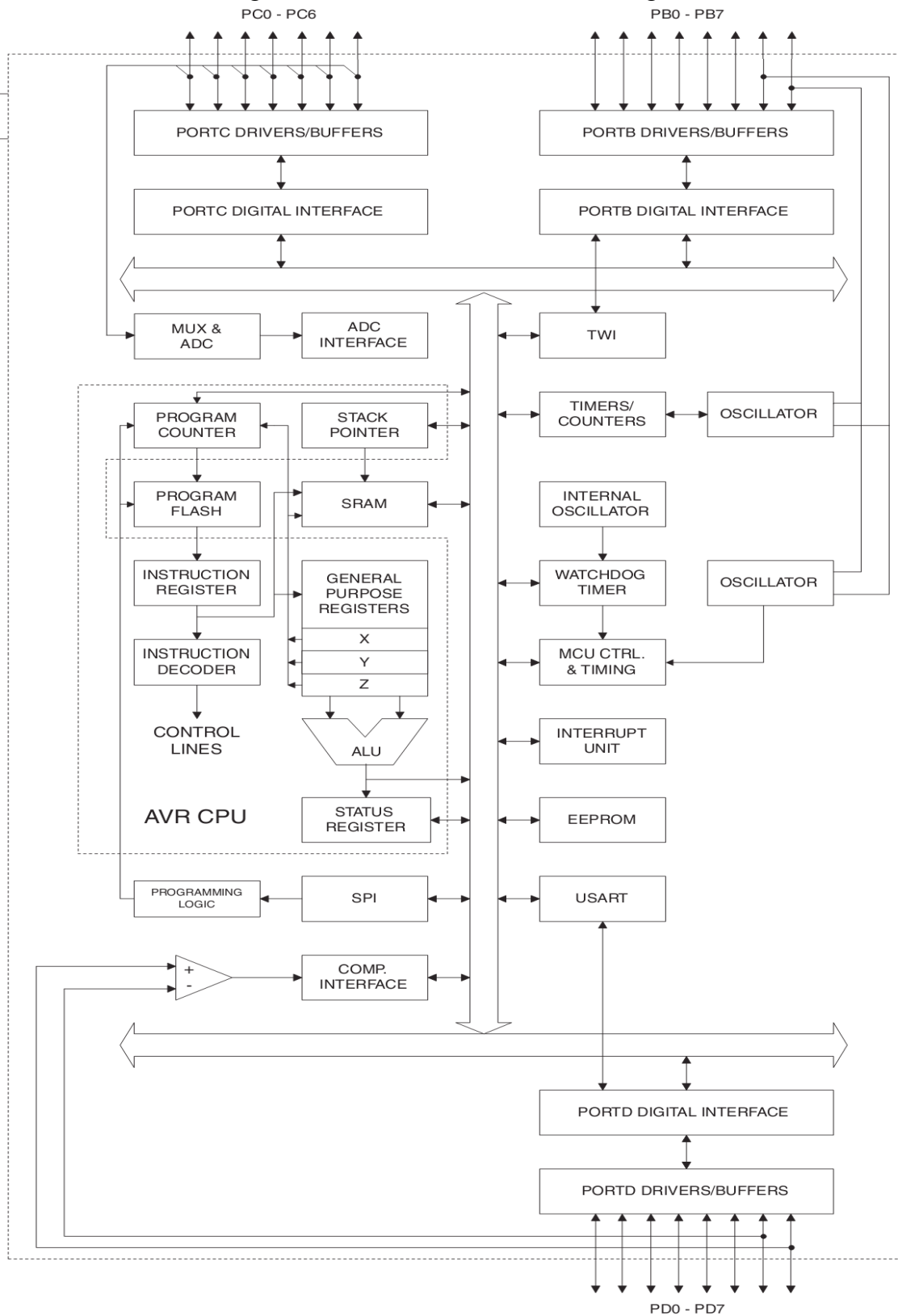
3.15 Consumo de corrente da fonte em relação à frequência de operação:

Figura 28: Consumo de corrente do ATmega8



A Figura 28 Apresenta as curvas de consumo do microcontrolador ATmega8, em função da tensão de alimentação e da frequência de clock.

Figura 29: Periféricos internos do ATmega8



Como pode ser observado na Figura 29, o microcontrolador ATmega8 tem um grande número de periféricos, eles serão estudados nas próximas aulas.

3.16 Exercícios

1. Supondo que um microcontrolador da família AVR esteja operando em 12Mhz, quantas instruções ele poderia executar em 5 segundos? Por quê?
2. A família AVR utiliza arquitetura Harvard ou Von Neumann? Por quê?
3. Qual o tipo de memória que os microcontroladores da família AVR utilizam para armazenar o programa, e quantas vezes esta memória pode ser regravada?
4. Supondo que um microcontrolador ATmega8 esteja trabalhando em 2,7V, qual a máxima frequência que ele pode operar?
5. Preencha a tabela a seguir com o número do pino do circuito correspondente a função indicada na primeira coluna.

Função	Pino
Reset	
PD1	
ADC0	
Vcc	
TXD	
PB1	
PC4	

6. Supondo que a um pino de saída é conectado um led, qual a máxima corrente que este led pode receber do microcontrolador ATmega8 sem que o mesmo sofra danos?
7. Qual a corrente consumida por um ATmega8 operando a 16MHz e alimentado com 5V.

AULA 4 - A LINGUAGEM ASSEMBLER

Os fundamentos da linguagem de programação de microcontroladores assembler

4.1 Objetivo:

O objetivo desta aula é apresentar aos alunos a linguagem de programação de microcontroladores AVR chamada assembler.

4.2 A linguagem assembler para microcontroladores AVR

Diferente das linguagens de alto nível como C a linguagem assembler varia de uma família de microcontroladores para outra. Isso acontece pois a linguagem assembler é uma tradução direta das instruções que o microcontrolador é capaz de executar.

É importante conhecer a linguagem assembler pois através dela é possível entender como a CPU executa suas operações. Mesmo quando se utiliza linguagens de alto nível, como por exemplo C, para programar os microcontroladores, o conhecimento da linguagem assembler é necessário, pois quando se analisa a memória de programa do microcontrolador o que se obtém é a codificação do programa em assembler.

4.3 Características de um programa em linguagem assembler

Os programas em linguagem assembler são arquivos de texto que são compilados por um assembler para se tornar linguagem de máquina, compreensível para os microcontroladores.

Os programas são compostos por instruções que são diretamente convertidas em comandos para o processador e por diretivas, que não são comandos do processador mas dizem ao assembler como se comportar.

O exemplo a seguir mostra um programa em linguagem assembler.

```
.include "m8def.inc"
.equ  INPORT      = PIND
.equ  OUTDDR      = DDRC
.equ  OUTPORT     = PORTC
.def  zero        = r1
.def  temp        = r18

.cseg
.org  0           ; inicia o segmento de código
rjmp  reset      ; salta para o inicio
.org  INT_VECTORS_SIZE ; define o local do inicio do programa
reset:
  clr  zero
  ldi  temp, 1
```

```

out   OUTDDR, temp           ; define o bit 0 da porta C como saída
main_loop:
in    temp, INPORT
andi  temp, $80             ; verifica o bit 7 da entrada
breq  output_low           ; se for 1 ativa a saída
ldi   temp, 1
out   OUTPORT, temp
rjmp  main_loop
output_low:
out   OUTPORT, zero
rjmp  main_loop           ; salta para o início do laço

```

4.4 Diretivas do assembler

A linguagem assembler suporta um conjunto de diretivas. As diretivas são comandos que não são traduzidos diretamente em comandos do processador. Estas diretivas servem para ajustar a posição do programa na memória e assim por diante. Um resumo das diretivas é apresentado na Tabela 9.

Tabela 9: Diretivas do assembler

BYTE	Reserva um byte para uma variável
CSEG	Segmento de código
DB	Define constantes do tipo byte
DEF	Define um nome simbólico para um registrador
DEVICE	Define qual o dispositivo usado pelo assembler
DSEG	Segmento de dados
DW	Define constantes do tipo word (16 bits)
ENDMACRO	Final de macro
EQU	Define um símbolo igual a uma expressão
ESEG	Segmento da EEPROM
EXIT	Sai do arquivo
INCLUDE	Lê fonte de outro arquivo (inclui)
LIST	Liga o gerador de arquivo de lista
LISTMAC	Liga o expensor de macro
MACRO	Inicia uma macro
NOLIST	Desliga o gerador de arquivo de lista
ORG	Define o início do programa na memória
SET	Define um símbolo para uma expressão

4.5 Descrição das diretivas

A seguir temos a descrição de cada uma das diretivas, com exemplos.

4.5.1 BYTE - Reserva um byte para uma variável

A diretiva BYTE reserva recursos na memória RAM. Para referenciar o local reservado, esta diretiva deve ser precedida por uma etiqueta (label). Esta diretiva recebe um parâmetro, que é o número de bytes que se deseja reservar. Esta diretiva só pode ser usada com o segmento de dados.

Sintaxe:

Etiqueta: .BYTE expressão

Exemplo:

```
.SET tab_size = 0x23
```

```
.DSEG
```

```
var1: .BYTE 1 ; reserva 1 byte para var1
```

```
table: .BYTE tab_size ; reserva tab_size bytes
```

```
.CSEG
```

```
ldi r30,low(var1) ; carrega o registrador Z baixo com o endereço de var1
```

```
ldi r31,high(var1) ; carrega o registrador Z alto com o endereço de var1
```

```
ld r1,Z ;carrega var1 no registrador 1
```

4.5.2 CSEG - Segmento de código

A diretiva CSEG define o início do segmento de código. Um arquivo em assembler pode ser constituído de vários segmentos de código, que é concatenado em um único segmento de código durante a compilação. A diretiva BYTE não pode ser usada no segmento de código. Em um arquivo assembler o segmento padrão é o segmento de código. A diretiva ORG pode ser usada para posicionar código e constantes em endereços específicos do segmento de código. Esta diretiva não recebe nenhum parâmetro.

Sintaxe:

```
.CSEG
```

Exemplo:

```
.DSEG ; inicia o segmento de dados
```

```
vartab: .BYTE 4 ; Reserva 4 bytes na RAM
```

```
.CSEG ; inicia o segmento de código
```

```
const: .DW 2 ; grava 0x0002 memória de programa (código)
```

```
mov r1,r0 ; carrega o conteúdo de r0 em r1
```

4.5.3 DB - Define constantes do tipo byte

A diretiva DB reserva recursos de memória, na memória de programa ou na memória

EEPROM. Para que se possa referenciar o local reservado é necessário que a diretiva venha precedida de uma etiqueta. A diretiva DB pode conter uma ou mais expressões, e deve ser alocada no segmento de código ou no segmento da EEPROM. As expressões devem ser separadas por vírgula e estar entre os valores -128 e 255. Se o número usado na expressão for negativo, será utilizada a notação de complemento de 2.

Sintaxe:

Etiqueta: .DB expressões

Exemplo:

.CSEG

consts: .DB 0, 255, 0b01010101, -128, 0xaa

.ESEG

eeconst: .DB 0xff

4.5.4 DEF - Define um nome simbólico para um registrador

A diretiva DEF permite que registradores sejam referenciados através de nomes simbólicos. Um símbolo definido pode ser utilizado no resto do programa, para referenciar o registrador ao qual ele foi atribuído. Um registrador pode ter vários nomes simbólicos ligados a ele. Um símbolo pode ser redefinido mais a frente no programa.

Sintaxe:

.DEF Símbolo=Register

Exemplo:

.DEF temp=R16

.DEF ior=R0

.CSEG

ldi temp,0xf0 ; Carrega 0xf0 no registrador temp

in ior,0x3f ; Lê o registrador SREG no registrador ior

eor temp,ior ; Executa a operação lógica ou exclusiva entre temp e ior

4.5.5 DEVICE - Define qual o dispositivo usado pelo assembler

A diretiva DEVICA permite ao usuário avisar ao assembler em qual dispositivo o código será executado. Se esta diretiva é usada, o assembler emite um aviso quando uma instrução não suportada é usada. Também é emitida uma mensagem de erro se a capacidade da memória é ultrapassada. Se nenhum dispositivo é definido o assembler assume que todas as instruções podem ser usadas e que não existe limite de memória. Se utilizarmos o AVRStudio na compilação não é necessário definir o dispositivo, pois o projeto já faz isso.

Sintaxe:

.DEVICE dispositivo

Exemplo:

```
.DEVICE AT90S1200      ; Usa o AT90S1200
.CSEG
    push r30           ; Irá gerar um aviso de erro.
```

4.5.6 DSEG - Segmento de dados

A diretiva DSEG define o início do segmento de dados. Um arquivo assembler pode conter vários segmentos de dados, que são concatenados em um pelo assembler. Um segmento de dados é normalmente constituído por diretivas BYTE e etiquetas. O segmento de dados possui seu próprio contador, que conta bytes. A diretiva ORG pode ser usada para alocar as variáveis em endereços específicos na memória RAM. Esta diretiva não recebe nenhum parâmetro.

Sintaxe:

```
.DSEG
```

Exemplo:

```
.DSEG                ; inicia o segmento de dados
var1: .BYTE 1        ; reserva um byte na memória
table: .BYTE tab_size ; reserva tab_size bytes.
```

```
.CSEG
```

```
    ldi    r30,low(var1) ; Carrega a parte baixa de Z
    ldi    r31,high(var1) ; Carrega a parte alta de Z
    ld     r1,Z          ; Carrega o conteúdo da memória apontado por var1 em r1
```

4.5.7 DW - Define constantes do tipo word (16 bits)

A diretiva DW reserva recursos de memória na memória de código ou na memória EEPROM. Para que se possa referenciar este local na memória é necessário usar uma etiqueta. Esta diretiva pode conter uma ou mais expressões. Esta diretiva só pode ser usada nos segmentos de código ou EEPROM. As expressões devem ser separadas por vírgula e os valores devem estar no intervalo entre -32768 a 65535. Se o valor for negativo o assembler irá utilizar a notação de complemento de 2 de 16 bits.

Sintaxe:

Etiqueta: .DW expressões

Exemplo:

```
.CSEG
varlist: .DW 0,0xffff,0b1001110001010101,-32768,65535
```

```
.ESEG
```

eevar: .DW 0xffff

4.5.8 ENDMACRO - Final de macro

A diretiva ENDMACRO define o final de uma macro. Esta diretiva não recebe nenhum parâmetro. Veja a diretiva MACRO para maiores informações sobre o que são macros.

Sintaxe:

.ENDMACRO

Exemplo:

```
.MACRO SUBI16 ; Inicia a definição de uma macro
    subi r16,low(@0) ; Subtrai o byte baixo
    sbci r17,high(@0) ; Subtrai o byte alto
.ENDMACRO ; Finaliza a definição da macro
```

4.5.9 EQU - Define um símbolo igual a uma expressão

A diretiva EQU atribui um valor a uma etiqueta, esta etiqueta pode ser usada mais a frente no programa. Uma etiqueta atribuída com a diretiva EQU é constante e não pode ser alterada ou redefinida.

Sintaxe:

.EQU etiqueta = expressão

Exemplo:

```
.EQU io_offset = 0x23
.EQU porta = io_offset + 2
.CSEG ; inicia o segmento de código
    clr r2 ; Zera o registrador 2
    out porta,r2 ; Escreve na porta
```

4.5.10 ESEG - Segmento da EEPROM

A diretiva ESEG determina o início do segmento da EEPROM. Um arquivo assembler pode conter vários destes segmentos que serão unidos em apenas um pelo assembler. A diretiva byte não pode ser usada na EEPROM. O segmento de EEPROM possui seu próprio contador de endereços, que conta bytes. A diretiva ORG pode ser utilizada para apontar endereços específicos na EEPROM. Esta diretiva não recebe nenhum parâmetro.

Sintaxe:

.ESEG

Exemplo:

```
.DSEG ; inicia o segmento de dados
vartab: .BYTE 4 ; reserva 4 bytes na memória RAM
```

```
.ESEG          ; inicia o segmento de EEPROM  
eevar: .DW 0xff0f ; inicializa uma palavra na EEPROM
```

```
.CSEG          ; inicia o segmento de código  
const: .DW 2    ; armazena o valor 2 na memória de programa  
mov  r1,r0     ; continua o programa ...
```

4.5.11 EXIT - Sai do arquivo

A diretiva EXIT instrui o assembler a parar de compilar o arquivo atual. Normalmente o assembler vai até o final do arquivo, porém se a diretiva EXIT é encontrada em um arquivo incluído com INCLUDE, o assembler continua da linha em seguida a diretiva INCLUDE no arquivo que a contém.

Sintaxe:

```
.EXIT
```

Exemplo:

```
.EXIT ; sai do arquivo atual
```

4.5.12 INCLUDE - Lê fonte de outro arquivo (inclui)

A diretiva INCLUDE instrui o assembler a começar a ler de um arquivo específico. O assembler então compila este arquivo até encontrar o seu fim, ou a diretiva EXIT. Um arquivo incluído com esta diretiva pode por sua vez incluir outros arquivos.

Sintaxe:

```
.INCLUDE “nome_do_arquivo”
```

Exemplo:

```
                                ; iodefs.asm:  
.EQU sreg=0x3f                 ; atribui nome ao registrador de status  
.EQU sphigh=0x3e               ; atribui nome a parte alta do ponteiro da pilha  
.EQU splow=0x3d                ; atribui nome a parte baixa do ponteiro da pilha  
  
                                ; incdemo.asm  
.INCLUDE “iodefs.asm”         ; Inclui definições de entrada e saída  
  
    in r0,sreg                 ; Lê o registrador de status
```

4.5.13 LIST - Liga o gerador de arquivo de lista

A diretiva LIST instrui o assembler a gerar o arquivo de lista. O assembler irá assim gerar um arquivo que é a combinação de código fonte assembler, endereços e instruções. Esta opção é ativada por padrão, e pode ser desativada com a diretiva NOLIST. Assim é possível gerar lista

apenas das partes do programa que interessam.

Sintaxe:

```
.LIST
```

Exemplo:

```
.NOLIST           ; desativa a geração da lista
.INCLUDE "macro.inc" ; os arquivos incluídos não serão listados
.INCLUDE "const.def"
.LIST             ; reinicia a geração da lista
```

4.5.14 LISTMAC - Liga o expensor de macro

A diretiva LISTMAC instrui o assembler que quando uma macro é chamada, a expansão da macro é mostrada no arquivo de lista. O padrão é mostrar apenas a chamada da macro com os parâmetros.

Sintaxe:

```
.LISTMAC
```

Exemplo:

```
.MACRO MACX           ; Define um exemplo de macro
    add r0,@0
    eor r1,@1
.ENDMACRO             ; Finaliza a macro

.LISTMAC              ; habilita a expansão da macro
    MACX r2,r1        ; Chama a macro
```

4.5.15 MACRO - Inicia uma macro

A diretiva MACRO instrui o assembler que ali começa uma macro. Esta diretiva armazena o nome da macro e na sequência do programa, quando este nome é encontrado, esta macro é expandida. Uma macro pode receber até 10 parâmetros, estes parâmetros são referenciados por @0 a @9.

Quando se usa uma macro, os parâmetros são passados no formato de uma lista separada por vírgula. A definição da macro é terminada com a diretiva ENDMACRO. Por padrão só a chamada a macro é mostrada no arquivo de lista. Se desejarmos que a macro seja expandida no arquivo de lista é necessário utilizar a diretiva LISTMAC. As macros são marcadas com um sinal de + nos arquivos de lista.

Sintaxe:

```
.MACRO nome da macro
```

Exemplo:

```
.MACRO SUBI16 ; inicia a macro de nome SUBI16
    subi @1,low(@0) ; subtrai o byte baixo
    sbci @2,high(@0) ; subtrai o byte alto
.ENDMACRO ; finaliza a macro

.CSEG ; Início do segmento de código
    SUBI16 0x1234,r16,r17 ; Sub.0x1234 de r17:r16
```

4.5.16 NOLIST - Desliga o gerador de arquivo de lista

A diretiva NOLIST instrui o assembler a desligar a geração do arquivo de lista. O assembler normalmente gera um arquivo de lista, mas esta opção pode ser desligada por esta diretiva. Esta diretiva pode ser utilizada em conjunto com a diretiva LIST para gerar a lista de um determinado trecho de código.

Sintaxe:

```
.NOLIST ; desliga a geração do arquivo de lista
```

Exemplo:

```
.NOLIST ; desliga a geração do arquivo de lista
.INCLUDE "macro.inc" ; inclui um arquivo de código fonte
.INCLUDE "const.def" ; inclui outro arquivo de código fonte
.LIST ; liga a geração do arquivo de lista
```

4.5.17 ORG - Define o início do programa na memória

A diretiva ORG ajusta o contador de endereço para um determinado valor. O valor é passado através de um parâmetro. Se a diretiva ORG é utilizada em um segmento de dados, é o contador de endereço da memória RAM que é ajustado, já de esta diretiva é usada em um segmento de código é o contador de memória de programa que é atualizado, e assim por diante. Se a diretiva é precedida por uma etiqueta, esta etiqueta recebe o valor do endereço fornecido como parâmetro. O valor padrão para o contador de memória do programa e da EEPROM é zero. Já o contador de memória RAM é iniciado com o valor 32, isso por que os registradores de uso geral utilizam os endereços de 0 a 31. É importante lembrar que os contadores da memória RAM e da memória EEPROM contam de um em um byte, já o contador da memória de programa conta de dois em dois, uma vez que cada instrução ocupa dois bytes.

Sintaxe:

```
.ORG expressão
```

Exemplo:

```
.DSEG ; início do segmento de dados
.ORG 0x67 ; ajusta o contador de endereços da RAM para 0x67
variable: .BYTE 1 ; reserva um byte na RAM, no endereço 0x67
.ESEG ; inicia o segmento da EEPROM
```

```
.ORG 0x20          ; ajusta o endereço na EEPROM
eevar:            .DW 0xfeff ; grava uma palavra (16 bits) na EEPROM
.CSEG
.ORG 0x10          ; ajusta o contador da memória de programa para 0x10
    mov r0,r1      ; inicia o programa
```

4.5.18 SET - Define um símbolo para uma expressão

A diretiva SET atribui um valor a uma etiqueta. Esta etiqueta pode ser usada na sequência em expressões. Uma etiqueta que recebeu uma atribuição através da diretiva SET pode ser alterada e receber uma nova atribuição mais a frente no programa.

Sintaxe:

```
.SET etiqueta = expressão
```

Exemplo:

```
.SET io_offset = 0x23 ; atribui o valor 0x23 a io_offset
.SET porta = io_offset + 2 ; atribui io_offset + 2 para porta

.CSEG ; inicia o programa
    clr r2 ; zera o registrador 2
    out porta,r2 ; move o conteúdo do registrador 2 para a porta
```

4.6 Expressões

A linguagem assembler incorpora expressões. Expressões consistem em operandos, operadores e funções. Todas as expressões são internamente de 32 bits

4.6.1 Operandos

Os seguintes operandos podem ser utilizados.

- Etiquetas definidas pelo usuário, que fornecem o endereço da memória onde foram inseridos.
- Variáveis definidas pelo usuário com a diretiva SET.
- Constantes definidas pelo usuário com a diretiva EQU.
- Constantes inteiras, que podem aparecer em vários formatos, incluindo:
 - a. Decimal (padrão). Ex. 10, 255.
 - b. Hexadecimal. Ex. 0X0A, \$0A, 0xff.
 - c. Binário. Ex 0b00001010, 0b11111111.
- PC – que é o valor corrente do contador de memória de programa.

4.6.2 Funções

As seguintes funções estão definidas

- LOW(expressão) retorna o byte baixo de uma expressão
- HIGH(expressão) retorna o segundo byte de uma expressão

- BYTE2(expressão) o mesmo que HIGH
- BYTE3(expressão) retorna o terceiro byte de uma expressão
- BYTE4(expressão) retorna o quarto byte de uma expressão
- LWRD(expressão) retorna os bits de 0-15 de uma expressão
- HWRD(expressão) retorna os bits de 16-31 de uma expressão
- PAGE(expressão) retorna os bits de 16-21 de uma expressão
- EXP2(expressão) retorna $2^{\text{expressão}}$
- LOG2(expressão) retorna a parte inteira de $\log_2(\text{expressão})$

4.6.3 Operadores

A linguagem assembler suporta um grande número de operadores, que serão descritos a seguir. A prioridade é por ordem de precedência. Expressões entre parênteses serão avaliadas primeiro.

4.6.4 Operação de negação lógica

símbolo: !

Descrição: Operador unário, que retorna um se a expressão é zero e retorna zero se a expressão for diferente de zero.

Precedência: 14

Exemplo: ldi r16, !0xf0 ; Carrega r16 com 0x00

4.6.5 Operação de negação bit a bit

símbolo: ~

Descrição: Operador unário que retorna o operador com todos os bits invertidos.

Precedência: 14

Exemplo: ldi r16, ~0xf0 ; carrega r16 com 0x0f

4.6.6 Sinal de menos unário

Símbolo: -

Descrição: Operador unário que retorna a negação aritmética de uma expressão.

Precedência: 14

Exemplo: ldi r16, -2 ; Carrega -2(0xfe) em r16

4.6.7 Multiplicação

símbolo: *

Descrição: Operador binário que retorna o produto entre dois operadores.

Precedência: 13

Exemplo: ldi r30, label*2 ; Carrega r30 com label*2

4.6.8 Divisão

símbolo: /

Descrição: Operador binário que retorna o quociente da expressão da esquerda dividida pela expressão da direita.

Precedência: 13

Exemplo: ldi r30,label/2 ; Carrega r30 com label/2

4.6.9 Adição

símbolo: +

Descrição: Operador binário que retorna a soma de duas expressões

Precedência: 12

Exemplo: ldi r30,c1+c2 ; Carregar r30 com c1+c2

4.6.10 Subtração

símbolo: -

Descrição: Operador binário que faz a subtração de duas expressões

Precedência: 12

Exemplo: ldi r17,c1-c2 ;Carrega r17 com c1-c2

4.6.11 Deslocamento para a esquerda

símbolo: <<

Descrição: Operador binário que retorna a expressão da esquerda deslocada o número de bits da expressão da direita para a esquerda.

Precedência: 11

Exemplo: ldi r17,1<<bitmask ;Carrega r17 com 1 deslocado bitmask vezes para a esquerda

4.6.12 Deslocamento para a direita

símbolo: >>

Descrição: Operador binário que retorna a expressão da esquerda deslocada o número de bits da expressão da direita para a direita.

Precedência: 11

Exemplo: ldi r17,c1>>c2 ;Carrega r17 com c1 deslocado c2 vezes para a direita

4.6.13 Menor que

símbolo: <

Descrição: Operador binário que retorna um se a expressão a esquerda [e menor do que a expressão da direita, senão retorna zero.

Precedência: 10

Exemplo: ori r18,bitmask*(c1<c2)+1 ; Ou entre r18 e uma expressão.

4.6.14 Menor ou igual que

símbolo: <=

Descrição: Operador binário que retorna um se a expressão da esquerda é menor ou igual à expressão da direita, senão retorna zero.

Precedência: 10

Exemplo: `ori r18, bitmask*(c1<=c2)+1 ;` Operação ou entre r18 e uma expressão.

4.6.15 Maior que

símbolo: `>`

Descrição: Operador binário que retorna um se a expressão a esquerda é maior que a expressão da direita, senão retorna zero.

Precedência: 10

Exemplo: `ori r18, bitmask*(c1>c2)+1 ;` Operação ou entre r18 e uma expressão.

4.6.16 Maior ou igual que

símbolo: `>=`

Descrição: Operador binário que retorna um se a expressão a esquerda é maior ou igual à expressão da direita, senão retorna zero.

Precedência: 10

Exemplo: `ori r18, bitmask*(c1>=c2)+1 ;` Operação ou entre r18 e uma expressão.

4.6.17 Igual

símbolo: `==`

Descrição: Operador binário que retorna um se os dois operadores forem igual, senão retorna zero.

Precedência: 9

Exemplo: `andi r19, bitmask*(c1==c2)+1 ;` Operação e entre r19 e uma expressão.

4.6.18 Diferente

símbolo: `!=`

Descrição: Operador binário que retorna um quando as duas expressões são diferentes, senão retorna zero.

Precedência: 9

Exemplo: `.SET flag=(c1!=c2) ;` Atribui 1 ou zero a flag dependendo dos valores de c1 e c2.

4.6.19 Operação e bit a bit

Símbolo: `&`

Descrição: Operador binário que retorna o resultado da operação e entre duas expressões.

Precedência: 8

Exemplo: `ldi r18, High(c1&c2) ;` Carrega r18 com o a expressão

4.6.20 Operação ou exclusivo bit a bit

símbolo: `^`

Descrição: Operador binário que retorna o resultado da operação ou exclusivo entre duas expressões.

Precedência: 7

Exemplo: ldi r18,Low(c1^c2) ; Carrega r18 com o resultado da expressão.

4.6.21 Operação ou bit a bit

símbolo: |

Descrição: Operador binário que retorna o resultado da operação ou entre duas expressões.

Precedência: 6

Exemplo: ldi r18,Low(c1|c2) ; Carrega r18 com o resultado da expressão.

4.6.22 Operação Logica e

Símbolo: &&

Descrição: Operador binário que retorna um se ambas as expressões forem diferentes de zero, senão retorna zero.

Precedência: 5

Exemplo: ldi r18,Low(c1&&2) ; Carrega r18 com o resultado da expressão.

4.6.23 Operação Logica ou

Símbolo: ||

Descrição: Operador binário que retorna um se uma ou ambas as expressões forem diferentes de zero, senão retorna zero.

Precedência: 4

Exemplo: ldi r18,Low(c1||c2) ; Carrega r18 com o resultado da expressão.

4.7 Exercícios

1. Desenvolva um programa em linguagem assembler que faz a soma de dois números de 8 bits. Os números serão recebidos nas portas C e D, e o resultado deve ser enviado a porta B.

AULA 5 - O AMBIENTE AVRSTUDIO

Estudo do conjunto de ferramentas para desenvolvimento de programas

5.1 Objetivo

O objetivo desta aula é apresentar aos alunos o ambiente de programação e simulação de microcontroladores chamado AVR Studio®. Após esta aula o aluno deverá ser capaz de adquirir, instalar e operar as ferramentas necessárias à programação dos microcontroladores da família AVR, bem como desenvolver e simular os programas.

5.2 O software AVR Studio®

O AVR Studio® é um software disponibilizado gratuitamente pela ATMEL, e serve para programar e simular os microcontroladores da família AVR. Esta ferramenta compila programas em linguagem assembler e linguagem C. A última versão deste software pode ser adquirida no site www.atmel.com.

O processo de instalação do AVR Studio® é um pouco demorado, pois o software necessita instalar algumas bibliotecas para que tudo funcione adequadamente. Em alguns sistemas é necessário reiniciar o computador para concluir a instalação. A Figura 30 apresenta a tela de abertura deste software.

Figura 30: Tela de abertura do AVR Studio

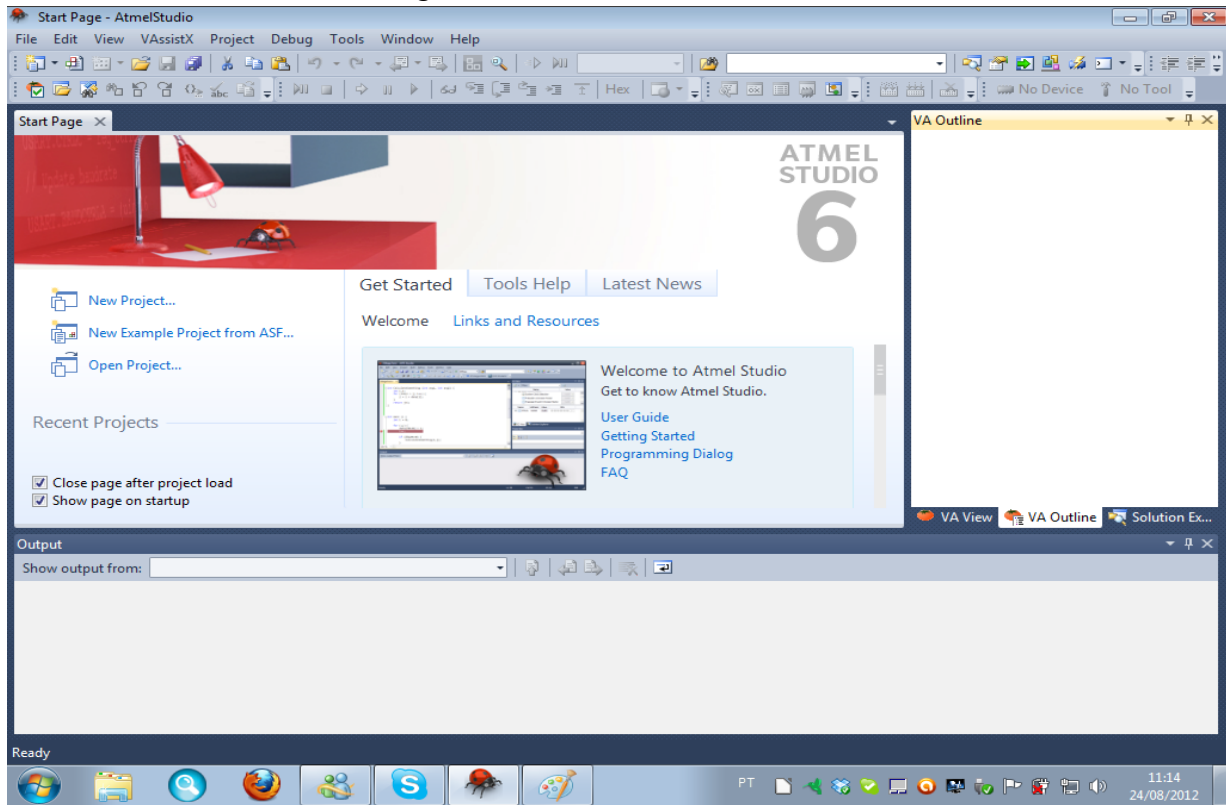


5.3 Criando projetos

Um programa de microcontrolador pode conter vários arquivos, assim para que não haja confusão o ambiente de desenvolvimento exige que se crie um projeto para cada programa. Os projetos são necessários, pois é neles que ficam armazenadas algumas informações necessárias à compilação do programa, como por exemplo, o modelo do microcontrolador usado. Nas próximas páginas temos um tutorial que ensina a criar um novo projeto.

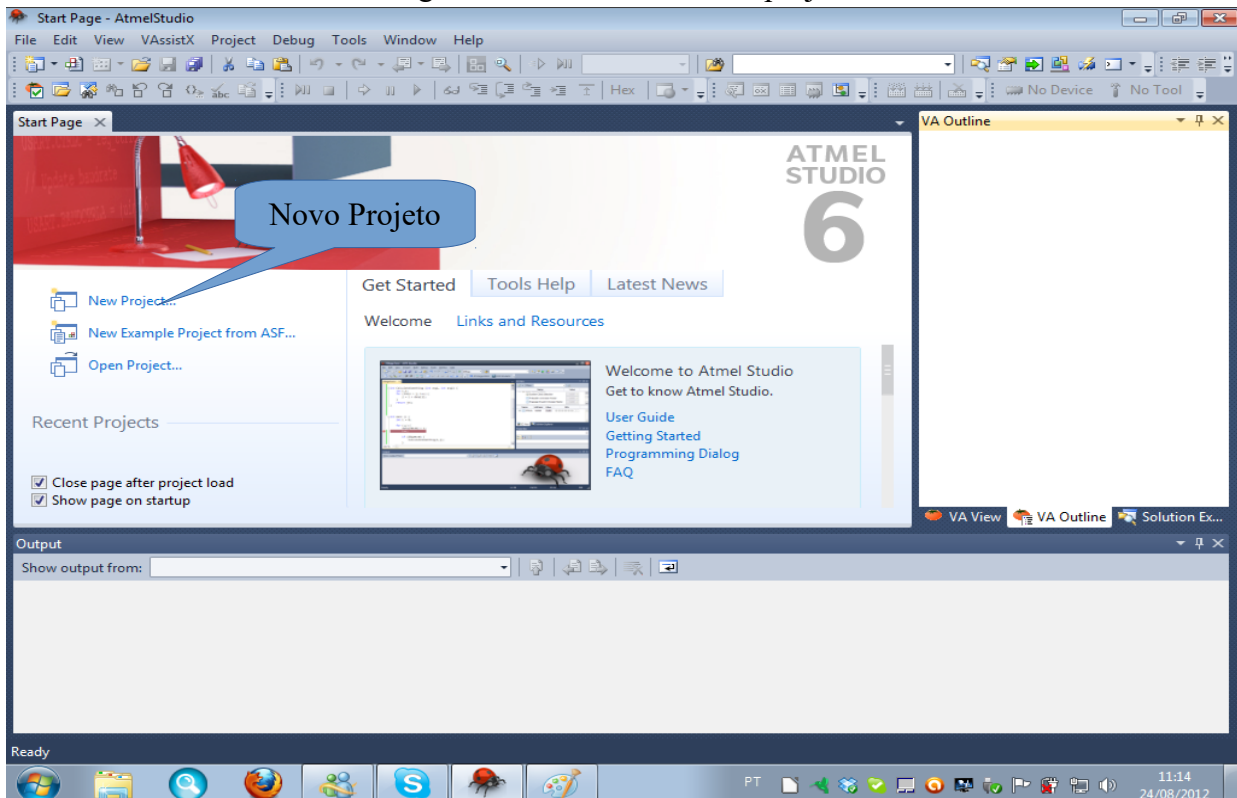
Passo 1: Através do menu iniciar abra o programa AVR Studio, aguarde até o programa carregar, conforme a Figura 31.

Figura 31: Interface do AVR Studio



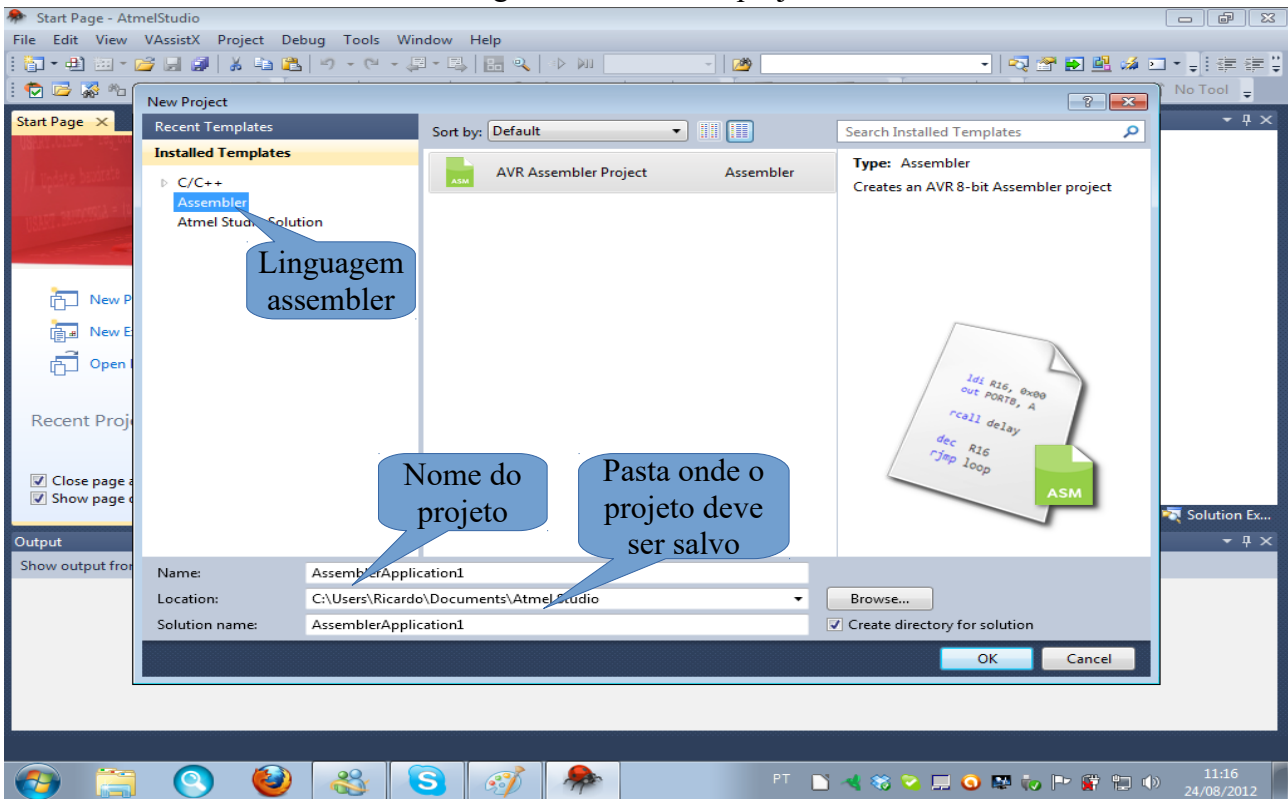
Passo 2: Seleção o link *New Project*, conforme indicado na Figura 32.

Figura 32: Criando um novo projeto



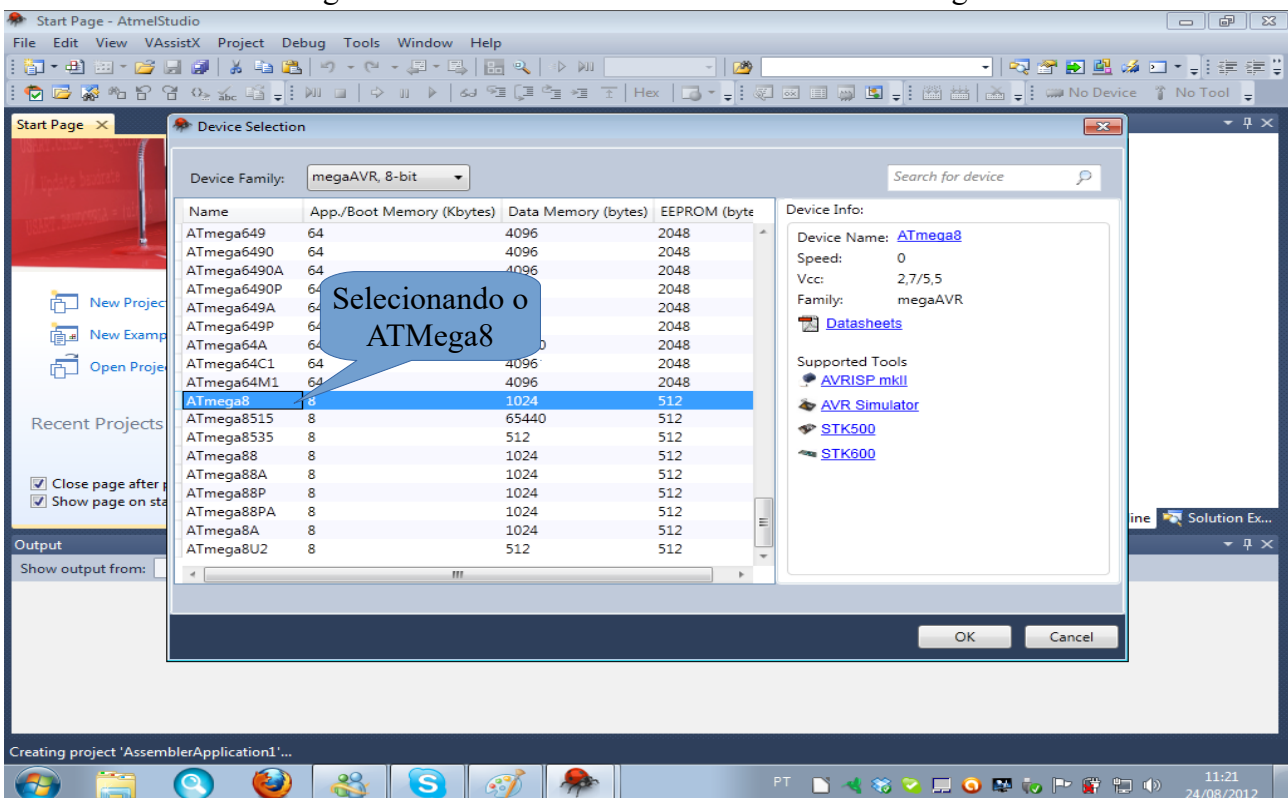
Passo 3: Preencha os dados do projeto conforme a Figura 33 e clique OK.

Figura 33: Dados do projeto



Passo 4: Selecione o microcontrolador ATmega8 conforme a Figura 34 e clique OK.

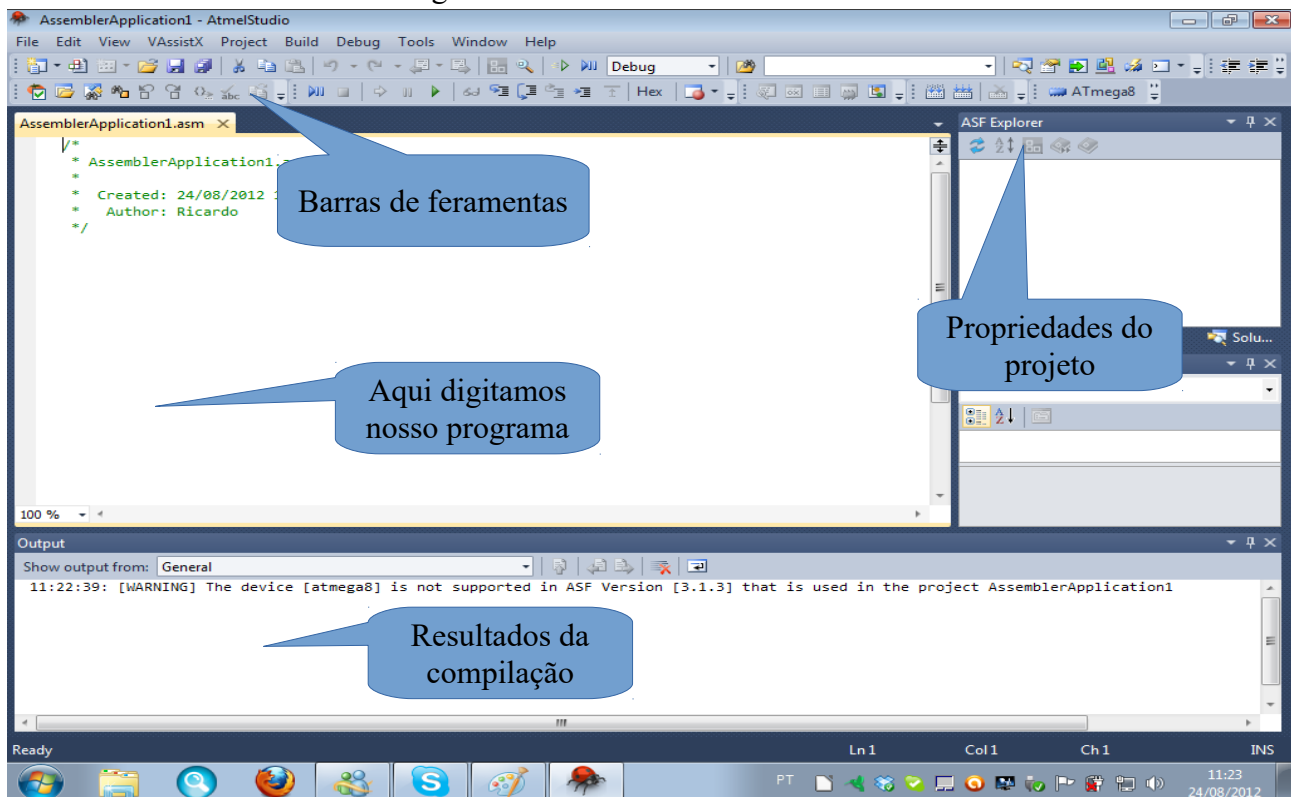
Figura 34: Selecionando o microcontrolador ATmega8



5.4 Ambiente de trabalho

Agora que o projeto foi criado o ambiente de trabalho está pronto para que seja digitado o programa. A Figura 35 mostra o ambiente de trabalho e suas partes.

Figura 35: O ambiente do AVR Studio



A seguir temos um programa para treinar o procedimento de compilação e simulação de programas.

```

.include "m8def.inc"
.equ INPORT      = PIND
.equ OUTDDR     = DDRC
.equ OUTPORT    = PORTC
.def zero       = r1
.def temp       = r18

.cseg
.org 0          ; inicia o segmento de código
rjmp reset     ; salta para o inicio
.org INT_VECTORS_SIZE ; define o local do inicio do programa
reset:
clr zero
ldi temp, 1

```

```

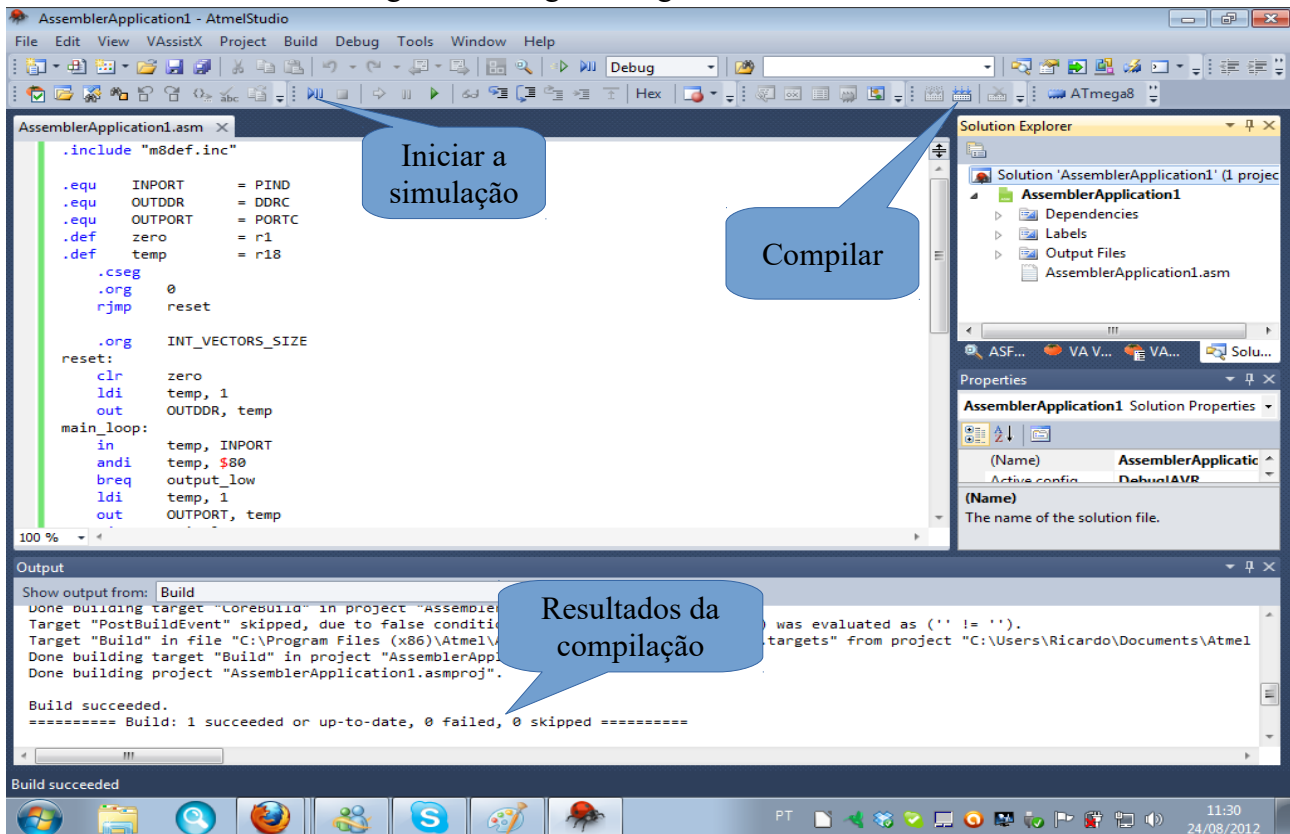
out   OUTDDR, temp           ; define o bit 0 da porta C como saída
main_loop:
in    temp, INPORT
andi  temp, $80              ; verifica o bit 7 da entrada
breq  output_low            ; se for 1 ativa a saída
ldi   temp, 1
out   OUTPORT, temp
rjmp  main_loop
output_low:
out   OUTPORT, zero
rjmp  main_loop             ; salta para o início do laço

```

Este programa deve ser digitado no AVR Studio® para que possamos executar a compilação e a simulação, conforme os passos a seguir.

Passo 1: Digite o programa e clique no botão indicado na Figura 36 para compilá-lo. Em seguida verifique o resultado da compilação. Se a compilação obteve sucesso inicie a simulação.

Figura 36: Programa digitado no AVR Studio

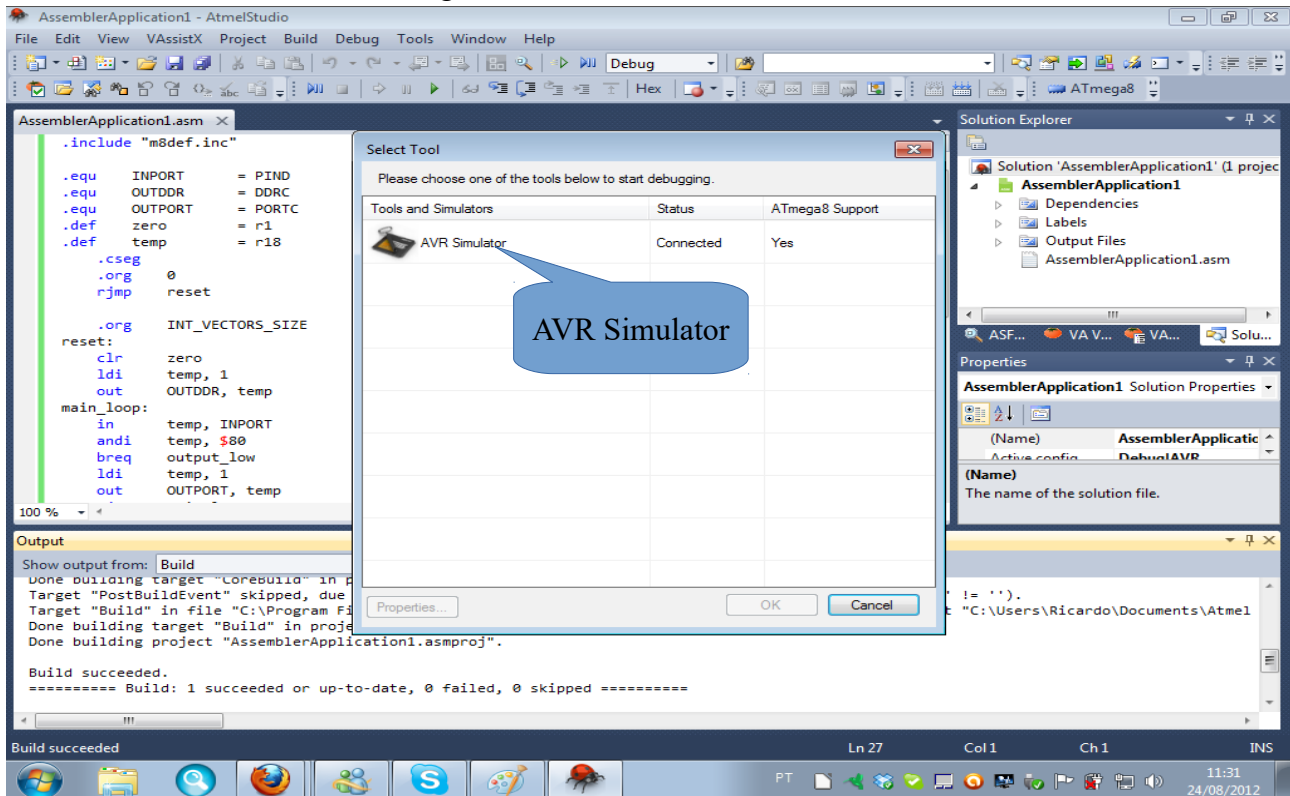


Para iniciar a simulação clique no botão indicado na figura.

Se for a primeira vez que o projeto é simulado, o compilador ira mostrar uma janela pedindo qual o simulador que desejamos utilizar.

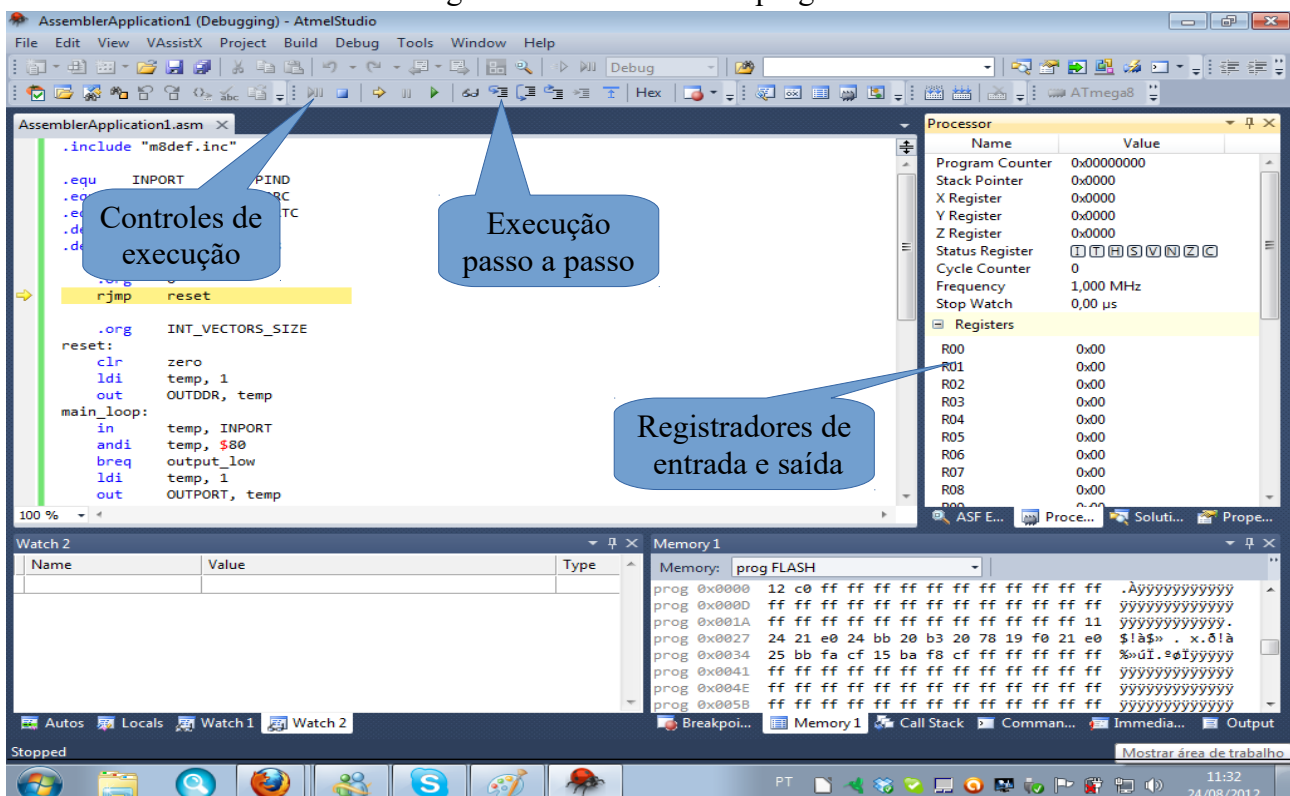
Passo 2: Selecciona a opção AVR Simulator como simulador para o projeto, como mostrado na Figura 37 e clique OK

Figura 37: Seleccionando o simulador



Passo 3: Para executar a simulação clique no botão indicado na Figura 38.

Figura 38: Simulando um programa



Utilize os controles para pausar e executar, ou executar passo a passo seu programa. Visualize e edite as saídas no painel lateral.

Com um pouco de prática o processo de compilação e simulação se torna natural. A simulação é uma etapa importante no desenvolvimento de um programa, pois evita que erros de programação danifiquem os circuitos elétricos.

5.5 Exercícios

1. Crie um novo projeto de forma que o microcontrolador usado seja o ATmega8 e que os arquivos do projeto fiquem dentro da pasta “Documentos”.
2. Utilizando o projeto anterior digite o seguinte programa. Faça a compilação e simule este programa.

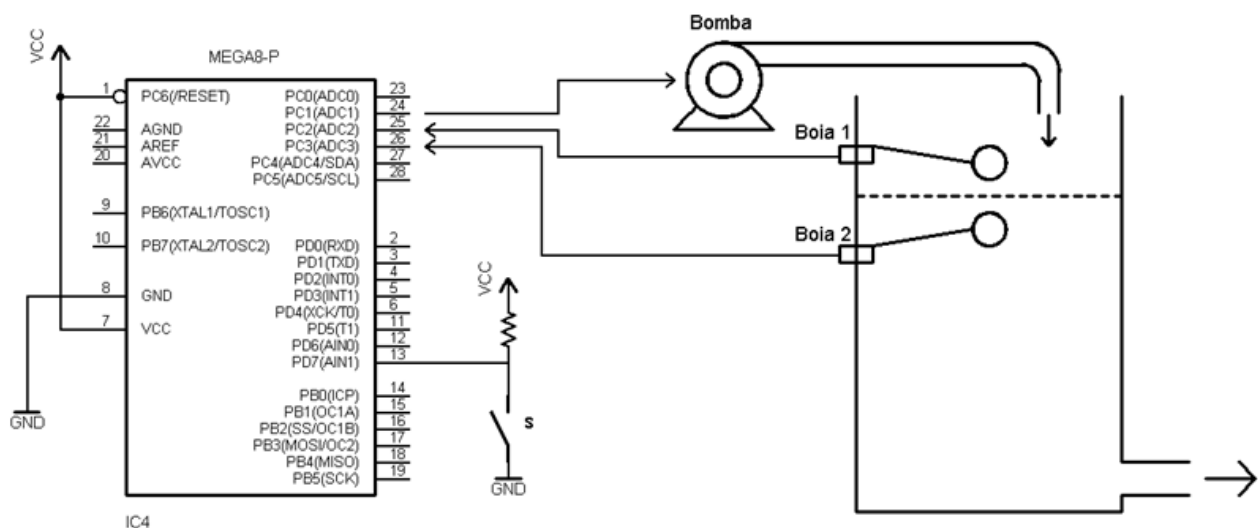
```
.include "m8def.inc"  
  
.dseg  
variavel:  
.BYTE 1 ; reserva 1 byte RAM  
  
.cseg  
.org 0  
rjmp reset  
  
.org INT_VECTORS_SIZE  
reset:  
ldi r16, 65  
sts variavel, r16  
in r16, PINB  
ldi r30, low(variavel)  
ldi r31, high(variavel)  
ld r17, Z  
add r16,r17  
out PORTD, r16  
rjmp reset
```

3. Simule o programa anterior usando a janela inferior direita para inserir valores na porta B (insira os valores no registrador “PINB”) e verifique o resultado do programa na porta D. Utilizando este método preencha a tabela a seguir.

Entrada (PINB)	Saída (PORTD)
1	
2	
5	
12	
25	
240	

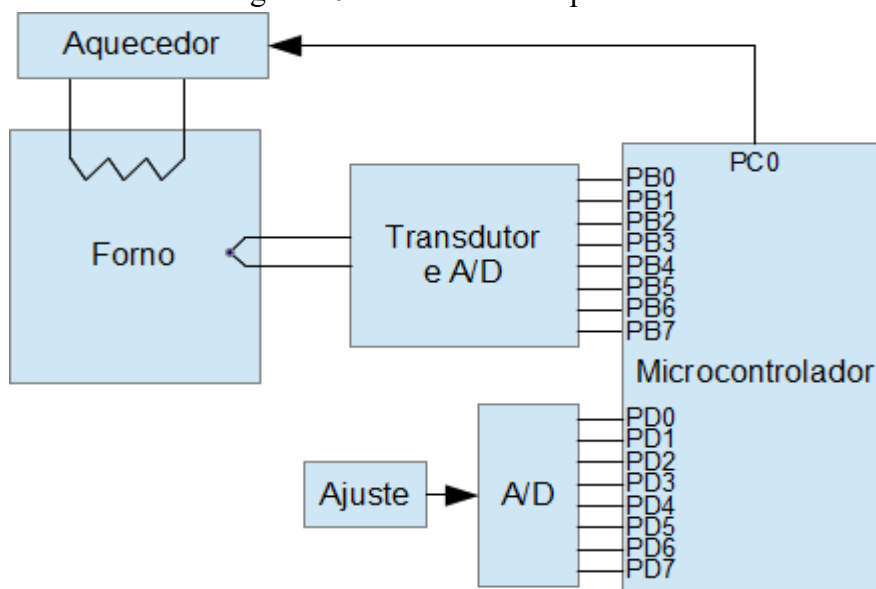
4. A Figura 39 apresenta uma planta de controle de nível. A bomba está ligada ao pino um da porta C, a boia um está ligada ao pino dois da porta C e boia dois está ligada ao pino três da porta C. O funcionamento é simples, quando o interruptor S está fechado o microcontrolador monitora as duas entradas das boias, se o nível estiver acima da boia 1 a bomba é desligada e se o nível estiver abaixo da boia 2 a bomba é ligada. Ambas boias enviam sinal lógico 1 quando o nível estiver acima delas e a bomba liga em nível lógico 1. Faça um programa para controlar esta planta.

Figura 39: Controle de nível



5. A Figura 40 apresenta um controle de temperatura, onde a temperatura medida é enviada a porta B e o valor de ajuste é enviado a porta D. Faça um programa que liga o aquecedor através do pino C0, quando a temperatura estiver 2 graus a baixo do ajuste e desliga quando a temperatura estiver 2 graus acima do ajuste.

Figura 40: Controle de temperatura



AULA 6 -DESENVOLVIMENTO DE PROGRAMAS EM C

Desenvolvimento de programas em C para microcontroladores AVR

6.1 Objetivo:

Esta aula vai mostrar aos alunos como a família AVR de microcontroladores pode ser programada na linguagem C. Não é objetivo desta aula ensinar a linguagem C, mas sim demonstrar as diferenças entre o a linguagem C para computadores e a linguagem C para microcontroladores AVR.

6.2 Características da programação de microcontroladores

A programação de microcontroladores é diferente da programação de computadores, a principal diferença é que no microcontrolador normalmente não existe um sistema operacional. Se não existe um sistema operacional então o programa nunca pode terminar, pois não existe mais nada para controlar o hardware. Outra diferença importante é com relação aos dispositivos de entrada e saída, que no computador são padronizados, tais como teclado e monitor. Nos microcontroladores os dispositivos de entrada e saída dependem do projeto e podem variar muito, então para cada projeto o programador tem que desenvolver uma interface personalizada com o hardware.

6.3 Modelo de programa

A seguir esta um modelo de programa que ajuda na criação de novos projetos.

```
#include <avr/io.h> // nomes dos pinos de i/o
// variaveis globais
// subrotinas

int main() // principal
{
// variaveis locais
// inicializações
while(1) // loop eterno
{
}
}
```

Este pequeno modelo utiliza um comando “**while**” com o parâmetro “1” para formar um loop sem fim, pois assim o programa não termina nunca evitando problemas para o microcontrolador. As palavras após “//” são comentários, e servem apenas para ajudar o

programador. É sempre um hábito saudável usar comentários nos programas, pois isso facilita a manutenção e a reutilização do código.

6.4 Uma pequena revisão

Para facilitar a compreensão dos próximos assuntos será feita uma pequena revisão sobre a linguagem C, já focando sua aplicação nos microcontroladores AVR.

6.5 Declaração de variáveis

A declaração de variáveis em C é feita declarando-se primeiro o tipo da variável depois o nome e terminando com ponto e vírgula. Como por exemplo:

```
int contador; // declara uma variável do tipo int com o nome contador.
```

A seguir estão os tipos de dados mais usados.

Tipo	Numero de Bits	Numero de Bytes	Faixa de valores
char	8	1	-128 a 127
unsigned char	8	1	0 a 255
int	16	2	-32.768 a 32.767
unsigned int	16	2	0 a 65535
long	32	4	-2.147.483.648 a 2.147.483.647
unsigned long	32	4	0 a 4.294.967.295
float	32	4	1.175494351 E – 38 a 3.402823466 E + 38
double	64	8	2.2250738585072014 E – 308 a 1.7976931348623158 E + 308

6.6 Operadores

Para que possamos compreender o funcionamento dos programas em C é necessário compreendermos o funcionamento dos operadores. Existem vários tipos de operadores, como, operadores de atribuição, de comparação e operadores matemáticos. A tabela a seguir mostra os operadores mais comuns.

Operador	Exemplo	Descrição
++	a++	Incrementa o valor de a (a=a+1)
--	a--	Decrementa o valor de a (a=a-1)
+	a + b	a mais b
-	a - b	a menos b
*	a * b	a vezes b

/	a / b	a dividido por b
%	a % b	Resto de a/b
>>	a >> b	A deslocado b bits para a direita
<<	a << b	A deslocado b bits para a esquerda
<	a < b	Testa se a é menor que b
>	a > b	Testa se a é maior que b
<=	a <= b	Testa se a é menor ou igual a b
>=	a >= b	Testa se a é maior ou igual a b
==	a == b	Testa se a é igual a b
!=	a != b	Testa se a é diferente de b
&	a & b	operação AND entre a e b
	a b	operação OR entre a e b
^	a ^ b	operação XOR entre a e b
&&	a && b	operação AND entre a e b usado dentro de IF, while, etc.
	a b	operação OR entre a e b usado dentro de IF, while, etc.
!	!a	Operação NOT em a
=	a = b	a recebe o valor de b
+=	a += b	a recebe o valor de a + b
-=	a -= b	a recebe o valor de a - b
*=	a *= b	a recebe o valor de a * b
/=	a /= b	a recebe o valor de a / b
%=	a %= b	a recebe o resto de a / b
>>=	a >>= b	a recebe a deslocado b bits para a direita
<<=	a <<= b	a recebe a deslocado b bits para a esquerda
&=	a &= b	a recebe a AND b
=	a = b	a recebe a OR b
^=	a ^= b	a recebe a XOR b

Operadores comuns da linguagem C

6.7 Estruturas de controle

A seguir faremos uma revisão das principais estruturas de controle usadas na linguagem C. Estruturas de controle são aquelas que mudam o fluxo do programa baseado em informações como contagens e comparações. São estas estruturas de controle que realmente tomam as decisões, portanto é importante compreendê-las.

6.8 if

O comando **if** executa um bloco de código se a condição for verdadeira. Ele pode ter duas sintaxes, uma para um único comando como no exemplo a seguir.

```
if(a==3) b=10;
```

E outra sintaxe para um bloco de código.

```
if(a==3)
{
  b=23;
  c=12;
}
```

6.9 If / else

O comando **if / else** é derivado do comando **if**, a única diferença é que se a condição for falsa o bloco **else** é executado. Veja os exemplos.

```
if(a==3) b=10;
else b=0;
```

```
if(a==3)
{
  b=23;
  c=12;
}
else
{
  b=0;
  c=0;
}
```

6.10 while

O comando **while** fica repetindo um bloco de código enquanto a condição for verdadeira. Veja os exemplos.

```
while(contador<30) contador++;
```

Ou para um bloco com mais de um comando.

```
while(a<3)
{
  contador++;
  a=2*contador;
}
```

6.11 for

O comando **for** executa um bloco de código um determinado numero de vezes, dependendo do valor de seus três parâmetros. O **for** necessita de uma variável para fazer a contagem, o primeiro parâmetro é a inicialização desta variável, o segundo parâmetro é a comparação que mantém o **for** trabalhando, e o terceiro parâmetro é a condição de incremento. Veja os exemplos.

```
int cont;  
for(cont=0; cont<10; cont++)  
{  
    a=cont+3;  
    b=a/2;  
}
```

6.12 Sub-rotinas

Outra facilidade que a linguagem C tem é o uso de sub-rotinas. Tarefas que se repetem muitas vezes podem ser transferidas para uma sub-rotina, assim o código ocupa menos memória e fica mais fácil de compreender. Veja o exemplo a seguir.

```
float perimetro(float raio)  
{  
    float resposta;  
    resposta=2*PI*raio;  
    return(resposta);  
}
```

A sub-rotina acima recebe como parâmetro o valor do raio e retorna o valor do perímetro. Para utilizar a sub-rotina basta chamá-la diretamente no programa como no exemplo a seguir:

```
valor=perimetro(23.35);
```

6.13 Exercícios

1. Crie um programa que conta de 0 a 100 usando o comando **for**. O resultado desta contagem deve ser enviado para o registrador **PORTB**;
2. Utilizando o comando **while** faça um programa que fica incrementando uma variável chamada “**cont**” enquanto o valor do registrador **PIND** for igual a zero.
3. Construa uma sub-rotina que recebe o valor de uma variável do tipo **long**, decrementa o valor desta variável de 1 em 1 e retorna quando a variável for igual a zero.

AULA 7 - ENTRADAS E SAÍDAS DIGITAIS

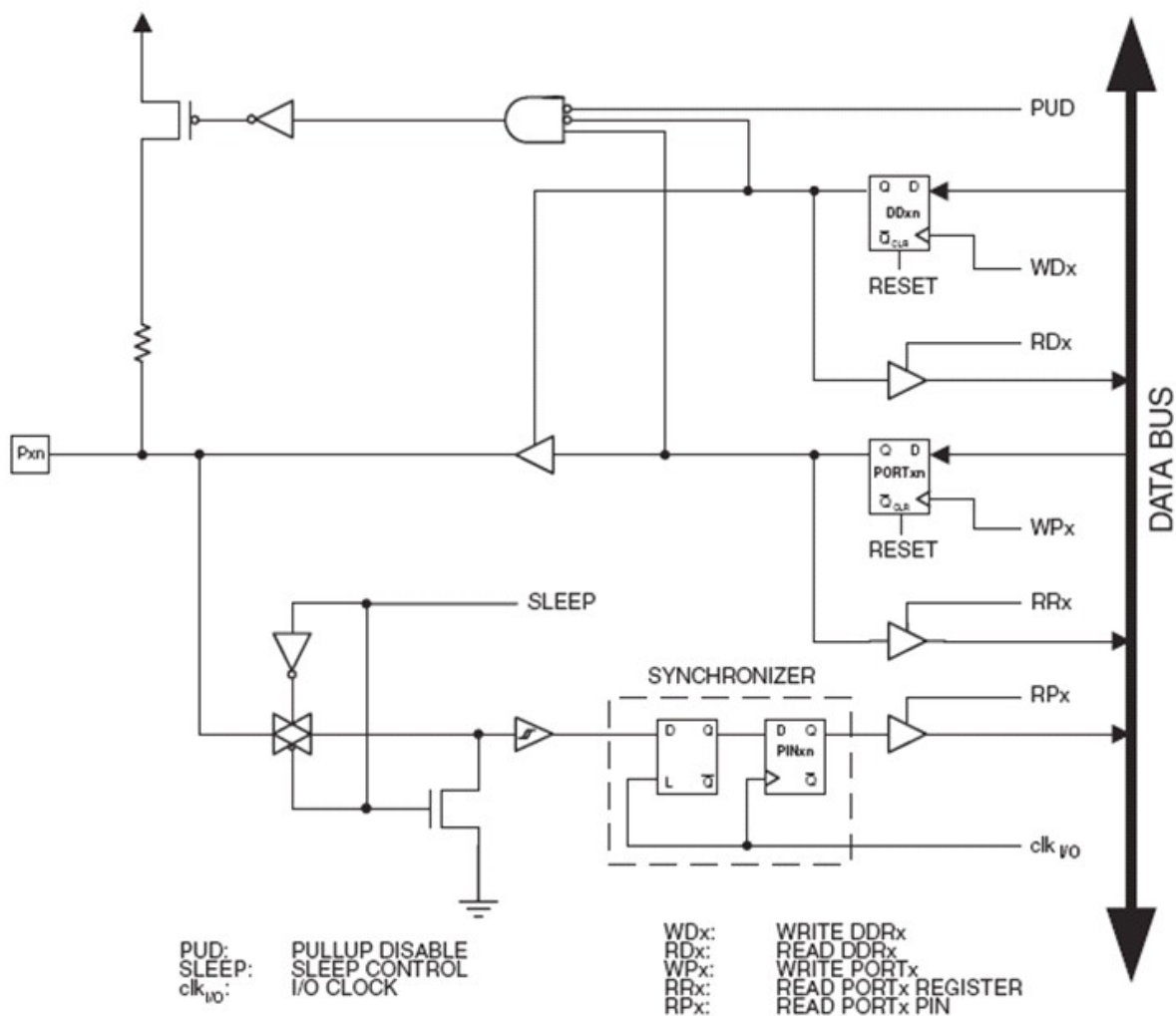
Estudo das entradas e saídas digitais nos microcontroladores AVR

7.1 Objetivo:

Os microcontroladores têm normalmente um grande número de entradas e saídas digitais, porém para que possam ser usadas é necessário que se utilizem circuito de interligação. O objetivo desta aula é estudar os circuitos físicos e o software utilizados na interligação do microcontrolador e outros componentes de entrada e saída.

7.2 Estrutura interna das entradas e saídas na família AVR

A figura a seguir mostra o circuito interno de um pino típico de entrada e saída de um microcontrolador.



Estrutura de um pino de entrada e saída do microcontrolador.

Observando a figura anterior podemos notar que a complexidade de cada pino não é pequena, mas por outro lado esta complexidade nos dá uma flexibilidade muito grande. Cada um

dos pinos pode assumir vários modos de funcionamento, por exemplo, entrada, saída, etc.

7.3 Registradores de controle dos pinos de entrada e saída

O controle dos pinos de entrada e saída é feito basicamente através de três registradores. A tabela a seguir lista estes registradores.

DDRx	Registrador de direção (o bit em um (1) indica saída).
PORTx	Registrador que contem o valor enviado a porta x.
PINx	Registrador que recebe o valor a ser enviado a porta.
x = nome da porta (B, C, D etc.)	

Cada um dos registradores é responsável por toda uma porta, assim se for necessário configurar apenas um pino temos que acessar todo o registrador e garantir que os outros pinos não tenham seu valor alterado. O mapeamento dos pinos de cada uma das portas nos registradores é direto, assim se, por exemplo, queremos acessar o pino três da porta D, devemos acessar o bit três do registrador PIND.

7.4 Alteração de um pino individual de saída

A linguagem C nos fornece ferramentas para acessar um bit individualmente em um registrador como no exemplo a seguir onde é gravado o valor um no pino cinco da porta B.

```
PORTB=PORTB | 0b00100000; // grava 1 em PORTB5
```

Ou de forma reduzida,

```
PORTB|=0b00100000; // grava 1 em PORTB5
```

Como pode ser visto acima usamos a operação “ou” para gravar um em um único pino.

Já para gravar zero em um determinado pino é comum usarmos a operação “e” com um valor que tenha zero apenas no bit que queremos atuar.

```
PORTB=PORTB & 0b11110111; // grava 0 em PORTB3
```

Ou de forma reduzida,

```
PORTB&=0b11110111; // grava 0 em PORTB3
```

Utilizando estas técnicas é possível alterar apenas um bit em qualquer registrador facilitando assim o comando de apenas um pino.

7.5 Leitura de um pino de entrada

Para a leitura de um único pino também podemos usar a operação “e” como no exemplo a seguir.

```
if((PINC & 0b00000001)!=0) a=0; // se o pino PINC0 for igual a 1 “a” recebe 0
```

7.6 Registradores de controle das portas

As portas dos microcontroladores podem ser configuradas como entrada ou saída também podem ser conectadas a um resistor de polarização positiva, chamado “Pull-up”. A tabela a seguir mostra como os registradores comandam o estado de cada um dos pinos.

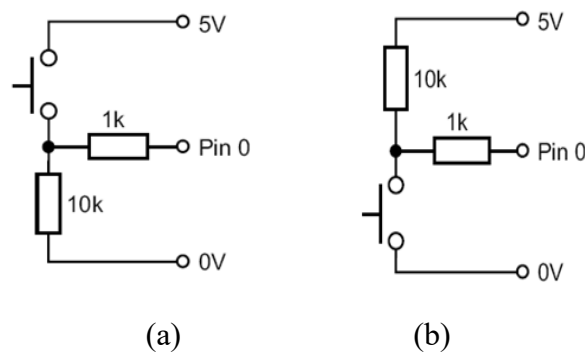
DDxn	PORTxn	I/O	Pull-up	Comentário
0	0	Entrada	Não	Terceiro estado
0	1	Entrada	Sim	Entrada com resistor para o Vcc
1	0	Saída	Não	Saída em zero
1	1	Saída	Não	Saída em um

Observando a tabela é possível notar que quando um pino está configurado como entrada o registrador PORTx serve para ligar ou desligar o resistor de pull-up. Este resistor é usado para polarizar o pino em determinadas situações onde o pino não pode ficar flutuando.

7.7 Interface entre o microcontrolador e os circuitos de entrada digital

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

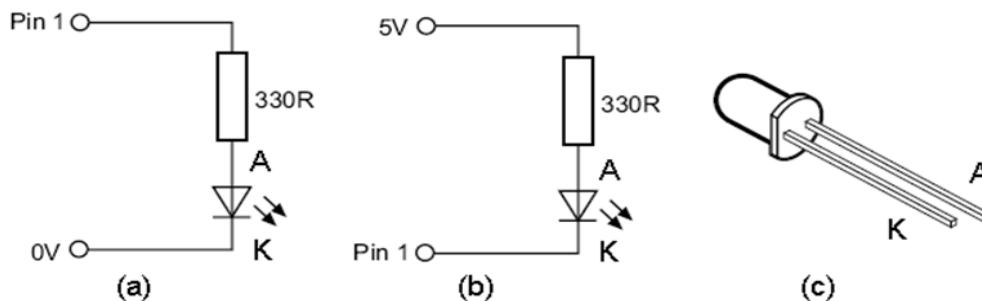
Entradas para botões. A figura a seguir apresenta dois modelos de conexão para botões.



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

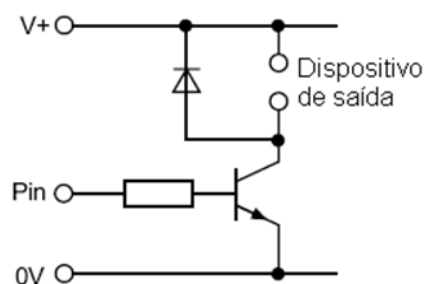
É sempre importante lembrar que na comutação de chaves mecânicas sempre pode ocorrer a geração de ruídos, para evitar que o microcontrolador interprete erroneamente o sinal é comum utilizar um pequeno intervalo de tempo após a detecção da mudança de estado da entrada.

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

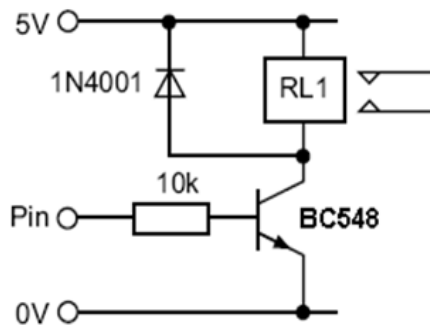
Já para cargas maiores que necessitam de mais corrente é necessário utilizar um circuito auxiliar. A figura a seguir apresenta um circuito genérico para este fim.



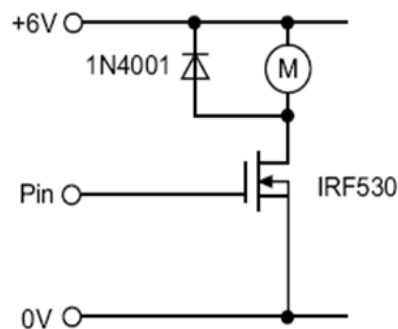
O circuito da figura aciona cargas com uma corrente mais elevada. O tipo de transistor e o valor do resistor dependem da corrente da carga. O diodo serve para proteger o circuito contra descargas.

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

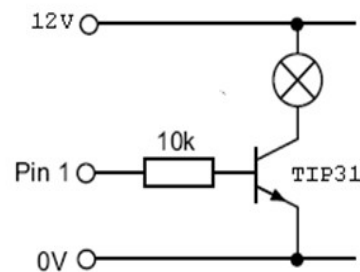
7.8 Conexão de reles.



7.9 Conexão de um motor utilizando transistor mosfet.



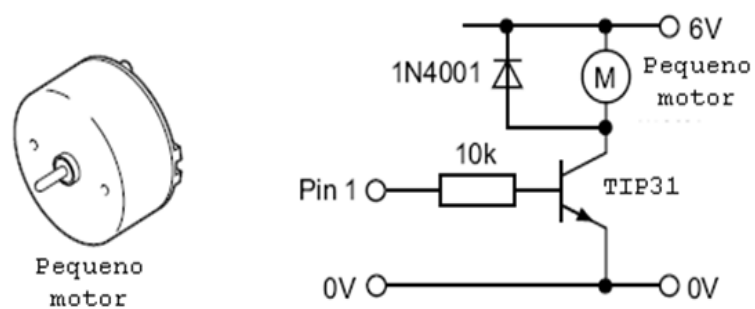
7.10 Conexão de lâmpada de sinalização



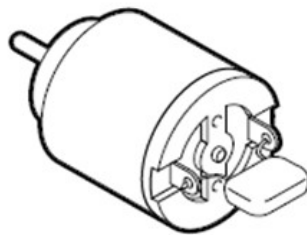
Se observarmos com atenção os exemplos anteriores veremos que a tensão que alimenta a carga é diferente da tensão do microcontrolador (5V), isso nos ajuda a interconectar cargas com tensões diferentes.

7.11 Conexão de pequenos motores

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

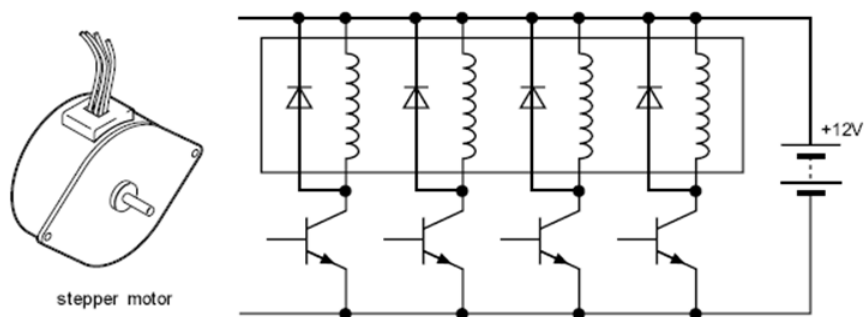


Alguns motores geram muitos ruídos que podem interferir no resto do circuito, assim é comum conectarmos em paralelo com o motor um capacitor cerâmico de 220nF, como na figura a seguir.



7.12 Conexão de motores de passo

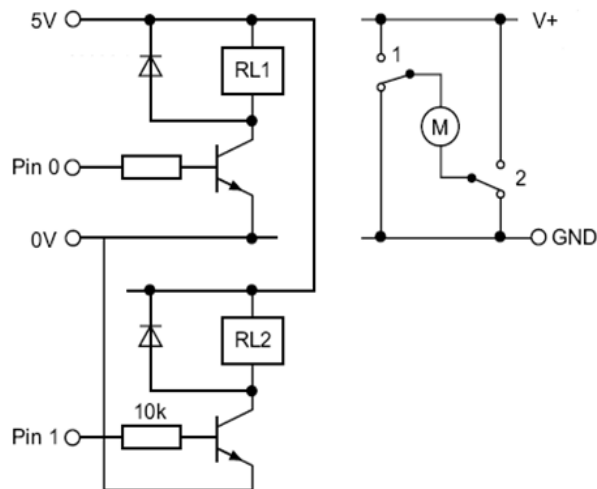
Existem configurações mais complexas, como por exemplo, a conexão de um motor de passo mostrada a seguir.



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 seqüência de pulsos com tempo de duração proporcional a velocidade, para maiores detalhes procure o manual do fabricante do motor.

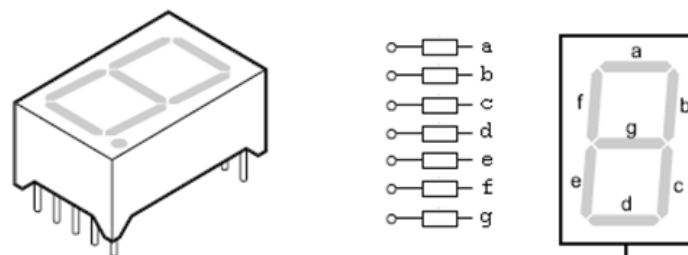
7.13 Conexão de motores cc com reversão

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



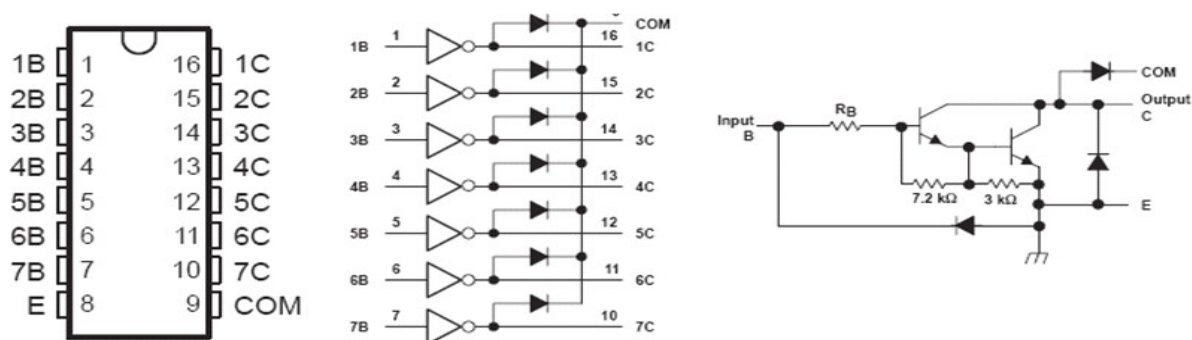
7.14 Conexão de displays de LED

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.



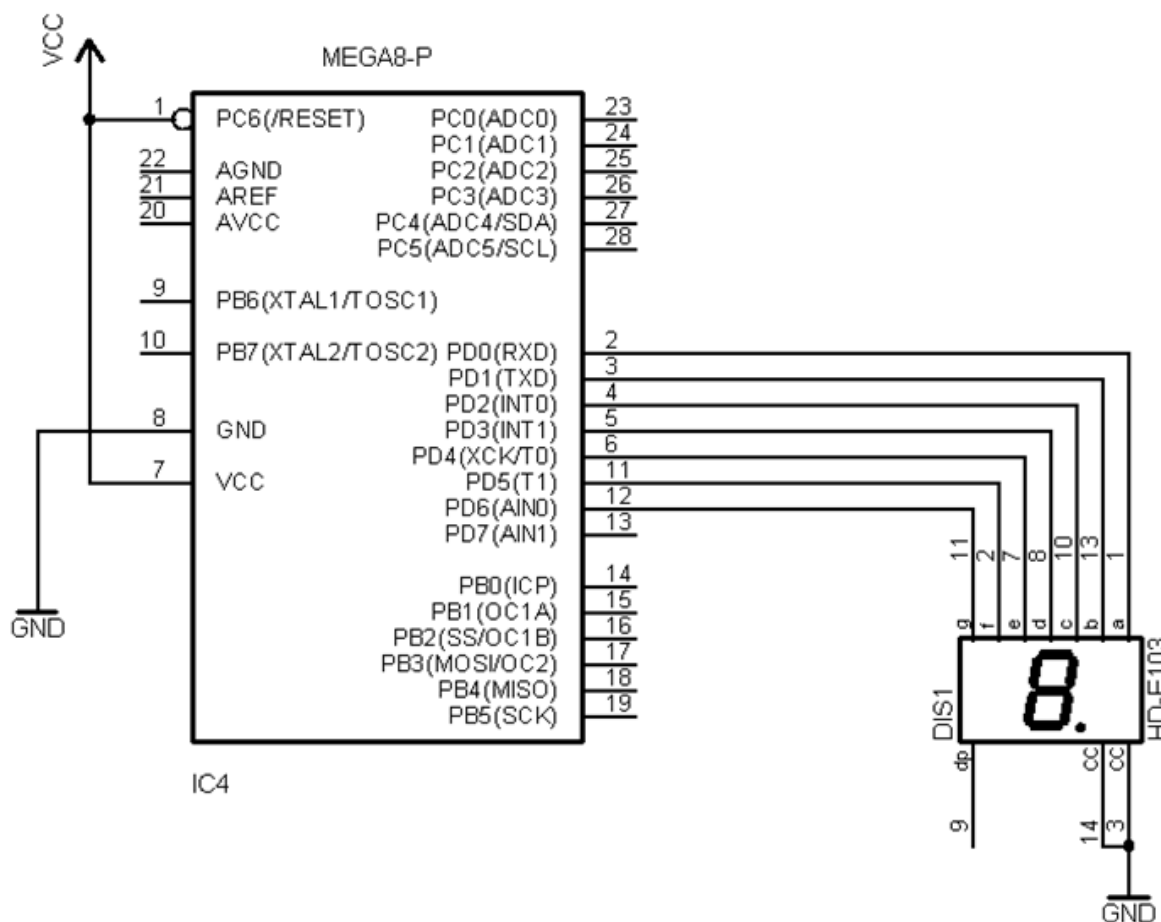
7.15 Circuitos especializados de saída

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 deles.



7.16 Exercícios

1. Desenhe um circuito que conecta dois botões e dois LEDs ao microcontrolador, os botões devem enviar nível lógico um quando pressionados e os dois LEDs devem acender quando receberem nível zero.
2. Faça um programa para o circuito do exercício anterior de modo que quando o primeiro botão for pressionado os dois LEDs acendam e quando o segundo botão for pressionado os dois LEDs se apagam.
3. Faça um programa que envia os números de um a nove repetidamente para o display de sete segmentos. O circuito é o da figura a seguir.



AULA 8 - AUTOMAÇÃO USANDO LINGUAGEM C

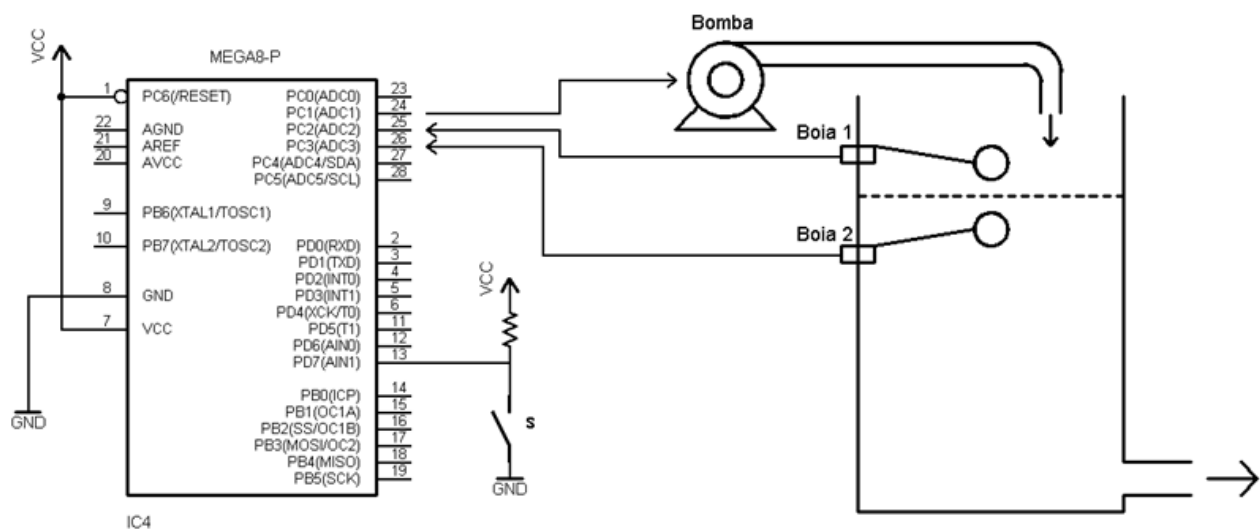
A aplicação da linguagem C na automação de processos industriais

8.1 Objetivo:

O objetivo desta aula é mostrar aos alunos como programar os microcontroladores em linguagem C para resolver problemas da área de automação industrial. O aprendizado deste conteúdo será conduzido através de exemplos e exercícios.

8.2 Exemplo de controle de nível

Para começarmos a entender como um microcontrolador programado em C pode ser usado na automação de um processo veremos um exemplo. A figura a seguir mostra um sistema de controle de nível com duas boias e uma bomba.



Controle de nível.

A bomba está ligada ao pino um da porta C, a boia um está ligada ao pino dois da porta C e boia dois está ligada ao pino três da porta C. O funcionamento é simples, quando o interruptor S está fechado o microcontrolador monitora as duas entradas das boias, se o nível estiver acima da boia 1 a bomba é desligada e se o nível estiver abaixo da boia 2 a bomba é ligada. Ambas boias enviam sinal lógico 1 quando o nível estiver acima delas e a bomba liga em nível lógico 1.

O programa a seguir foi desenvolvido para controlar o processo acima.

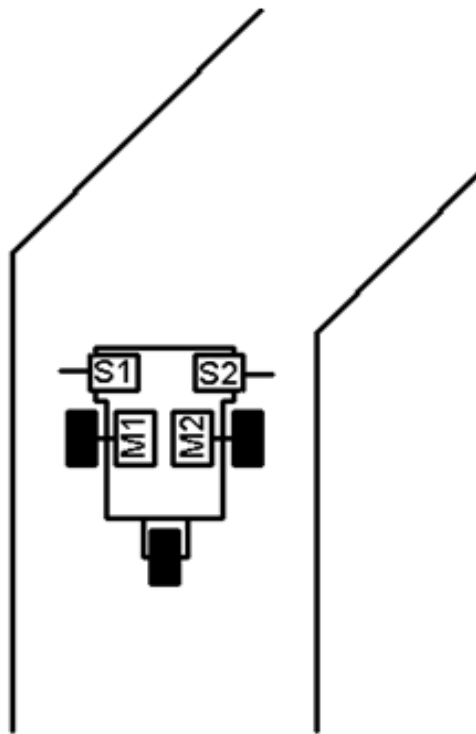

```
#include <avr/io.h> // inclui biblioteca padrão
int main()
{
DDRC=0b00000010; // define o pino dois da porta C como saída (bomba)
while(1)
{
if((PIND & 0b10000000)==0) // Le o estado da chave S
{
if((PINC & 0b00000100)!=0) //Se a bóia um envia um desliga a bomba
{
PORTC=PORTC & 0b11111101; // desliga a bomba
}
if((PINC & 0b00001000)==0) // Se a bóia dois envia zero liga a bomba
{
PORTC=PORTC | 0b00000010; // liga a bomba
}
}
else
{
PORTC=PORTC & 0b11111101; // se a chave S estiver aberta mantém
} // a bomba desligada
}
}
```

8.3 Exemplo de controle de um robô

Para o próximo exemplo utilizaremos um pequeno robô que deve se deslocar entre duas paredes sinuosas. O robô é composto por dois motores que comandam o deslocamento conforme tabela a seguir.

Motor 1	Motor 2	Ação do robô
Desligado	Desligado	Parado
Ligado	Desligado	Vira para a direita
Desligado	Ligado	Vira para a esquerda
Ligado	Ligado	Vai para sempre

A figura a seguir mostra o posicionamento dos dois motores e dos dois sensores no robô.



Robô utilizado no exemplo

O funcionamento do robô é simples, se o sensor S1 toca na parede ele envia nível lógico 1 para o microcontrolador, e este desliga o motor M2 para que o robô se afaste da parede. Se o sensor S2 tocar a parede o microcontrolador desliga o motor M1 para que o robô vire para a direita a se afaste da parede. Se nenhum dos sensores tocarem a parede, os dois motores são ligados para que o robô ande para frente. A tabela a seguir mostra a conexão dos sensores e motores com o microcontrolador.

Componente	Pino	Nível de sinal
Motor M1	PORTC0	Ligado em nível lógico 1.
Motor M2	PORTC1	Ligado em nível lógico 1.
Sensor S1	PORTD0	O nível lógico 1 indica que tocou a parede.
Sensor S2	PORTD1	O nível lógico 1 indica que tocou a parede.

O software para controlar este robô deve ficar monitorando os dois sensores para determinar se está tocando a parede em algum dos lados, caso um dos sensores enviando nível lógico 1 significa que o robô está em contato em um dos lados e o software deve comandar os motores para que ele mude de direção e se afaste desta parede.

A seguir esta um exemplo de software para ser usado neste robô.

```
#include <avr/io.h> // inclui biblioteca padrão

//Motor M1 PORTC0
//Motor M2 PORTC1
//Sensor S1 PORTD0
//Sensor S2 PORTD1

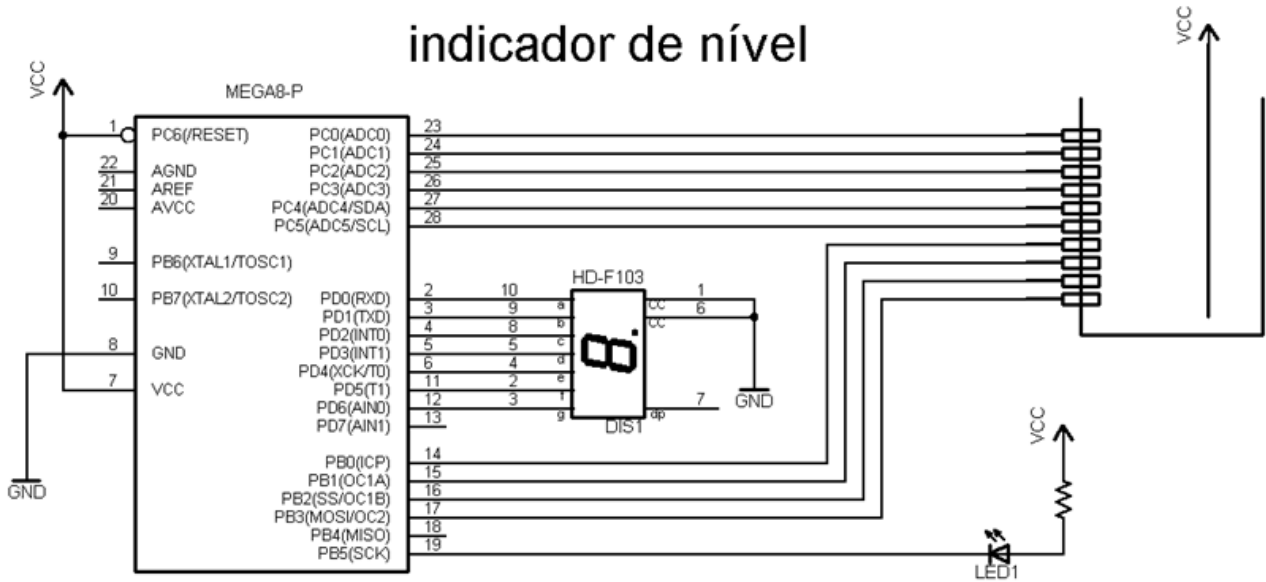
int main()
{
  DDRC=0b00000011; // define os pinos dos motores como saída

  while(1)
  {
    if((PIND & 0b00000001)==0) // se o sensor 1 é zero
    {
      PORTC=PORTC | 0b00000010; // liga o motor 2
    }
    else
    {
      PORTC=PORTC & 0b11111101; // senão desliga o motor 2
    }

    if((PIND & 0b00000010)==0) // se o sensor 2 é zero
    {
      PORTC=PORTC | 0b00000001; // liga o motor 1
    }
    else
    {
      PORTC=PORTC & 0b11111110; // senão desliga o motor 1
    }
  }
}
```

8.4 Exercícios

1. Para o circuito do desenho abaixo construa um programa que indica no display o nível de água no tanque. O tanque possui dez sensores que enviam nível lógico 1 quando a água os atinge, o software deve indicar o número de zero a nove conforme o número de sensores que a água atingir. Se a água atingir o décimo sensor o display deve indicar nove e o LED1 deve ser aceso para indicar alarme.



AULA 9 -INTERRUPÇÕES NOS MICROCONTROLADORES

Estudo das rotinas de interrupção nos microcontroladores AVR

9.1 Objetivo:

O objetivo desta aula é discutir o funcionamento das interrupções nos microcontroladores da família AVR. As interrupções são instrumentos muito úteis na programação dos microcontroladores, pois é através delas que um microcontrolador pode executar várias tarefas ao mesmo tempo.

9.2 O que são interrupções

Interrupções são desvios realizados no programa pelo hardware do microcontrolador, ou seja, uma sub-rotina do programa é chamada quando algum evento acontece no hardware do microcontrolador.

Existem aproximadamente 20 fontes de interrupção nos microcontroladores AVR, que serão listadas a seguir. Como exemplo podemos citar a interrupção que ocorre quando o conversor analógico / digital termina sua conversão.

Para que possamos fazer uso das interrupções no microcontroladores é necessário que o software faça alguns ajustes no hardware. Inicialmente devemos incluir a biblioteca <avr/interrupt.h> e providenciar a sub-rotina de interrupção. As sub-rotinas de interrupção tem o seguinte formato.

```
ISR(vetor de interrupção)
{
}

```

Onde “ISR” é o nome usado em todas as interrupções e o vetor de interrupção é o nome dado a cada uma das interrupções, conforme a tabela a seguir.

Nome do Vetor	Descrição
ADC_vect	Conversão completa do ADC
ANA_COMP_vect	Comparador Analógico
EE_RDY_vect	EEPROM pronta
INT0_vect	Interrupção externa 0
INT1_vect	Interrupção externa 1
SPI_STC_vect	Transferência serial completa na SPI
SPM_RDY_vect	Memória de programa pronta
TIMER0_OVF_vect	Estouro Timer/Counter 0

TIMER1_CAPT_vect	Captura Timer/Counter
TIMER1_COMPA_vect	Comparação do Timer/Counter 1 A
TIMER1_COMPB_vect	Comparação do Timer/Counter 1 B
TIMER1_OVF_vect	Estouro Timer/Counter 1
TIMER2_COMP_vect	Comparação do Timer/Counter 2
TIMER2_OVF_vect	Estouro Timer/Counter 2
USART_RXC_vect	Caractere recebido na USART
USART_UDRE_vect	Registrador de dados vazio na USART
USART_TXC_vect	Caractere transmitido na USART
TWI_vect	Interface Serial 2-wire

Após construir a sub-rotina de interrupção é necessário habilitarmos a interrupção desejada através do registrador de controle do hardware em questão. Cada dispositivo de hardware tem um registrador de controle específico, cujos detalhes podem ser encontrados no manual do dispositivo.

E o último passo na utilização das interrupções é habilitar o microcontrolador a executar as interrupções. Isto é feito chamando a função sei().

A seguir temos um exemplo onde as interrupções externas 0 e 1 são utilizadas.

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

int cont;

ISR(INT0_vect)
{
  cont++;
}

ISR(INT1_vect)
{
  cont--;
}

int main()
{
  MCUCR|=0b00001111; // int0 e int1 na subida do sinal
  GICR|=0b11000000; // ativa int0 e int 1
  sei(); // ativa todas as interrupções

  cont=0;

  while(1)
  {
  }
}
```

Neste exemplo foi utilizado o registrador “MCUCR”, onde são ajustados os níveis de tensão em que a interrupção deve acontecer. E o registrador GICR onde as interrupções são habilitadas. As tabelas a seguir detalham estes registradores.

O registrador MCUCR é o registrador de controle do microcontrolador.

Registrador MCUCR	
Bit	Significado
0	ISC00 - Sensibilidade INT0 bit 0
1	ISC01 - Sensibilidade INT0 bit 1
2	ISC10 - Sensibilidade INT1 bit 0
3	ISC11 - Sensibilidade INT1 bit 1
4	Modo de hibernação bit 0
5	Modo de hibernação bit 1
6	Modo de hibernação bit 2
7	Habilita a hibernação

Neste exemplo foram utilizados os 4 primeiros bits, as tabelas a seguir apresentam o significado dos registradores ISC00 a ISC11.

ISC01	ISC00	Significado
0	0	O nível baixo em INT0 gera a interrupção
0	1	Qualquer mudança em INT0 gera a interrupção
1	0	Uma borda de descida em INT0 gera a interrupção
1	1	Uma borda de subida em INT0 gera a interrupção

ISC11	ISC10	Significado
0	0	O nível baixo em INT1 gera a interrupção
0	1	Qualquer mudança em INT1 gera a interrupção
1	0	Uma borda de descida em INT1 gera a interrupção
1	1	Uma borda de subida em INT1 gera a interrupção

O registrador GICR é o registrador de controle geral das interrupções.

Registrador GICR	
Bit	Significado
0	IVSEL – escolha do vetor de interrupção
1	IVCE – Habilita troca de vetor de interrupção
2	-

3	-
4	-
5	-
6	INT0 - Habilita interrupção externa 0
7	INT1 - Habilita interrupção externa 1

No exemplo foram utilizados os bits 6 e 7, para habilitar as interrupções externas.

Para realizarmos os testes neste programa utilizaremos os pinos PD2 e PD3, cujas segundas funções são as interrupções externas 0 e 1.

O exemplo a seguir utiliza a interrupção do temporizador 0 para indicar o final da contagem de tempo.

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

ISR(TIMER0_OVF_vect )
{
    TCNT0=158; // reinicia a contagem em 158
    if((PINB&0b00000001)==0)
    {
        PORTB|=0b00000001;
    }
    else
    {
        PORTB&=0b11111110;
    }
}

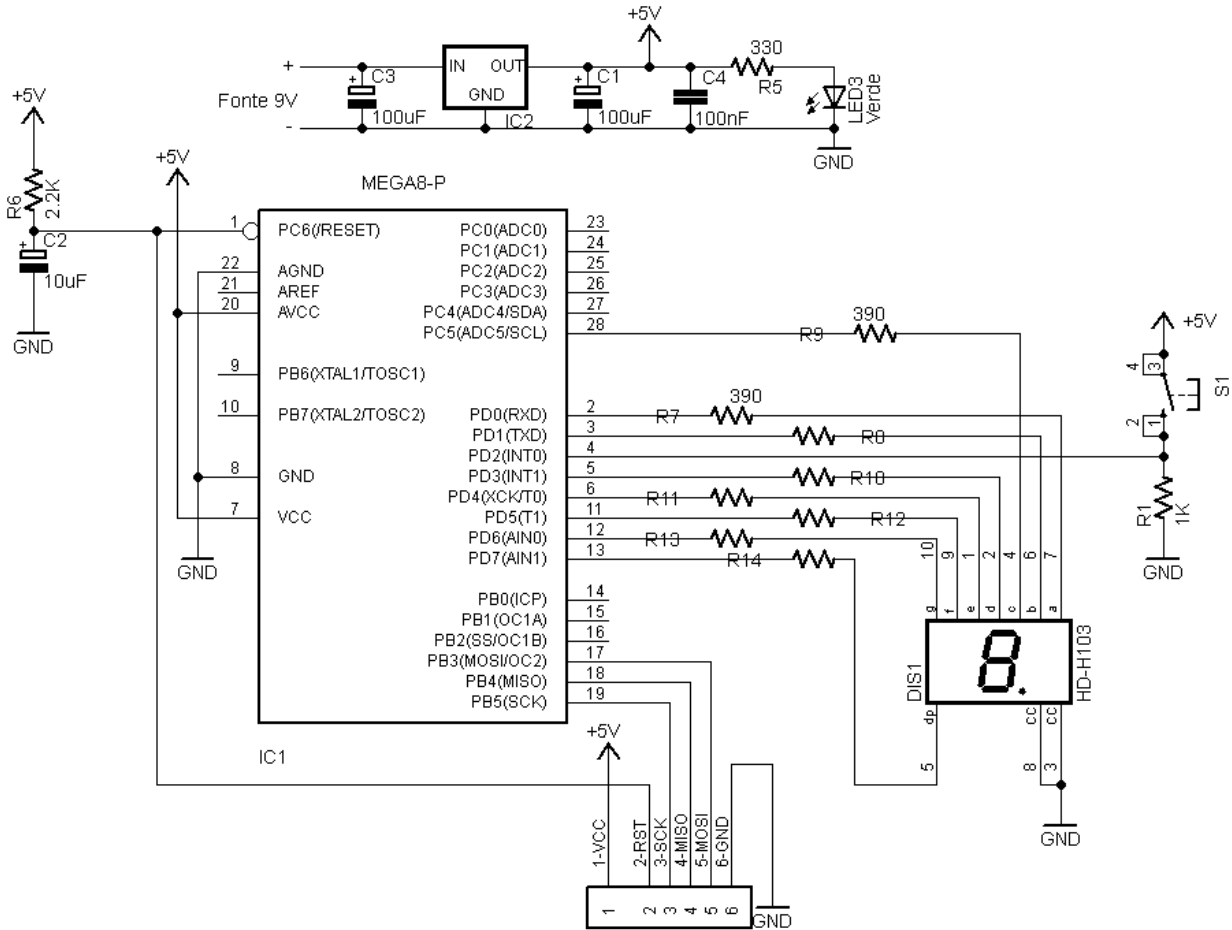
int main()
{
    DDRB=0b00000001; // define PB0 como saída
    TCCR0=0b00000101; // Habilita o timer 0 com prescaler 1024
    TCNT0=158; // inicia a contagem em 158
    TIMSK|=1; // Habilita a interrupção do timer 0
    sei(); // ativa todas as interrupções

    while(1)
    {
    }
}
```

Os detalhes do funcionamento dos temporizadores serão estudados mais a frente.

9.3 Exercícios

1. Determine qual o valor que devemos gravar no registrador “MCUCR”, para que a interrupção externa 0 seja ativada na descida do sinal e a interrupção externa 1 seja ativada na subida do sinal.
2. Para o circuito da figura a seguir faça um programa utilizando interrupções que conta o número de vezes que o botão S1 foi pressionado. O número deve aparecer no display.



AULA 10 - TEMPORIZADORES E CONTADORES

Estudo dos temporizadores e contadores nos microcontroladores AVR

10.1 Objetivo:

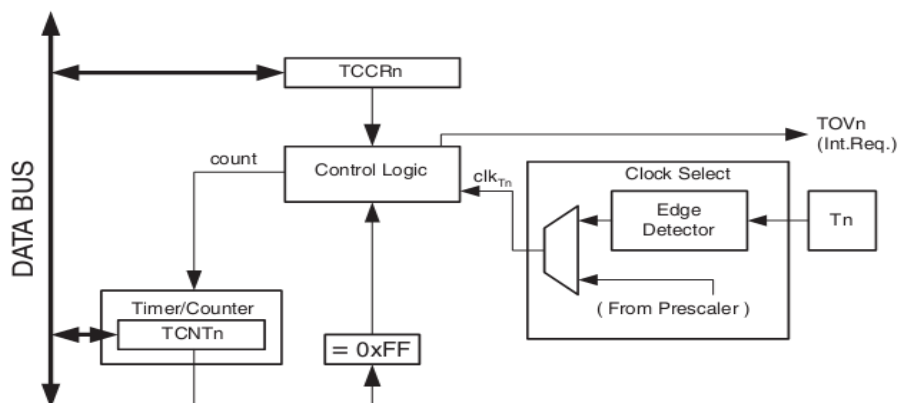
O objetivo desta aula é discutir o funcionamento dos temporizadores e dos contadores nos microcontroladores da família AVR. É comum nos circuitos que utilizam microcontroladores que se tenha a necessidade de realizar contagens tanto de tempo como de eventos. Na família AVR de microcontroladores é possível realizar estas duas funções no hardware, poupando assim processamento e memória de programa.

10.2 Introdução

Trataremos nesta aula os temporizadores e os contadores como um só elemento, uma vez que o circuito do microcontrolador responsável por ambas as tarefas é o mesmo. Isso se deve ao fato de um contador poder contar eventos e assim trabalhar como um contador ou então contar tempo e funcionar como um temporizador. Assim sendo um temporizador nada mais é do que um contador que conta o tempo. O microcontrolador ATmega8 possui três temporizadores / contadores, estudaremos cada um deles nos parágrafos a seguir.

10.3 Temporizador / contador 0

O temporizador / contador 0 é composto por um contador de 8 bits que pode contar eventos ou pulsos de clock, operando assim como temporizador. Ele também pode trabalhar como um gerador de frequência. O temporizador / contador 0 possui também um divisor de frequência, 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 / contador 0



O temporizador / contador 0 pode receber os pulsos para sua contagem tanto do circuito interno como de um sinal externo. A fonte dos pulsos de contagem é definida no registrador TCCR0. As tabelas a seguir descrevem este registrador.

Registrador TCCR0

Bit	Significado
0	CS00 – Bit de seleção de clock 0
1	CS01 – Bit de seleção de clock 1
2	CS02 – Bit de seleção de clock 2
3	–
4	–
5	–
6	–
7	–

Descrição dos bits do registrador TCCR0.

CS02	CS01	CS00	Significado
0	0	0	Sem pulsos de contagem (Contador desligado)
0	0	1	Pulsos direto do clock do microcontrolador
0	1	0	Pulsos do clock do microcontrolador dividido por 8
0	1	1	Pulsos do clock do microcontrolador dividido por 64
1	0	0	Pulsos do clock do microcontrolador dividido por 256
1	0	1	Pulsos do clock do microcontrolador dividido por 1024
1	1	0	Pulsos do pino externo T0, na borda de descida
1	1	1	Pulsos do pino externo T0, na borda de subida

Para que o temporizador / contador 0 entre em funcionamento basta gravar um valor diferente de 0 no registrador TCCR0, e para parar sua contagem basta gravar 0.

O temporizador / contador 0 é capaz de executar uma interrupção toda a vez que sua contagem ultrapassa o valor máximo de seus oito bits que é 255. Quando isso acontece a contagem recomeça de zero.

Para que a interrupção aconteça devemos criar a rotina de interrupção, conforme exemplo da aula anterior, usando o vetor de nome “TIMER0_OVF_vect”, e habilitando a interrupção no registrador TIMSK. A tabela a seguir mostra o significado de cada bit deste registrado.

Registrador TIMSK	
Bit	Significado
0	TOIE0 – Interrupção de estouro do temporizador 0
1	-
2	TOIE1 – Interrupção de estouro do temporizador 1
3	OCIE1B – Interrupção do comparador B do temporizador 1
4	OCIE1A – Interrupção do comparador B do temporizador 1

5	TICIE1 – Interrupção de captura externa do temporizador 1
6	TOIE2 – Interrupção de estouro do temporizador 2
7	OCIE2 – Interrupção do comparador do temporizador 2

É importante observarmos que este registrador é compartilhado por todos os temporizadores / contadores, assim apenas o bit 0 é utilizado pelo temporizador / contador 0. É possível acessar e acompanhar o valor da contagem do temporizador / contador 0 através do registrador TCNT0. Também é neste registrador que gravamos o valor inicial da contagem, uma vez que a contagem sempre finaliza em 255, e a interrupção sempre ocorre no próximo pulso, na passagem de 255 para 0. A seguir temos um exemplo onde o contador / temporizador é configurado para contar 10 pulsos no pino externo T0 e em seguida executar a interrupção e acender um LED no pino PB0.

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

ISR(TIMER0_OVF_vect)
{
  PORTB|=0b00000001;
  TCCR0=0b00000000; // desabilita o temporizador / contador 0
}

int main()
{
  DDRB=0b00000001; // define PB0 como saída
  TCCR0=0b00000111; // Habilita o contador para contagem de pulsos externos no pino T0
  TCNT0=246; // inicia a contagem em 246 para contar 10 pulsos
  TIMSK|=1; // Habilita a interrupção do timer 0
  sei(); // ativa todas as interrupções
  // T0 é a segunda função do pino PD4

  while(1)
  {
  }
}
```

O exemplo inicializa o registrador TCNT0 com 246 para contar 10 pulsos, ou seja:

$$246 = 256 - 10$$

ou seja:

$$TCNT0 = 256 - N_{Pulsos}$$

O exemplo a seguir configura o temporizador / contador 0 como temporizador, para contar um intervalo de tempo de 0,1s.

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

ISR(TIMER0_OVF_vect )
{
  TCNT0=158; // reinicia a contagem em 158
  if((PINB&0b00000001)==0)
  {
    PORTB|=0b00000001;
  }
  else
  {
    PORTB&=0b11111110;
  }
}

int main()
{
  DDRB=0b00000001; // define PB0 como saída
  TCCR0=0b00000101; // Habilita o timer 0 com prescaler 1024
  TCNT0=158; // inicia a contagem em 158
  TIMSK|=1; // Habilita a interrupção do timer 0
  sei(); // ativa todas as interrupções

  while(1)
  {
  }
}
```

O exemplo inicializa o registrador TCNT0 com 158 para temporizar um intervalo de 0,1s. A formula para definir o valor inicial de TCNT0 é:

$$TCNT0 = 256 - \frac{T_{segundos} \times clock}{prescaler}$$

Onde:

T = tempo em segundos desejado

clock = frequencia de operação do microcontroladores

prescaler = valor do divisor de clock

Como o registrador TCNT0 possui apenas 8 bits é importante lembrar que o valor máximo que ele aceita é 255. assim o tempo que ele pode contar é limitado.

10.4 Exercícios

1. Faça um programa utilizando o temporizador / contador 0 que conta 15 pulsos na entrada T0 e quando termina acende um LED na porta PC0.
2. Determine o valor do registrador TCNT0 para que um microcontrolador operando em 1MHz utilize o temporizador / contador 0 para medir um intervalo de tempo de 75ms.
3. Faça um programa utilizando o temporizador / contador 0 que pisca um LED na porta PC0 em 6Hz.
4. Faça um programa utilizando o temporizador / contador 0 que envia uma onda quadrada com frequência igual a 200Hz durante 1 segundo no pino C1, toda vez que o microcontrolador é ligado.

AULA 11 - TEMPORIZADORES AVANÇADOS

Estudo dos temporizadores e contadores com funções avançadas

11.1 Objetivo:

O objetivo desta aula é discutir as funções avançadas dos temporizadores e dos contadores nos microcontroladores da família AVR.

11.2 Introdução

Os microcontroladores da família AVR possuem temporizadores com características especiais, capazes de gerar por exemplo sinais PWM, medir a duração de pulsos, etc. Na aula anterior foi estudado o temporizador / contador 0, agora estudaremos o temporizador / contador 1.

11.3 Temporizador / contador 1

O temporizador / contador 1 é composto por um contador de 16 bits que pode contar eventos ou pulsos de clock, operando assim como temporizador. Ele também pode trabalhar como um gerador de frequência. O temporizador / contador 1 possui também 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 seguir temos um resumo de suas características.

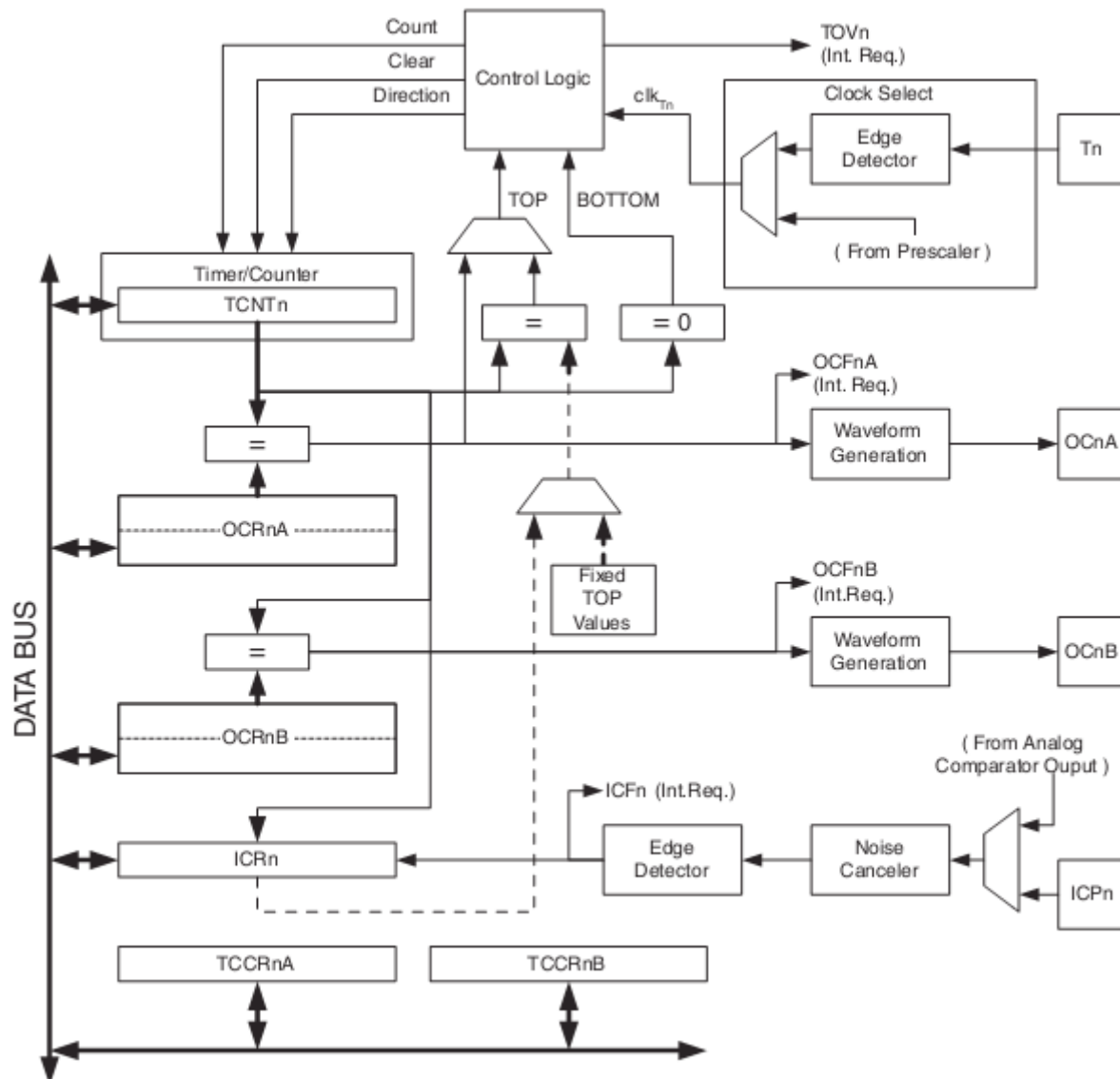
- O contador utilizado pelo temporizador / contador 1 é de 16 bits.
- Possui duas unidades de comparação independentes
- Possui uma unidade de captura externa
- Filtro de ruído na entrada
- Recarga automática
- Função PWM
- Quatro Fontes de interrupção independentes

O temporizador / contador 1 possui um grupo de registradores que armazenam as informações durante sua operação. Os principais são:

- TCNT1 Registrador de contagem
- OCR1A Registrador de comparação A
- OCR1B Registrador de comparação B
- ICR1 Registrador de captura

Todos estes registradores são de 16 bits. O controle do temporizador / contador 1 é feito através dos registradores TCCR1A e TCCR1B, que são registradores de 8 bits.

A figura a seguir mostra um diagrama de funcionamento deste temporizador / contador.



O temporizador / contador 1 possui quatro fontes de interrupção que podem ser controladas independentemente no registrador TIMSK.

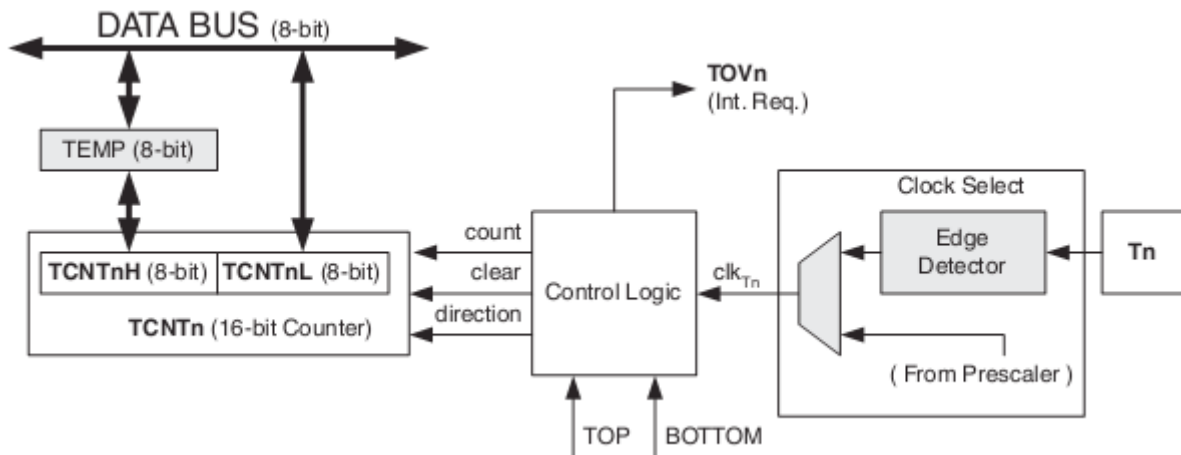
Os pulsos de contagem do temporizador / contador 1 podem vir do clock interno através de um prescaler, ou do pino T1. A contagem inicia quando a fonte de clock é selecionada.

Os registradores de comparação A e B (OCR1A e B) são comparados com o valor de contagem a todo momento e podem ser utilizados para gerar interrupções ou sinais PWM na saída. Estes sinais são fisicamente conectados aos pinos OC1A e OC1B.

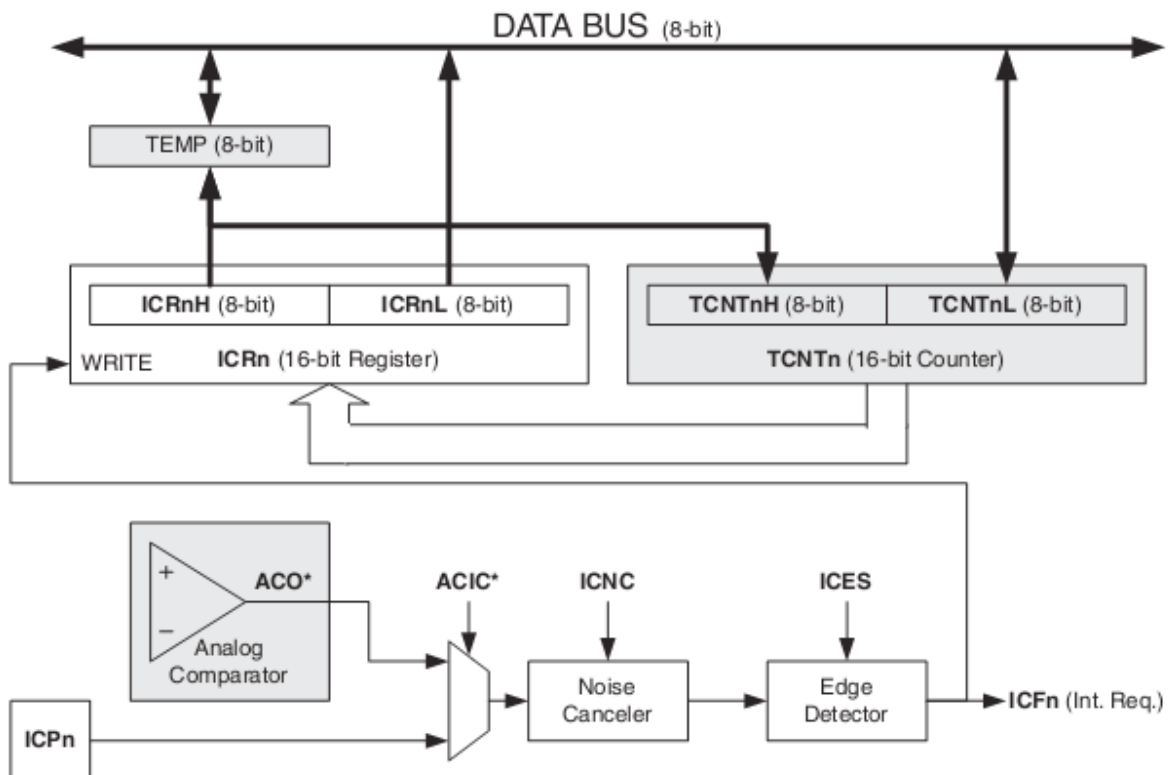
O registrador de captura serve para armazenar o valor do contador quando recebe um sinal externo, vindo do pino ICP1 ou do comparador analógico. A unidade de captura possui um redutor de ruído, que diminui a chance de falsos disparos.

Diferente do contador / temporizador 0 este contador / temporizador não conta sempre até o topo de sua contagem. Como os registradores são de 16 bits o normal seria contar até 0xFFFF (65535), porém o final da contagem depende do modo de operação, como veremos mais a frente.

O sinal de contagem para este contador pode vir de fontes internas ou externas, como pode ser visto na figura a seguir.

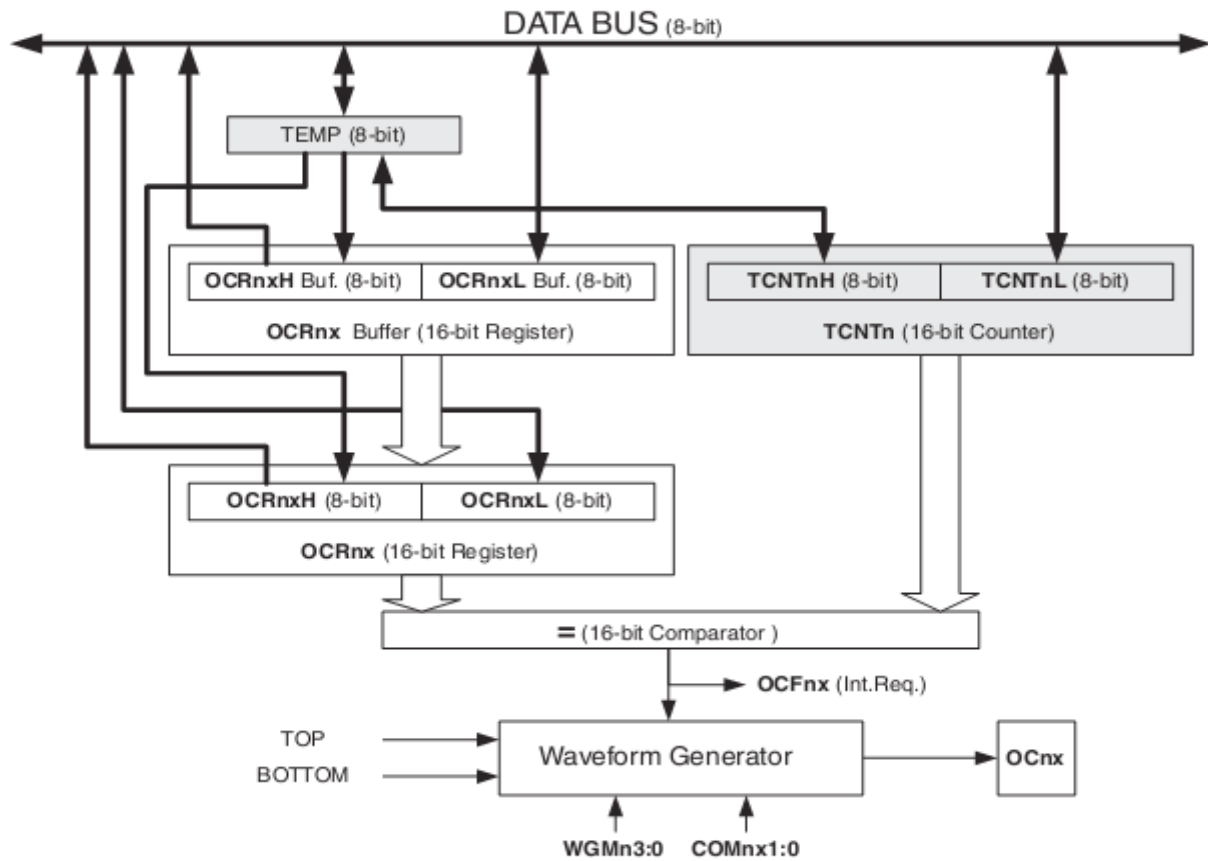


A unidade de captura por sua vez é usada para fazer uma cópia do valor do contador quando um evento externo ocorre. Este mecanismo serve por exemplo para medir a largura de um pulso externo. A figura a seguir mostra o funcionamento do mecanismo de captura.

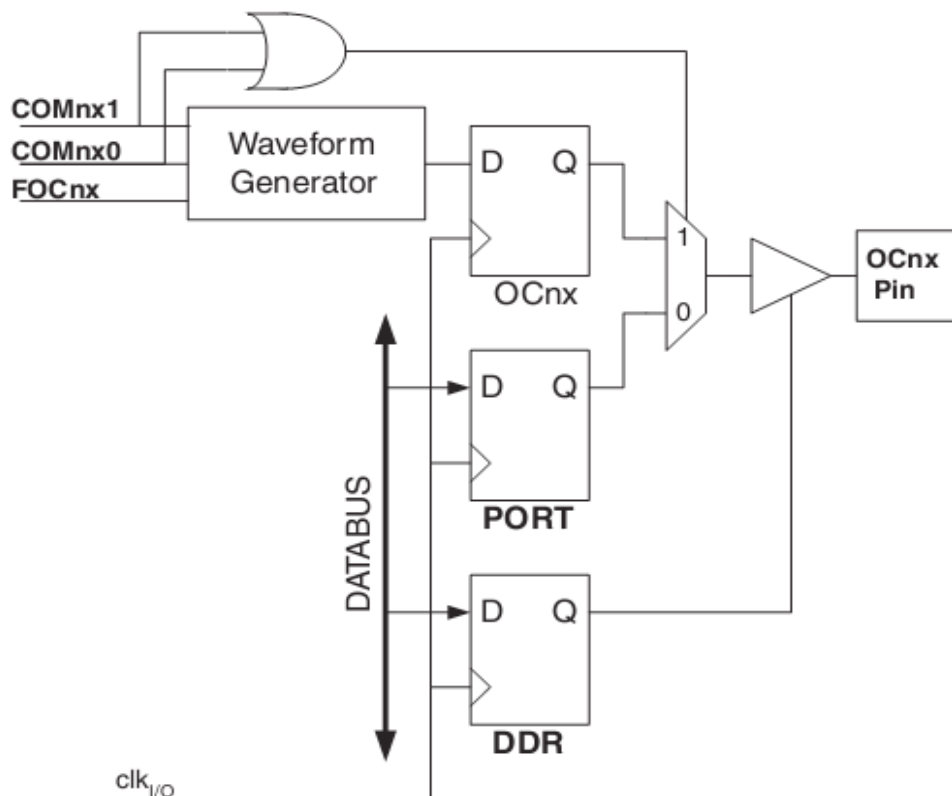


Os registradores de comparação comparam constantemente o valor do temporizador / contador com o valor dos registradores OCR1A e OCR1B, e conforme a configuração é possível gerar uma interrupção ou alterar um pino de saída quando estes valores forem iguais.

A figura a seguir mostra o funcionamento destas unidades de comparação.



Estas unidades de comparação funcionam em uma grande variedade de modos, e na maioria deles a unidade de comparação altera o estado de um pino de saída para gerar um determinado tipo de sinal. A figura a seguir apresenta o circuito de saída das unidades de comparação.



11.4 Registradores de controle do temporizador / contador 1

Todo o funcionamento do temporizador / contador 1 é controlado através de alguns registradores. A seguir serão apresentados estes registradores.

Registrador TCCR1A	
Bit	Significado
0	WGM10 Modo de operação
1	WGM11 Modo de operação
2	FOC1B Força comparação no canal B
3	FOC1A Força comparação no canal A
4	COM1B0 Modo de comparação B
5	COM1B1 Modo de comparação B
6	COM1A0 Modo de comparação A
7	COM1A1 Modo de comparação A

Os bits de 0 a 3 serão analisados mais a frente, os bits de 4 a 7 tem sua função determinada pelo modo de operação do temporizador. As próximas quatro tabelas mostram as funções destes bits em cada um dos modos de operação.

Modo de comparação não PWM.

COM1A1/ COM1B1	COM1A0/ COM1B0	Significado
0	0	Modo normal de operação dos pinos, OC1A e OC1B (desconectados)
0	1	Inverte os sinais dos pinos OC1A e OC1B na comparação
1	0	Envia zero aos pinos OC1A e OC1B na comparação
1	1	Envia um aos pinos OC1A e OC1B na comparação

Modo de comparação PWM rápido.

COM1A1/ COM1B1	COM1A0/ COM1B0	Significado
0	0	Modo normal de operação dos pinos, OC1A e OC1B (desconectados)
0	1	WGM13:0 = 15: inverte OC1A na comparação, OC1B fica desconectado (opera normalmente como I/O). Para qualquer outra combinação de WGM1 os sinais OC1A/OC1B são desconectados (operam normalmente como I/O).
1	0	Zera OC1A/OC1B na comparação, seta OC1A/OC1B quando a contagem chega a zero.
1	1	Seta OC1A/OC1B na comparação, zera OC1A/OC1B quando a contagem chega a zero.

Nos modos de comparação e saída, PWM de fase correta e PWM de frequência correta.

COM1A1/ COM1B1	COM1A0/ COM1B0	Significado
0	0	Modo normal de operação dos pinos, OC1A e OC1B (desconectados)
0	1	WGM13:0 = 9 ou 14: Alterna OC1A na comparação, OC1B desconectado (operando como I/O). Para todas as outras combinações de WGM1, OC1A/OC1B desconectados (operando como I/O).
1	0	Zera OC1A/OC1B na comparação quando contando de forma crescente e seta OC1A/OC1B na comparação quando contando de forma decrescente.
1	1	Seta OC1A/OC1B na comparação quando contando de forma crescente e zera OC1A/OC1B na comparação quando contando de forma decrescente.

O segundo registrador de controle do temporizador / contador 1 é o registrador TCCR1B, a tabela a seguir mostra a função de cada um dos seus bits.

Registrador TCCR1B	
Bit	Significado
0	CS10 Seleção da entrada de clock
1	CS11 Seleção da entrada de clock
2	CS12 Seleção da entrada de clock
3	WGM12 Modo de operação
4	WGM13 Modo de operação
5	-
6	ICES1 seletor de borda da entrada de captura
7	ICNC1 Redutor de ruídos da entrada de captura

- ICNC1 Redutor de ruídos da entrada de captura

Setando este bit é ativado o filtro de cancelamento de ruído. Quando este filtro é ativado o sinal de entrada vindo do pino ICP1 deve permanecer estável por quatro amostragens para que seja considerada a alteração.

- ICES1 seletor de borda da entrada de captura

Este bit determina qual a borda do sinal de captura vindo do pino ICP1 deve ser usado para disparar o evento de captura. Se este bit estiver em zero a captura é executada na borda de descida do sinal. Se este bit estiver em um a captura é executada na borda de subida do sinal. Quando acontece um evento de captura o valor do contador é copiado para o registrador ICR1 e o bit ICF1 é ativado. Se a interrupção estiver ativa, ela também é executada.

A tabela a seguir apresenta a função dos bits dos registradores TCCRA e TCCR1B, responsáveis pela definição do modo de operação.

Modo	WGM13	WGM12	WGM11	WGM10	Modo de operação	Topo	Atualização de OCR1x	Liga o bit TOV1
0	0	0	0	0	Normal	0xFFFF	Imediato	Máximo
1	0	0	0	1	PWM fase correta 8 bits	0x00FF	Topo	Base
2	0	0	1	0	PWM fase correta 9 bits	0x01FF	Topo	Base
3	0	0	1	1	PWM fase correta 10 bits	0x03FF	Topo	Base
4	0	1	0	0	CTC	OCR1A	Imediato	Máximo
5	0	1	0	1	PWM rápido 8 bits	0x00FF	Base	Topo
6	0	1	1	0	PWM rápido 9 bits	0x01FF	Base	Topo
7	0	1	1	1	PWM rápido 10 bits	0x03FF	Base	Topo
8	1	0	0	0	PWM fase e frequência correta	ICR1	Base	Base
9	1	0	0	1	PWM fase e frequência correta	OCR1A	Base	Base
10	1	0	1	0	PWM fase correta	ICR1	Topo	Base
11	1	0	1	1	PWM fase correta	OCR1A	Topo	Base
12	1	1	0	0	CTC	ICR1	Imediato	Máximo
13	1	1	0	1	(Reservado)	-	-	-
14	1	1	1	0	PWM rápido	ICR1	Base	Topo
15	1	1	1	1	PWM rápido	OCR1A	Base	Topo

Os bits 0, 1 e 2 do registrador TCCR1B são responsáveis por definir a fonte do sinal de entrada do contador conforme a tabela a seguir.

CS12	CS11	CS10	Significado
0	0	0	Sem pulsos de contagem (Contador desligado)
0	0	1	Pulsos direto do clock do microcontrolador
0	1	0	Pulsos do clock do microcontrolador dividido por 8
0	1	1	Pulsos do clock do microcontrolador dividido por 64
1	0	0	Pulsos do clock do microcontrolador dividido por 256
1	0	1	Pulsos do clock do microcontrolador dividido por 1024
1	1	0	Pulsos do pino externo T1, na borda de descida
1	1	1	Pulsos do pino externo T1, na borda de subida

Assim como os outros periféricos deste microcontrolador, o temporizador / contador 1 utiliza interrupções para chamar sub-rotinas de software que ajudam em seu funcionamento.

A tabela a seguir descreve o registrador TIMSK, detalhando o funcionamento dos bits utilizados no temporizador / contador 1.

Registrador TIMSK	
Bit	Significado
0	TOIE0 – Interrupção de estouro do temporizador 0
1	-
2	TOIE1 – Interrupção de estouro do temporizador 1
3	OCIE1B – Interrupção do comparador B do temporizador 1
4	OCIE1A – Interrupção do comparador A do temporizador 1
5	TICIE1 – Interrupção de captura externa do temporizador 1
6	TOIE2 – Interrupção de estouro do temporizador 2
7	OCIE2 – Interrupção do comparador do temporizador 2

- TICIE1: Habilita a interrupção de captura do temporizador / contador 1.
- OCIE1A: Habilita a interrupção do comparador A do temporizador / contador 1.
- OCIE1B: Habilita a interrupção do comparador B do temporizador / contador 1.
- TOIE1: Habilita a interrupção de estouro do temporizador / contador 1

O temporizador / contador 1 é bastante complexo e este material é apenas para demonstrar as capacidades deste componente. É necessário que se estude no manual do componente cada um dos modos de operação do temporizador / contador 1.

A seguir temos um exemplo onde o temporizador / contador 1 é utilizado no modo PWM de fase correta.

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

int main(void)
{
    DDRB=0b00000010;// o pino OC1A (PB1) é definido como saída

    OCR1A = 511;      // Ajusta o PWM para 50% para o modo de 10bit

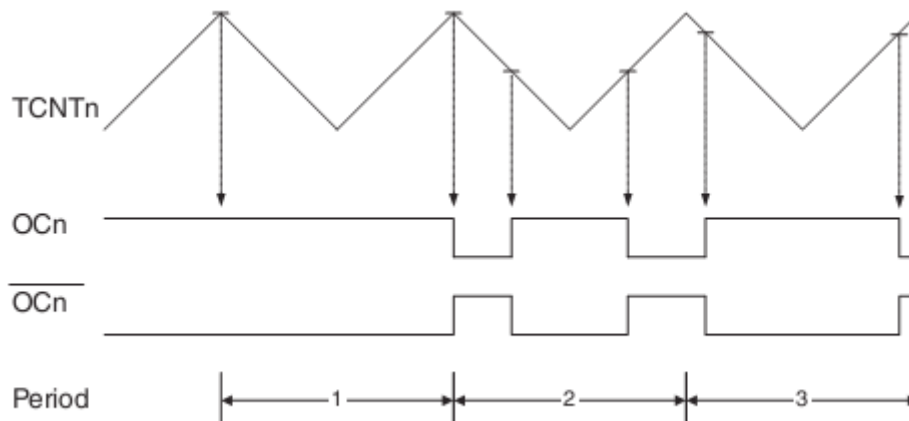
    TCCR1A |= 0b10000000; // Seta e zera OC1A na comparação

    TCCR1A |= 0b00000011; //modo 3: PWM de fase correta com 10 bits

    TCCR1B |= 0b00000010; // set prescaler to 8 and starts PWM

    while (1)
    {
    }
}
```

A figura a seguir mostra o funcionamento do temporizador / contador 1 operando no modo de PWM de fase correta, como no exemplo.



Na figura as setas mostram o ponto onde o contador ultrapassa o valor do comparador A, forçando uma alteração no pino de saída OC1A.

11.5 Exercícios

1. Utilizando o manual do atmega8 determine como funciona o modo de operação PWM Rápido (Fast PWM) do temporizador / contador 1.
2. Utilizando as tabelas apresentadas neste material e com a ajuda do manual do atmega8 altere o exemplo desta aula, de modo que ele possui uma rotina de interrupção, que é chamada no final de cada ciclo do PWM. A rotina de interrupção deve incrementar de 1 em 1 o valor do comparador A.
3. Faça um programa utilizando o temporizador / contador 1 que chama uma interrupção a cada 0,1 segundos.

AULA 12 - CONVERSOR ANALÓGICO DIGITAL

Estudo do conversor analógico digital interno do microcontrolador AVR

12.1 Objetivo:

O objetivo desta aula é discutir o funcionamento e a utilização do conversor analógico digital que está instalado no interior dos microcontroladores da família AVR da atmel.

12.2 Introdução

Trataremos nesta aula do conversor analógico digital, bem como da sua utilização em nossos projetos. Trata-se de um assunto importante, pois grande parte dos sinais existentes na automação e no controle de processos, bem como, em sistemas embarcados são sinais analógicos e assim devem ser convertidos para sinais digitais antes que possam ser processados pelos microcontroladores.

Para facilitar o processo, abreviaremos o conversor analógico digital como conversor A/D.

12.3 O hardware do conversor analógico digital

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

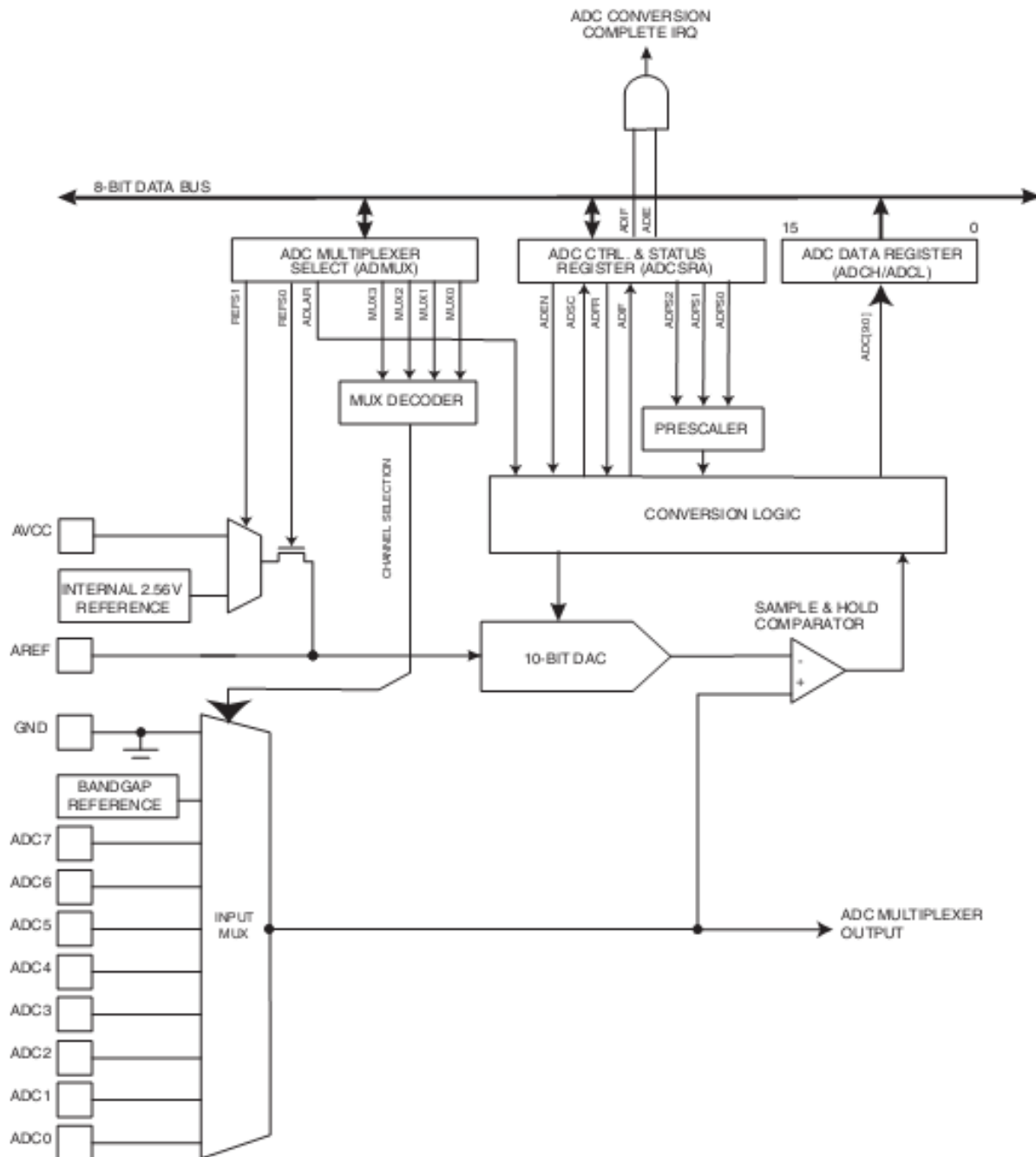
- 10 bits de resolução
- Linearidade de 0,5 LSB
- Precisão absoluta de ± 2 LSB
- Tempo de conversão de 13 - 260 μ s
- Até 15K amostras por segundo em resolução máxima
- 6 canais multiplexados
- Opção de trabalhar em 8 bits
- Tensão de entrada de 0 - VCC
- Referência interna opcional de 2,56V
- Interrupção de final de conversão

O conversor A/D da família AVR tem sua alimentação separada do resto do microcontrolador, assim para utilizarmos o conversor A/D devemos conectar o pino AVCC a alimentação positiva e o pino AGND a alimentação negativa. A alimentação do conversor A/D é separada para que possamos fornecer uma alimentação limpa de ruídos, aumentando assim a qualidade das medidas realizadas.

Este conversor A/D pode utilizar três diferentes referências de tensão, que são:

- Referência externa conectada no pino AREF
- Referência conectada a alimentação do conversor A/D (Pino AVCC)
- Referência interna de 2,56V

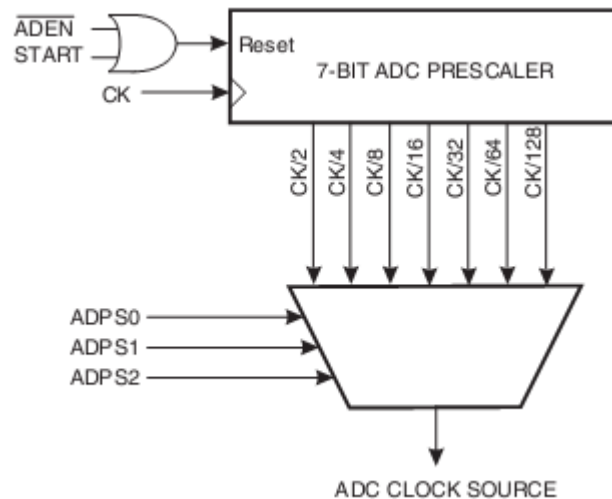
A figura a seguir apresenta um diagrama de blocos do conversor analógico digital da família AVR.



É possível escolher qual a referência para a conversão e qual o canal de entrada através dos registradores de controle do A/D. Estes registradores serão estudados mais a frente.

12.4 Prescaler do conversor A/D

Como qualquer circuito o conversor A/D necessita de um sinal de clock para trabalhar, este sinal é derivado do clock do microcontrolador através de um circuito de prescaler. A figura a seguir apresenta um diagrama de blocos deste circuito.



É importante salientar que em resolução máxima (10 bits), o conversor A/D não pode operar em uma frequência superior a 200KHz.

12.5 Resultados das conversões

O resultado das conversões depende de dois fatores, da tensão na entrada e da referência utilizada no conversor A/D. A fórmula a seguir mostra como calcular o valor de saída do conversor A/D.

$$ADC = \frac{V_{entrada} \times 1024}{V_{ref}}$$

No programa o resultado da conversão pode ser acessado através do registrador ADCW.

O restante dos ajustes do conversor A/D é feito através dos registradores ADMUX e ADCSRA. Que serão vistos a seguir.

12.6 Registradores de controle do conversor A/D

As tabelas a seguir apresentam as funções dos bits do registrador ADCMUX

Registrador ADCMUX	
Bit	Significado
0	MUX0 – bit 0 do multiplexador
1	MUX1 – bit 1 do multiplexador
2	MUX2 – bit 2 do multiplexador

3	MUX3 – bit 3 do multiplexador
4	–
5	ADLAR – Ajuste da saída do A/D
6	REFS0 – bit 0 da seleção de referência
7	REFS1 – bit 1 da seleção de referência

Os primeiros 4 bits são responsáveis por selecionar a entrada analógica, a tabela a seguir detalha seu funcionamento.

MUX3	MUX2	MUX1	MUX0	Significado
0	0	0	0	ADC0
0	0	0	1	ADC1
0	0	1	0	ADC2
0	0	1	1	ADC3
0	1	0	0	ADC4
0	1	0	1	ADC5
0	1	1	0	ADC6
0	1	1	1	ADC7
1	0	0	0	-
1	0	0	1	-
1	0	1	0	-
1	0	1	1	-
1	1	0	0	-
1	1	0	1	-
1	1	1	0	1,3V interno
1	1	1	1	0V interno

O bit 5 diz respeito a forma que o resultado é armazenado, e se for necessário o manual deve ser consultado. Já os bits 6 e 7 são responsáveis pela escolha da entrada referencia do conversor A/D. A tabela a seguir mostra o significado de cada bit.

REFS1	REFS0	Significado
0	0	Referência externa no pino AREF
0	1	Referencia externa no pino AVCC (Utilizar filtro em AREF)
1	0	-
1	1	Referência interna de 2,56 (Utilizar filtro em AREF)

Outro registrador envolvido no controle do conversor A/D é o registrador ADCSRA. A tabela a seguir mostra o significado de cada um dos seus bits.

Registrador ADCSRA	
Bit	Significado
0	ADPS0 – bit 0 do prescaler do A/D
1	ADPS1 – bit 1 do prescaler do A/D
2	ADPS2 – bit 2 do prescaler do A/D
3	ADIE – habilita a interrupção do A/D
4	ADCIF – Indicador de interrupção
5	ADFR – Modo de conversão
6	ADSC – Inicia a conversão
7	ADEN – habilita o conversor A/D

Os bits 0, 1 e 2 são responsáveis pela escolha do prescaler, conforma tabela a seguir.

ADPS2	ADPS1	ADPS0	Significado
0	0	0	Divide o clock por 2
0	0	1	Divide o clock por 2
0	1	0	Divide o clock por 4
0	1	1	Divide o clock por 8
1	0	0	Divide o clock por 16
1	0	1	Divide o clock por 32
1	1	0	Divide o clock por 64
1	1	1	Divide o clock por 128

O bit 3 habilita a interrupção de final de conversão do conversor A/D. O bit 4 indica quando a conversão acabou e o valor já é válido. O bit 5 serve para escolhermos se as conversões deve acontecer sucessivamente, ou se devemos iniciar cada conversão manualmente. O bit 6 inicia a conversão. E finalmente o bit 7 habilita o conversor A/D.

12.7 Exemplo de programa utilizando o conversor A/D

A seguir temos um exemplo de programa que utiliza o conversor A/D do microcontrolador.

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

ISR(ADC_vect)
{
if(ADCW>512)
{
PORTD|=0b00000001;
}
```

```
    }  
else  
    {  
        PORTD&=0b11111110;  
    }  
ADCSRA|=0b01000000; // inicia nova conversão  
}  
  
int main()  
{  
    DDRD=0b00000001; // define PD0 como saída  
    ADMUX=0b01000000; // escolhe a entrada analógica 0, com a referência no pino AVCC  
    ADCSRA=0b11001111; // habilita o A/D e inicia a conversão com prescaler de 128,  
    habilitando sua interrupção.  
    sei(); // ativa todas as interrupções  
  
    while(1)  
    {  
    }  
}
```

O programa do exemplo habilita o conversor A/D, faz sua leitura e compara o valor lido com 512, se o valor for maior que 512 o LED no pino PD0 é ligado, caso contrário é desligado.

12.8 Exercícios

1. Suponha que em um tanque de 1 metro de altura tenhamos instalado um sensor de nível que envia um sinal de 0 a 5V. 0V para nível de 0 m e 5 V para nível de 1 m. Suponha também que no pino D0 é instalado um relê que aciona a bomba. Esta bomba enche o tanque. Para este sistema faça um programa que controla o nível do tanque entre 70 e 80cm, uma vez que o sensor esta conectado a entrada analógica 2.

AULA 13 - COMUNICAÇÃO SERIAL NA FAMÍLIA AVR

Comunicação serial nos microcontroladores da família AVR

13.1 Objetivo:

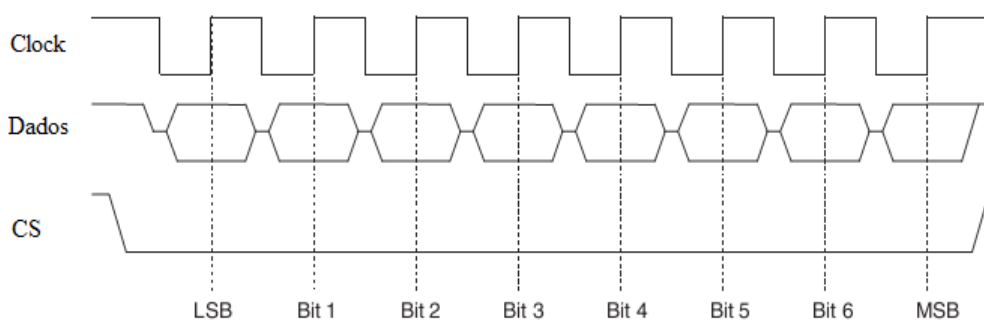
O objetivo desta aula é demonstrar aos alunos o que é e como funciona a comunicação serial, bem como estudar o funcionamento da porta de comunicação serial presente nos microcontroladores da família AVR.

A comunicação serial é um meio eficiente de troca de informações entre sistemas digitais. Suas principais vantagens são a distância e o número reduzido de fios. No controle e na automação de processos é comum o uso da comunicação serial, isso devido a sua flexibilidade e das vantagens já citadas. As redes industriais são em sua maioria seriais, variando apenas em especificação e protocolo.

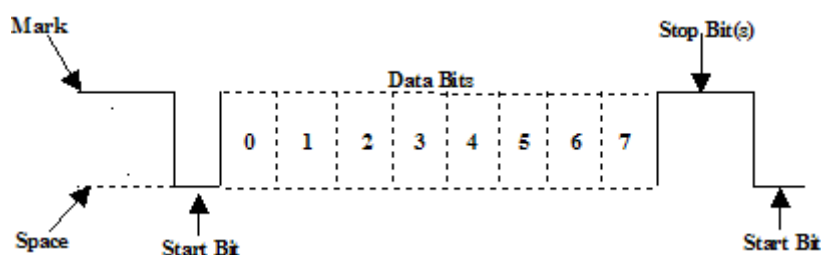
13.2 A comunicação serial

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

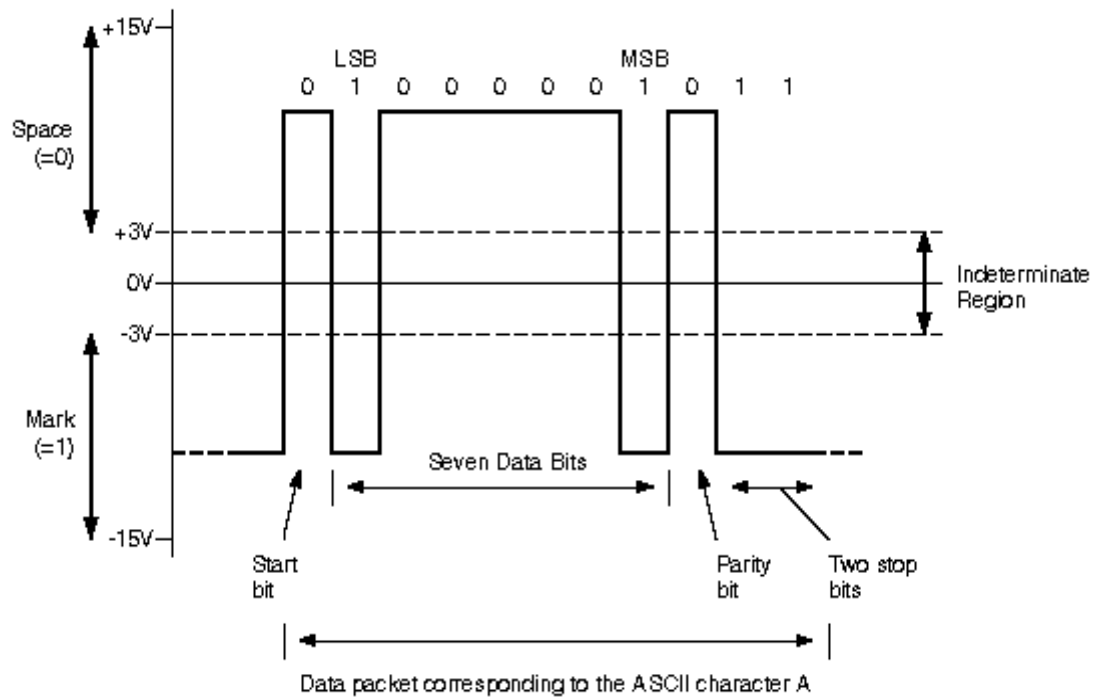
Para comunicações em distâncias maiores são utilizados vários padrões, dentre os quais estudaremos o RS232 e o RS485.

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.



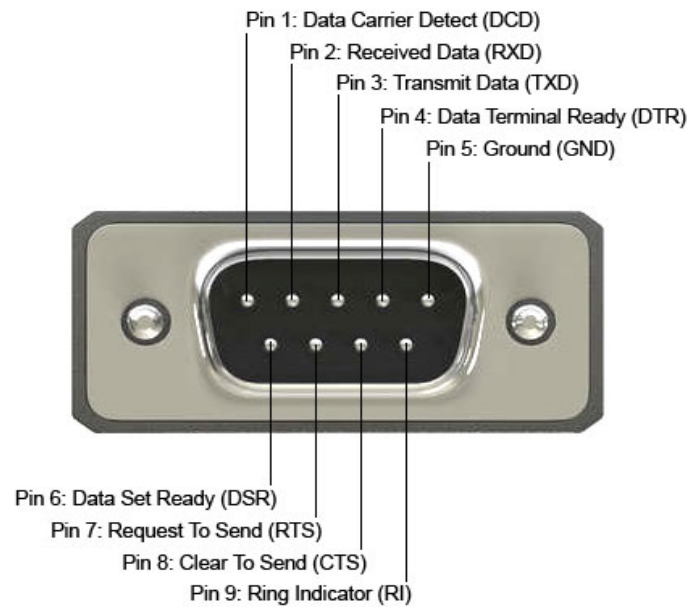
Utilizando níveis de tensão mais elevados é possível alcançar distâncias maiores, na casa de alguns metros. Um fator importante nesta distância é a blindagem do cabo, cabos blindados fornecem maior imunidade a ruído permitindo maior alcance.

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

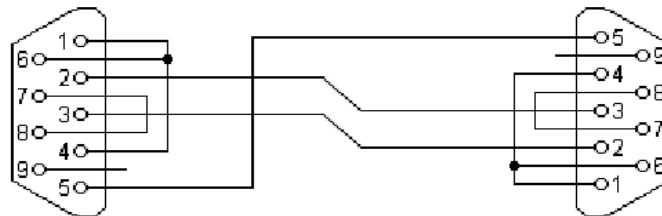
- DTR: (Data Terminal Ready) Usado para avisar que está pronto para comunicar
- RTS: (Request To Send) Usado para avisar que deseja enviar dados agora.
- DSR: (Data Set Ready) Usado para avisar que está pronto para operar.
- CTS: (Clear To Send) Usado para avisar que está pronto para aceitar transmissão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

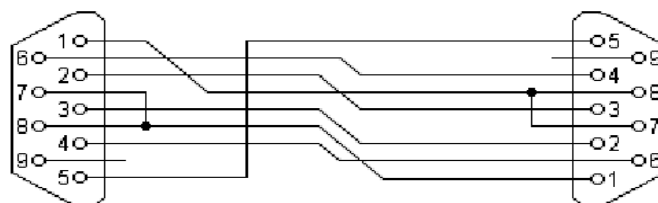
A transmissão dos dados é realizada pela saída TX, e a recepção é realizada pela entrada RX. O conector padrão nos computadores para a porta serial é o DB9 macho, e a distribuição dos pinos é mostrada na figura a seguir.

RS232 Pinout

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.



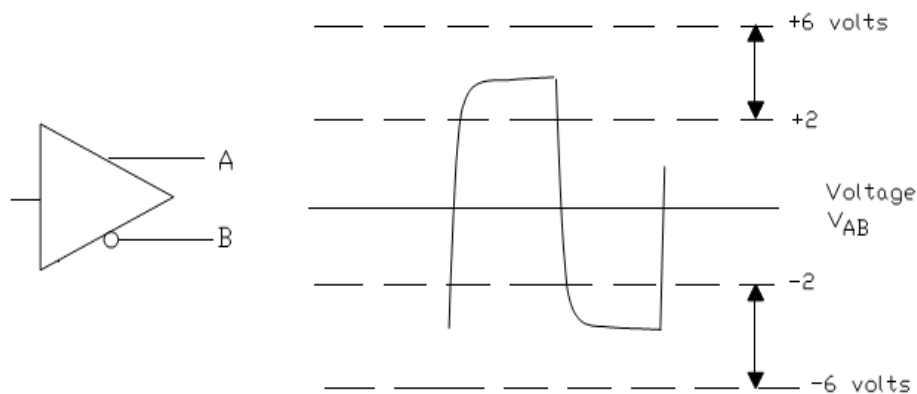
Analisando esta informações é possível observar que o padrão RS232 permite comunicação em ambos os sentidos ao mesmo tempo.

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 e o que define o nível lógico do sinal é a direção da corrente nestes dois fios. Este arranjo é muito imune a ruído, permitindo comunicações com distâncias superiores a 1 Km.

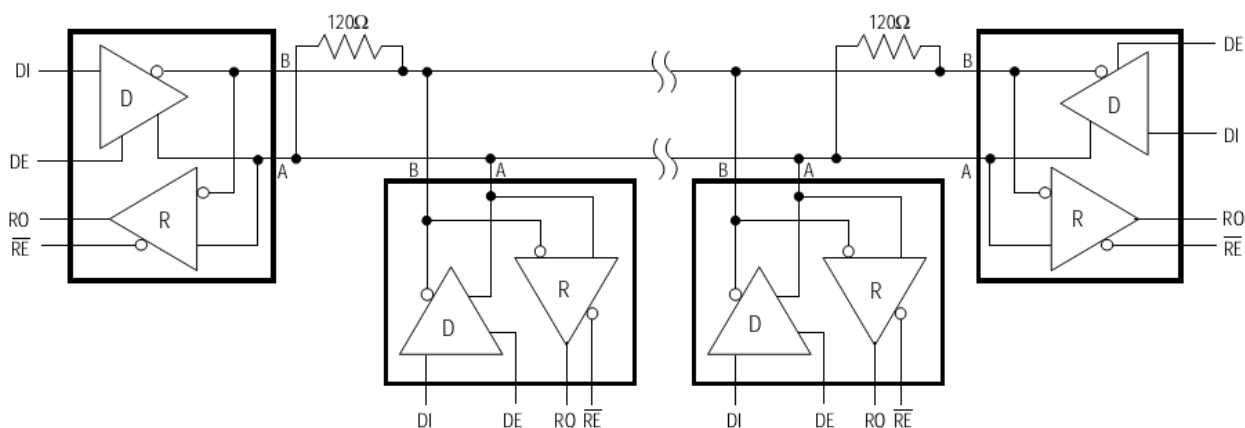
Eletricamente falando o sinal que transmite os dados na comunicação RS485 é um sinal

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.

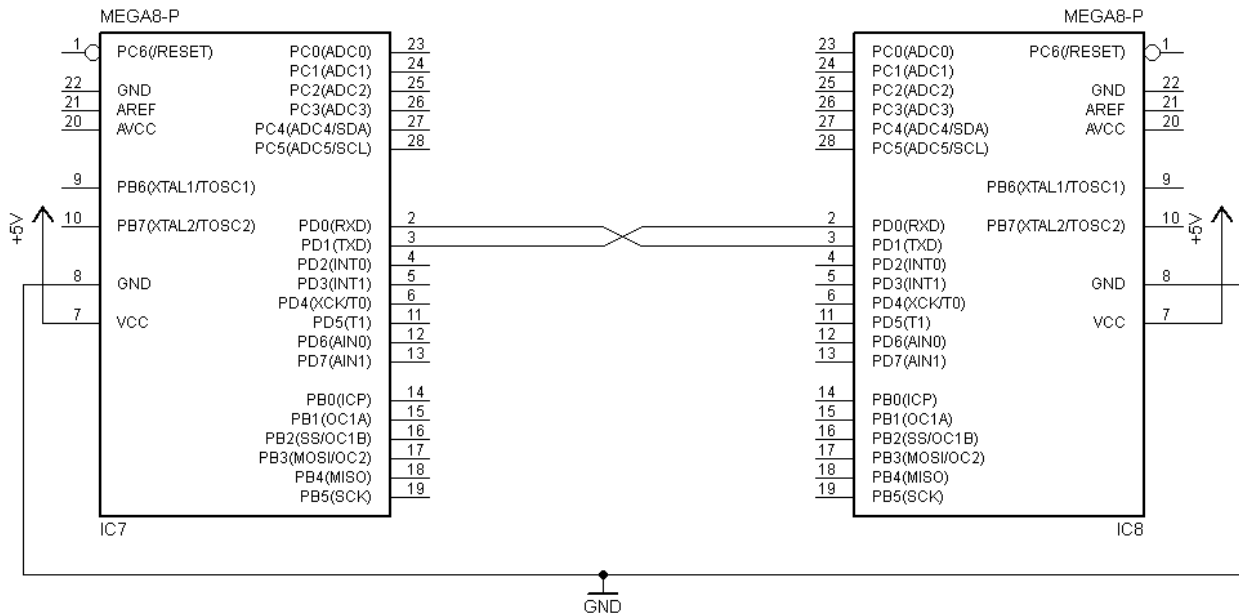


Observando a figura podemos notar a presença de 2 resistores de 120Ω , estes resistores são necessários para casar a impedância dos equipamentos com a rede, tornando a mesma mais imune a ruídos e a erros de comunicação.

13.5 Comunicação serial nos microcontroladores

Os microcontroladores da família AVR trazem 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 devemos adicionar dispositivos externos para adequar os sinais transmitidos e recebidos.

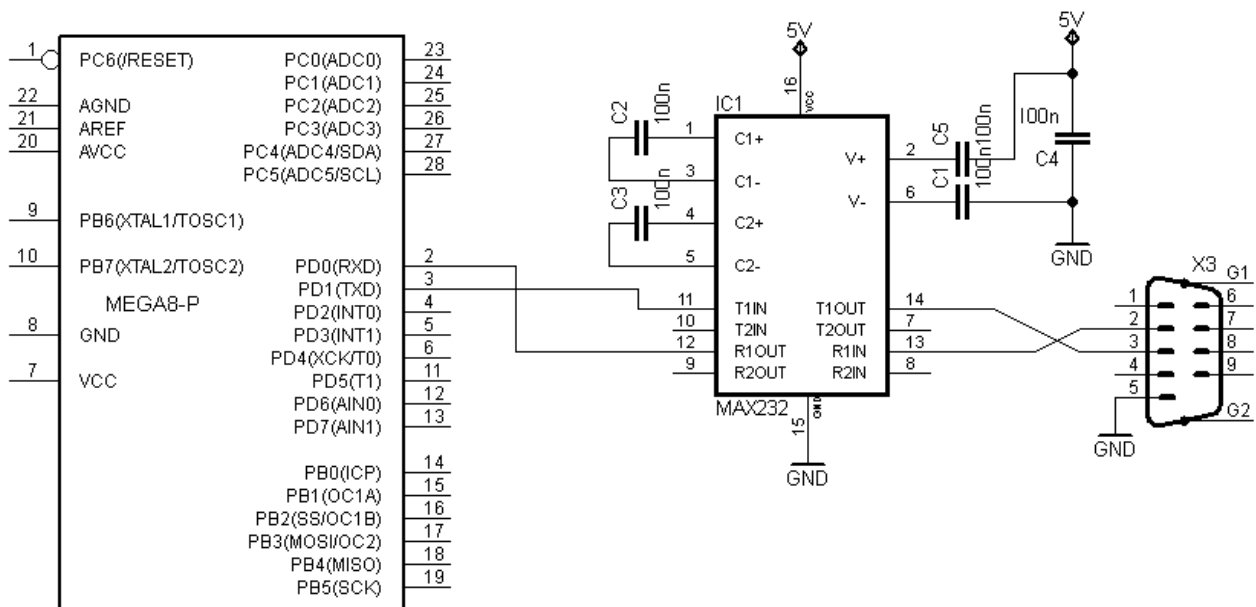
Se desejarmos conectar diretamente dois microcontroladores não é necessário utilizar nenhum padrão complexo, basta apenas conectá-los como mostrado na figura a seguir.



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.

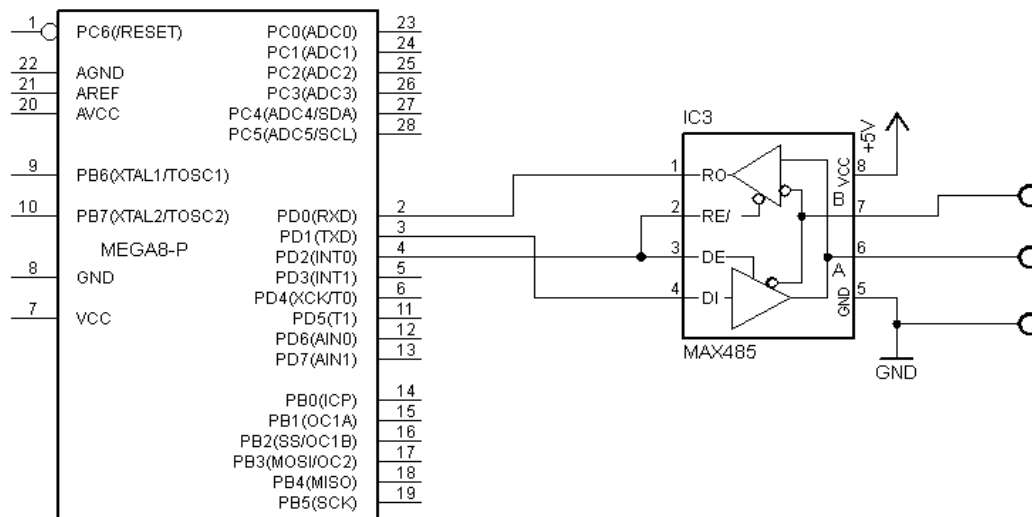
13.5.1 O padrão RS232 nos microcontroladores

Para que um microcontrolador 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.

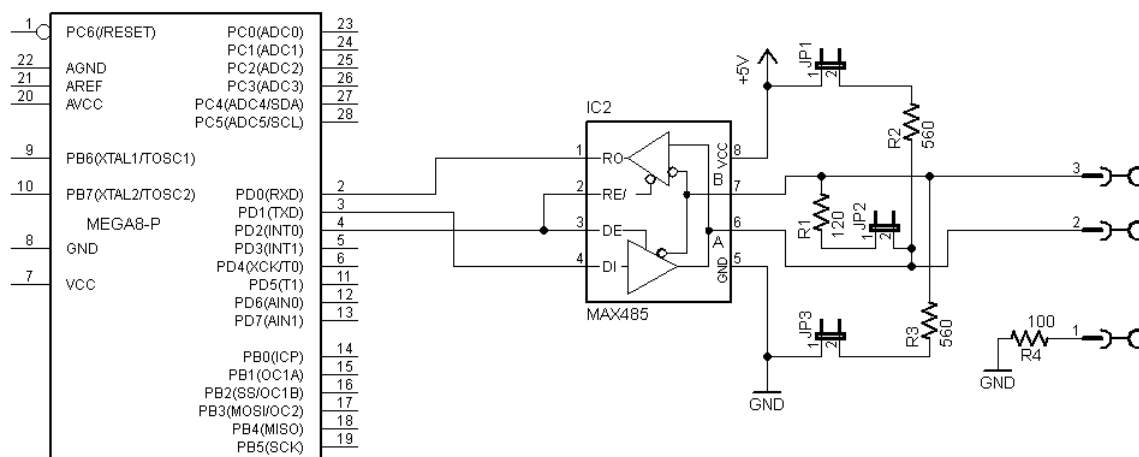


13.5.2 O padrão RS485 nos microcontroladores

Para que um microcontrolador 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 adsorver picos de tensão.

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 A configuração do microcontrolador

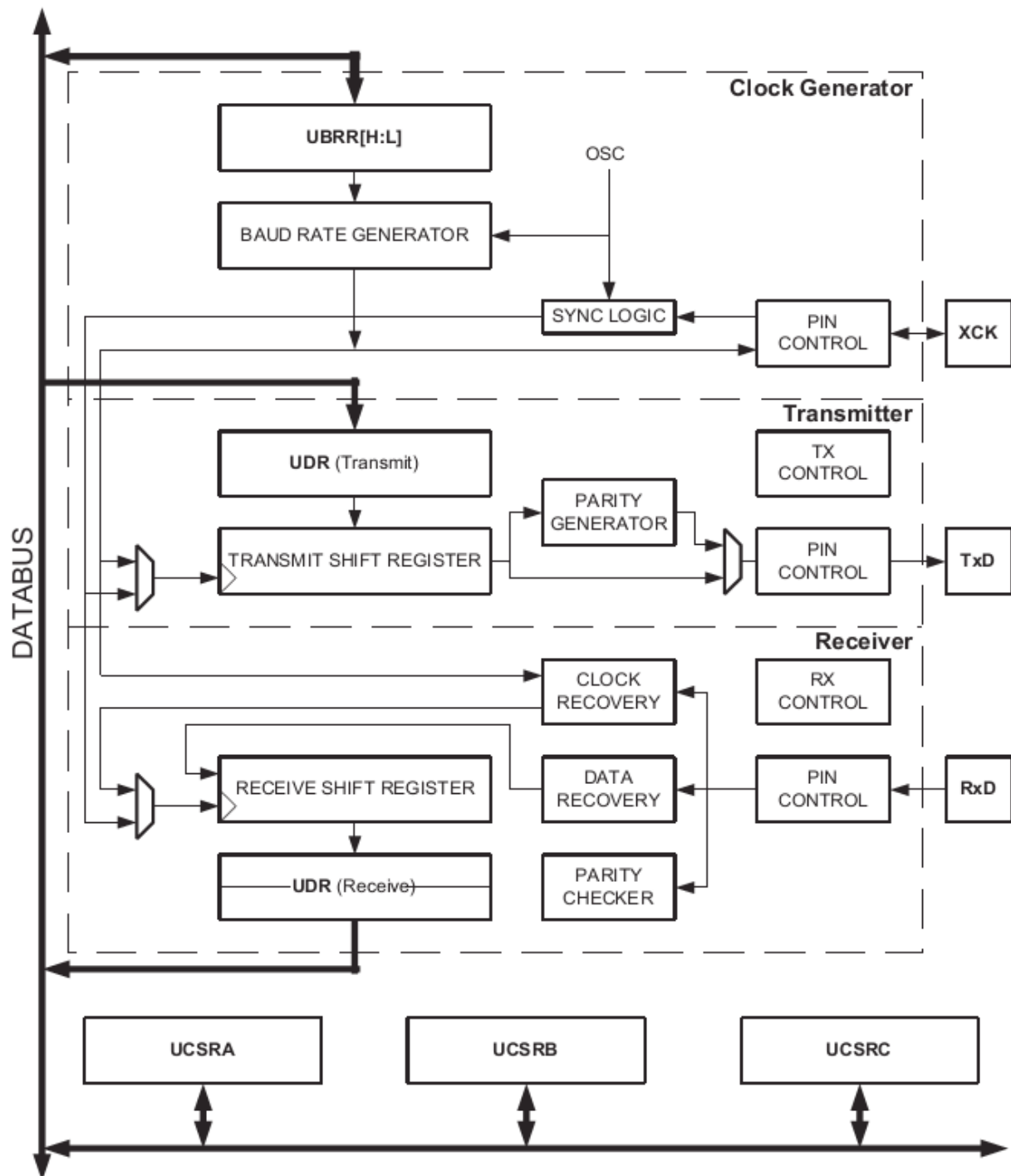
Nos microcontroladores da família AVR a porta de comunicação serial é chamada de USART (Universal Synchronous and Asynchronous serial Receiver and Transmitter). Esta porta é um dispositivo de comunicação serial extremamente versátil. Suas principais características são:

- Transmissão e recepção simultânea de dados (Full Duplex).
- Operação assíncrona ou síncrona.
- Opera como mestre ou escravo em uma rede síncrona.
- Gerador de taxa de transferência de alta resolução.
- Suporta pacotes de dados de 5, 6, 7, 8 ou 9 bits com 1 ou 2 bits de parada.
- Gerador e verificador de paridade em hardware, com paridade par e ímpar.
- Detecta sobrescrita de dados.
- Detecta erro de pacote.
- Diversos filtros anti ruídos.
- Três fontes de interrupção, transmissão completa, registrador de transmissão vazio e recepção completa.
- Modo de comunicação multi-processador.
- Modo de comunicação “Double Speed” para comunicações assíncronas de alta velocidade.

A figura a seguir apresenta um diagrama de blocos da USART presente nos microcontroladores da família. Os retângulos pontilhados separam as três principais partes deste hardware. De cima para baixo temos, o gerador de clock, o transmissor e receptor e os registradores de controle. A lógica de geração de clock consiste de uma lógica de sincronização para a entrada de clock externo usado pela modo escravo síncrono, e do gerador de taxa de transferência. O pino XCK é usado apenas no modo de transferência síncrona. O transmissor consiste de um registrador temporário, um registrador de deslocamento serial, um gerador de paridade e lógica de controle. O registrador temporário de transmissão permite uma transmissão contínua entre pacotes, sem atrasos.

O receptor é a parte mais complexa da USART, isso devido a lógica necessária para recuperar o sinal de clock no modo assíncrono. Além do recuperador de clock a unidade de recepção possui um circuito responsável por checar erros de paridade no pacote, lógica de controle, registrador de deslocamento serial e dois níveis de registradores temporários. O receptor suporta os mesmos formatos de pacotes do que o transmissor e é capaz de detectar erros na transmissão.

A lógica de geração de clock gera a base de clock tanto para o receptor como para o transmissor. A USART suporta quatro modos de operação de clock: assíncrono normal, assíncrono de velocidade dobrada, síncrono mestre e síncrono escravo.



Cada um dos modos de operação da USART possui configurações específicas, não é o objetivo deste material detalhar o funcionamento destes modos.

Neste material iremos nos ater principalmente ao modo de operação assíncrono, uma vez que este é o mais usado para comunicação entre o microcontrolador e o computador pessoal e nas redes industriais.

Nos próximos parágrafos serão detalhados os registradores utilizados na configuração da USART.

O registrador que armazena os dados enviados e recebidos é o UDR é através deste

registrados que o programador interage com o hardware.

O primeiro registrador de controle que estudaremos é o UCSRA. A tabela a seguir detalha seus bits.

Registrador UCSRA	
Bit	Significado
0	MPCM: Modo multiprocessadores.
1	U2X: Dobra a velocidade da USART.
2	PE: Erro de paridade
3	DOR: Sobreposição de dados
4	FE: Erro no pacote
5	UDRE: Registrador temporário de dados vazio.
6	TXC: Transmissão completa.
7	RXC: Recepção completa.

O próximo registrador é o UCSRB.

Registrador UCSRB	
Bit	Significado
0	TXB8: Bit 8 da transmissão.
1	RXB8: Bit 8 da recepção.
2	UCSZ2: Tamanho do pacote, bit 2.
3	TXEN: Habilita a transmissão.
4	RXEN: Habilita a recepção.
5	UDRIE: Habilita a interrupção de registrador temporário vazio.
6	TXCIE: Habilita a interrupção de transmissão.
7	RXCIE: Habilita a interrupção de recepção.

O último registrador de controle que estudaremos é o UCSRC.

Registrador UCSRC	
Bit	Significado
0	UCPOL: Polaridade do clock
1	UCSZ0: tamanho do pacote 0
2	UCSZ1: tamanho do pacote 1
3	USBS: Seleção de stop bit.
4	UPM0: Modo da paridade 0

5	UPM1: Modo da paridade 1
6	UMSEL: Seleção de modo de operação da USART
7	URSEL: Seleciona o acesso ao registrador.

Alguns destes bits são combinados para configurar aspectos da comunicação serial na USART. O bit UMSEL faz a seguinte configuração.

UMSEL	Significado
0	Operação em modo assíncrono
1	Operação em modo síncrono

Os bits UPM1 e UPM0 são responsáveis pela configuração da paridade e seu significado é o seguinte.

UPM1	UPM0	Significado
0	0	Paridade desabilitada
0	1	Reservado
1	0	Paridade habilitada no modo par
1	1	Paridade habilitada no modo ímpar

O bit USBS tem a seguinte função.

USBS	Significado
0	1 stop bits
1	2 stop bits

Os bits UCSZ2, UCSZ1 e UCSZ0 são responsáveis por determinar o número de bits no pacote.

UCSZ2	UCSZ1	UCSZ0	Significado
0	0	0	5 bits
0	0	1	6 bits
0	1	0	7 bits
0	1	1	8 bits
1	0	0	Reservado
1	0	1	Reservado
1	1	0	Reservado
1	1	1	9 bits

Outra tarefa importante quando se trabalha com comunicação serial é o ajuste da velocidade de transmissão. Na família AVR este ajuste é feito no registrador UBRR.

Para entendermos funcionamento da USART veja o exemplo a seguir.

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

#define F_CPU 1000000 // frequencia do cristal
#define USART_BAUDRATE 300 // taxa de transmissão desejada
#define BAUD_PRESCALE ((F_CPU/ (USART_BAUDRATE * (long)16))) //calcula o valor
do prescaler da usart

void USART_send( unsigned char data)
{
while((UCSRA&0b00100000)==0)
    {
    }
UDR = data;
}

ISR(USART_RXC_vect) // interrupção de recepção do pacote
{
char c;
c=UDR;
if(c=='x')
    {
    PORTB=PORTB|0b00000001;
    }
else
    {
    PORTB=PORTB&0b11111110;
    }
}

int main(void)
{
long cont;
DDRB=0b00000001;
UCSRA=0b00000000;
UCSRB=0b10011000;
UCSRC=0b10000110;
UBRRL = BAUD_PRESCALE;
UBRRH = (BAUD_PRESCALE >> 8);
```

```
sei();  
  
while(1)  
{  
    USART_send('A');  
    cont=3000;  
    while(cont>0) cont--;  
}  
}
```

Este material serve apenas como orientação, para compreender mais sobre o funcionamento da porta serial dos microcontroladores da família AVR é importante consultar o datasheet do componente.

13.6 Exercícios

1. Baseado no exemplo anterior faça um programa que recebe pela porta serial conjuntos de 2 caracteres, onde o primeiro é a letra D ou a letra L, e o segundo é um número de 0 a 7. O programa deve interpretar os caracteres recebidos da seguinte forma: a letra D significa desligar e a letra L ligar e o número corresponde ao pino da porta B. Assim se o microcontrolador receber o comando D5 ele deve desligar o pino 5 da porta B ou se ele receber o comando L7 ele deve ligar o pino 7 da porta B. Conjuntos de letras caracteres que não sejam comandos válidos devem ser ignorados.
2. Faça um programa baseado no exemplo anterior e inclua nele a seguinte declaração de variável `char txt[]="Microcontroladores";` A variável `txt` irá assim armazenar a palavra Microcontroladores. O programa deve enviar a palavra contida na variável repetidamente através da porta serial a 9600 bits por segundo. Utilize um comando `for` para selecionar as letras da palavra.

AULA 14 - LADDER NOS MICROCONTROLADORES

Programação de microcontroladores AVR em linguagem Ladder

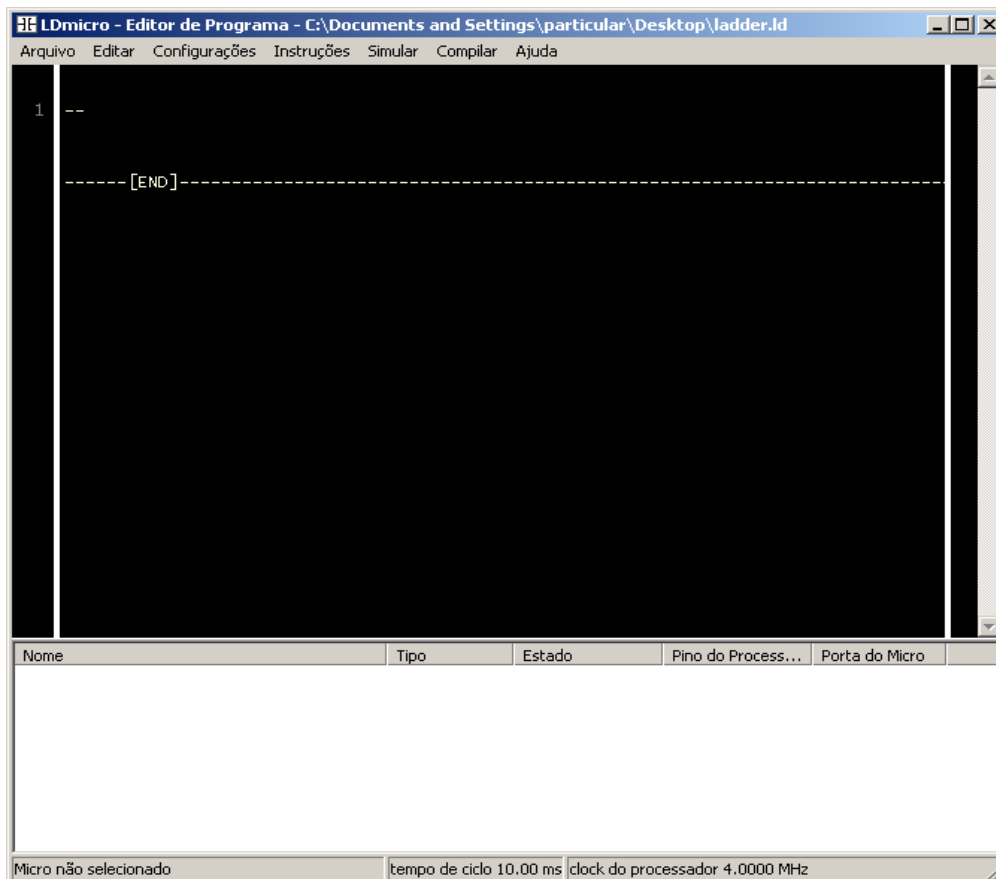
14.1 Objetivo:

O objetivo desta aula é demonstrar aos alunos o uso da linguagem Ladder na programação dos microcontroladores, de forma a tornar o processo de programação mais simples e rápido.

Utilizaremos o compilador LDmicro, que converte a linguagem ladder para a linguagem de máquina dos microcontroladores AVR.

14.2 O programa LDmicro

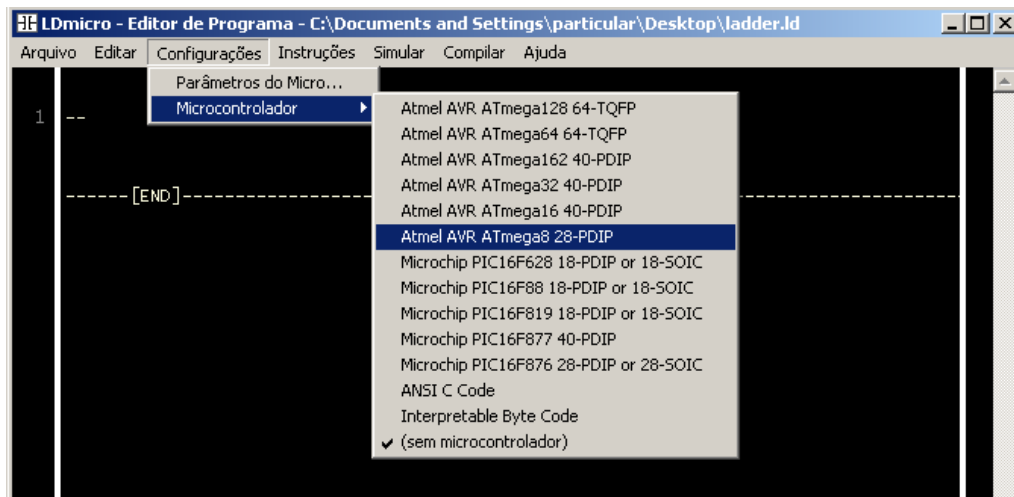
Para programarmos os microcontroladores em linguagem ladder utilizaremos o programa LDmicro, que pode ser obtido no site: <http://www.cq.cx/ladder.pl> A versão em português se chama “ldmicro-pt.exe”. O programa não necessita de instalação, basta salvá-lo. A figura a seguir apresenta a interface do software.



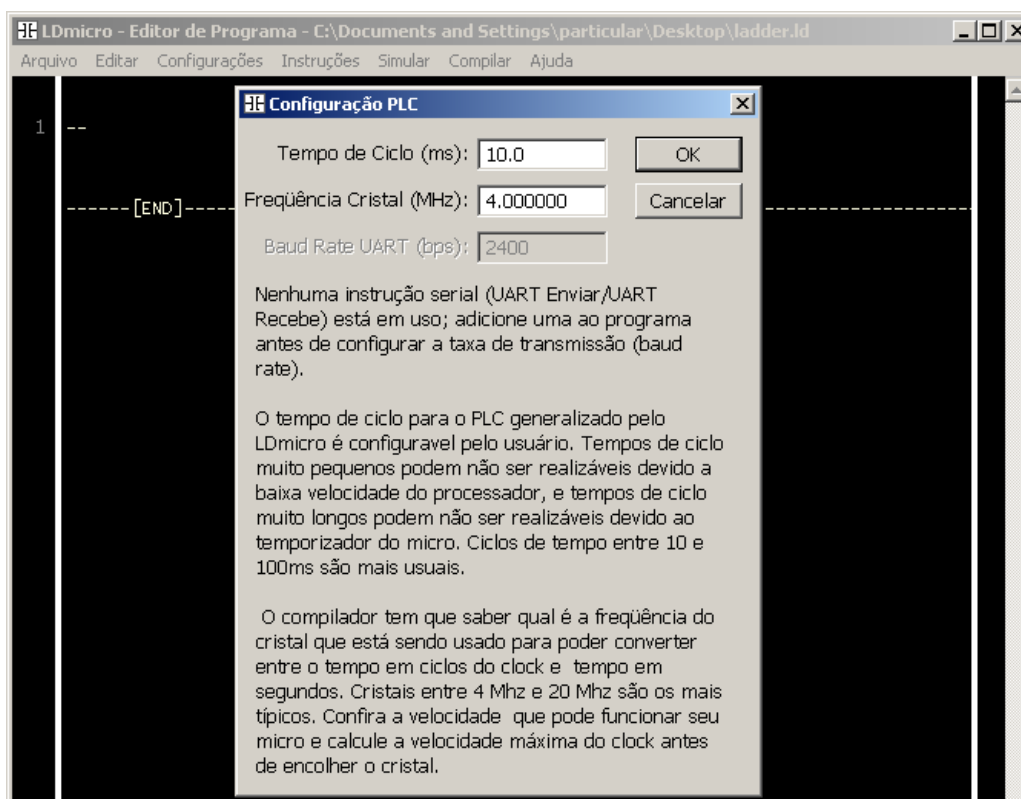
O programa é bastante simples e intuitivo, assim os detalhes a respeito da linguagem Ladder não serão abordados.

14.3 Iniciando um Programa em Ladder

Para iniciar um programa basta apenas escolher qual o microcontrolador como apresentado nas figuras a seguir.



Em seguida deve-se definir a frequência de clock utilizada, como na figura a seguir.



A seguir estão os comandos da linguagem Ladder que este programa disponibiliza.

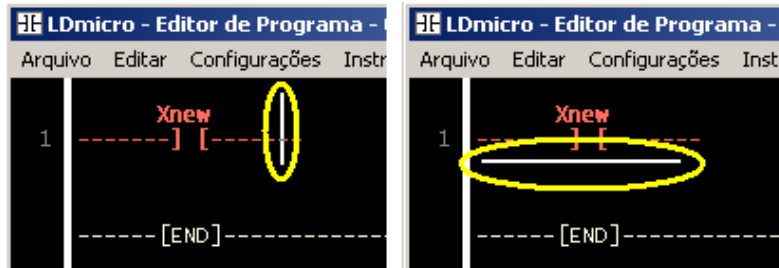
14.4 Programando no Ldmicro

Obs. O fabricante do programa não recomenda seu uso em aplicações que envolvam risco de vida.

A programação no LDmicro é feita através da inserção de instruções, estes comandos podem ser escolhidos do menu “Instruções”.

O comando é inserido na posição onde o cursor esta posicionado, assim o que define se uma instrução esta em série ou em paralelo com outra é a posição do cursor.

Veja dois exemplos nas figuras a seguir.



Outro ponto que deve ser levado em consideração é a nomenclatura dos comandos, o LDmicro utiliza a seguinte nomenclatura:

Xnome – Utilizado para dar nomes aos pinos de entrada.

Ynome – Utilizado para dar nomes aos pinos de saída.

Rnome – Utilizado para dar nomes às variáveis internas de 1 bit.

Tnome – Utilizado para dar nomes aos temporizadores.

Cnome – Utilizado para dar nomes aos contadores.

Anome – Utilizado para dar nomes aos valores das entradas analógicas.

nome – Utilizado para dar nomes às variáveis internas de 16 bits.

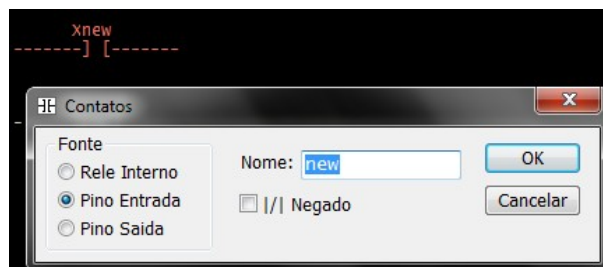
Obs: “nome” é o nome digitado pelo usuário.

Todas as variáveis são inteiros de 16 bits, assim podem assumir valores de -32768 à 32767.

Os comandos disponíveis no ldmicro serão vistos a seguir.

14.5 Instruções do Ldmicro

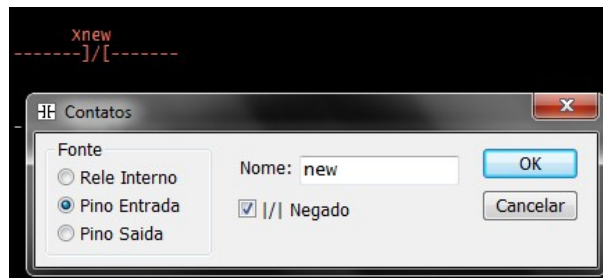
14.5.1 Contato Normalmente Aberto



Se o sinal que chega a esta instrução é falso então a saída também é falsa. Se o sinal que chega a esta instrução for verdadeiro então a saída é verdadeira se e somente se o rele interno ou o

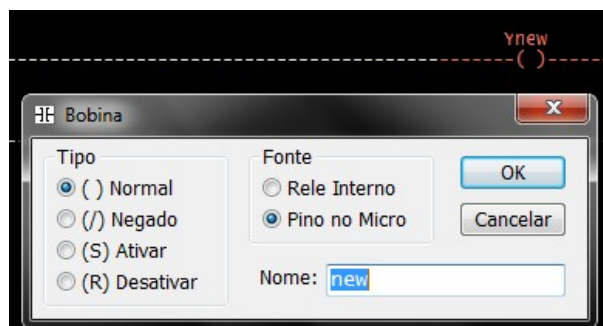
pino de entrada ou o pino de saída dado for verdadeiro. Ou seja, esta instrução examina o estado de um rele interno, ou de um pino de entrada ou de um pino de saída. A tecla de atalho é “c”

14.5.2 Contato Normalmente Fechado



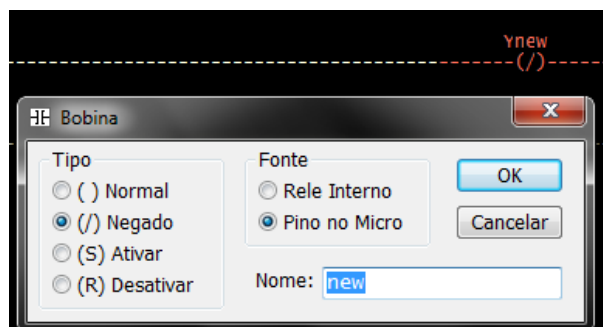
Se o sinal que chega a esta instrução é falso então a saída também é falsa. Se o sinal que chega a esta instrução for verdadeiro então a saída é verdadeira se e somente se o rele interno ou o pino de entrada ou o pino de saída dado for Falso. Esta instrução também examina o estado de um rele interno, ou de um pino de entrada ou de um pino de saída. A função desta instrução é o oposto do contato normalmente aberto. A tecla de atalho é “c”

14.5.3 Bobina Normal



Se o sinal que chega a esta instrução é falso então o pino de saída ou o rele interno associado a ela também é falso. Se o sinal que chega a esta instrução é verdadeiro então o pino de saída ou o rele interno associado a ela também é verdadeiro. Esta instrução deve sempre ser a instrução mais a direita da linha. A tecla de atalho é “l”

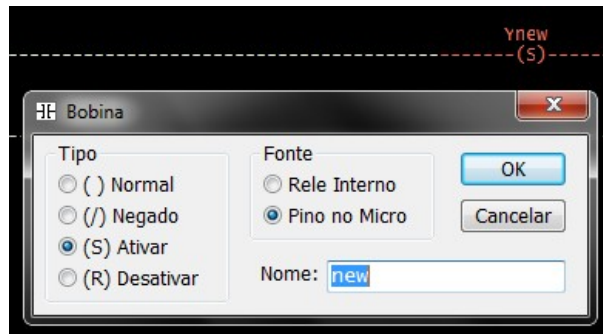
14.5.4 Bobina Negada



Se o sinal que chega a esta instrução é verdadeiro então o pino de saída ou o rele interno

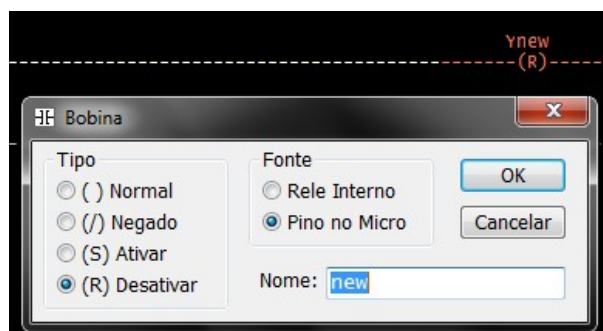
associado a ela é falso. Se o sinal que chega a esta instrução é falso então o pino de saída ou o rele interno associado a ela é verdadeiro. Esta instrução deve sempre ser a instrução mais a direita da linha. A tecla de atalho é “I”

14.5.5 Bobina Ativar



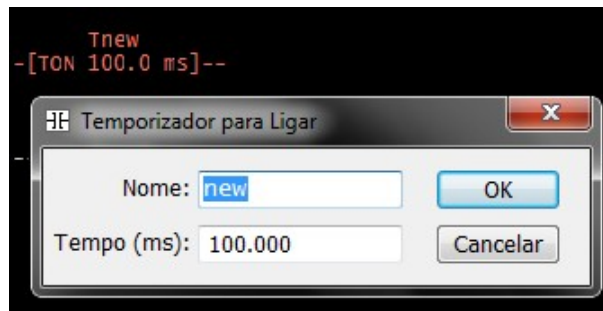
Se o sinal que chega a esta instrução é verdadeiro então o pino de saída ou o rele interno é ativado. Caso contrário o estado do pino de saída ou do rele interno não é alterado. Esta instrução consegue ativar um pino de saída ou um rele interno, mas não consegue desativá-los. Esta instrução é usada junto com a instrução bobina desativar. Esta instrução deve sempre ser a instrução mais a direita da linha. A tecla de atalho é “I”

14.5.6 Bobina Desativar



Se o sinal que chega a esta instrução é verdadeiro então o pino de saída ou o rele interno é desativado. Caso contrário o estado do pino de saída ou do rele interno não é alterado. Esta instrução consegue desativar um pino de saída ou um rele interno, mas não consegue ativá-los. Esta instrução é usada junto com a instrução bobina ativar. Esta instrução deve sempre ser a instrução mais a direita da linha. A tecla de atalho é “I”

14.5.7 Temporizador para Ligar



Quando o sinal que chega a esta instrução passa de falso para verdadeiro, o sinal de saída desta instrução permanece falso durante o tempo ajustado e após este tempo passa a verdadeiro. Quando o sinal que chega a esta instrução passa de verdadeiro para falso, o sinal de saída desta instrução se torna falso imediatamente.

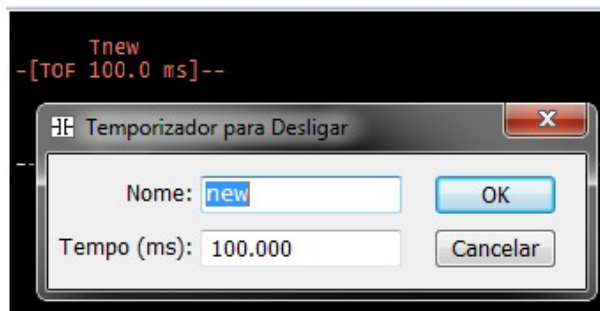
O temporizador é reiniciado toda vez que o sinal de entrada se torna falsa.

A variável utilizada no temporizador conta a partir de zero em unidades do tempo de scan, definido para o programa.

Esta instrução torna seu sinal de saída verdadeiro quando a contagem ultrapassa o valor ajustado.

A variável do contador pode ser manipulada por instruções “MOV” dentre outras.

14.5.8 Temporizador para Desligar



Quando o sinal que chega a esta instrução passa de verdadeiro para falso, o sinal de saída desta instrução permanece verdadeiro durante o tempo ajustado e após este tempo passa a falso. Quando o sinal que chega a esta instrução passa de falso para verdadeiro, o sinal de saída desta instrução se torna verdadeiro imediatamente.

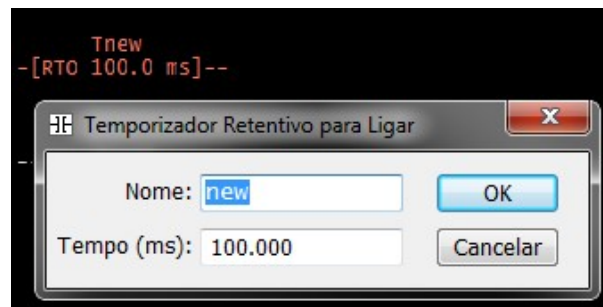
O temporizador é reiniciado toda vez que o sinal de entrada se torna verdadeiro.

A variável utilizada no temporizador conta a partir de zero em unidades do tempo de scan, definido para o programa.

Esta instrução torna seu sinal de saída verdadeiro quando a contagem ultrapassa o valor ajustado.

A variável do contador pode ser manipulada por instruções “MOV” dentre outras.

14.5.9 Temporizador Retentivo para Ligar



Esta instrução monitora quanto tempo o seu sinal de entrada permanece verdadeiro, se o sinal permanecer verdadeiro pelo tempo ajustado o sinal de saída se torna verdadeiro. Não é necessário que o sinal de entrada permaneça o tempo todo verdadeiro, ele pode passar a falso e voltar a ser verdadeiro, pois a instrução conta o tempo total que o sinal permaneceu verdadeiro, não zerando sua contagem quando o sinal de entrada passa a ser falso. Quando o sinal que chega a esta instrução passa de falso para verdadeiro, o sinal de saída desta instrução se torna verdadeiro imediatamente.

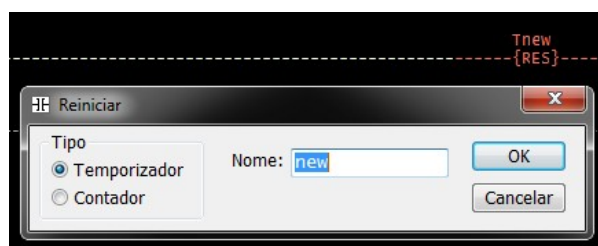
Este temporizador deve ser zerado manualmente utilizando a instrução “RES”.

A variável utilizada no temporizador conta a partir de zero em unidades do tempo de scan, definido para o programa.

Esta instrução torna seu sinal de saída verdadeiro quando a contagem ultrapassa o valor ajustado.

A variável do contador pode ser manipulada por instruções “MOV” dentre outras.

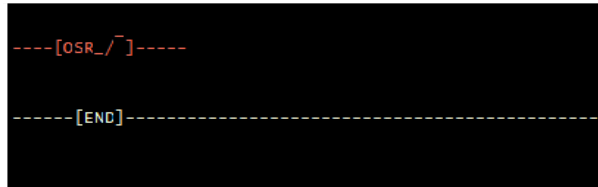
14.5.10 Reinicia Contador / Temporizador



Esta instrução reinicia um contador ou um temporizador. O temporizador retentivo para ligar, o contador incremental e o contador decremental não são reiniciados automaticamente, assim é necessário utilizar esta instrução para fazê-lo.

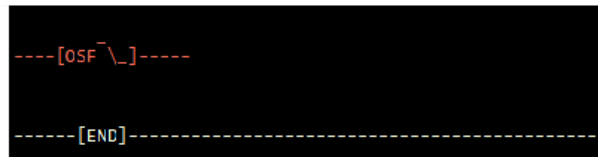
Quando o sinal de entrada desta instrução é verdadeiro o contador ou temporizador associado é reiniciado. Quando o sinal de entrada é falso, nada acontece. Esta instrução deve ser a instrução mais a direita na linha.

14.5.11 Detecta Borda de Subida



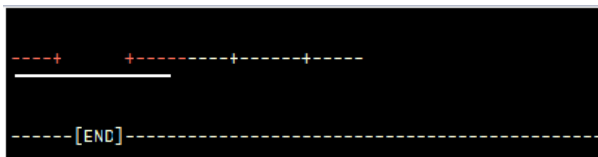
Esta instrução normalmente apresenta um sinal falso em sua saída. Se o sinal de entrada estiver verdadeiro durante o ciclo de scan atual e estava falso no ciclo de scan anterior, então o sinal de saída permanece verdadeiro durante um ciclo de scan. Esta instrução é útil se desejarmos disparar um evento na borda de subida de um sinal.

14.5.12 Detecta Borda de Descida



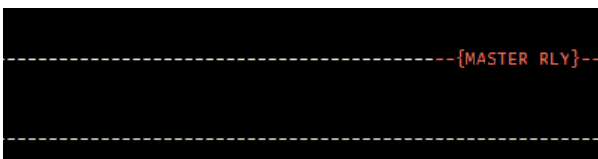
Esta instrução normalmente apresenta um sinal falso em sua saída. Se o sinal de entrada estiver falso durante o ciclo de scan atual e estava verdadeiro no ciclo de scan anterior, então o sinal de saída permanece verdadeiro durante um ciclo de scan. Esta instrução é útil se desejarmos disparar um evento na borda de descida de um sinal.

14.5.13 Circuito Aberto e Curto Circuito



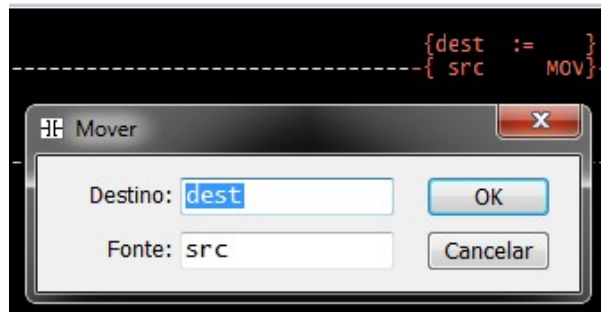
O sinal de saída de uma instrução de circuito aberto é sempre falso e o sinal de saída de uma instrução de curto circuito é sempre igual ao sinal de entrada. Estes comandos são utilizados para realizar teste durante o desenvolvimento dos programas.

14.5.14 Rele de Controle Mestre



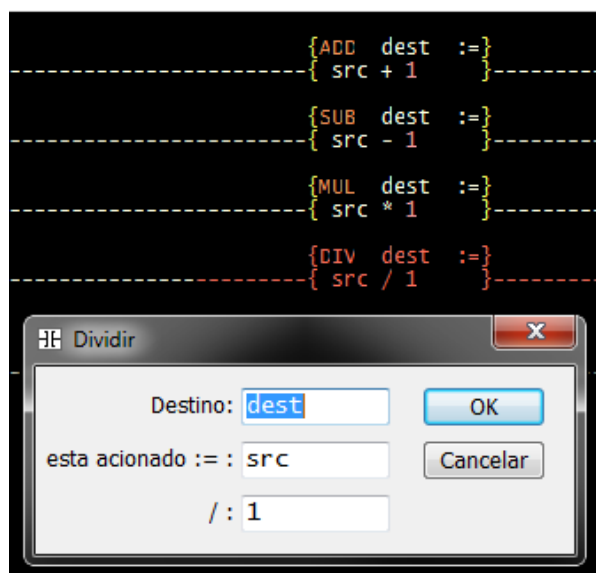
mestre for executada com um sinal de entrada falso, então o sinal de entrada para as linhas a seguir também se torna falso. Isto permanece até que outra instrução de rele de controle mestre seja executada, independente de su sinal de entrada. Esta instrução deve sempre ser usada em pares, uma para iniciar a região a ser desabilitada e outra para finalizá-la.

14.5.15 Mover



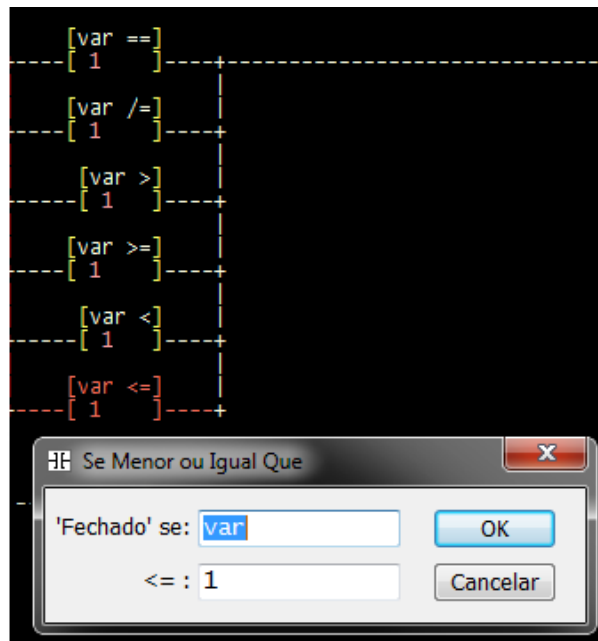
Quando o sinal de entrada desta instrução é verdadeiro, ela copia o valor da variável fonte para a variável destino. Quando o sinal de entrada é falso, nada acontece. É possível utilizar qualquer variável na instrução mover, isso inclui as variáveis de temporizadores, contadores etc. É necessário adicionar as letras que antecedem o nome das variáveis como por exemplo C e T nos contadores e temporizadores.

14.5.16 Operações Aritméticas



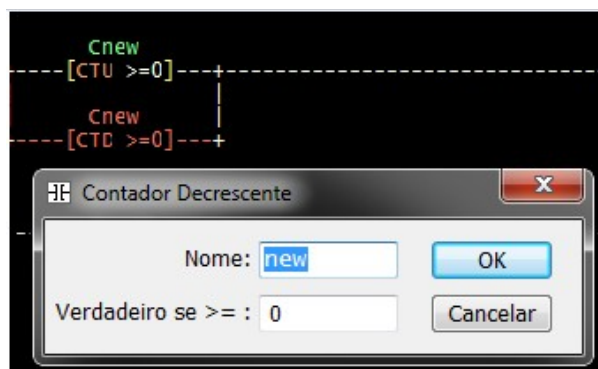
Quando o sinal de entrada desta instrução é verdadeiro, a variável de destino recebe o resultado da expressão selecionada. Os operadores podem ser variáveis ou constantes. É importante lembrar que são utilizadas variáveis inteiras de 16 bits. As operações suportadas são a soma, a subtração, a multiplicação e a divisão. Esta instrução deve ser a instrução mais a direita da linha.

14.5.17 Comparações



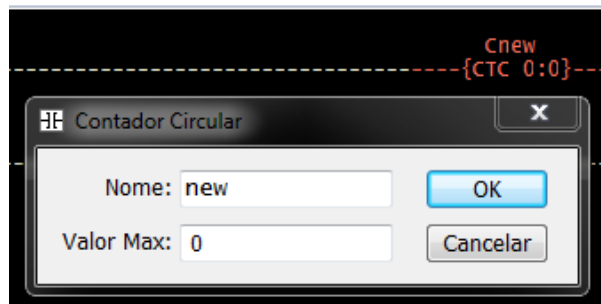
Quando o sinal de entrada desta instrução é falso, o sinal de saída também é falso. Quando o sinal de entrada desta instrução é verdadeiro e a condição do teste também é verdadeira o sinal de saída é verdadeiro. Esta instrução pode comparar variáveis ou constantes. Os testes realizados são: igual, diferente, maior, maior ou igual, menor e menor ou igual.

14.5.18 Contadores



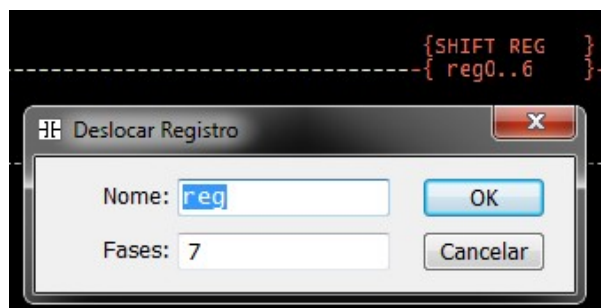
Um contador incrementa ou decrementa o valor de uma variável a cada borda de subida do seu sinal de entrada. O sinal de saída é verdadeiro se o valor do contador for maior ou igual ao valor ajustado. O sinal de saída permanece verdadeiro mesmo se o sinal de entrada for falso. As instruções "RES" e "MOV" pode ser usada para atribuir valores aos contadores.

14.5.19 Contador Circular



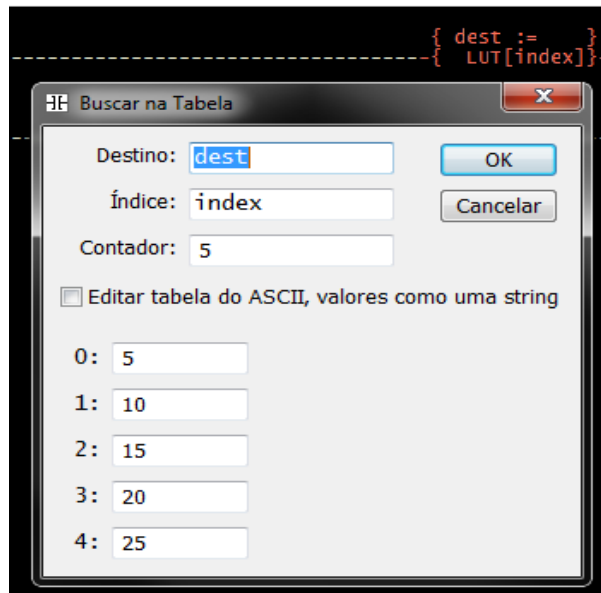
Um contador circular inicia contando de forma a incrementar a variável de controle, quando o valor ajustado é atingido o contador retorna a zero e inicia uma nova contagem. Esta instrução é utilizada junto com as instruções de comparação. Esta instrução deve ser a instrução mais a direita na linha.

14.5.20 Deslocar Registro



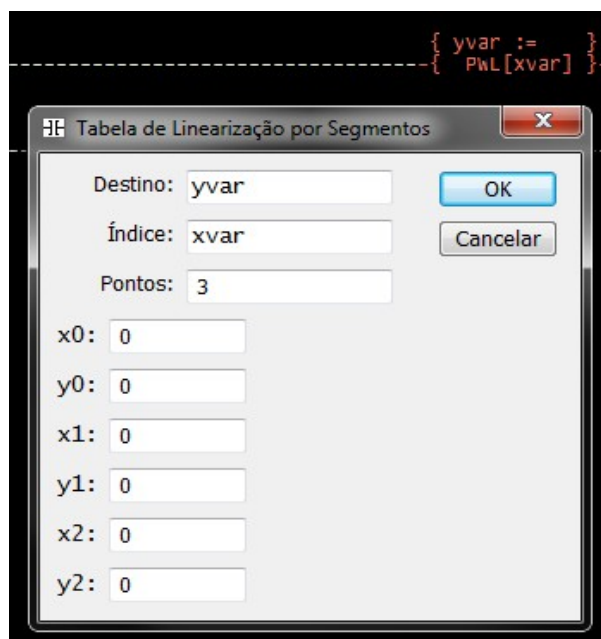
A instrução de deslocar registro é associada a um grupo de registrador, e toda vez que o sinal em sua entrada tem uma transição de falso para verdadeiro, os valores destes registradores são deslocados. Por exemplo, digamos que a instrução esteja operando com 5 registradores, reg0, reg1, reg2, reg3 e reg4. Quando a instrução detectar uma borda de subida no sinal de entrada ela vai copiar o conteúdo do registrador reg3 para o registrador reg4, o conteúdo do registrador reg2 para o registrador reg3, o conteúdo do registrador reg1 para o registrador reg2 e o conteúdo do registrador reg0 para o registrador reg1. Esta instrução deve ser a instrução mais a direita na linha.

14.5.21 Busca na Tabela



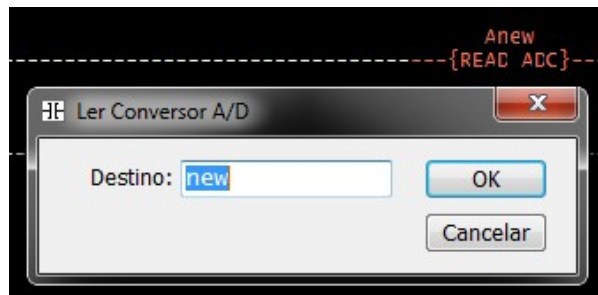
A instrução de busca na tabela trabalha com um conjunto de valores armazenados em uma tabela. Quando o sinal de entrada da instrução é verdadeiro, a variável de destino recebe o valor da tabela apontado pela variável de índice. É importante lembrar que a variável de índice não deve ultrapassar o número de elementos da tabela menos 1. Esta instrução deve ser a instrução mais a direita na linha.

14.5.22 Linearização por Segmentos



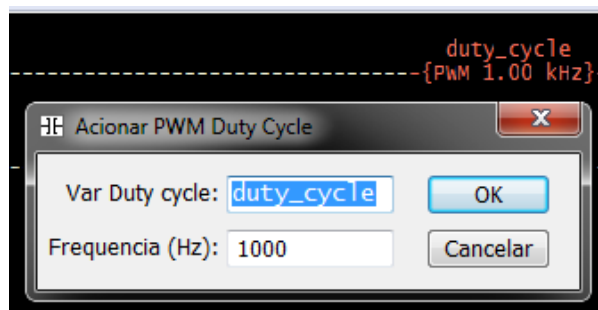
Esta instrução é utilizada para aproximar funções complexas ou curvas através de um conjunto de pares de pontos. Esta instrução pode ser utilizada por exemplo para transformar a entrada de um sensor no valor correto da grandeza medida.

14.5.23 Ler Conversor A/D



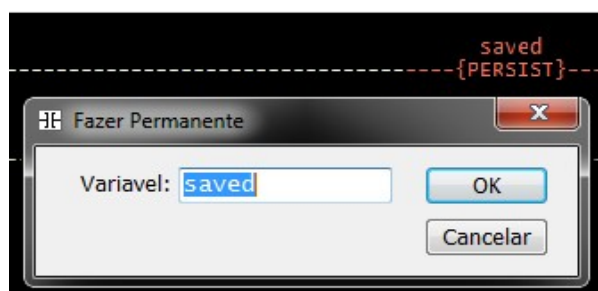
O Ldmicro pode gerar o código necessário para usar o conversor analógico digital interno dos microcontroladores. Se o sinal de entrada desta instrução é verdadeira, então uma amostra do conversor A/D é adquirida e armazenada na variável de destino. É necessário atribuir um pino a cada instrução de leitura do A/D. Se o sinal de entrada da instrução é falso o conteúdo da variável não é alterado. Esta instrução deve ser a instrução mais a direita na linha.

14.5.24 Valor do PWM



O Ldmicro é capaz de gerar o código necessário para utilizar o PWM presente nos microcontroladores. Se o sinal de entrada desta instrução for verdadeiro, então a razão cíclica do PWM é ajustada de acordo com o valor da variável fornecida. O valor deve estar entre 0 e 100. É possível ajustar a frequência do PWM em hertz, porem a frequência não é muito precisa. Esta instrução deve ser a instrução mais a direita na linha.

14.5.25 Fazer Permanente



Quando o sinal de entrada desta instrução é verdadeiro, a variável especificada é armazenada na memória EEPROM do microcontrolador. Assim o conteúdo desta variável não é perdido quando o microcontrolador é desligado.

O conteúdo da variável é lido da EEPROM toda vez que o microcontrolador é ligado.

É necessário tomar cuidado pois a memória EEPROM só suporta 100000 operações de escrita em um determinado endereço. Esta instrução deve ser a instrução mais a direita na linha.

14.5.26 UART Receber

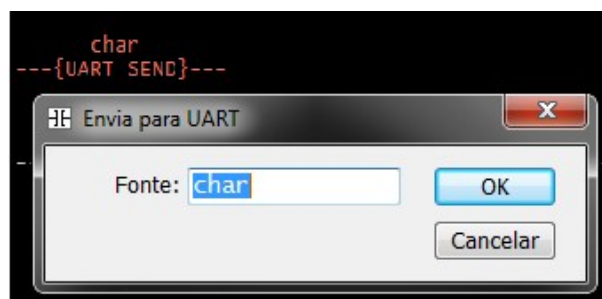


O Ldmicro gera código para utilizarmos a porta de comunicação serial do microcontrolador, chamada UART. A velocidade de transmissão é ajustada no menu configurações. Nem todas as taxas de transmissão são possíveis dependendo do clock do processador.

Se o sinal de entrada desta instrução for verdadeiro, a instrução irá tentar receber um caractere da UART.

Se não houver caractere para ser lido o sinal de saída permanece falso. Se houver caractere para ser lido, este caractere é armazenado na variável fornecida e o sinal de saída permanece verdadeiro por 1 ciclo.

14.5.27 UART Enviar



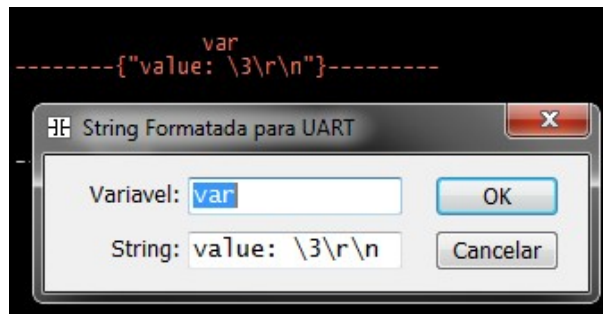
O Ldmicro gera código para utilizarmos a porta de comunicação serial do microcontrolador, chamada UART. A velocidade de transmissão é ajustada no menu configurações. Nem todas as taxas de transmissão são possíveis dependendo do clock do processador.

Se o sinal de entrada desta instrução for verdadeiro, a instrução irá escrever um caractere na

UART. O valor ascii do caractere a ser enviado deve previamente ser armazenado na variável fornecida. O sinal de saída desta instrução é verdadeiro se a UART estiver ocupada enviando um caractere.

É importante lembrar que os caracteres demoram um certo tempo para ser enviados, assim é importante verificar se a UART não esta ocupada antes de enviar um segundo caractere.

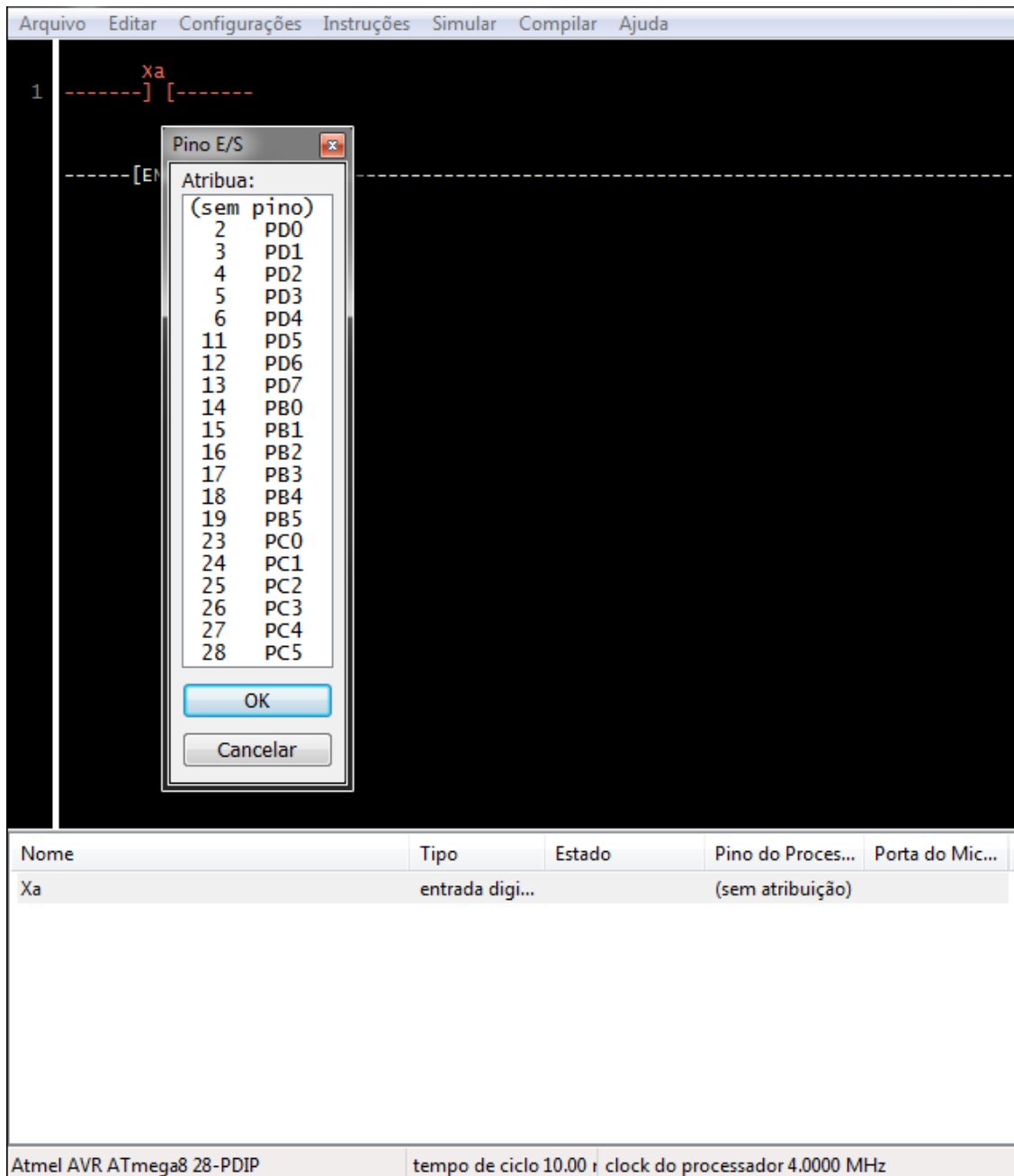
14.5.28 UART Enviar



O Ldmicro gera código para utilizarmos a porta de comunicação serial do microcontrolador, chamada UART. A velocidade de transmissão é ajustada no menu configurações. Nem todas as taxas de transmissão são possíveis dependendo do clock do processador. Quando o sinal de entrada da instrução passa de falso para verdadeiro, a instrução inicia a transmissão de uma cadeia de caracteres (string). Se a cadeia de caracteres conter a sequencia “\n” onde n é o número máximo de casas, então esta sequencia é substituída para valor da variável fornecida. Seu conteúdo é transformado em texto. Mais informações sobre esta e outras instruções podem ser obtidas no manual do software.

14.6 Definição dos Pinos

Depois de finalizado o programa é necessário definir quais serão os pinos de entrada e saída, isto pode ser feito clicando com o mouse sobre a linha com o nome do sinal e na coluna chamada “Pino do processador”. Quando clicado aparecerá uma janela onde o pino do microcontrolador referente a aquele sinal pode ser escolhido. A figura a seguir mostra este procedimento.



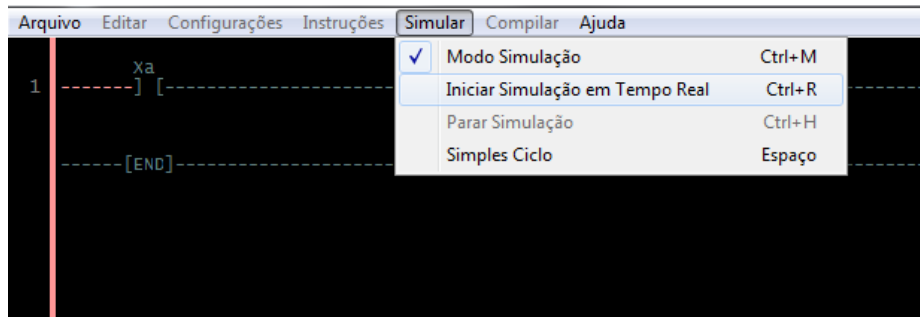
The screenshot shows a software interface for configuring a microcontroller. A dialog box titled "Pino E/S" is open, displaying a list of pins and their associated ports. The list includes pins 2 through 28, grouped into PD0-7, PB0-5, and PC0-5. Below the list are "OK" and "Cancelar" buttons. In the background, a circuit diagram shows a pin labeled "Xa" connected to a digital input. Below the diagram is a table with the following data:

Nome	Tipo	Estado	Pino do Proces...	Porta do Mic...
Xa	entrada digi...		(sem atribuição)	

At the bottom of the interface, the microcontroller model is identified as "Atmel AVR ATmega8 28-PDIP" with a "tempo de ciclo 10.00" and "clock do processador 4.0000 MHz".

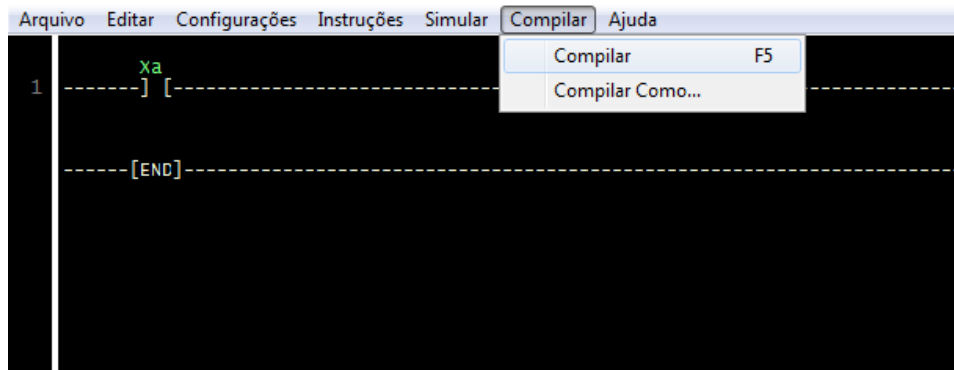
14.7 Simulação do Programa

É possível simular o programa para verificar seu funcionamento, para isso é necessário clicar em Simular / modo de simulação e em seguida Iniciar simulação em tempo real. A figura a seguir ilustra este processo.



14.8 Compilar o Programa

Para que se possa transferir o programa para o microcontrolador é necessário compilá-lo, de forma a gerar o arquivo de saída em formato HEX. Para isso basta clicar em Compilar / Compilar. A figura a seguir ilustra este processo.



14.9 Exercícios

1. Faça um programa em linguagem ladder, utilizando o software Ldmicro. O programa deve controlar o nível de líquido em um tanque. O sensor de nível envia um sinal analógico ao conversor A/D 0 do microcontrolador, onde após convertido o sinal é 0 para 0 metros de líquido e 1000 para 10 metros de líquido. O controle deve acionar a bomba conectada ao pino PD0 quando o nível for inferior a 8 metro e desligar quando for superior a 9 metros.
2. Simule o programa do exercício anterior, utilizando o simulador do Ldmicro.

PRÁTICAS

AULA 1 - CIRCUITOS COM MICROCONTROLADORES

Estudo dos circuitos elétricos necessários utilizados com microcontroladores

1.1 Objetivo:

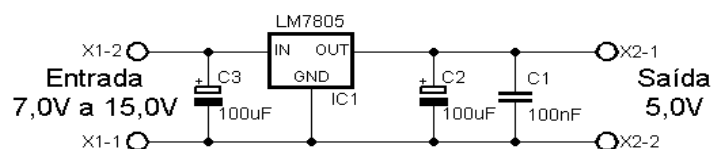
O objetivo desta aula é apresentar alguns circuitos utilizando microcontroladores, assim os alunos irão compreender como os microcontroladores são conectados ao restante do circuito. As principais conexões são entre o microcontrolador e a fonte de alimentação e entre o microcontrolador e o circuito clock. O aprendizado deste conteúdo será conduzido através de exemplos e exercícios.

1.2 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 AVR 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.

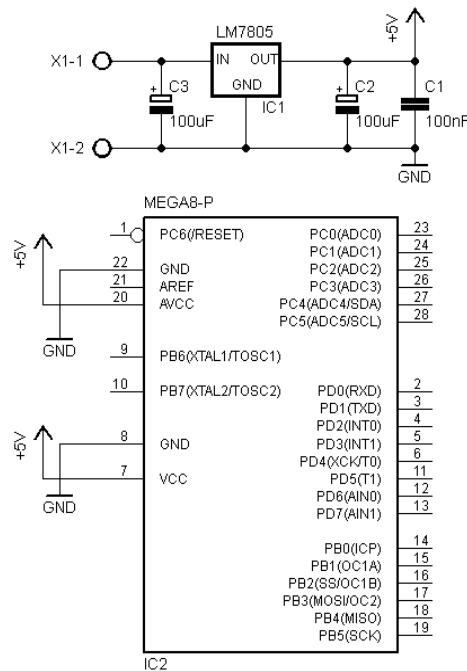
Na grande maioria das aplicações de microcontroladores a tensão de operação é de 5,0V, e para que esta tensão permaneça estável e livre de ruídos é apropriado que se utilize um circuito regulador de tensão protegido por filtros capacitivos.

A figura a seguir apresenta um circuito para esta função onde o regulador de tensão é o LM7805.



1.3 A conexão do microcontrolador com a fonte

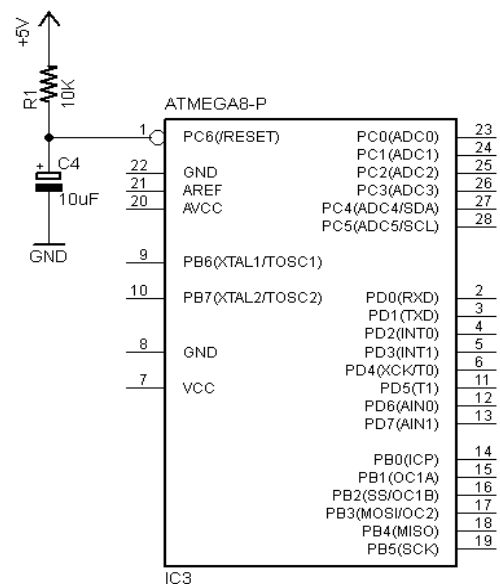
Os microcontroladores da família AVR possuem duas entradas de alimentação distintas, uma para os circuitos digitais e outra para o circuito analógico do conversor analógico digital. A figura a seguir apresenta a conexão dos pinos 7 e 8 do microcontrolador a fonte, correspondendo a alimentação dos circuitos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

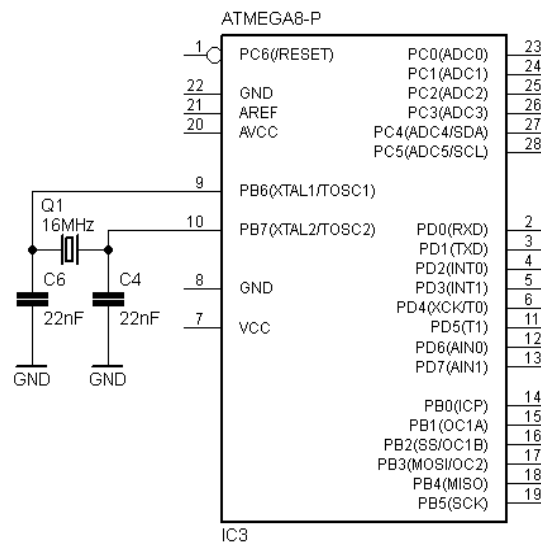
1.4 O circuito de reset

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, porem o pino de reset deve ser conectado a um resistor e a um capacitor conforme a figura ao lado.



1.5 O circuito de clock

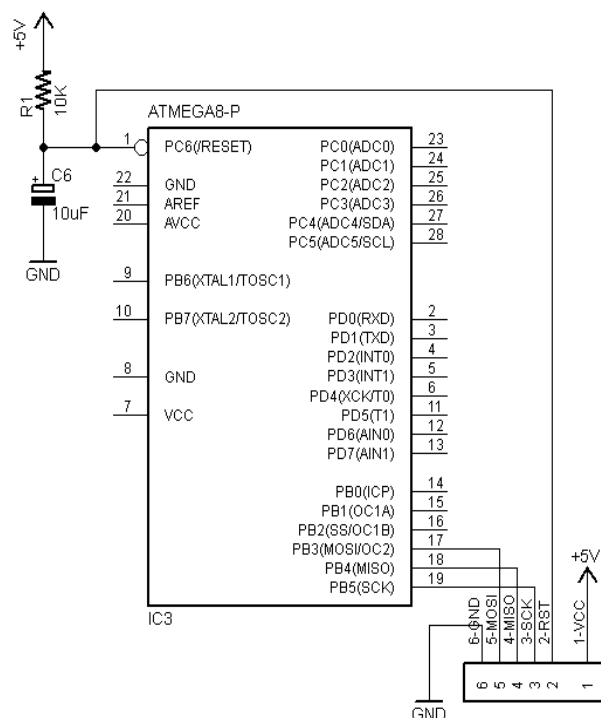
Todos os circuitos microprocessados, inclusive os microcontroladores necessitam de um circuito de relógio, ou circuito de clock. O microcontrolador ATmega8 possui um circuito interno de clock, que pode funcionar de duas formas. A primeira é através de um oscilador RC interno, o que simplifica o circuito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

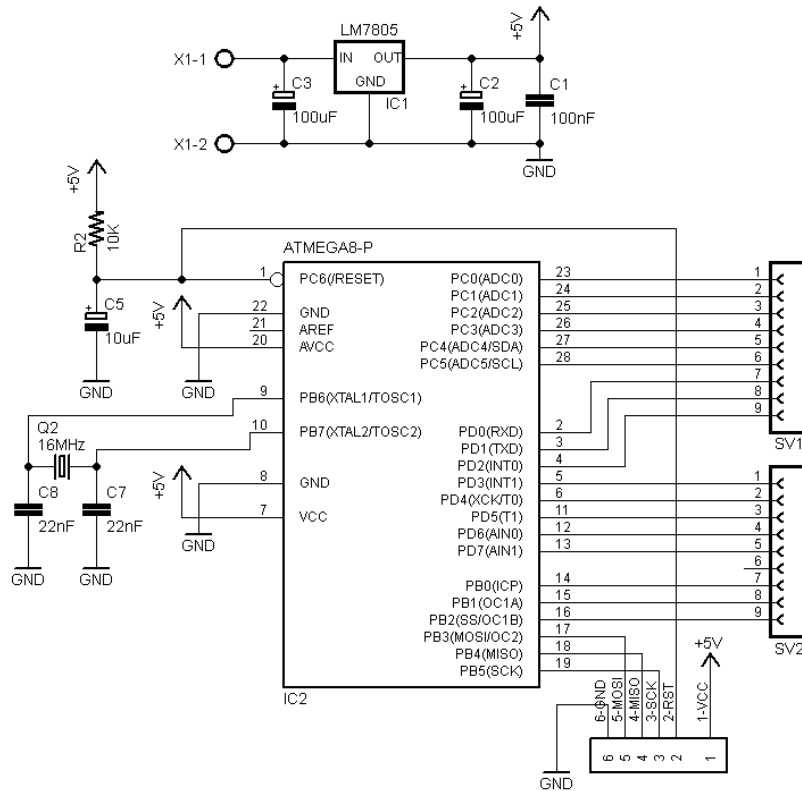
1.6 Conector de programação

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, porem em nossas aulas utilizaremos o circuito da figura ao lado.

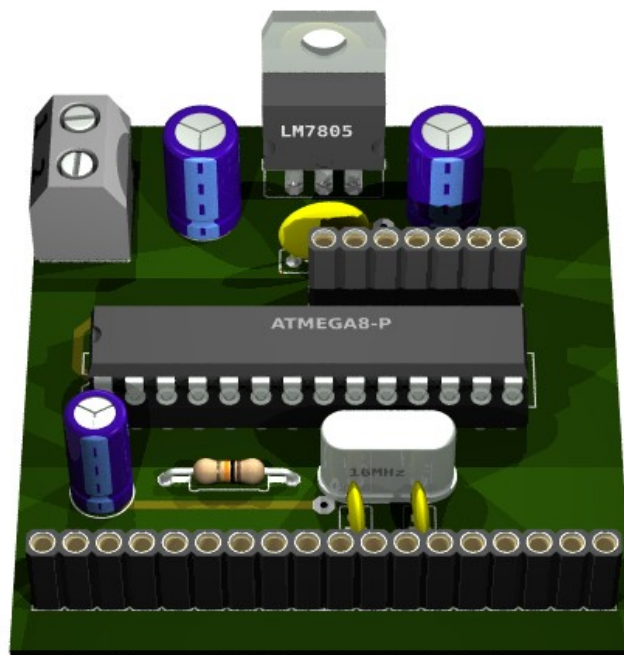


1.7 Circuito completo

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

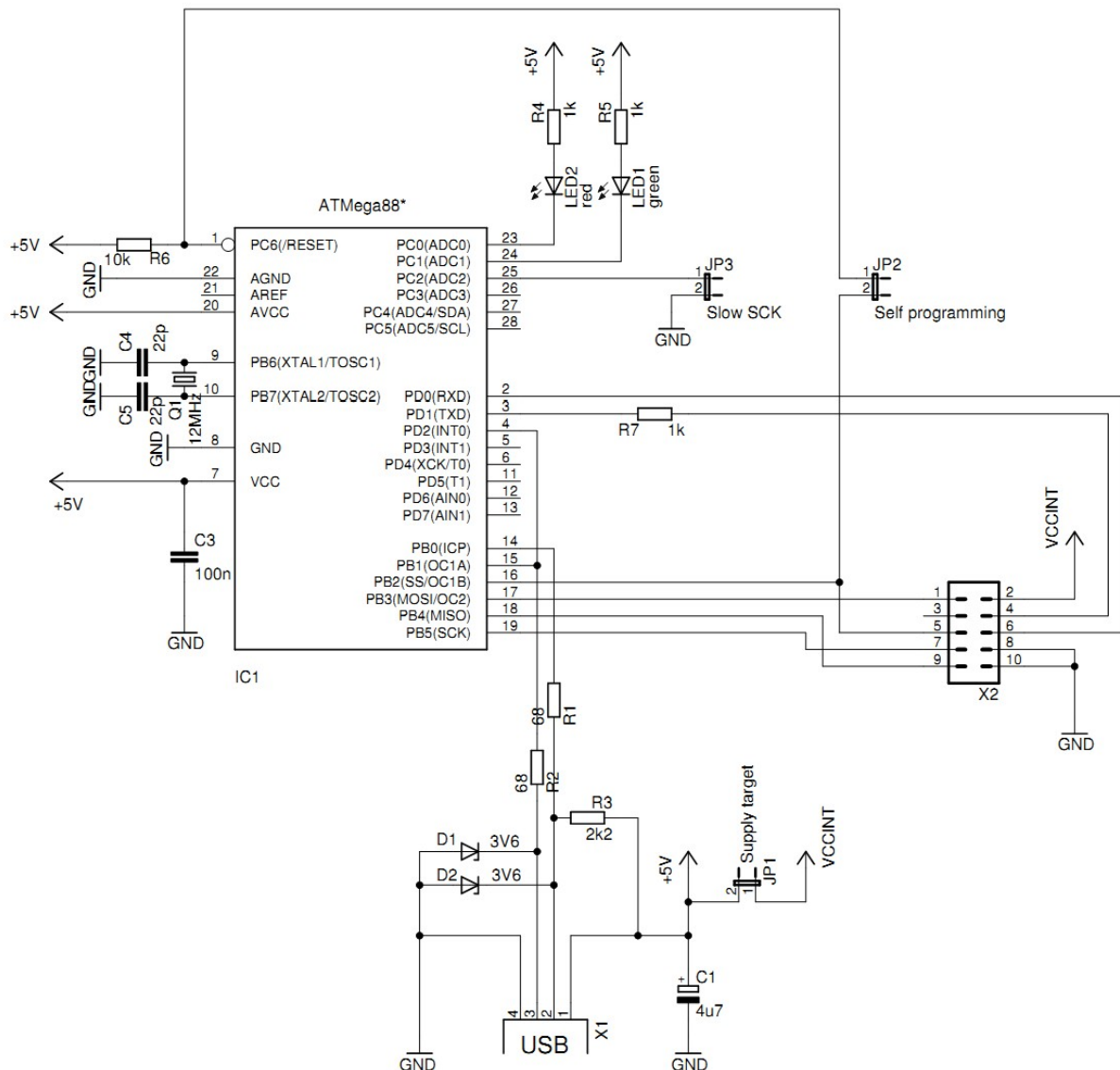


Foram adicionados dois conectores para interligar os pinos de entrada e saída na figura anterior. A imagem final da placa fica como na figura a seguir.



1.8 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. Em nossas aulas utilizaremos o circuito da figura a seguir.



Mais informações sobre o circuito de programação utilizado podem ser obtidas no site: <http://www.fischl.de/usbsp/>

1.9 Gravação do programa no microcontrolador

Para transferirmos nossos programas para os microcontroladores é necessário um programa que rode no computador, o programa utilizado em nossas aulas será o “Khazama AVR Programme” que pode ser baixado no site: <http://www.khazama.com/project/programmer/>.

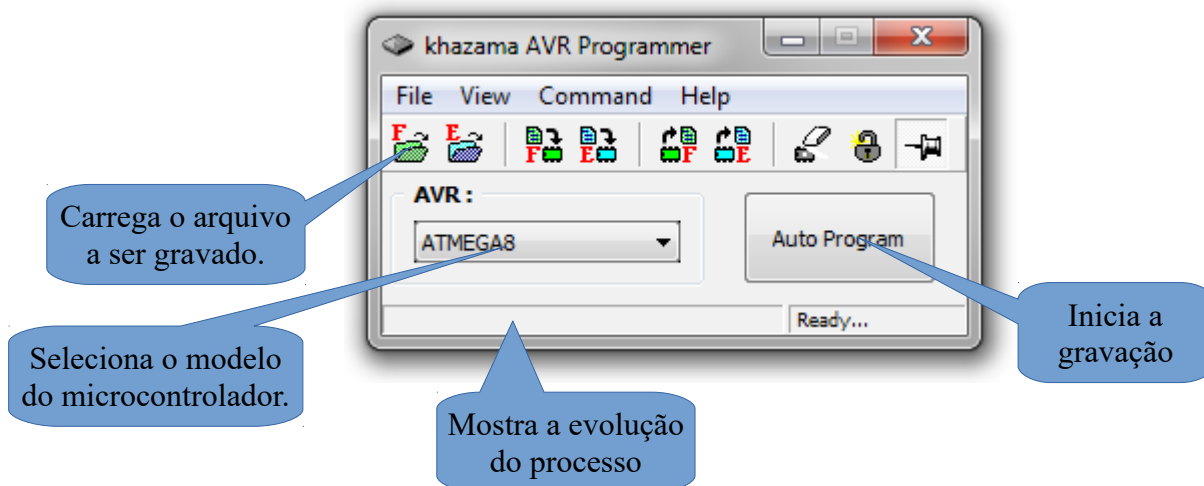
Este programa é responsável por ler o arquivo resultante da compilação de nosso programa e transferi-lo para o microcontrolador utilizando o circuito de programação apresentado anteriormente.

Na primeira vez em que conectamos o circuito na porta USB de nosso computador é necessário fornecer um driver para seu funcionamento. O driver pode ser baixado no próprio site do gravador.

Após conectarmos o gravador ao computador e ao microcontrolador é possível iniciar a gravação com a abertura do programa Khazama Programmer.

A Figura 41 apresenta a interface deste programa.

Figura 41: Interface do Khazama Programmer



Os balões que estão na Figura 41 mostram suas principais funções. Inicialmente é possível carregar o programa que se deseja gravar. Este programa está no formato hexadecimal, com a extensão “.hex”, no interior da pasta do projeto. Também é possível selecionar o modelo do microcontrolador que se está usando. Após os ajustes basta clicar no botão “Auto Programa” e observar a evolução do processo de gravação.

Este programa também pode realizar outras tarefas além de transferir programa. É através dele, por exemplo, que selecionamos a fonte de clock e protegemos nosso programa contra pirataria. Mas estes assuntos serão vistos em um outro momento.

1.10 Exercícios

1. Desenhe um circuito onde o microcontrolador ATmega8 é conectado a dois botões, dois LEDs e um relê. O circuito deve conter tudo que o microcontrolador necessita para funcionar, não é necessário utilizar um cristal de clock.
2. Monte com muito cuidado o circuito projetado no protoboard, conectando os circuitos de alimentação aos bornes do mesmo.
3. Faça um programa para o circuito do exercício anterior, em que quando um botão é pressionado um LED acende, o outro fica apagado e o relê liga. Quando o outro botão é pressionado os LEDs se alternam e o rele é desligado.
4. Transfira o programa para o microcontrolador do protoboard e comprove seu funcionamento.
5. Elabore para a próxima aula um relatório do experimento realizado, respeitando as normas para elaboração de trabalhos acadêmicos.

AULA 2 - ENTRADAS E SAÍDAS DIGITAIS

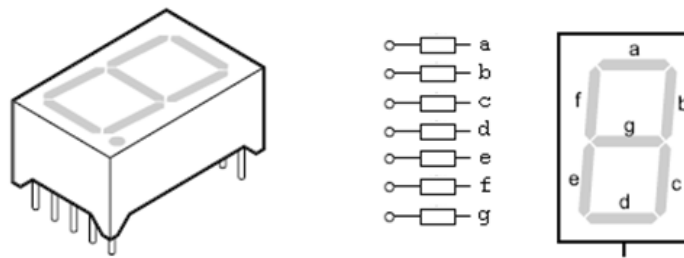
Nesta aula será feita uma montagem abordando entradas e saídas digitais

2.1 Objetivo:

O objetivo desta aula é exercitar com os alunos a montagem de circuitos microcontrolados operando entradas e saídas digitais, assim os alunos irão montar um circuito contendo o microcontrolador, seus circuitos auxiliares e alguns circuitos de entrada e saída.

2.2 Fundamentação

No experimento desta aula sera utilizado um display de sete segmentos, cuja configuração é apresentada na figura a seguir.



Também será necessário gerar intervalos de tempo para que os números apresentados no display se tornem visíveis ao operador. Estes intervalos de tempo podem ser gerados de várias formas, uma delas é apresentada no exemplo a seguir.

```
#include <avr/io.h>
void tempo(long temp)
{
while(temp>0)
    {
temp--;
    }
}

int main(void)
{
DDRB=0b000000100;
while(1)
    {
PORTB=PORTB&0b11111011;
tempo(10000);
PORTB=PORTB|0b00000100;
tempo(10000);
    }
}
```

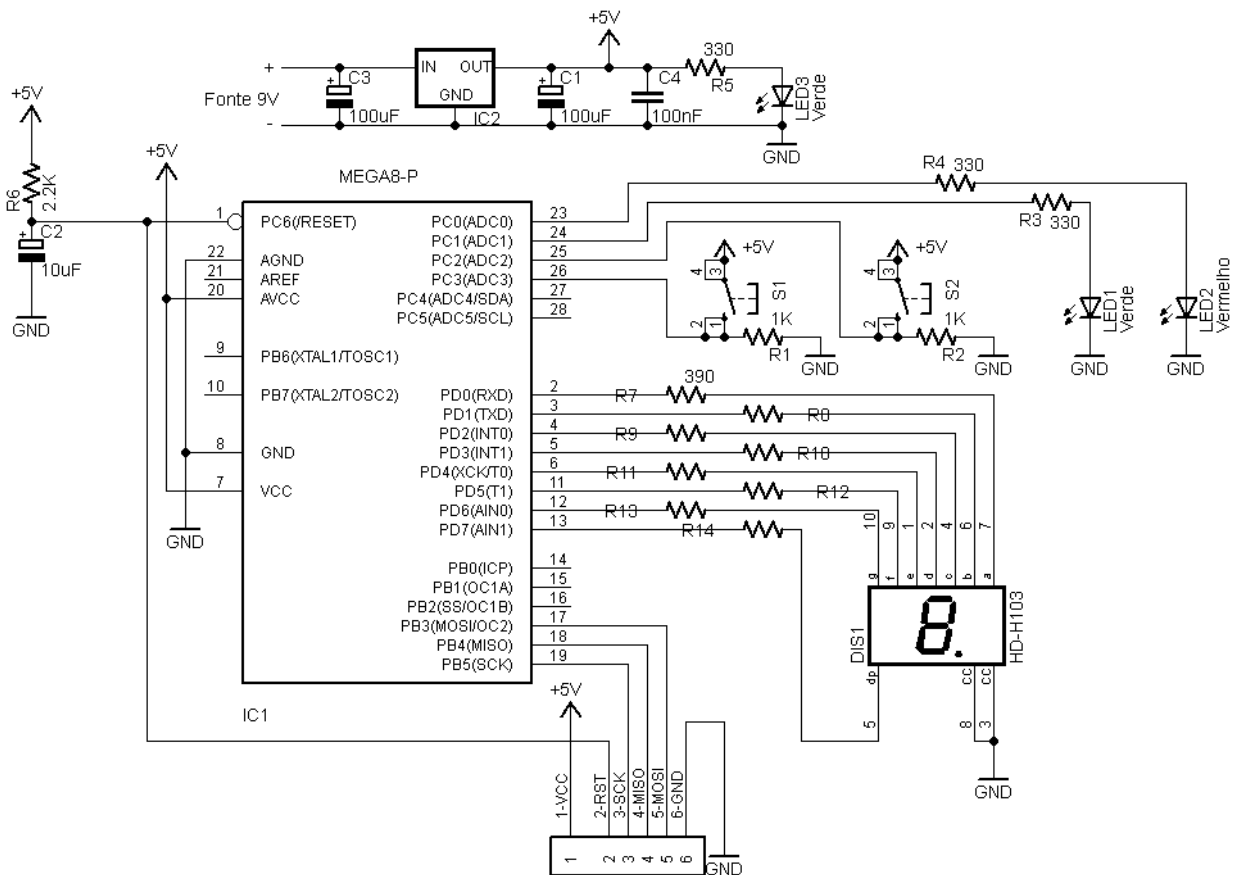

Neste exemplo foi utilizada uma sub-rotina chamada tempo, que recebe como argumento um número inteiro. Quanto maior o número passado para esta sub-rotina maior o intervalo de tempo gerado por ela. Existem formas mais eficientes de gerenciar intervalos de tempo nos microcontroladores, estes serão abordados em outras aulas.

No exemplo a sub-rotina de tempo foi utilizada para ligar e desligar o pino dois da porta B em intervalos de tempo definidos.

O intervalo de tempo gasto pela sub-rotina depende de dois fatores, do valor passado para a mesma e da frequência de clock que o microcontrolador está operando.

2.3 Exercícios

1. Construa o circuito da figura a seguir.



2. Faça um programa para o circuito do exercício anterior em que quando a chave s1 for pressionada o LED1 liga, o LED2 fica desligado e no display aparecem os números de 0 a 9 sequencialmente. Cada número deve permanecer por 1 segundo. Quando a chave s2 for pressionada o LED2 liga, o LED1 fica desligado e no display aparecem os números de 9 a 0 sequencialmente. Cada número deve permanecer por 1 segundo. Obtenha os tempos de 1 segundo ajustando o valor enviado a sub-rotina tempo.
3. Transfira o programa para o microcontrolador do protoboard e comprove seu funcionamento.
4. Elabore para a próxima aula um relatório do experimento realizado, respeitando as normas para elaboração de trabalhos acadêmicos.

AULA 3 - MODULAÇÃO PWM

Como utilizar as saídas digitais para gerar sinais PWM

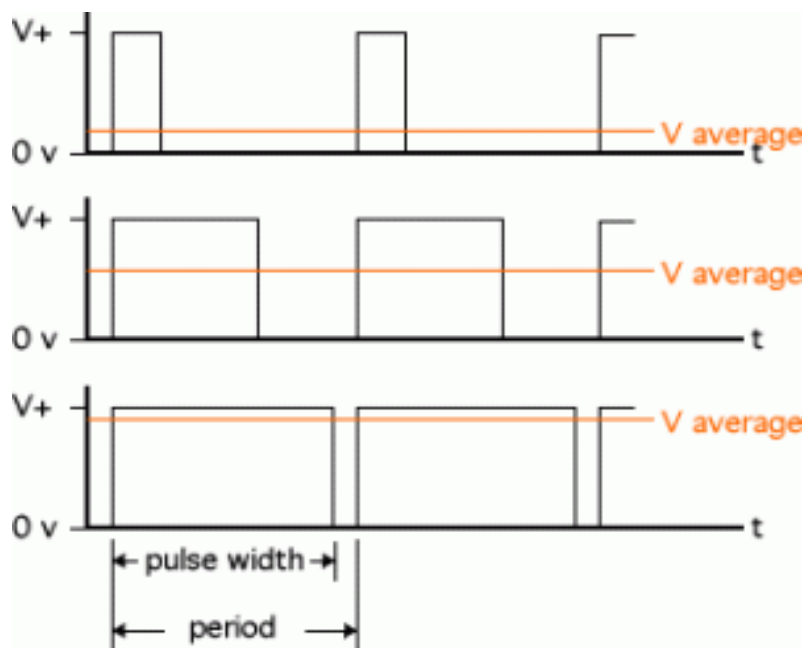
3.1 Objetivo:

O objetivo desta aula é estudar os sinais modulados por largura de pulso, e desenvolver um programa que utilize as saídas digitais para gerar estes sinais.

3.2 Fundamentação

Sinais modulados por largura de pulso (PWM), são sinais digitais pulsantes onde a frequência do sinal é mantida constante mas a razão cíclica varia de acordo com o valor médio desejado.

A figura a seguir apresenta este tipo de sinal.



A tensão média de saída pode ser determinada pela seguinte equação.

$$V_{out_{m\u00e9dio}} = \frac{V_{in} * T_{on}}{T}$$

onde:

$V_{out_{m\u00e9dio}}$ é a tensão de saída.

V_{in} é a tensão de entrada no nível alto.

T_{on} é o tempo em que o sinal fica em nível alto

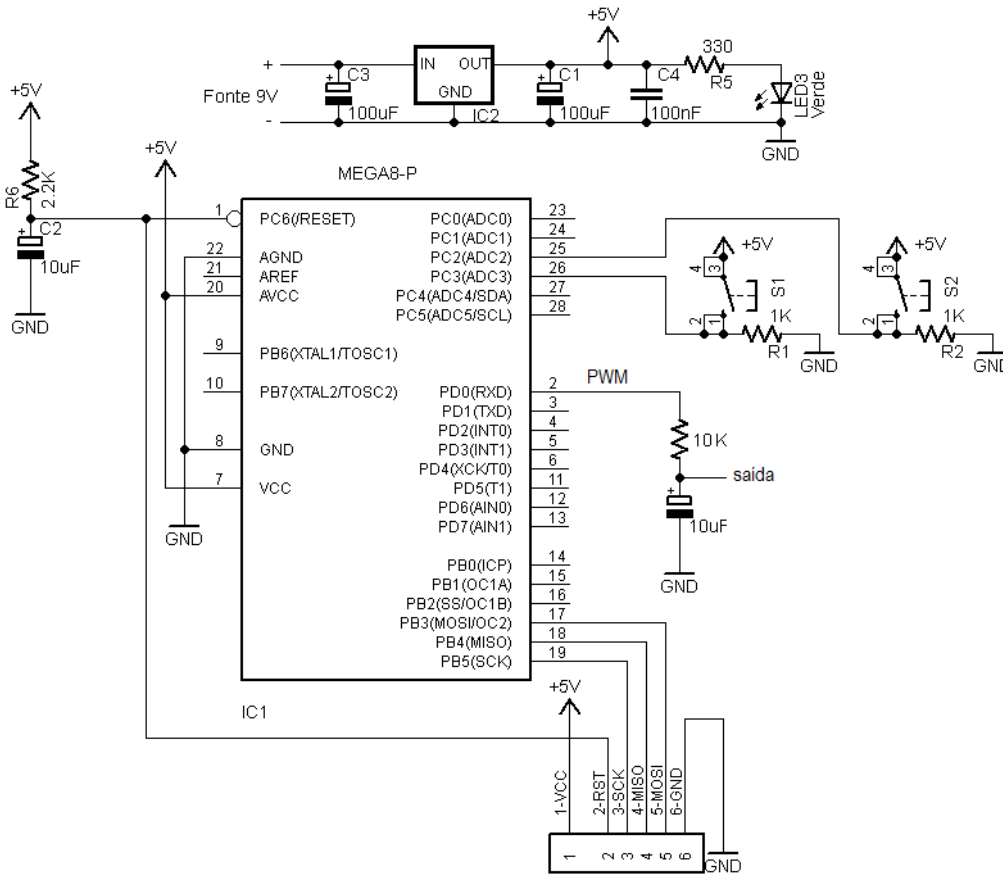
T é o tempo total de cada pulso (per\u00edodo)

Para transformar o sinal PWM em um sinal contínuo com valor igual a tensão média basta somente utilizar um filtro passa baixas com frequência de corte inferior a frequência do sinal PWN.

Utilizando esta técnica é possível gerar sinais analógicos a partir de qualquer saída digital.

3.3 Exercícios

1. Construa o circuito da figura a seguir.



2. Faça um programa para o circuito do exercício anterior que gera uma saída tipo PWM no pino D0. A razão cíclica do PWM deve variar de acordo com a posição das chaves S1 e S2, conforme a tabela a seguir.

S1	S2	Razão Cíclica
0	0	20%
1	0	40%
0	1	60%
1	1	80%

3. Transfira o programa para o microcontrolador do protoboard e comprove seu funcionamento.
4. Elabore para a próxima aula um relatório do experimento realizado, respeitando as normas para elaboração de trabalhos acadêmicos.

AULA 4 - INTERRUPTÕES NO MICROCONTROLADOR

Experimentos envolvendo rotinas de interrupção nos microcontroladores

4.1 Objetivo:

O objetivo desta aula é verificar o funcionamento das interrupções nos microcontroladores da família AVR.

4.2 Fundamentação

A seguir temos um exemplo onde as interrupções externas 0 e 1 são utilizadas.

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

int cont;

ISR(INT0_vect)
{
  cont++;
}

ISR(INT1_vect)
{
  cont--;
}

int main()
{
  MCUCR|=0b00001111; // int0 e int1 na subida do sinal
  GICR|=0b11000000; // ativa int0 e int 1

  sei(); // ativa todas as interrupções

  cont=0;

  while(1)
  {
  }
}
```

Neste exemplo foi utilizado o registrador “MCUCR”, onde são ajustados os níveis de tensão em que a interrupção deve acontecer. E o registrador GICR onde as interrupções são habilitadas. As tabelas a seguir detalham estes registradores.

O registrador MCUCR é o registrador de controle do microcontrolador.

Registrador MCUCR	
Bit	Significado
0	ISC00 - Sensibilidade INT0 bit 0
1	ISC01 - Sensibilidade INT0 bit 1
2	ISC10 - Sensibilidade INT1 bit 0
3	ISC11 - Sensibilidade INT1 bit 1
4	Modo de hibernação bit 0
5	Modo de hibernação bit 1
6	Modo de hibernação bit 2
7	Habilita a hibernação

Neste exemplo foram utilizados os 4 primeiros bits, as tabelas a seguir apresentam o significado dos registradores ISC00 a ISC11.

ISC01	ISC00	Significado
0	0	O nível baixo em INT0 gera a interrupção
0	1	Qualquer mudança em INT0 gera a interrupção
1	0	Uma borda de descida em INT0 gera a interrupção
1	1	Uma borda de subida em INT0 gera a interrupção

ISC11	ISC10	Significado
0	0	O nível baixo em INT1 gera a interrupção
0	1	Qualquer mudança em INT1 gera a interrupção
1	0	Uma borda de descida em INT1 gera a interrupção
1	1	Uma borda de subida em INT1 gera a interrupção

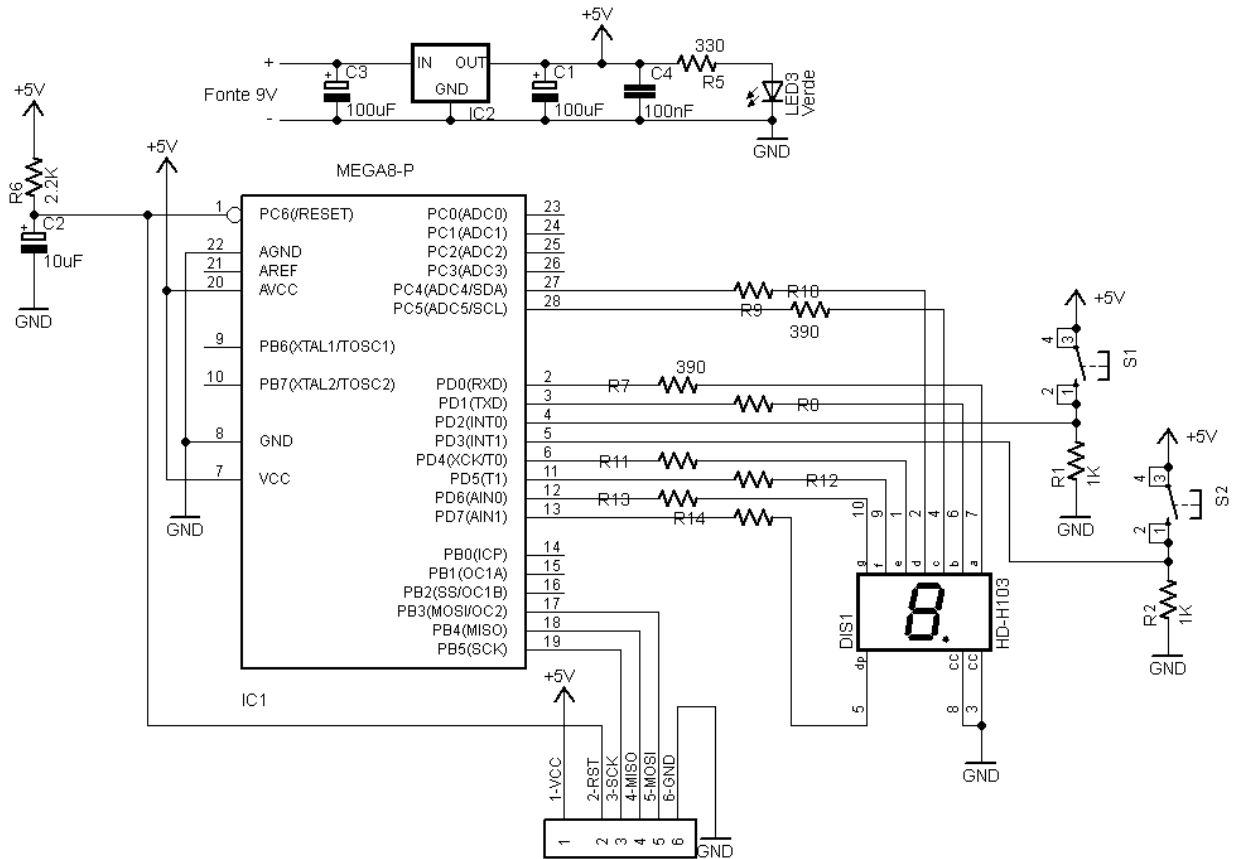
O registrador GICR é o registrador de controle geral das interrupções.

Registrador GICR	
Bit	Significado
0	IVSEL – escolha do vetor de interrupção
1	IVCE – Habilita troca de vetor de interrupção
2	-
3	-
4	-
5	-
6	INT0 - Habilita interrupção externa 0
7	INT1 - Habilita interrupção externa 1

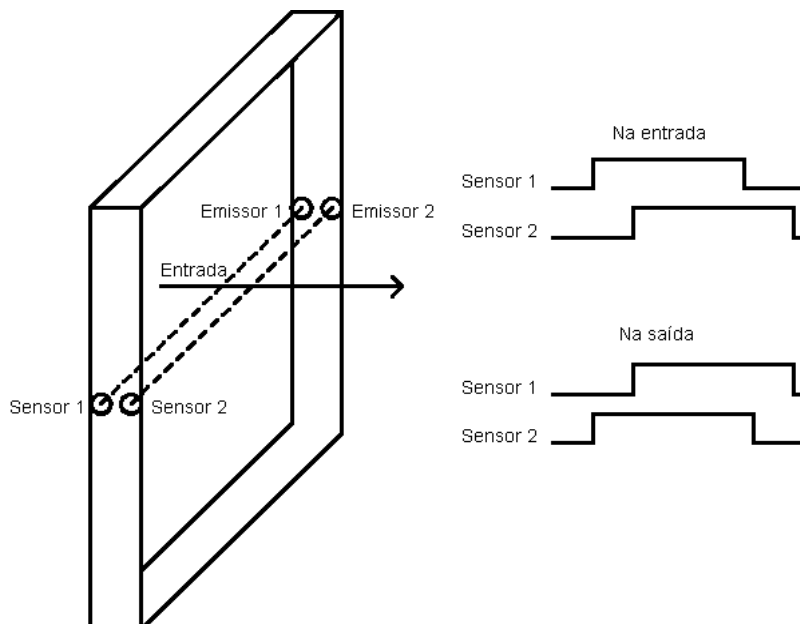
No exemplo foram utilizados os bits 6 e 7, para habilitar as interrupções externas.

4.3 Exercícios

1. Monte o circuito da figura a seguir, observando que as chaves S1 e S2 estão conectadas nas entradas de interrupção INT0 e INT1.



2. Faça um programa para o circuito do exercício anterior supondo que as chaves S1 e S2 são sensores ópticos montados na porta de uma sala. O programa deve apresentar no display o número de pessoas que estão no interior da sala. A figura a seguir mostra a montagem dos sensores na porta e os sinais que eles enviam quando alguma pessoa a atravessa.



3. Transfira o programa para o microcontrolador do protoboard e comprove seu funcionamento.
5. Elabore para a próxima aula um relatório do experimento realizado, respeitando as normas para elaboração de trabalhos acadêmicos.

AULA 5 - TEMPORIZADORES E CONTADORES

Desenvolver experimentos envolvendo temporizadores e contadores

5.1 Objetivo:

O objetivo desta aula é desenvolver experimentos envolvendo temporizadores e contadores, de forma a verificar na prática o funcionamento destes periféricos.

5.2 Temporizador / contador 0

As tabelas a seguir mostram os bits de configuração do temporizador / contador 0, conforme estudado nas aulas teóricas.

O registrador TCCR0.

Registrador TCCR0	
Bit	Significado
0	CS00 – Bit de seleção de clock 0
1	CS01 – Bit de seleção de clock 1
2	CS02 – Bit de seleção de clock 2
3	–
4	–
5	–
6	–
7	–

Descrição dos bits do registrador TCCR0.

CS02	CS01	CS00	Significado
0	0	0	Sem pulsos de contagem (Contador desligado)
0	0	1	Pulsos direto do clock do microcontrolador
0	1	0	Pulsos do clock do microcontrolador dividido por 8
0	1	1	Pulsos do clock do microcontrolador dividido por 64
1	0	0	Pulsos do clock do microcontrolador dividido por 256
1	0	1	Pulsos do clock do microcontrolador dividido por 1024
1	1	0	Pulsos do pino externo T0, na borda de descida
1	1	1	Pulsos do pino externo T0, na borda de subida

Para que o temporizador / contador 0 entre em funcionamento basta gravar um valor diferente de 0 no registrador TCCR0, e para parar sua contagem basta gravar 0.

O registrador TIMSK.

Registrador TIMSK	
Bit	Significado
0	TOIE0 – Interrupção de estouro do temporizador 0
1	-
2	TOIE1 – Interrupção de estouro do temporizador 1
3	OCIE1B – Interrupção do comparador B do temporizador 1
4	OCIE1A – Interrupção do comparador A do temporizador 1
5	TICIE1 – Interrupção de captura externa do temporizador 1
6	TOIE2 – Interrupção de estouro do temporizador 2
7	OCIE2 – Interrupção do comparador do temporizador 2

Quando utilizamos o temporizador / contador 0 no modo de contagem a formula para o valor de recarga é:

$$TCNT0 = 256 - N_{Pulsos}$$

Já quando usamos o temporizador / contador 0 no modo de temporização a formula para o valor de recarga é:

$$TCNT0 = 256 - \frac{T_{segundos} \times clock}{prescaler}$$

Onde:

T = tempo em segundos desejado

clock = frequencia de operação do microcontroladores

prescaler = valor do divisor de clock

5.3 Exercícios

1. Faça um programa utilizando o temporizador / contador 0 que conta 15 pulsos na entrada T0 e quando termina acende um LED na porta PC0.
2. Monte um circuito no protoboard e transfira o programa para o microcontrolador, comprovando seu funcionamento.
3. Faça um programa utilizando o contador / temporizador 0 que pisca um led na porta C0 em uma frequência de 3 Hz.
4. Monte um circuito no protoboard e transfira o programa para o microcontrolador, comprovando seu funcionamento.
5. Mude o programa para que o led pisque em 1 Hz e repita o teste no protoboard.
6. Utilizando um display de 7 segmentos altere o software do exercício anterior para que o número mostrado no display seja incrementado de 1 em 1 segundos (de 0 a 9). Comprove seu funcionamento no protoboard.
7. Elabore para a próxima aula um relatório dos experimentos realizados, respeitando as normas para elaboração de trabalhos acadêmicos.

AULA 6 - TEMPORIZADORES AVANÇADOS

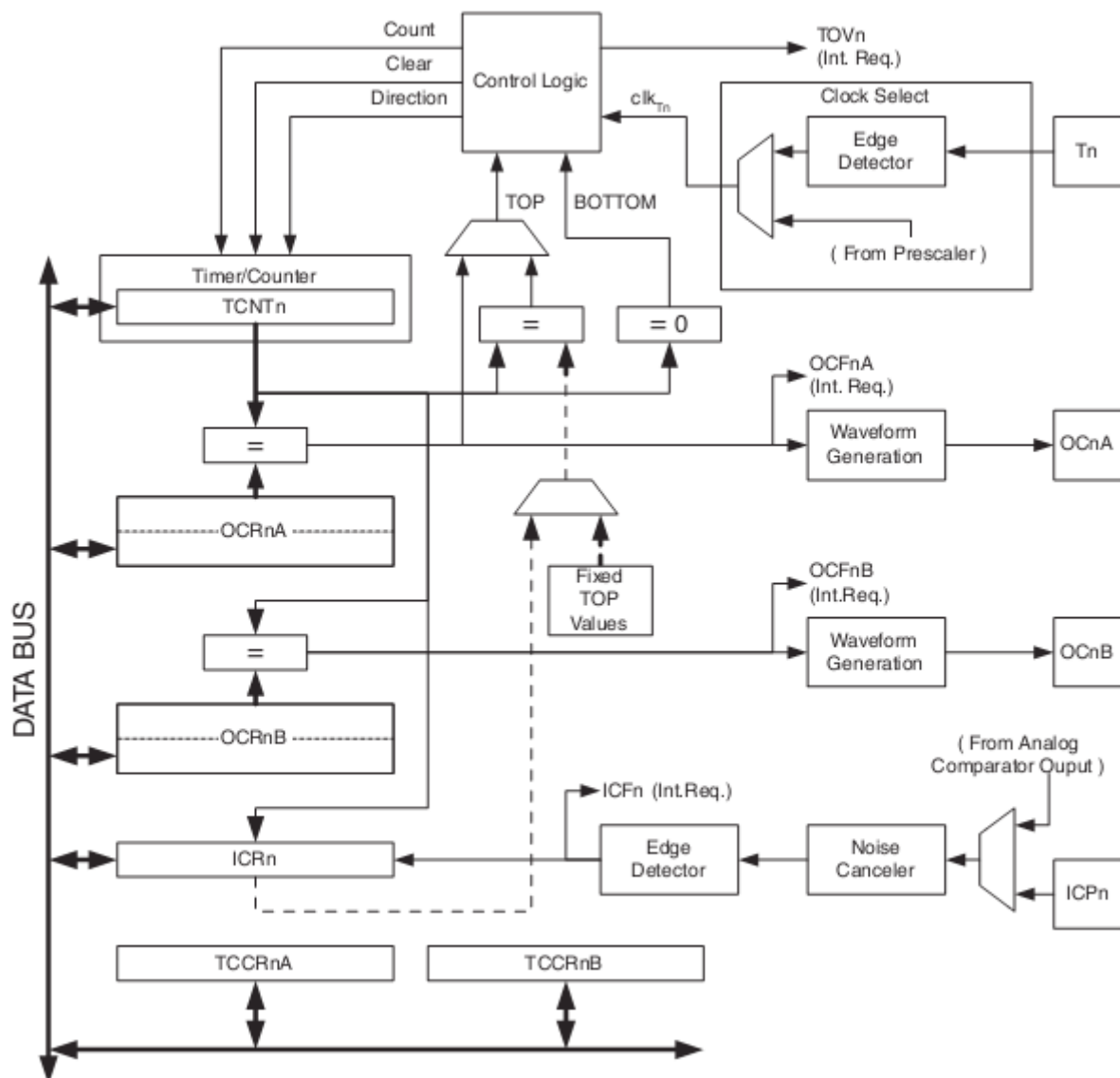
Experimentos com temporizadores e contadores com funções avançadas

6.1 Objetivo:

O objetivo desta aula é utilizar as funções avançadas dos contadores e temporizadores instalados no interior dos microcontroladores da família AVR da Atmel.

6.2 Fundamentação

O temporizador / contador 1 possui diversos componentes de hardware que possibilitam que ele funcione de várias formas diferentes. A figura a seguir mostra um diagrama de blocos deste temporizador / contador.



Como este temporizador / contador possui duas unidades de comparação é possível que ele seja configurado para gerar vários tipos de forma de onda, inclusive sinais PWM. O temporizador / contador 1 também está conectado aos pinos de saída, o que permite que os sinais por ele gerados

sejam enviados diretamente aos pinos do microcontrolador.

A tabela a seguir mostra os modos de operação do temporizador / contador 1.

Modo	WGM13	WGM12	WGM11	WGM10	Modo de operação	Topo	Atualização de OCR1x	Liga o bit TOV1
0	0	0	0	0	Normal	0xFFFF	Imediato	Máximo
1	0	0	0	1	PWM fase correta 8 bits	0x00FF	Topo	Base
2	0	0	1	0	PWM fase correta 9 bits	0x01FF	Topo	Base
3	0	0	1	1	PWM fase correta 10 bits	0x03FF	Topo	Base
4	0	1	0	0	CTC	OCR1A	Imediato	Máximo
5	0	1	0	1	PWM rápido 8 bits	0x00FF	Base	Topo
6	0	1	1	0	PWM rápido 9 bits	0x01FF	Base	Topo
7	0	1	1	1	PWM rápido 10 bits	0x03FF	Base	Topo
8	1	0	0	0	PWM fase e frequência correta	ICR1	Base	Base
9	1	0	0	1	PWM fase e frequência correta	OCR1A	Base	Base
10	1	0	1	0	PWM fase correta	ICR1	Topo	Base
11	1	0	1	1	PWM fase correta	OCR1A	Topo	Base
12	1	1	0	0	CTC	ICR1	Imediato	Máximo
13	1	1	0	1	(Reservado)	-	-	-
14	1	1	1	0	PWM rápido	ICR1	Base	Topo
15	1	1	1	1	PWM rápido	OCR1A	Base	Topo

Para escolher o modo de operação do temporizador / contador 1 é necessário configurarmos um conjunto de registradores, conforme estudado nas aulas teóricas.

6.3 Exercícios

1. Monte um circuito utilizando o microcontrolador Atmega8 e todos os periféricos necessários ao seu funcionamento. Conecte a este microcontrolador dois botões, um LED e um display de 7 segmentos. O LED deve estar conectado a saída PWM do temporizador / contador 1.
2. Faça um programa para o circuito da questão anterior de forma que seja possível ajustar o brilho do LED através do ajuste do sinal PWM. O PWM deve ser gerado pelo temporizador / contador 1 e deve possuir 10 níveis, indicados no display com os números de 0 a 9, onde 0 é LED desligado e 9 é LED completamente ligado. Os dois botões devem permitir alterar os níveis de brilho do LED. A frequência do PWM deve ser de 500 Hz.
3. Transfira o programa para o microcontrolador do protoboard e comprove seu funcionamento.
4. Visualize a forma de onda do sinal PWM no osciloscópio e verifique se a frequência é de 500 Hz.
5. Elabore para a próxima aula um relatório do experimento realizado, respeitando as normas para elaboração de trabalhos acadêmicos.

AULA 7 - CONVERSÃO ANALÓGICA DIGITAL

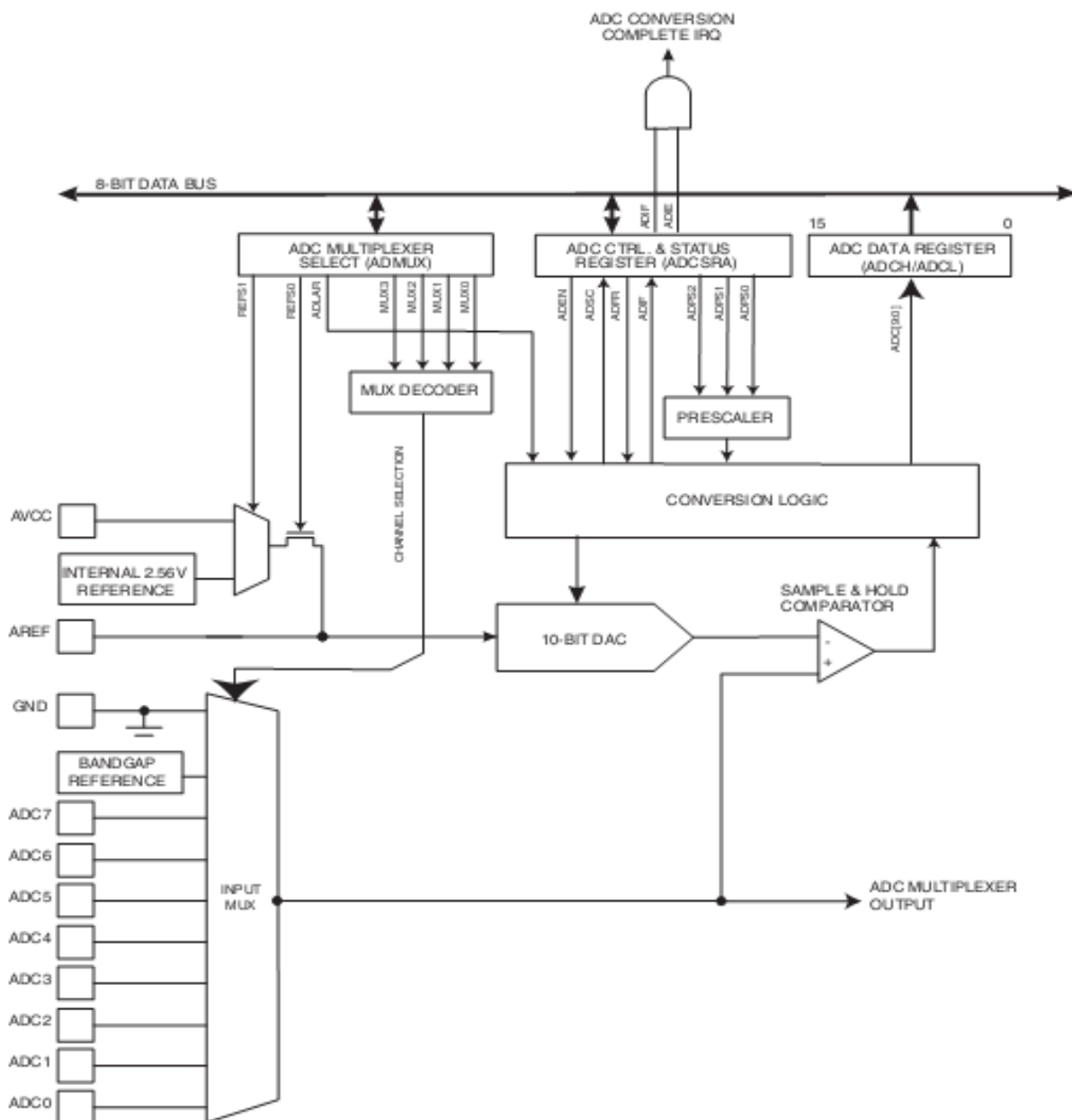
Estudo do conversor analógico digital interno do microcontrolador AVR

7.1 Objetivo:

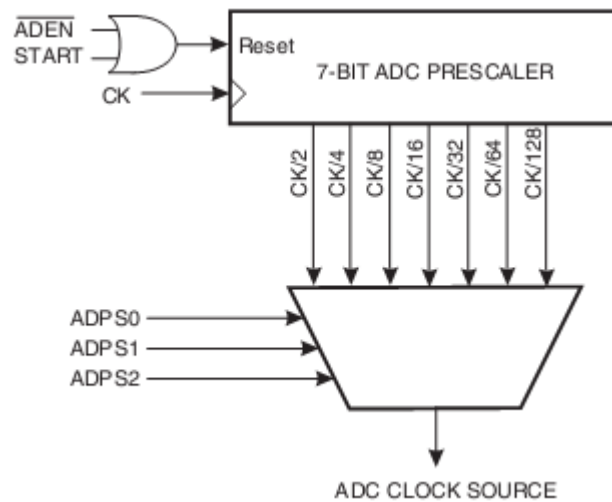
O objetivo desta aula é verificar experimentalmente o funcionamento do conversor analógico digital presente nos microcontroladores da família AVR da Atmel.

7.2 Fundamentação

7.2.1 O hardware do conversor analógico digital



7.2.2 Prescaler do conversor A/D



É importante salientar que em resolução máxima (10 bits), o conversor A/D não pode operar em uma frequência superior a 200KHz.

7.2.3 Resultados das conversões

$$ADC = \frac{V_{entrada} \times 1024}{V_{ref}}$$

No programa o resultado da conversão pode ser acessado através do registrador ADCW.

7.2.4 Registradores de controle do conversor A/D

As tabelas a seguir apresentam as funções dos bits do registrador ADCMUX

Registrador ADCMUX	
Bit	Significado
0	MUX0 – bit 0 do multiplexador
1	MUX1 – bit 1 do multiplexador
2	MUX2 – bit 2 do multiplexador
3	MUX3 – bit 3 do multiplexador
4	–
5	ADLAR – Ajuste da saída do A/D
6	REFS0 – bit 0 da seleção de referência
7	REFS1 – bit 1 da seleção de referência

Os primeiros 4 bits são responsáveis por selecionar a entrada analógica, a tabela a seguir detalha seu funcionamento.

MUX3	MUX2	MUX1	MUX0	Significado
0	0	0	0	ADC0
0	0	0	1	ADC1
0	0	1	0	ADC2
0	0	1	1	ADC3
0	1	0	0	ADC4
0	1	0	1	ADC5
0	1	1	0	ADC6
0	1	1	1	ADC7
1	0	0	0	-
1	0	0	1	-
1	0	1	0	-
1	0	1	1	-
1	1	0	0	-
1	1	0	1	-
1	1	1	0	1.3V interno
1	1	1	1	0V interno

O bit 5 diz respeito a forma que o resultado é armazenado, e se for necessário o manual deve ser consultado. Já os bits 6 e 7 são responsáveis pela escolha da entrada referencia do conversor A/D. A tabela a seguir mostra o significado de cada bit.

REFS1	REFS0	Significado
0	0	Referência externa no pino AREF
0	1	Referencia externa no pino AVCC (Utilizar filtro em AREF)
1	0	-
1	1	Referência interna de 2.56 (Utilizar filtro em AREF)

Outro registrador envolvido no controle do conversor A/D é o registrador ADCSRA. A tabela a seguir mostra o significado de cada um dos seus bits.

Registrador ADCSRA	
Bit	Significado
0	ADPS0 – bit 0 do prescaler do A/D
1	ADPS1 – bit 1 do prescaler do A/D
2	ADPS2 – bit 2 do prescaler do A/D
3	ADIE – habilita a interrupção do A/D
4	ADCIF – Indicador de interrupção
5	ADFR – Modo de conversão
6	ADSC – Inicia a conversão
7	ADEN – habilita o conversor A/D

Os bits 0, 1 e 2 são responsáveis pela escolha do prescaler, conforma tabela a seguir.

ADPS2	ADPS1	ADPS0	Significado
0	0	0	Divide o clock por 2
0	0	1	Divide o clock por 2
0	1	0	Divide o clock por 4
0	1	1	Divide o clock por 8
1	0	0	Divide o clock por 16
1	0	1	Divide o clock por 32
1	1	0	Divide o clock por 64
1	1	1	Divide o clock por 128

7.3 Exercícios

1. Monte um circuito utilizando o microcontrolador Atmega8 e todos os periféricos necessários ao seu funcionamento. Conecte na entrada analógica 2 deste microcontrolador um potenciômetro, de forma a enviar um sinal de 0 a 5 V e no pino D0, conecte um LED. O LED no pino D0 irá simbolizar uma bomba e o potenciômetro irá simbolizar um sensor de nível.
2. Suponha que em um tanque de 1 metro de altura tenhamos instalado um sensor de nível que envia um sinal de 0 a 5V. 0V para nível em 0cm e 5V par nível de 1m. Suponha também que no pino D0 é instalado um relê que aciona a bomba que enche este tanque. Para este tanque faça um programa que controla o nível do tanque entre 70 e 80cm, uma vez que o sensor esta conectado a entrada analógica 2.
3. Transfira o programa para o microcontrolador do protoboard e comprove seu funcionamento.
4. Agora instale no protoboard um display de 7 segmentos de forma a mostrar os números de 0 a 9.
5. Faça um programa que lê o valor da entrada analógica 0 e mostra no display os números de 0 a 9, proporcionalmente a tensão de entrada. O display deve mostrar 0 para 0 V e 9 para 5V.
6. Transfira o programa para o microcontrolador do protoboard e comprove seu funcionamento.
7. Elabore para a próxima aula um relatório do experimento realizado, respeitando as normas para elaboração de trabalhos acadêmicos.

AULA 8 - COMUNICAÇÃO SERIAL NA FAMÍLIA AVR

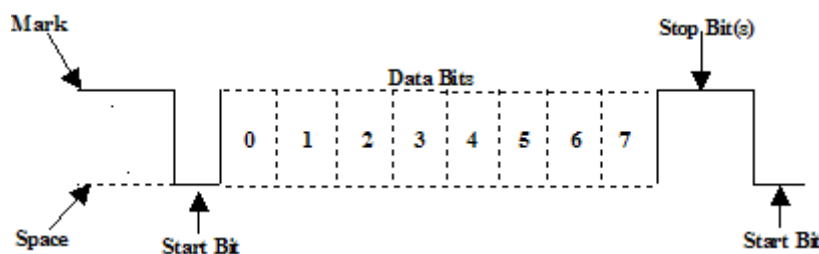
Comunicação serial nos microcontroladores da família AVR

8.1 Objetivo:

O objetivo desta aula é demonstrar experimentalmente aos alunos o que é e como funciona a comunicação serial nos microcontroladores da família AVR.

8.2 Fundamentação

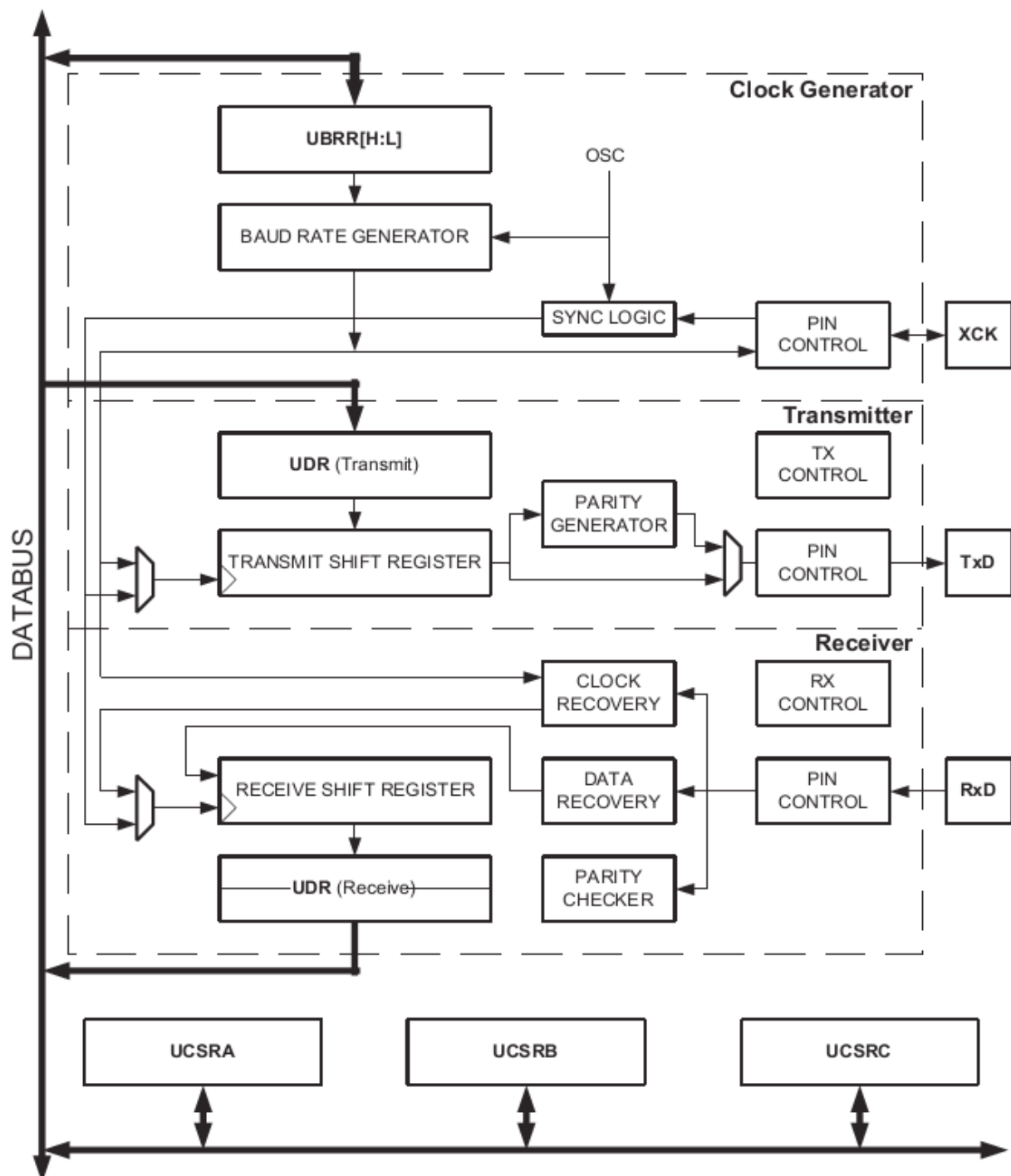
Na comunicação serial as informações são transmitidas digitalmente um bit de cada vez. A comunicação serial pode ser síncrona, onde existe um sinal de clock ou assíncrona, onde não é necessário o sinal de clock. Na automação e controle de processos industriais a comunicação serial assíncrona é a mais utilizada. A figura a seguir mostra a forma de onda de um pacote de dados no formato serial assíncrono.



A família de microcontroladores AVR possui uma unidade de comunicação serial chamada USART. Esta USART possui as seguintes características.

- Transmissão e recepção simultânea de dados (Full Duplex).
- Operação assíncrona ou síncrona.
- Opera como mestre ou escravo em uma rede síncrona.
- Gerador de taxa de transferência de alta resolução.
- Suporta pacotes de dados de 5, 6, 7, 8 ou 9 bits com 1 ou 2 bits de parada.
- Gerador e verificador de paridade em hardware, com paridade par e ímpar.
- Detecta sobrescrita de dados.
- Detecta erro de pacote.
- Diversos filtros anti ruídos.
- Três fontes de interrupção, transmissão completa, registrador de transmissão vazio e recepção completa.
- Modo de comunicação multi-processador.
- Modo de comunicação “Double Speed” para comunicações assíncronas de alta velocidade.

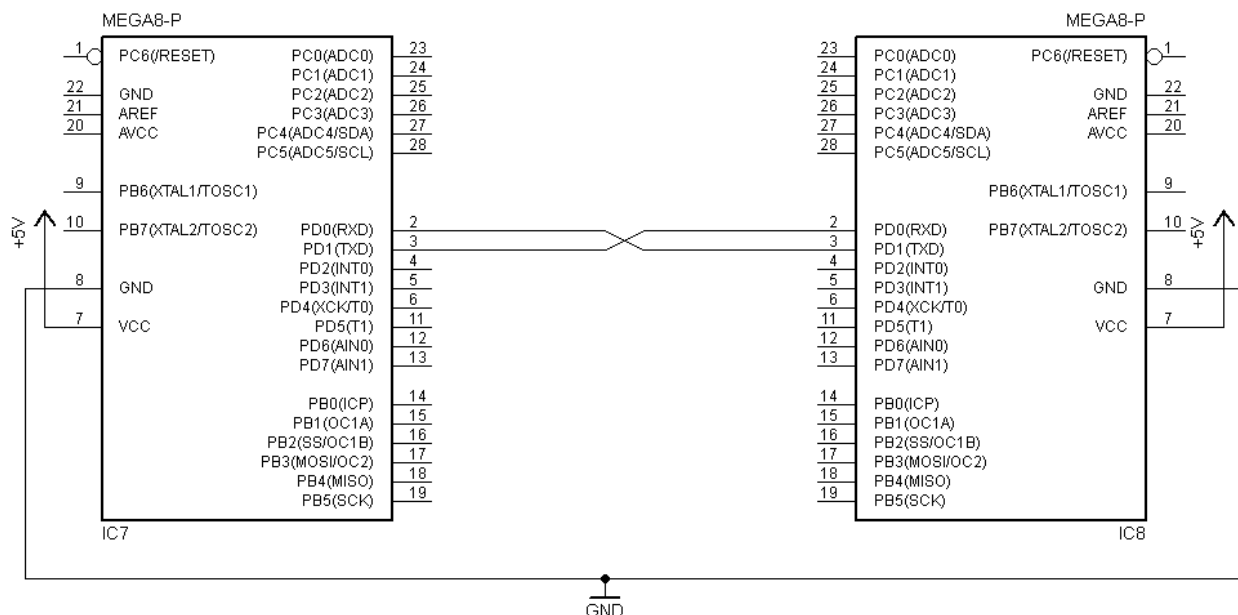
A figura a seguir mostra um diagrama de blocos do hardware da USART.



Como se trata de um hardware bastante complexo, não é o objetivo deste material detalhar todo seu funcionamento. Assim para maiores informações sobre as propriedades e configurações da USART o datasheet do microcontrolador deve ser consultado.

8.3 Exercícios

1. Monte um circuito utilizando o microcontrolador Atmega8 e todos os periféricos necessários ao seu funcionamento. Conecte a este microcontrolador dois botões e dois LEDs. Interligue os pinos de transmissão e recepção da porta serial (pinos 2 e 3) de forma que o microcontrolador receba o sinal que ele mesmo esta enviando pela serial.
2. Para o circuito do exercício anterior faça um programa que utiliza a porta serial no modo assíncrono. A comunicação serial deve ser configurada para transmitir 1200 bps, com 8 bits por pacote, sem paridade e com 2 stop bits. Quando o primeiro botão for pressionado o microcontrolador deve enviar o valor 10 e quando o segundo botão for pressionado o microcontrolador deve enviar o valor 20. Quando o microcontrolador receber o valor 10 na entrada serial ele deve ligar o LED 1, qualquer outro valor desliga este LED. Quando o microcontrolador receber o valor 20 na entrada serial ele deve ligar o LED 2, qualquer outro valor desliga este LED.
3. Transfira o programa para o microcontrolador do protoboard e comprove seu funcionamento.
4. Agora combine com outra equipe e conecte os dois circuito conforme a figura a seguir, assim uma equipe pode enviar comandos para o circuito da outra comprovando o funcionamento da comunicação serial.



5. Elabore para a próxima aula um relatório do experimento realizado, respeitando as normas para elaboração de trabalhos acadêmicos.

AULA 9 - LADDER NOS MICROCONTROLADORES

Programação de microcontroladores AVR em linguagem Ladder

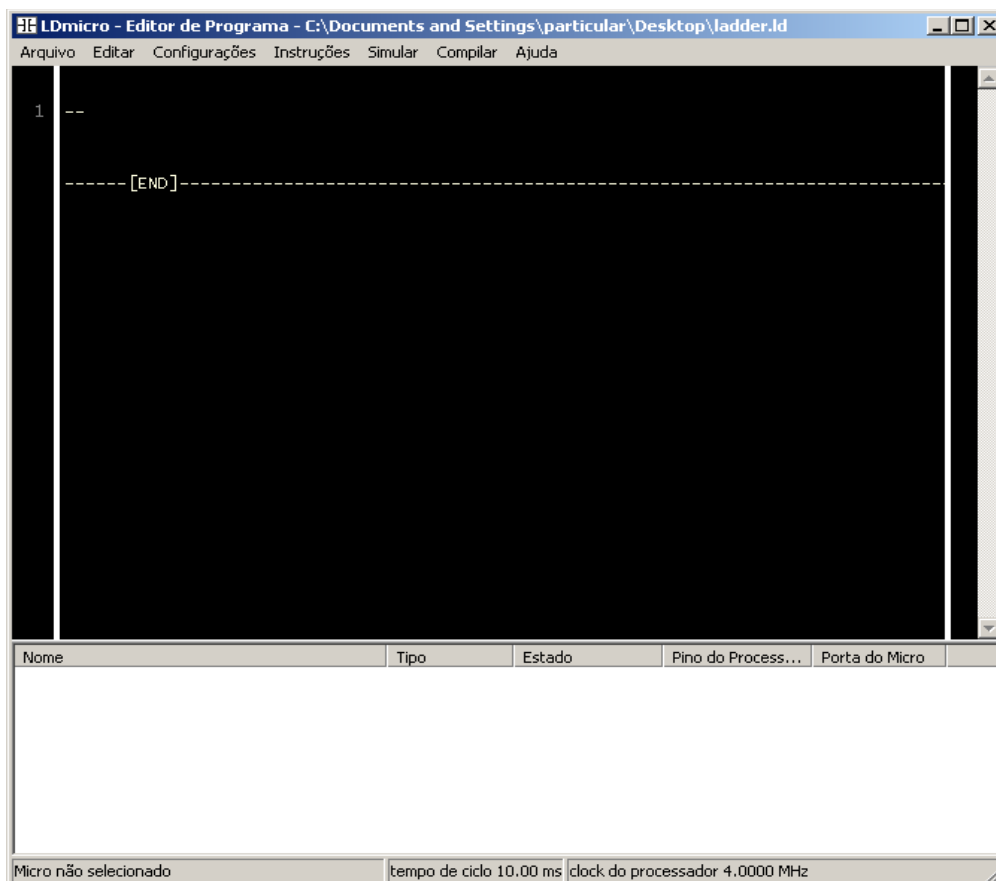
9.1 Objetivo:

O objetivo desta aula é demonstrar experimentalmente aos alunos o uso da linguagem Ladder na programação dos microcontroladores.

Utilizaremos o compilador LDmicro, que converte a linguagem ladder para a linguagem de máquina dos microcontroladores AVR.

9.2 Fundamentação

O programa LDmicro, que pode ser obtido no site: <http://www.cq.cx/ladder.pl> A versão em português se chama “ldmicro-pt.exe”.



9.3 Exercícios

1. Faça um programa em linguagem ladder, utilizando o software Ldmicro. O programa deve controlar o nível de líquido em um tanque. O sensor de nível envia um sinal analógico ao conversor A/D 0 do microcontrolador, onde após convertido o sinal é 0 para 0 metros de líquido e 1000 para 10 metros de líquido. O controle deve acionar a bomba conectada ao pino PD0 quando o nível for inferior a 8 metro e desligar quando for superior a 9 metros.
2. Simule o programa do exercício anterior, utilizando o simulador do Ldmicro.
3. Monte um circuito utilizando o microcontrolador Atmega8 e todos os periféricos necessários ao seu funcionamento, de forma a ser possível simular o programa do exercício anterior.
4. Transfira o programa para o microcontrolador do protoboard e comprove seu funcionamento.
5. Elabore para a próxima aula um relatório do experimento realizado, respeitando as normas para elaboração de trabalhos acadêmicos.

ANEXO I – CONJUNTO DE INSTRUÇÕES

A linguagem assembler não diferencia maiúsculas de minúsculas, os operandos tem a seguinte forma.

Rd: R0-R31 ou R16-R31 (dependendo da instrução)

Rr: R0-R31

b: Constante (0-7), pode ser uma expressão constante

s: Constante (0-7), pode ser uma expressão constante

P: Constante (0-31/63), pode ser uma expressão constante

K: Constante (0-255), pode ser uma expressão constante

k: Constante, A faixa de valores depende da instrução. pode ser uma expressão constante

q: Constante (0-63), pode ser uma expressão constante

Instrução	Op.	Descrição	Operação	Flags	Clocks
ARITHMETIC					
ADD	Rd, Rr	Add two Registers	$Rd \leftarrow Rd + Rr$	Z, C, N, V, H	1
ADC	Rd, Rr	Add with Carry two Registers	$Rd \leftarrow Rd + Rr + C$	Z, C, N, V, H	1
ADIW	Rdl,K	Add Immediate to Word	$Rdh:Rdl \leftarrow Rdh:Rdl + K$	Z, C, N, V, S	2
SUB	Rd, Rr	Subtract two Registers	$Rd \leftarrow Rd - Rr$	Z, C, N, V, H	1
SUBI	Rd, K	Subtract Constant from Register	$Rd \leftarrow Rd - K$	Z, C, N, V, H	1
SBC	Rd, Rr	Subtract with Carry two Registers	$Rd \leftarrow Rd - Rr - C$	Z, C, N, V, H	1
SBCI	Rd, K	Subt. with Carry Const. from Reg	$Rd \leftarrow Rd - K - C$	Z, C, N, V, H	1
SBIW	Rdl,K	Subtract Immediate from Word	$Rdh:Rdl \leftarrow Rdh:Rdl - K$	Z, C, N, V, S	2
AND	Rd, Rr	Logical AND Registers	$Rd \leftarrow Rd \cdot Rr$	Z, N, V	1
ANDI	Rd, K	Logical AND Register and Const.	$Rd \leftarrow Rd \cdot K$	Z, N, V	1
OR	Rd, Rr	Logical OR Registers	$Rd \leftarrow Rd \vee Rr$	Z, N, V	1
ORI	Rd, K	Logical OR Register and Constant	$Rd \leftarrow Rd \vee K$	Z, N, V	1
EOR	Rd, Rr	Exclusive OR Registers	$Rd \leftarrow Rd \oplus Rr$	Z, N, V	1
COM	Rd	One's Complement	$Rd \leftarrow 0xFF - Rd$	Z, C, N, V	1
NEG	Rd	Two's Complement	$Rd \leftarrow 0x00 - Rd$	Z, C, N, V, H	1
SBR	Rd,K	Set Bit(s) in Register	$Rd \leftarrow Rd \vee K$	Z, N, V	1
CBR	Rd,K	Clear Bit(s) in Register	$Rd \leftarrow Rd \cdot (0xFF - K)$	Z, N, V	1
INC	Rd	Increment	$Rd \leftarrow Rd + 1$	Z, N, V	1
DEC	Rd	Decrement	$Rd \leftarrow Rd - 1$	Z, N, V	1
TST	Rd	Test for Zero or Minus	$Rd \leftarrow Rd \cdot Rd$	Z, N, V	1
CLR	Rd	Clear Register	$Rd \leftarrow Rd \oplus Rd$	Z, N, V	1
SER	Rd	Set Register	$Rd \leftarrow 0xFF$	None	1
MUL	Rd, Rr	Multiply Unsigned	$R1:R0 \leftarrow Rd \times Rr$	Z, C	2
MULS	Rd, Rr	Multiply Signed	$R1:R0 \leftarrow Rd \times Rr$	Z, C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	$R1:R0 \leftarrow Rd \times Rr$	Z, C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	$R1:R0 \leftarrow (Rd \times Rr) \ll 1$	Z, C	2
FMULS	Rd, Rr	Fractional Multiply Signed	$R1:R0 \leftarrow (Rd \times Rr) \ll 1$	Z, C	2
FMULSU	Rd, Rr	Fract. Mult. Signed - Unsigned	$R1:R0 \leftarrow (Rd \times Rr) \ll 1$	Z, C	2
BRANCH INSTRUCTIONS					

RJMP	k	Relative Jump	$PC \leftarrow PC + k + 1$	None	2
IJMP		Indirect Jump to (Z)	$PC \leftarrow Z$	None	2
RCALL	k	Relative Subroutine Call	$PC \leftarrow PC + k + 1$	None	3
ICALL		Indirect Call to (Z)	$PC \leftarrow Z$	ZNone	3
RET		Subroutine Return	$PC \leftarrow STACK$	None	4
RETI		Interrupt Return	$PC \leftarrow STACK$	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) $PC \leftarrow PC + 2$ or 3	None	1 2 3
CP	Rd,Rr	Compare	Rd - Rr	Z, N, V, C, H	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z, N, V, C, H	1
CPI	Rd,K	Compare Register with Immediate	Rd - K	Z, N, V, C, H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) $PC \leftarrow PC + 2$ or 3	None	1 2 3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) $PC \leftarrow PC + 2$ or 3	None	1 2 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) $PC \leftarrow PC + 2$ or 3	None	1 2 3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) $PC \leftarrow PC + 2$ or 3	None	1 2 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s)=1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s)=0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N \oplus V=0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N \oplus V=1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRVC	k	Branch if Overf. Flag is Cleared	if (V = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then $PC \leftarrow PC + k + 1$	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then $PC \leftarrow PC + k + 1$	None	1 / 2
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	$Rd \leftarrow Rr$	None	1
MOVW	Rd, Rr	Copy Register Word	$Rd+1:Rd \leftarrow Rr+1:Rr$	None	1
LDI	Rd, K	Load Immediate	$Rd \leftarrow K$	None	1
LD	Rd, X	Load Indirect	$Rd \leftarrow (X)$	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	$Rd \leftarrow (X), X \leftarrow X + 1$	None	2
LD	Rd, -X	Load Indirect and Pre-Dec.	$X \leftarrow X - 1, Rd \leftarrow (X)$	None	2
LD	Rd, Y	Load Indirect	$Rd \leftarrow (Y)$	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	$Rd \leftarrow (Y), Y \leftarrow Y + 1$	None	2
LD	Rd, -Y	Load Indirect and Pre-Dec.	$Y \leftarrow Y - 1, Rd \leftarrow (Y)$	None	2

LDD	Rd, Y+q	Load Indirect with Displacement	$Rd \leftarrow (Y + q)$	None	2
LD	Rd, Z	Load Indirect	$Rd \leftarrow (Z)$	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	$Rd \leftarrow (Z), Z \leftarrow Z+1$	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	$Z \leftarrow Z - 1, Rd \leftarrow (Z)$	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	$Rd \leftarrow (Z + q)$	None	2
LDS	Rd, k	Load Direct from SRAM	$Rd \leftarrow (k)$	None	2
ST	X, Rr	Store Indirect	$(X) \leftarrow Rr$	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	$(X) \leftarrow Rr, X \leftarrow X + 1$	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	$X \leftarrow X - 1, (X) \leftarrow Rr$	None	2
ST	Y, Rr	Store Indirect	$(Y) \leftarrow Rr$	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	$(Y) \leftarrow Rr, Y \leftarrow Y + 1$	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	$Y \leftarrow Y - 1, (Y) \leftarrow Rr$	None	2
STD	Y+q, Rr	Store Indirect with Displacement	$(Y + q) \leftarrow Rr$	None	2
ST	Z, Rr	Store Indirect	$(Z) \leftarrow Rr$	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	$(Z) \leftarrow Rr, Z \leftarrow Z + 1$	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	$Z \leftarrow Z - 1, (Z) \leftarrow Rr$	None	2
STD	Z+q, Rr	Store Indirect with Displacement	$(Z + q) \leftarrow Rr$	None	2
STS	k, Rr	Store Direct to SRAM	$(k) \leftarrow Rr$	None	2
LPM		Load Program Memory	$R0 \leftarrow (Z)$	None	3
LPM	Rd, Z	Load Program Memory	$Rd \leftarrow (Z)$	None	3
LPM	Rd, Z+	Load Program Mem. and Post-Inc	$Rd \leftarrow (Z), Z \leftarrow Z+1$	None	3
SPM		Store Program Memory	$(Z) \leftarrow R1:R0$	None	-
IN	Rd, P	In Port	$Rd \leftarrow P$	None	1
OUT	P, Rr	Out Port	$P \leftarrow Rr$	None	1
PUSH	Rr	Push Register on Stack	$STACK \leftarrow Rr$	None	2
POP	Rd	Pop Register from Stack	$Rd \leftarrow STACK$	None	2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	$I/O(P,b) \leftarrow 1$	None	2
CBI	P,b	Clear Bit in I/O Register	$I/O(P,b) \leftarrow 0$	None	2
LSL	Rd	Logical Shift Left	$Rd(n+1) \leftarrow Rd(n), Rd(0) \leftarrow 0$	Z, C, N, V	1
LSR	Rd	Logical Shift Right	$Rd(n) \leftarrow Rd(n+1), Rd(7) \leftarrow 0$	Z, C, N, V	1
ROL	Rd	Rotate Left Through Carry	$Rd(0) \leftarrow C, Rd(n+1) \leftarrow (n), C \leftarrow Rd(7)$	Z, C, N, V	1
ROR	Rd	Rotate Right Through Carry	$Rd(7) \leftarrow C, Rd(n) \leftarrow (n+1), C \leftarrow Rd(0)$	Z, C, N, V	1
ASR	Rd	Arithmetic Shift Right	$Rd(n) \leftarrow Rd(n+1), n=0..6$	Z, C, N, V	1
SWAP	Rd	Swap Nibbles	$Rd(3..0) \leftarrow (7..4), Rd(7..4) \leftarrow (3..0)$	None	1
BSET	s	Flag Set	$SREG(s) \leftarrow 1$	SREG(s)	1
BCLR	s	Flag Clear	$SREG(s) \leftarrow 0$	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	$T \leftarrow Rr(b)$	T	1
BLD	Rd, b	Bit load from T to Register	$Rd(b) \leftarrow T$	None	1
SEC		Set Carry	$C \leftarrow 1$	C	1
CLC		Clear Carry	$C \leftarrow 0$	C	1
SEN		Set Negative Flag	$N \leftarrow 1$	N	1
CLN		Clear Negative Flag	$N \leftarrow 0$	N	1
SEZ		Set Zero Flag	$Z \leftarrow 1$	Z	1

CLZ		Clear Zero Flag	$Z \leftarrow 0$	Z	1
SEI		Global Interrupt Enable	$I \leftarrow 1$	I	1
CLI		Global Interrupt Disable	$I \leftarrow 0$	I	1
SES		Set Signed Test Flag	$S \leftarrow 1$	S	1
CLS		Clear Signed Test Flag	$S \leftarrow 0$	S	1
SEV		Set Twos Complement Overf	$V \leftarrow 1$	V	1
CLV		Clear Twos Complement Ove	$V \leftarrow 0$	V	1
SET		Set T in SREG	$T \leftarrow 1$	T	1