

INTRODUÇÃO AOS MICROCONTROLADORES

SEGUNDA EDIÇÃO - 2007

Marco Valério Miorim Villaça

Centro Federal de Educação Tecnológica – SC

mvillaca@cefetsc.edu.br



Centro Federal de Educação Tecnológica – SC

Departamento Acadêmico de Eletrônica

Curso Superior de Tecnologia em Sistemas Digitais

Copyright © 2007 para
Gerência Educacional de Eletrônica
Av. Mauro Ramos, 950
Florianópolis, SC – CEP 88020-301

Todos os direitos reservados. É proibida a duplicação ou reprodução deste volume, no todo ou em parte, sob quaisquer formas ou por quaisquer meios, sem permissão expressa da Gerência Educacional de Eletrônica

SUMÁRIO

I - CONCEITOS BÁSICOS	1
1.1 – Memórias : termos e conceitos importantes	1
1.2 – Tipos de Memória	3
1.2.1 - RAM	3
1.2.2 - ROM	4
1.2.3 - Híbridas	5
1.3 - Operação em memória	8
1.3.1 - Entradas de Endereço	8
1.3.2 - Entrada de leitura (R) e escrita (\overline{W})	10
1.3.3 - Seleção do dispositivo	10
1.4 – Fluxogramas	10
1.5 – Representação de números em microcontroladores	13
1.5.1 – Representação de números inteiros	13
1.5.2 – Representação e complemento de 2	16
1.5.3 - Subtração usando adição	17
1.5.4 - A representação BCD	18
1.5.5 - O Código ASCII	21
II - SISTEMAS COM MICROPROCESSADOR	24
2.1 – Descrição:	24
2.2 – Arquitetura Von-Neumann x Harvard :	27
2.3 – Microcontroladores:	28
III - O MICROCONTROLADOR AT89S8252	32
3.1 – Introdução:	32
3.2 – Oscilador Interno:	37
3.3 – Reset:	38
3.4 – Organização de Memória:	39
3.5 – Estrutura e operação da portas de E/S:	52
3.6 – O Conjunto de Instruções dos Microcontroladores MCS-51™ compatíveis.....	60
3.6.1 – A Palavra de Estado do Programa (<i>Program Status Word -PSW</i>)	60
3.6.2 – Modos de endereçamento	62
3.6.3 - Comentário sobre algumas instruções:	64
3.7 – Interrupções:.....	83
3.8 – Temporizadores/contadores:.....	91
3.8.1 – Os <i>Timers</i> 0 e 1 e seus modos de Modos de Operação	92
3.8.2 – O Timer 2 e seus modos de Modos de Operação	105
3.8.3 – O Timer Cão de Guarda Programável (WDT)	112
3.9 – Interface Serial	113
3.9.1 – Modo 0 – Modo Síncrono	114
3.9.2 – Modo 1 – Modo Assíncrono de 10 bits e taxa variável	114
3.9.3 – Modo 2 – Modo Assíncrono de 11 bits e taxa programável	115
3.9.4 – Modo 3 – Modo Assíncrono de 11 bits e taxa variável	115

3.9.5 - Comunicação Multiprocessadores	115
3.9.6 - O Registrador de controle do canal serial - SCON	117
3.9.7 - Taxas de Transmissão	117
3.10 – Modos de Redução de Consumo	125
3.10.1 – Modo de Espera	125
3.10.2 – Modo de Baixo Consumo	127
3.11 – Acesso a Memória Externa.....	128
3.11.1 – Acesso a Memória de Programa Externa.....	128
3.11.2 – Acesso a Memória de Dados Externa	129
3.12 – Interface Serial Periférica - SPI.....	132
3.13 – Programação em C.....	136
3.13.1 – Especificadores e modelos de memória	136
3.13.2 – Constantes, variáveis e tipos de dados	138
3.13.3 – Exemplos	139
IV – FERRAMENTAS DE DESENVOLVIMENTO	131
4.1 – Etapas de Desenvolvimento de um Programa.....	131
4.2 – Uma Visão Rápida do <i>ProView</i>	133
V – PLACA DE DESENVOLVIMENTO.....	138
5.1 – Apresentação da Placa de Desenvolvimento AT89S.....	138
5.2 – Interface para programação	138
5.3 – Interface Serial RS-232.....	143
5.4 – Interface para Display LCD.....	147
5.5 – Interface para Display de Sete Segmentos.....	164
VI - OS MICROCONTROLADORES AVR.....	173
6.1 – Introdução:	173
6.2 – Apresentação do Microcontrolador ATMEL AT90S8515.....	177
6.3 – Oscilador Interno:	180
6.4– Reset:	181
6.6 – Organização de Memória:	182
6.6.1 – Modo de endereçamento direto de registrador, um registrador Rd.....	187
6.6.2 – Modo de endereçamento direto de registrador, dois registradores Rd e Rr... 188	
6.6.3 – Modo de endereçamento indireto de registrador	189
6.6.4 – Modo de endereçamento direto de E/S.....	189
6.6.5 – Modo de endereçamento relativo de memória de programa.....	190
6.6.6 – Modo de endereçamento direto da memória de dados	191
6.6.7 – Modo de endereçamento indireto com deslocamento	192
6.6.8 – Modo de endereçamento indireto com pré-decremento	193
6.6.9 – Modo de endereçamento indireto com pós-incremento	194
6.6.10 – Modo de endereçamento de constantes na memória de programa.....	195
6.6.11 – Modo de endereçamento indireto de programa.....	196
6.6.12 – Leitura e escrita na EEPROM	197
6.7 – Estrutura e operação das portas de E/S:	201
6.8 – O conjunto de instruções do AT90S8515	206
6.8.1 – O Registrador de Estado (<i>Status Register</i> - SREG)	206
6.8.2 – Instruções lógicas e aritméticas	208
6.8.3 – Instruções de desvio.....	209
6.8.4 – Instruções de transferência dados.....	210

6.8.5 – Instruções de manipulação e teste de bits.....	213
6.9 – Como criar um programa em assembly para o AVR.....	216
6.10 – Interrupções.....	221
6.10.1 – Introdução	221
6.10.2 – Tratamento de interrupções	224
6.10.3 – Interrupções Externas	226
6.11 – Temporizadores/Contadores.....	229
6.11.1 – Timer/counter0	229
6.11.2 – Temporizador/Contador 1 de 16 bits – Timer 1	237
6.11.3 – O Timer Cão de Guarda (Watchdog)	253
6.12 – O comparador analógico	255
6.13 – UART.....	258
6.13.1 – Transmissão de dados	259
6.13.2 – Recepção de dados	262
6.13.3 – Controle da UART	264
6.13.4 – Gerador de Taxa de Comunicação.....	264
6.14 – Modos redução de consumo (sleep modes)	269
6.14.1 – Modo de espera (idle)	269
6.14.2 – Modo de baixo consumo (power-down).....	270
6.15 – Acesso a Memória de Dados SRAM Externa	270
6.16 – Programação em C com o AVR AT90S8515.....	271
REFERÊNCIAS BIBLIOGRÁFICAS.....	280
ANEXO I.....	281
<i>CONJUNTO DE INSTRUÇÕES DA FAMÍLIA MCS-51™</i>	281
ANEXO II	287
<i>MACROS</i>	287
ANEXO III.....	290
<i>DIRETIVAS DO ASSEMBLER</i>.....	290

I - CONCEITOS BÁSICOS

1.1 – Memórias : termos e conceitos importantes

- **Célula de Memória:** dispositivo empregado para armazenar um único bit. Exemplos: Flip-flops, capacitor.
- **Palavra de Memória:** Grupo de bits que representa instruções ou dados. Exemplo: um registrador constituído por um grupo de oito flip-flops.
- **Byte:** Palavra de oito bits
- **Capacidade:** O número de bits que pode ser armazenado em uma memória.

Exemplo: Considere uma memória que pode armazenar 1024 bytes; esta memória pode armazenar 8192 bits ou 1024×8 , sendo comum o uso da designação 1k para representar 1024 (2^{10}). Assim, uma memória de 4k x 8, é na verdade uma memória de 4096 x 8 .

- **Endereço:** Número que identifica a posição de uma palavra de memória. A seguir, mostramos uma pequena memória composta de 16 palavras, sendo que cada uma dessas palavras apresenta um endereço específico composto por 4 bits binários ou um algarismo hexadecimal.

Endereço		Palavra	Endereço		Palavra
0000b	0h	Palavra 0	1000b	8h	Palavra 8
0001b	1h	Palavra 1	1001b	9h	Palavra 9
0010b	2h	Palavra 2	1010b	Ah	Palavra 10
0011b	3h	Palavra 3	1011b	Bh	Palavra 11
0100b	4h	Palavra 4	1100b	Ch	Palavra 12
0101b	5h	Palavra 5	1101b	Dh	Palavra 13
0110b	6h	Palavra 6	1110b	Eh	Palavra 14
0111b	7h	Palavra 7	1111b	Fh	Palavra 15

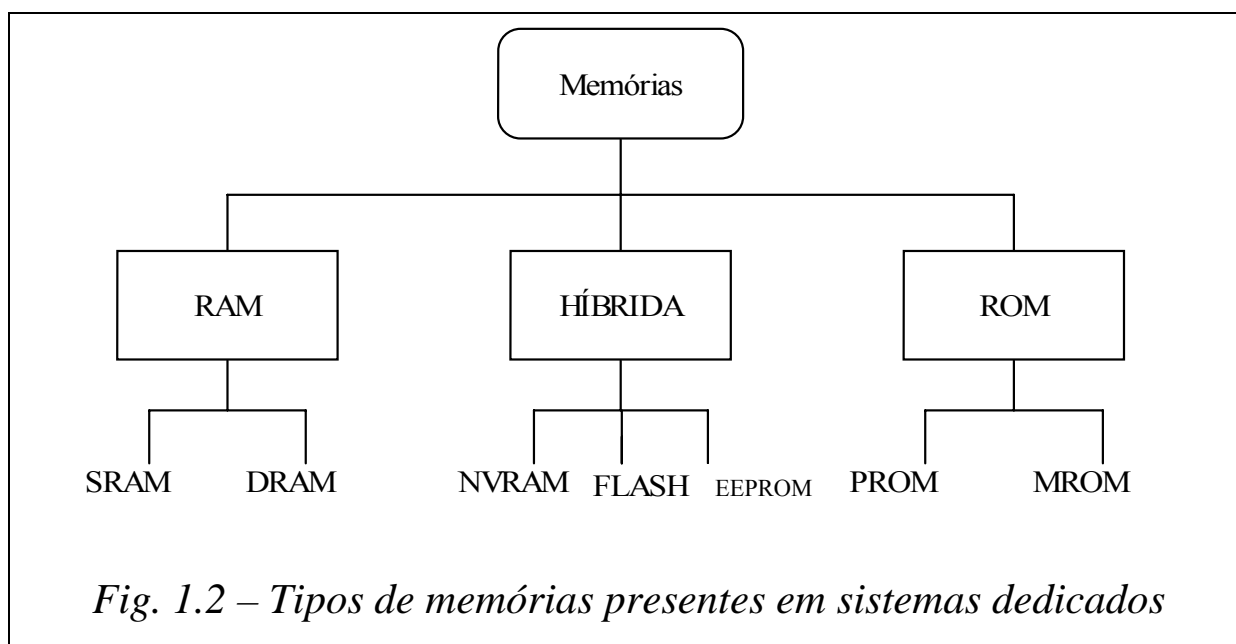
Fig.1.1 – Memória de 16 palavras.

- **Operação de Leitura:** Operação em que uma palavra binária armazenada em um endereço é transferida para outro dispositivo de um sistema digital. Por exemplo, se efetuarmos uma operação de leitura no endereço 0110b estaremos transferindo a palavra 6 para outro dispositivo.
- **Operação de Escrita:** Operação em que uma palavra é colocada em um endereço ou posição de memória. Toda vez que essa operação é executada em um determinado endereço, a uma nova palavra é armazenada nesse endereço.
- **Tempo de Acesso:** É o tempo necessário para a efetivação de uma operação de leitura/escrita.

- **Memória volátil:** Memória que necessita de energia elétrica para reter a informação armazenada; uma vez retirada a alimentação, a informação armazenada é perdida.

1.2 – Tipos de Memória

Muitos tipos de dispositivos de memória estão disponíveis para uso em sistemas modernos de processamento. A Fig. 1.2 classifica os dispositivos de memória que apresentaremos a seguir.



1.2.1 - RAM

As Memórias de Acesso Randômico (RAMs), possuem tempo de acesso constante para qualquer endereço da memória, são voláteis e permitem a gravação e a leitura de dados. Elas incluem dois tipos importantes de dispositivos: A RAM estática (SRAM) e a RAM

dinâmica (DRAM). A diferença básica entre elas é o tempo de vida dos dados armazenados. A SRAM retém os dados enquanto for garantido o fornecimento de energia para a pastilha; se a energia é desligada ou perdida temporariamente, o conteúdo é perdido. Uma DRAM, por outro lado, tem um tempo de vida dos dados extremamente curta, tipicamente cerca de quatro milisegundos, mesmo quando a alimentação é aplicada constantemente. Para permitir que o conteúdo de uma DRAM seja guardado pelo tempo necessário, emprega-se um controlador de DRAM, o qual atualiza (refresca) periodicamente os dados armazenados na DRAM antes que eles expirem.

Por oferecer um acesso extremamente rápido, cerca de quatro vezes mais rápido que uma DRAM, a SRAM, geralmente, é usada onde a velocidade de acesso é extremamente importante. Entretanto, as SRAM possuem um custo bem maior de produção, o que torna o uso das DRAM mais atrativo quando for requerida uma grande quantidade de RAM.

1.2.2 - ROM

Memória somente de leitura (**R**ead **O**nly **M**emory). Projetadas para aplicações onde a taxa de operação de leitura é infinitamente mais alta que a de escrita. Os vários tipos de ROM são apresentados na Fig. 1.3. As primeiras ROMs eram dispositivos que continham um conjunto pré-programado de dados ou instruções. Os

conteúdos da ROM eram especificados antes da produção da pastilha, conseqüentemente os dados reais eram usados para arranjar os transistores dentro da pastilha. Essas memórias ainda são usadas e são chamadas de ROMs mascaradas (MROM) para distinguir de outros tipos de ROM. Sua principal vantagem é o baixo custo de produção quando uma grande quantidade da mesma ROM é solicitada.

Uma evolução das MROMs é a OTP PROM (ROM programável uma vez), a qual é comprada desprogramada. Através de um dispositivo programador os dados são escritos, uma palavra por vez, pela aplicação de sinais elétricos nos pinos da pastilha. Uma vez programados, os conteúdos de uma OTP PROM não podem ser alterados

Uma EPROM (ROM programável e apagável) é programada da mesma maneira que uma OTP PROM. Entretanto, uma EPROM pode ser apagada e reprogramada repetidamente, o que faz que ela apresente um custo mais elevado que a OTP PROM. A UVE EPROM é o tipo de EPROM que é apagada quando exposta a uma fonte de luz ultravioleta.

1.2.3 - Híbridas

Como a tecnologia de memórias amadureceu recentemente, a linha divisória entre uma RAM e uma ROM é indefinida. Muitos tipos de memória combinam características de ambas e podem ser referenciadas como dispositivos de memória híbridos. As memórias

híbridas podem ser lidas e escritas como uma RAM, mas mantêm seus conteúdos como uma ROM. Dois desses dispositivos híbridos, a Standart EEPROM e a FLASH EEPROM, são descendentes diretas dos dispositivos ROM e, por isso, as incluímos na Fig. 1.3. Um terceiro híbrido, a NVRAM, é uma versão modificada da SRAM.

As Standart EEPROMs são apagáveis e programáveis eletricamente. Internamente, elas são parecidas com as UVE EPROMs, mas a operação de apagar é realizada eletricamente, ao invés da exposição a luz ultravioleta. A barreira primária a esta funcionalidade aperfeiçoada é o custo mais alto, embora os ciclos de escrita sejam, também, maiores que os de escrita em uma RAM.

A Flash EEPROM combina as melhores características dos dispositivos descritos anteriormente. As memórias Flash apresentam alta densidade, custo moderado, não volatibilidade, leitura rápida (a escrita não é rápida) e reprogramabilidade elétrica. Essas vantagens são irresistíveis, o que fez com que sua utilização em sistemas dedicados fosse dramaticamente incrementada.

Do ponto de vista do software, as tecnologias Flash e Standart EEPROM são muito similares. A maior diferença é que os dispositivos Flash são apagados um setor a cada vez, e não byte a byte. O tamanho típico dos setores está na faixa dos 256 bytes a 16 kB. Apesar dessa desvantagem, a tecnologia Flash é muito mais popular que a Standart EEPROM e está rapidamente substituindo os outros dispositivos ROM.

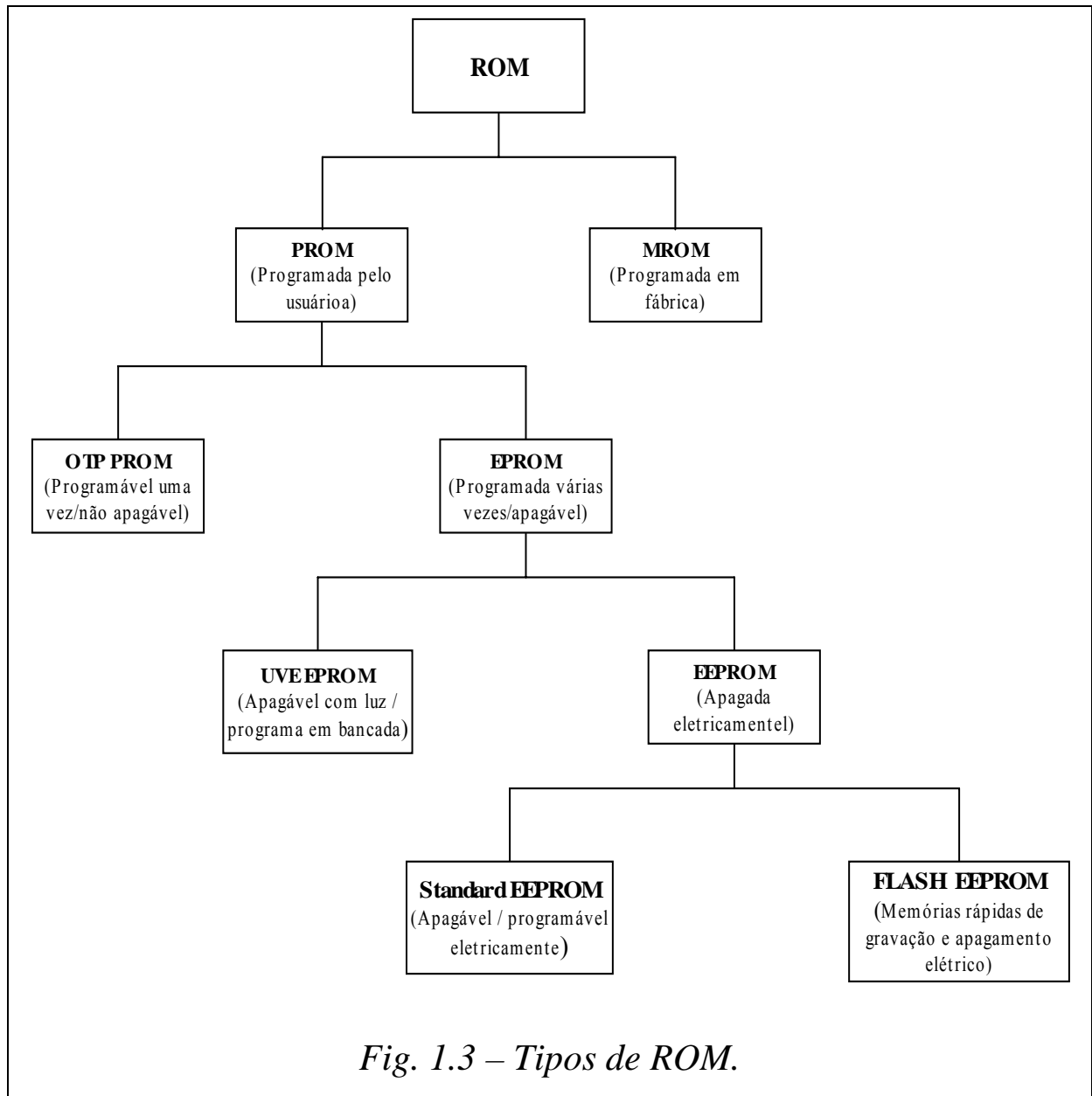


Fig. 1.3 – Tipos de ROM.

A RAM não volátil (NVRAM) é fisicamente diferente dos dois tipos de memórias híbridas discutidas anteriormente; basicamente ela é uma SRAM com uma bateria de *back-up*, a qual garante a retenção dos dados quando a alimentação é desligada. Devido ao seu custo muito elevado quando comparado ao de uma SRAM, sua aplicação é limitada ao armazenamento de algumas centenas de bytes de informações críticas em sistemas que não podem armazenar dados de outra maneira.

A Tabela 1.1 resume as características dos dispositivos de memória discutidos nesta seção, lembrando, porém, que diferentes memórias servem a propósitos diferentes, o que faz com que uma simples comparação pode não ser muito efetiva

1.3 - Operação em memória

1.3.1 - Entradas de Endereço

A memória apresentada na Fig. 1.4, uma vez que ela armazena 16 (2^4) palavras de 0000b a 1111b, necessita de quatro entradas de endereços: A_0 , A_1 , A_2 e A_3 . Em geral, uma memória com capacidade de 2^n palavras necessita de n linhas de entrada de endereço.

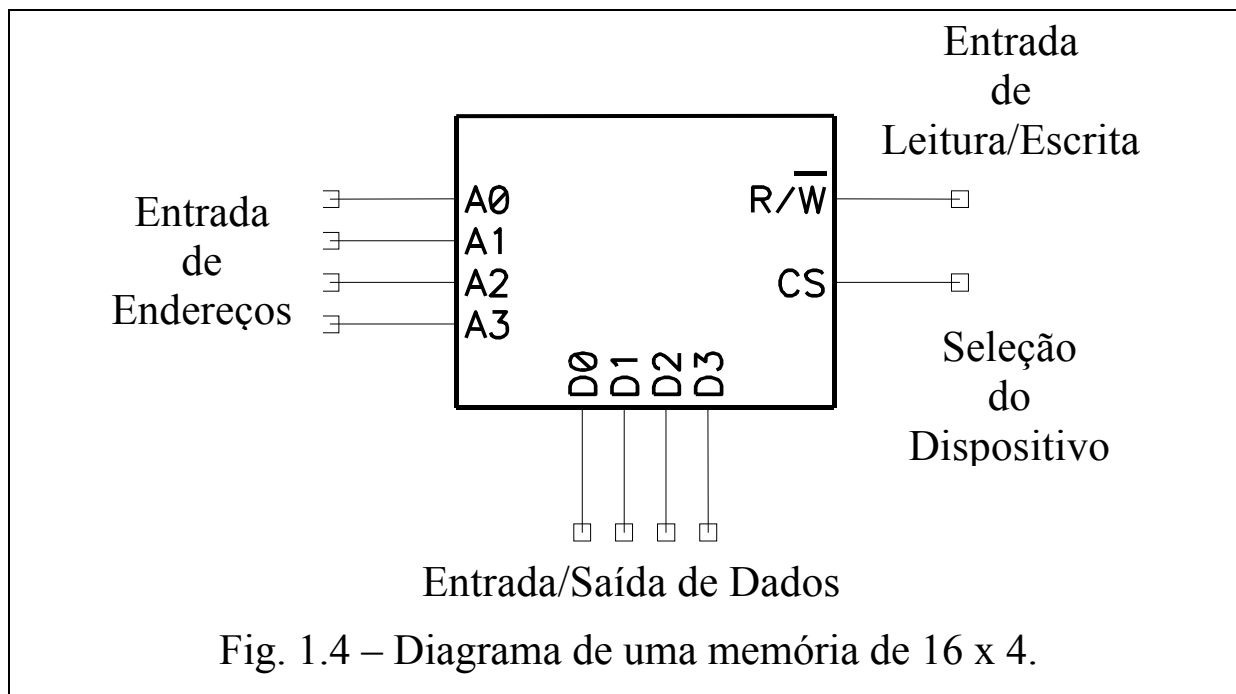


Fig. 1.4 – Diagrama de uma memória de 16 x 4.

TABELA 1.1¹
CARACTERÍSTICAS DE ALGUNS TIPOS DE MEMÓRIA

Tipo	Volátil?	Regravável?	Como é apagada?	Ciclos de apagamento	Custo/ byte	Velocidade
MROM	Não	Não	N/a	N/a	baixo	rápida
OTP PROM	Não	Uma vez	N/a	N/a	moderado	rápida
UVE EPROM	Não	Sim	Toda a pastilha de uma vez	Limitado	moderado	rápida
EEPROM	Não	Sim	Por Byte	Limitado	caro	leitura rápida, escrita/apagamento lentos
Flash	Não	Sim	Por setor	Limitado	moderado	leitura rápida, escrita/apagamento lentos
SRAM	Sim	Sim	Por byte	ilimitado	caro	rápida
DRAM	Sim	Sim	Por byte	ilimitado	moderado	moderada
NVRAM	Não	Sim	Por byte	ilimitado	muito caro	moderada

¹ As memórias híbridas são apresentadas em linhas sombreadas.

1.3.2 - Entrada de leitura (R) e escrita (\overline{W})

A linha de entrada de leitura/escrita (R/\overline{W}) determina qual operação será realizada na memória, quando $R/\overline{W} = 1$ temos uma operação de leitura, quando $R/\overline{W} = 0$ temos uma operação de escrita. É importante observar que uma operação de leitura não altera a informação contida no endereço.

1.3.3 - Seleção do dispositivo

É responsável pela seleção da memória. Na Fig. 1.2 esta entrada é ativa em nível lógico “1”, ou seja, quando esse sinal estiver no nível lógico alto a memória opera normalmente. Por outro lado, $CS = 0$ desabilita a memória, ou seja, ela não responde às entradas R/\overline{W} e de endereços.

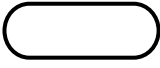
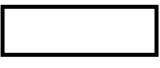
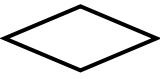
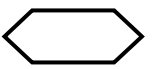

1.4 – Fluxogramas

Um fluxograma representa graficamente as tarefas que serão realizadas por um programa de um microcontrolador e constitui-se em uma ferramenta que permite economizar tempo de desenvolvimento de software, pois facilita a alteração ou correção do programa pelo programador ou, ainda, por outra pessoa. Os símbolos mais utilizados na construção de um fluxograma são apresentados na Tabela 1.2.

Na Fig. 1.5 representa-se um fluxograma que contém as ações necessárias para o cálculo das raízes de uma equação de

segundo grau (equação do tipo $ax^2 + bx + c$).

TABELA 1.2
SÍMBOLOS BÁSICOS DE FLUXOGRAMAS

Símbolo	Função	Descrição
	Terminação	Representa o início e o fim de um programa. Quando representar o início, deve conter o nome da rotina, sendo interessante que o nome lembre a função executada.
	Processo	Representa a ação executada.
	Decisão	Através do teste de uma informação, determina o caminho que deve ser seguido pelo fluxo de informações.
	Sub-rotina	Quando um conjunto de instruções específicas é utilizado mais que uma vez em um programa, elas são reunidas em um programa separado do programa principal chamado de sub-rotina. O hexágono representa o ponto de chamada de uma sub-rotina no programa principal.
	Fluxo	Indica a direção do fluxo de informações.

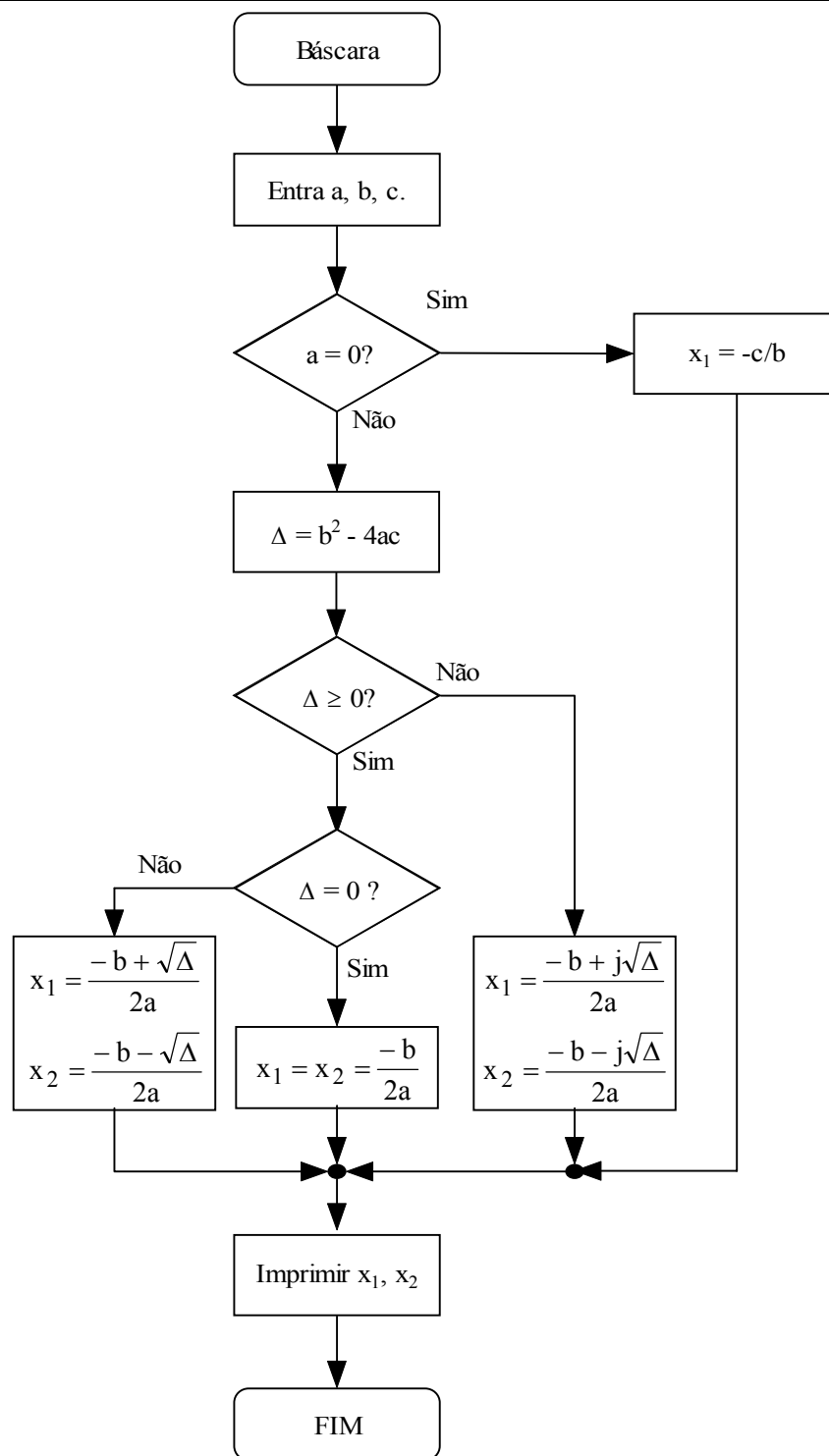


Fig. 1.5 – Fluxograma para o cálculo das raízes de $f(x) = ax^2 + bx + c$.

1.5 – Representação de números em microcontroladores

1.5.1 – Representação de números inteiros

Como computadores lidam com números positivos e números negativos, é necessário encontrar uma representação satisfatória. Para representar números positivos, geralmente se utiliza o número binário puro. Desta forma, os números decimais 4 e 9 são representadas, respectivamente, por 0100 e 1001. Para representar os números negativos, a forma mais adequada é a representação em complemento de 2, descrita com mais detalhes na seção 1.5.2.

Em sistemas digitais existe um limite para o maior número representável, uma vez que a memória sempre será finita. Esse limite é 2^n números, onde n representa o número de bits dos registradores. Em um microcontrolador com registradores de 4 bits é possível representar ao todo 16 (2^4) números diferentes, de 0000 a 1111. Após o limite 1111, teríamos 10000 ($1111 + 1$). Mas, como a representação é de quatro dígitos, estaríamos de volta ao 0000, pois 10000 tem um bit a mais do que o permitido.

Esta volta à origem sugere dispor os números ao redor de um círculo, e não ao longo de um eixo infinito, como se faz na matemática convencional. Esta representação aparece na Fig. 1.6, que permite visualizar o que acontece quando fazemos operações de soma e subtração. Para executar a soma de dois números a e b , basta encontrar a representação de a no círculo e avançar b posições no sentido horário. Para fazer a subtração $a - b$, por outro lado, basta

recuar b posições a partir de a , no sentido anti-horário. Exemplos de cálculos possíveis são $5 + 3$ ou $6 - 4$.

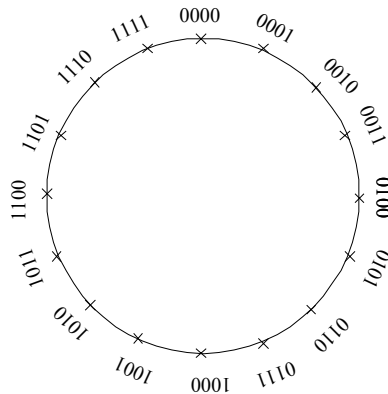


Fig. 1.6 – Representação circular dos números de 4 bits.

A Fig. 1.6 também mostra o que acontece quando o resultado de uma soma excede o maior número representável, no caso, 1111. Se tentarmos somar $10 + 6$, por exemplo, chegamos a 0000 e não a 10000, que seria o resultado correto, exatamente como acontece quando somamos dois números num processador e não temos como registrar o "vai 1".

O modelo apresentado até aqui é suficiente para os casos em que não aparecem valores negativos nos cálculos realizados. No entanto, sabemos que existem grandezas que podem assumir valores negativos, e por isso é natural perguntar o que acontece quando o resultado de uma operação é um número negativo. Se, na Fig. 1.6, tentarmos representar o cálculo de $0 - 1$, começamos na posição 0000 e recuamos uma posição no sentido anti-horário, o que nos leva a 1111. Da mesma forma, os números negativos subsequentes ficariam em 1110, 1101, 1100, etc. Poderíamos continuar desta forma e chegar até o número 0001, que representaria então o valor -15. Mas aí

ficaríamos sem números positivos, e por isso é necessário definir uma fronteira que separe os números positivos dos negativos.

Costuma-se considerar positivos os números cujo bit mais significativo é 0 e negativos os números cujo bit mais significativo é 1, o que divide ao meio o conjunto de números representáveis. Isto também facilita a tarefa de determinar o sinal de um número em um programa, pois basta analisar esse bit. No caso dos números de 4 bits, ficamos com 8 números negativos e 8 números positivos, considerando-se o zero incluído nestes últimos. A disposição desses números ao redor do círculo aparece na Fig. 1.7.

Note que, agora, a de representação dos números não é mais de 0 a 15, mas sim de -8 até +7. Para representar números além destes limites, seria preciso adotar registradores maiores, por exemplo de 8, 16 ou 32 bits.

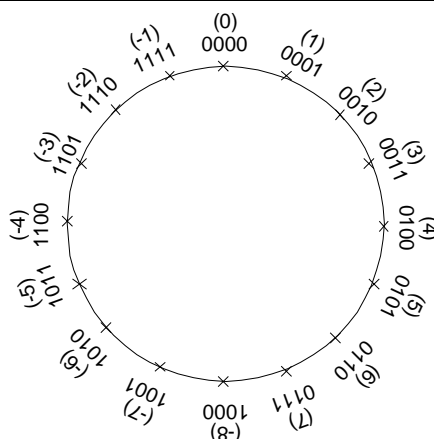


Fig. 1.7 – Representação de números positivos e negativos.

No caso geral de números de n bits, o aspecto do círculo é semelhante ao da Fig. 1.5. O número de divisões é sempre igual a 2^n , os números positivos vão de 0 até $2^{n-1} - 1$ e os números negativos de -1

até -2^{n-1} . Estes números são encontrados na documentação de compiladores de linguagens como o C, ao se apresentar diferentes tipos de variáveis e os seus respectivos limites de representação.

1.5.2 – Representação e complemento de 2

Ao se comparar as representações de 1 e -2, e 2 e -3, e assim por diante, conforme ilustram as linhas tracejadas da Fig. 1.8, percebe-se uma pode ser obtida invertendo-se os seus bits. O resultado obtido a partir da inversão de todos os bits de um número chama-se *complemento de 1*.

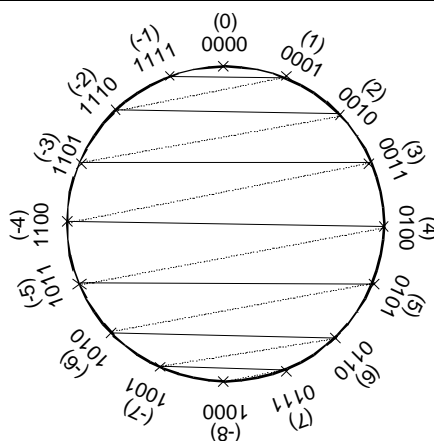


Fig. 1.8 – Relações de complemento entre simétricos.

O simétrico de um número a , $-a$, representado pelas linhas cheias pode ser obtido através da expressão:

$$-a = (\text{complemento de 1 de } a) + 1 \quad (1.1)$$

O membro direito dessa expressão é conhecido como *complemento de 2 de a* . Pode-se mostrar que

$$(\text{complemento de 2 de } a) = 2^n - a, \quad (1.2)$$

em que n é o número de bits utilizados para representar os números.

Aplicar a equação (1.1) para se obter o complemento de 2 ou simétrico de um número é uma tarefa simples. Por exemplo, para se obter a representação de -5 (1011) a partir de 5 (0101), inverte-se os bits deste (1010) e adiciona-se 1, obtendo 1011.

1.5.3 - Subtração usando adição

A representação em complemento de 2 permite a realização de subtrações utilizando a adição. Isto é particularmente interessante na implementação de circuitos digitais, uma vez que o mesmo circuito pode ser utilizado para executar ambas as operações.

O ponto de partida para fazer a subtração $a - b$ com auxílio da adição é observar que, se podemos representar números de no máximo, n bits, é verdade que

$$a - b = a + 2^n - b \quad (1.3)$$

pois, o fator 2^n representa uma volta completa no círculo da Fig. 1.6 e não afeta o resultado. Pela equação (1.2),

$$a - b = a + (\text{complemento de 2 de } b) \quad (1.4)$$

Por exemplo, o resultado de (7-5) pode ser obtido adicionando-se 0111 (7) e o complemento de 2 de 0101 (5), 1011 (-5), desconsiderando-se o "vai 1", que pode ser armazenado:

$$\begin{array}{r} 0111 \\ + \underline{1011} \\ (1) 0010. \end{array}$$

Observe, a partir da Fig. 1.8, que somar 0111 a 1011 é o mesmo que dar uma volta completa no círculo e voltar 5 posições.

1.5.4 - A representação BCD

Freqüentemente, em sistemas digitais, é necessário apresentar o conteúdo de um registrador em algum tipo de mostrador, tal como um display de sete segmentos. Como a maioria das pessoas pode somente compreender e manipular números decimais, o valor binário contido no registrador deve, geralmente, ser mostrado no sistema decimal. Assim, se faz necessário obter a representação decimal desse valor, para depois apresentá-lo. A forma de representação mais usual é no formato BCD (*binary coded decimal*), decimal codificado como binário.

O formato BCD utiliza quatro bits binários para representar um único dígito decimal de 0 até 9. Como números maiores que nove não são usados, a representação de 4 bits dos números de 10 a 15 não podem aparecer em uma saída BCD, o que diminui a quantidade máxima de números que podem ser representados com oito bits de 256 para 100. Por exemplo, no formado BCD o número 00010010b representa o número decimal 12 e não o número hexadecimal 12h que, convertido em decimal, representaria a quantidade 18 ($2 + 1 * 16$). Isso causa problemas quando se fazem operações com esses números. No entanto, é possível elaborar regras simples que permitem fazer adições e subtrações com números BCD.

Seja a Fig. 1.9, que representa os números hexadecimais possíveis com 4 bits. Observe que a soma de dois dígitos será correta no formato BCD enquanto não exceder a 9, uma vez que a soma hexadecimal $3h + 5h = 8h$ é a mesma que a soma BCD $2 + 6 = 8$. Quando o valor passa de 9, porém, em BCD o resultado estaria incorreto pois, ou surgem letras no resultado (como em $4 + 7 = B$) ou então o resultado pode estar incorreto (como em $9 + 8 = 11$, considerando o “vai um”). Em qualquer um dos casos, o problema é que as seis letras de A a F deveriam ser “saltadas” para que se encontrasse o resultado correto em BCD, ou seja, ao invés de avançar sobre o círculo, fosse utilizado o atalho indicado pela linha tracejada da Fig. 1.9.

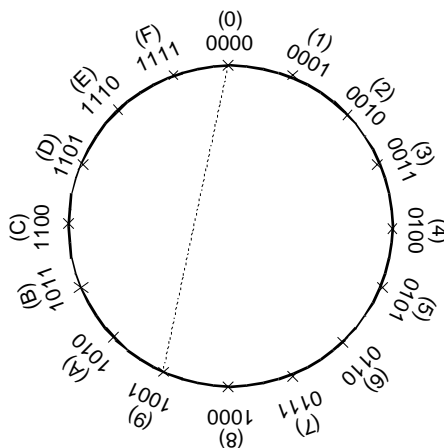


Fig. 1.9 – Os dígitos do sistema hexadecimal e um atalho.

De posse dessas informações, se pode elaborar o seguinte algoritmo para a soma de números BCD:

1. somar os dígitos como hexadecimais;
2. se a soma for superior a 9, somar mais 6.

Por exemplo, ao somar-se os números BCD 73 e 39, o resultado correto é 112. No sistema hexadecimal, o resultado seria ACh. Aplicando o algoritmo sugerido, temos os seguintes passos:

$$\begin{array}{r}
 1. \\
 \quad 73 \\
 + \quad \underline{39} \\
 \quad \text{AC} \\
 2. \\
 \quad \quad 1 \\
 \quad \text{AC} \\
 + \quad \underline{66} \\
 \quad \underline{112}
 \end{array}$$

Considerações semelhantes podem ser feitas em relação a subtração. Por exemplo, o cálculo $12 - 4$, na ausência de correção, dá o resultado hexadecimal 0Eh. Subtraindo mais 6, obtém-se o resultado BCD desejado, 8.

É comum encontrar em microcontroladores a instrução que implementa a correção para a soma de dois números BCD, mas não para a subtração. Por isso, a subtração deve ser implementada a partir da adição, empregando o complemento de 10. O complemento de 10 de um número é obtido subtraindo-o de 10^N , onde N é a quantidade de dígitos do número em questão. Expressando numa equação, temos:

$$\text{Complemento de 10 de } a = 10^N - a \quad (1.5)$$

Por isso, a subtração $a - b$ pode ser resolvida como $a + (10^N - b)$, onde N é o número de dígitos disponíveis para representar a e b . Haverá um "vai 1", que excede o número de dígitos da representação e será desconsiderado.

Para o exemplo, em uma representação de 2 dígitos, a subtração BCD discutida acima, $12 - 4$, obedeceria o seguinte procedimento:

- Obter o complemento de 10 de 4, no caso 96 ($100 - 4$);
- Adicionar 12 e 96, resultando em A8h;
- corrigir o resultado pelo algoritmo da *soma BCD*, obtendo 108h;
- descartar a centena, uma vez que, por hipótese, a representação era de 2 dígitos, obtendo o resulta 08.

1.5.5 - O Código ASCII




Com o objetivo de facilitar a comunicação sob a forma de texto entre dois computadores e entre computadores e dispositivos de entrada e saída (E/S) foi criado, em 1961, o Código ASC II (*American Standart Code for Integrated Interchange*). O Código ASCII, apresentado na Tabela 1.3, é um código numérico que usa a escala de 00h a 7Fh para representar letras, números, acentos e sinais diversos. Observe que os caracteres ASCII precisam de 7 bits para a sua codificação. É importante observar que os caracteres imprimíveis são apenas 95, os trinta e dois primeiros códigos para caracteres utilizados no controle de equipamentos.

TABELA 1.3
TABELA ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACL	BEL	BS	HT	LF	VT	FF	SR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Com o passar do tempo, surgiu a necessidade de padronizar a representação de caracteres acentuados (diacríticos), caracteres utilizados em molduras de texto e muitos outros. Foi desta forma que a IBM ao lançar o seu PC ampliou a tabela ASCII dando a ela a capacidade de 256 caracteres codificados, incluindo então além dos diacríticos uma série de caracteres semi-gráficos. Esta ampliação da tabela ASCII não acomodava todas as possibilidades de caracteres acentuados. Para atender todos os idiomas, a Microsoft implementou o conceito de página de código (*code pages*), onde haveriam diversas tabelas ASCII estendidas, de acordo com as necessidades da língua do usuário. A página de códigos que melhor atende os países da América Latina é a página de códigos 850. A Tabela ASCII estendida ASC 850 é apresentada na Tabela 1.4.

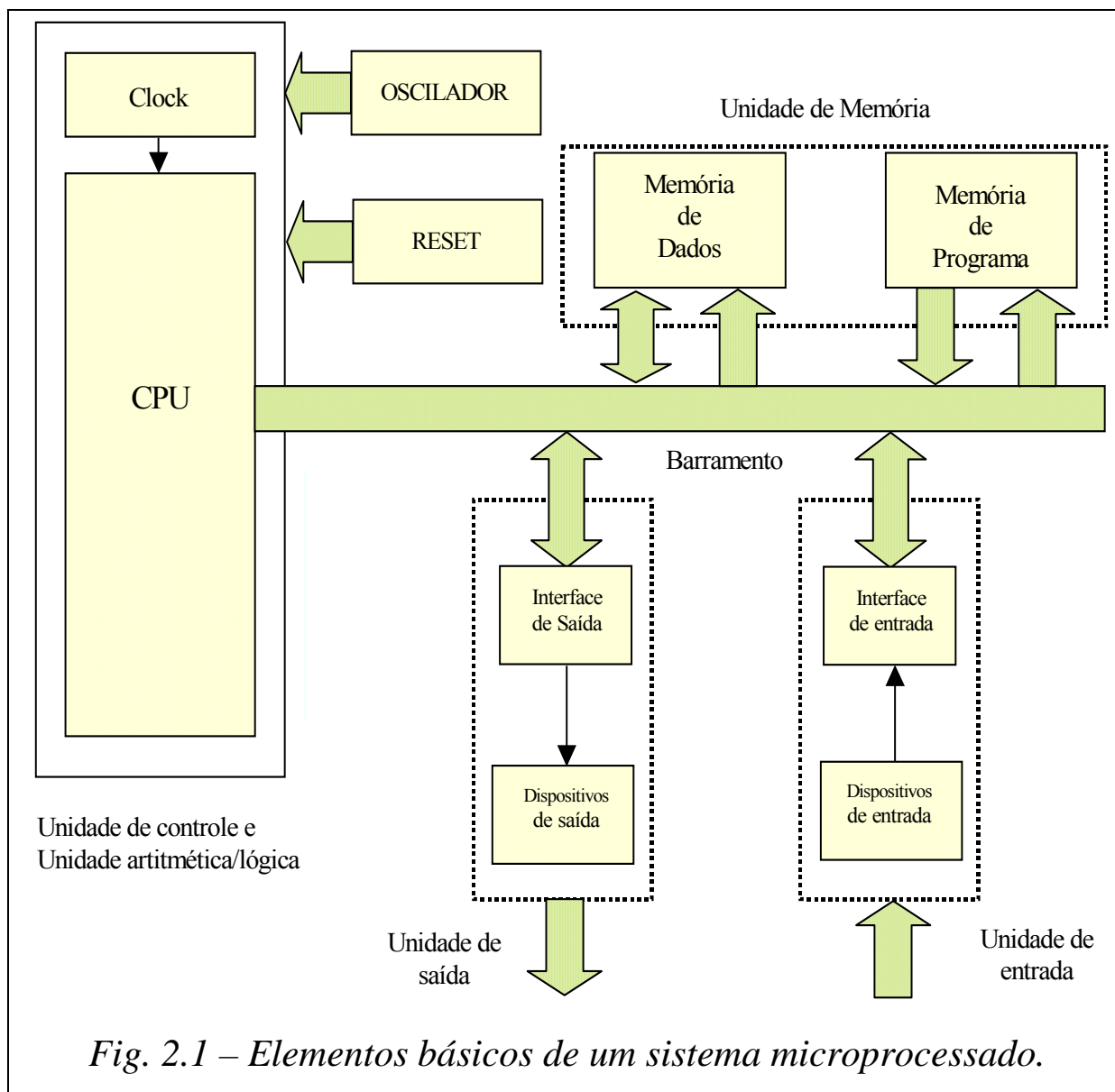
TABELA 1.4
EXTENSÃO DO CÓDIGO ASCII – ASC 850

	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>F</i>
8	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	Ä	Å
9	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	¢	£	¥	Ps	f
A	á	í	ó	ú	ñ	Ñ						½	¼	-	«	»
B					¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬
C																
D																
E	α	β	Γ	Π	Σ	σ	μ	τ	Φ	Θ	Ω	δ	∞	ø	ε	∩
F	≡	±	≥	≤	∫	∫	v	≈	◻	◻	·	√	n	z	■	□

II - SISTEMAS COM MICROPROCESSADOR

2.1 – Descrição:

- **Circuito de Reset:** Responsável pela inicialização do sistema.
- **Oscilador:** Gera os sinais de clock para sincronismo do sistema
- **Memória de Programa:** Responsável pelo armazenamento das instruções que compõe o programa que o microprocessador irá executar.
- **Memória de Dados:** Responsável pelo armazenamento temporário das informações.
- **Unidade de Saída:** Durante o processamento de um programa o microprocessador pode ler informações em dispositivos de entrada ou escrevê-las em dispositivos de saída. Os dispositivos de entrada/saída, como, por exemplo, interruptores, teclados , LEDs e displays de cristal líquido, são conectados ao barramento do sistema através de interfaces de entrada/saída.
- **Barramento:** Através do barramento fluem o **endereço** da memória ou do dispositivo de entrada/saída que está sendo acessado pelo microprocessador, os **dados** que estão sendo escritos ou lidos nos mesmos e os **sinais de controle** necessários para a sincronização dos diversos elementos do sistema.



- **CPU:** É o coração do sistema. É responsável por:
 - Fornecer sinais de controle e temporização do sistema;
 - Buscar as instruções na memória de programa e decodificá-las;
 - Buscar ou transferir dados para a memória de dados;
 - Realizar operações lógicas e aritméticas requisitadas pelas instruções;

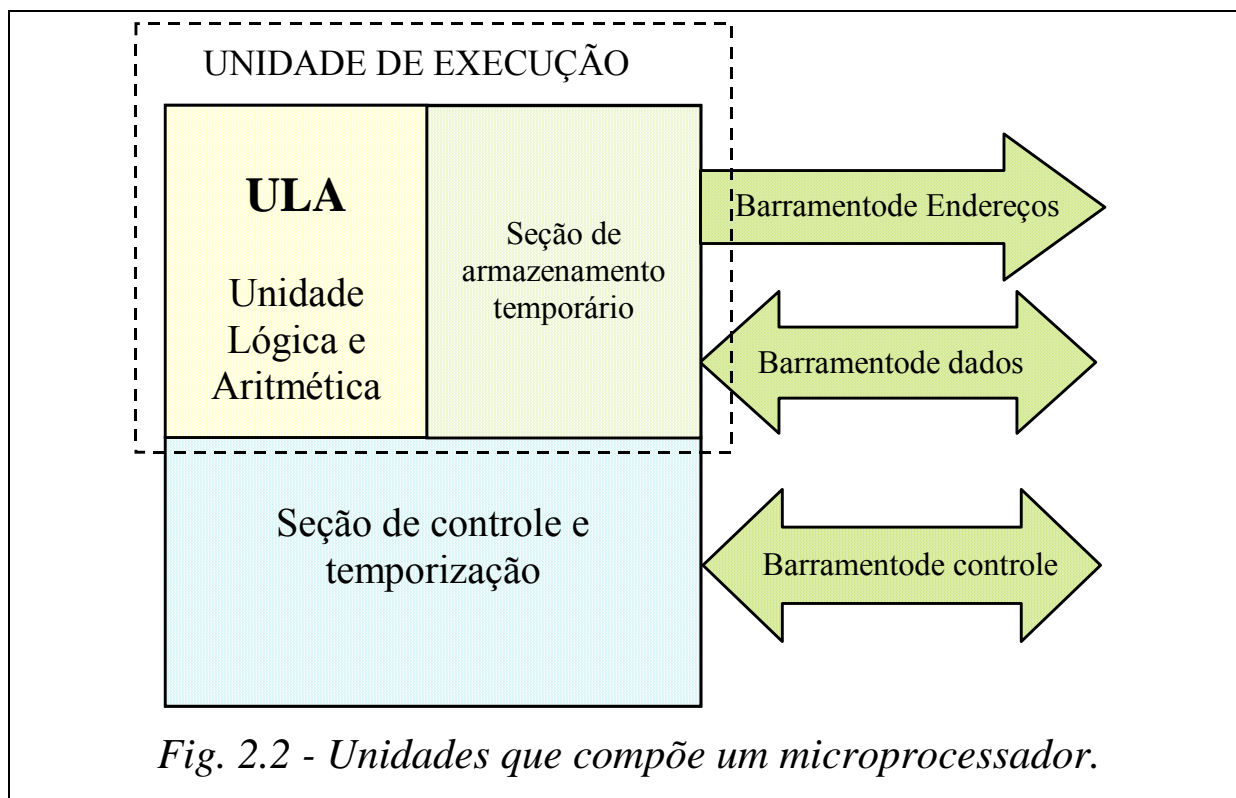
-
- Responder aos sinais gerados por outros dispositivos.

Para realizar tais funções a CPU, conforme ilustra a Fig. 2.2, dispõe de:

- **Unidade Lógica e Aritmética:** Responsável pela realização das operações lógicas e aritméticas, tais como adição, subtração, multiplicação, divisão e as operações lógicas OU, E e OU-EXCLUSIVO.
- **Seção de Armazenamento Temporário:** Contém diversos registradores utilizados durante a execução de uma instrução para armazenamento de dados temporariamente, como por exemplo, o Contador de Programa (PC). Este registrador armazena o endereço da próxima instrução que o processador irá buscar na memória de programa.
- **Unidade de Controle e Temporização:** Responsável pela decodificação do código da instrução buscada na memória de programa e pela geração dos sinais de controle exigido pela instrução.

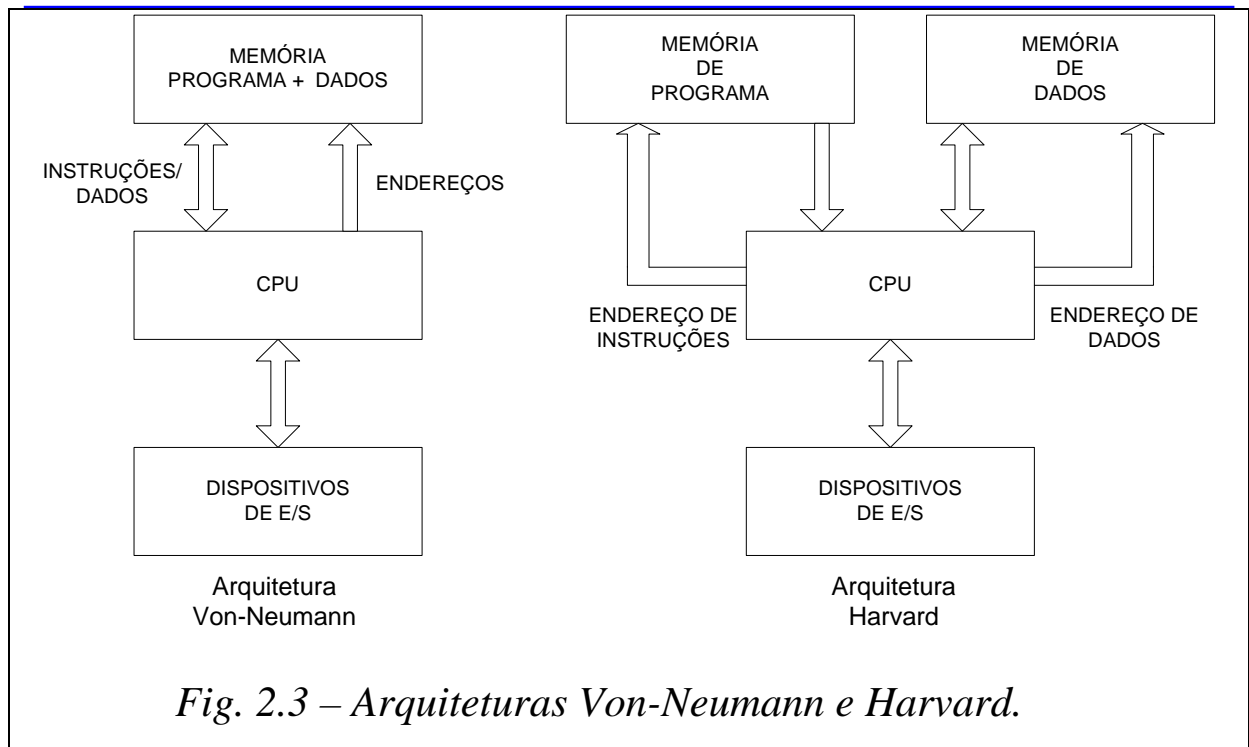
A CPU busca as instruções na memória de programa via barramento de endereço e de dados, sendo que as instruções são buscadas de locações sucessivas de memória até a execução de uma instrução de ramo ou de desvio. Uma vez buscada, uma instrução espera no registrador de instrução até que a lógica de controle da CPU esteja pronta para decodificá-la. Uma instrução é decodificada pela lógica de controle para produzir sinais de controle, os quais são

introduzidos na unidade de execução (ULA + Seção de Armazenamento Temporário) para produzir micro-operações que executam a função de uma instrução.



2.2 – Arquitetura Von-Neumann x Harvard :

Na arquitetura Von-Neumann existe apenas um barramento interno por onde circulam instruções e dados. A arquitetura Harvard, por sua vez, é caracterizada por dois barramentos internos, sendo um de instruções e outro de dados, o que permite que dados e instruções possam ser buscadas simultaneamente, incrementando a velocidade do processamento. A Fig. 2.3 ilustra as duas arquiteturas.



2.3 – Microcontroladores:

O microcontrolador é um microprocessador altamente integrado projetado especificamente para uso em sistemas dedicados, ou seja, em sistemas projetados para executar uma função dedicada. Sua operação se baseia na ação lógica que ele deve executar em função do estado dos dispositivos de entrada e saída. Por exemplo, em um controle de acesso por senha, caso o usuário digite, através de um teclado, corretamente a sua senha, o microcontrolador libera a trava eletrônica da porta. Enquanto os projetistas de microprocessadores preocupam-se com maiores comprimentos de palavra e espaço de endereçamento, os projetistas de microcontroladores preocupam-se com a integração de periféricos necessários para garantir controle rápido em sistemas dedicados.

Os microcontroladores reúnem em um único CI elementos que em um sistema microprocessado baseado em microprocessadores estavam presentes em outros CIs, formando um sistema computacional completo. Os Microcontroladores, comumente, contém CPU, memória, oscilador, portas de E/S, interface serial e temporizadores. O diagrama de blocos de um microcontrolador simples é mostrado na Fig. 2.4.

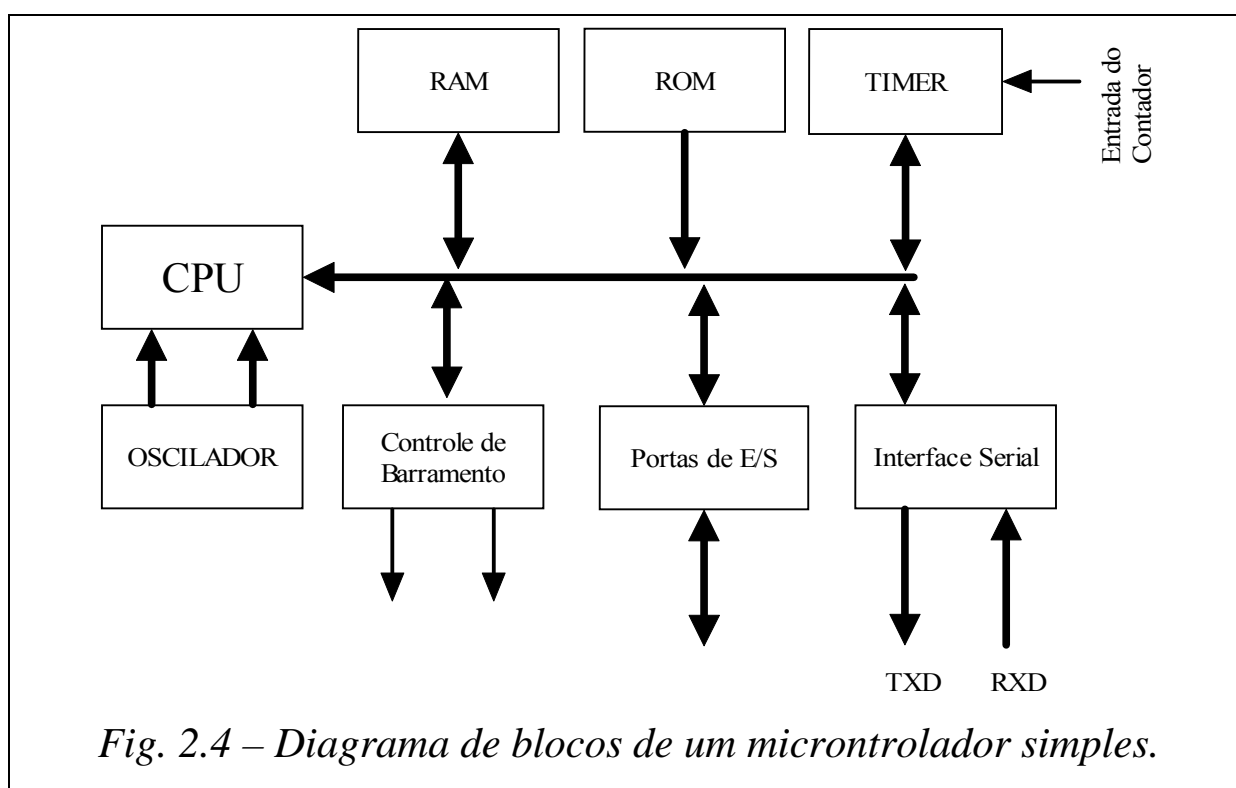


Fig. 2.4 – Diagrama de blocos de um microcontrolador simples.

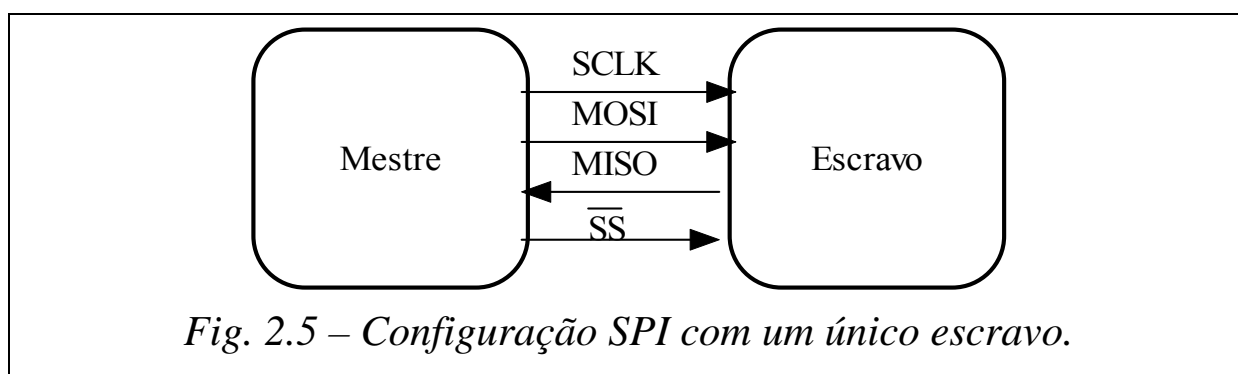
As portas de E/S de propósito geral são os periféricos mais básicos disponíveis em microcontroladores. Cada pino de saída pode ser colocado em nível lógico zero ou um pelo emprego de instruções para “setar” ou “limpar” o bit correspondente no registro de dados da porta. Do mesmo modo, o estado lógico de cada pino de entrada pode ser lido pela CPU usando instruções de programa.

Um temporizador é um periférico relacionado com a medida ou geração de eventos com base no tempo. Os Temporizadores, usualmente, medem tempos relativos a um relógio interno do microcontrolador, embora alguns possam ser controlados por uma fonte externa. Eles podem executar funções como geração de interrupções periódicas e medida de largura de pulso.

Para permitir que a CPU comunique-se serialmente bit a bit com dispositivos externos, inclui-se nos microcontroladores algum tipo de interface serial. Este tipo de comunicação quando comparada a paralela requer menos pinos de E/S, o que a torna a comunicação menos onerosa, porém mais lenta. As transmissões seriais podem ser executadas síncrona ou assincronamente. A Transmissão/Recepção Assíncrona Universal (UART) comunica-se assincronamente com outros dispositivos e requer uma interface de hardware muito simples. Somente dois pinos são necessários para transferência bidirecional de dados: os dados são transmitidos para fora do microcontrolador por um pino (TXD) e recebidos por outro pino (RXD). Cada porção de dados transmitida ou recebida pela UART tem um bit de início, diversos bits e um bit de parada, sendo que os bits de início e de parada são usados para sincronizar os dois dispositivos em comunicação. Com um interface RS-232 conectada aos pinos TXD e RXD, a UART pode se comunicar com computadores pessoais.

Alguns microcontroladores utilizam uma interface serial conhecida como SPI – Interface Serial Periférica. Ela é um canal de dados serial síncrono que opera em modo *full-duplex*, ou seja, carrega

dados simultaneamente em ambas as direções. Os dispositivos comunicam-se usando uma relação mestre/escravo, na qual o mestre inicia a janela de dados. Quando o mestre gera um relógio e seleciona um dispositivo escravo, os dados podem ser transmitidos em ambas as direções simultaneamente. A interface serial Periférica especifica quatro sinais: relógio (SCLK); saída de dados do mestre, entrada de dados do escravo (MOSI); entrada de dados do mestre, saída de dados do escravo (MISO) e seleção de escravo (\overline{SS}). A Fig. 2.5 mostra esses sinais em uma configuração de um único escravo. O sinal SCLK é gerado pelo mestre; a linha MOSI transmite dados do mestre para o escravo; a linha MISO transmite dados do escravo para o mestre; o escravo é selecionado quando o mestre baixa seu sinal \overline{SS} . Se muitos dispositivos escravos existem, o mestre gera um sinal \overline{SS} para cada escravo.



III - O MICROCONTROLADOR AT89S8252

3.1 – Introdução:

O AT89S8252 tem a sua arquitetura baseada na família MCS-51™ da INTEL, cujo membro original foi apresentado ao mercado em 1980. Suas principais características, as quais estudaremos detalhadamente nas unidades seguintes, são:

- 64 kbytes de endereçamento de memória de programa e memória de dados;
- Memória de programa interna Flash de 8k bytes reprogramável “In-System” com 1000 ciclos de gravação;
- Memória de dados interna EEPROM de 2k bytes com 100000 ciclos de gravação;
- Faixa de alimentação de 4.0 a 6.0 V;
- Frequência de operação de 0 a 24 MHz;
- Memória RAM interna de 256 bytes;
- 32 linhas programáveis de entrada/saída;
- 3 contadores/temporizadores de 16 bits;
- 9 fontes de interrupção
- Canal serial programável;
- Interface Serial SPI ;

- Temporizador programável Watchdog (Cão de guarda);

Para efeito de comparação, na Tabela 3.1 apresentamos as características relevantes de alguns microcontroladores da família MCS-51™ da INTEL e da família Flash da ATMEL com arquitetura compatível com a primeira.

TABELA 3.1

MICROCONTROLADORES COMPATÍVEIS COM O 8051

Nome do dispositivo	Memória de Programa (bytes)	Memória de Dados (bytes)	Temporizadores de 16 bits
80C51BH	MROM de 4k	RAM de 128	2
80C31BH	-	RAM de 128	2
87C51	OTP/EPROM de 4k	RAM de 128	2
80C52	MROM de 8k	RAM de 256	3
AT89C51	Flash de 4k	RAM de 128	2
AT89C52	Flash de 8k	RAM de 256	3
AT89S8252	Flash de 8k	RAM de 256 EEPROM de 2k	3
AT89S53	Flash de 12k	RAM de 256	3

Na Tabela 3.2 mostramos a evolução do microcontrolador AT89S8252 a partir da plataforma 80C52.

TABELA 3.2
EVOLUÇÃO DO MICROCONTROLADOR 80C52

80C52 ⇒ AT89C52 ⇒ AT89S8252		
<ul style="list-style-type: none"> • Pinagem e código compatível com o 80C51 • Memória ROM de 8kB • Memória RAM de 256B • 3 temporizadores /contadores • Interface Serial UART • 32 pinos de E/S • Extensa gama de ferramentas de suporte 	<ul style="list-style-type: none"> • Substituição direta do 80C52 • Memória Flash de 8kB • 1000 Ciclos de Gravação • Operação de 0Hz a 24 MHz 	<ul style="list-style-type: none"> • Programação In-System por 4 fios • EEPROM de 2 kB de alta durabilidade • Interface Serial SPI • Temporizador Watchdog

Sua arquitetura básica é apresentada no diagrama de blocos da Fig. 3.1, onde estão salientadas as diferenças básicas entre os microcontroladores AT89S8252 e 80C52. O AT89S8252 é um CI de 40 pinos, pinos estes cujo a função descrevemos sucintamente na Tabela 3.3.

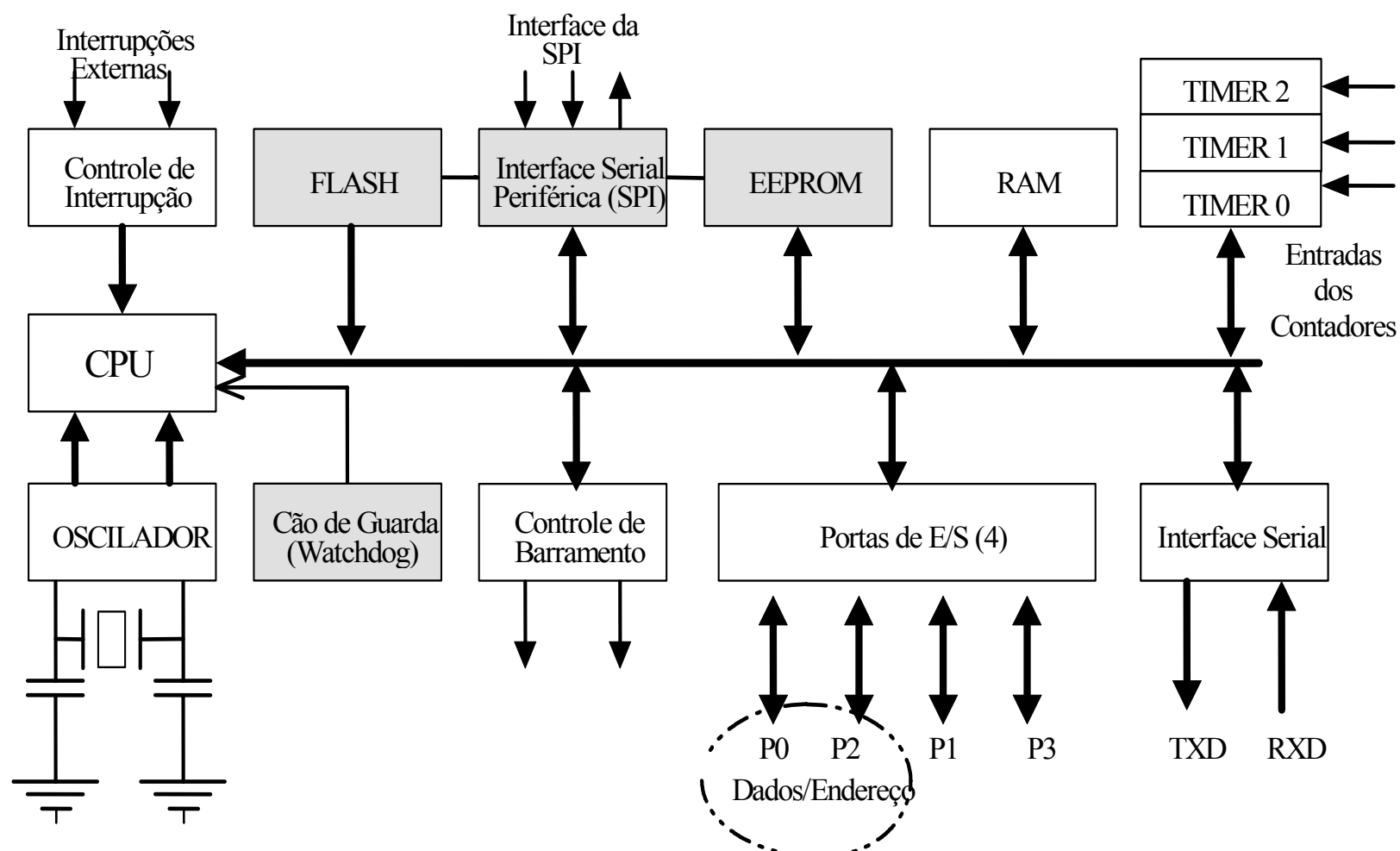


Fig. 3.1 – Diagrama de Blocos do AT89S8252.

TABELA 3.3
PINAGEM do AT89S8252

XTAL 1: Entrada para um amplificador inversor que pode ser configurado como oscilador interno.

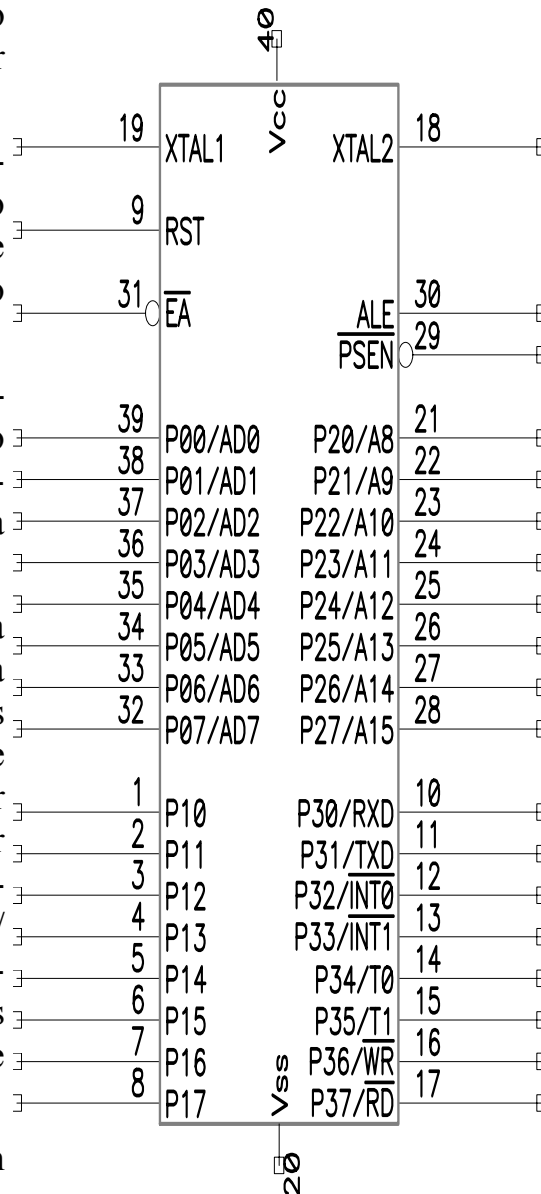
RST: Um nível lógico alto nesse pino por dois ciclos de máquina reinicia o dispositivo.

\overline{EA} : Habilita, quando em nível lógico zero, acesso a memória de programa externa.

Porta 0: A porta zero é uma porta bidirecional de 8 bits em dreno aberto e que pode ser configurada para ser o barramento multiplexado de dados / byte menos significativo de endereços para acesso de memória externa.

Porta 1: A porta um é uma porta bidirecional de 8 bits com “pullups” internos. Alguns pinos provêm funções adicionais apresentadas na Tabela 3.4 e que descreveremos nas seções posteriores.

Vcc: Alimentação, de 4 a 6 V



Vss: Referência (GND)

XTAL 2: Saída para um amplificador inversor.

ALE: “Adress Latch Enable” é um pulso de saída para travar o byte menos significativo de endereços durante o acesso à memória externa.

\overline{PSEN} : “Program Store Enable” é um pulso de saída para leitura de memória externa.

Porta 2: A porta dois é uma porta bidirecional de 8 bits com “pullups” internos. Essa porta emite, também, o byte mais significativo de endereços.

Porta 3: Esta porta é uma porta bidirecional de 8 bits com “pullups” internos. Seus bits desempenham funções alternativas importantes descritas a posteriori e sumarizadas na Tabela 3.4.

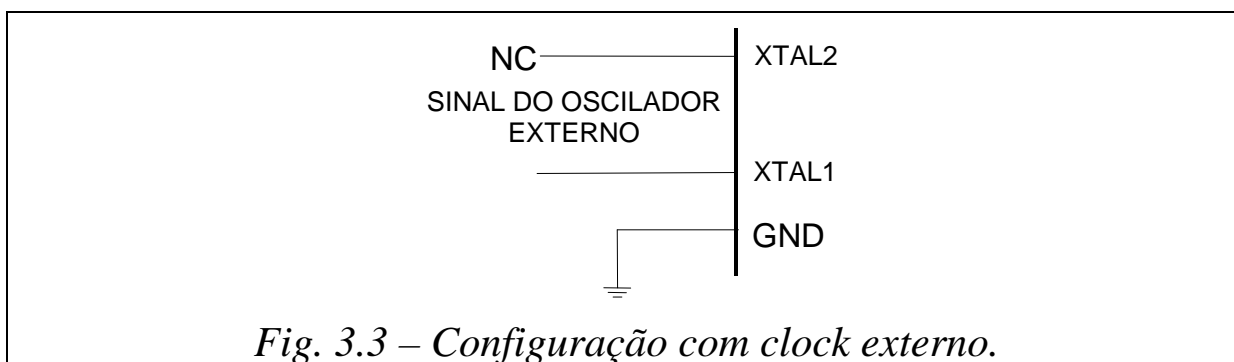
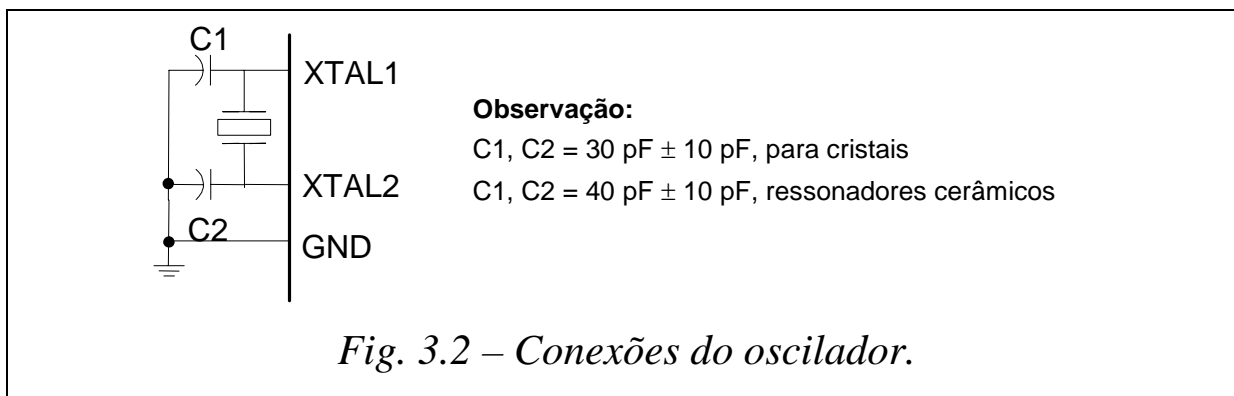
TABELA 3.4
PINOS DAS PORTAS DE E/S E SUAS FUNÇÕES
ALTERNATIVAS

Pino da Porta	Função Alternativa
P1.0	T2 - entrada de relógio externa do Timer 2
P1.1	T2EX – trigger de recarga/captura do Timer 2
P1.4	SS – Entrada de seleção de escravo
P1.5	MOSI – Saída de dados do mestre / entrada de dados do escravo para o canal SPI
P1.6	MISO – Entrada de dados do mestre / saída de dados do escravo para o canal SPI
P1.7	SCK – Saída de relógio do mestre / entrada de relógio do escravo para o canal SPI
P3.0	RXD – entrada de dados da porta serial
P3.1	TXD – saída de dados da porta serial
P3.2	$\overline{\text{INT0}}$ – Interrupção externa 0
P3.3	$\overline{\text{INT1}}$ – Interrupção externa 1
P3.4	T0 - entrada de clock externa do Timer 0
P3.5	T1 - entrada de clock externa do Timer 1
P3.6	$\overline{\text{WR}}$ – sinal de escrita em memória externa
P3.7	$\overline{\text{RD}}$ – sinal de leitura em memória externa

3.2 – Oscilador Interno:

Todo microcontrolador flash da Atmel tem um oscilador interno que pode ser usado como uma fonte de relógio para a CPU. Para usá-lo, devem ser conectados um cristal ou um ressonador cerâmico conectado entre os terminais XTAL1 e XTAL2, um capacitor entre XTAL1 e a referência e outro entre XTAL2 e a

referência, conforme mostra a Fig. 3.2. Para controlar o dispositivo a partir de uma fonte de relógio externa, XTAL2 deve ser mantido desconectado, conforme mostra a Fig. 3.3.



3.3 – Reset:

Um reset é obtido mantendo-se o terminal RST alto por pelo menos dois ciclos de máquina (24 ciclos do oscilador), enquanto o oscilador está operando. A CPU responde gerando um sinal de reset interno, cujo algoritmo escreve 0 lógico em todos os registradores de função especial (SFRs, Fig. 3.9) exceto os latches das portas (inicializados com 0FFh), o Stack Pointer (inicializado com 07h) e o SBUF (indeterminado). A RAM interna não é afetada pelo reset, que na energização do sistema apresenta conteúdo indeterminado.

O circuito de reset necessário para garantir um reset válido na energização do microcontrolador é mostrado na Fig. 3.4. Para os microcontroladores MCS-51™ de tecnologia CMOS, o resistor externo pode ser removido, pois o pino RST possui um resistor interno de *pull-down*, cujo valor se situa entre 50 k Ω e 300 k Ω no caso do AT89S8252. O valor do capacitor pode, então, ser reduzido para 1 μ F. Quando a energia é ligada, o circuito mantém o pino RST alto por um tempo que depende da constante de tempo do circuito RC. Para garantir um reset válido, o pino RST deve ser mantido alto o tempo suficiente para permitir a inicialização do oscilador mais dois ciclos de máquina. A inicialização do oscilador depende da frequência de operação, sendo que para um cristal de 10 MHz, o tempo de inicialização é tipicamente 1 ms.

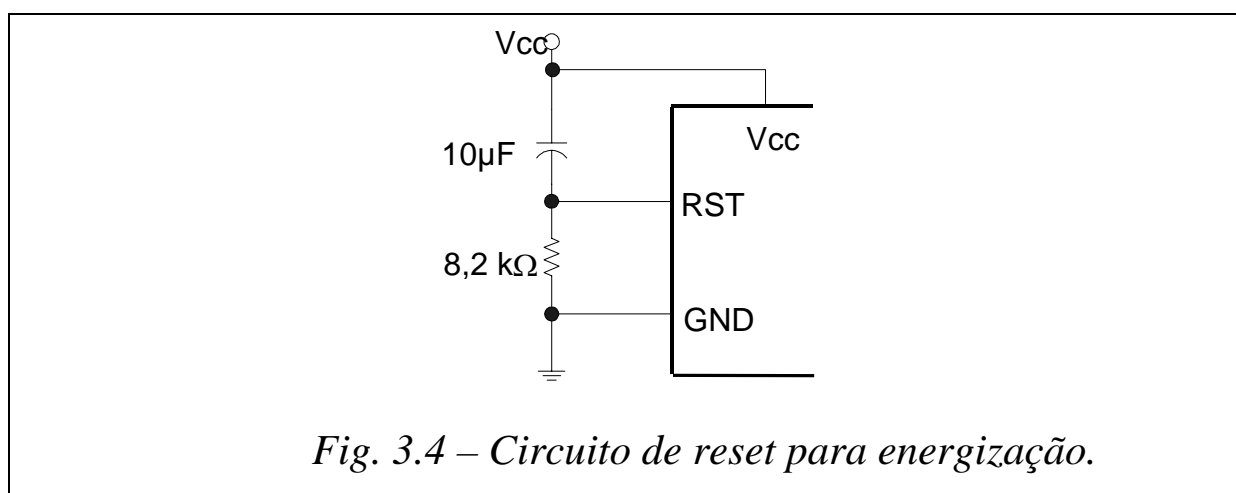


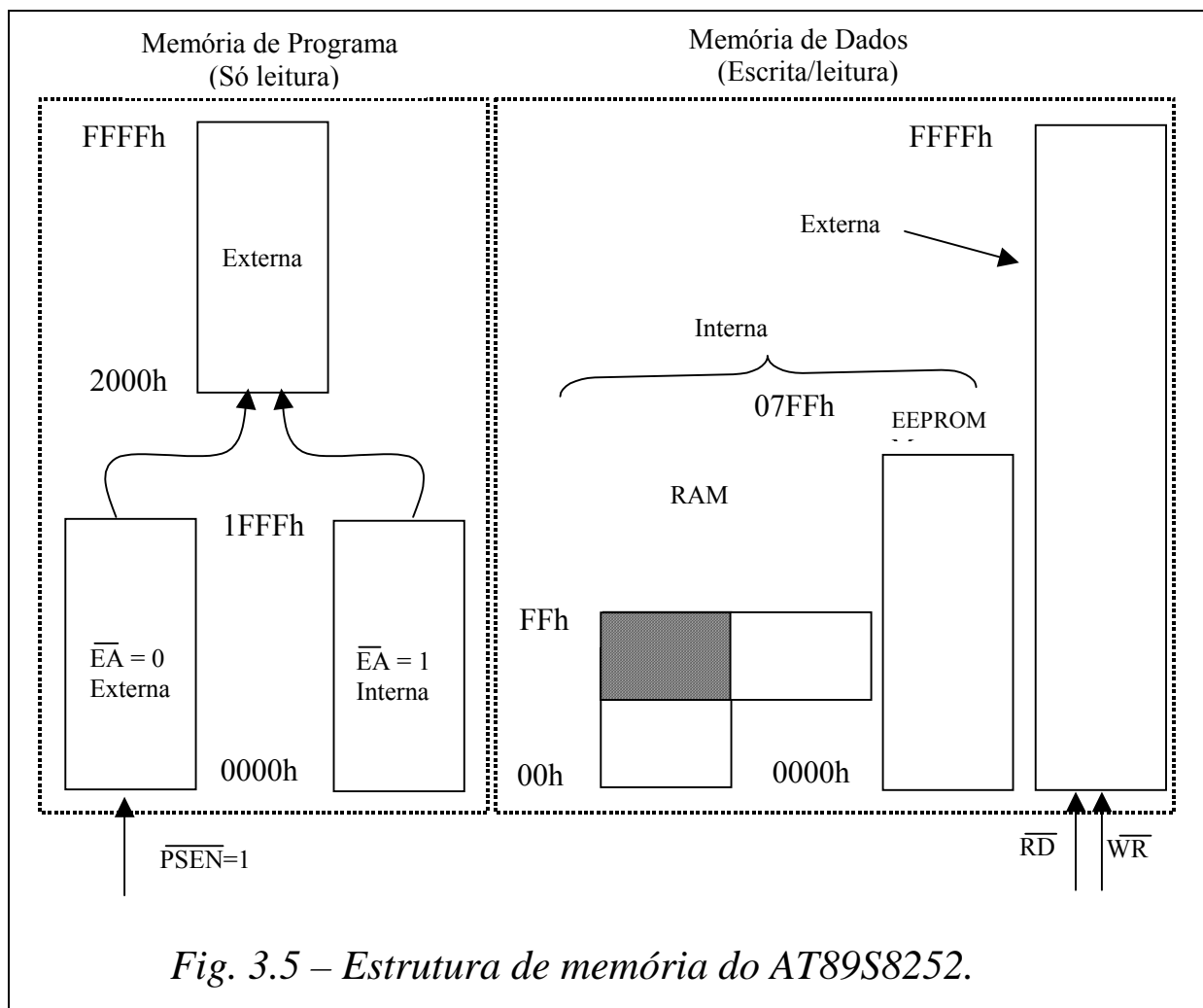
Fig. 3.4 – Circuito de reset para energização.

3.4 – Organização de Memória:

Todos os microcontroladores Flash Atmel tem endereços separados para memória de programa e memória de dados, conforme ilustra a Fig. 3.5. Essa separação lógica permite que a memória interna

volátil seja acessada por endereços de 8 bits, o que garante um acesso rápido à mesma. Os endereços de memória de 16 bits podem ser gerados através do registrador DPTR.

A memória de programa somente pode ser lida. A CPU pode acessar diretamente 64k bytes, sendo que os endereços menores (até 8k) podem ser acessados internamente fazendo $\overline{EA}=1$ ou externamente fazendo $\overline{EA}=0$.



Por exemplo, se o terminal \overline{EA} for conectado a V_{CC} , a CPU busca as instruções dos endereços 0000h a 1FFFh na flash interna e as instruções a partir de 2000h na memória externa. Se o terminal \overline{EA} for

conectado a GND, todo o programa é buscado na memória externa, sendo que a configuração de hardware para a execução de programa na mesma será discutida na seção 3.10.1.

A Fig. 3.6 apresenta um mapa detalhando os primeiros endereços da memória de programa. Após o *reset*, a CPU inicia a execução do programa a partir do endereço 0000h. Como ilustra esta figura, cada interrupção apresenta um endereço fixo de atendimento na memória de programa. As interrupções fazem com que a CPU desvie o processamento para um determinado endereço, onde ela executa uma determinada rotina. A Interrupção Externa 0, por exemplo, tem como endereço de atendimento a posição de memória 0003h. Assim, esse endereço, caso a mesma seja utilizada, deve conter a sua rotina de atendimento; caso ela não seja utilizada, esse endereço está disponível para o programador. Conforme ilustra a Fig. 3.6, para cada interrupção estão reservados 8 bytes. Se a rotina de alguma interrupção for muito extensa, para que ela não invada o espaço reservado a outra interrupção, seu endereço de atendimento deve conter um desvio para uma outra área de memória.

Na Fig. 3.5, à direita, mostra-se os espaços disponíveis, interna e externamente, para memória de dados no AT89S8252. A configuração de hardware para acessar dados externamente será, a exemplo do acesso a programa externo, discutida posteriormente.

O AT89S8252 possui internamente, para armazenar dados, 2k bytes de EEPROM e 256 bytes de RAM. A RAM interna, conforme ilustra a Fig. 3.7, é dividida em três blocos, a saber, os

primeiros 128 bytes, os últimos 128 bytes e o espaço destinado aos registradores de função especial. Os últimos 128 bytes de RAM ocupam um espaço em paralelo ao espaço destinado aos registradores de função especial, ou seja, eles são entidades fisicamente separadas. Quando uma instrução acessa uma locação interna acima do endereço 7Fh, o modo de endereçamento usado na instrução é que especifica se a CPU está acessando os últimos 128 bytes de RAM ou os referidos registradores.

Por exemplo, a instrução de endereçamento direto que segue, escreve 0Ah na porta 0, a qual é um dos registradores de função especial:

MOV 80h, #0Ah (MOV DIRETO, #DADO) (i)

Instruções que usam endereçamento indireto acessam os últimos 128 bytes da RAM. Por exemplo, as que seguem, escrevem 0Bh no endereço 80h da RAM:

MOV R0, #80h (MOV Rn, #DADO) (ii)

MOV @R0, #0Bh (MOV @Ri, #DADO) (iii)

Então, após a execução dessas instruções, a Porta 0 contém 0Ah e o endereço 80h da RAM contém 0Bh.

A instrução (iii) é o que denominamos de operação com ponteiro, ou seja, o registrador R0 indica a posição de memória onde será armazenado o dado. As operações com ponteiro são exemplos de endereçamento indireto.

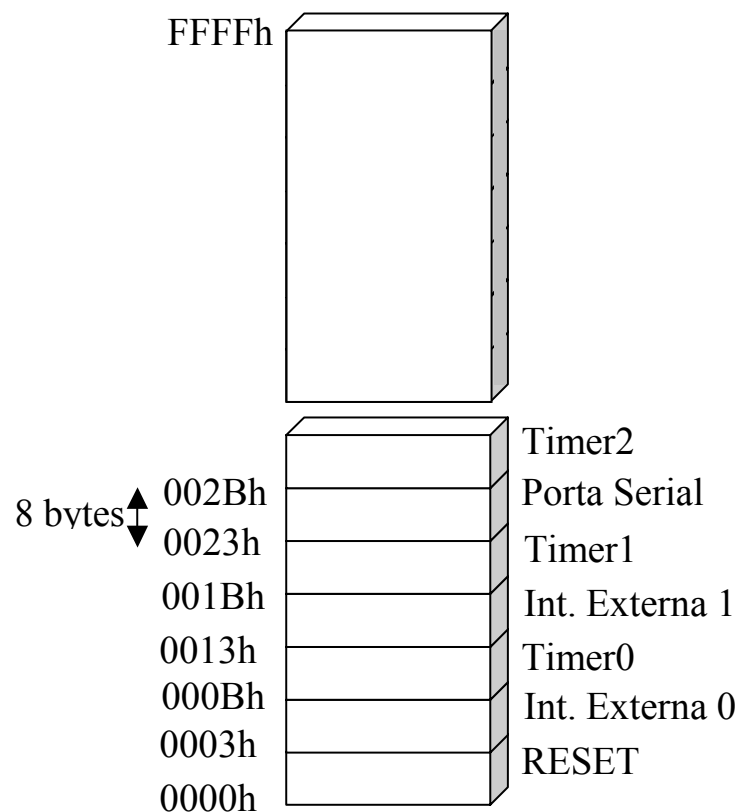


Fig. 3.6 – Os endereços mais baixos da memória de programa do AT89S8252.

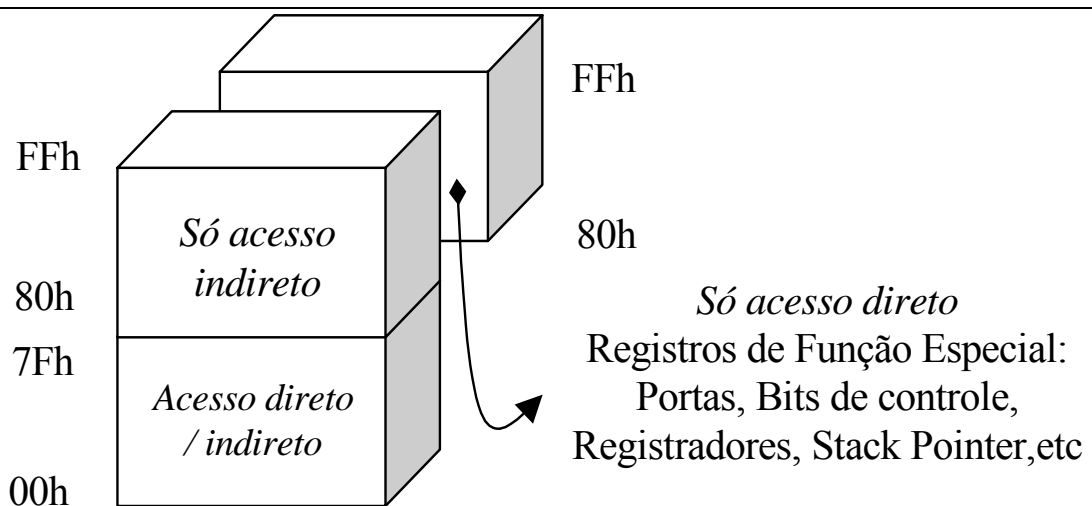


Fig. 3.7 – Memória RAM interna do AT89S8252.

A Fig. 3.8 mostra como os primeiros 128 bytes de RAM são mapeados. Esses bytes podem ser acessados por endereçamento direto

ou indireto, podendo ser divididos em 3 segmentos:

1. Banco de Registradores 0-3: Os primeiros 32 bytes (00h a 1Fh) são agrupados em 4 bancos de 8 registradores. As instruções de programa referenciam estes registradores como R0, R1, ..., R7. O banco padrão na inicialização (*reset*) é o Banco 0, sendo que para usar outro banco de registradores, o usuário deve selecioná-lo por software, conforme veremos mais adiante.

O *Stack Pointer* (SP) é um registrador residente na RAM internade 8 bits utilizado como ponteiro no controle do acesso de uma área de memória utilizada, geralmente denominada pilha, para armazenar valores temporários, como registros que serão alterados no acesso de subrotinas e endereços de retorno de sub-rotinas e de rotinas de atendimento de interrupção. Na família MCS-51TM e arquiteturas compatíveis, o *reset* inicializa o *Stack Pointer* (SP) no endereço 07h. Uma vez incrementado o SP aponta para o endereço 08h, o qual é o primeiro registrador (R0) do segundo banco de registradores. Então, para usar mais de um banco de registradores, o SP deve ser inicializado em um endereço da RAM que não será utilizado para armazenar dados, de preferência na área de rascunho.

2. Área endereçável por bit: Dezesseis bytes são reservados para endereçamento por bit – de 20h a 2Fh. Cada um desses

128 bits deste segmento podem ser diretamente endereçados. Esses bits podem ser referidos de duas maneiras. Um modo é referindo-se ao seu endereço, isto é, 0 a 7Fh. O outro modo é referindo-se aos bytes 20h a 2Fh. Por exemplo, a instrução “SETB 01h”, que é uma instrução de bit, coloca em nível lógico 1 apenas o bit 1 do endereço 20h, sem alterar os outros bits desse byte. O mesmo efeito pode ser produzido através do uso da instrução “ORL 20h,#00000010b” (OU LÓGICO entre a posição de memória 20h e a constante 00000010b), a qual é uma instrução de byte (lembre que “0 ou $\varphi = \varphi$ ” e “1 ou $\varphi = 1$ ”).

3. Área de rascunho. Os bytes 30h até 7Fh, são disponíveis para usar como uma RAM de dados de uso geral.

A Fig 3.9 oferece uma pequena visão da área de registradores de funções especiais. Essa área inclui os *latches* das portas, temporizadores, controles de periféricos, acumulador, etc. Estes registros somente podem ser acessados por endereçamento direto. Em geral, todos os microcontroladores da Atmel possuem, nesse espaço, os mesmos registradores nos mesmos endereços dos microcontroladores compatíveis com a família MCS-51™ da Intel. Entretanto, o AT89S8252 possui registradores adicionais, que não estão presentes nos microcontroladores com núcleo C51, os quais destacamos, por contraste, na Fig. 3.9. Os endereços que apresentam números no espaço de registradores de função especial (SFR) são endereçáveis por bit, conforme destaca a mesma figura.

7Fh		<div>Área de rascunho</div> <div>ou</div> <div>de propósito geral</div>							
ÁREA BIT END.	30h								
	2Fh	7F	7E	7D	7C	7B	7A	79	78
	2Eh	77	76	75	74	73	72	71	70
	2Dh	6F	6E	6D	6C	6B	6A	69	68
	2Ch	67	66	65	64	63	62	61	60
	2Bh	5F	5E	5D	5C	5B	5A	59	58
	2Ah	57	56	55	54	53	52	51	50
	29h	4F	4E	4D	4C	4B	4A	49	48
	28h	47	46	45	44	43	42	41	40
	27h	3F	3E	3D	3C	3B	3A	39	38
	26h	37	36	35	34	33	32	31	30
	25h	2F	2E	2D	2C	2B	2A	29	28
	24h	27	26	25	24	23	22	21	20
	23h	1F	1E	1D	1C	1B	1A	19	18
	22h	17	16	15	14	13	12	11	10
	21h	0F	0E	0D	0C	0B	0A	09	08
20h	07	06	05	04	03	02	01	00	
1Fh		Banco 3							
18h									
17h		Banco 2							
10h									
0Fh		Banco 1							
08h									
07h		Banco 0 – Padrão para R0 – R7							
00h									

Fig. 3.8 – Os primeiros 128 bytes da RAM interna.

Endereço do byte em h	Endereço do bit em h								Nome do registrador
FF									B
F0	F7	F6	F5	F4	F3	F2	F1	F0	
E0	E7	E6	E5	E4	E3	E2	E1	E0	A
D5	DF	DE	DD	DC	DB	DA	D9	D8	
D0	D7	D6	D5	D4	D3	D2	D1	D0	PSW
CD	não é endereçável por bit								
CC	não é endereçável por bit								
CB	não é endereçável por bit								
CA	não é endereçável por bit								
C9	não é endereçável por bit								
C8	C7	C6	C5	C4	C3	C2	C1	C0	
B8	-	-	BD	BC	BB	BA	B9	B8	IP
B0	B7	B6	B5	B4	B3	B2	B1	B0	
AA	não é endereçável por bit								SPSR
A8	AF	-	AD	AC	AB	AA	A9	A8	
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
99	não é endereçável por bit								
98	9F	9E	9D	9C	9B	9A	99	98	SBUF SCON
96	não é endereçável por bit								
90	97	96	95	94	93	92	91	90	P1
8D	não é bit endereçável								
8C	não é bit endereçável								TH1 TH0 TL1 TL0
8B	não é bit endereçável								
8A	não é bit endereçável								TMOD TCON
89	não é bit endereçável								
88	8F	8E	8D	8C	8B	8A	89	88	PCON
87	não é bit endereçável								
86	não é bit endereçável								SPDR DP1H DP1L
85	não é bit endereçável								
84	não é bit endereçável								DP0H DP0L
83	não é bit endereçável								
82	não é bit endereçável								SP P0
81	não é bit endereçável								
80	87	86	85	84	83	82	81	80	

Fig. 3.9 – Registradores de Função Especial.

As funções dos *SFRs* são resumidas a seguir.

- Acumulador (ACC ou A) – O Acumulador participa de todas as operações lógicas e aritméticas e é o único registrador que pode ter seus bits rotacionados.
- Registrador B – O registrador B é utilizado nas operações de multiplicação e divisão. Para outras instruções ele pode ser tratado como um registrador de rascunho;
- Palavra de Estado do Programa (*PSW*) – O *Program Status Word* contém as informações sobre o andamento do programa;
- Ponteiro de Pilha (SP) – O *Stack Pointer* já foi apresentado nessa seção;
- Ponteiro de Dados Dual – Para facilitar o acesso a EEPROM interna e a memória de dados externa, dois bancos de Registradores Ponteiro de Dados (*Data Pointer* ou DPTR) estão disponíveis: DP0 nos endereços 82h e 83h da área SFRs e DP1 nos endereços 82h e 83h. Se o bit DPS do registrador WMCON (Fig. 3.10) estiver em 0 lógico (situação no reset) DP0 está ativo; se DPS for posto em 1 lógico, seleciona-se DP1. Os *Data Pointer* são registradores formados por um byte alto (DP0H/DP1X) e por um byte baixo (DP0L/DP1L), que têm como função guardar endereços de 16 bits. Cada

banco pode ser manipulado como um registrador de 16 bits ou como dois registradores de 8 bits independentes;

- P0, P1, P2 e P3 – São os *latches* das portas de E/S;
- Buffer de Dados Serial (*SBUF*)- O *Serial Data Buffer* é, na realidade, dois registradores separados, um para transmissão e outro para a recepção de dados;
- Registradores dos *Timers* – Os pares de registradores (TH0, TL0), (TH1, TL1) e (TH2, TL2) são, respectivamente, os registradores de 16 bits para os temporizadores/contadores 0, 1 e 2;
- Registradores de captura - Os pares de registradores (RCAP2H, RCAP2L) são os registradores de captura para o Modo de Captura do Timer 2;
- Registrador de Controle de Memória e do Cão de Guarda (WMCON) – O registrador *Watchdog and Memory Control* contém bits de controle para o temporizador Cão de Guarda, para o acesso da memória EEPROM interna de 2k bytes e o bit DPS de seleção de um dos dois registradores DPTR.
- Registradores da Interface Serial Periférica (SPI) – Os bits de estado e de controle da Serial Peripheral Interface estão localizados nos registradores SPCR (*SPI Control Register*) e SPDR (*SPI Status Register*);

- Registradores de Controle – Os registradores de Função Especial IP, IE, TMOD, TCON, T2MOD, T2CON, SCON e PCON contém os bits de controle e de estado para o sistema de interrupção, para os timers e para o canal serial.

A memória de dados EEPROM interna ao CI é selecionada colocando-se em nível lógico 1 o bit 3 (EEMEN - *Internal EEPROM Access Enable*) do registrador WMCON (Fig. 3.10) no endereço 96h da área *SFRs*. Para acessar a EEPROM devem ser utilizadas as instruções MOVX; por outro lado, para acessar a memória de dados externa com as instruções MOVX, o bit EEMEN deve estar em nível lógico 0, o qual é seu estado na inicialização do sistema. O bit 4 (EEMWE - *EEPROM Data Memory Write Enable Bit*) do registrador WMCON deve ser colocado em nível lógico 1 antes de se escrever qualquer byte na EEPROM e, se nenhuma operação de escrita for mais necessária, o programador deve colocar esse bit em nível lógico 0. O progresso de escrita na EEPROM pode ser monitorada pela leitura do bit RDY/ $\overline{\text{BSY}}$. Um 0 lógico em RDY/ $\overline{\text{BSY}}$, significa que a escrita ainda está em progresso; o nível lógico 1 neste pino, por sua vez, significa que o ciclo de escrita da EEPROM está completo e outro ciclo pode ser iniciado. Uma vez que está relacionada com o uso do temporizador Cão de Guarda, a função dos demais bits do registrador WMCON será estudada na seção 3.7.3. A Tabela 3.5 seguinte resume a descrição feita nesse parágrafo.

Registrador WMCON – endereço 96h ; **Valor na inicialização** = 00000010b

	PS2	PS1	PS0	EEMWE	EEMEN	DPS	WDTRST	WDTEN
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
PSx	Bits de Prescaler do Cão de Guarda. Quando todos os 3 bits estão em 1 lógico, o cão de guarda apresenta um período nominal de 16 ms, quando todos estão em 1 lógico, o período nominal é 2,048 s.							
EEMWE	O bit que habilita a escrita na memória EEPROM. Este bit deve ser colocado em nível lógico 1 antes da operação de escrita na EEPROM interna com a instrução MOVX e deve ser colocado em 0 lógico após a operação de escrita se completar.							
EEMEN	Habilita o acesso a memória EEPROM interna. Se EEMEN = 1, a instruções MOVX com DPTR acessam a EEPROM interna ao invés de um dispositivo externo; se EEMEN = 0, a instruções MOVX com DPTR acessam memória de dados externa.							
DPS	Seletor do registrador <i>Data Pointer</i> . DPS = 0 seleciona o primeiro banco de registradores <i>Data Pointer</i> , DP0; DPS = 1 seleciona o segundo banco, DP1.							
WDTRST RDY/ BSY	Bit de reset do Cão de Guarda e Flag Pronta/Ocupada da EEPROM. Cada vez que o WDTRST é colocado em 1 lógico pelo software, é gerado um pulso para inicializar o Cão de Guarda, sendo que no ciclo de instrução seguinte ele é automaticamente posto em 0 lógico. O bit WDTRST é somente de escrita. Este bit também serve como um flag (somente leitura) para indicar o estado da EEPROM durante a escrita. RDY/BSY = 1 significa que a EEPROM está pronta para ser programada; enquanto as operações de programação estão em execução, o bit RDY/BSY está em 0 lógico, sendo automaticamente posto em 1 lógico quando a programação se completa.							
WDTEN	Bit que habilita o temporizador Cão de Guarda. WDTEN = 1 habilita o Cão de Guarda; WDTEN = 0, deshabilita.							

Fig. 3.10 – Registrador WMCON no AT89S8252.

TABELA 3.5
ACESSO À MEMÓRIA DE DADOS

Operação	EEMEM	EEMWE
Acesso à memória de dados externa	0	φ^2
Leitura na memória EEPROM interna	1	0
Escrita na memória EEPROM interna	1	1

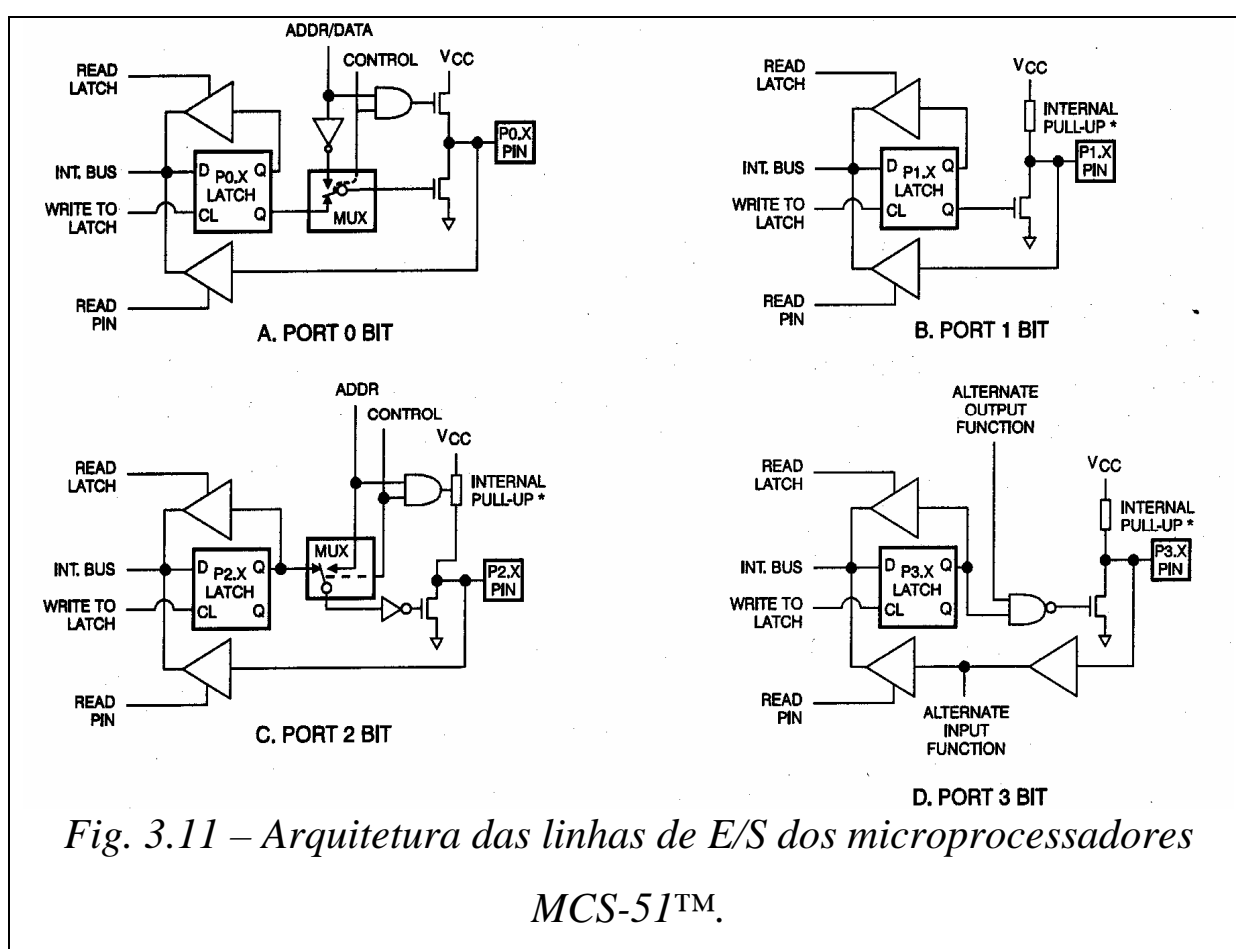
3.5 – Estrutura e operação da portas de E/S:

A Fig 3.11 apresenta a arquitetura interna de uma linha de cada uma das quatro portas de E/S dos microprocessadores MCS-51™ compatíveis. Cada bit de uma porta na área de registradores de função especial é representado como um flip-flop tipo D, o qual armazena um valor do barramento interno em resposta a um sinal de escrita no *latch* (write to latch) vindo da CPU. A saída Q do flip-flop é colocada no barramento em resposta a um sinal de leitura do *latch* (read latch) vindo da CPU, enquanto que o nível do pino da porta é colocado no barramento em resposta a um sinal de leitura do pino. Algumas instruções que lêem uma porta ativam o sinal de leitura do latch e outras ativam o sinal de leitura do pino.

A porta P0 apresenta saídas em dreno aberto (O FET de *pullup* na porta P0 é usado somente quando ela emite nível lógico 1 durante o acesso a memória externa) e as portas P1, P2 e P3 têm *pullups* internos, o que as leva a serem chamadas, algumas vezes, de portas quase-bidirecionais. Quando o FET é colocado em corte pelo

² Indica estado irrelevante.

latch, a saída apresenta nível lógico alto. Quando o FET conduz, ele força a saída para nível lógico baixo. O *pullup* interno das portas P1, P2 e P3 é, na realidade, uma configuração de três FETs, onde um deles gera uma corrente interna de pequena intensidade e, com o objetivo de incrementar a velocidade da transição de 0 para 1, um segundo FET gera uma corrente cerca de 100 vezes maior que a do primeiro durante um curto intervalo de tempo.



Para ilustrar o funcionamento das portas, elaboramos um programa que gera uma onda quadrada de 2 kHz na linha P1.0, cujo fluxograma é apresentado na Fig. 3.12. No fluxograma da direita expandimos a caixa “atrasa 250 μ s”, que passou a ser representada por

outras três. Na listagem obtida a partir dos fluxogramas, apresentamos, ao lado dos comentários, o número de ciclos de relógio que a CPU leva para executar cada instrução. Cada ciclo de instrução (ciclo de máquina) do AT89S8252 necessita de 12 ciclos de relógios para ser executado, sendo que a maior parte das instruções são executadas em um ou dois ciclos de máquina. Se utilizarmos um cristal com frequência de 11,0592 MHz ($T = 90,42 \text{ ns}$), um ciclo de máquina leva $1,085 \mu\text{s}$ ($12 \times 90,42 \text{ ns}$) para ser executado.

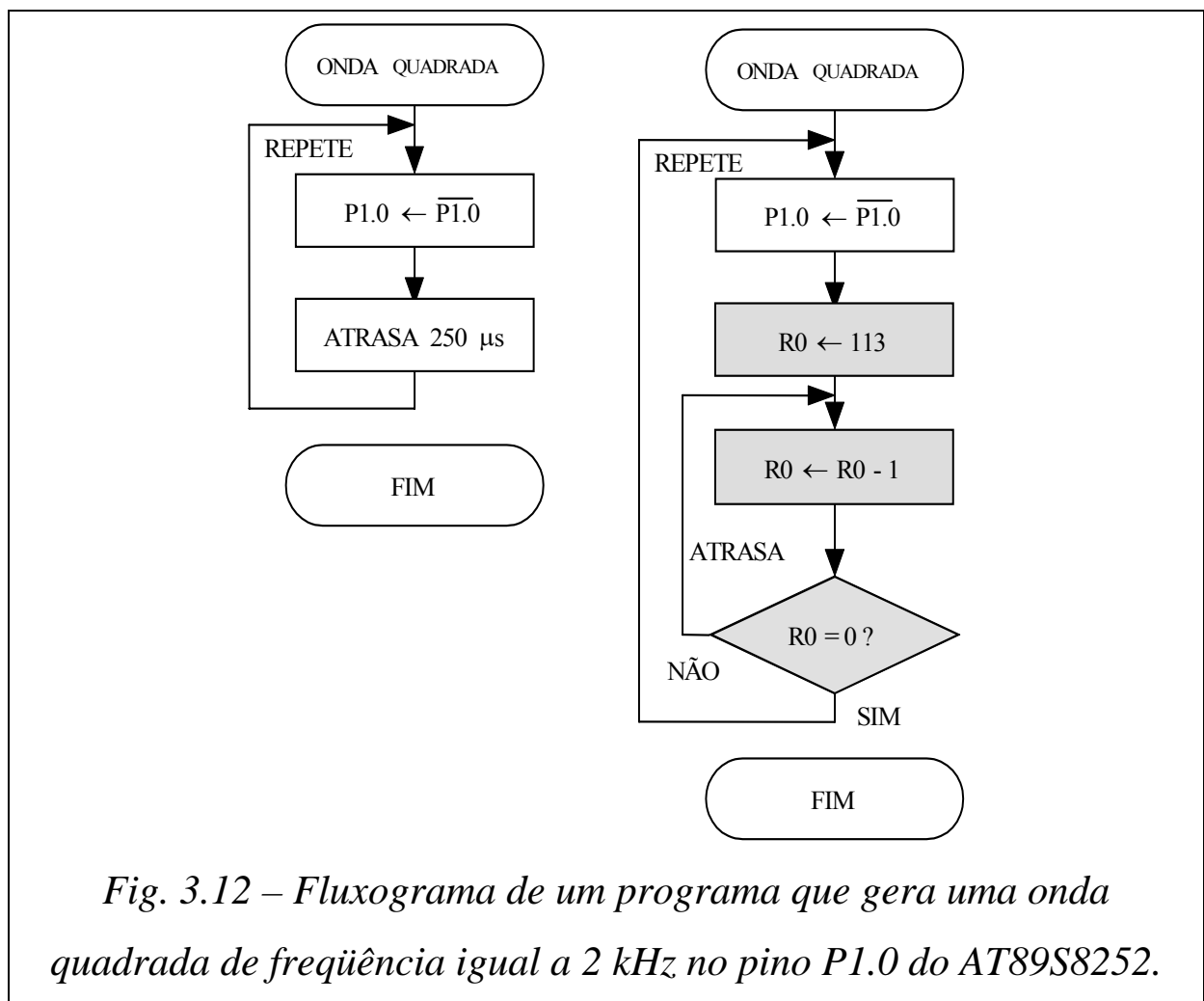


Fig. 3.12 – Fluxograma de um programa que gera uma onda quadrada de frequência igual a 2 kHz no pino P1.0 do AT89S8252.

Listagem em assembly do programa da Fig. 3.12

```

ORG3 00h      ; Define onde o código
                subsequente será escrito

repete: CPL P1.0    ; Complementa P1.0          (1,085 µs)

                MOV R0,#113    ; coloca o valor 113 decimal (2,170 µs)
                                no registrador R0
atrasa: DJNZ R0, atrasa ; Enquanto R0 não for zero (2,170 µs)
                                permanece nessa linha

                SJMP repetec    ; retorna para “repete”      (2,170 µs)

END            ; fim de compilação

```

Como a instrução “*DJNZ R0, atrasa*” é executada 113 vezes, o tempo total de execução da rotina é de 250,6 µs (1,085 µs + 2,170 µs + 113 x 2,170 µs + 2,170 µs). Podemos, para a rotina acima, estabelecer uma relação entre o número de vezes que a instrução “*DJNZ R0, atrasa*” deve ser executada e o atraso desejado através da seguinte expressão:

$$N^{\circ} \text{ de repetições} = (\text{atraso} / \text{tempo de 2 ciclos de máquina}) - 2,5$$

onde : $1 \leq n^{\circ} \text{ de repetições} \leq 256$

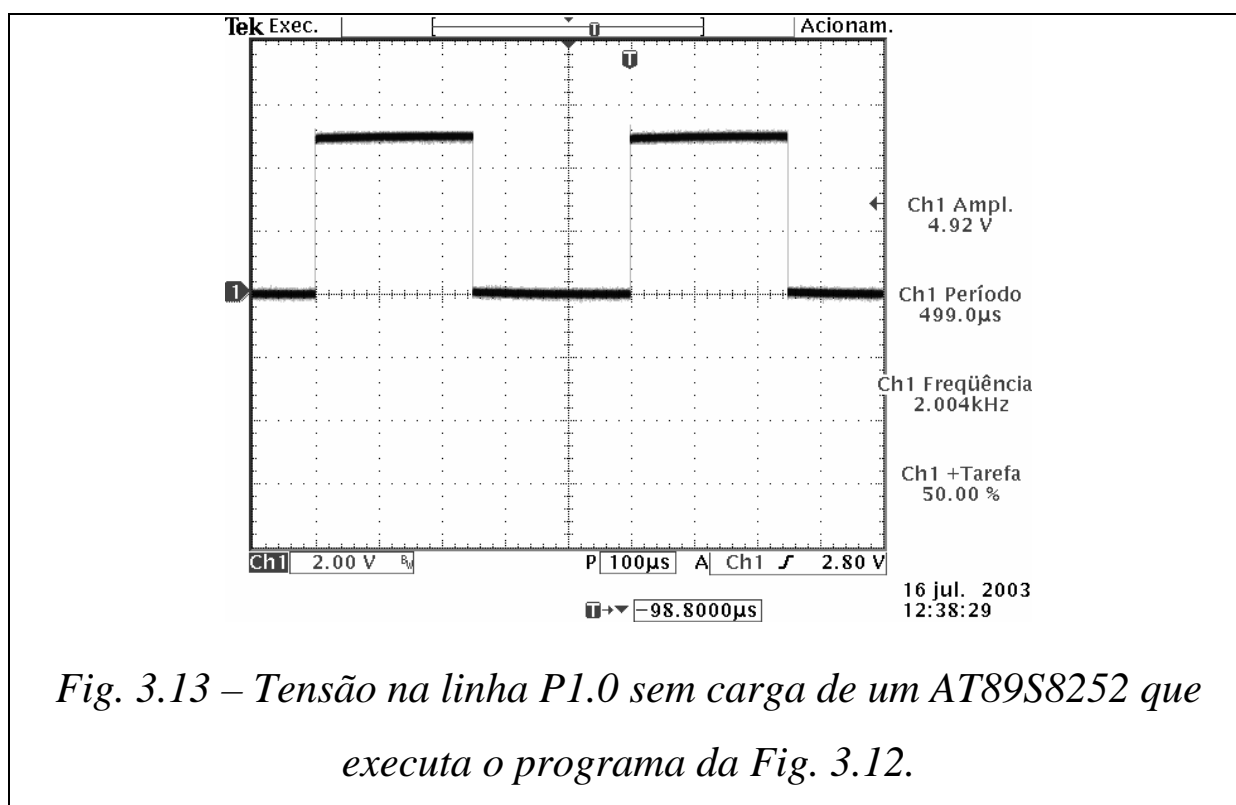
No nosso exemplo:

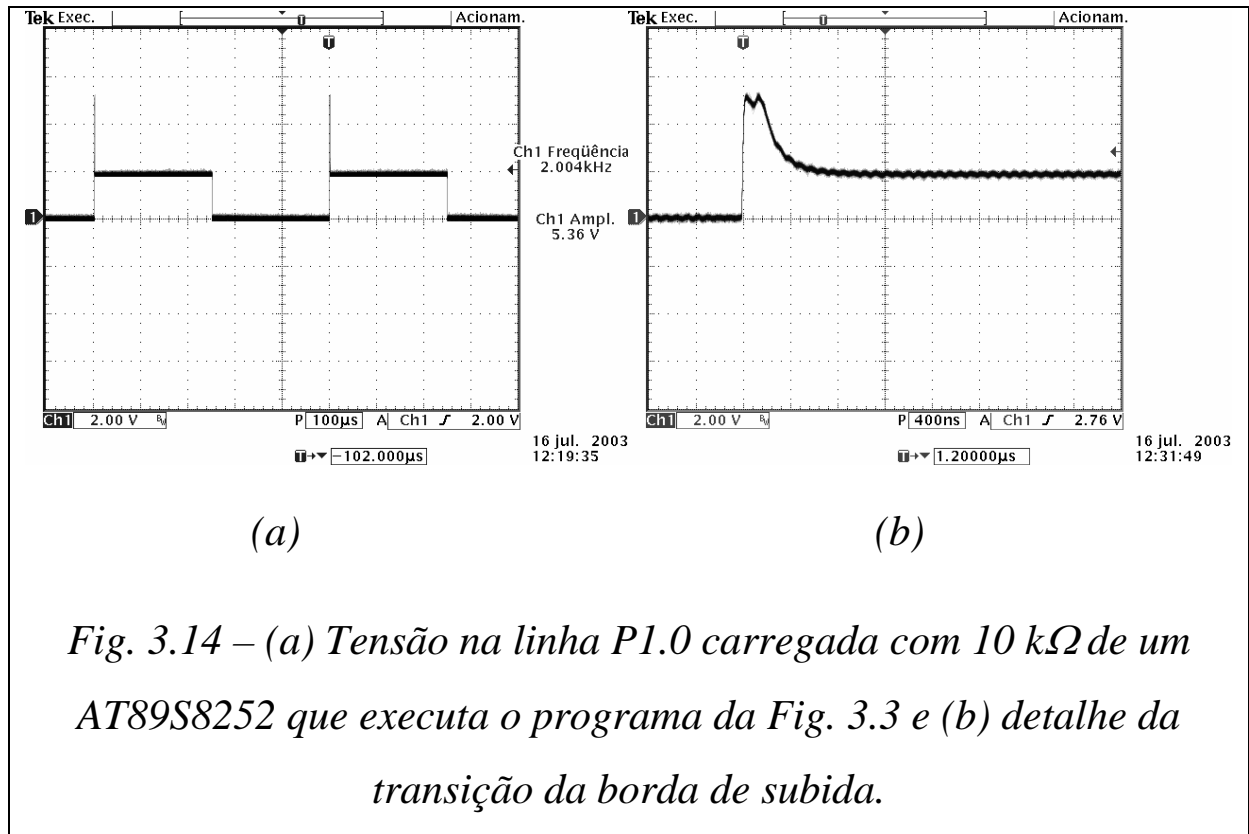
³ ORG e END não são instruções da família 8051, elas são diretivas do *Assembler*. As diretivas permitem definir símbolos, reservar e inicializar espaço de memória e estabelecer endereços específicos para trechos de programas. As diretivas suportadas pelo *Assembler A51* são apresentadas no Anexo III.

$$N^{\circ} \text{ de repetições} = (250 \mu\text{s} / 2,170 \mu\text{s}) - 2,5 \cong 113$$

É importante salientar que com essa rotina e um sinal de relógio de 11,0592 MHz, o atraso máximo obtido é de 560,9 μs ($258,5 \times 2,170 \mu\text{s}$).

Nas Fig. 3.13 e 3.14 apresenta-se a tensão presente no pino P1.0 de um AT89S8252 que executa o programa anterior. Na Fig. 3.13 o pino P1.0 encontra-se aberto e na Fig. 3.14 há um resistor de 10 k Ω conectado entre P1.0 e a referência (GND). Na Fig. 3.13 percebe-se que o microprocessador consegue manter a tensão de 5 V (Vcc) em P1.0 apenas por um curto espaço de tempo, ou seja, durante a ação do FET de comutação. Após, a tensão em P1.0 cai para 2 V, o que indica que, para essa situação de carga, P1.0 fornece uma corrente constante de 200 μA em regime permanente.





Uma corrente de cerca de 200 μA é insuficiente para excitar um LED ligado entre o terminal de E/S e o GND. Entretanto, no AT89S8252, cada terminal de uma porta pode drenar até 10mA, sendo que a Porta 0, no conjunto de seus 8 bits, pode drenar até 26 mA e as outras até 15 mA cada, totalizando um máximo de 71 mA para todos os pinos de saída. Assim, uma linha de uma porta pode excitar diretamente um LED se ele for ligado entre a linha de alimentação Vcc em série com um resistor de valor adequado, conforme ilustra a Fig. 3.15. Utilizando essa configuração, o LED seria excitado quando o pino de E/S fosse colocado em nível lógico zero, o que poderia ser feito empregando-se uma instrução “CLR P1.x”.

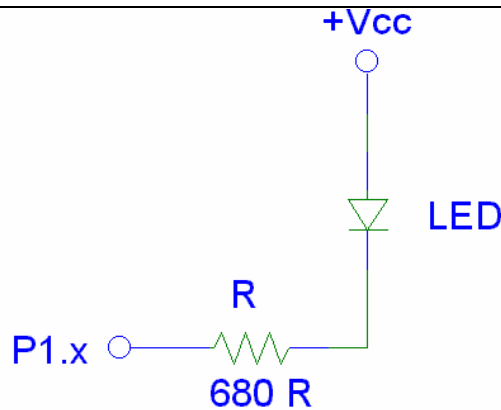


Fig. 3.15 – Linha de entrada e saída acionando um LED.

Verifique seu aprendizado

Elabore um programa que gere uma onda retangular de 1 kHz com razão cíclica de 40% (Tempo de nível alto = 400 μ s) na linha P2.7.

Conforme já foi dito, algumas instruções que lêem a porta lêem o *latch*, outras o pino. As instruções que lêem o *latch* são chamadas instruções lê-modifica-escreve; elas lêem um valor, possivelmente o alteram e então o reescrevem no *latch*. Quando o operando de destino é uma porta ou um bit seu, as instruções lê-modifica-escreve apresentadas na Tabela 3.6 lêem o *latch* ao invés do pino.

As instruções lê-modifica-escreve são direcionadas para o latch e não para o pino para evitar uma leitura incorreta do nível lógico no pino. Por exemplo, quando um bit de uma porta é ligado a base de um transistor, ao escrevermos 1 lógico no bit o transistor entra em condução. Se a CPU ler este bit no pino e não no *latch*, ela lerá a tensão na base do transistor (cerca de 0,7 V) e interpretará o bit como 0 lógico. Lendo o latch, o valor retornado será o valor correto, no caso, 1 lógico.

TABELA 3.6
INSTRUÇÕES DO TIPO LÊ-MODIFICA-ESCREVE

Mnemônico	Instrução	Exemplo
ANL	E lógico	ANL P2, #0Fh
ORL	Ou lógico	ORL P2, #0Fh
XRL	Ou exclusivo lógico	XRL P2, #0Fh
JBC	Desvia se bit = 1 e coloca 0 lógico no bit	JBC P1.0, SALTA
CPL	Complementa o bit	CPL P2.5
INC	Soma 1	INC P2
DEC	Diminui de 1	DEC P3
DJNZ	Decrementa e desvia se não zero	DJNZ P2, SALTA
MOV PX.Y, C	Transfere o valor do bit de Carry (vai um) para o bit Y da porta X	MOV P3.5, C
CLR PX.Y	Coloca em 0 lógico o bit Y da porta x	CLR P2.7
SETB PX.Y	Coloca em 1 lógico o bit Y da porta x	SETB P2.7

3.6 – O Conjunto de Instruções dos Microcontroladores MCS-51TM compatíveis

3.6.1 – A Palavra de Estado do Programa (*Program Status Word -PSW*)

O registrador PSW, descrito na Fig. 3.16, contém bits que refletem a situação atual da CPU. Ele contém o bit de carry, o carry auxiliar, os dois bits que selecionam o banco de registradores, o flag de overflow, o bit de paridade e dois flags definíveis pelo usuário.

Para ilustrar o mecanismo do registrador PSW, analisaremos o trecho de um programa que soma o conteúdo dos registradores R0 do banco 0 (posição de memória 00h) e do banco 2 (posição de memória 10h) e coloca o resultado no registrador R0 do banco 3 (posição de memória 18h), onde a posição 00h da RAM contém 9Ah e a posição 10h contém 0A9h.

```
MOV A, R0;  $A \leftarrow R0_0$  ( $A \leftarrow 9Ah$ )  
SETB RS1      ; Seleciona-se o banco 2  
ADD A, R0      ;  $A \leftarrow A + R0_2$  ( $A \leftarrow 9Ah + 0A9h$ ;  $A \leftarrow 33h$ )  
SETB RS0      ; Seleciona banco 3  
MOV R0, A      ;  $R0_3 \leftarrow A$  ( $R0_3 \leftarrow 33h$ )
```

Registrador PSW – endereço 0D0h ; **Valor na inicialização** = 00000000b

	CY	AC	F0	RS1	RSO	OV	?	P															
bit	7	6	5	4	3	2	1	0															
Símbolo	Função																						
CY	O Flag de carry recebe o “vai um” / “empresta um” resultante das operações aritméticas. É usado, também, em operações lógicas																						
AC	O Flag de carry auxiliar recebe o “vai um” do bit 3 do resultado das operações aritméticas; é usado em operações de ajuste decimal.																						
F0	Flag de propósito geral, ajustado por software.																						
RS1/RSO	Selecionam um dos 4 bancos de registradores, conforme tabela abaixo: <table><tr><th>RS1</th><th>RSO</th><th>Banco</th></tr><tr><td>0</td><td>0</td><td>Banco 0</td></tr><tr><td>0</td><td>1</td><td>Banco 1</td></tr><tr><td>1</td><td>0</td><td>Banco 2</td></tr><tr><td>1</td><td>1</td><td>Banco 3</td></tr></table>								RS1	RSO	Banco	0	0	Banco 0	0	1	Banco 1	1	0	Banco 2	1	1	Banco 3
									RS1	RSO	Banco												
									0	0	Banco 0												
									0	1	Banco 1												
									1	0	Banco 2												
1	1	Banco 3																					
OV	Útil para representações binárias com sinal, ou seja, em uma escala de −128 a +127. Quando OV = 1, significa que o resultado não pode ser representado nessa escala. Por exemplo, quando o resultado da soma de 2 números positivos é negativo.																						
?	Flag definido pelo usuário.																						
P	Paridade do acumulador, reflete o número de bits em nível lógico um do acumulador: Se P = 1, o acumulador contém um número ímpar de 1s, se P = 0, ele contém um número par de 1s.																						

Fig. 3.16 – Registrador PSW no AT89S8252.

O esquema a seguir, que representa a operação *ADD A,R0* , nos ajuda a compreender o estado do registrador PSW após a execução do trecho de programa acima:

C	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
	1	0	0	¹ 0	1	0	1	0	A
	1	0	1	0	1	0	0	1	R0 ₂
1	0	0	1	1	0	0	1	1	A + R0 ₂

Do esquema, pode-se auferir que após a execução desse trecho do programa a conteúdo do registrador PSW será:

PSW	C	AC	F0	RS1	RS0	OV	?	P
	1	1	φ	1	0	1	1	0

O *flag OV* foi setado porque, no caso da representação binária com sinal, a soma de dois números negativos resultou em um número positivo. O *flag P* foi resetado porque o acumulador contém um número par de '1s' (4).

3.6.2 – Modos de endereçamento

a. Endereçamento direto

No endereçamento direto, o operando é especificado por um campo de endereço de 8 bits na instrução. Somente a memória RAM de dados interna e os SFRs podem ser diretamente acessados.

Exemplo:

MOV A, 50h. Após a execução dessa instrução, o conteúdo da posição de memória de endereço 50h será copiado no acumulador. Simbolicamente: $(A) \leftarrow (50h)$

b. Endereçamento indireto

No endereçamento indireto, a instrução especifica um registrador que contém o endereço do operando. Tanto a RAM interna como a externa podem ser endereçadas indiretamente. Os registradores para endereçamento de 8 bits podem ser o SP (*Stack Pointer*), R0 ou R1. O registrador para endereçamento de 16 bits é unicamente o DPTR (data pointer register).

Exemplos:

1. MOV @R0, A. Após a execução dessa instrução, o conteúdo do acumulador será copiado na posição de memória cujo endereço é apontado por R0. Simbolicamente: $((R0)) \leftarrow (A)$

2. –PUSH ACC. Após a execução dessa instrução, o conteúdo do acumulador é copiado na posição de memória endereçada pelo SP e o SP é incrementado de 1. Simbolicamente:

$$(SP) \leftarrow (SP) + 1$$

$$((SP)) \leftarrow (A)$$

c. Constantes imediatas

O valor de uma constante pode compor uma instrução. Por exemplo, a instrução MOV A,#64h carrega o acumulador com o

número hexadecimal 64. O mesmo número poderia ser especificado na forma decimal como 100 ou na forma binária como 01100100b.

d. Endereçamento indexado

Pode-se ler tabelas na memória de programa através do endereçamento indexado. Um registrador de 16 bits usado como base, o DPTR ou o PC (*Program Counter*), aponta para o início da tabela e o acumulador é utilizado para especificar um elemento específico da tabela, ou seja, cada elemento buscado na tabela residente na memória de programa é formado adicionando o conteúdo do acumulador ao conteúdo do ponteiro.

3.6.3 - Comentário sobre algumas instruções:

a. DA A - Ajuste decimal do acumulador na adição:

Considere a adição abaixo representada de 08h com 09h.

C	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
	0	0	0	¹ 0	1	0	0	0	08h
	0	0	0	0	1	0	0	1	+ 09h
1	0	0	0	1	0	0	0	1	11h

Caso esses números estivessem representando números no formato BCD o resultado correto seria 17 (8 + 9) ao invés de 11h. A execução da instrução DA A, após uma instrução ADD ou ADDC,

converterá esse resultado para o formato BCD, somando 06h ao primeiro *nibble*⁴ do acumulador:

C	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
	0	0	0	1	0	0	0	1	11h
	0	0	0	0	0	1	1	0	+ 06h
0	0	0	0	1	0	1	1	1	17h

Nesse caso, o ajuste ocorre porque, após a operação de adição, o *flag AC* foi para '1'. Em linhas gerais, sempre que:

- Após uma adição, o primeiro *nibble* do acumulador for maior que 1001b ou $AC = 1$, a instrução *DA A* adiciona seis ao primeiro *nibble* do acumulador, proporcionando o número BCD apropriado;
- Após uma adição, o segundo *nibble* do acumulador for maior que 1001b ou $C = 1$, a instrução *DA A* adiciona seis ao segundo *nibble* do acumulador, proporcionando o número BCD apropriado.

Para reforçar, imaginemos um exemplo onde, representando números BCD, o acumulador contém 56h e o registrador R_2 contém 67h. O resultado da execução da instrução *ADD A, R2* é 0BDh, conforme ilustra o esquema a seguir:

⁴ Conjunto de 4 bits; decomposição de um byte em duas partes.

C	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
	¹ 0	1	0	1	¹ 0	¹ 1	1	0	A
	0	1	1	0	0	1	1	1	R2
0	1	0	1	1	1	1	0	1	A + R2

Como A_{3-0} e A_{7-4} são maiores que 1001h, uma posterior execução da instrução DA A, ajusta o resultado para 23h, os dois primeiros dígitos resultantes da soma de 56 com 67 (123). O esquema a seguir ilustra a operação de ajuste decimal:

C	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
	¹ 1	¹ 0	¹ 1	¹ 1	¹ 1	1	0	1	0BDh
	0	1	1	0	0	1	1	0	+ 66h
1	0	0	1	0	0	0	1	1	123h

A instrução DA A faz parte do grupo das **Instruções Aritméticas**, apresentadas em sua totalidade no Anexo I. Desse grupo fazem parte as instruções de adição, subtração, incremento, decremento, multiplicação e divisão.

b. Instruções de Desvio:

As instruções de desvio se dividem instruções de desvio incondicional e de desvio condicional. A Tabela 3.7 mostra a lista de instruções de desvio incondicional. Perceba que a Tabela 3.7 apresenta uma única instrução “*JMP end*”, entretanto, de fato, elas

são três – *SJMP*, *LJMP* e *AJMP* – e diferem entre si pelo formato do endereço de destino.

TABELA 3.7
INSTRUÇÕES DE DESVIO INCONDICIONAL

Mnemônico	Operação	Períodos de clock
JMP end	Desvia para o endereço	24
JMP @A+DPTR	Desvia para $A + DPTR$	24
CALL end	Chama subrotina no endereço	24
RET	Retorno de sub-rotina	24
RETI	Retorno de interrupção	24

A instrução *SJMP* codifica o endereço de destino como um byte de *offset* relativo. Esse byte de *offset* é um byte com sinal (complemento de dois) que é somado ao PC em uma aritmética de complemento de dois quando o desvio é executado, sendo que a faixa do desvio é -128 a $+127$ bytes de memória de programa relativos ao primeiro byte da instrução seguinte. A instrução é composta por dois bytes, um para o código de operação (*opcode*) e outro para o *offset*.

A instrução *LJMP* codifica o endereço de destino como uma constante de 16 bits, o que faz com que a instrução tenha 3 bytes, um para o *opcode* e outros dois para os bytes de endereço. O endereço de

destino, portanto, pode ser qualquer um nos 64k de espaço da memória de programa.

A instrução AJMP codifica o endereço de destino como uma constante de 11 bits, o que a torna uma instrução de dois bytes, um para o *opcode*, o qual contém 3 dos 11 bits de endereço, e outro byte para os 8 bits baixos do endereço de destino. Quando a instrução é executada, os 11 bits de endereços são os substitutos para os 11 bits baixos do PC. Como os bits altos permanecem os mesmos, o destino tem que estar dentro do mesmo bloco de 2 k da instrução seguinte a AJMP.

Em todos os casos, o programador especifica o endereço de destino para o assembler (programa montador) do mesmo jeito: como um rótulo ou como uma constante de 16 bits. A colocação do endereço de destino no formato correto para uma dada instrução é feita pelo assembler e, caso o formato requerido pela instrução não suporte a distância para o endereço de destino especificado, a mensagem “*Destination out of range*” (destino fora de faixa) é escrito na lista de erros.

A Tabela 3.7 mostra, também, uma única instrução CALL end, sendo que, de fato há duas: LCALL, que usa o formato de 16 bits, e ACALL que usa o formato de 11 bits. As sub-rotinas devem terminar com a instrução RET, a qual retorna a execução para a instrução seguinte a instrução CALL.

A instrução RETI, por sua vez, é utilizada para o retorno de uma rotina de atendimento de interrupção.

A Tabela 3.8 mostra a lista de instruções de desvio condicional disponíveis. Todas estas instruções especificam o endereço de destino por um byte de *offset* (rel).

TABELA 3.8
INSTRUÇÕES DE DESVIO CONDICIONAL

Mnemônico	Operação	Períodos de clock
JZ rel	Desvia se A = 0	24
JNZ rel	Desvia se A ≠ 0	24
DJNZ <byte>,rel	Decrementa destino e desvia se não for zero	24
CJNE A, <byte>, rel	Desvia de A ≠ <byte>	24
CJNE A, #data, rel	Desvia de A ≠ #dado	24

Maiores detalhes sobre as instruções de desvio condicional são fornecidos no Anexo I, onde o leitor pode consultar os modos de endereçamento disponíveis para cada uma delas.

O Programa da Fig. 3.17, que verifica a quantidade de números iguais a 40h presentes no bloco de memória entre 50h e 5Fh (16 posições), ilustra o emprego das instruções *JNZ rel* (desvia se

acumulador \neq zero) e *CJNE Rn, #DADO, rel* (Compara conteúdo do registrador com dado imediato e desvia se eles forem diferentes). A quantidade de números iguais a 40h será armazenada no registrador R1 e o registrador R0 será utilizado para a varredura do bloco de memória (em *loop* ou laço repetido 16 vezes).

Caso a tarefa fosse verificar a quantidade de números diferentes de 40 h, utilizaríamos em nosso programa a instrução *JZ rel* (desvia se o acumulador for zero). A instrução “*CJNE <byte 1>, <byte 2>, rel*” , por sua vez, aceita, além do formato apresentado, os seguintes formatos:

- *CJNE A, direto, rel;*
- *CJNE A, #dado, rel*
- *CJNE @Ri, #dado, rel*

Para que o programa anterior passe a verificar a quantidade de números menores que 40h presentes no bloco de memória, é necessária uma pequena modificação no fluxograma da Fig 3.17. Observe, no fluxograma modificado (Fig. 3.18), que após a execução da instrução “*SUBB A,#40h*”, o flag de *carry* será um sempre que o acumulador, que contém o conteúdo da posição de memória apontada por R0, for menor que 40h. Assim, se deve testar o *carry* para se detectar que o número lido é menor que 40h .

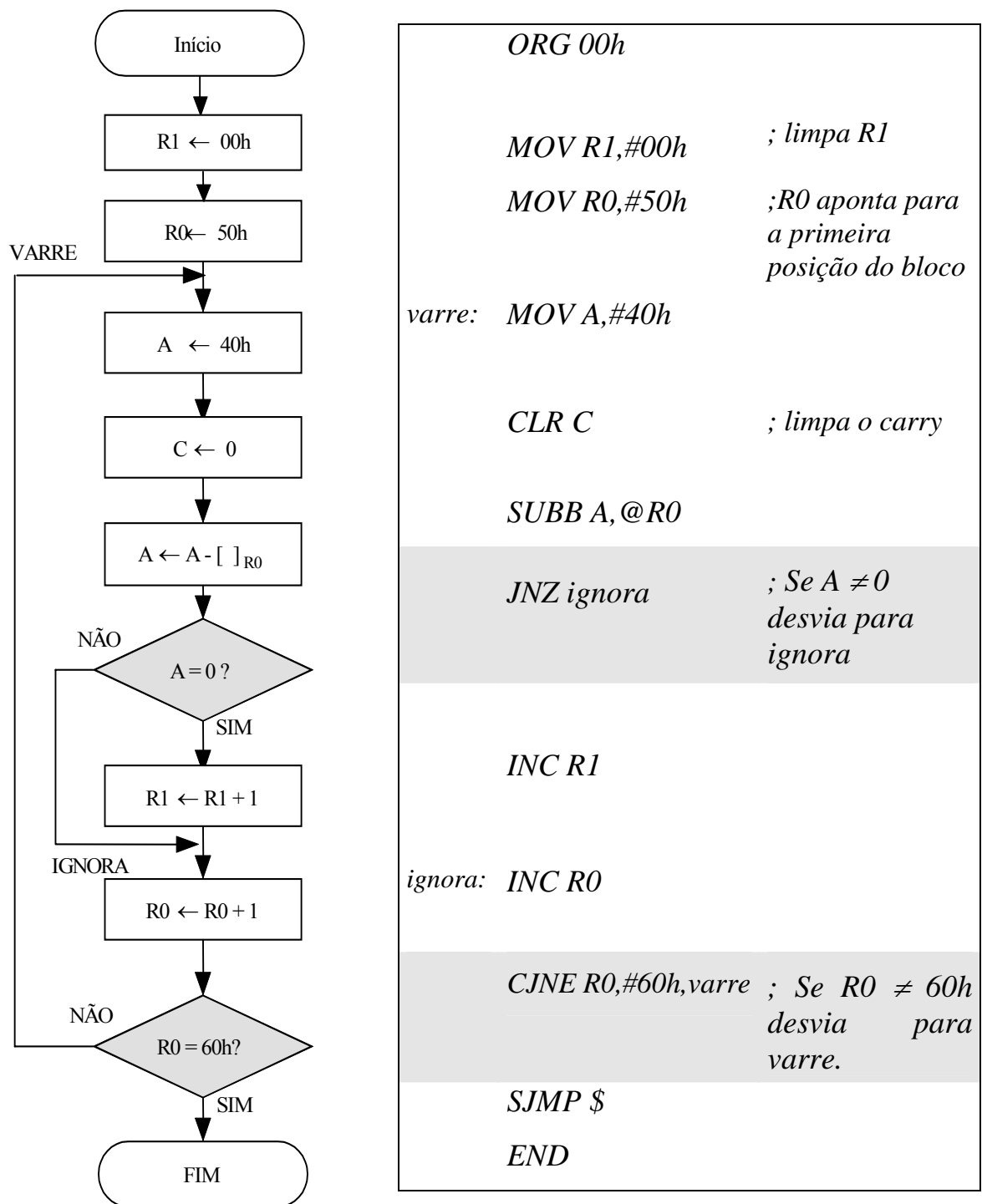


Fig. 3.17 - O programa que verifica a quantidade de números iguais a 40h presentes no bloco de memória entre 50h e 5Fh.

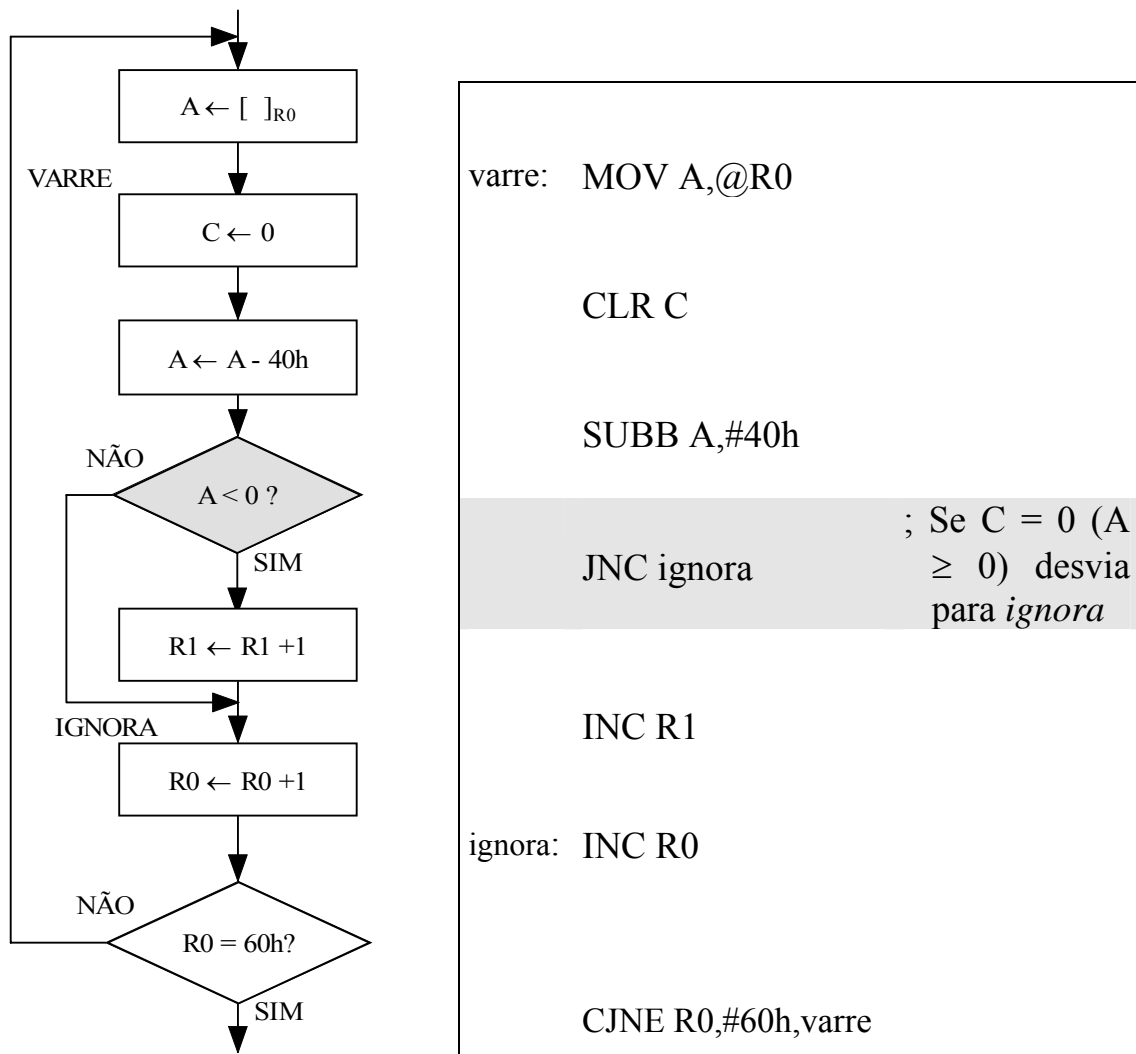
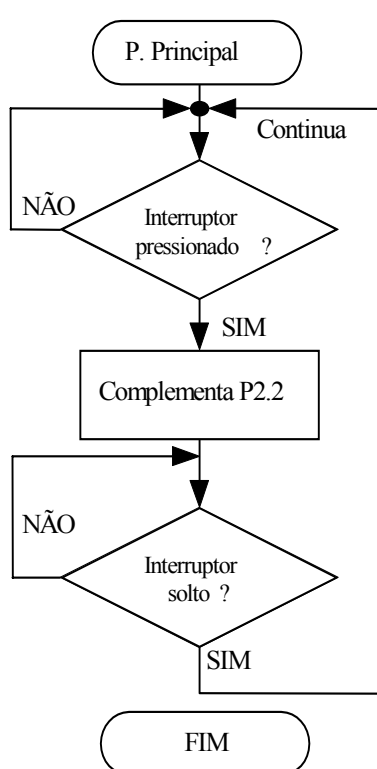


Fig. 3.18 - Programa que verifica a quantidade de números menores que 40h presentes no bloco de memória entre 50h e 5Fh.

c. Instruções de Manipulação de Variáveis Booleanas

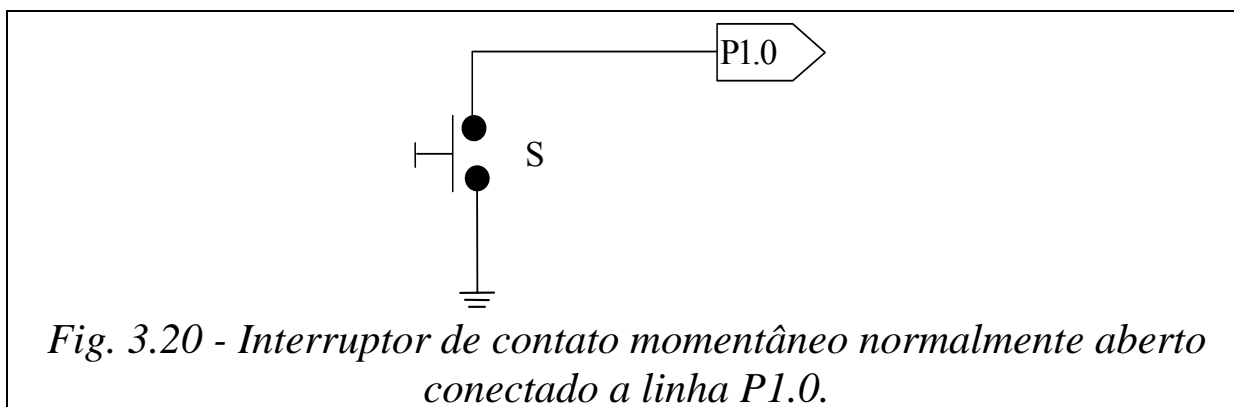
Conforme vimos na seção 3.4, existem bits endereçáveis na RAM interna, no espaço *SFR* e todas as linhas das portas de E/S são bits endereçáveis, o que permite que cada uma possa ser tratada como uma porta separada de um único bit. A totalidade das instruções de processamento booleano é mostrada no Anexo I no item “Manipulação de Variáveis Booleanas”.

Sempre que em um programa desejarmos testar se um interruptor conectado entre um pino de uma porta de E/S e a referência (GND) está ou não pressionado, utilizamos, dependendo da situação, as instruções *JB bit, rel* (desvia se o bit indicado for um) ou *JNB bit, rel* (desvia se o bit for zero). Para ilustrar, apresentamos na Fig. 3.19 um fluxograma, no qual sempre que o interruptor conectado entre o pino P1.0 e o GND (Fig. 3.20) for pulsado (pressionado e solto) é comutado o estado de um LED conectado a P2.2.



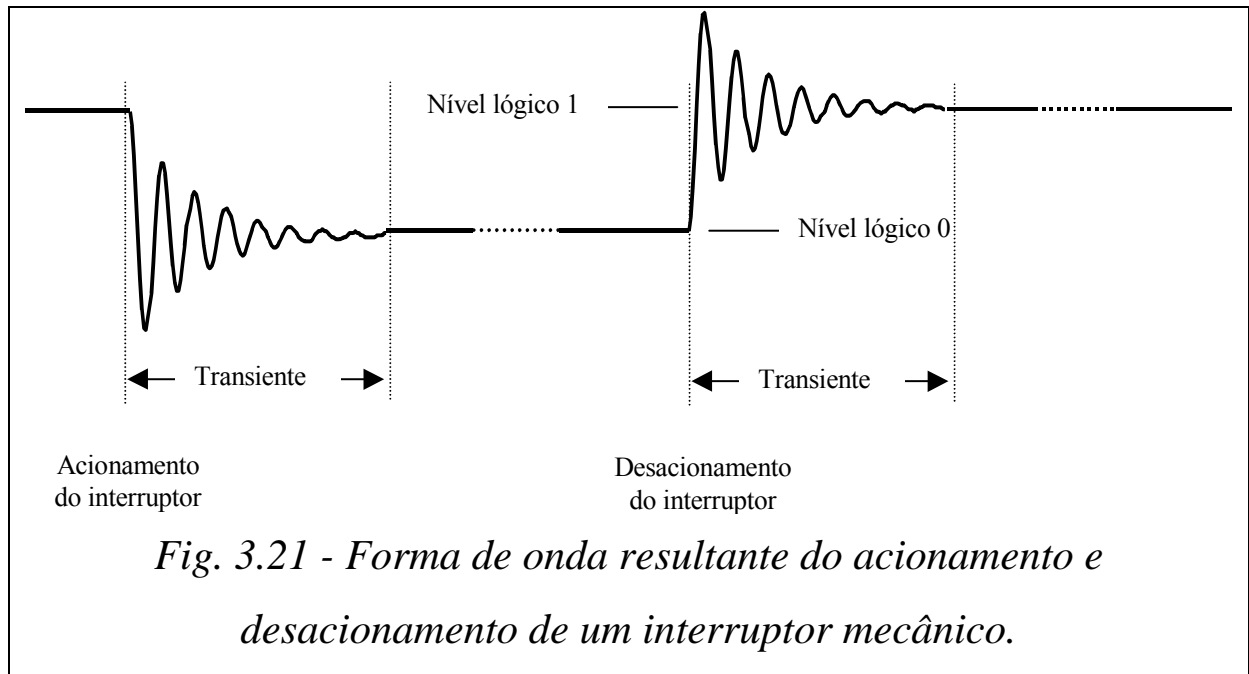
ORG 30h	
continua: JB P1.0, \$; Se P1.0 = 1 aguarda
CPL P2.2	; complementa P2.2
JNB P1.0, \$; Se P1.0 = 0 aguarda
SJMP continua	; desvia para continua
END	

Fig. 3.19 – Exemplo do emprego da instrução *JB bit, rel*.



Entretanto, sempre que um interruptor mecânico é acionado, ocorre uma oscilação na linha de E/S, conforme ilustra a Fig. 3.21. Essa oscilação interfere no software que monitora o acionamento/desacionamento do interruptor ligado a linha de E/S. O software deve prever os múltiplos pulsos gerados, removendo o seu efeito através de um procedimento conhecido como “*debouncing*” (eliminação de ruído). Ao registrar a alteração no nível lógico da linha à qual está ligada o interruptor, a rotina espera cerca de 10 ms e, então, amostra a linha novamente. Se a alteração no nível lógico da linha se confirma, o acionamento/desacionamento do interruptor é considerado válido e software responde a este acionamento/desacionamento. A Fig. 3.22 apresenta o fluxograma da Fig. 3.19 alterado pela incorporação do procedimento de *debouncing*. A sub-rotina “espera 10 ms” emprega dois registradores, R6 e R7, para gerar a espera após cada acionamento/desacionamento. Como a instrução “*DJNZ R6,\$*” é executada 4608 vezes (256 x 18) e a instrução “*DJNZ R7,\$-2*” é executada 18 vezes, o tempo total de execução da sub-rotina para um *clock* de 11,0592 MHz, incluindo a chamada no programa principal

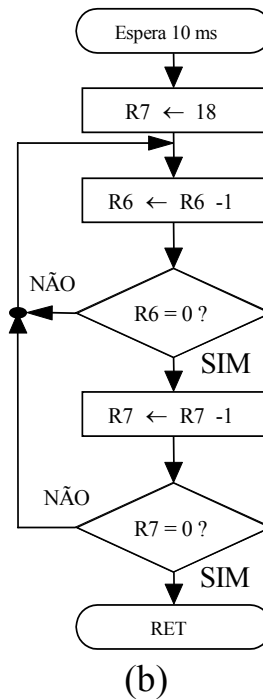
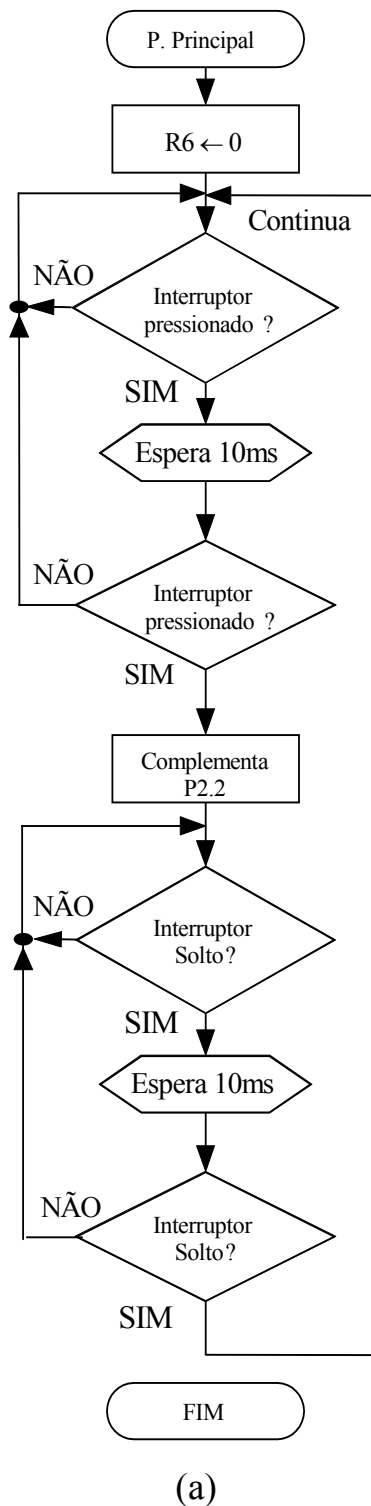
(“*ACALL espera*”), é de 10,04 ms ($2,170 \mu\text{s} + 2,170 \mu\text{s} + 4608 \times 2,170 \mu\text{s} + 18 \times 2,170 \mu\text{s} + 1,085 \mu\text{s}$).



d. Instruções Lógicas

A totalidade das Instruções Lógicas disponíveis em microcontroladores da família MCS-51™ são apresentados no Anexo I. Fazem parte deste rol, as instruções que executam operações booleanas (E, OU, OU Exclusivo e Não), as instruções de rotação que deslocam o acumulador 1 bit para a esquerda ou direita e a instrução “*SWAP A*” que comuta os *nibbles* alto e baixo do acumulador.

As instruções que executam operações booleanas (E, OU, OU Exclusivo e Não) sobre bytes operam manipulando bits. Por exemplo, se o acumulador contém 00110011b e <byte> contém 01010101b a instrução *ANL A,<byte>* deixa o acumulador com 00010001b. As aplicações mais comuns para essas instruções são:



```

ORG 00h
MOV R6,#00h
continua: JB P1.0, $      ; Se P1.0 =1
                        ; desvia para a
                        ; mesma linha

        ACALL espera
        JB P1.0, continua
        CPL P2.2
aguarda: JNB P1.0, $
        ACALL espera
        JNB P1.0, aguarda
        SJMP continua
;      SUBROTINA “ESPERA 10 ms”
espera:  MOV R7,#18
        DJNZ R6,$
        DJNZ R7,$-2
        RET
        END
  
```

(c)

Fig. 3.22 – Programa que comuta o estado de um LED quando o interruptor S conectado a P1.0 é pulsado: (a) fluxograma do programa principal, (b) fluxograma da sub-rotina e (c) listagem.

- *Colocar em nível lógico 1 um ou mais bits de um byte, empregando a lógica OU.* Para colocar em nível lógico 1 todos os bits do *nibble* alto do acumulador, sem alterar os demais, executamos a instrução ORL A, 0F0h.
- *Colocar em nível lógico 0 um ou mais bits de um byte, empregando a lógica E.* Para colocar em nível lógico 0 todos os bits do *nibble* baixo do acumulador, sem alterar os demais, executamos a instrução ANL A, 0F0h.
- *Inverter um ou mais bits de um byte, empregando a lógica OU Exclusivo.* Para complementar todos os bits *nibble* alto do acumulador, sem alterar os demais, executamos a instrução XRL A, 0F0h.

e. Transferência de Dados

- **RAM interna**

A Tabela 3.9 mostra o conjunto de instruções disponíveis para mover dados dentro do espaço de memória interna. Para conhecer os modos de endereçamento associados a estas instruções, o leitor de consultar o Anexo I.

TABELA 3.9
INSTRUÇÕES DE TRANSFERÊNCIA DE DADOS QUE
ACESSAM A RAM INTERNA

Mnemônico	Operação	Períodos de clock
MOV A, <fonte>	A = <fonte>	12
MOV <destino>, A	<destino> = A	12
MOV <destino>, <fonte>	<destino> = <fonte>	24
MOV DPTR, #dado 16	DPTR = constante de 16 bits	24
PUSH <fonte>	INC SP; MOV “@SP”, <fonte>	24
POP <destino>	MOV <destino>, “@SP”; DEC SP	24
XCH A, <byte>	A = <byte> e <byte> = A	12
XCHD A, @Ri	ACC e @Ri comutam os <i>nibbles</i> baixos	12

- **Procura em Tabelas**

A Tabela 3.10 mostra as duas instruções que estão disponíveis para buscar dados em tabelas residentes na memória de programa.

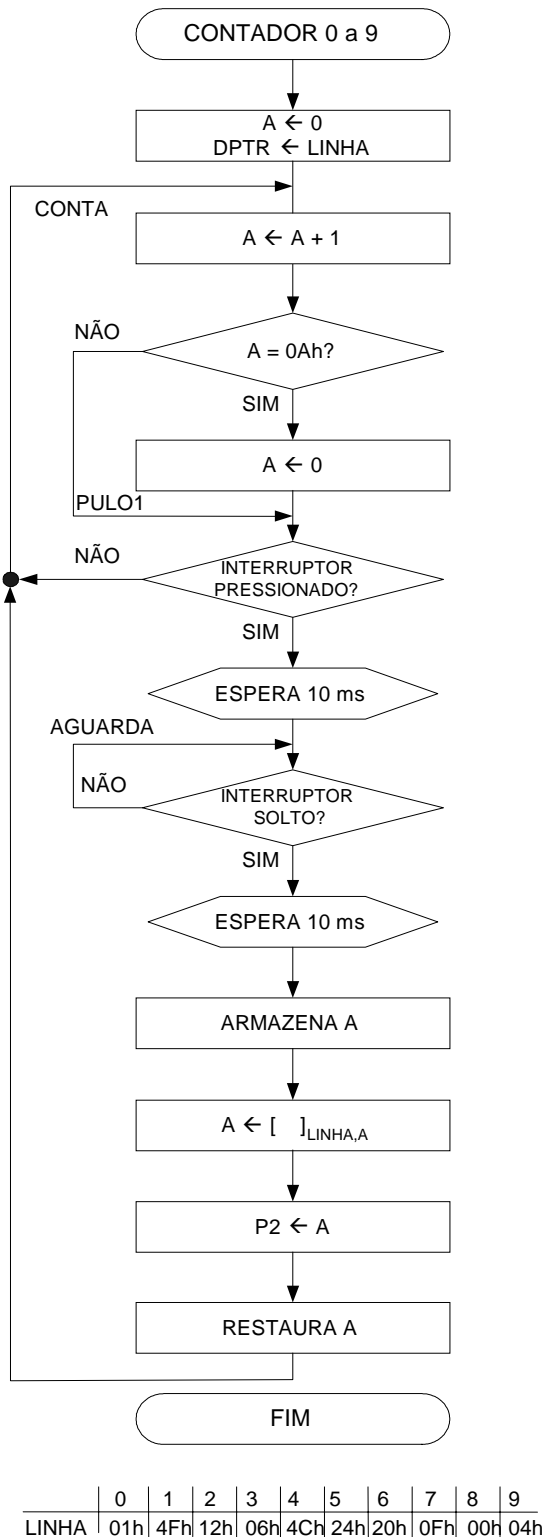
TABELA 3.10
INSTRUÇÕES DE LEITURA E PROCURA EM TABELAS

Mnemônico	Operação	Períodos de clock
MOVC A, @A+DPTR	Lê a memória de programa em (A+DPTR)	24
MOVC A, @A+PC	Lê a memória de programa em (A+PC)	24

O fluxograma da Fig. 3.23 apresenta um exemplo de emprego da instrução “*MOVC A, @A+DPTR*”. Esse fluxograma executa um contador de 0 a 9 que apresenta o valor atual de sua contagem em um Display de 7 segmentos toda vez que o interruptor conectado a P1.0 for pulsado. A instrução “*MOVC A, @A+DPTR*” executa o bloco “ $A \leftarrow []_{\text{LINHA},A}$ ”, convertendo o código BCD em saída para Display de 7 segmentos. Perceba que suprimimos nesse fluxograma as instruções de confirmação de acionamento/desacionamento do interruptor. A Fig. 3.24 apresenta o hardware conectado as portas de E/S do microcontrolador necessário para a implementação do fluxograma da Fig 3.23.

- RAM externa

As instruções que acessam a RAM externa serão discutidas na seção 3.10.1.



(a)

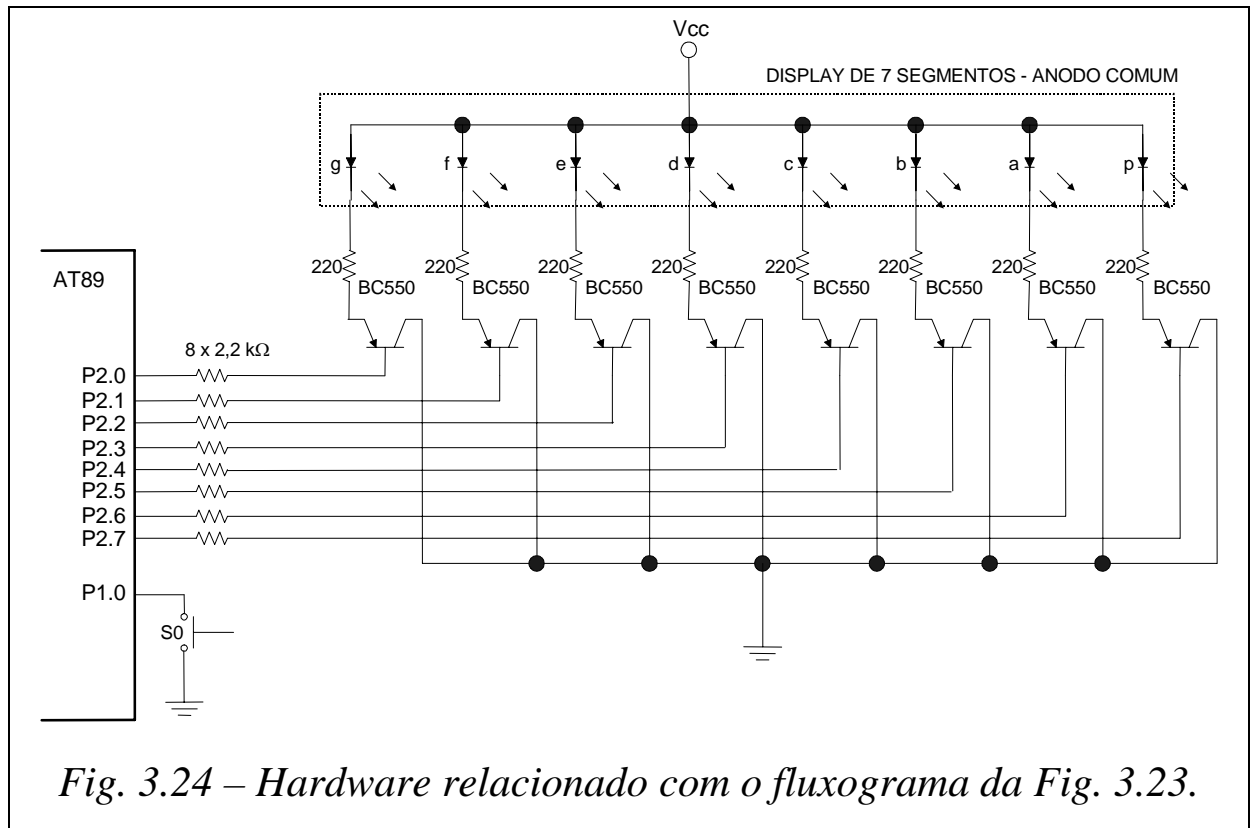
```

ORG 00h
CLR A
MOV DPTR,#LINHA
conta: INC A
      CJNE A, #0Ah, pulo1
      CLR A
pulo1: JB P1.0, conta
      ACALL espera
      JNB P1.0, $
      ACALL espera
      PUSH ACC
      MOVC A,@A+DPTR
      MOV P2, A
      POP ACC
      SJMP conta
linha: db 01h, 4Fh, 12h, 06h, 4Ch
      db 24h, 20h, 0Fh, 00h, 04h

      SUB-ROTINA ESPERA 10 ms
espera: MOV R7,#18
      DJNZ R6,$
      DJNZ R7,$-2
      RET
      END
  
```

(b)

Fig. 3.23 – Programa que conta de 0 a 9 com saída em Display de 7 segmentos: (a) Fluxograma e (b) listagem em assembly.



Podemos buscar em uma tabela empregando a instrução *MOVC A, @A+PC*. Nesse caso, o Contador de Programa (*PC*) é a base da tabela e a tabela é acessada por uma sub-rotina, na qual, no nosso exemplo, se faz a conversão BCD/7 segmentos. A modificação necessária na listagem da Fig. 3.23b é apresentada abaixo. Note que o incremento do acumulador é necessário para “compensar” a presença da instrução *RET* entre a instrução *MOVC* e a tabela.

	PUSH ACC
	ACALL converte
	MOV P2, A
	POP ACC
	SJMP conta
converte:	INC A
	MOV A, @A+PC
	RET
	db 01h, 4Fh, 12h, 06h, 4Ch
	db 24h, 20h, 0Fh, 00h, 04h
	<i>SUB-ROTINA ESPERA 10 ms</i>

Verifique seu Aprendizado

Para resolver os exercícios a seguir consulte o Anexo I, que apresenta o conjunto de instruções dos microcontroladores MCS-51™ compatíveis.

1. Elaborar um programa que coloque 033h em toda área de rascunho da memória RAM interna.
2. Elaborar um programa que conte o número de bits em 1 na porta P1 e apresente o resultado na posição de memória 70h.
3. Elaborar um programa que conte a quantidade de números ímpares armazenados nas posições de memória 50h a 7Fh de um microprocessador da família C51 e armazene o resultado no registrador R7.
4. Elaborar um programa para um processador da família C51 que execute a seguinte operação:
$$(\text{conteúdo memória})_{50H} = 2 \times (\text{conteúdo memória})_{51H} - 1$$
5. Elaborar um programa que some os conteúdos armazenados nas posições de memória 50h a 6Fh. Observe que a soma de números de 1 byte pode resultar em um número de 2 bytes.
6. Faça um programa que realize a operação lógica ou-exclusivo entre os conteúdos das posições de memória 40h e 70h e armazene o resultado no registrador R2 do banco 2.

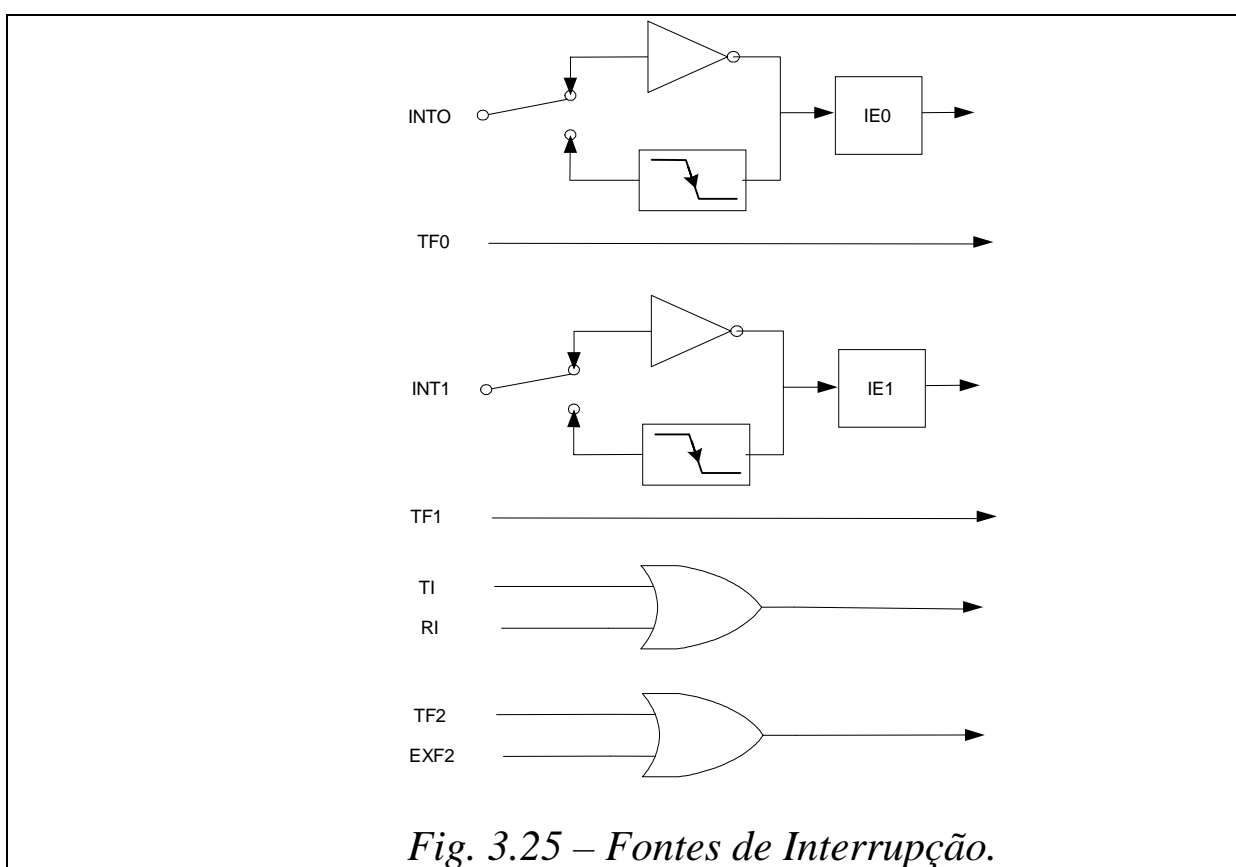
7. Faça um programa que calcule a quantidade de números maiores que 4Fh presentes na área de memória que vai de 30h a 3Fh. O resultado deve ser armazenado na posição de memória 40h.
8. Modifique o programa anterior, fazendo que ele calcule a quantidade de números entre 10h e 20h presentes na referida área de memória.
9. Faça um programa que calcule a quantidade de números ímpares e negativos presentes na área de memória que vai de 59h a 7Fh. O resultado deve ser armazenado no registrador R1 do banco 1.
10. Faça um programa que transfira, na ordem inversa, o conteúdo da área de memória entre 50h a 5Fh para a área de memória 60h a 6Fh.
11. Faça um programa que calcule a quantidade de números que apresentam o bit 5 igual a 1 na área de memória que vai de 63h a 75h e armazene o resultado na posição 76h.

3.7 – Interrupções:

Uma interrupção é o processo pelo qual um dispositivo externo ou interno pode interromper a execução de uma determinada tarefa e solicitar a execução de outra. Por exemplo, em um sistema de alarme, o arrombamento de uma janela faz que o sensor a ela conectado solicite que a CPU interrompa a tarefa que está executando

e passe a executar outra, que no caso poderia ser estabelecer a conexão com um modem, o qual discaria para um número pré-estabelecido.

O AT89S8252 possui, conforme ilustra a Fig. 3.25, um total de seis vetores de interrupção⁵: duas interrupções externas ($\overline{\text{INT0}}$ e $\overline{\text{INT1}}$), três interrupções por temporizadores (Timer 0, Timer 1 e Timer 2) e uma interrupção por canal serial.



Cada interrupção pode ser habilitada ou desabilitada individualmente agindo-se em um bit do registrador IE (Interrupt Enable). O registrador contém, também, um bit que pode desabilitar todas as interrupções de uma só vez. A Fig. 3.26 apresenta o

⁵ As interrupções nos microprocessadores MCS-51™ são vetoradas, isto é, elas possuem um endereço fixo para atendimento da interrupção denominado vetor de interrupção. Os vetores de interrupções foram apresentados na Fig. 3.6.

registrador IE, onde podemos observar que o bit 6 deste registrador não está implementado.

Cada fonte de interrupção pode, também, ser individualmente programada para ser uma fonte de baixa ou de alta prioridade, agindo-se em um bit do registrador IP (Priority Enable) apresentado na Fig. 3.27.

Uma interrupção de baixa prioridade pode ser interrompida apenas por uma interrupção de alta prioridade e uma interrupção de alta prioridade não pode ser interrompida por qualquer outra fonte de interrupção. Se duas interrupções de diferentes níveis de prioridade forem solicitadas simultaneamente, a de maior nível de prioridade será atendida primeiro; se duas interrupções de mesmo nível de prioridade forem solicitadas simultaneamente, o microcontrolador possui um processo interno de seleção que determina qual interrupção será atendida primeiro. Nesse caso, da mais alta para a mais baixa prioridade, as fontes de interrupção são apresentadas abaixo.

FONTE	VETOR
Interrupção Externa 0;	03h
Interrupção do <i>Timer 0</i> ;	0Bh
Interrupção Externa 1;	13h
Interrupção do <i>Timer 1</i> ;	1Bh
Interrupção de Canal Serial;	23h
Interrupção do <i>Timer 2</i> .	2Bh

Registrador IE – endereço 0A8h ; **Valor na inicialização** =0φ000000b

	EA	-	ET2	ES	ET1	EX1	ET0	EX0
bit	7	6	5	4	3	2	1	0
Bit = 1, habilita a interrupção.								
Bit = 0, desabilita a interrupção.								
Símbolo		Função						
EA		Desabilita todas as interrupções se em nível lógico 0. Se EA = 0, cada interrupção pode ser individualmente habilitada ou desabilitada.						
-		Reservado.						
ET2		Bit de habilitação da interrupção por timer 2.						
ES		Bit de habilitação da interrupção por canal serial.						
ET1		Bit de habilitação da interrupção por timer 1.						
EX1		Bit de habilitação da interrupção externa 1.						
ET0		Bit de habilitação da interrupção por timer 0.						
EX0		Bit de habilitação da interrupção externa 0.						
O programador não deve escrever 1 nos bits reservados, porque eles podem ser usados em produtos futuros.								

Fig. 3.26 – Registrador IE no AT89S8252.

Registrador IP – endereço B8h ; **Valor na inicialização** =φφ000000b

	-	-	PT2	PS	PT1	PX1	PT0	PX0
bit	7	6	5	4	3	2	1	0
Bit = 1, interrupção de alta prioridade Bit = 0, interrupção de baixa prioridade								
Símbolo	Função							
PT2	Define o nível de prioridade da interrupção por <i>Timer 2</i> .							
PS	Define o nível de prioridade da interrupção por canal serial.							
PT1	Define o nível de prioridade da interrupção por <i>Timer 1</i> .							
PX1	Define o nível de prioridade da interrupção externa 1.							
PT0	Define o nível de prioridade da interrupção por <i>Timer 0</i> .							
PX0	Define o nível de prioridade da interrupção externa 0.							

Fig. 3.27 – Registrador IP no AT89S8252.

As interrupções externas podem ainda ser programadas para serem detectadas por transição de nível 1 para 0 ou pelo nível 0. Os bits de controle das interrupções externas estão no registrador TCON (*Timer/Counter Control*), sendo que, no momento nos interessam apenas os quatro bits menos significativos, apresentados na Fig. 3.28.

Registrador TCON – endereço 88h ; **Valor na inicialização** = 00000000b

bit	Estudados a posteriori				IE1	IT1	IE0	IT0
	7	6	5	4	3	2	1	0
Símbolo	Função							
IE1	Este Flag é colocado em nível lógico 1 pelo hardware interno quando uma transição de 1 para 0 é detectada no terminal $\overline{INT1}$; é colocado em nível lógico 0 quando a interrupção é atendida.							
IT1	Caso este bit seja colocado em nível lógico 1 a interrupção externa 0 será aceita na transição de 1 para 0 no terminal $\overline{INT1}$. Colocado em nível lógico 0 a interrupção será aceita quando um nível lógico 0 estiver presente em $\overline{INT1}$.							
IE0	Este Flag é colocado em nível lógico 1 pelo hardware interno quando uma transição de 1 para 0 é detectada no terminal $\overline{INT0}$; é colocado em nível lógico 0 quando a interrupção é atendida.							
IT0	Caso este bit seja colocado em nível lógico 1 a interrupção externa 0 será aceita na transição de 1 para 0 no terminal $\overline{INT0}$. Colocado em nível lógico 0 a interrupção será aceita quando um nível lógico 0 estiver presente em $\overline{INT0}$.							

Fig. 3.28 – Registrador TCON no AT89S8252.

Como os terminais externos de interrupção são amostrados uma vez a cada 12 ciclos de clock, uma entrada alta ou baixa deve se manter por pelo menos 12 ciclos para garantir a amostragem. Se a

interrupção externa é ativada por transição, a fonte externa tem que manter o terminal em nível 1 por pelo menos 12 ciclos e depois mantê-lo em nível 0 pelo mesmo período para garantir que a transição seja percebida, colocando o flag IEx em nível lógico 1. Se for ativada por nível, a fonte externa deverá manter o pedido ativo até que a interrupção seja atendida. Uma vez atendida a interrupção, a fonte externa deve desativar a solicitação antes que a rotina de atendimento termine, pois se isso não ocorrer outro pedido será gerado.

Quando uma interrupção é aceita, ocorre o processo que segue:

1. A instrução em andamento é encerrada.
2. O PC é salvo na pilha.
3. O estado atual das interrupções é salvo internamente.
4. Interrupções do nível da interrupção atendida são bloqueadas.
5. O PC é carregado com endereço de atendimento da interrupção solicitada.
6. A rotina de interrupção é executada.

A rotina de interrupção encerra com uma instrução RETI. Isto recarrega o PC com seu antigo valor a partir da pilha e restabelece o antigo estado das interrupções. A execução do programa principal reinicia a partir do ponto que parou.

Para ajudar na compreensão do sistema de interrupções, modificamos o programa que gerava uma onda quadrada de 2 kHz (Fig. 3.12), incorporando-lhe novas funções. Agora, ao perceber uma alteração de 1 para 0 no nível lógico da entrada de interrupção externa $\overline{INT0}$, o programa altera a frequência de saída para 1 kHz, retornando a 2kHz em caso de um novo pedido de interrupção em $\overline{INT0}$ (nova transição de 1 para 0). E, ao perceber uma alteração de 1 para 0 na entrada $\overline{INT1}$, o programa suspende a geração da onda, tornando a gerá-la caso ocorra um novo pedido de interrupção em $\overline{INT1}$. A Fig. 3.29 mostra o fluxograma resultante e o Assembly gerado a partir dele é apresentado após essa figura.

Observe no fluxograma da Fig. 3.29 que o valor responsável pelo atraso de 250 μ s é armazenado no acumulador (113) e o responsável por 500 μ s é colocado em R1 (228). O valor corrente do atraso é transferido do acumulador para o registrador R0. Sempre que ocorrer uma interrupção $\overline{INT0}$, haverá uma permuta de conteúdos entre o acumulador e R1, alterando-se, conseqüentemente, o valor corrente do atraso. A onda quadrada, por sua vez, só é gerada se o Bit 00h (localizado na área endereçável por bit da RAM) estiver em nível lógico 0, sendo que o nível lógico presente nesse bit será trocado a cada interrupção $\overline{INT1}$.

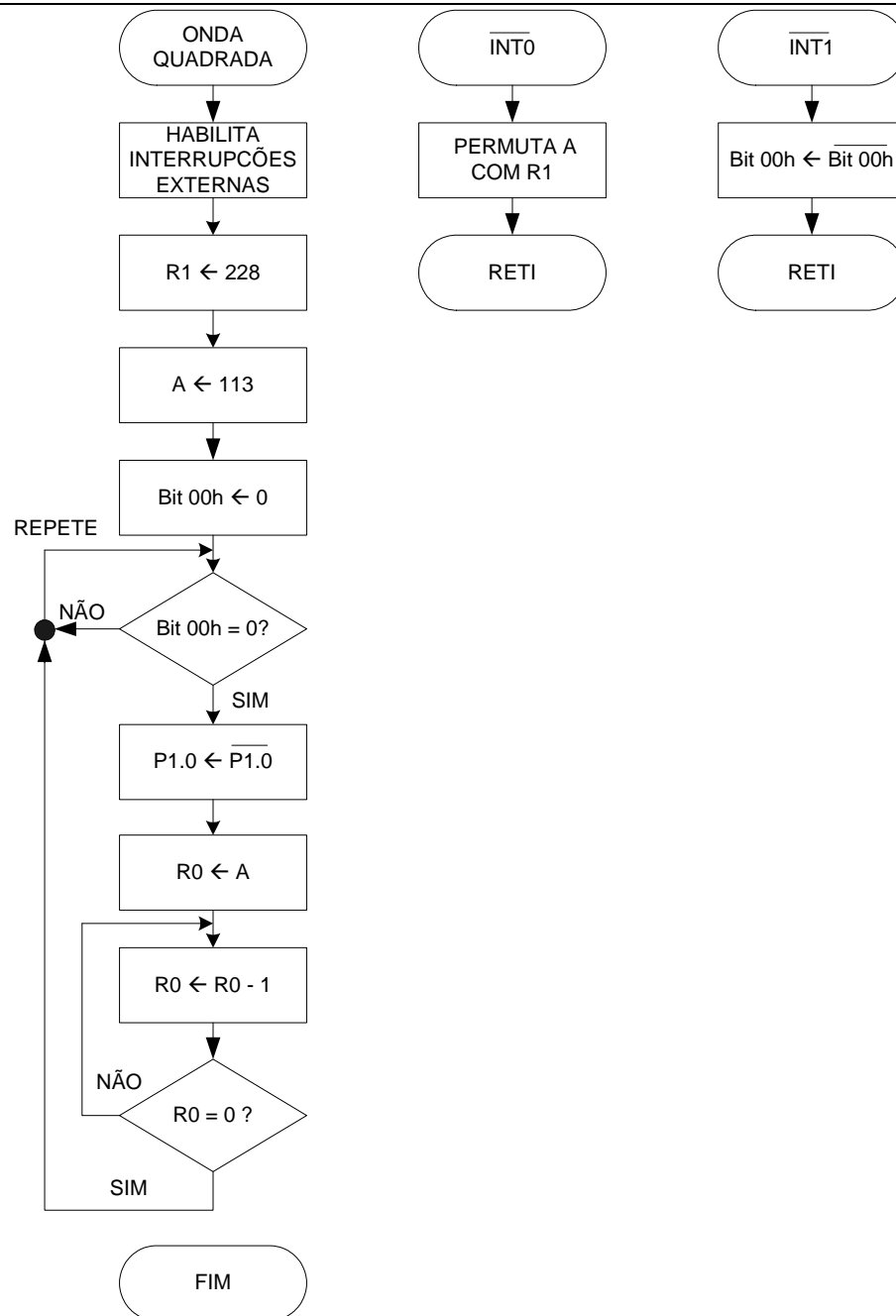


Fig. 3.29 – Programa que gera onda quadrada habilitada por interrupção e com duas frequências diferentes.

Listagem em assembly do programa da Fig. 3.29

SJMP main	; desvia para main
; Sub-rotina INT0	
ORG 03 h ;	; define onde o código subsequente será escrito
XCH A, R1	; permuta o conteúdo do acumulador e do registrador
RETI	; retorno da interrupção
; Sub-rotina INT1	
ORG 13h	; define onde o código será escrito
CPL 00h	; complementa o Bit 00h
RETI	; retorno da interrupção
; Geração da Onda Quadrada	
main: MOV IE,#85h	;habilita interrupções externas
MOV TCON,#05h	;interrupções por transição
MOV R1, #228	; transfere 228 para R1
MOV A, #113	; transfere 113 para o acumulador
CLR 00h	; coloca em 0 lógico o Bit 00h
repete: JB 00h, repete ;	; se o Bit 00h for 1 lógico, repete a instrução
CPL P1.0	; complementa P1.0
MOV R0,A	; transfere o conteúdo do acumulador para R0
DJNZ R0,\$; decrementa R0 e repete a instrução enquanto R0 não for 0
SJMP repete	; desvia para repete
END	; fim do Assembly

3.8 – Temporizadores/contadores:

O AT89S8252 possui três registradores temporizadores/contadores de 16 bits: timer 0, timer 1 e timer 2. Todos eles podem ser configurados para operar como temporizadores ou contadores de

eventos externos. Como um temporizador, o registrador é incrementado a cada ciclo de máquina (12 períodos de clock), ou seja, o registrador conta ciclos de máquina, o que resulta numa taxa de contagem igual a 1/12 da frequência do oscilador. Como um contador, o registrador é incrementado em resposta a uma transição de nível lógico um para zero no seu terminal externo correspondente, sendo que o sinal presente no terminal externo é amostrado no final de um ciclo de máquina. A contagem é incrementada quando a amostragem mostra um valor alto em um ciclo e um baixo em outro. Assim, como dois ciclos de máquina são necessários para reconhecer uma transição de 1 para 0, a taxa máxima de contagem é 1/24 da frequência do oscilador. E, para garantir a amostragem de todos os pulsos, um dado nível do sinal deve se manter por pelo menos um ciclo de máquina completo.

Os *timers* 0 e 1 possuem quatro modos de operação e o *timer* 2, por sua vez, possui 3 modos de operação, todos descritos a seguir.

3.8.1 – Os *Timers* 0 e 1 e seus modos de Modos de Operação

Sempre que ocorrer o término da contagem ou estouro (*overflow*) em um destes *timers*, o flag TFx correspondente no registrador TCON (Timer/counter Control Register, Fig. 3.30) vai para nível lógico 1, gerando um pedido de interrupção que será atendido caso essa função esteja habilitada. No atendimento da rotina de interrupção, ele é automaticamente colocado em nível lógico zero pelo hardware; caso a CPU não esteja programada para atender a

Registrador TCON – endereço 88h ; **Valor na inicialização** = 00000000b

	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
TF1	Flag de overflow do <i>timer</i> 1. Este flag é colocado em nível lógico 1 pelo hardware interno no término da contagem; é colocado em nível lógico zero no atendimento da rotina de interrupção.							
TR1	Bit de partida do <i>timer</i> 1. É colocado em nível lógico um/zero pelo software para ligar/desligar o <i>timer</i> 1.							
TF0	Flag de overflow do <i>timer</i> 0. Este flag é colocado em nível lógico 1 pelo hardware interno no término da contagem; é colocado em nível lógico zero no atendimento da rotina de interrupção.							
TR0	Bit de partida do <i>timer</i> 0. É colocado em nível lógico um/zero pelo software para ligar/desligar o <i>timer</i> 0.							
IE1	Este Flag é colocado em nível lógico 1 pelo hardware interno quando uma transição de 1 para 0 é detectada no terminal $\overline{INT1}$; é colocado em nível lógico 0 quando a interrupção é atendida.							
IT1	Caso este bit seja colocado em nível lógico 1 a interrupção externa 0 será aceita na transição de 1 para 0 no terminal $\overline{INT1}$. Colocado em nível lógico 0 a interrupção será aceita quando um nível lógico 0 estiver presente em $\overline{INT1}$.							
IE0	Este Flag é colocado em nível lógico 1 pelo hardware interno quando uma transição de 1 para 0 é detectada no terminal $\overline{INT0}$; é colocado em nível lógico 0 quando a interrupção é atendida.							
IT0	Caso este bit seja colocado em nível lógico 1 a interrupção externa 0 será aceita na transição de 1 para 0 no terminal $\overline{INT0}$. Colocado em nível lógico 0 a interrupção será aceita quando um nível lógico 0 estiver presente em $\overline{INT0}$.							

Fig. 3.30 – Registrador TCON no AT89S8252.

interrupção, o flag TFX deve ser colocado em nível lógico zero por software (SETB TFX).

Para que um *timer* inicie uma contagem é necessário colocar em nível lógico um o bit TRx correspondente no registrador TCON. A Fig. 3.30 apresenta o registrador TCON, incluindo os bits já estudados quando tratamos das interrupções.

A função temporizador ou contador nos *timers* 1 e 2 é selecionada pelo bit de controle C/ \bar{T} no registrador TMOD (Timer/Counter Mode Control Register), o qual é apresentado na Fig. 3.31. O modo de operação desses timers é selecionado pelos pares (M1,MO) em TMOD. O bit GATEx no registrador TMOD define a forma de habilitação do timer correspondente. Se GATEx for colocado em nível lógico 0, para habilitar a contagem basta colocar o bit TRx correspondente em nível lógico 1; caso GATEx for colocado em nível lógico 1, além de TRx = 1, é necessário que a entrada de interrupção externa \overline{INTx} correspondente seja colocada em nível lógico 1.

a. Modo 0 – Modo de 8 bits com prescaler

Cada *timer* conforme mostra a Fig. 3.32, é composto por 2 registradores de 8 bits THx e TLx. Neste modo os *timers* operam como temporizadores/contadores de 8 bits com um prescaler divisor por 32, sendo que o registro THx recebe o valor inicial da contagem e o registro TLx serve com divisor por 32 (2^5), ou seja, toda vez que TLx contar até 32, THx é incrementado em uma unidade. Quando

THx atingir 00h (passar de FFh para 00h), o flag TFX é colocado em nível lógico 1, solicitando um pedido de interrupção. O *timer* continuará contando até que, por software, TRx seja colocado em nível lógico 1. Nesse modo é possível contar 8192 eventos (256 x 32). O modo 0 é ilustrado pela Fig. 3.33.

Registrador TMOD – endereço 89h ; **Valor na inicialização** = 00000000b

	GATE -1	C/ \overline{T} -1	M1-1	M0-1	GATE -0	C/ \overline{T} -0	M1-0	M0-0
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
GATE -x	Se GATE _x for colocado em nível lógico 0, para habilitar a contagem basta colocar o bit TR _x correspondente em nível lógico 1; caso GATE _x for colocado em nível lógico 1, além de TR _x = 1, é necessário que a entrada de interrupção externa \overline{INT}_x correspondente seja colocada em nível lógico 1.							
C/ \overline{T} -x	Bit seletor de função. É colocado em nível lógico 0 para operação como temporizador e em nível lógico 1 para operação como contador							
M1-x	Bit 1 de modo							
M0-x	Bit 0 de modo							

M1-x	M0-x	Modo	Modo de operação
0	0	0	Temporizador/contador THx de 8 bits com TLx como prescaler de 5 bits
0	1	1	Temporizador/contador de 16 bits
1	0	2	Temporizador/contador TLx de 8 bits com recarga automática por THx
1	1	3	Timer 0 dividido em dois contadores de 8 bits. Timer 1 parado.

Fig. 3.31 – Registrador TMOD no AT89S8252.

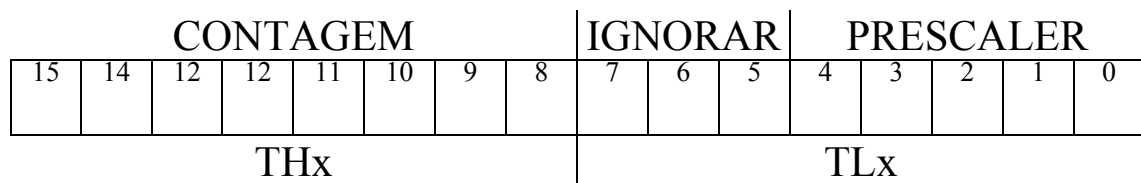
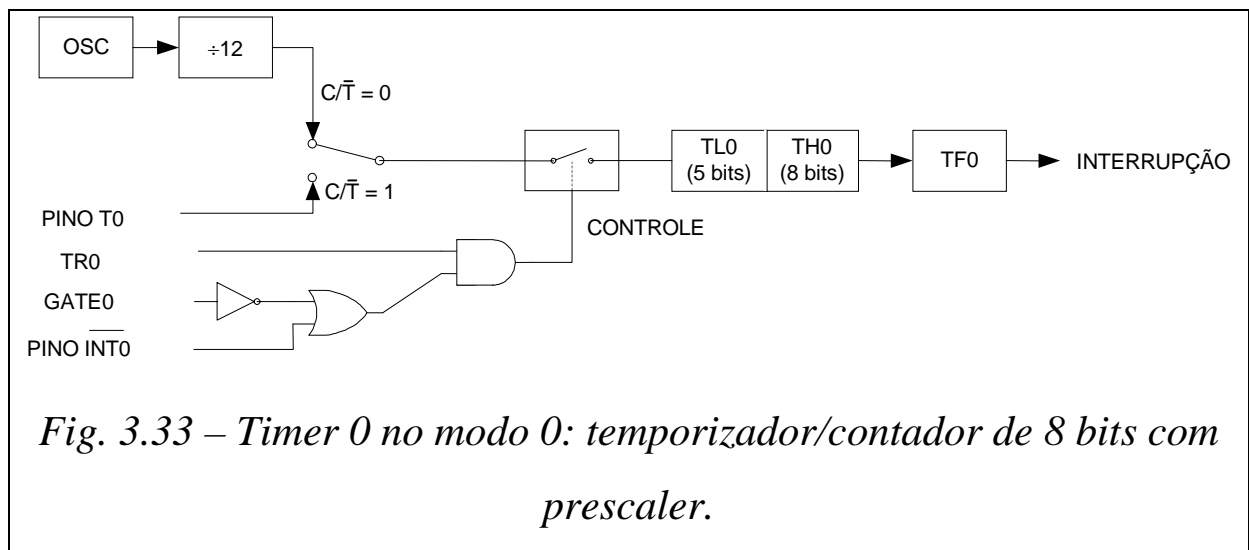


Fig. 3.32 – Operação dos timers no modo 0.



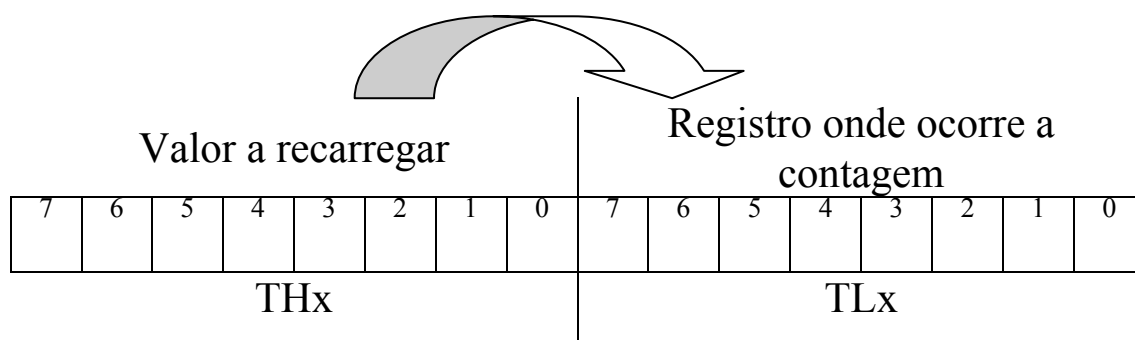
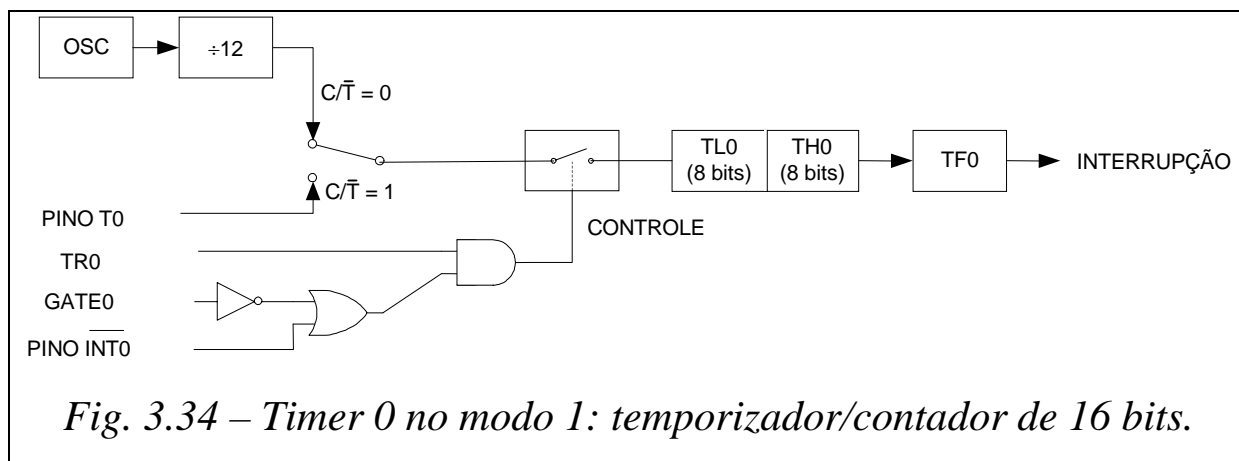
b. Modo 1 – Modo de 16 bits

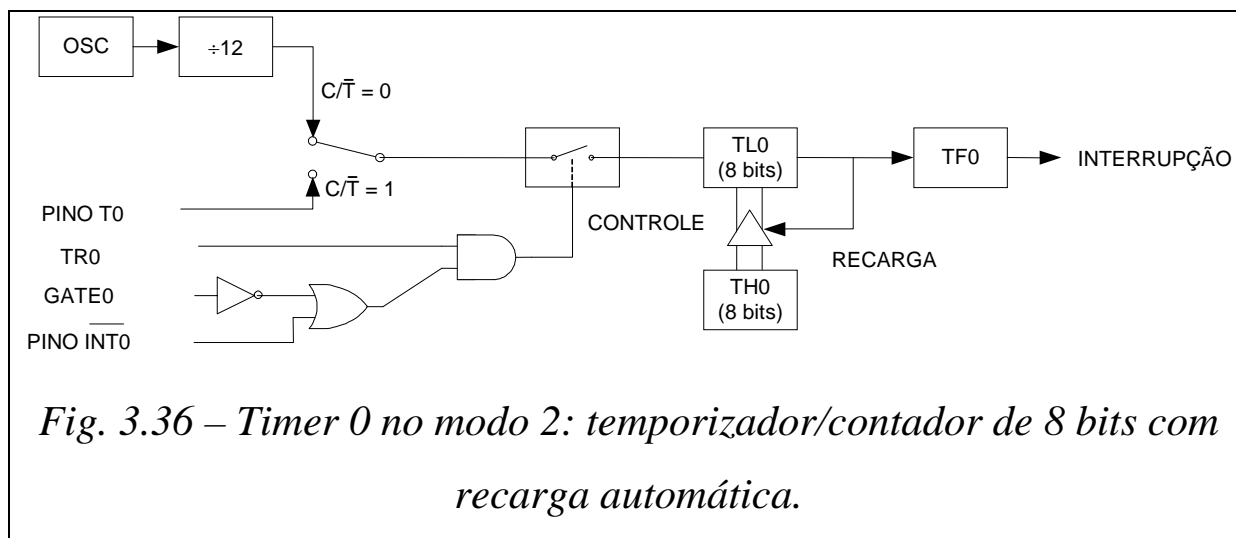
Nesse modo ambos os *timers* operam como temporizadores/contadores de 16 bits, com os registradores THx e TLx compondo um único registrador de 16 bits para efeito de contagem. Quando o sinal de clock é recebido, o temporizador conta para cima : 0000h, 0001h, 0002h, etc.; um overflow ocorre quando a contagem vai de FFFFh para 0000Fh. Nesse modo de operação podemos contar 65536 eventos. A Fig. 3.34 ilustra o modo de 16 bits.

c. Modo 2 – Modo de 8 bits com recarga automática

Nesse modo ambos os *timers* operam como temporizadores/contadores de 8 bits (TLx), com recarga automática (THx) conforme

ilustra a Fig. 3.35. Um overflow a partir de TLx não somente coloca em nível lógico um TFx, como também recarrega TLx com o conteúdo de THx, sendo que, na recarga, THx permanece inalterado. Nesse modo de operação podemos contar apenas 256 eventos. O modo de 8 bits com recarga automática é ilustrado pela 3.36.

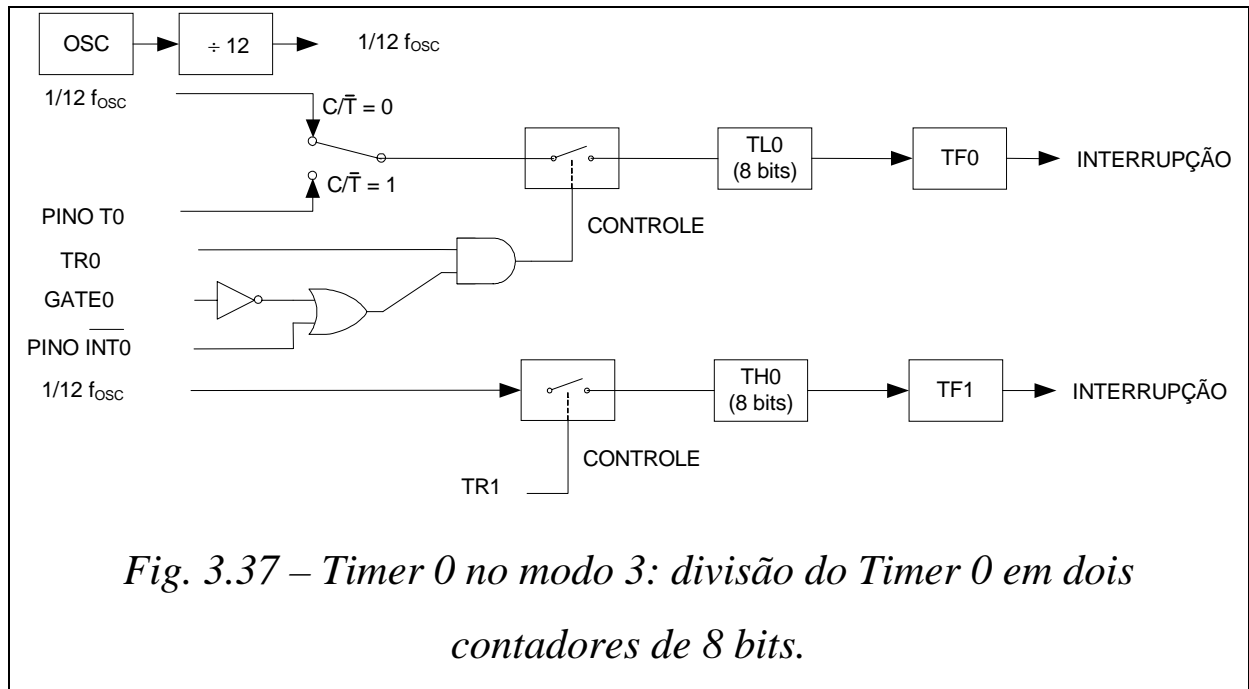




d. Modo 3 – Modo de divisão do Timer 0

No modo 3, o Timer 1 apenas retém a sua contagem; o efeito é o mesmo que fazer $TR1 = 0$. Por sua vez, o Timer 0 é transformado em dois contadores separados, conforme mostra a Fig. 3.37. O registrador TL0 usa os bits de controle do Timer 0 - $C/\overline{T} = 0$, GATE 0, TR0 e $\overline{INT0}$. Já o registrador TH0 é travado na sua função de temporizador (contando ciclos de máquina) e utiliza os bits de controle TR1 e TF1 do Timer 1. Conseqüentemente, TH0 passa a controlar a interrupção do Timer 1.

O modo 3 é indicado para aplicações que requerem um temporizador ou contador extra de 8 bits. Quando o Timer 0 está no modo 3, o Timer 1 pode ser ligado ou desligado retirando-o ou colocando-o em seu próprio modo 3. Neste caso, o Timer 1 pode ser usado pelo canal serial como um gerador de taxa de transmissão ou em qualquer outra aplicação que não requeira uma interrupção.



Para exemplificar o funcionamento dos *timers*, reelaboramos o programa que gerava uma onda quadrada de 2 kHz (Fig. 3.12), de tal forma que a contagem dos 250 μ s passou a ser realizada pelo *Timer 1* operando como temporizador. O fluxograma resultante é apresentado na Fig. 3.38 e, a partir dele, três listagens em assembly são geradas: uma para o *Timer1* no modo 0, outra no modo 1 e uma terceira no modo 2. Observe que a listagem gerada para o modo 0 não configura TMOD, pois na inicialização (*reset*) o *Timer 1* está configurado para operar como temporizador ligado por software no modo 0. Note, ainda, que a caixa "*recarrega valor inicial*" foi construída tracejada na Fig. 3.38, pois no modo 2 o valor da contagem é recarregado automaticamente. Observe, também, que a função do programa principal é restrita apenas a inicialização do sistema. A geração da onda quadrada é realizada na rotina de atendimento da interrupção. Após o *reset*, o programa principal repetirá indefinidamente a instrução "SJMP \$", aguardando o pedido de interrupção do *Timer 1*.

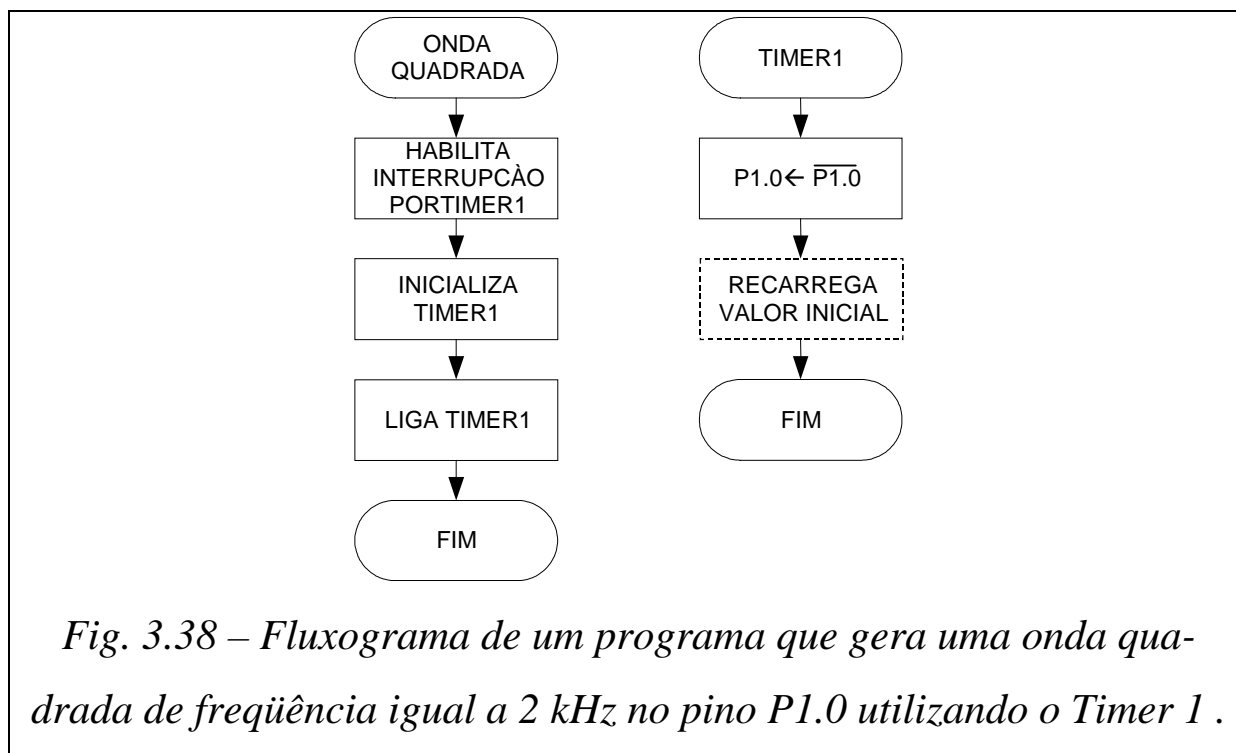
A geração da onda quadrada em uma rotina de interrupção configura-se na grande vantagem do fluxograma da Fig. 3.38 sobre o da Fig. 3.12, pois não prende o microcontrolador a uma única função.

O cálculo dos valores carregados nos registradores do *Timer 1* para cada um dos modos de operação são apresentados a seguir.

No modo 0, o registrador TL1 funciona como um divisor da frequência por 32. Assim, TH1 é incrementado a cada 32 ciclos de máquina. Utilizando um cristal com frequência de 11,0592 MHz ($T = 90,42 \text{ ns}$), um ciclo de máquina leva $1,085 \mu\text{s}$ ($12 \times 90,42 \text{ ns}$) para ser executado. Logo, TH1 é incrementado a cada $34,72 \mu\text{s}$ ($32 \times 1,085 \mu\text{s}$). Então, para gerarmos um tempo de cerca de $250 \mu\text{s}$, precisamos de 7 contagens do *Timer 1* ($250 \mu\text{s} / 34,72 \mu\text{s}$). Assim, o valor a ser carregado no registrador TH1 será 249 ($256 - 7$).

No modo 1, TH1 e TL1 compõe um registrador de 16 bits incrementado a cada ciclo de máquina. Nesse caso, para obtermos $250 \mu\text{s}$, precisamos de aproximadamente 230 contagens do *Timer 1* ($250 \mu\text{s} / 1,085 \mu\text{s}$). Assim, o valor a ser carregado no par (TH1, TL1) é 0FF1Ah, o valor hexadecimal correspondente a 65306 ($65536 - 230$).

No modo 2, TL1 é incrementado a cada ciclo de máquina e a cada *overflow* é recarregado automaticamente por TH1. Nesse caso, para obtermos $250 \mu\text{s}$, precisamos novamente de aproximadamente 230 contagens do *Timer 1*. Assim, o valor a ser carregado em TH1 e TL1 é 26 ($256 - 230$).



Listagem em assembly do programa da Fig. 3.38 com o *Timer 1* no modo 0:

	SJMP main	; Desvia para main
	ORG 1bh	; Define onde o código subsequente será escrito
	CPL P1.0	; Complementa P1.0
	MOV TH1, #249	; Recarrega valor inicial da contagem em TH1
	RETI	; retorno de interrupção
main:	MOV IE, #88h	; Habilita Timer 1 a gerar pedido de interrupção
	MOV TH1, #249	; Carrega valor inicial da contagem em TH1
	SETB TR1	; Liga Timer 1
	SJMP \$; Para a execução e espera interrupção
	END	; Fim do Assembly

Listagem em assembly do programa da Fig. 3.38 com o *Timer 1* no modo 1:

	SJMP main	; Desvia para main
	ORG 1bh	; Define onde o código subsequente será escrito
	CPL P1.0	; Complementa P1.0
	MOV TH1, #0FFh	; Recarrega valor inicial da contagem em TH1 e TL1
	MOV TL1, #1Ah	
	RETI	; retorno de interrupção
main:	MOV IE, #88h	; Habilita Timer 1 a gerar pedido de interrupção
	MOV TMOD, #10h	; Timer 1 no modo 1, como temporizador e ligado por software
	MOV TH1, #0FFh	; Carrega valor inicial da contagem em TH1 e TL1
	MOV TL1, #1Ah	
	SETB TR1	; Liga Timer 1
	SJMP \$; Para a execução e espera interrupção
	END	; Fim do Assembly

Listagem em assembly do programa da Fig. 3.38 com o *Timer 1* no modo 2:

	SJMP main	; Desvia para main
	ORG 1bh	; Define onde o código subsequente será escrito
	CPL P1.0	; Complementa P1.0
	RETI	; retorno de interrupção
main:	MOV IE, #88h	; Habilita Timer 1 a gerar pedido de interrupção
	MOV TMOD, #20h	; Timer 1 no modo 2, como temporizador e ligado por software
	MOV TH1, #26	; Carrega valor inicial da contagem em TL1 e da recarga em TH1
	MOV TL1, #26	
	SETB TR1	; Liga Timer 1
	SJMP \$; Para a execução e espera interrupção
	END	; Fim do Assembly

Um segundo exemplo é apresentado no fluxograma da Fig. 3.39. Nesse fluxograma, a função executada é o acionamento seqüencial de um conjunto de 8 *LEDS* conectados a Porta 2, sendo que o Timer 0 é o responsável por gerar um intervalo de tempo de 1 s entre as mudanças de estado. Para implementar esse contador de 1 s, é necessário que o *Timer 0* conte um total de 921600 ciclos de máquina (1 s / 1,085 μ s). Como o número máximo de contagens no modo 1 é 65536, são necessários 14,06 *overflows* do *Timer 0* para gerar 1 s (921600 / 65536). Utilizando 15 *overflows*, o número inteiro imediatamente superior a 14,06, não será necessário que o *Timer 0* seja programado para o número máximo de contagens, pois:

$$\text{Contagens necessárias por overflow} = \frac{921600}{15} = 61440 \text{ contagens}$$

Assim, o valor inicial da contagem que deve ser carregado no par (TH0,TL0) será:

$$\text{Valor inicial de (TH0,TL1)} = 65536 - 61440 = 4096 = 1000h$$

Observe no fluxograma da Fig. 3.39 que a função do programa principal é restrita apenas a inicialização do sistema. O acionamento seqüencial dos *LEDS* é realizado a cada 15 atendimentos da rotina de interrupção por *Timer 0*.

Observe na listagem em assembly do Fluxograma da Fig. 3.39 o uso da diretiva do assembler "*EQU*", cujo a função é associar um valor numérico ou um registrador a uma variável.

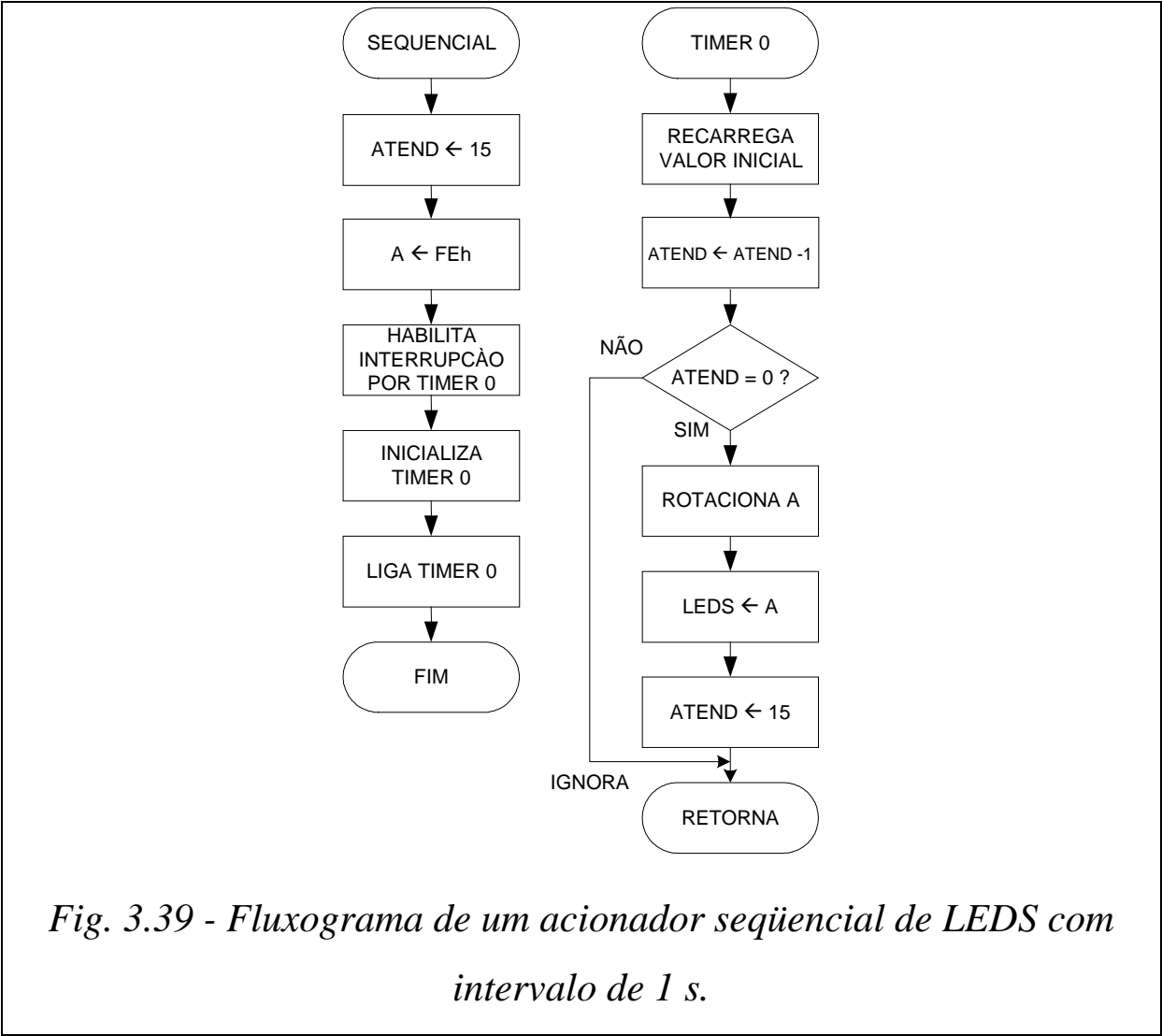


Fig. 3.39 - Fluxograma de um acionador seqüencial de LEDS com intervalo de 1 s.

Listagem em assembly do fluxograma da Fig. 3.39:

ATEND	EQU	R2	; O reg. R2 é associado a variável ATEND
LEDS	EQU	P2	; A Porta 2 é associada a variável LEDS
;			
	SJMP	main	; Desvia para main
; Sub-rotina de Timer 0			
	ORG	0Bh	; Define onde o código subsequente será escrito
	MOV	TH0, #10h	; Só é necessária a recarga de TH0
	DJNZ	ATEND, ignora	; Enquanto não ocorrerem 15 atendimentos desvia para ignora
	RL	A	; Rotaciona acumulador à esquerda
	MOV	LEDS, A	; Troca LED aceso
	MOV	ATEND, #15	; Reestabelece 15 atendimentos
ignora:	RETI		

; Programa Principal

main:	MOV ATEND, #15	; Programa 15 atendimentos
	MOV A, #0FEh	; Inicia aceso <i>LED</i> da direita
	MOV IE, #84h	; Habilita Interrupção por <i>Timer 0</i>
	MOV TMOD, #01h	; Timer 0 como temporizador no modo 1, acionado por software
	SETB TR0	; Liga Timer 0
	SJMP \$; Para execução e aguarda interrupção
	END	; Fim do Assembly

3.8.2 – O Timer 2 e seus modos de Modos de Operação

O Timer 2 é um temporizador/contador de 16 bits presente somente nos microncontroladores com núcleo C52. Nesse núcleo, para acomodar o Timer 2, seis registradores de função especial extras são acrescentados: os registradores de temporização TL2 e TH2, T2CON, T2MOD e os registradores de captura RCAP2L e RCAP2H. Tal qual os outros timers, ele pode operar como um temporizador ou contador de eventos, dependendo do valor de $C/\bar{T} - 2$ no registrador T2CON (Fig. 3.40). O Timer 2 tem três modos de operação: captura, recarga automática e gerador de taxa de transmissão, os quais são selecionados pelos bits de T2CON, conforme mostra a Tabela 3.11.

a. Modo de captura

No modo de captura, ilustrado pela Fig. 3.41, o bit EXEN2 do registrador T2CON seleciona duas opções:

- Se $EXEN2 = 0$, o Timer 2 é um registrador de 16 bits cujo estouro coloca $TF2$ em 1, o que pode gerar um pedido de interrupção;
- Se $EXEN2 = 1$, o Timer 2 trabalha do mesmo modo, porém uma transição de 1 para 0 na entrada externa $T2EX$ faz com que o valor corrente dos registradores do Timer 2, $TL2$ e THs , sejam capturados, respectivamente, por $RCAP2L$ e $RCAP2H$. Ainda, a transição em $T2EX$ coloca em 1 o bit $EXF2$ em $TCON2$, o qual pode gerar uma interrupção.

TABELA 3.11

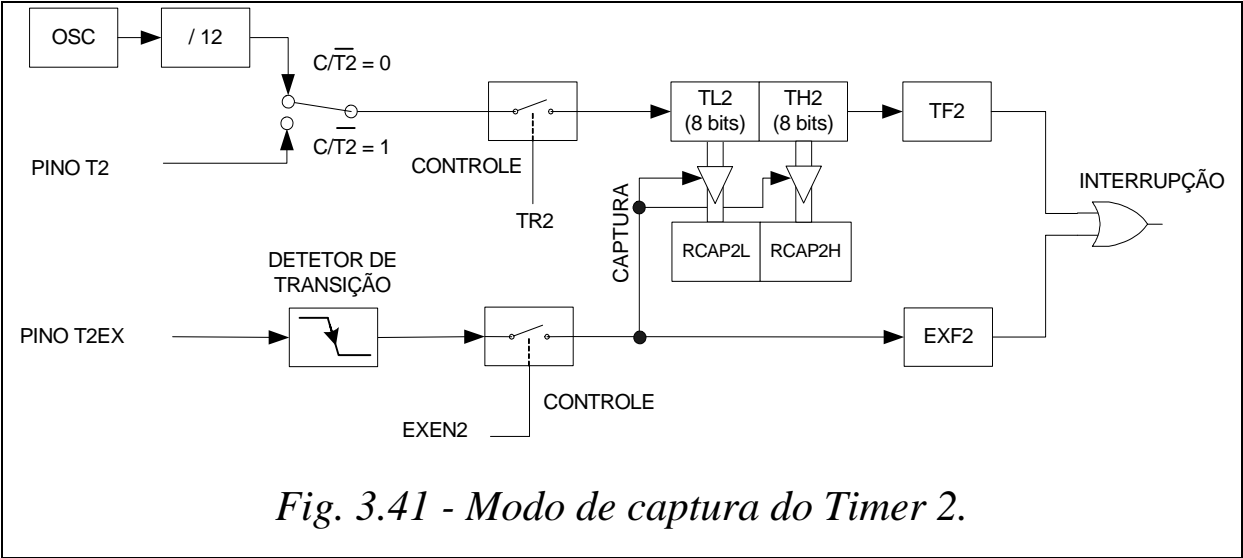
MODOS DE OPERAÇÃO DO TIMER 2

RCLK + TCLK	CP/RL2	TR2	MODO
0	0	1	16 bits com recarga automática
0	1	1	captura de 16 bits
1	ϕ	1	gerador de taxa de transmissão
ϕ	ϕ	0	desligado

Registrador T2CON – endereço 0C8h ; **Valor na inicialização** = 00000000b

	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/ \overline{T} - 2	CP/ $\overline{RL2}$
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
TF2	Flag de overflow do <i>timer</i> 2. Ele é colocado em nível lógico 1 pelo hardware interno no término da contagem e deve ser limpo por software no final da interrupção. Ele não será colocado em nível lógico 1 quando RCLK = 1 ou TCLK = 1.							
EXF2	Flag da entrada externa do Timer 2. Ele é colocado em nível lógico 1 quando uma captura ou uma recarga é motivada por uma transição negativa em T2EX com EXEN2 = 1. Quando a interrupção por Timer 2 está habilitada, EXF2 = 1 fará a CPU atender a rotina de interrupção. Ele deve ser colocado em nível lógico zero por software.							
RCLK	Flag do clock de recepção. Quando em nível lógico 1 faz com que o canal serial use os pulsos de <i>overflow</i> do <i>Timer</i> 2 como clock de recepção nos modos 1 e 3. Se RCLK = 0, os <i>overflows</i> do Timer 1 são usados para o clock de recepção.							
TCLK	Flag do clock de transmissão. Quando em nível lógico 1 faz com que o canal serial use os pulsos de <i>overflow</i> do Timer 2 como clock transmissão nos modos 1 e 3. Se TCLK = 0, os pulsos de <i>overflow</i> do Timer 1 são usados para o clock de transmissão.							
EXEN2	Flag de habilitação da entrada externa do Timer 2. Quando colocado em nível lógico 1 permite que ocorra uma captura ou recarga como resultado de uma transição negativa em T2EX, desde que o Timer 2 não esteja sendo usado como clock do canal serial. Se EXEN2 = 0 o Timer 2 ignora eventos em T2EX.							
TR2	Bit de partida do <i>timer</i> 2. É colocado em nível lógico um/zero pelo software para ligar/desligar o <i>Timer</i> 2.							
C/ \overline{T} - 2	Seletor de temporizador ou contador. Se C/ \overline{T} - 2 = 0 o <i>Timer</i> 2 opera como temporizador; se C/ \overline{T} - 2 = 1 como contador.							
CP/ $\overline{RL2}$	Flag de captura/recarga. Caso este bit seja colocado em nível lógico 1, capturas ocorrerão nas transições negativas de T2EX se EXEN2 = 1; quando em nível lógico zero, recargas ocorrerão nestas condições. Quando RCLK = 0 ou TCLK = 0, este bit é ignorado e o timer é forçada a se recarregar no overflow do Timer 2.							

Fig. 3.40 – Registrador TCON no AT89S8252.



b. Modo de recarga automática

Neste modo, o *Timer 2* pode ser programado como contador crescente ou decrescente. A forma de contagem depende do estado do bit DCEN (Down Counter Enable) do registrador T2MOD (Fig. 3.42). Se DCEN = 0 (situação no reset), o *Timer 2* é um contador crescente; se DCEN = 1, dependendo do valor do pino T2EX, o *Timer 2* pode ser um contador crescente ou decrescente.

Registrador T2MOD – endereço 0C9h ; Valor na inicialização = 00000000b

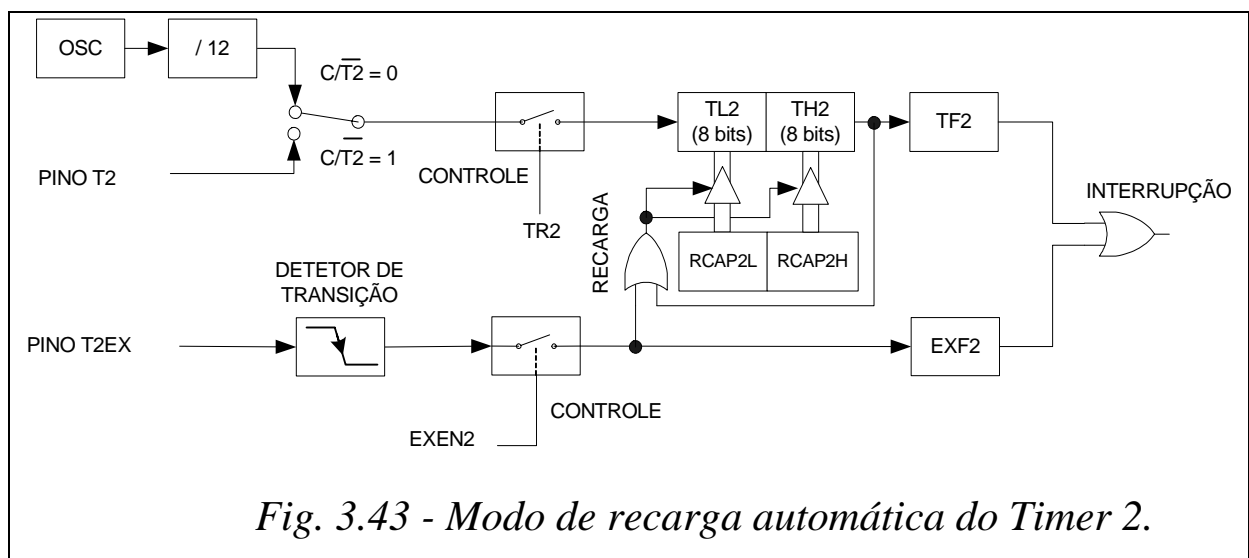
	-	-	-	-	-	-	T2OE	DCEN
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
-	Reservado para uso futuro							
T2OE	Em 1 lógico habilita a saída do <i>Timer 2</i> .							
DCEN	Em 1 lógico permite que o <i>Timer 2</i> seja configurado como contador crescente/decrescente.							

Fig. 3.42 - Registrador de Controle de Modo do Timer 2 - T2MOD.

Quando $DCEN = 0$, o Timer 2 é um contador crescente e o bit $EXEN2$ de $T2CON$ seleciona duas opções:

- Se $EXEN2 = 0$, o estouro do Timer 2 coloca $TF2$ em nível lógico 1 e recarrega os seus registros com os valor de 16 bits presente nos registradores $RCAP2L$ e $RCAP2H$, os quais são definidos por software;
- Se $EXEN2 = 1$, o Timer 2 trabalha do mesmo modo, porém uma transição de 1 para 0 na entrada externa $T2EX$ também faz a recarga de 16 bits e, ainda, coloca em 1 o bit $EXF2$ em $TCON2$.

A Fig. 3.43 ilustra o modo de recarga automática.



Quando $DCEN = 1$, o Timer 2 está habilitado para contar para cima ou para baixo, como mostra a Fig. 3.44. Nesse modo, o pino $T2EX$ controla a direção da contagem. Um 1 lógico em $T2EX$ faz com que o Timer 2 conte para cima e estoure em $0FFFFh$, ocasionando um

1 lógico no bit TF2. Este *overflow* causa a recarga do valor de 16 bits presente em RCAP2L e RCAP2H, respectivamente, em TL2 e TH2. Um 0 lógico em T2EX faz o Timer 2 contar para baixo. Quando TH2 e TL2 igualam-se aos valores armazenados em RCAP2L e RCAP2H ("*underflow*"), o bit TF2 vai para nível lógico 1 e o valor 0FFFFh é recarregado nos registradores do timer.

O bit EXF2 comuta quando ocorre um *overflow/underflow* do *Timer 2*, o que permite que ele seja usado como um décimo sétimo bit de resolução. Em contrapartida, nesse modo, EXF2 não sinaliza interrupção.

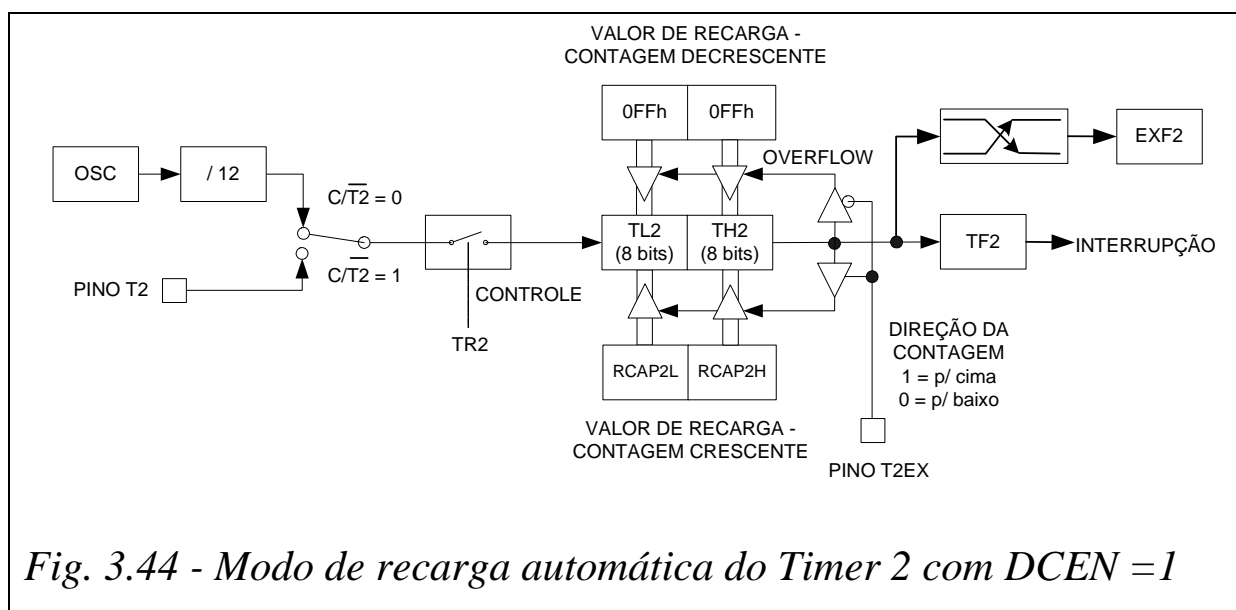


Fig. 3.44 - Modo de recarga automática do Timer 2 com $DCEN = 1$

c. Modo gerador de taxa de transmissão

Este modo é selecionado por $RCLK = 1$ e/ou $TCLK = 1$ e será descrito na seção que discute a interface serial.

d. Saída de relógio programável

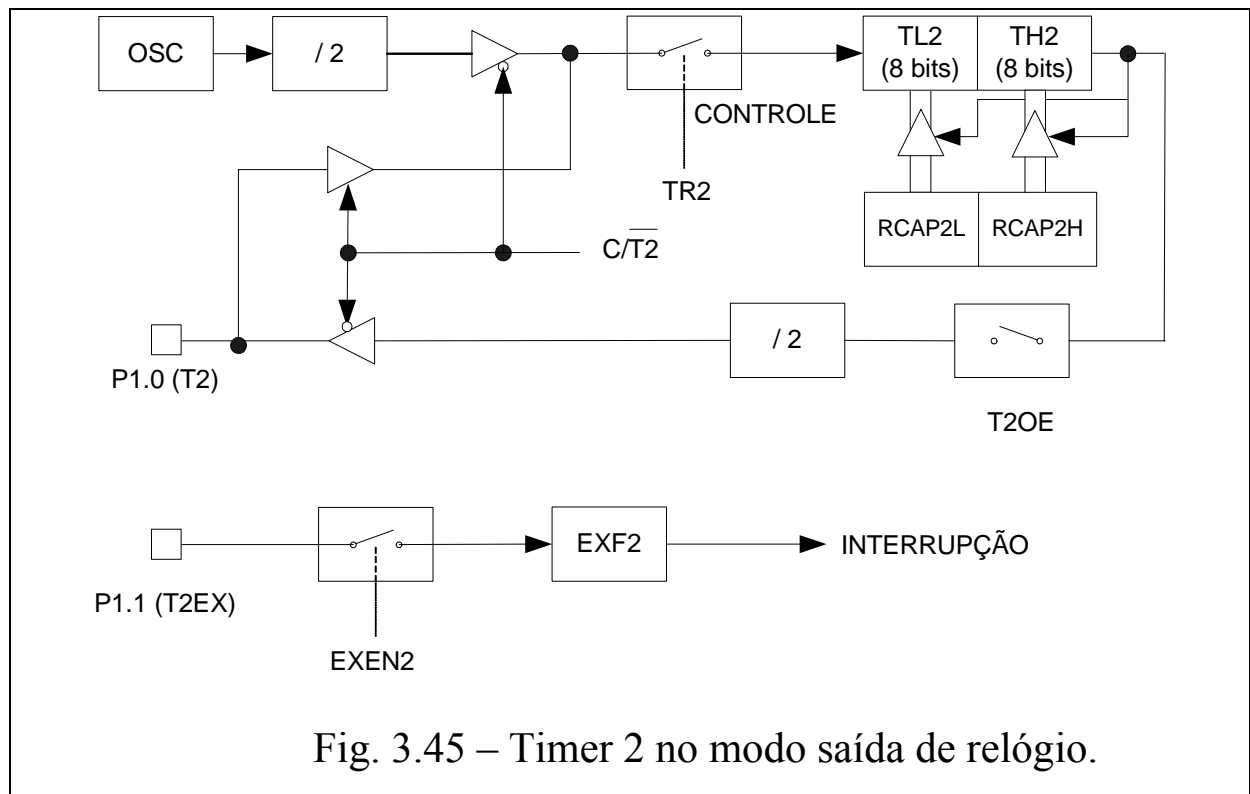
Um relógio de razão cíclica 50% pode ser programado sair no pino P1.0, conforme mostra Fig. 3.45. Esse pino, além de ser um terminal regular de E/S, tem duas funções alternativas:

- Entrada de relógio externo para o *Timer 2*;
- Saída de relógio de razão cíclica 50%, que excursiona de 61 Hz a 4 Mhz para uma frequência de operação de 16 MHz.

Para configurar o Timer 2 como um gerador de clock, o bit $C/\bar{T}-2$ no registrador T2CON deve ser posto em 0 lógico e o bit T2OE do registrador TMOD deve estar em 1 lógico. O bit TR2 liga e desliga o timer. A frequência da saída do relógio é calculada pela expressão a seguir.

$$\text{Frequência de saída do relógio} = \frac{f_{\text{osc}}}{4 \cdot [(65536 - (\text{RCAP2L}, \text{RCAP2H}))]} \quad (3.1)$$

Nesse modo, o *overflow* do *Timer 2* não gerará uma interrupção.



3.8.3 – O Timer Cão de Guarda Programável (WDT)

O Cão de Guarda opera a partir de um oscilador independente. Os bits de prescaler PS0, PS1 e PS2 do registrador WMCON (Fig 3.10) são usados para determinar o período do Cão de Guarda de 16 ms a 2048 ms, sendo que os períodos disponíveis são mostrados na Tabela 3.12. É importante salientar que o período real (para $V_{CC} = 5\text{ V}$) está dentro de uma faixa de $\pm 30\%$ do nominal.

TABELA 3.12

SELEÇÃO DO PERÍODO DO TIMER CÃO DE GUARDA

Bits de prescaler do Cão de Guarda			Período (nominal)
PS2	PS1	PS0	
0	0	0	16 ms
0	0	1	32 ms
0	1	0	64 ms
0	1	1	128 ms
1	0	0	256 ms
1	0	1	512 ms
1	1	0	1024 ms
1	1	1	2048 ms

O Cão de Guarda é desabilitado durante o *reset* e no modo *Power Down*. Ele é habilitado colocando em 1 lógico o bit WDTEN no registrador WMCON e inicializado colocando em 1 lógico o bit WDTRST de WMCON. Quando o Cão de Guarda termina a contagem sem ser inicializado ou desabilitado, um pulso de reset interno é gerado e a CPU é inicializada.

3.9 – Interface Serial

A porta serial dos microprocessadores MCS-51™ é *full-duplex* e de recepção *bufferizada*, isto é, ela pode receber um segundo byte antes que um byte previamente recebido tenha sido lido a partir do registro de recepção. Entretanto, para não haver perda de dados, o

primeiro byte tem de ser lido antes que recepção do segundo esteja completa. Os registros de recepção e transmissão são acessados no Buffer de dados serial, o SBUF - . Quando um dado é transferido para o SBUF carrega-se o registro de transmissão e quando um dado é lido a partir do registrador SBUF acessa-se o registrador de recepção. Portanto, o registrador SBUF é, na realidade composto por dois registradores fisicamente separados, um para recepção e outro para transmissão.

A porta serial pode operar em quatro modos, os quais serão descritos a seguir. Em todos estes modos, a transmissão é iniciada pelo uso de qualquer instrução que usa SBUF como destino.

3.9.1 – Modo 0 – Modo Síncrono

Neste modo de operação os dados entram e saem pela linha RXD (P3.0) e o sinal de clock para sincronismo sai através da linha TXD (P3.1). A taxa de transmissão é fixa em 1/12 da frequência do oscilador, sendo transmitidos ou recebidos oito bits por vez, iniciando com o menos significativo (LSB).

3.9.2 – Modo 1 – Modo Assíncrono de 10 bits e taxa variável

Nesse e nos modos seguintes os dados são transmitidos por TXD e recebidos por RXD. O modo 1, que opera com taxa variável, utiliza 10 bits: um bit de início de nível lógico zero (start bit), os oito bits de dados (primeiro o LSB) e um bit de parada (stop bit) de nível

lógico 1. Na recepção o bit de parada vem pelo bit RB8 do registrador SCON (*Serial Port Control Register*), estudado na seção 3.8.6.

3.9.3 – Modo 2 – Modo Assíncrono de 11 bits e taxa programável

Nesse modo são transmitidos ou recebidos 11 bits: um bit de início de nível lógico zero (start bit), os oito bits de dados (primeiro o LSB), um bit programável e um bit de parada (stop bit) de nível lógico 1. Na transmissão, o nono bit (TB8 em SCON) pode receber o valor 0 ou 1; na recepção, o nono bit vem pelo bit RB8 do registrador SCON e o bit de parada é ignorado. A taxa de transmissão é programável em em 1/32 ou 1/64 da frequência do oscilador.

3.9.4 – Modo 3 – Modo Assíncrono de 11 bits e taxa variável

A única diferença entre este modo e o anterior (Modo 2), é que este modo opera com taxa de transmissão variável.

3.9.5 - Comunicação Multiprocessadores

Os modos 2 e 3, onde 9 bits são recebidos antes do bit de parada, foram pensados para facilitar a comunicação multiprocessadores. O canal pode ser programado de tal forma que ao receber o bit de parada, a interrupção por canal serial é ativada apenas se o nono bit recebido RB8 for igual a 1. Esta função é habilitada colocando-se em nível lógico 1 o bit SM2 em SCON.

A interface serial para comunicação serial pode ser usada da forma indicada a seguir. Quando o processador mestre vai transmitir uma seqüência de dados para um dos seus escravos, ele primeiro envia um byte de endereço que identifica o escravo desejado. Um byte de endereço difere de um byte de dados, pois no primeiro o nono bit é 1 e no segundo ele é zero. Com $SM2 = 1$, nenhum escravo é interrompido por um byte de dados ($TB8 = 0$). Um byte de endereço ($TB8 = 1$) ao contrário, interrompe todos os escravos, o que possibilita que cada escravo examine o byte recebido e verifique se é ele que está sendo endereçado. O escravo endereçado coloca em nível lógico 0 o seu bit $SM2$, preparando-se para receber os bytes de dados que virão. Os escravos que não foram endereçados mantêm em nível lógico 1 seus bits $SM2$ e ignoram os bytes de dados. A Fig. 3.46 ilustra esse processo.

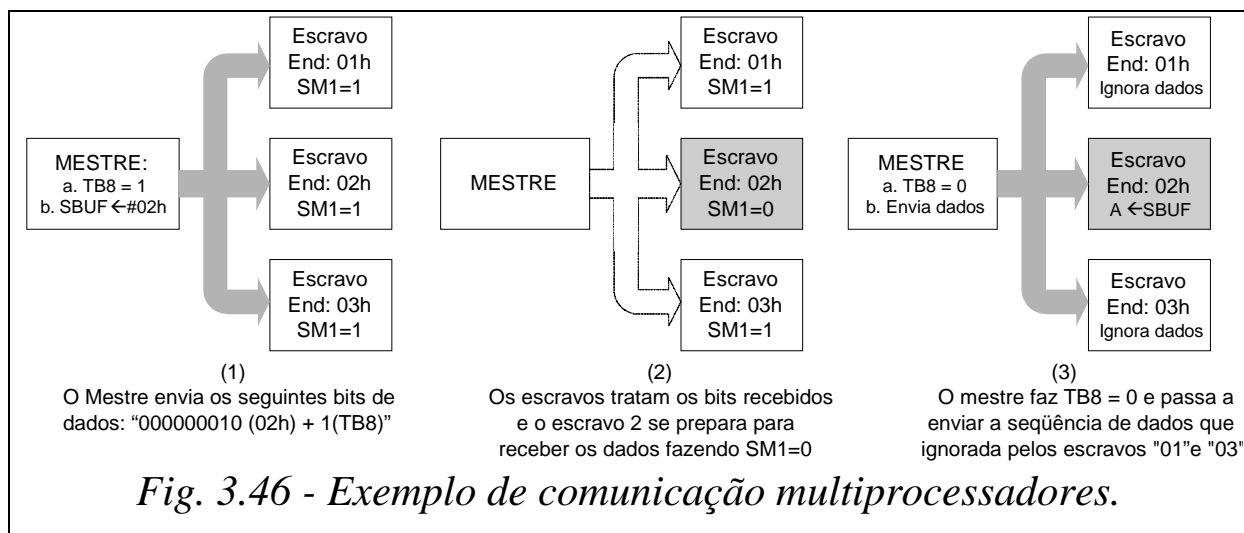
Resumindo, na comunicação multiprocessadores, o sinal para carregar $SBUF$ e $RB8$ e solicitar interrupção por recepção (fazer $RI = 1$) é gerado se e somente se as seguintes condições são encontradas:

1. $RI = 0$
2. $SM2 = 0$ OU $RB8=1$

Se estas condições não são encontradas, os dados recebidos são irremediavelmente perdidos e RI não vai para 1. Se ambas as condições são encontradas, o nono bit recebido vai para $RB8$ e os primeiros oito bits de dados para $SBUF$.

O bit $SM2$ não tem nenhum efeito no modo 0 e no modo 1

pode ser usado para checar a validade do bit de parada. Na recepção no modo 1, se $SM2 = 1$, a interrupção por recepção só é ativada se um bit de parada válido é recebido.



3.9.6 - O Registrador de controle do canal serial - SCON

O Registrador de controle do canal serial - SCON é mostrado na Fig. 3.47. O registrador contém os bits de seleção de modo, o nono bit de dados para transmissão e recepção ($TB8$ e $RB8$) e os bits de interrupção do canal serial (TI e RI).

3.9.7 - Taxas de Transmissão

a. Modos 0 e 2

A taxa de transmissão no modo 0 é fixa e é calculada pela seguinte equação:

$$\text{Taxa de Transmissão no Modo 0} = \frac{f_{osc}}{12} \quad (3.2)$$

onde f_{osc} é a frequência do oscilador.

Registrador SCON – endereço 98h ; **Valor na inicialização** = 00000000b

	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
SM0	Bit 0 de modo do canal serial.							
SM1	Bit 1 de modo do canal serial.							
SM2	Habilita a característica de comunicação multiprocessadores nos modos 2 e 3. Nestes modos, se SM2 = 1, o flag RI não será ativado caso o nono bit recebido (RB8) seja 0. No modo 1, se SM2 = 1, RI não será ativado se um bit de parada válido não for recebido. No modo 2, SM2 pode ser 0.							
REN	Habilita a recepção serial quando posto em nível lógico 1 por software. Quando colocado em nível 0 ele desabilita a recepção							
TB8	O nono bit que será transmitido nos modos 2 e 3. Controlado por software.							
RB8	O nono bit recebido nos modos 2 e 3. No modo 1, se SM2 = 0, RB8 será o bit de parada que foi recebido. No modo 0 ele não é usado							
TI	Flag de interrupção por transmissão. Colocado em nível lógico 1 no final da transmissão do 8 bit no modo 0 ou ao iniciar-se a transmissão do bit de parada nos outros modos, em qualquer transmissão serial. Deve ser colocado em nível 0 por software.							
RI	Flag de interrupção por recepção. Colocado em nível lógico 1 no final da recepção do 8 bit no modo 0 ou na metade da recepção do bit de parada nos outros modos, em qualquer recepção serial (exceções, ver SM2). Deve ser colocado em nível 0 por software.							

SM0	SM1	Modo	Descrição	Taxa de Transmissão
0	0	0	Registrador de deslocamento	fixa ($f_{osc}/12$)
0	1	1	UART de 8 bits	variável(controlado por timer)
1	0	2	UART de 9 bits	fixa ($f_{osc}/64$ ou $f_{osc}/32$)
1	1	3	UART de 9 bits	variável(controlado por timer)

Fig. 3.47 – Registrador SCON.

No modo 2, a taxa de transmissão depende do valor do bit 7 do registrador PCON⁶ - SMOD. Se SMOD = 0, seu valor no reset, a taxa de transmissão é 1/64 da frequência do oscilador; se SMOD = 1, a taxa de transmissão é 1/32 da frequência do oscilador, conforme indica a equação 3.3.

$$\text{Taxa de Transmissão no Modo 2} = \frac{2^{SMOD}}{64} \cdot f_{osc} \quad (3.3)$$

b. Modos 1 e 3

Nos modos 1 e 3 de operação do canal serial podem ser usados os timers 1 e 2 para gerar a taxa de transmissão.

• Empregando o Timer 1

Quando o Timer 1 é usado, a taxa de transmissão é determinada pela sua taxa de estouro e pelo valor de SMOD, de acordo com a equação 3.4.

$$\text{Taxa de Transmissão - Modos 1, 3} = \frac{2^{SMOD}}{32} \cdot \text{Taxa de estouro}_{\text{TIMER1}} \quad (3.4)$$

Nesta aplicação a interrupção por Timer 1 deve ser desabilitada, sendo que ele pode ser configurado como temporizador ou contador em qualquer um de seus modos de operação. Habitualmente, ele é configurado para operar como temporizador no

⁶ “Power Control Register”, apresentado na seção 3.9, que trata dos modos de redução de consumo.

modo de recarga automática (TMOD = 0010φφφφb). Neste caso a taxa de transmissão é dada pela equação abaixo.

$$\text{Taxa de Transmissão - Modos 1, 3} = \frac{2^{\text{SMOD}}}{32} \cdot \frac{f_{\text{OSC}}}{12 \cdot (256 - \text{TH1})} \quad (3.5)$$

Por sua vez, o valor de recarga do Timer 1 para gerar uma certa taxa de transmissão é dado pela equação 3.6.

$$\text{TH1} = 256 - \left(2^{\text{SMOD}} \cdot \frac{f_{\text{OSC}}}{384 \cdot \text{taxa}} \right) \quad (3.6)$$

Para alcançar taxas de transmissão baixas com o Timer 1, o programador pode deixar a interrupção deste habilitada e configurá-lo para operar como temporizador no modo 1 (TMOD = 0001φφφφb), usando a interrupção do Timer 1 para fazer a recarga de 16 bits.

A Tabela 3.13 apresenta as taxas de transmissão usualmente utilizadas e como elas podem ser obtidas com o Timer 1.

• Empregando o Timer 2

Fazendo TCLK = 1e/ou RCLK = 1 no registrador T2CON o Timer é selecionado como um gerador de taxa de transmissão, conforme ilustra a Fig. 3.48. Esse modo é similar ao modo de recarga automática, no qual um estouro em TH2 recarrega os registradores do Timer 2 com o valor de 16 bits presente em RCAP2L e RCAP 2H, os quais são determinados por software. Nesse caso, a taxa de transmissão é determinada pela sua taxa de estouro de acordo com a equação a seguir:

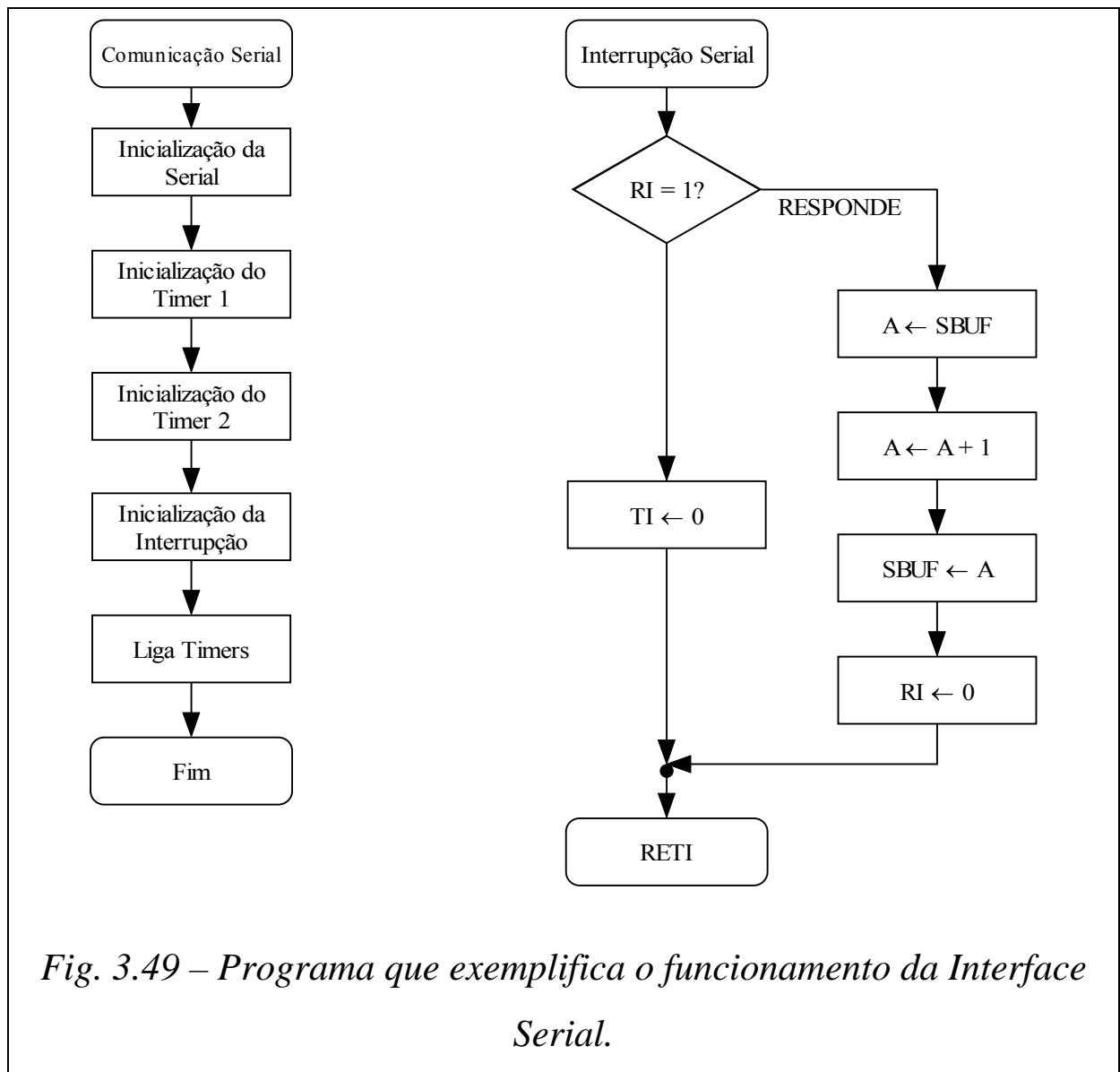
TABELA 3.13
TAXAS DE TRANSMISSÃO HABITUALMENTE USADAS COM
O TIMER 1 NOS MODOS 1,3

Taxa (Hz)	f_{OSC} (MHz)	SMOD	Valor de Recarga	Taxa Real	Erro (%)
9600	12	1	0F9h	8923	7
2400	12	0	0F3h	2404	0,16
1200	12	0	0E6h	1202	0,16
19200	11,0592	1	0FDh	19200	0
9600	11,0592	0	0FDh	9600	0
4800	11,0592	0	0FAh	4800	0
2400	11,0592	0	0F4h	2400	0
1200	11,0592	0	0E8h	1200	0

$$\text{Taxa de Transmissão - Modos 1, 3} = \frac{\text{Taxa de estouro}_{\text{TIMER2}}}{16} \quad (3.7)$$

Nessa aplicação o Timer 2 pode ser configurado para operar como temporizador ou contador. Habitualmente, ele é configurado como temporizador ($C/\bar{T} - 2 = 0$). Normalmente, um timer incrementa a cada ciclo de máquina, entretanto, quando usado como gerador de taxa de transmissão, o Timer 2 incrementa a cada mudança de estado ($1/2 f_{\text{OSC}}$). Nesse caso, a taxa é dada pela seguinte equação:

$$\text{Taxa de Transmissão - Modos 1, 3} = \frac{f_{\text{OSC}}}{32 \cdot [(65536 - (\text{RCAP2L}, \text{RCAP2H}))]} \quad (3.8)$$



O valor de recarga do Timer 1 para gerar a taxa de recepção, considerando a frequência do oscilador igual a 11,0592 MHz, é obtido da equação (3.6) como segue:

$$TH1 = 256 - \left(2^0 \cdot \frac{11,0592 \text{ MHz}}{384 \cdot 2400} \right) = 244$$

Por sua vez, o valor de recarga para o par RACP2H, RCAP2L do Timer 2 é obtido a partir da equação 3.8 como segue:

$$RCAP2H, RCAP2L = 65536 - \frac{11,0592\text{MHz}}{32 \cdot 4800} = 65464 = \text{FFB8h}$$

Assim, RCAP2H = 0FFh e RCAP2L = 0B8h.

A seguir apresentamos a listagem em assembly obtida a partir do fluxograma da Fig. 3.49, onde acrescentamos alguns comentários para facilitar a compreensão do programa.

Listagem em assembly do programa da Fig. 3.49

```

$ INCLUDE (c:\fsi\inc\reg52.inc)      ; inclui o arq. de definições do 8052
    SJMP main                        ; Desvia para o programa principal

; Sub-rotina de interrupção por canal serial

    ORG 23h                          ;
    JB RI, responde                  ; Se recebeu dado, responde
    CLR TI                           ; Se foi transmissão, habilita nova
                                   ; transmissão
    RETI                             ; Retorno de interrupção
responde: MOV A, SBUF                ; Coloca dado recebido no
                                   ; acumulador
    INC A                           ; Soma 1 ao conteúdo do acumulador
    MOV SBUF, A                     ; Transmite o conteúdo do
                                   ; acumulador
    CLR RI                           ; Habilita nova recepção
    RETI                             ; Retorno de interrupção

; Programa Principal - Inicializações
main:  MOV SCON, #01010000b          ; Canal serial no modo 1, com SM2 =
                                   ; 0 e recepção habilitada (REN = 1)
    MOV TMOD, #00010000b            ; Timer 1 no modo 2, operando como
                                   ; temporizador (C/T - 1) = 0).
    MOV TH1, #244                    ; Carrega valor de recarga para o
                                   ; Timer 1.
    MOV TL1, #244                    ; Carrega valor inicial para o Timer 1.
    MOV T2CON, #00010000b           ; Timer 2 no modo gerador de taxa de
                                   ; transmissão (TCLK = 1)
    MOV TH2, #0FFh                   ; Carrega valor inicial para o Timer 2.
    MOV TL2, #0B8h
    MOV RCAP2H, #0FFh                ; Carrega valor de recarga para o

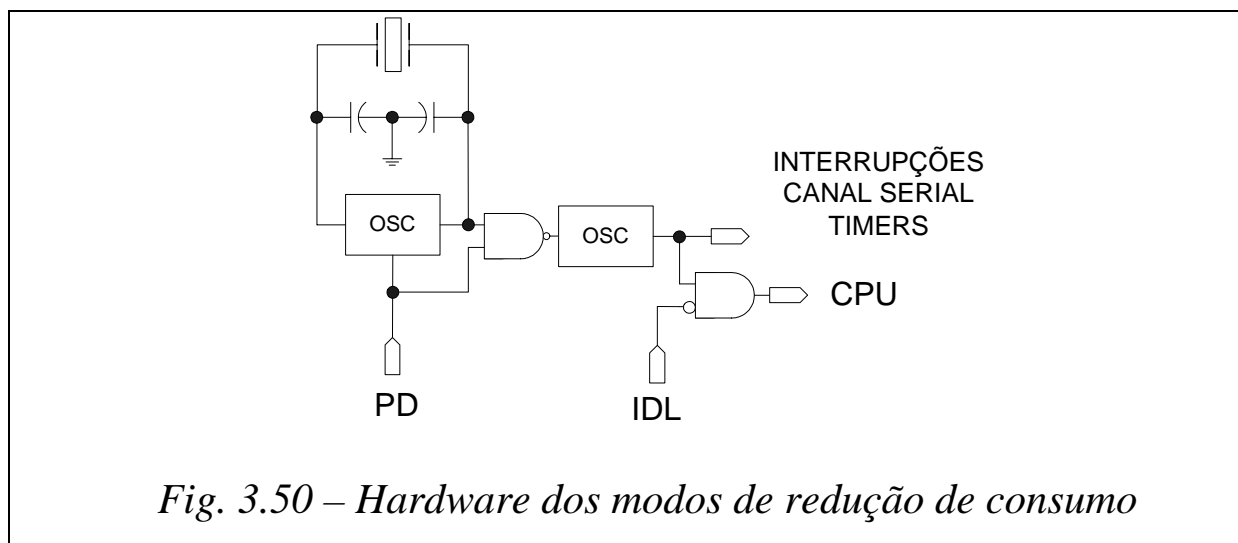
```

Listagem em assembly do programa da Fig. 3.49

MOV RCAP2L, #0B8h	Timer 2.
MOV IE, #10010000b	; Habilita interrupção por canal serial.
SETB TR1	; Liga Timer 1
SETB TR2	; Liga Timer 2
SJMP \$; Aguarda a interrupção
END	; fim de compilação

3.10 – Modos de Redução de Consumo

Os Microcontroladores CHMOS, como o AT89S8252, possuem dois modos de redução de consumo, o modo de espera (*Idle*) e o modo de baixo consumo (*Power Down*). A Fig. 3.50 mostra o circuito interno que implementa esta característica.

**3.10.1 – Modo de Espera**

Uma instrução que coloca 1 lógico no bit IDL do registrador PCON (Fig. 3.29) inicia o modo de espera. Neste modo, o sinal de relógio interno é desconectado da CPU, mantendo em operação as interrupções, os timers e o canal serial. O estado da CPU é preservado

no seu todo: o Stack Pointer, o contador de programa, o PSW, o Acumulador e todos os outros registradores mantêm seus dados durante o modo de espera. Os pinos das portas mantêm seus estados lógicos que tinham antes do modo de espera ser ativado e os sinais ALE e PSEN mantêm nível lógico 1. No modo de espera a corrente consumida é reduzida para cerca de 15% da consumida pelo dispositivo em plena atividade.

O modo de espera pode ser finalizado de duas maneiras:

- A ativação de qualquer interrupção habilitada, levará o bit IDL para 0 lógico. A interrupção será atendida e após a instrução RETI será executada a instrução seguinte àquela que colocou o dispositivo em espera. Os bits GF0 e GF1 podem ser usados para indicar se uma interrupção ocorreu durante a operação normal ou durante a espera. Por exemplo, a instrução que ativa o modo de espera pode também colocar em 1 lógico um ou ambos os bits. Assim, quando o modo de espera é finalizado por uma interrupção, a rotina de atendimento pode examinar GF0/GF1.
- A geração de um reset por hardware. A CPU inicia a execução do programa a partir da instrução que segue aquela que chamou o modo de espera.

3.10.2 – Modo de Baixo Consumo

Uma instrução que coloca 1 lógico no bit PD do registrador PCON (Fig. 3.51) é a última instrução executada antes do início do modo de baixo consumo. Neste modo, o oscilador do dispositivo para e ocasiona a suspensão de todas as funções, porém os conteúdos da RAM interna e dos SFRs são mantidos. Por sua vez, os sinais ALE e PSEN vão para nível lógico 0. No modo de baixo consumo, o dispositivo consome menos do que 15 μA .

No AT89S8252, a saída desse modo pode ser ocasionada por:

- Um reset de Hardware, o qual deve ser mantido por um tempo suficiente (cerca de 10ms) para permitir que o oscilador reinicie e estabilize. O reset redefine todos os SFRs sem, contudo, alterar o conteúdo da RAM interna;
- Uma interrupção externa habilitada como sensível a nível antes do início do modo de baixo consumo. A sub-rotina de atendimento da interrupção inicia cerca de 16 ms após o pino de interrupção ser ativado.

No modo de baixo consumo, V_{cc} pode ser reduzido para cerca de 2 V. Entretanto, ele não deve ser reduzido antes do modo de espera ser chamado e deve ser restaurado para o nível normal (4 a 6 V para o AT89S8252) antes do modo de baixo consumo terminar.

Registrador PCON – endereço 87h ; **Valor na inicialização** = 0φφ00000b

	SMOD	-	-	POF	GF1	GF0	PD	IDL
bit	7	6	5	4	3	2	1	0
Símbolo		Função						
SMOD		Quando em 1 lógico, este bit dobra a taxa de transmissão do canal serial gerada pelo Timer 1 nos modos 1, 2 ou 3.						
-		Reservado para futuras implementações.						
POF		O <i>Power Off Flag</i> . Este bit é levado a 1 lógico durante a energização do sistema.						
GF1/GF2		<i>Flags</i> de propósito geral.						
PD		O bit <i>Power Down</i> . Quando colocado em 1 lógico, ativa o modo de baixo consumo.						
IDL		O bit <i>Idle Mode</i> . Quando colocado em 1 lógico, ativa o modo de espera.						

Fig. 3.51 – O Registrador PCON.

3.11 – Acesso a Memória Externa

Os microcontroladores da família MCS-51 permitem o acesso tanto a memória externa de programa como a memória externa de dados. O acesso à memória de programa externa usa o sinal PSEN (Program Store Enable) para acessar a leitura. Por sua vez, o acesso a memória de dados externa utiliza os sinais RD e WR, para acessar a memória.

3.11.1 – Acesso a Memória de Programa Externa

A configuração de Hardware necessária para a execução de programa colocado em memória externa é apresentada na Fig. 3.52. Observe, na referida figura, que as 16 linhas de E/S são dedicadas as

funções de barramento durante a busca de programa na memória externa. A Porta 0 serve como um barramento multiplexado de dados e endereços. Ela emite o byte menos significativo do Contador de Programa (PCL) como um endereço e então vai para um estado de flutuação enquanto aguarda a chegada do byte de código proveniente da memória de programa. Durante o tempo que o PCL é válido em 0, o sinal ALE (*Address Latch Enable*) trava este byte em um *latch* de endereços. Enquanto isso, a Porta 2 emite o byte mais significativo do Contador de Programa, PCH. Com o endereço completo disponível, o sinal PSEN acessa a memória externa e o microcontrolador lê o byte de código.

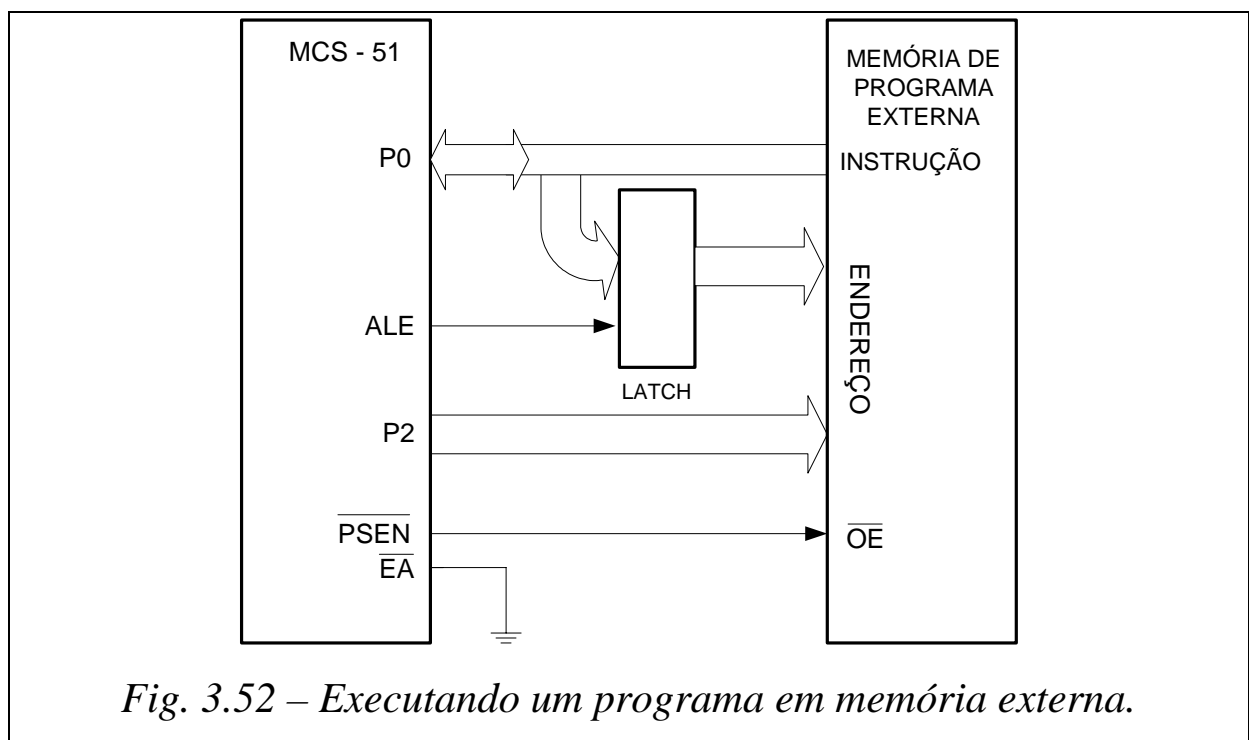


Fig. 3.52 – Executando um programa em memória externa.

3.11.2 – Acesso a Memória de Dados Externa

A Fig. 3.53 mostra uma configuração de hardware para

acesso a 2 kbytes de memória RAM externa (8 x 256 bytes), onde a CPU busca o programa na memória interna. A Porta 0, novamente, serve como um barramento multiplexado de dados/endereços para a RAM e 3 linhas da Porta 2 são usadas para mapeá-la. Os sinais RD e WR são gerados pela CPU quando necessários.

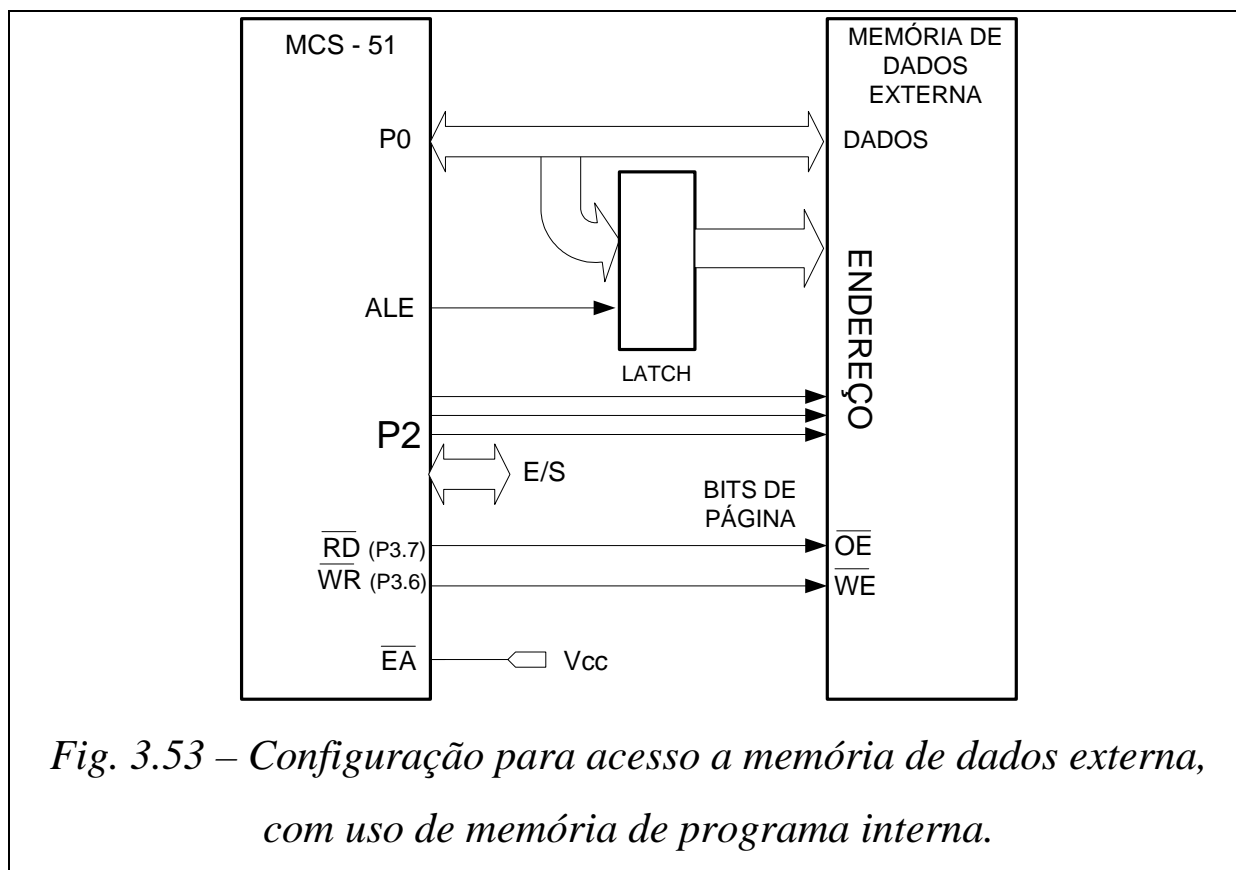


Fig. 3.53 – Configuração para acesso a memória de dados externa, com uso de memória de programa interna.

A Tabela 3.14 lista as instruções que acessam a memória de dados externa e a EEPROM interna do AT89S8252, todas de endereçamento indireto. O endereçamento de 16 bits, @DPTR, usa todos os bits da Porta 2 como barramento de endereço. Por outro lado, o endereçamento de 8 bits, @Ri, permite que poucos kbytes de RAM sejam usados sem sacrificar toda a Porta 2, conforme mostra a Fig.

3.31. Os pinos da Porta 2 são controlados por alguma instrução de saída precedendo *MOVX*.

TABELA 3.14
INSTRUÇÕES DE ACESSO A MEMÓRIA DE DADOS EXTERNA

ENDEREÇOS COM	Mnemônico	Operação	Períodos de clock
8 bits	MOVX A, @Ri	Lê a RAM externa em (Ri)	24
8 bits	MOVX @Ri, A	Escreve na RAM externa em (Ri)	24
16 bits	MOVX A, @DPTR	Lê a RAM externa em (DPTR)	24
16 bits	MOVX @DPTR, A	Escreve na RAM externa em (DPTR)	24

Note que em todos os acessos a RAM externa o acumulador é a única opção de destino ou fonte dos dados.

Exemplo: Uma RAM externa de 256 bytes é conectada a Porta P0 de um AT89S8252 e a porta P3 provê as linhas de controle RD e WR. Se R0 e R1 contém, respectivamente, 24h e 55h, e a posição de memória 24h da RAM externa contém 20h, a sequência de instruções

MOVX A, @R0

MOV X @R1, A

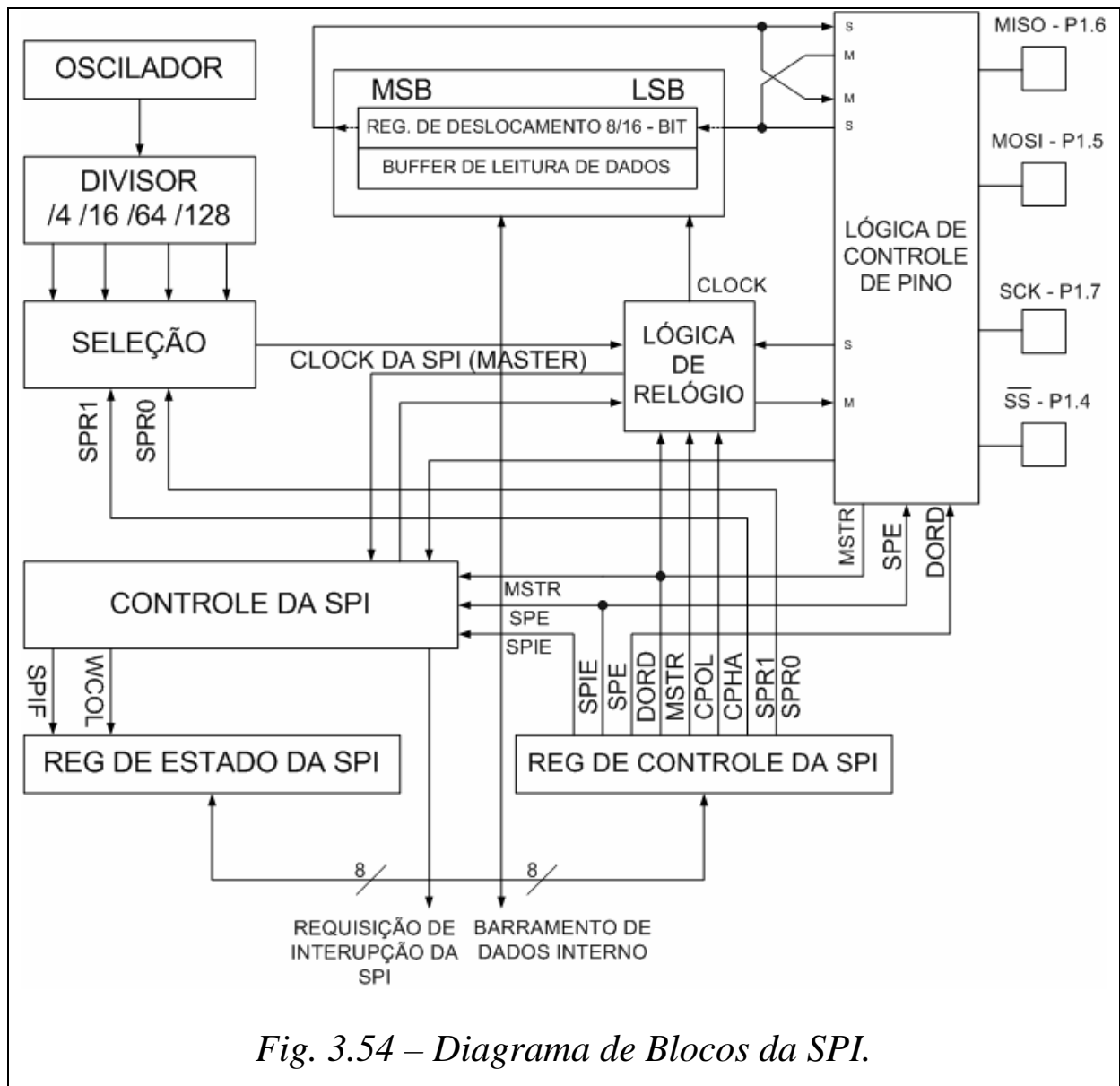
Copia o valor 20h para o acumulador e para o endereço 55h da RAM externa.

3.12 – Interface Serial Periférica - SPI

A SPI permite transferência de dados síncrona de alta velocidade entre o AT89S8252 e dispositivos periféricos ou entre diversos dispositivos AT89S8252. A SPI do AT89S8252, ilustrada na Fig. 3.54, apresenta as seguintes características:

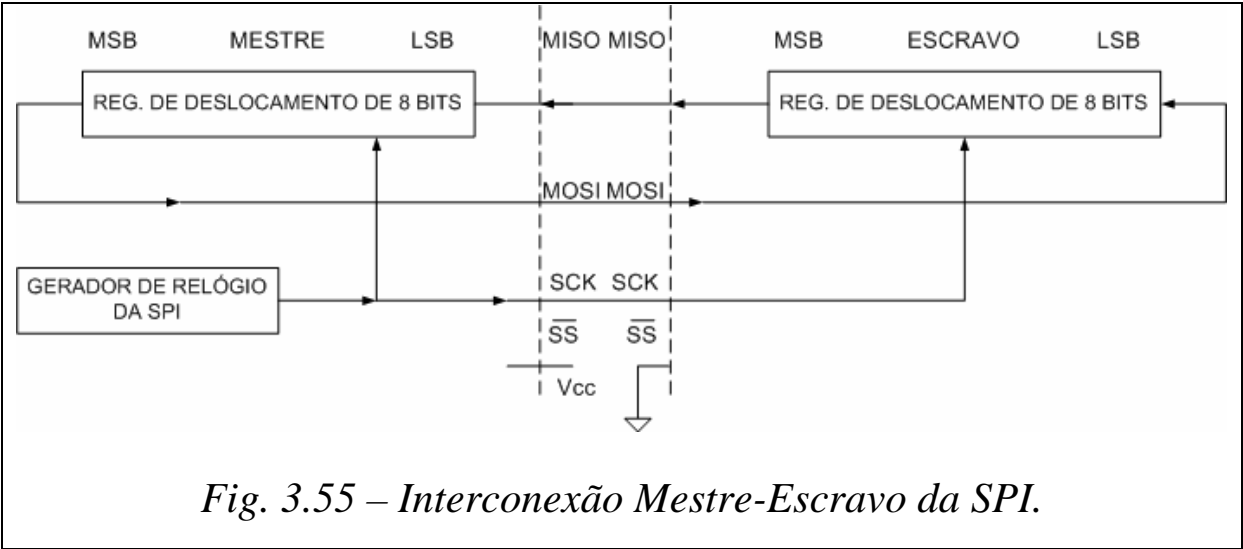
- ✓ Full-Duplex, transferência de dados síncrona a três fios;
- ✓ Operação como Mestre ou Escravo;
- ✓ Transferência a partir do bit mais ou menos significativo;
- ✓ Quatro taxas programáveis;
- ✓ Flag de interrupção por fim de transmissão;
- ✓ Flag de proteção contra colisão de escrita;
- ✓ No Modo Escravo, despertar a partir do Modo de Espera (Idle);

A interconexão entre MCUs mestre e escravo com a SPI é mostrada na Fig. 3.55. O pino SCK é a saída de *clock* no modo mestre e a entrada de *clock* no modo escravo. A operação de escrita no Registrador de dados SPI (SPDR, endereço 86h) liga o gerador de relógio da SPI e os dados escritos deslocam-se do pino MOSI do mestre para o pino MISO do escravo. Após o deslocamento de um byte, o gerador de relógio da SPI para, colocando em nível lógico 1 o flag de fim de transmissão (SPIF) no Registrador de Estado da SPI (SPSR, endereço AAh, Fig. 3.56). Uma interrupção será solicitada a MCU se o bit de habilitação da interrupção SPI (SPIE) no Registrador de Controle da SPI (SPCR, endereço D5h, Fig. 3.57) e o bit ES do Registrados IE estiverem em 1 lógico.



No modo mestre, a entrada de Seleção de Escravo, \overline{SS} , é ignorada e pode ser usada como uma linha de E/S de propósito geral. No modo escravo, \overline{SS} é posta em 0 lógico para selecionar um determinado dispositivo SPI como escravo. Colocando \overline{SS} em nível alto, a SPI escrava é desativada e o pino MOSI – P1.5 pode ser usado como uma entrada. Há quatro combinações de fase e polaridade de SCK com relação ao dado serial, as quais são determinadas pelos bits de controle CPHA e CPOL no registrador SPCR. Para mais detalhes

sobre os formatos de transferência de dados da SPI o leitor deve consultar a folha de dados do AT89S8252.



Registrador SPSR – endereço AAh ; **Valor na inicialização** = 00φφφφφφφb

	SPIF	WCOL	-	-	-	-	-	-
bit	7	6	5	4	3	2	1	0
Símbolo		Função						
SPIF		Flag de Interrupção da SPI. Quando uma transferência serial está completa, o bit SPIF é posto em 1 lógico e uma interrupção é gerada se SPIE = 1 e ES = 1. O bit SPIF é posto em 0 lógico pela leitura do registrador SPSR com os bits SPIF e WCOL em 1 lógico e com o acesso do Registrador de dados da SPI (SPDR).						
WCOL		Flag de colisão na escrita. O bit WCOL é posto em 1 lógico se for feita uma escrita no registrador de dados da SPI (SPDR) durante uma transferência de dados. Durante a transferência de dados, o resultado de leitura do SPDR pode ser incorreta e escrever nele não tem efeito. O bit WCOL é posto em 0 lógico pela leitura do registrador SPSR com os bits SPIF e WCOL em 1 lógico e com o acesso do Registrador de dados da SPI (SPDR).						

Fig. 3.56 – O Registrador SPSR.

Registrador SPCR – endereço D5h ; **Valor na inicialização** = 000001φφb

	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
SPIE	Habilita a Interrupção SPI. Este bit e o bit ES no registrador IE habilitam a interrupção SPI.							
SPE	Habilita SPI. SPI = 1 habilita o canal SPI e conecta seus pinos a Porta 1.							
DORD	Ordem de dados. DOR = 1, a transmissão inicia pelo LSB; DORD = 0, a transmissão inicia pelo MSB.							
MSTR	Seletor de Mestre/Escravo. MSTR = 1, SPI no modo mestre; MSTR = 0, SPI no modo escravo.							
CPOL	Polaridade do clock. Com CPOL = 1, SCK é alto quando ocioso. Quando CPOL = 0, o clock do dispositivo mestre é baixo enquanto não transmite.							
CPHA	Fase do clock. CPHA e CPOL controlam o clock e a relação entre mestre e escravo. Mais informações, consulte a folha de dados							
SPR0 SPR1	Seleção da Taxa de Clock. Estes dois bits controlam a taxa de SCK do dispositivo mestre. SPR1 e SPR0 não tem efeito sobre escravos. A Relação entre SCK e a Frequência do oscilador, F_{OSC} , é apresentada na Tabela 3.15							

Fig. 3.56 – O Registrador SPCR.

TABELA 3.15
RELAÇÃO ENTRE A TAXA DE SCK E A F_{OSC}

SPR1	SPR0	RELAÇÃO
0	0	$F_{OSC} / 4$
0	1	$F_{OSC} / 16$
1	0	$F_{OSC} / 64$
1	1	$F_{OSC} / 128$

3.13 – Programação em C

A linguagem C é uma linguagem que permite operações de alto nível e programação estruturada, mantendo a capacidade de gerar comandos de baixo nível para acessar diretamente a memória de dados e periféricos. Nesta seção, apresentaremos a programação em C a partir do compilador C51 da versão de avaliação do software μ vision da Keil.

3.13.1 – Especificadores e modelos de memória

Existem seis tipos de especificadores de memória:

- *code* – memória de programa, até 64 Kbytes;
- *data* – memória de dados dentro dos primeiros 128 bytes de Ram interna;
- *idata* – memória de dados com direcionamento indireto, acessa os 256 bytes do núcleo C52;
- *bdata* – área de dados com direcionamento bit a bit, 16 bytes (128 bits);
- *xdata* – memória de dados externa, até 64 kbytes;
- *pdata* – memória de dados externa paginada, permite acesso aos primeiros 256 bytes pela porta 0.

Exemplos:

```
unsigned char data a, b;
```

```
int xdata serial[16];
```

```
unsigned char code tab [] = {0x01, 0x02, 0x03, 0x05, 0x05};
```

Através das definições de modelo de memória podemos escolher o uso de um tipo de memória para todo o programa ou um tipo de memória para todas as variáveis que não foram declaradas explicitamente. Os modelos de memória, os quais podem ser definidos na compilação através da opção *model* ou no código fonte através da diretiva `#pragma`, são três:

- *small* – as variáveis e dados locais residirão na memória de dados interna, equivale ao especificador *data*;
- *compact* – as variáveis e dados locais residirão na primeira página da memória de dados externa, equivale ao especificador *pdata*;
- *large* – as variáveis e dados locais residirão na memória de dados externa, equivale ao especificador *xdata*;

A opção ROM permite a configuração do tamanho da memória de programa, sendo permitida três opções:

- *small* – a memória máxima de programa é 2 k. O tamanho máximo de uma função é 2 k. As instruções CALL e JMP são codificadas como ACALL e AJMP;
- *compact* – a memória máxima de programa é 64 k. O tamanho de uma função não deve exceder 2 k. As instruções CALL são codificadas como LCALL e as instruções JMP são codificadas como AJMP;
- *large* – a memória máxima de programa é 64 k. O tamanho máximo de uma função é 64 k. As instruções CALL e JMP são codificadas como LCALL e LJMP.

3.13.2 – Constantes, variáveis e tipos de dados

O compilador C51 do software μ vision da Keil suporta os tipos de dados apresentados na Tabela 3.16.

TABELA 3.16
TIPOS DE DADOS SUPORTADOS PELO C51

Tipo	Nº de bits	Nº de bytes	Faixa de valores
char	8	1	-128 a + 127
unsigned char	8	1	0 a 255
enum	16	2	-32768 a +32768
short	16	2	-32768 a +32768
unsigned short	16	2	0 a 65535
int	16	2	-32768 a +32768
unsigned int	32	4	0 a 65535
long	32	4	-2147483648 a +2147483647
unsigned long	32	4	0 a 4294967295
float	32	4	$\pm 1,175494\text{E}-38$ a $3,402823\text{E}+38$
bit	1		0 ou 1
sbit	1		0 ou 1
sfr	8	1	0 a 255
sfr16	16	2	0 a 65536

Obs - os tipos de dados em negrito não fazem parte do ANSI C, são específicos do 8051.

Os tipos de dados **sbit** são usados para declarar variáveis tipo bit associadas aos registradores de função especial (SFR). Por exemplo, as declarações

```
sbit at 0x90 P10
```


ou

```
sbit P10 = 0x90
```

estabelecem P10 como uma variável tipo bit de direção 0x90 da memória interna. Observe-se que a maioria dos bits de estado e controle dos registradores função especial estão pré-declarados no arquivo REG8252.H.

A declaração bit pode ser empregada para se definir bits de objetos declarados pelo especificador de memória *bdata*, Por exemplo, as declarações

```
char bdata i;  
sbit bit_5 = i^5;
```

atribui o nome bit_5 ao quinto bit da variável i.

3.13.3 – Exemplos

Nesta seção consideramos que o leitor esteja familiarizado com o ANSI C. Antes da análise dos exemplos a seguir, recomenda-se a revisão dos seguintes tópicos da linguagem C:

- Arrays, estruturas e uniões;
- Ponteiros;
- Sentenças de controle – *while* e *do...while*, *if...else*, *for*, *switch*, *case*, *break* e *default*, *continue*;
- Funções.

A seguir, apresentamos, reescrito em C, alguns exemplos já codificados anteriormente em *assembly*.

Exemplo 1 – Gerando uma onda quadrada.

```
#include <REG8252.H>
    unsigned char i;
    void main (void)
    {
        for (;;)                // Laço infinito
        {
            for(i=0;i<28;i++);    // Laço de 250us
                P1_0=~P1_0;        // Complementa P1.0
        }
    }
```

Exemplo 2 – Testa número iguais a 40h no bloco de memória de 50h a 5Fh.

```
#include <REG8252.H>
#define DBYTE ((unsigned char volatile data *) 0)
/* A macro DBYTE permite acessar bytes individuais*/
// na memoria interna do 8051
unsigned char inicial = 0x50;
unsigned char final = 0x5f;
unsigned char teste = 0x40;
unsigned char mem ;
unsigned char iguais = 0;
void main (void)
{
    while(inicial<=final)
    {
        mem = DBYTE [inicial];
        if (mem==teste)
            iguais++;
            inicial++;
    }
    for (;;)
    {
    }
}
```

Exemplo 3 – Sequencial de LEDS controlado por chave de contato momentâneo.

```
#include <REG8252.H>

void atraso()
{
    unsigned int tempo;
    for(tempo=0;tempo<4608;tempo++);
}

void solta()
{
    do{
        while(P1_0) // Fica aqui enquanto P1.0=1
        { }
        atraso(); // Atrasa
    }
    while (P1_0); // Confirma se a chave
                // foi pressionada
}

void press()
{
    do{
        while(P1_0) // Fica aqui enquanto P1.0=0
        { }
        atraso(); // Atrasa
    }
    while (P1_0); // Confirma se a
                // foi solta
}

void main (void)
{
    ACC=0x80;
    while (1)
    {
        solta();
        ACC=ACC>>1; // Desloca à direita
        if(ACC==0) // Verifica-se porque o C
        ACC=0x80; // não tem rotate
        press();
    }
}
```

Exemplo 4 – Contador de pulsos em chave com saída no display de sete segmentos

```
#include <REG8252.H>

void atraso()
{
    unsigned int tempo;
    for(tempo=0;tempo<4608;tempo++);
}

void solta()
{
    do{
        while(P1_0) // Fica aqui enquanto P1.0=1
        { }
        atraso(); // Atrasa
    }
    while (P1_0); //Confirma se a foi press.
}

void press()
{
    do{
        while(P1_0) // Fica aqui enquanto P1.0=0
        { }
        atraso(); // Atrasa
    }
    while (P1_0); // Confirma se a
}

void main (void)
{
    char tabela[10] = {0x01, 0x4F, 0x12,
0x06, 0x4C, 0x24, 0x20, 0x0F, 0x00, 0x04};
    unsigned int i;
    for (;;)
    {
        solta();
        i++;
        if(i==10)
            i=0;
        press();
        P2=tabela[i];
    }
}
```

Exemplo 5 – Programa que usa a interrupção externa 1 para ativar/desativar a onda quadrada e a interrupção externa 0 para alterar a frequência da onda quadrada.

```
#include <REG8252.H>

bit teste = 0;
bit index = 0;

void external0 (void) interrupt 0
{
    index=~index;
}
void external1 (void) interrupt 2
{
    teste=~teste;
}
void main (void)
{
    char i;
    char valores[2] = {11,24};
    IE=0x85;
    TCON=0x05;
    for (;;)                // Laço infinito
    {
        while(teste)
        {}
        for(i=0;i<=valores[index];i++);
        P1_0=~P1_0;        // Complementa P1.0
    }
}
```

Exemplo 6 – Programa que usa o timer 0 para gerar um seqüencial de LEDs.

```
#include <REG8252.H>
char atend;
char rotate;
void timer0 (void) interrupt 1
{
    TH0=0x10;
    atend--;
    if (atend==0)
    {rotate=rotate<<1; // Desloca à esquerda
      if(rotate==0)      // Verifica-se porque
        rotate=0x01;    // o C não tem rotate
    }
    P2=~rotate;
    atend=15;
}

void main (void)
{
    P2=0xFE;
    rotate=0x01;
    atend=15;
    IE=0x82;
    TMOD=0x01;
    TH0=0x10;
    TR0=1;
    for (;;)
    {}
}
```

Exemplo 7 – Programa para escrever em display LCD.

```
#include <REG8252.H>
#include <stdio.h>

// Rotina para enviar comandos ao LCD
void cmlcd(unsigned char c, char cd)
{
    int t;
    P0 = c;
    if (cd==0) P3_7 =0; else P3_7=1;
    P3_6=1;
    P3_6=0;
    for (t=0; t < 8;t++);
    if ((cd==0) && (c<127)) for (t=0;t<650;t++);
}

// Rotina de Inicialização do LCD
void initlcd(void)
{
    cmlcd (0x38,0);
    cmlcd (0x0e,0);
    cmlcd (0x01,0);
}

// Rotina de escrita no LCD
void write(char *c)
{
    for (; *c!=0; c++) cmlcd(*c,1);
}
```

```
void main()  
{  
    initlcd();  
    cmlcd (0x80,0);    // linha 1 do lcd  
    write (" Demonstracao ");  
    cmlcd (0xc0,0);    // linha 2 do lcd  
    write (" Display LCD");  
    for(;;);          // o programa para aqui esse for  
                      // deixa o programa aqui para sempre  
}
```


IV – FERRAMENTAS DE DESENVOLVIMENTO

4.1 – Etapas de Desenvolvimento de um Programa

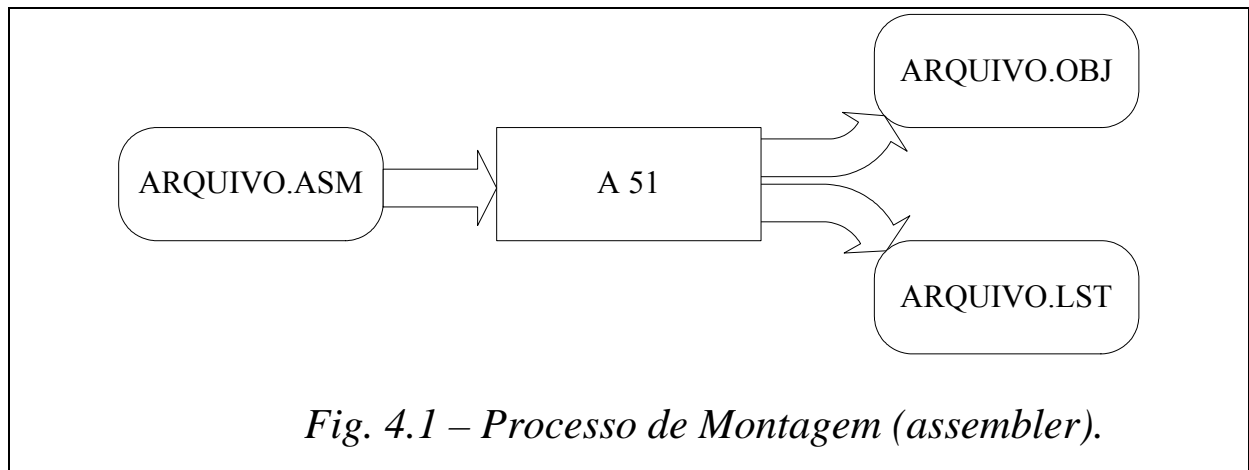
A edição do programa fonte em *assembly* é a primeira etapa para o desenvolvimento de um projeto. Nessa etapa, utiliza-se um editor que permita gravar arquivos em formato texto ASC II. O arquivo fonte de um programa em *assembly* pode incluir controles do *assembler* (montador), diretivas do *assembler* e instruções da linguagem *assembly* da família MCS-51™. Por exemplo, o programa abaixo consiste de quatro comandos – um controle do *assembler*, \$INCLUDE, duas diretivas do *assembler*, ORG e END, e duas instruções da linguagem *assembly*, CLR e SJMP.

```
$INCLUDE (c:\fsi\inc\reg52.inc);inclui o arquivo com  
                                ; definições do 8052  
    ORG 00h  
    CLR P1.0  
    SJMP $  
    END
```

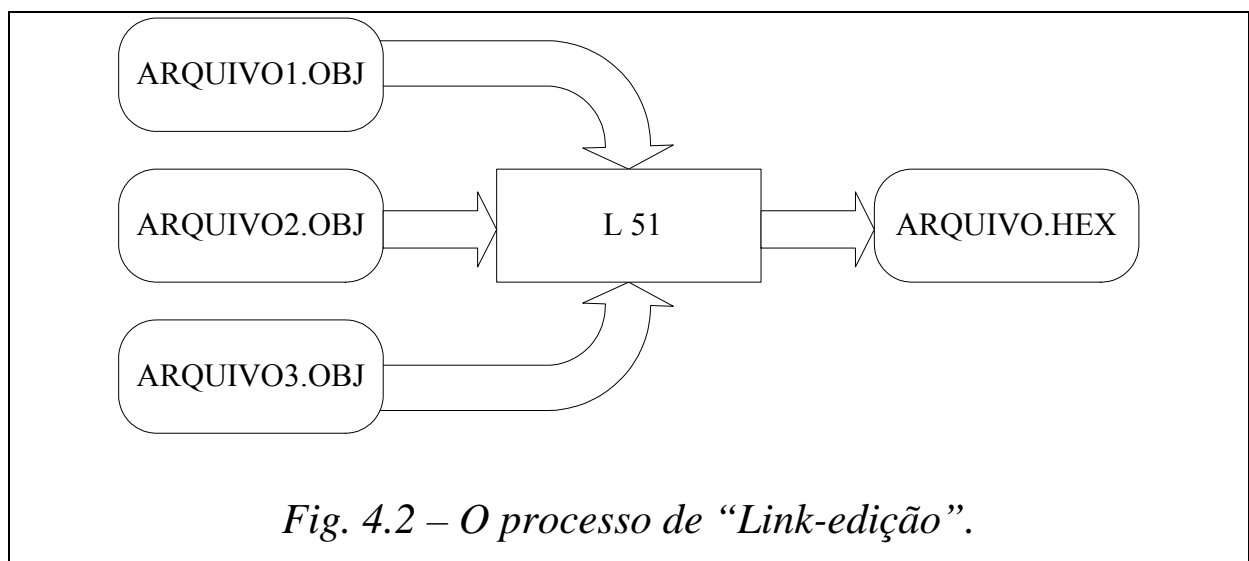
Cada linha de um programa em *assembly* contém apenas um comando que pode iniciar em qualquer coluna. Todo programa em *assembly* do MCS-51™ deve encerrar com a diretiva END.

O *Assembler* é uma ferramenta de software que faz a conversão de um arquivo texto em linguagem *assembly* em um arquivo objeto contendo códigos de operação. O *Assembler A51* da Franklin Software Inc. gera, ainda, um arquivo com extensão .lst

contendo informações úteis ao programador. O Processo de montagem realizado pelo *A51* é mostrado na Fig. 4.1.



O Código objeto gerado precisa ser colocado em um formato que possa ser carregado em um microcontrolador, como o formato *Intel Hex*, formato utilizado pelo software de gravação da Placa de Desenvolvimento AT89S, que será abordado no capítulo V. O *Linker L51* da Franklin Software Inc, tomando um ou mais arquivos objeto, é capaz de gerar o arquivo no formato *Intel Hex*, conforme ilustra a Fig. 4.2.



O arquivo .hex é do tipo texto e apresenta o formato ilustrado pelo esquema a seguir.

:02	0000	00	8023	5B
Número de bytes de programa na linha (≤ 16)	Endereço de carga na memória	Presente em todas as linhas, menos na última	Código de Programa	Checksum = complemento de 2 de (02h+00h+00h+80h+23h)
<p style="text-align: center;">• • •</p> <p>:00000001FF → Última linha do arquivo</p>				

As tarefas de edição, montagem e *link*-edição podem ser executadas no ambiente de desenvolvimento integrado *ProView-32* para *Microsoft Windows* da Franklin Software Inc, disponível em versão *demo* no site www.fsinc.com. O *ProView-32* inclui, ainda, um Depurador/Simulador integrado que permite ao programador testar e depurar sua aplicação antes de implementar o hardware. O depurador/simulador *WinSim* simula uma grande variedade de periféricos como a porta serial, as linhas de E/S e os *timers*.

4.2 – Uma Visão Rápida do *ProView*

Esta seção fornecerá a base para a criação de um projeto, criação e vinculação de arquivos fonte ao projeto, compilação, *link*-edição e depuração do programa.

Para criar um arquivo de projeto, selecione "New" no menu "Project". Esta ação abrirá a caixa de diálogo "New Project" (Fig. 4.3). Digite "c:\fsi\temp\alo.prj" na caixa "Name" e selecione "8051" como o tipo de projeto.

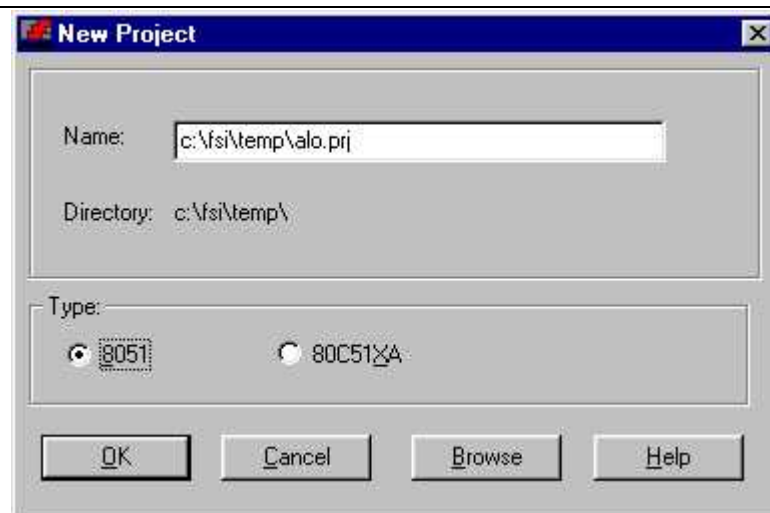
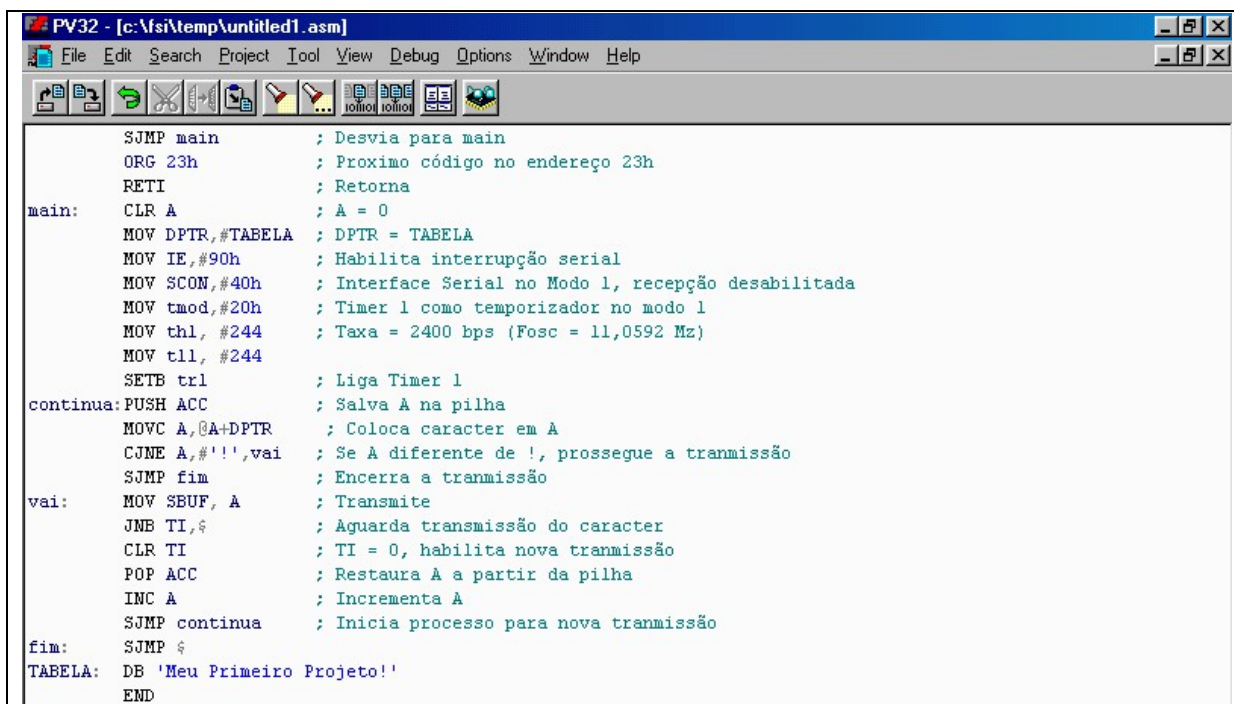


Fig. 4.3 - Caixa de diálogo "New Project".

Para criar o arquivo fonte em *assembly* selecione "New" no menu "File". Após essa ação aparecerá um pequeno menu intitulado "Document Type", onde você selecionará "Assembler Files". Surgirá, então, a janela de edição "untitled1.asm", onde você editará o arquivo fonte em *assembly*, conforme apresenta a Fig. 4.4.

Após a edição, selecione "Save" no menu "File". Esta ação abrirá a caixa de diálogo "Salvar como", onde você digitará "alo" na caixa "Nome do Arquivo" e, após, pressionará [Salvar].

Uma vez gravado o arquivo "alo.asm", você o vinculará ao projeto "alo" selecionando, com a caixa "Project - c:\fsi\temp\alo.prj" ativa, "Add File" no menu "Project". Esta ação abrirá a caixa de diálogo "Add file". Selecione, então, "alo.asm" e *click* em [Abrir].



```
PV32 - [c:\fsi\temp\untitled1.asm]
File Edit Search Project Tool View Debug Options Window Help

; Desvia para main
SJMP main
; Proximo código no endereço 23h
ORG 23h
; Retorna
RETI
main:
; A = 0
CLR A
; DPTR = TABELA
MOV DPTR, #TABELA
; Habilita interrupção serial
MOV IE, #90h
; Interface Serial no Modo 1, recepção desabilitada
MOV SCON, #40h
; Timer 1 como temporizador no modo 1
MOV tmod, #20h
; Taxa = 2400 bps (Fosc = 11,0592 Mz)
MOV th1, #244
MOV tll, #244
; Liga Timer 1
SETB tr1
continua:
; Salva A na pilha
PUSH ACC
; Coloca caracter em A
MOVC A, @A+DPTR
; Se A diferente de '!', prossegue a transmissão
CJNE A, #'!', vai
; Encerra a transmissão
SJMP fim
vai:
; Transmite
MOV SBUF, A
; Aguarda transmissão do caracter
JNB TI, $
; TI = 0, habilita nova transmissão
CLR TI
; Restaura A a partir da pilha
POP ACC
; Incrementa A
INC A
; Inicia processo para nova transmissão
SJMP continua
fim:
SJMP $
TABELA: DB 'Meu Primeiro Projeto!'
END
```

Fig. 4.4 – Arquivo fonte em assembly para o projeto "alo".

Após essa ação, selecione "Make" no menu "Project" para compilar o arquivo fonte e *link*-editar o arquivo objeto gerado na compilação. Quando completar o processo, o *ProView* mostrará na janela "Message" a mensagem apresentada pela Fig. 4.5. Se algum erro for encontrado, o *ProView* o reportará nessa janela.

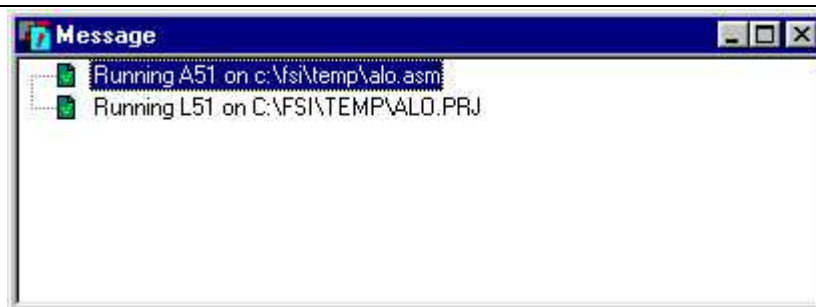
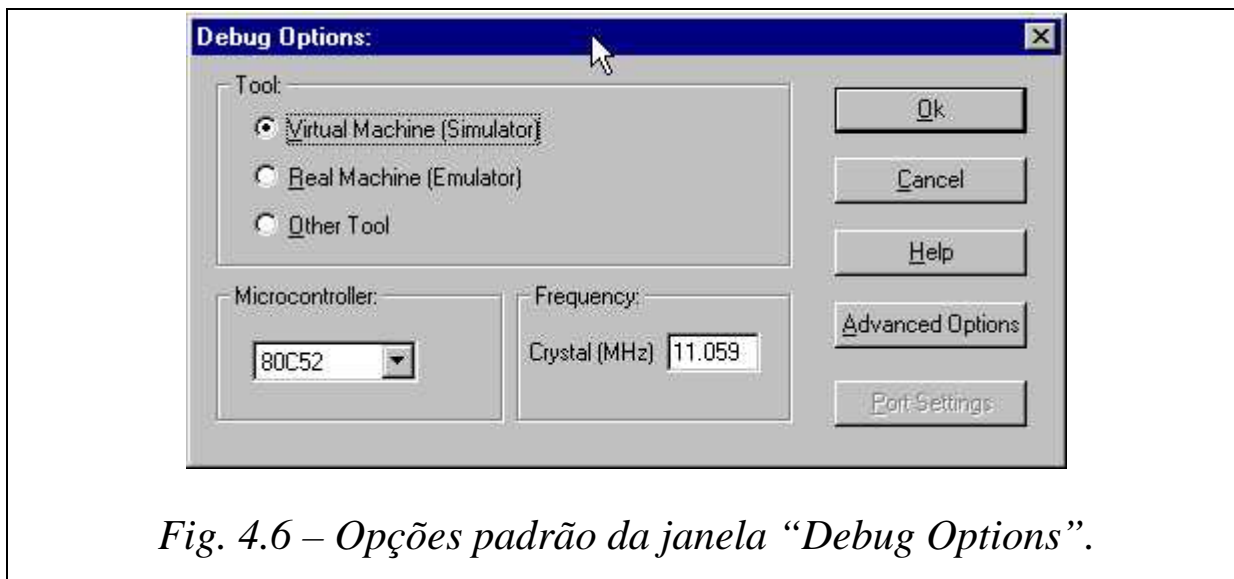


Fig. 4.5 – Mensagem apresentada pelo Proview após encerrar os processos de compilação e link-edição.

A próxima ação é rodar o depurador. Para isso, selecione "Start" no menu "Debug". Após essa ação, aparecerá a caixa de diálogo "Debug Options", onde você pode modificar as opções que são usadas para rodar o depurador/simulador WinSim. A Fig. 4.6 apresenta as opções padrão, usadas nesse exemplo.



Você agora está no ambiente de simulação. Selecione, agora, "Hardware" no menu "View" e, na sequência "UART". Após essa ação, a janela "UART" (Fig 4.7) será mostrada.

Selecione, então, "Run" no menu "Debug". A tela apresentada na Fig. 4.8 mostra que após a execução de "alo", a janela "UART" mostra o texto "Meu Primeiro Projeto".

Para encerrar a seção de depuração/simulação selecione "Terminate" no menu "Debug", retornando ao modo de edição do ProView.

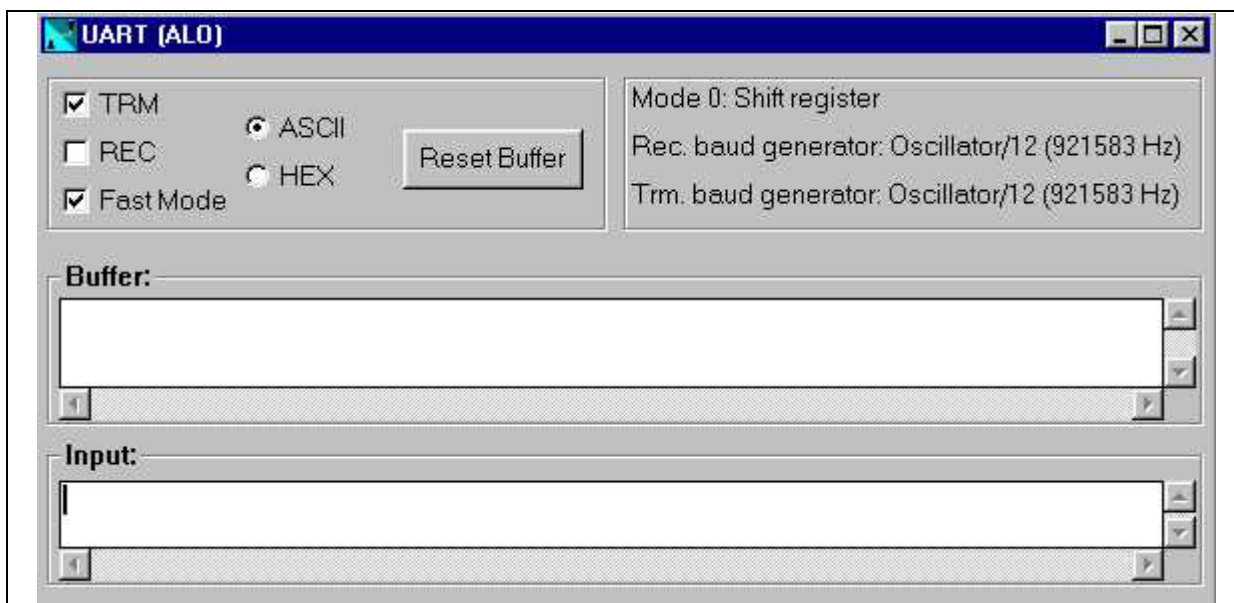


Fig. 4.7 – Janela “UART” no depurador/simulador ProView.

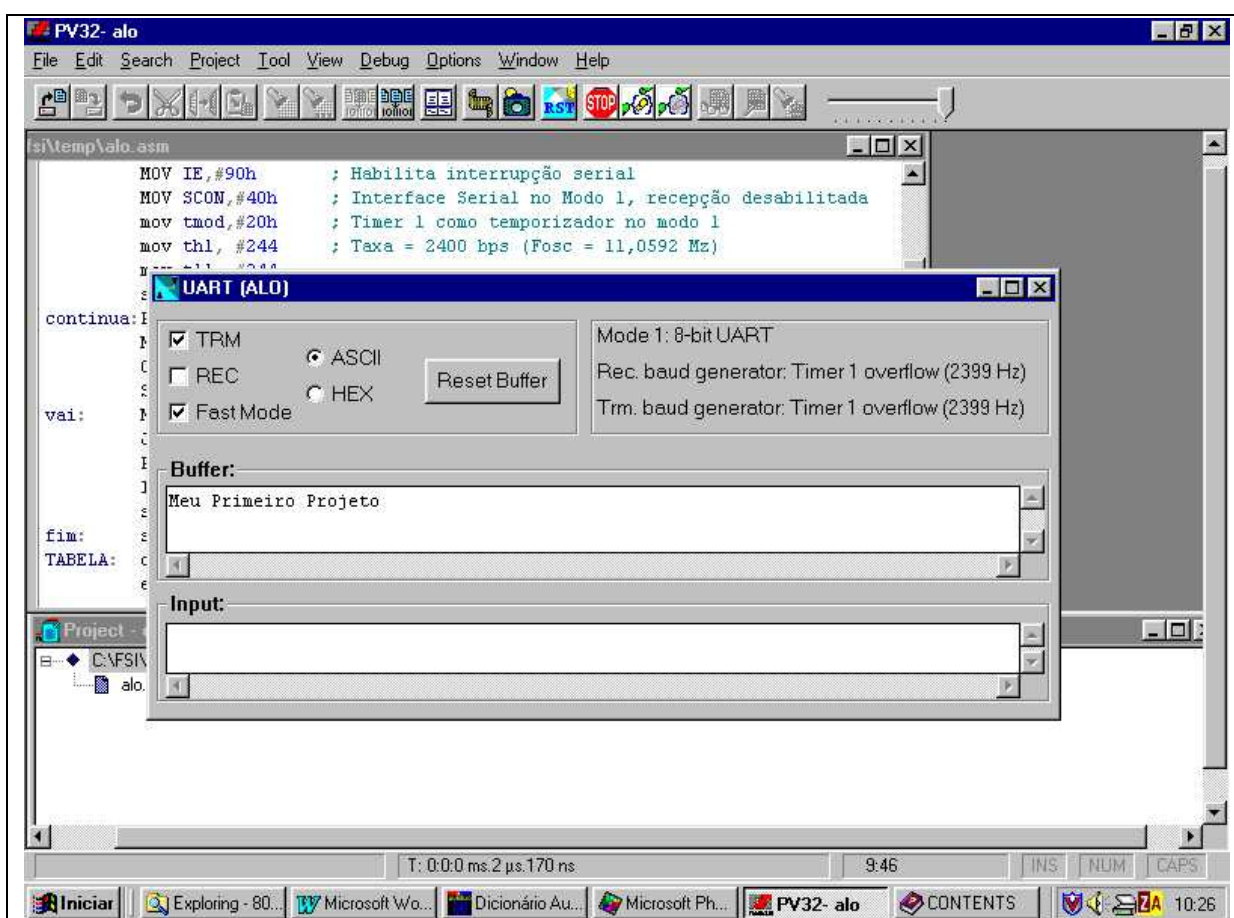


Fig. 4.8 – A janela “UART” após a execução do programa “alo”.

V – PLACA DE DESENVOLVIMENTO

5.1 – Apresentação da Placa de Desenvolvimento AT89S

A Placa de Desenvolvimento AT89S desenvolvida pelo CEFET-SC contém um microprocessador AT89S8252, uma interface de programação “*in-system*”, uma interface para *display* LCD, uma interface para *display* de 7 segmentos e uma interface serial RS – 232. As interfaces de programação e RS-232 compartilham o mesmo conector DB-9. Todos os terminais de E/S são acessíveis através de barras de terminais 180⁰, o que facilita a ligação da placa a circuitos externos como sensores, atuadores e teclados. A placa contém, ainda duas micro-chaves que podem ser utilizadas para solicitação de interrupções externas ou para entrada de dados. As Fig. 5.1 e 5.2 apresentam, respectivamente, uma fotografia e o diagrama de blocos da Placa de Desenvolvimento AT89S.

5.2 – Interface para programação

O microcontrolador é programado na própria Placa de Desenvolvimento conectando as linhas SCK (P1.7), MISO (P1.6) e MOSI (P1.5) da Interface Serial Periférica e a entrada de reset (RST) a porta paralela de um PC (lpt1 ou lpt2). A Tabela 5.1 mostra as conexões entre o PC e o microcontrolador.

Para habilitar a programação do dispositivo é necessário colocar os *jumpers* localizados próximos à micro-chave “INT0” na posição “GRAVA”, conforme mostra a Fig. 5.3.

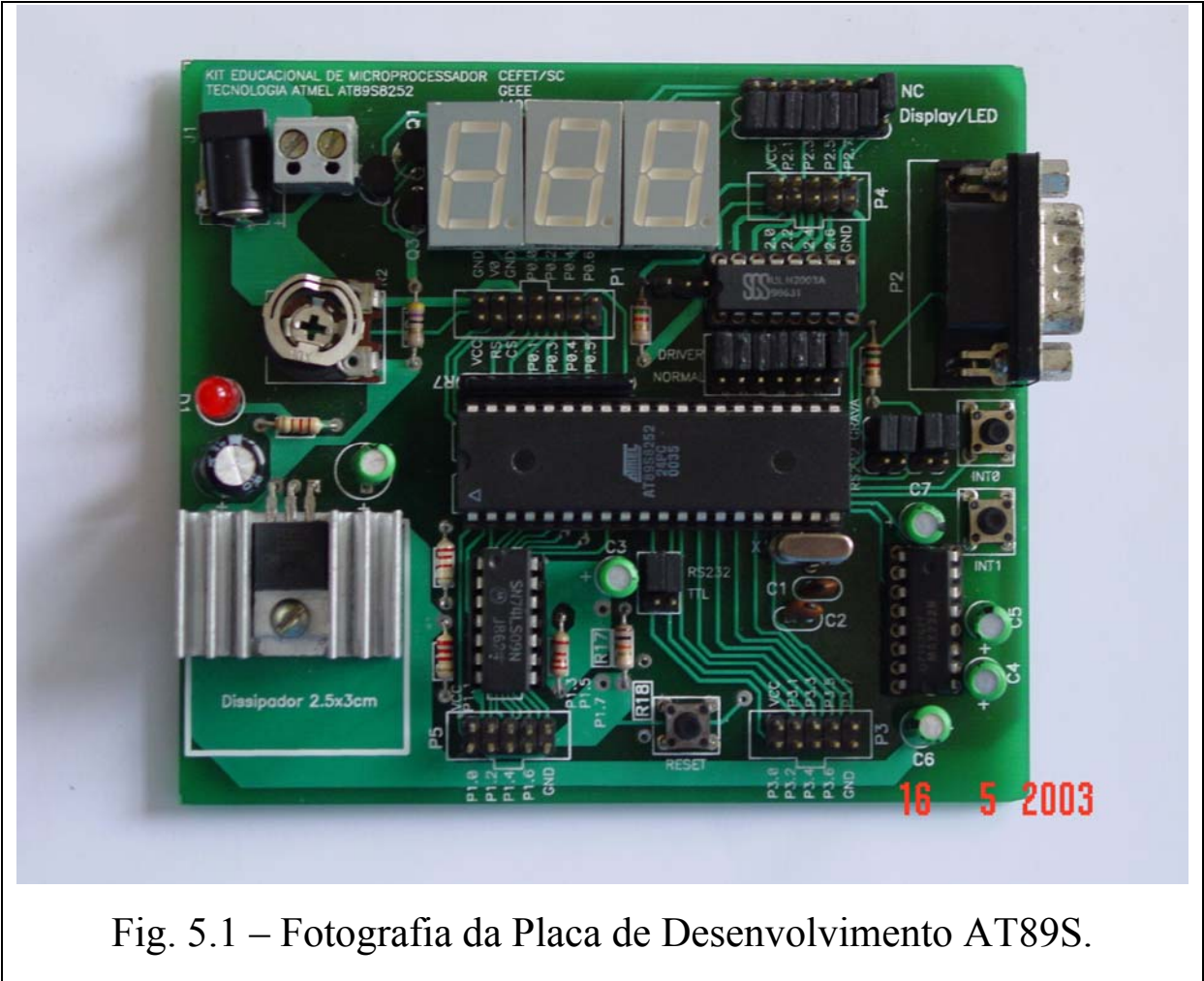
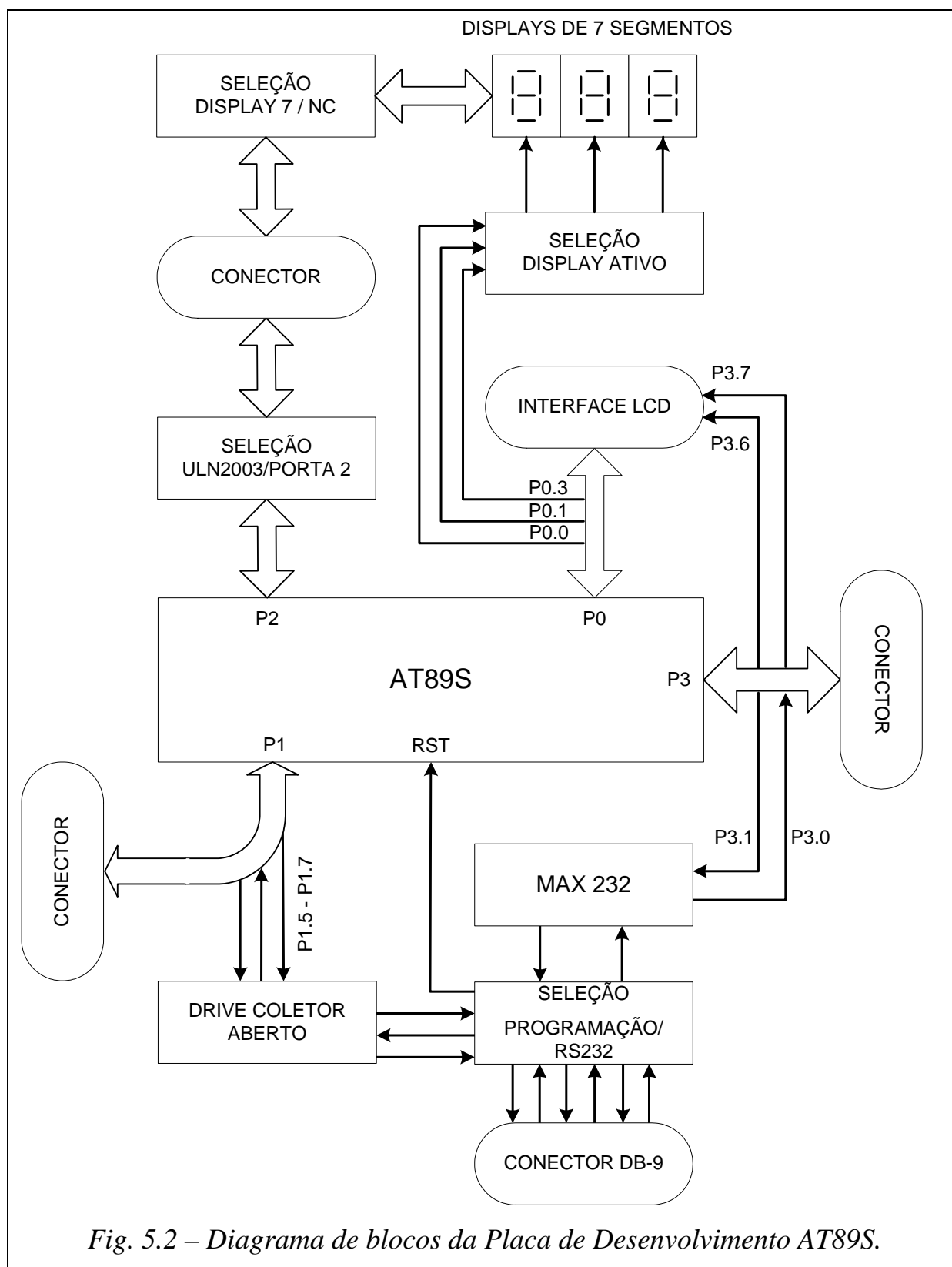


Fig. 5.1 – Fotografia da Placa de Desenvolvimento AT89S.

TABELA 5.1

CONEXÕES ENTRE O PC E O MICROCONTROLADOR

Pinos da Porta Paralela	Pinos do AT89S
6 (D4)	9 (RST)
7 (D5)	6 (MOSI - P1.5)
8 (D6)	8 (SCK - P1.7)
10 (ACK)	7 (MISO -P1.6)
25 (GND)	20 (GND)



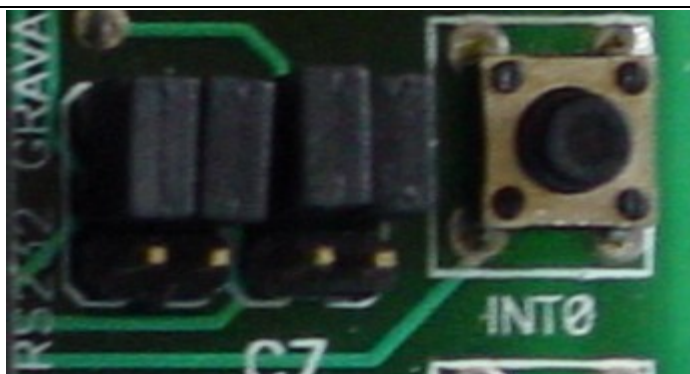


Fig. 5.3 – Habilitando a interface de programação.

A programação é realizada pelo software *freeware AEC_ISP* da AEC Eletronics⁷, executável em janela do Windows. Este software permite gravar, apagar e verificar tanto as memórias de programa Flash, como a memória de dados EEPROM do AT89S8252. O programa permite programar, ainda, os microcontroladores AT89S51, AT89S52 e o AT89S53. O menu de entrada do *AEC_ISP* é apresentado na Fig. 5.4 e o procedimento para gravação é descrito a seguir.

Selecionando (A) no menu principal, aparecerá a mensagem “*Input Filename:*” e o programa aguardará que você digite o nome de um arquivo no formato Intel HEX (inclua a extensão) armazenado no mesmo diretório do *Aec_isp.exe* e pressione [enter]. Após essa ação, o programa apresentará os dados do arquivo e a mensagem “*Hex file loaded, Press any key to continue*”. Após executar a ação solicitada, reaparece o menu principal. Para carregar o dispositivo basta selecionar (E) no menu principal. E, para executar o programa

⁷ Disponível em www.aec-electronics.co.nz.

carregado, é necessário selecionar (I) no menu principal, procedimento que levará a linha de reset para 0 lógico. A Sessão de gravação é encerrada selecionando (X) no menu principal.

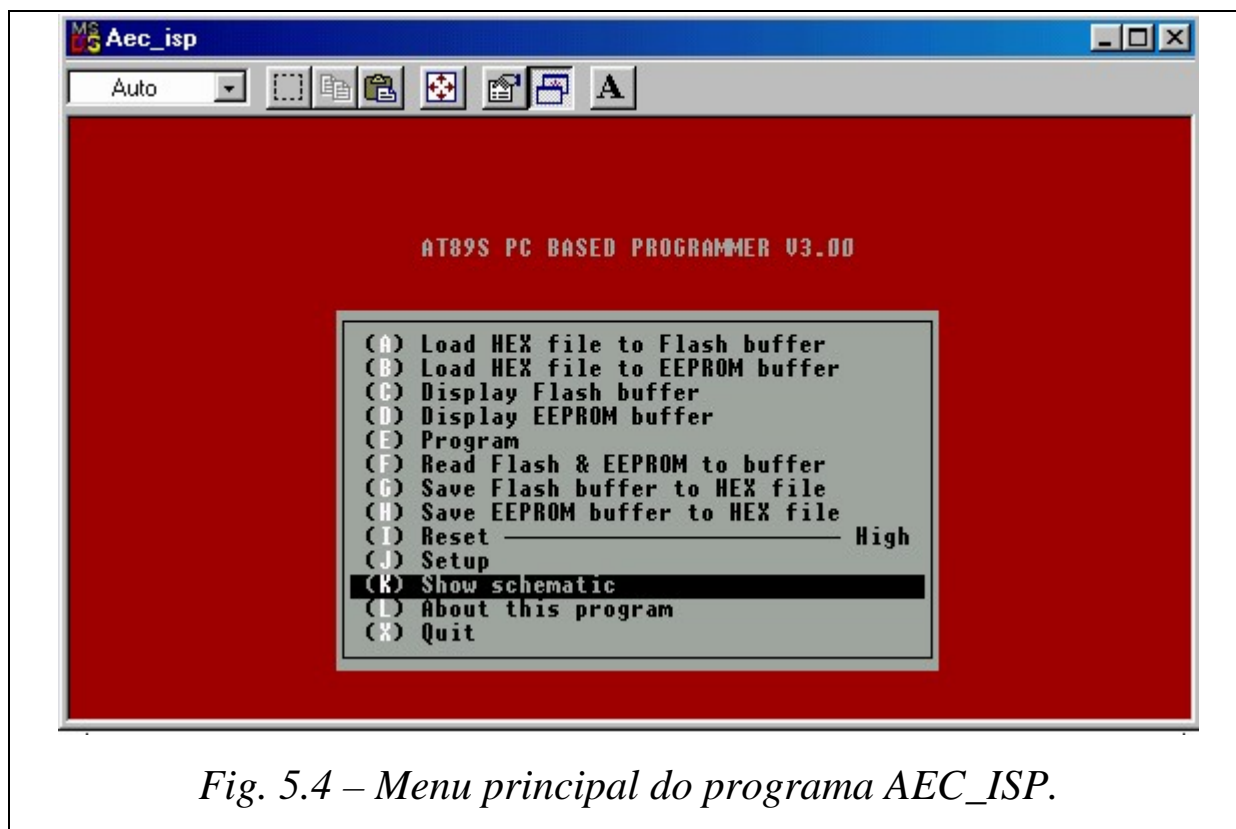


Fig. 5.4 – Menu principal do programa AEC_ISP.

A configuração (*setup*) do AEC_ISP é acessível selecionando (J) no menu principal. Acessando o *setup* você pode definir, por exemplo, se a gravação será realizada na memória Flash ou na EEPROM. O menu “Setup” é apresentado na Fig. 5.5.

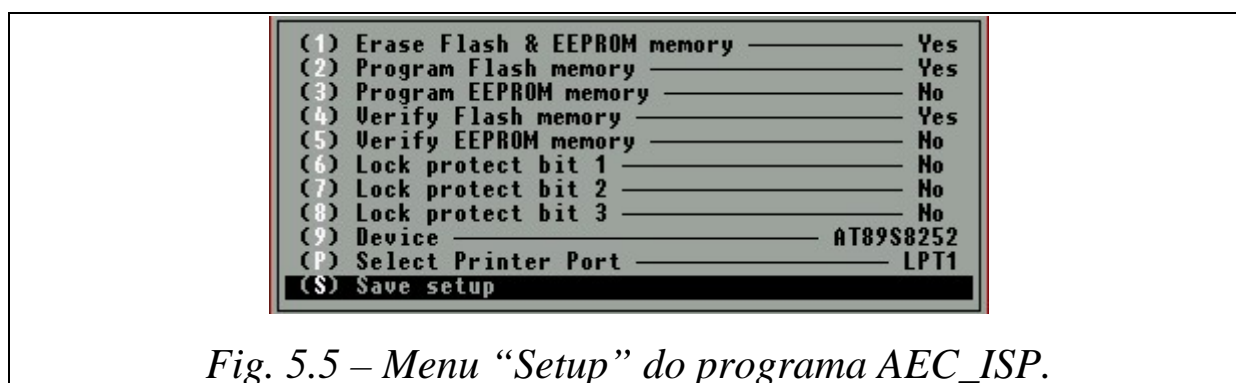
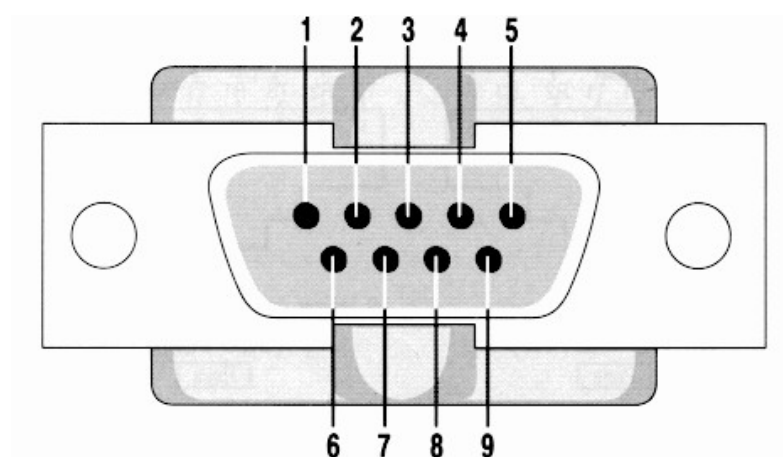


Fig. 5.5 – Menu “Setup” do programa AEC_ISP.

Em cada operação de programação, é recomendado apagar o dispositivo (1), programá-lo (2 e/ou 3) e verificar a correção dos dados enviados (4 e/ou 5).

5.3 – Interface Serial RS-232

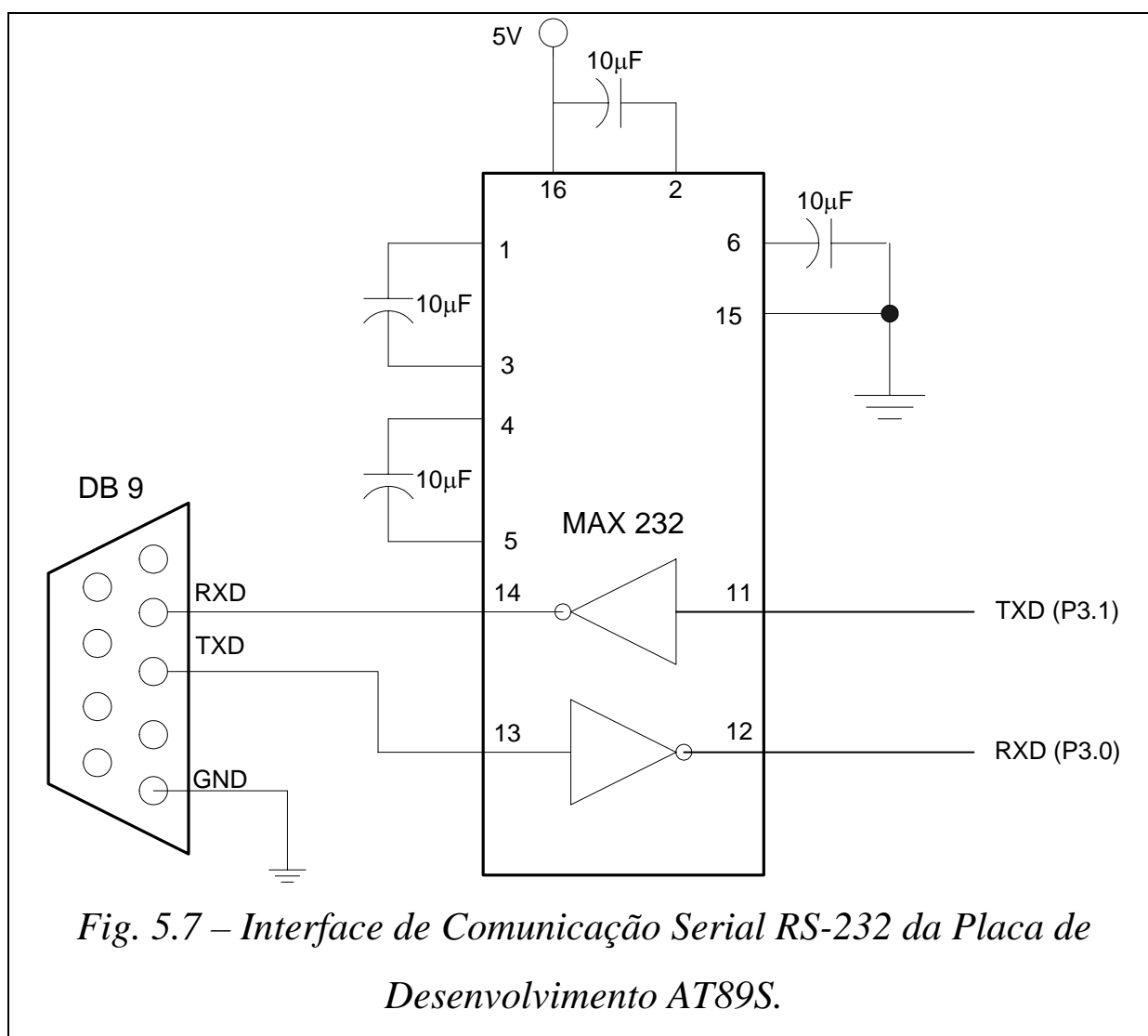
A interface serial padrão RS-232 para PC (Fig. 5.6) requer lógica negativa, isto é, -3V a -12V indicam 1 lógico e +3V a 12 V indicam 0 lógico.



Pino	Sinal	Pino	Sinal
1	Data Carrier Detect (DCD) (Detector de Portadora)	6	Data Set Ready (DSR) (Dados Prontos)
2	Receive Data (RXD) Dados Recebidos	7	Request to Send (RTS) (Requisição para Envio)
3	Transmitted Data (TXD) (Dados Transmitidos)	8	Clear to Send (CTS) (Permissão para Envio)
4	Data Terminal Ready (DTR) (Terminal de Dados Pronto)	9	Ring Indicator (Indicador de Linha)
5	GND		

Fig. 5.6 – Porta de Comunicação (COM) do PC – Terminais de Saída RS-232 em um conector DB-9.

Para executar a adaptação de amplitude dos sinais entre a lógica TTL dos terminais RXD e TXD do microcontrolador e a interface RS-232 do PC, a Placa de Desenvolvimento utiliza o circuito integrado MAX232. A conexão entre o microcontrolador e o conector DB-9 da placa, intermediada pelo MAX232, é mostrada na Fig. 5.7.



Entretanto, como não se utiliza *handshaking*⁸, para poder utilizar programas de terminal no PC que esperam por um sinal indicador de

⁸ No *Handshaking*, quando o PC está pronto a aceitar informações do microcontrolador (periférico), ele envia para este um sinal que inicia a comunicação.

que a placa está pronta para receber dados, as linhas RTS (*Ready to send*) e CTS (*Clear to send*) devem ser interligadas no cabo utilizado na comunicação serial.

Para utilizar a Interface Serial RS-232 da Placa de Desenvolvimento é necessário colocar os *jumpers* localizados próximos à micro-chave “INT0” (Fig. 5.3) e à micro-chave “RESET” (Fig. 5.8) na posição “RS232”. Por outro lado, para utilizar os pinos P3.0 e P3.1 como linhas de E/S é necessário colocar os *jumpers* mostrados na Fig. 5.8 na posição “TTL”.



Fig. 5.8 – Utilizando os pinos P3.0 e P3.1 como terminais de E/S.

Para estabelecer a comunicação entre um computador PC e a Placa de Desenvolvimento é necessário a utilização de um software

emulador de terminal como, por exemplo, o “terminal.exe” e o “term232.exe” para, ambos para *windows* e *freewares*. Aqui, discutiremos o uso do primeiro.

Para configurar no *Terminal* os parâmetros da comunicação serial, selecione a opção “Settings” do menu “Settings”. Após essa ação, aparecerá uma caixa de diálogo (Fig. 5.9), onde você pode alterar os seguintes parâmetros do canal serial: porta de comunicação (COM1, COM2, COM3 e COM4), taxa de transmissão (300 a 115.200 bps), bits de dados (5 a 8), paridade e bits de parada (1 ou 2). Para o *Terminal* receber a mensagem “Meu primeiro projeto” (Fig. 5.10) enviada pela Placa de Desenvolvimento, após a execução do programa “alo” (Fig. 4.4), é preciso alterar a configuração corrente, conforme ilustra a Fig. 5.9.

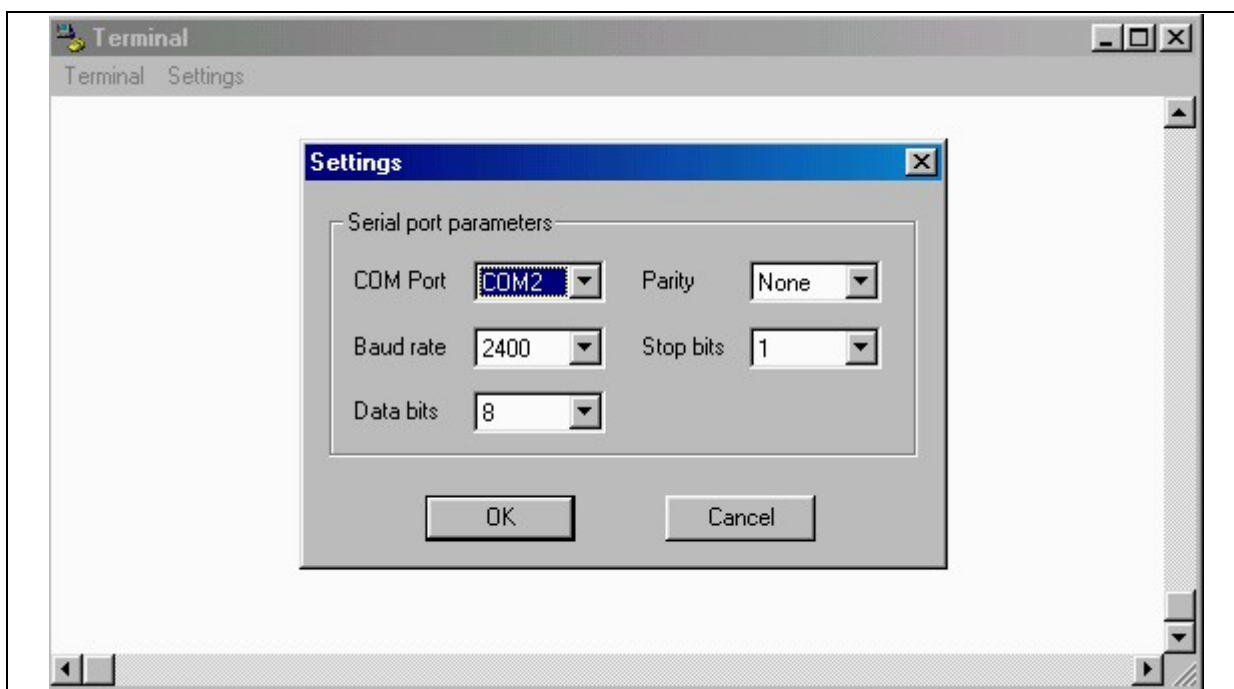


Fig. 5.9 – Configurando os parâmetros da comunicação serial no *Terminal*.

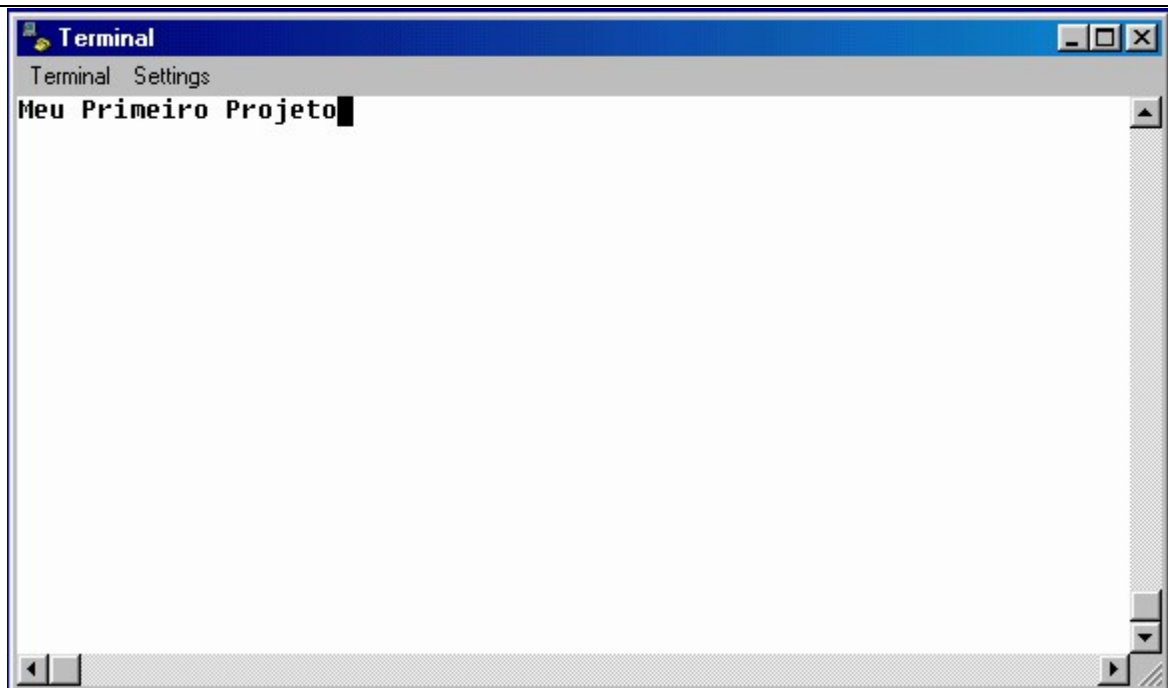


Fig. 5.10 – A janela do programa Terminal após a recepção da mensagem “Meu primeiro projeto”.

5.4 – Interface para Display LCD

Nesta seção, descreveremos como estabelecer a comunicação entre um microcontrolador MCS-51™ e um módulo LCD a caractere baseado no controlador Hitachi HD44780, o tipo mais comum de controlador LCD.

Os Módulos ou Displays LCD (*Liquid Cristal Displays*) são especificados em número de colunas por linhas, estando disponíveis em modelos de 8 colunas e 1 linha até modelos de 40 colunas e 4 linhas, conforme ilustra a Tabela 5.2.

TABELA 5.2
ALGUMAS CONFIGURAÇÕES DE MÓDULOS LCD

Número de Colunas	Número de Linhas	Quantidade de Pinos
8	1	14
8	2	14
16	1	14/16 ⁹
16	2	14/16
16	4	14/16
20	1	14/16
20	2	14/16
20	4	14/16
24	2	14/16
24	4	14/16
40	2	16
40	4	16

O conector mais comum usado em Módulos LCD baseados no 44780 apresenta 14 terminais em linha, conforme ilustra a Fig. 5.11. A Tabela 5.3 apresenta a descrição desses 14 terminais.

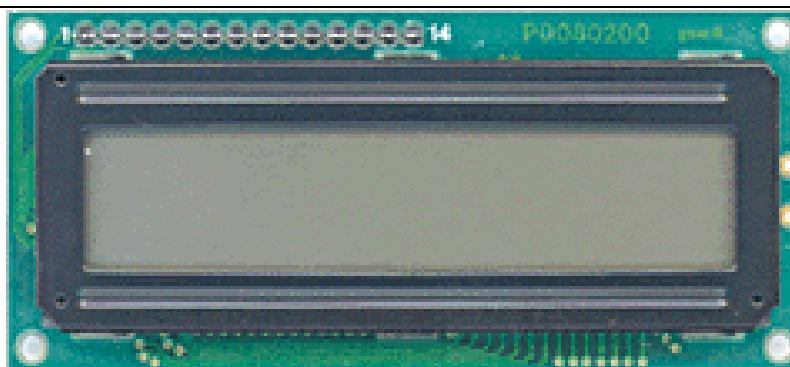


Fig. 5.11 – Display LCD 16 x 2 linhas.

⁹ Os módulos de 16 pinos possuem iluminação de fundo (backlight).

TABELA 5.3
PINAGEM DOS MÓDULOS LCD

Pino	Símbolo	Função
1	V_{SS}	GND
2	V_{DD}	+ 5 V
3	V_0	Tensão de ajuste do contraste
4	RS	1 – Entrada de Dados 0 – Entrada de Instruções
5	R/W	1 – Leitura 0 – Escrita
6	E	Habilita a operação do dispositivo.
7 - 10	DB0 – DB3	Linhas de mais baixa ordem do Barramento de Dados com “three state” e bidirecionais.
11 - 14	DB4 – DB7	Linhas de mais alta ordem do Barramento de Dados com “three state” e bidirecionais. Estas linhas não são usadas quando o interfaceamento é de 4 bits.
15	A	Anodo do LED de Luz de Fundo
16	K	Catodo do LED de Luz de Fundo

Os códigos dos caracteres recebidos pela unidade LCD do microcontrolador são armazenados em uma RAM chamada DDRAM (*Data Display RAM*), transformados em um caractere no padrão matriz de pontos e apresentados em uma tela LCD. Para produzir os caracteres no padrão matriz de pontos, o módulo LCD incorpora uma CGROM (*Character Generator ROM*). Os Displays LCD também possuem uma CGRAM (*Character Generator RAM*) para a gravação de caracteres especiais.

O Conjunto de caracteres presentes no 44780 (Fig. 5.12) é basicamente o ASCII; basicamente porque alguns caracteres não seguem a norma ASCII. Por exemplo, o caractere ASCII de código

“5Ch” é o “\”, enquanto o 44780 disponibiliza em “5Ch” um “¥”. Oito caracteres programáveis estão disponíveis na CGRAM e utilizam os códigos 00h a 07h. Os dados na CGRAM são representados em um mapa de bits de 8 bytes, conforme ilustra a Fig. 5.13. Como há espaço para 8 caracteres, estão disponíveis 64 bytes nos seguintes endereços base: 40h, 48h, 50h, 58h, 60h, 68h, 70h e 78h.

O Endereço da DDRAM, colocado no contador de endereços, é expresso em números hexadecimais, conforme ilustra a Fig. 5.14.

		HIGH		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
LOW		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111				
x0	xxxx 0000	CG RAM (1)				0	0	B	P	¿	P					—	9	E	o	p	
x1	xxxx 0001	CG RAM (2)				!	1	A	Q	a	a					7	7	4	3	a	
x2	xxxx 0010	CG RAM (3)				"	2	B	R	b	r					「	イ	ツ	×	Þ	Þ
x3	xxxx 0011	CG RAM (4)				#	3	C	S	c	s					」	ウ	テ	モ	ε	ε
x4	xxxx 0100	CG RAM (5)				\$	4	D	T	d	t					√	い	ト	ワ	Ω	Ω
x5	xxxx 0101	CG RAM (6)				%	5	E	U	e	u					・	オ	タ	1	8	0
x6	xxxx 0110	CG RAM (7)				&	6	F	V	f	v					ヲ	カ	ニ	ヨ	Σ	Σ
x7	xxxx 0111	CG RAM (8)				'	7	G	W	w						ア	キ	ヲ	ヲ	q	π
x8	xxxx 1000	(1)				(8	H	X	h	x					イ	ク	ズ	U	フ	又
x9	xxxx 1001	(2))	9	I	Y	i	y					お	ク	ル	ル	ウ	ウ
xA	xxxx 1010	(3)				*	:	J	Z	j	z					エ	コ	ン	ク	i	キ
xB	xxxx 1011	(4)				+	:	K	L	k	l					ア	サ	ヒ	ロ	×	ア
xC	xxxx 1100	(5)				,	<	L	¥	l						ト	ヨ	フ	フ	Φ	ア
xD	xxxx 1101	(6)				—	=	M	I	m						ユ	ズ	△	△	÷	÷
xE	xxxx 1110	(7)				.	>	N	^	n	^					ヨ	エ	ト	△	△	△
xF	xxxx 1111	(8)				/	?	O	_	o	+					ウ	ヨ	ヲ	ア	0	■

Fig. 5.12 – O conjunto de caracteres do 44780 e seus códigos.

Endereço da CGRAM	Mapa de Bits					Dado
50h						0Eh
51h						11h
52h						10h
53h						10h
54h						15h
55h						0Eh
56h						10h
57h						00h

Fig. 5.13 – Gravação do ç na CGRAM, matriz 5 x 7. Esse caractere será selecionado pelo código 02h.

A Fig. 5.15 apresenta o endereço da DDRAM e a posição correspondente no visor do LCD para um display com 40 caracteres por linha configurado no modo de duas linhas. Note que os endereços da segunda linha não são contíguos ao da primeira.

← Bits de mais alta ordem				Bits de mais baixa ordem →			
AC	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Dígito Hexadecimal				Dígito Hexadecimal			

Fig. 5.14 – O endereçamento da DDRAM.

Dígito	1	2	3	4	5	6	7	8		39	40
Linha 1	00h	01h	02h	03h	04h	05h	06h	07h	...	26h	27h
Linha 2	40h	41h	42h	43h	44h	45h	46h	47h	...	66h	67h

Fig. 5.15– Endereçamento do visor, modo de duas linhas.

Para um módulo LCD com uma capacidade do visor menor que 40 caracteres por linha, um número de caracteres iguais a sua capacidade são mostrados a partir da posição 1 para cada linha. A Fig. 5.16 ilustra o endereçamento do visor para um display 16 x 2.

Dígito	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L. 1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
L. 2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Fig. 5.16 – Endereçamento do visor em hexadecimal para um display 16 x 2 no modo de duas linhas.

A Fig. 5.17 apresenta o endereço da DDRAM e a posição correspondente no visor do LCD para um display de uma linha com endereçamento lógico de duas linhas.

Dígito	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L. 1	00	01	02	03	04	05	06	07	40	41	42	43	44	45	46	47

Fig. 5.17 – Endereçamento do visor em hexadecimal para um display 16 x 1 com endereçamento lógico de duas linhas.

Um módulo LCD tem dois registradores de 8 bits – um registrador de instrução (IR) e um registrador de dados (DR). O registrador de instrução armazena códigos de instrução tais como “limpa display” ou “desloca cursor” e armazena informação de

endereçamento da DDRAM e da CGRAM. O conjunto de instruções do controlador 44780 é mostrado na Tabela 5.4.

O registrador de dados é utilizado para armazenamento temporário de dados durante as transações com o microcontrolador. Por exemplo, quando dados são escritos na unidade LCD, o dado é inicialmente armazenado no registrador de dados e é, então, automaticamente escrito na DDRAM ou na CGRAM, de acordo com a operação em curso.

Conforme podemos inferir a partir da Tabela 5.3, um Display LCD requer três linhas de controle, as quais descrevemos a seguir:

- A linha *Enable* (E) permite acessar o *display* a partir das linhas “R/W” e “RS”. Quando “E” está em nível lógico 0, o *display* está desabilitado e ignora os sinais vindos de “R/W e RS”; quando “E” está em 1 lógico, o Display LCD checa o estado das outras linhas de controle e age de acordo com os sinais destas linhas.
- A linha *Read/Write* (R/W) determina a direção dos dados entre o display e o microcontrolador. Quando em 1 lógico, os dados são lidos a partir do *display* LCD; quando em 0 lógico, os dados são escritos no *display* LCD.

TABELA 5.4
CONJUNTO DE INSTRUÇÕES DO CONTROLADOR HD44780

Instrução	Código										Função	Tempo de execução (max)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Limpa Display	0	0	0	0	0	0	0	0	0	1	Limpa o display e retorna o cursor à 1ª posição da 1ª linha.	1,64 ms
Retorno	0	0	0	0	0	0	0	0	1	φ	Retorna a mensagem e o cursor à 1ª posição da 1ª linha.	1,64 ms
Fixa o Modo de Operação	0	0	0	0	0	0	0	1	I/D	S	Especifica o sentido de deslocamento do cursor e o modo de deslocamento da mensagem	40 μs
Display Ativo/Inativo	0	0	0	0	0	0	1	D	C	B	Especifica a ativação do display (D), do cursor (C) e a intermitência do caractere na posição do cursor (B)	40 μs
Deslocamento do Cursor / Mensagem	0	0	0	0	0	1	S/C	R/L	φ	φ	Desloca a mensagem e move o cursor	40 μs
Fixa o Modo de Utilização	0	0	0	0	1	DL	N	F	φ	φ	Fixa o número de bits da interface (DL), de linhas (L) e a matriz de pontos	40 μs
Fixa o endereço da CGRAM	0	0	0	1	A _{CG}						Carrega o contador de endereços com o endereço da CGRAM. O dado subsequente é o da CGRAM.	40 μs
Fixa o endereço da DDRAM	0	0	1	A _{DD}							Carrega o contador de endereços com o endereço da DDRAM. O dado subsequente é o da DDRAM.	40 μs

TABELA 5.4
CONJUNTO DE INSTRUÇÕES DO CONTROLADOR HD44780

Instrução	Código										Função	Tempo de execução (max)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Leitura do Contador de Endereços e do <i>Busy Flag</i>	0	1	AC								Lê o conteúdo do contador de endereços e o <i>Busy Flag</i> .	0
Escrita de dados na CG ou DDRAM	1	0	Dados a escrever								Escreve dados na CG ou DDRAM.	40 μ s
Leitura de dados na CG ou DDRAM	1	1	Dados lidos								Lê dados na CG ou DDRAM.	40 μ s

I/D = 1: incrementa; I/D = 0: decrementa

S = 1: deslocamento automático da mensagem

D = 1: display ativo; D = 0: display inativo

C = 1: cursor ativo; C = 0: cursor inativo

B = 1: cursor intermitente

S/C = 1: deslocamento da mensagem; S/C = 0: deslocamento do cursor.

R/L = 1: deslocamento à direita; R/L = 0: deslocamento à esquerda.

DL = 1: interface de 8 bits; DL = 0: interface de 4 bits

N = 1: duas linhas; N = 0: linha única

F = 1: 5 x 10 pontos; F = 0: 5 x 7 pontos

BF = 1: operação interna; BF = 0: pronto para instrução

A_{CG}: endereço da CGRAM

A_{DD}: endereço da DDRAM ;

AC: contador de endereços

- A linha *Register Select* (RS) permite ao *display* LCD interpretar o tipo de dado presente na via de dados. Quando em 1 lógico, um caractere está sendo escrito no Display LCD; quando em 0 lógico, uma instrução está sendo escrita.

O diagrama de tempo da Fig. 5.18 apresenta uma operação de escrita de um caractere na tela do LCD, sendo que os valores máximos e mínimos de cada parâmetro integrante do diagrama é apresentado na Tabela 5.5.

A unidade LCD pode operar com uma transferência de dados dual de 4 bits ou com uma única de 8 bits. Se o modo de 4 bits é usado, o *nibble* mais significativo é enviado primeiro, sendo necessário um pulso “E” de largura mínima de 450ns para cada *nibble*.

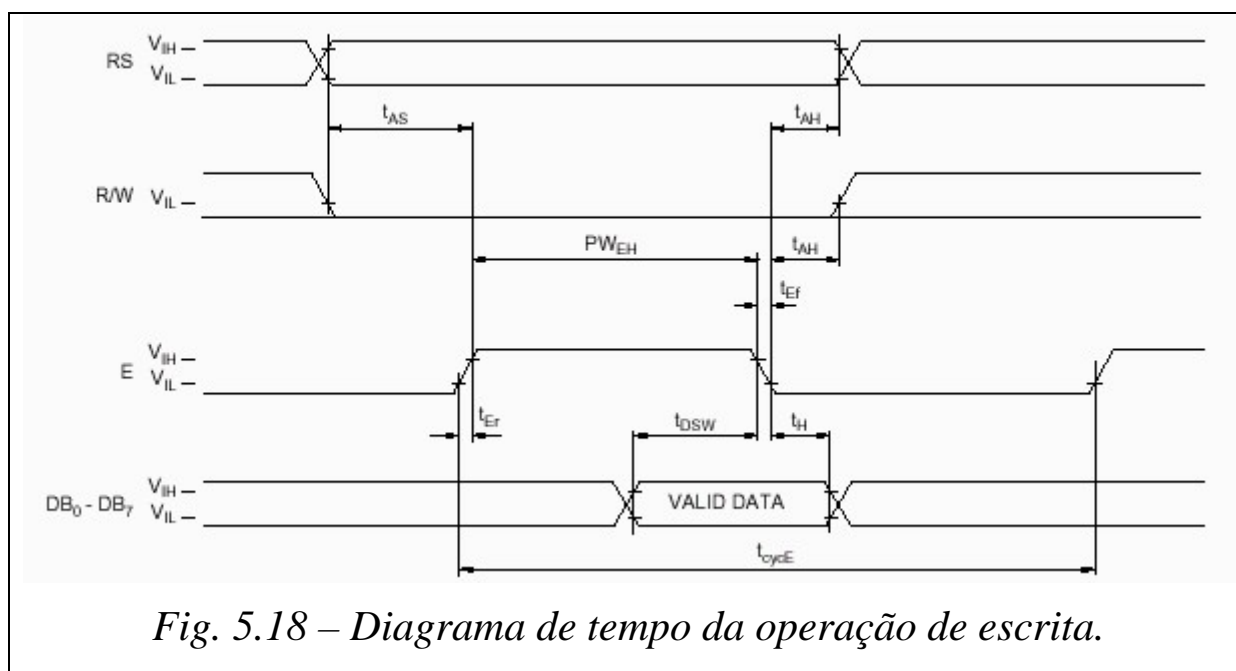


TABELA 5.5
PARÂMETROS CARACTERÍSTICOS DA OPERAÇÃO DE
ESCRITA ($V_{DD} = 5 \pm 5 \%$, $V_{SS} = 0$ e $T_A = 0 \sim 50^{\circ}\text{C}$)

PARÂMETRO	SÍMBOLO	VALOR		UNIDADE
		MÍN.	MÁX.	
Tempo de ciclo de habilitação	t_{cycE}	1000	-	ns
Largura do pulso <i>enable</i> (nível 1)	PW_{EH}	450	-	ns
Tempo de queda/subida do pulso <i>enable</i>	$t_{\text{Er}}, t_{\text{Ef}}$	-	25	ns
Tempo de configuração	t_{AS}	140	-	ns
Tempo de retenção do endereçamento	t_{AH}	10	-	ns
Tempo de configuração dos dados	t_{DSW}	195	-	ns
Tempo de retenção dos dados	t_{H}	10	-	ns

Para ler ou escrever em um *display* LCD é necessário seguir a seguinte sequência:

1. Levar a linha “R/W” para 0 lógico se a operação for de escrita e para 1 lógico se for de leitura;
2. Levar a linha “RS” para nível lógico 0 ou 1 (instrução ou caractere);
3. Transferir os dados para a via de dados;
4. Levar a linha “E” para 1 lógico;
5. Levar a linha “E” para 0 lógico;
6. Intercalar uma rotina de atraso entre as instruções ou fazer a leitura do *Busy Flag* (o bit B7 da linha de dados)

antes do envio da instrução e enviá-la somente quando ele for 0 lógico.

Toda vez que a alimentação do módulo é ligada, o *display* LCD executa uma rotina de inicialização. Durante a execução dessa rotina o *Busy Flag* está em 1 lógico. Este estado “ocupado” dura por 10ms após a tensão de alimentação atingir 4,5 V. As instruções executadas na inicialização da unidade LCD são:

- Limpa display (01h);
- Fixa o modo de utilização (20h), com DL = 1 (interface de 8 bits, N = 0 (linha única) e F = 0 (matriz 5 x 7) ;
- Controle ativo/inativo do display (08h), com D = 0 (mensagem não aparente), C = 0 (cursor inativo) e B = 0 (função intermitente inativa);
- Fixa o modo de operação (06h), com I/D = 1 (modo incrementa) e S = 0 (deslocamento do display inativo).

Caso as condições de energização apresentadas na Tabela 5.6 não sejam satisfeitas, o circuito interno de *reset* não operará adequadamente e o módulo não será inicializado. Nesse caso, o procedimento de inicialização deve ser executado pelo microcontrolador externo. SHARP (1999) apresenta o procedimento de inicialização por instruções.

TABELA 5.6
CONDIÇÕES DE ENERGIZAÇÃO PARA O *RESET* INTERNO

PARÂMETRO	SÍMBOLO	VALOR			UNIDADE
		MÍN.	TIP.	MÁX.	
Tempo de crescimento da tensão	t_{rcc}	0,1	-	10	ms
Período de desenergização	t_{OFF}	1	-	-	ms

A Placa de Desenvolvimento possui uma interface para ligação de módulos LCD tipo caractere, conforme ilustra a Fig. 5.19. Fazem parte desta interface o conector P1 e um trimpot de 10 k Ω para ajuste do contraste. O esquema de ligação da Interface de Display LCD da Placa de Desenvolvimento é apresentado na Fig. 5.20.

Como para a maioria das aplicações, não há nenhuma razão para ler dados do LCD, a linha “R/W” da interface da Fig. 5.8 é aterrada. Essa ação faz necessário intercalar uma rotina de atraso que respeite o tempo necessário para executar cada instrução, tempos que constam na Tabela 5.4.

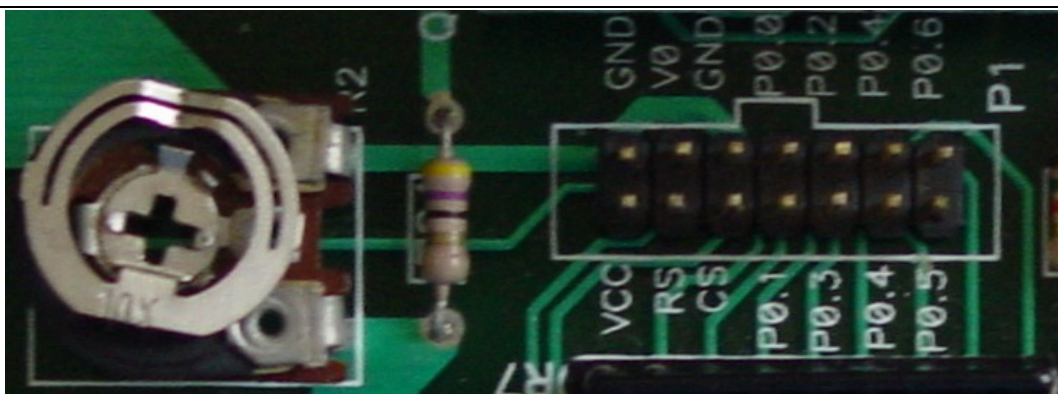
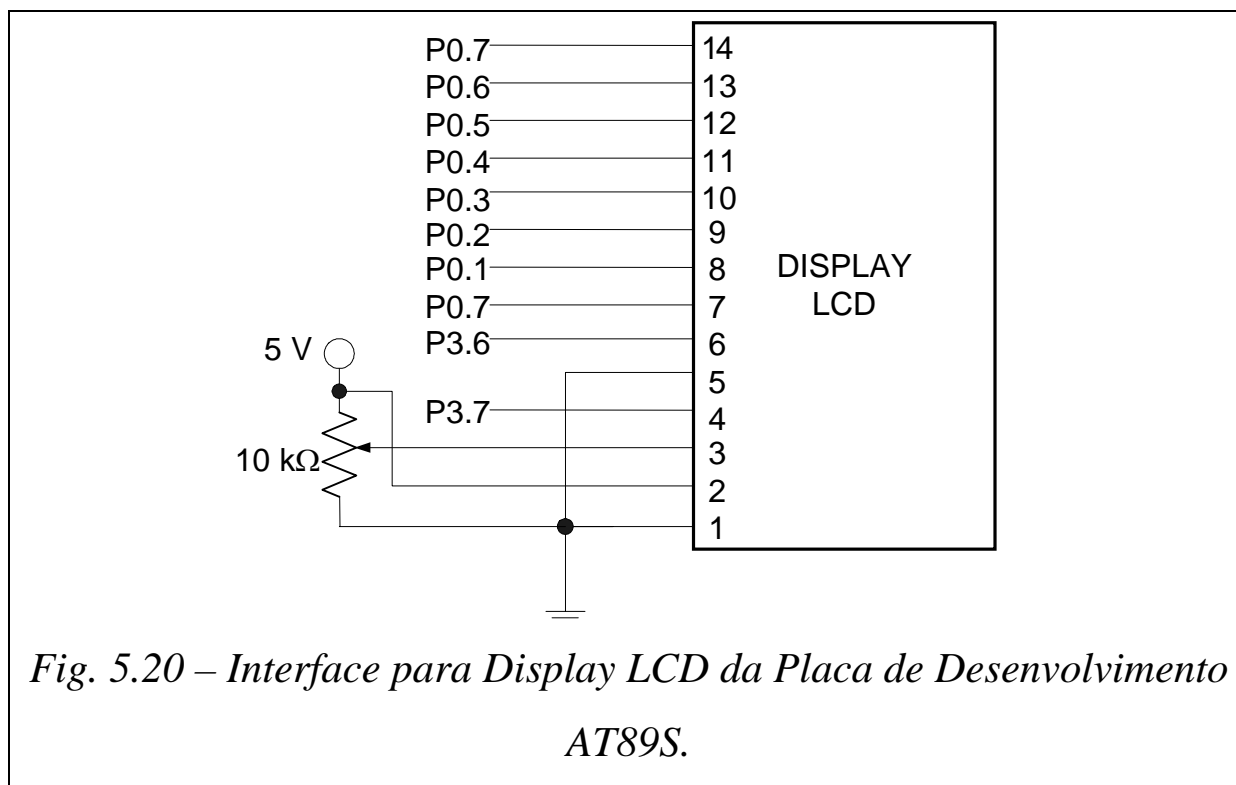


Fig. 5.19 – Fotografia da Interface para Display LCD



Na listagem que segue, apresentamos o programa “LCD_DEMO”, que escreve em um módulo LCD 16 x 2 conectado a Placa de Desenvolvimento AT89S a mensagem:

*“ DEMONSTRAÇÃO:
DISPLAY LCD.”*

Para escrever essa mensagem o programa inclui uma rotina que grava nos endereços base 50h e 58h da CGRAM, respectivamente, os caracteres “ç” e “Ã”. Chamamos a sua atenção que, na sua auto-inicialização, o display é configurado em linha única, com mensagem e cursor inativos. Por isso, é necessário incluir dois comandos para alterar essas configurações.

LISTAGEM DO PROGRAMA LCD_DEMO

```
#include <REG8252.H>
;
sjmp main
;
;*****
; ESCRITA DE INSTRUÇÃO
;
COMAND: CLR P3.7           ;identifica instrução (RS=0)
        SETB P3.6         ;pulso p/ enable = 1,085 us
        CLR P3.6
        MOV R0,#00h       ;tempo para executar
                           ;instrução = 2,2 ms
        MOV R1,#04h
        DJNZ R0,$
        DJNZ R1,$-2
        RET               ;retorno
;
;*****
; ESCRITA DE DADO
;
DADO:   SETB P3.7          ;identifica dado (RS=1)
        SETB P3.6         ;pulso p/ enable = 1,085 us
        CLR P3.6
        MOV R0,#00h       ;tempo para executar dado =
                           ;555 us
        DJNZ R0,$
        RET
;
;*****
; Inicializacao do display LCD
;
ini_lcd: MOV P0,#38H       ;modo de 8 bits / 2 linhas
        CALL COMAND
        MOV P0,#0EH       ;mensagem aparente - cursor
                           ;ligado/não intermitente
        CALL COMAND
        MOV P0,#01H
        CALL COMAND
        RET
;
```

LISTAGEM DO PROGRAMA LCD_DEMO

```
;*****
; CRIAÇÃO DE CARACTERES ESPECIAIS
;
cria_carac: INC R7
            MOV A,R7
            MOV DPTR,#CARAC
            MOVC A,@A+DPTR
            CJNE A,#0FFH,CONT1
            RET
CONT1:     MOV P0,A
            CALL DADO
            SJMP cria_carac
;
;*****
; MATRIZ DE PONTOS P/ CARACTERES ESPECIAIS
;
CARAC: DB 0EH, 11H, 10H, 10H, 15H, 0EH, 10H, 00H ;ç
DB 09h, 16h, 04h, 0Ah, 11h, 1Fh, 11h, 00h, 0FFh ;Ã
;
;*****
;ESCRITA DE LINHA
;
esc_linha: MOV A,#0
            MOVC A,@A+DPTR
            CJNE A,#0FFh,escreve
            RET
escreve:   CJNE A,#0FDH,SALTO ;Se A = 0fdh, mostra
                                caractere especial
            MOV P0,R6          ; move caracter especial
                                para lcd
            CALL DADO
            INC R6              ; código p/ o proximo
                                caracter especial
            SJMP SALTO2
SALTO:    MOV P0,A              ;move caracter para P0 (lcd)
            CALL DADO
SALTO2:   INC DPTR
            SJMP esc_linha
;
;
```


LISTAGEM DO PROGRAMA LCD_DEMO

```
;*****
; Programa Principal
;
; Inicializações
;
main:    MOV R6,#02H ;código para o primeiro caractere
          especial ("ç")
          MOV R7,#0FFH ;registrador para construção dos
          caracteres especiais
          CALL ini_lcd      ; inicializações do LCD
          MOV P0,#50H       ;endereço base para o
          primeiro caractere especial

          CALL COMAND
          CALL cria_carac
          MOV DPTR,#FRASE1
          MOV P0,#80h       ; endereça primeira linha
          CALL COMAND
          CALL esc_linha
          MOV DPTR,#FRASE2
          MOV P0,#0C0h      ; endereça segunda linha
          CALL COMAND
          CALL esc_linha

;
; Finalização
;
          MOV P0,#0Ch       ; desliga cursor
          CALL COMAND
          SJMP $            ; parada

;
;*****
; PRIMEIRA LINHA
;
frasel:  db " DEMONSTRA",0FDH,0FDH,"O:",0FFH
;*****
; SEGUNDA LINHA
;
frase2:  db " MODULO LCD.",0FFH
;
END
```

5.5 – Interface para Display de Sete Segmentos

A Placa de Desenvolvimento possui uma interface para ligação de três displays de 7 segmentos tipo anodo comum, conforme ilustra o esquema da 5.21. Observe na figura, que o microcontrolador controla os três displays empregando um esquema “multiplexação”. Nesse esquema, os segmentos equivalentes dos displays são conectados em paralelo e cada display é mantido ligado pelo transistor ligado entre a alimentação e o seu anodo durante 1/3 do período de multiplexação. Como o AT89S não tem capacidade de corrente para acionar os *displays* de sete segmentos, utiliza-se um CI ULN2003 para interfacear o microcontrolador e os displays.

O ULN2003 é constituído por uma rede com sete transistores Darlington de capacidade de corrente de 500 mA cada um. A base desses transistores está em série com resistores que permitem conexão direta com as lógicas TTL ou CMOS. O esquema de cada um dos drivers do UL2003 é apresentado na Fig. 5.22.

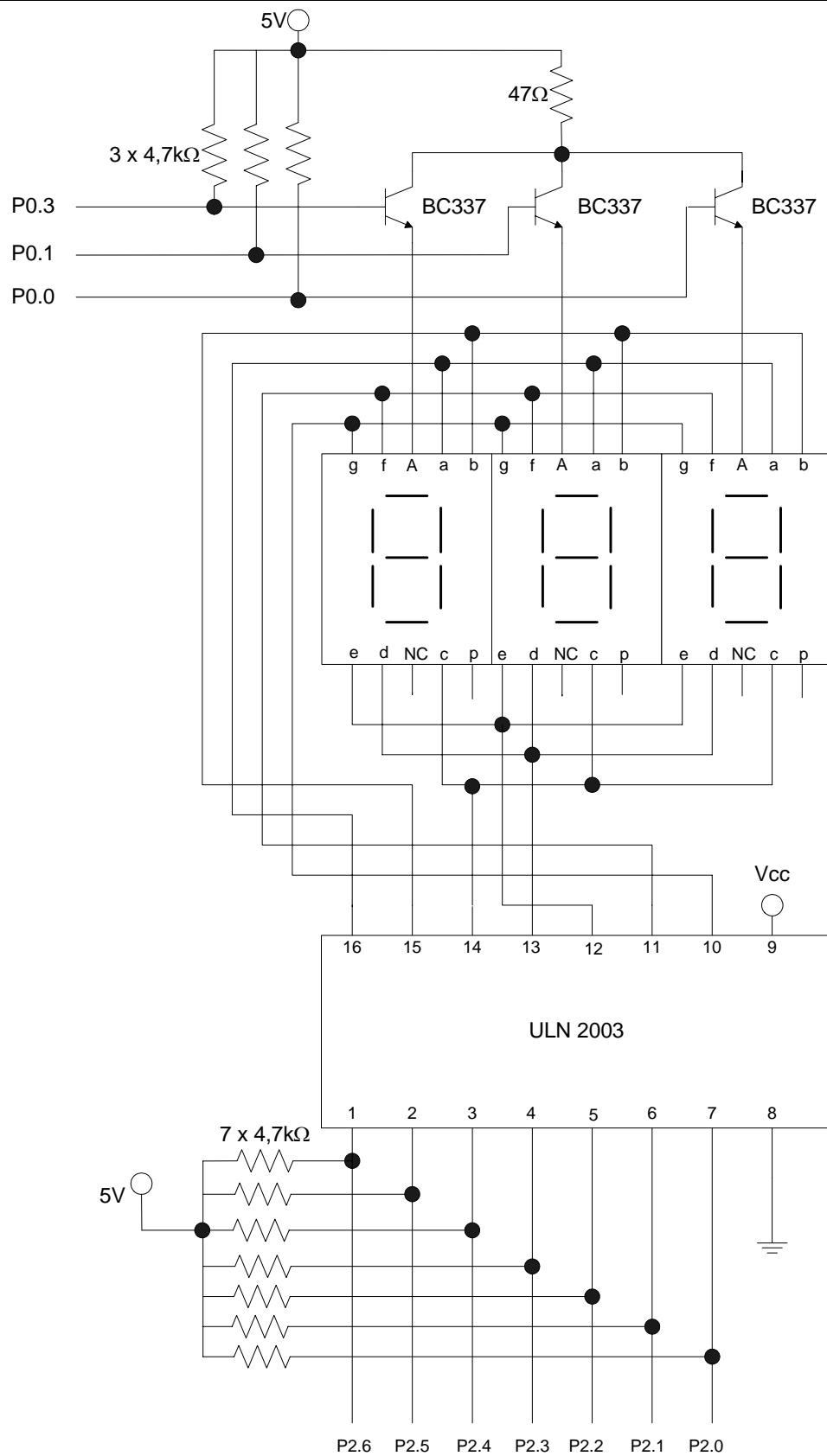


Fig. 5.21 – Interface para Display de 7 segmentos da Placa de Desenvolvimento.

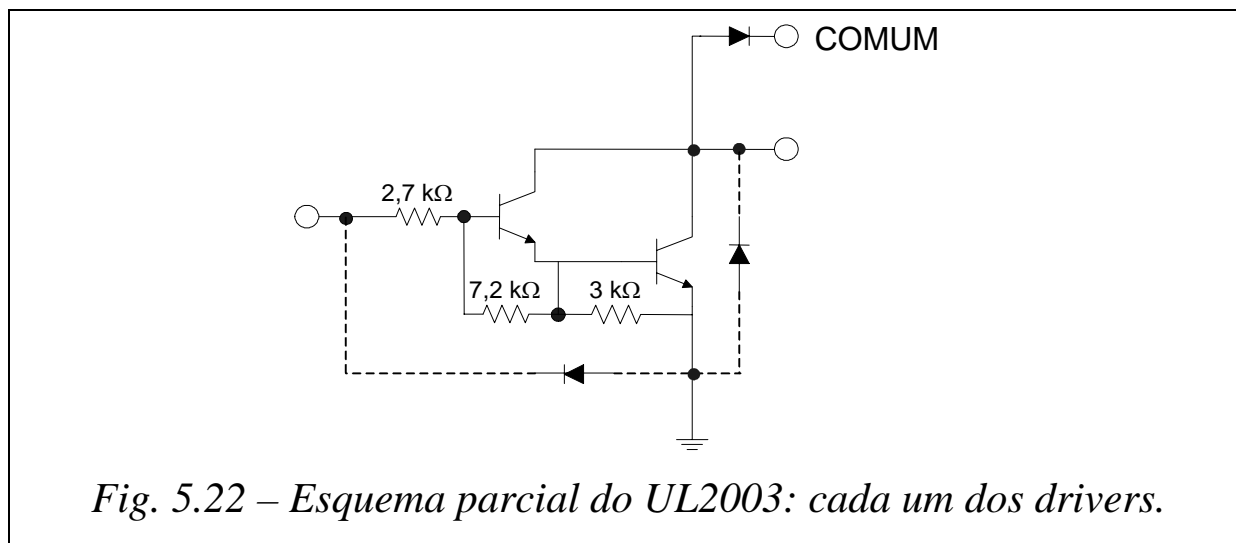


Fig. 5.22 – Esquema parcial do UL2003: cada um dos drivers.

Para utilizar a interface para display de 7 segmentos é necessário configurar dois conjuntos de *jumpers*, a saber, um situado entre o microcontrolador e o ULN2003 deve ser colocado na posição “DRIVE”, o outro, localizado próximo ao conector P4, deve ser colocado na posição “DISPLAY/LED”. Esses *jumpers* são apresentados pela Fig. 5.23. A colocação dos jumpers na posição “NORMAL” liga o conector P4 e os displays diretamente ao microcontrolador e a colocação dos *jumpers* do outro conjunto na posição “NC” rompe a conexão entre os displays e o conector P4/microcontrolador. É importante frisar que conectando o *jumper* da extremidade direita do segundo conjunto na posição “DISPLAY/LED”, você estará ligando os segmentos “p” (ponto) dos displays diretamente ao mesmo microcontrolador, mesmo que os *jumpers* do primeiro conjunto estejam na posição “DRIVE”.

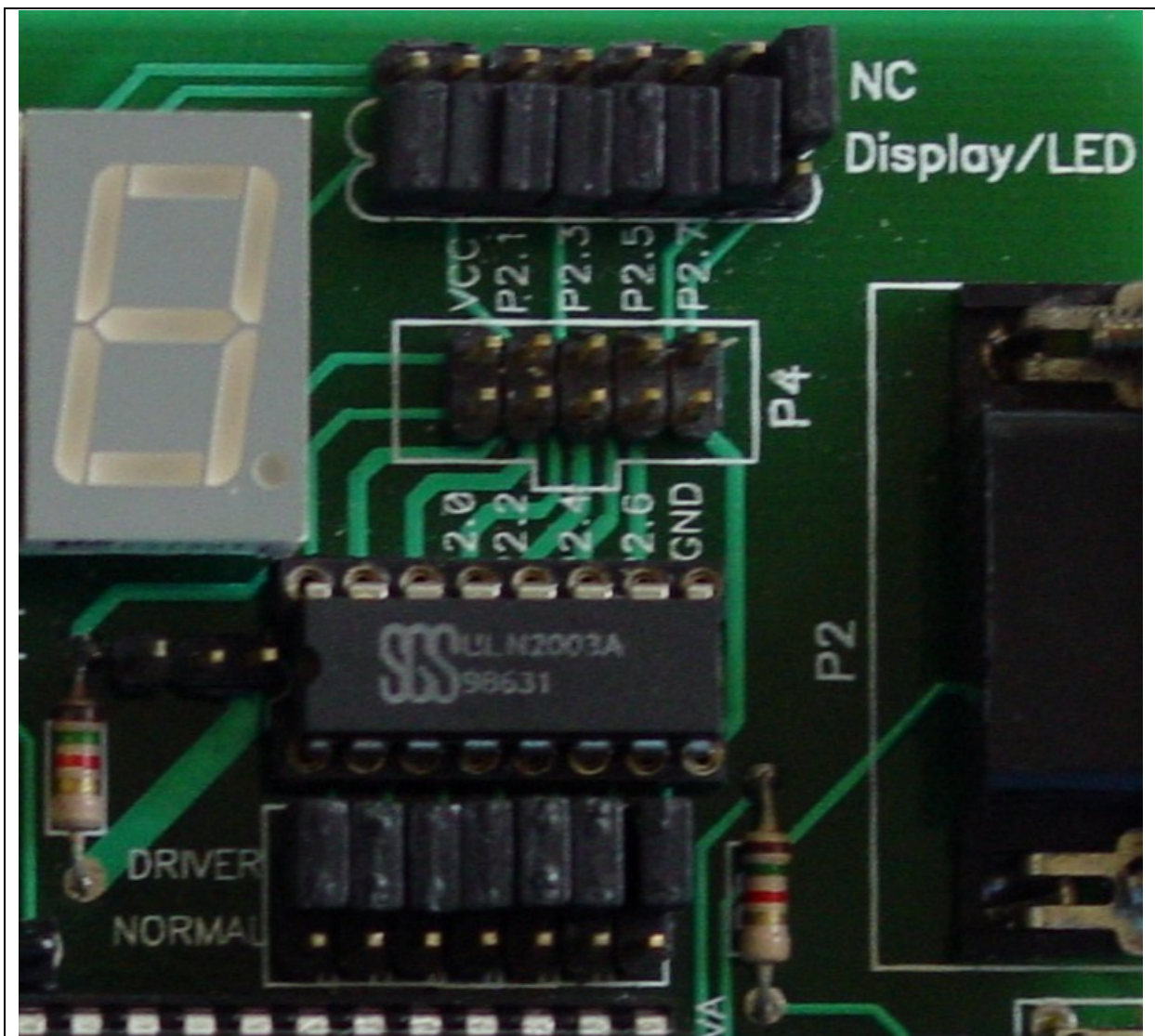
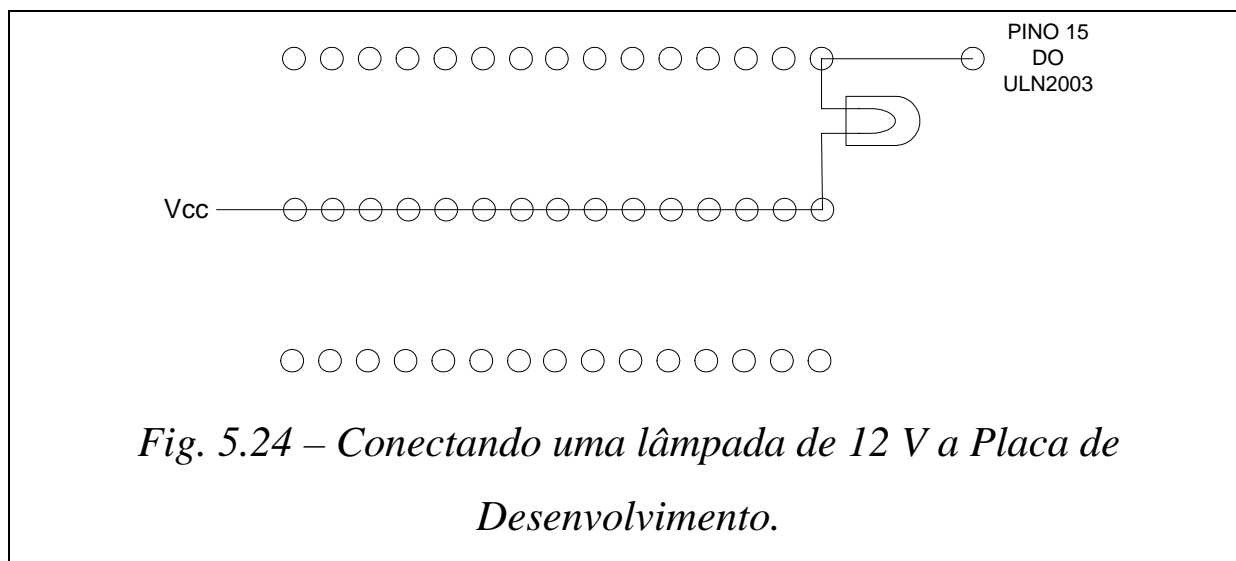


Fig. 5.23 – Utilizando a interface para display de 7 segmentos.

Retirando os displays dos soquetes, você pode utilizar essa área da Placa de Desenvolvimento para acionar LEDs, Lâmpadas e relés, conforme ilustra a Fig. 5.24. Certifique-se, contudo, da configuração dos *jumpers*.



Para ilustrar o funcionamento da interface, apresentamos, a seguir, a listagem do programa “CRONO”, que utiliza os *displays* para implementar um cronômetro com registro máximo de 10 min. Nesse programa, cada *display* é acionado por 6,6 ms, resultando em uma frequência de multiplexação de 50 Hz, suficientemente elevada para que não se perceba o acionamento intermitente dos displays. O *display* da esquerda registra os minutos, o central, a dezena do segundo e o *display* da direita, a unidade do segundo.

Listagem do Programa CRONO

```

; Saída em Display de 7 segmentos
;
;      a                seg pino
;      - - -            p P2.7
; f |          | b      a P2.6
;   |    g    |        b P2.5
;      - - -            c P2.4
; e |          | c      d P2.3
;   |          |        e P2.2
;      - - - . p        f P2.1
;      d                g P2.0
;
; Definições
un    EQU    R0          ; unidade
dz    EQU    R1          ; dezena
cn    EQU    R3          ; centena
atras EQU    R5          ; define atraso
atend EQU    R2          ; atendimentos p/ 1s
du    BIT    P0.3        ; liga disp. da unidade
dd    BIT    P0.1        ; liga disp. da dezena
dc    BIT    P0.0        ; liga disp. da centena
;
        SJMP MAIN
        ORG 0Bh
; Rotina de Atendimento de Interrupção de Timer 0
        MOV TH0,#10h      ; recarga do valor
                           ; inicial de Timer 0
        DJNZ atend,SALTO  ; após 15 atendimentos 1
                           ; segundo
        INC un            ; incrementa unidade

```

Listagem do Programa CRONO

```

CJNE un,#0Ah,SALTO1 ; Se un não chegou a 10
                        desvia p/ SALTO1

MOV un,00h            ; zera a unidade
INC dz                ; incrementa a dezena
CJNE dz,#06h,SALTO1 ; Se dz não chegou a 6
                        desvia p/ SALTO1

MOV dz,#00h           ; Zera a dezena
INC cn                ; incrementa a centena
CJNE cn,#0Ah,SALTO1 ; Se cn não chegou a 10
                        desvia p/ SALTO1

MOV cn,#00h           ; Zera a centena
SALTO1: MOV atend,#0Fh ;restaura 15 atendimen-
                        tos

SALTO: RETI
;Inicializações
MAIN:  CLR du          ; Desliga displays
        CLR dd
        CLR dc
        MOV un,#0h     ; Zera unidade, dezena e
                        centena

        MOV dz,#0h
        MOV cn,#0h
        MOV DPTR,#TAB  ; Põe no DPTR end. que
                        aponta p/ a formação dos
                        caracteres no Display

        MOV atend,#0Fh ; Define 15 atendimentos
        MOV IE,#82h    ; Habilita interrupção
                        por Timer 0

        MOV TMOD,#01h  ; Timer 0 - Temp. de 16
                        bits, ligado por soft.

        MOV TH0,#10h   ; Carrega valor inicial
        MOV TL0,#00h

```


Listagem do Programa CRONO

```

      SETB TR0                ; Liga Timer 0
; Rotina de Multiplexação
LOOP:  MOV A,un              ; unidade no acumulador
      MOVC,@A+DPTR          ; busca na tabela os
                              segmentos que devem
                              acender em função do
                              valor atual de A

      CLR dc                ; desliga display da
                              centena

      MOV P2,A              ; unidade no display
      SETB du               ; liga disp. da unidade
      MOV atras,#12         ; Seta atraso da rotina
                              Delay: R5 x 256 x T x 24

      ACALL DEL             ; Chama Delay
;

      MOV A,dz              ; dezena no acumulador
      MOVC,@A+DPTR          ; busca na tabela os
                              segmentos que devem
                              acender em função do
                              valor atual de A

      CLR du                ; desliga display da
                              unidade

      MOV P2,A              ; dezena no display
      SETB dd               ; liga disp. da unidade
      MOV atras,#12         ; Seta atraso da rotina
                              Delay: R5 x 256 x T x 24

      ACALL DEL             ; Chama Delay
;

      MOV A,cn              ; dezena no acumulador
      MOVC,@A+DPTR          ; busca na tabela os
                              segmentos que devem
                              acender em função do
                              valor atual de A

```

Listagem do Programa CRONO

```

CLR dd                ; desliga display da
                      unidade

MOV P2,A              ; centena no display

SETB dc               ; liga disp. da unidade

MOV atras,#12         ; Seta atraso da rotina
Delay: R5 x 256 x T x 24

ACALL DEL              ; Chama Delay

SJMP LOOP             ; Desvia para o inicio
                      da rot. de multiplexação

;Rotina de Atraso
DEL:  DJNZ R4,$
      DJNZ R5,$-2
      RET

;
TAB:
;      db pabcdefgB      ; "Algarismo"
                      ; (0 p/ segmento apagado
                      ; e 1 p/ segmento aceso,
                      ; exceto o ponto)

      db 11111110B      ; "0"
      db 10110000B      ; "1"
      db 11101101B      ; "2"
      db 11111001B      ; "3"
      db 10110011B      ; "4"
      db 11011011B      ; "5"
      db 11011111B      ; "6"
      db 11110000B      ; "7"
      db 11111111B      ; "8"
      db 11111011B      ; "9"

      END              ; Fim do Assembly

```

VI - OS MICROCONTROLADORES AVR

6.1 – Introdução:

Os microcontroladores AVR, desenvolvidos na Noruega (1995) e produzidos pela ATMEL Corporation, apresentam, a exemplo dos conhecidos *PICs*, uma arquitetura Harvard com processador RISC de 8 bits em dispositivos disponíveis em encapsulamentos de 8 a 64 pinos. Eles são recomendados para pequenos sistemas de controle e de monitoramento, embutem uma ampla gama de periféricos e possuem, também, a capacidade de expansão para aumentar a sua funcionalidade.

O processador núcleo em todos os dispositivos é o mesmo, sendo que todos eles, exceto a série FPSLIC¹⁰, têm memória de programa flash e executam essencialmente o mesmo conjunto de instruções. Alguns modelos AVRs são compatíveis pino a pino com os processadores MCS-51™ (com exceção de terem um pino de reset ativo baixo) e todos incorporam uma EEPROM interna, com capacidade variando de 64 bytes a 4 kbytes, dependendo do dispositivo, e que pode ser escrita pelo software. As memória flash e EEPROM são ambas reprogramáveis “in circuit” através de uma

¹⁰ *Field Programmable System Level Integrated Circuits*. Estes dispositivos combinam blocos lógicos, memória e um microcontrolador AVR em um dispositivo programável baseado em SRAM.

interface serial síncrona SPI. Para prevenir a leitura de seu programa estão disponíveis bits de proteção (lock bits). Outras características comuns a todos os membros da família são:

- Um vetor de interrupção para cada fonte de interrupção;
- Um comparador analógico, o qual é também uma fonte de interrupção;
- Ao menos um pino de interrupção externa;
- Ao menos um temporizador/contador de alta resolução com *prescaler*;
- Um temporizador *watchdog* ou cão de guarda, programável de 16 ms a 2 s, que supervisiona o correto funcionamento da CPU ;
- 32 registradores de 1 byte que podem ser encarados como acumuladores ou *workings*. Alguns pares de registradores podem realizar operações de 16 bits.

A família oferece muitas opções. Alguns componentes têm temporizadores de 8 e 16 bits. Outros oferecem ADCs com múltiplos canais de 10 bits e outros uma ou mais UARTS para comunicação serial. A maioria oferece um oscilador interno para eliminar a necessidade de um cristal externo. A Tabela 6.1 apresenta alguns dispositivos com arquitetura AVR. Há três famílias básicas dentro da arquitetura AVR. A família original é a AT90; para aplicações mais complexas, a ATMEL oferece a família ATmega; para aplicações que requerem microcontroladores menores, disponibiliza a família ATtiny.

TABELA 6.1
COMPARAÇÃO DE DISPOSITIVOS DA FAMÍLIA AVR

Dispositivo	Nº de pinos, pinos de E/S	Flash (kbytes)	RAM (bytes)	EEPROM (bytes)	Frequência máxima (MHz)	Timers de 8 bits	Timers de 16 bits	Outros periféricos
ATtiny 11	8,6	1	-	-	6	1	-	-
ATtiny 12	8,6	1	-	64	8	1	-	-
ATtiny 13	8,6	1	64	64	20	1	-	2 canais PWM, ADC com 4 canais de 10 bits.
AT90S1200	20,15	1	-	64	12	1	-	
AT90S2313	20,15	2	128	128	10	1	1	1 canal PWM, 1 UART.
ATmega128	64, 53	128	4096	4096	16	2	2	8 canais PWM, 2 UARTs, ADC com 8 canais de 10 bits, detector brown out, RTC, SPI.
ATmega8515	40,35	8	512	512	16	1	1	3 canais PWM, 1 UARTs, , detector brown out, SPI
ATmega8535	40,32	8	512	512	16	2	1	4 canais PWM, 1 UARTs, ADC com 8 canais de 10 bits, detector brown out, SPI.

Em uma ruptura a partir da arquitetura CISC tradicional, os AVRs usam o que é chamado de arquitetura RISC ampliada. É ampliada no sentido que eles tem de 89 a 120 instruções diferentes, muito mais do que se esperaria em um RISC tradicional. A *filosofia* RISC é a execução de uma instrução em um único ciclo de relógio, permitindo alcançar um pico de 16 MIPS (milhões de instruções por segundo) com um cristal de 16 MHz. Enquanto muitos processadores dizem apresentar uma execução de ciclo único, a definição de ciclo único varia de acordo com o fabricante. Usualmente, um ciclo, na realidade, consiste de 4, 6 ou mais ciclos do oscilador. Alguns vão mais além e usam a palavra *clock* para significar ciclo da CPU, aumentando ainda mais a confusão. Nos AVRs, um ciclo significa exatamente um ciclo do oscilador, tal que a velocidade que se obtém é a frequência do cristal.

Para efeito de comparação (CATSOULIS, 2002), a ATMEL apresenta um código de um programa em C, o qual é compilado e executado em diversos processadores, cujo resultado é apresentado na Tabela 6.2.

Esta tabela indica que, quando operando com o mesmo relógio, um AVR é sete vezes mais rápido que um PIC, 15 vezes mais rápido que um 68HC11 e 28 vezes mais rápido que um 8051. Este resultado deve certamente estar superestimado, mas é certo que os AVRs oferecem um código significativamente denso e uma execução muito mais rápida.

TABELA 6.2
COMPARAÇÃO ENTRE DIVERSOS PROCESSADORES

Processador	Tamanho do código compilado	Tempo de execução (ciclos)	Velocidade relativa(%)
AVR	46	335	100
8051	112	9384	3,6
PIC16C74	87	2492	13,4
68HC11	57	5244	6,4

Mesmo que você não necessite da alta performance oferecida pelo AVR, devido à drástica redução do *clock*, você pode reduzir o consumo de energia. Por exemplo, uma aplicação que requer um cristal de 12 MHz em um MCS-51™ pode ser implementada em um AVR com um cristal de 1 MHz. Como o consumo dos CMOS é proporcional a velocidade do seu *clock*, o AVR pode, potencialmente, reduzir o consumo de energia e, também, reduzir as emissões eletromagnéticas.

6.2 – Apresentação do Microcontrolador ATMEL AT90S8515

O AT90S8515 está disponível em um encapsulamento de 40 pinos compatível pino a pino com o AT89S8252, exceto pelos pinos ICP (31) e OC1B (29).

Suas principais características são:

- Dispõe de 118 instruções, a maioria executada em um único ciclo de *clock*;

- Possui 32 registradores de trabalho de propósito geral de 8 bits conectados diretamente a ULA;
- Executa até 12 MIPS a 12 MHz;
- Apresenta uma memória flash de 8 Kbytes que permite 1.000 ciclos de apagamento/escrita e 512 bytes de memória EEPROM de 100.000 ciclos, ambas programáveis *in-system*. Para a segurança dos dados permitem o uso de bits de segurança;
- Apresenta uma memória SRAM de 512 bytes;
- Apresenta os seguintes periféricos:
 - ✓ 2 Temporizadores/contadores com *prescaler* separado: um de 8 bits e outro de 16 bits com função PWM;
 - ✓ Comparador analógico;
 - ✓ Temporizador watchdog programável com oscilador próprio;
 - ✓ Interface serial SPI;
 - ✓ UART.
- Dispõe dos modos de redução consumo (*sleep*)- espera (*idle*) e baixo consumo (*power-down*);
- Apresenta fontes de interrupção externa e interna;
- Apresenta 32 linhas programáveis de E/S.

O diagrama de blocos do AT90S1200, o primeiro integrante da família, é apresentado na Fig. 6.1. Seu núcleo AVR conecta todos

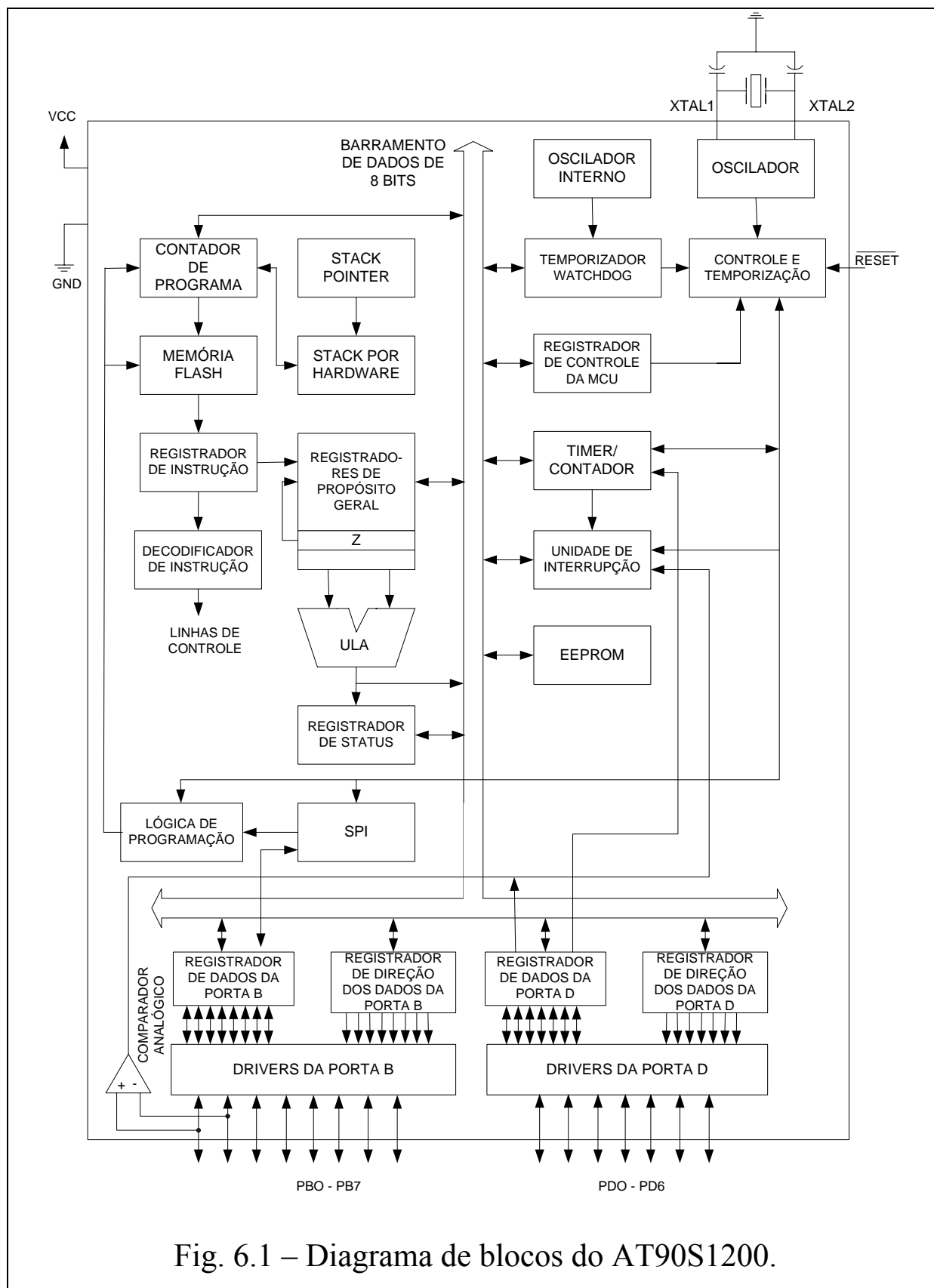
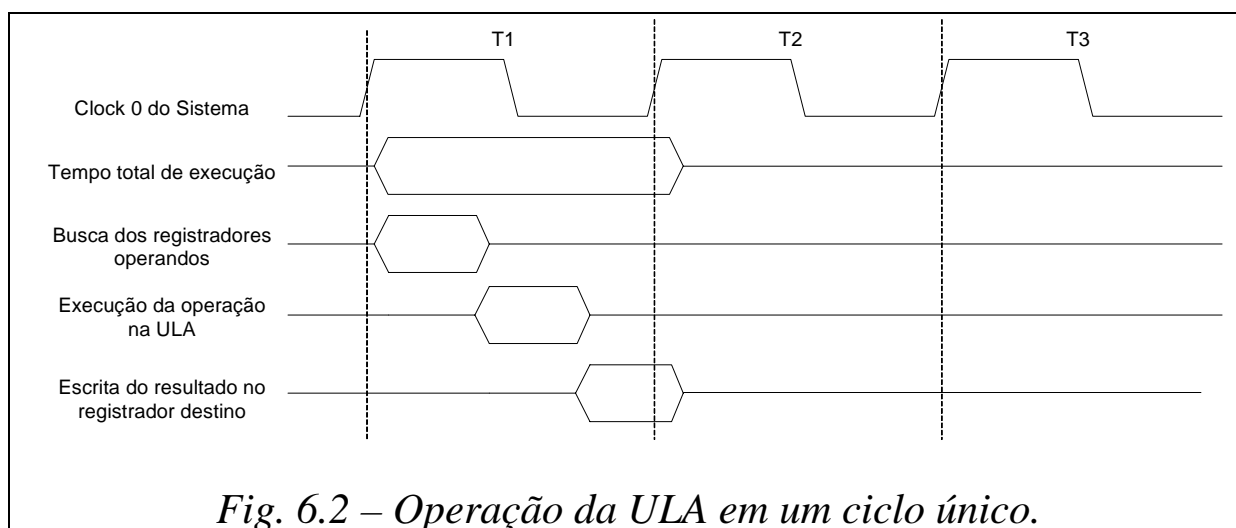


Fig. 6.1 – Diagrama de blocos do AT90S1200.

os 32 registradores a ULA, permitindo que 2 registradores independentes sejam acessados em uma única instrução executada em um ciclo de clock. Esta característica, aliada a sua arquitetura Harvard, permite a busca da próxima instrução em paralelo com a execução da instrução atual. Este é o conceito básico de *pipeline* que permite obter 1 MIPS por MHz.

A Fig. 6.2 mostra que em um único ciclo uma operação da ULA usando dois registradores é executada e o resultado é armazenado no registrador de destino.



6.3 – Oscilador Interno:

O oscilador interno do AT90S8515 pode ser usado como uma fonte de relógio para a CPU. Para usá-lo, devem ser conectados um cristal ou um ressonador cerâmico conectado entre os terminais XTAL1 e XTAL2, um capacitor entre XTAL1 e a referência e outro entre XTAL2 e a referência, conforme mostra a Fig. 6.3. Para controlar o dispositivo a partir de uma fonte de relógio externa, XTAL2 deve ser mantido desconectado, conforme ilustra a Fig. 6.4.

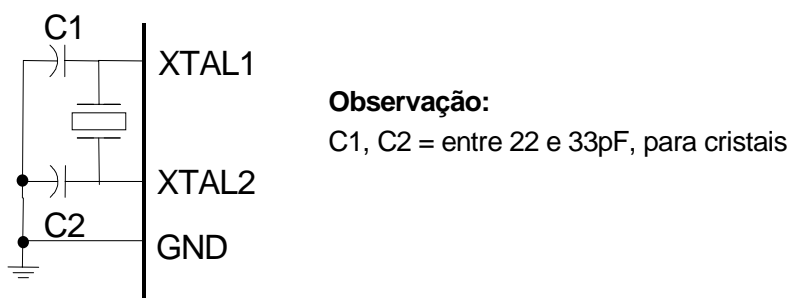


Fig. 6.3 – Conexões do oscilador.

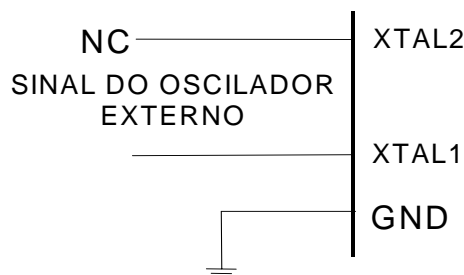
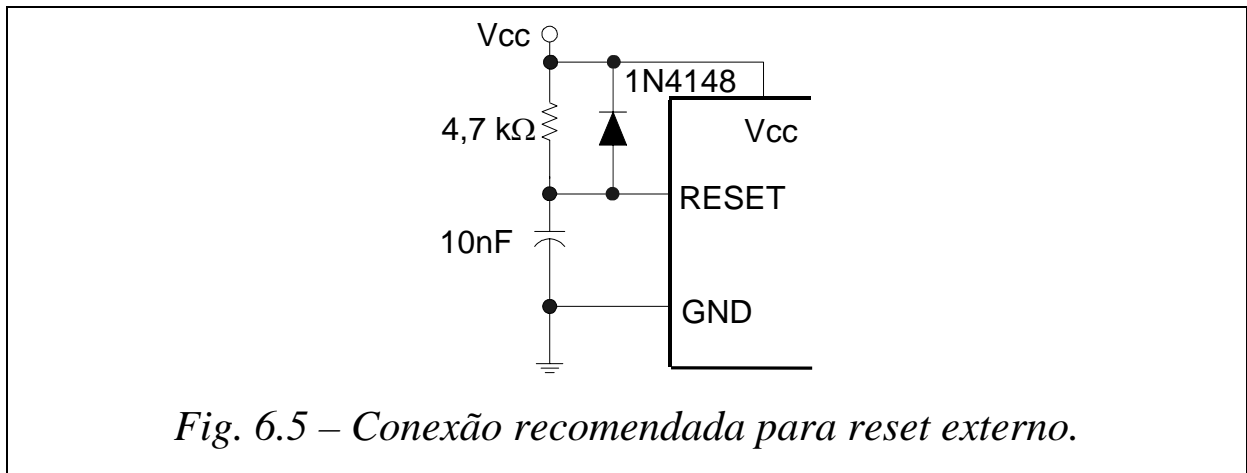


Fig. 6.4 – Configuração com clock externo.

6.4– Reset:

Um reset externo é obtido colocando-se o terminal $\overline{\text{RESET}}$ externamente em nível baixo por mais de 50ns. A CPU responde gerando um sinal de reset interno, cujo algoritmo coloca todos os terminais em alta impedância, inicializa todos os registradores de E/S e coloca o contador de programa em zero. A linha de RESET possui um resistor interno de pull-up, cujo valor se situa entre 100 k Ω e 500 k Ω .

O circuito de RESET sugerido pela ATMEL é apresentado na Fig. 6.5. Se o reset externo não for necessário e não for utilizada programação “in system” a linha de reset pode ser conectada diretamente a V_{CC} .



O AT90S8515 possui, ainda, outras duas fontes de reset:

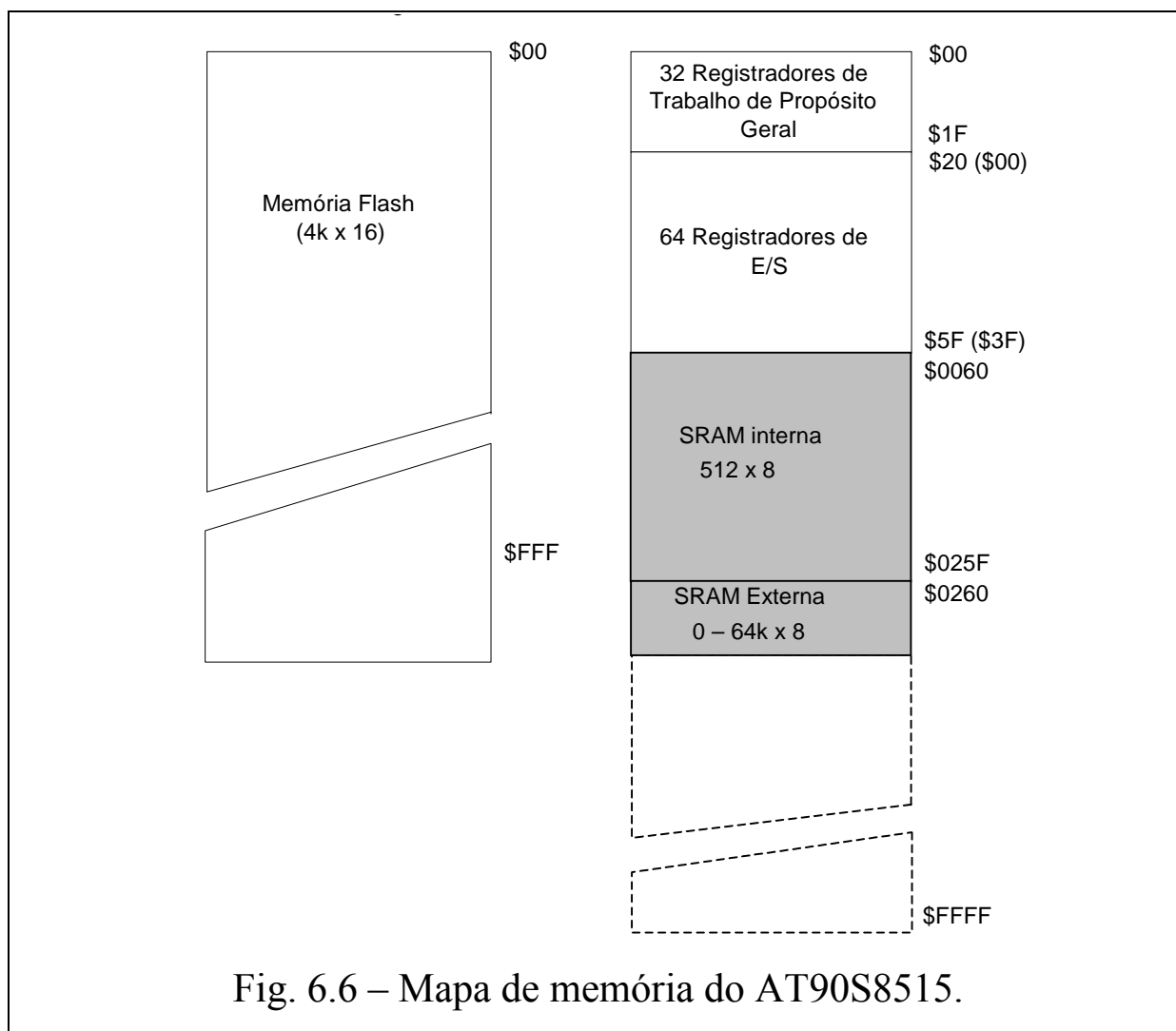
- Reset ao ligar (Power-on Reset): o microcontrolador é inicializado um certo tempo¹¹ após quando a tensão de alimentação atinge um nível limiar (V_{POT}).
- Reset por watchdog: quando o período do temporizador watchdog expira, é gerado um pulso estreito de duração igual a um ciclo do relógio do sistema. Um certo tempo (t_{out}) após a borda de descida desse pulso a CPU reinicializa.

6.6 – Organização de Memória:

A Fig. 6.6 apresenta os mapas da memória de programa e de dados do AT90S8515. A memória de programa apresenta 4k posições de 16 bits, sendo que cada instrução ocupa uma posição. As 32 primeiras posições da memória de dados são ocupadas pelo banco de registradores de propósito geral. A estrutura dos 32 registradores de propósito geral é apresentada na Fig. 6.7. Toda instrução que opera

¹¹ Esse tempo é chamado período de atraso do reset (t_{out}) e vale, tipicamente, 16 ms para $V_{CC} = 5$ V.

com os registradores tem acesso direto e de ciclo único a todos os registradores, exceto as cinco instruções aritméticas e lógicas SBCI, SUBI, CPI, ANDI e ORI entre uma constante e um registrador e a instrução LDI para carga imediata de uma constante. Essas instruções se aplicam apenas a segunda metade dos registradores (R16 a R31) do banco.



Os registradores R_{16} a R_{31} constituem os ponteiros X, Y e Z para o acesso indireto da memória de dados. A Fig. 6.8 apresenta os registradores X, Y e Z.

	7	0	
	R1		\$00
	R2		\$01
	R3		\$02
	...		
	R13		\$0D
	R14		\$0E
	R15		\$0F
	R16		\$10
	R17		\$11
	...		
	R29		\$1D
	R30 (Z)		\$1E
	R31		\$1F

Fig. 6.7 – Registradores de propósito geral.

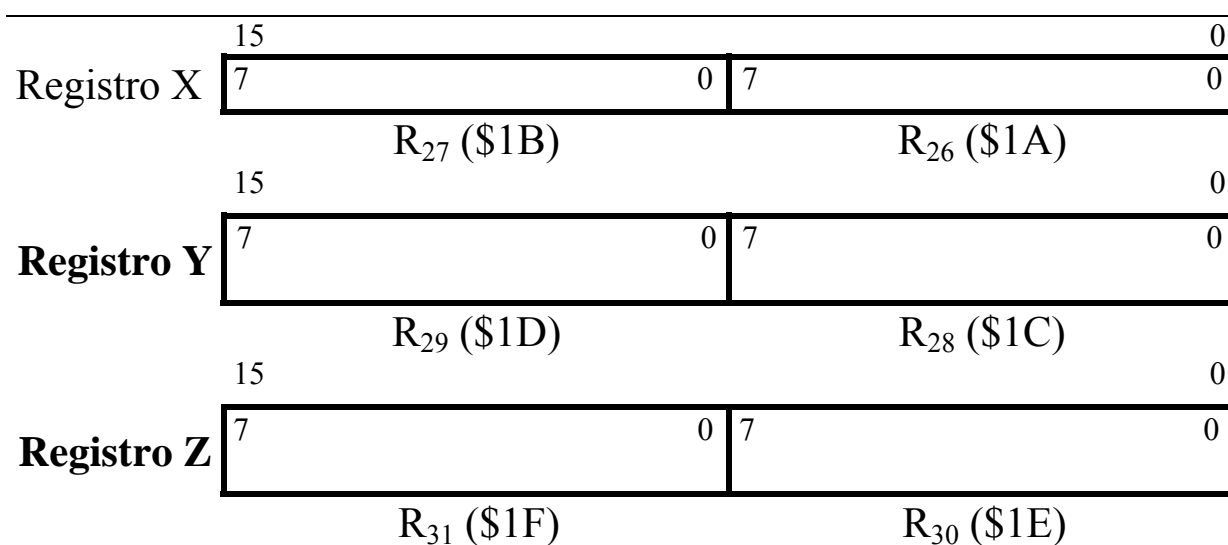


Fig. 6.8 – Registradores X, Y e Z.

A Estrutura dos registradores de E/S é apresentada na Tabela 6.3. Se as instruções IN ou OUT forem usadas, os endereços de E/S vão de \$00 a \$3F. Se os registradores de E/S forem acessados como uma SRAM, os endereços anteriores devem ser somados a \$20, indo de \$20 a \$5F.

A SRAM aloca a pilha. O Stack Pointer, acessível no espaço de E/S, deve ser inicializado pelo programa do usuário antes da execução de qualquer sub-rotina.

TABELA 6.3
ESPAÇO DE E/S DO AT90S8515

Endereço	Nome	Função	Endereço	Nome	Função
\$3F (\$5F)	SREG	Registrador de Estado	\$2C(\$4C)	TCNT1L	LSB do Registrador do Timer1
\$3E(\$5E)	SPH	Stack Pointer MSB	\$2B(\$4B)	OCR1AH	MSB do Registrador de Saída de Comparação A
\$3D(\$5D)	SPL	Stack Pointer LSB	\$2A(\$4A)	OCR1AL	LSB do Registrador de Saída de Comparação A
\$3B(\$5B)	GIMSK	Registrador Geral de Máscara de Interrupção	\$29(\$49)	OCR1BH	MSB do Registrador de Saída de Comparação B
\$3A(\$5A)	GIFR	Registrador Geral de Flags de Interrupção	\$28(\$48)	OCR1BL	LSB do Registrador de Saída de Comparação B
\$39(\$59)	TIMSK	Registrador Geral de Máscara de Interrupção por Timer	\$25(\$45)	ICR1H	MSB do Registrador de Entrada de Captura
\$38(\$58)	TIFR	Registrador Geral de Flags de Interrupção por Timer	\$24(\$44)	ICR1L	LSB do Registrador de Entrada de Captura
\$35(\$55)	MCUCR	Registrador de Controle Geral da MCU	\$21(\$41)	WDTCR	Registrador de Controle do Watchdog
\$33(\$53)	TCCR0	Registrador de Controle do Timer0	\$1F(\$3F)	EEARH	MSB do Registrador de Endereço da EEPROM
\$32(\$52)	TCNT0	Registrador do Timer0	\$1E(\$3E)	EEARL	LSB do Registrador de Endereço da EEPROM
\$2F(\$4F)	TCCR1A	Registrador de Controle A do Timer1	\$1D(\$3D)	EEDR	Registrador de Dados da EEPROM

TABELA 6.3
ESPAÇO DE E/S DO AT90S8515

\$2E(\$4E)	TCCR1B	Registrador de Controle B do Timer1	\$1C(\$3C)	EECR	Registrador de Controle da EEPROM
\$2D(\$4D)	TCNT1H	MSB do Registrador do Timer1	\$1B(\$3B)	PORTA	Registrador de Dados da PORTA A
\$1A(\$3A)	DDRA	Registrador de Direção de Dados da PORTA A	\$10(\$30)	PIND	Pinos de Entrada da PORTA D
\$19(\$39)	PINA	Pinos de Entrada da PORTA A	\$0F(\$2F)	SPDR	Registrador de Dados de E/S da SPI
\$18(\$38)	PORTB	Registrador de Dados da PORTA B	\$0E(\$2E)	SPSR	Registrador de Estado da SPI
\$17(\$37)	DDRB	Registrador de Direção de Dados da PORTA B	\$0D(\$2D)	SPCR	Registrador de Controle da SPI
\$16(\$36)	PINB	Pinos de Entrada da PORTA B	\$0C(\$2C)	UDR	Registrador de Dados de E/S da UART
\$15(\$35)	PORTC	Registrador de Dados da PORTA C	\$0B(\$2B)	USR	Registrador de Estado da UART
\$14(\$34)	DDRC	Registrador de Direção de Dados da PORTA C	\$0A(\$2A)	UCR	Registrador de Controle da UART
\$13(\$33)	PINC	Pinos de Entrada da PORTA C	\$09 (\$29)	UBRR	Registrador de Taxa de Transmissão UART
\$12(\$32)	PORTD	Registrador de Dados da PORTA D	\$08 (\$28)	ACSR	Registrador de Controle e Estado do Comparador Analógico
\$11(\$31)	DDRD	Registrador de Direção de Dados da PORTA D			

O AT90S8515 também contém uma memória EEPROM de 512 bytes, organizada em um espaço de dados separado. Essa EEPROM permite pelo menos 100.000 ciclos de escrita/apagamento.

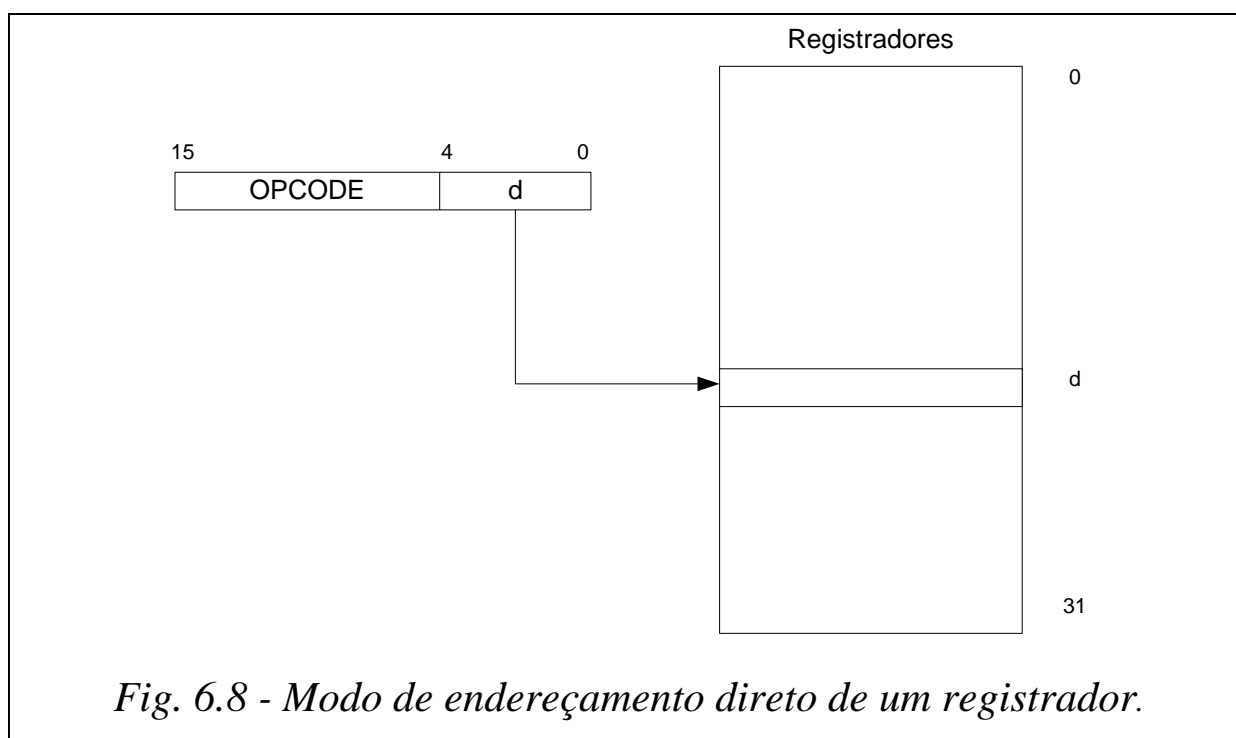
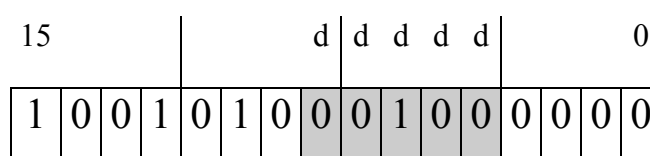
O AT90S8515 dispõe de doze modos de acesso à memória de dados que apresentmos a seguir. Para simplificar, as figuras que

ilustram os modos de endereçamento não mostrarão a localização exata dos bits de endereçamento.

6.6.1 – Modo de endereçamento direto de registrador, um registrador Rd

A instrução contém um campo de 5 bits que permite especificar qualquer um dos 32 registradores. A Fig. 6.8 ilustra este modo de endereçamento.

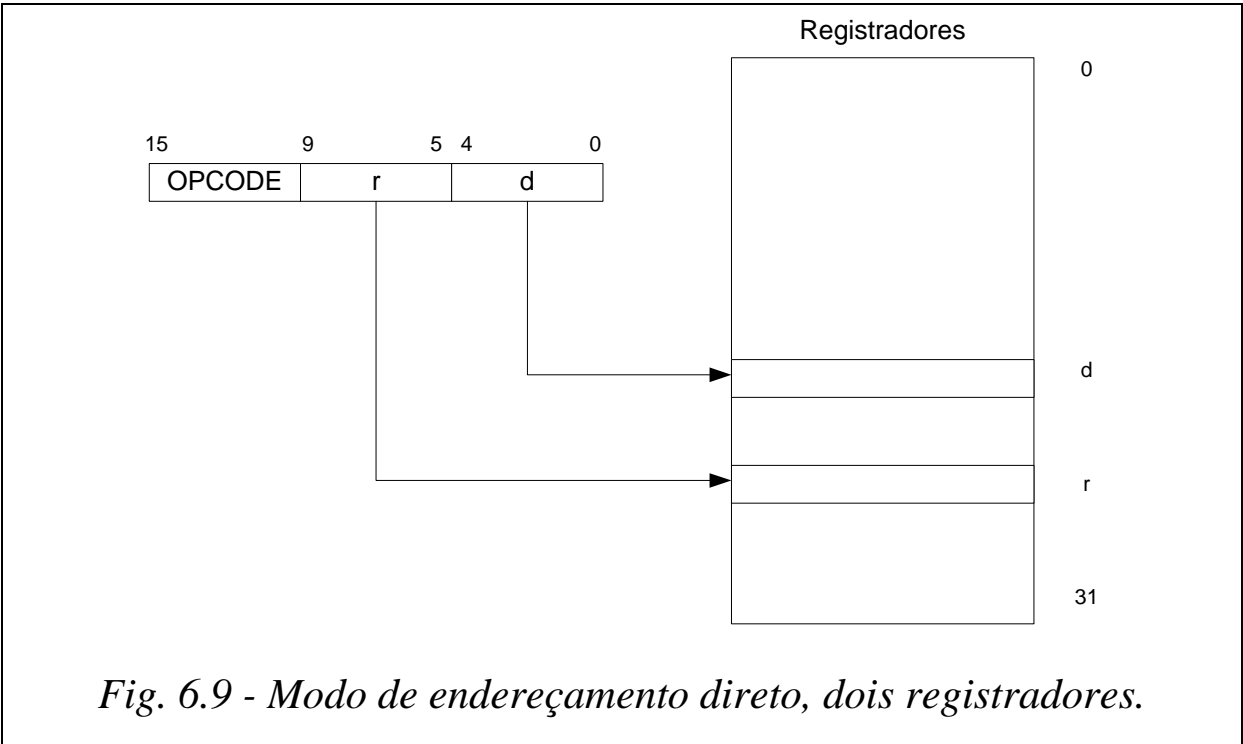
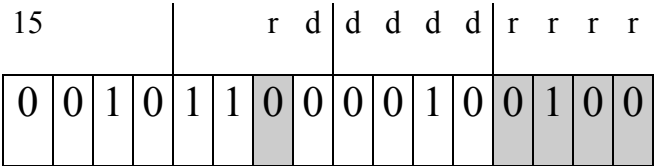
Exemplo: a instrução *COM R4* que calcula o complemento de 1 do registrador R4. O Código de máquina dessa instrução reserva um campo de 5 bits (em destaque) para especificar o número do registrador endereçado.



6.6.2 – Modo de endereçamento direto de registrador, dois registradores Rd e Rr

A instrução contém um campo de 5 bits para especificar o operando fonte (Rr) e outro o de destino (Rd). A Fig. 6.9 ilustra este modo de endereçamento.

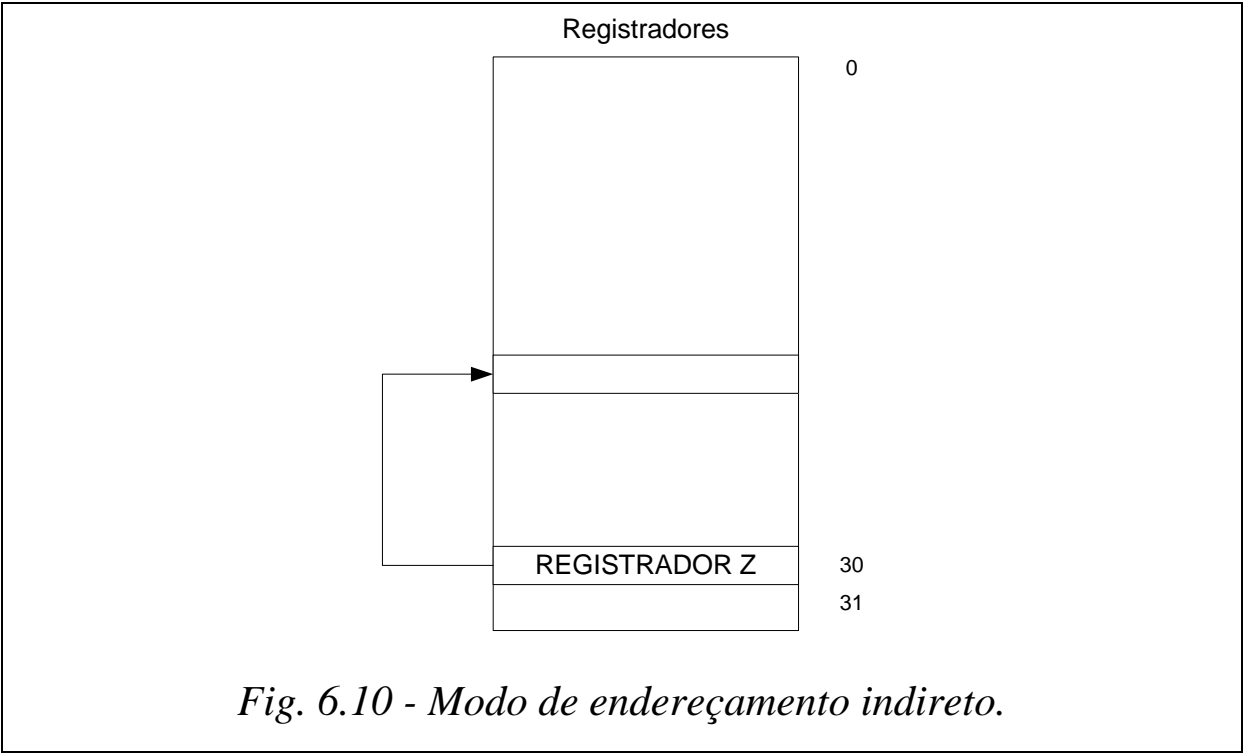
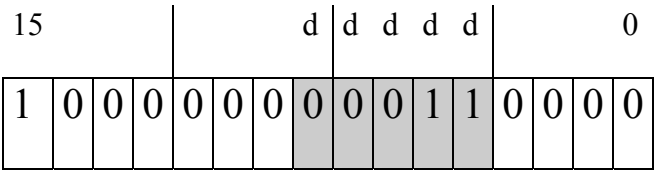
Exemplo: a instrução *MOV R2, R4* que copia o conteúdo de R4 em R2. O Código de máquina dessa instrução reserva 5 bits para especificar cada um dos registradores.



6.6.3 – Modo de endereçamento indireto de registrador

O registrador acessado por essa instrução é o apontado pelo registrador Z (R30). A Fig. 6.10 ilustra este modo de endereçamento.

Exemplo: a instrução *LD R3,Z* que copia o conteúdo da posição de memória apontada pelo registrador Z em R3. O Código de máquina dessa instrução reserva 5 bits (em destaque) para especificar o registrador.



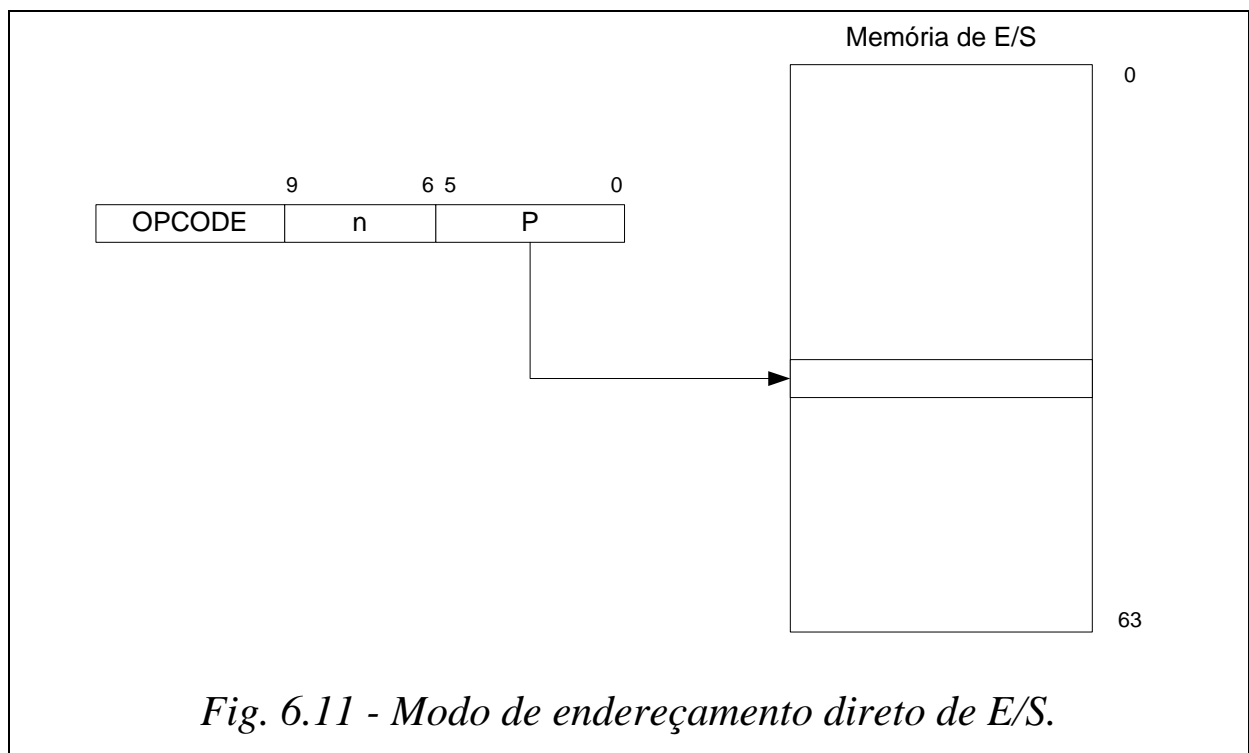
6.6.4 – Modo de endereçamento direto de E/S

Neste caso, a instrução contém um campo P para especificar o endereço de 6 bits do registrador de E/S e outro, n, para o registrador

que atua como fonte ou destino da informação. A Fig. 6.11 ilustra este modo de endereçamento.

Exemplo: a instrução *OUT \$18, R16* que copia o conteúdo de R16 para a Porta B. O Código de máquina dessa instrução reserva 5 bits para especificar o registrador fonte e outros 6 para o endereço da Porta B (em destaque).

15					A	A	r	r	r	r	r	A	A	A	A
1	0	1	1	1	0	1	1	0	0	0	0	1	0	0	0

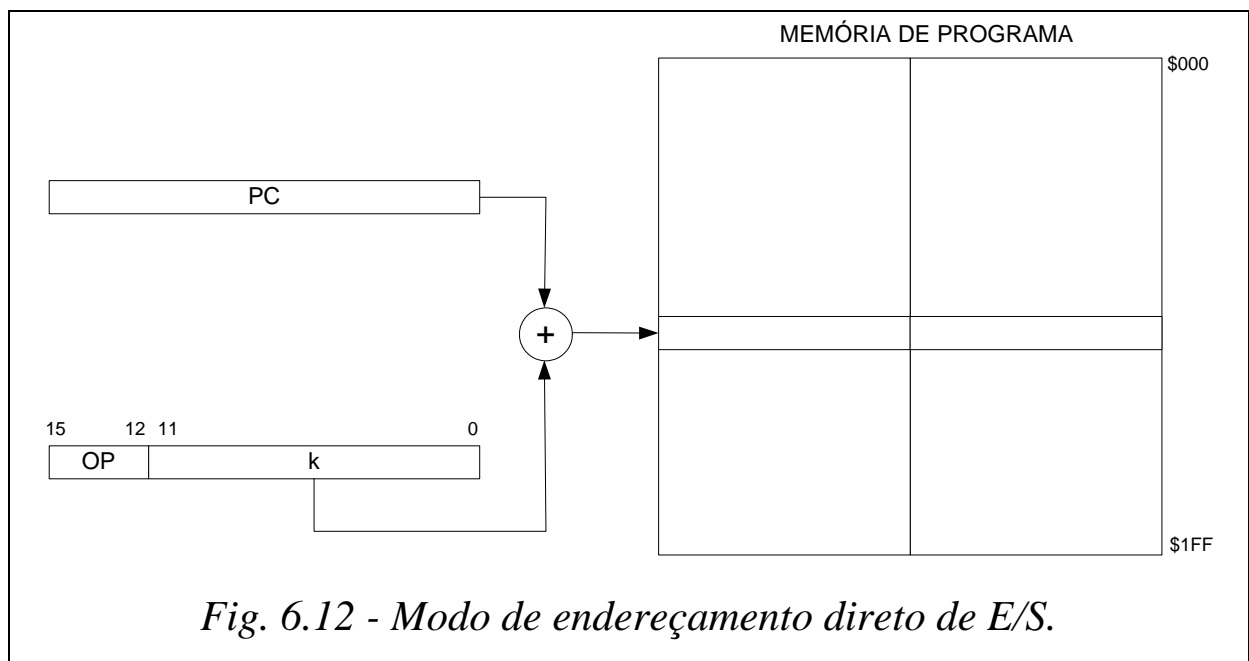
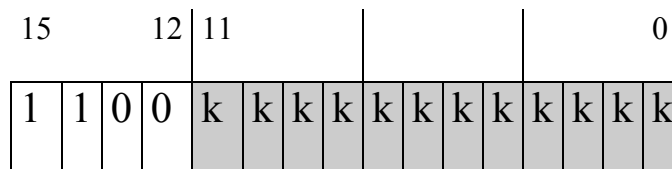


6.6.5 – Modo de endereçamento relativo de memória de programa

Neste caso, a instrução contém um deslocamento k que é somado ao conteúdo do PC para gerar a próxima instrução a ser

executada, ou seja, a execução do programa continua no endereço $PC + k + 1$, onde k varia de -2048 a 2047 . A Fig. 6.12 ilustra este modo de endereçamento.

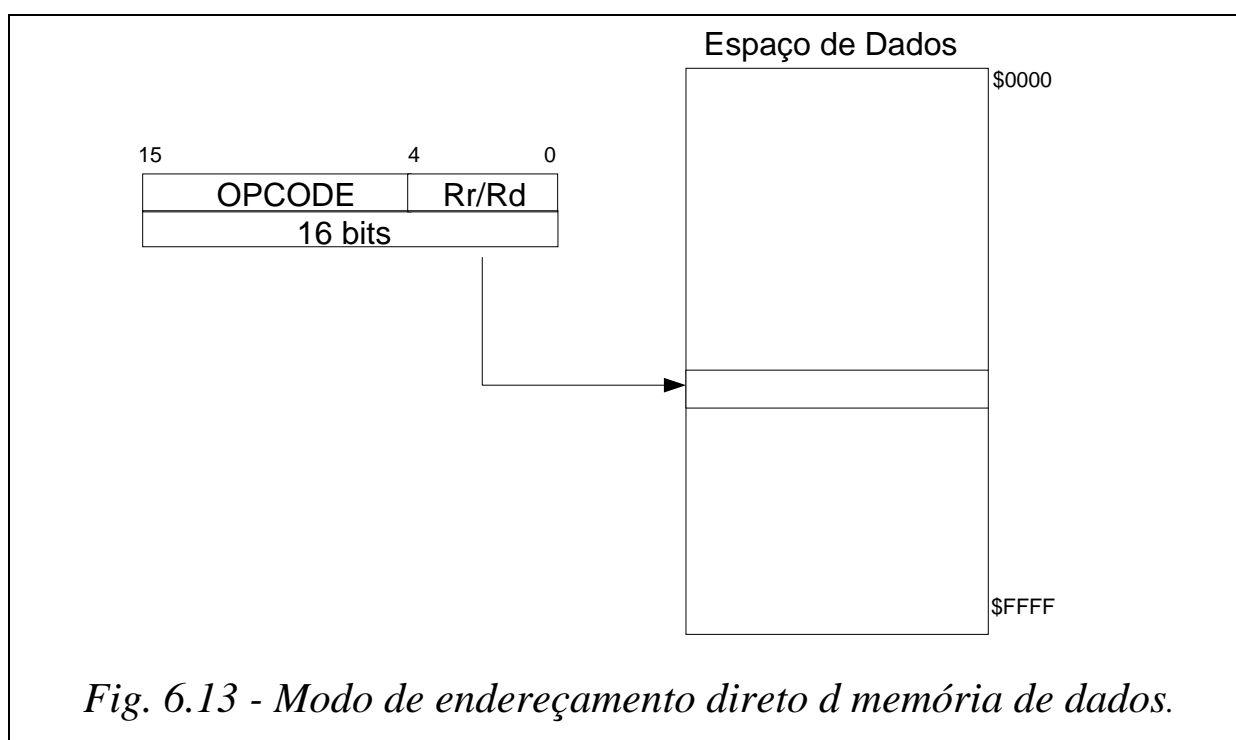
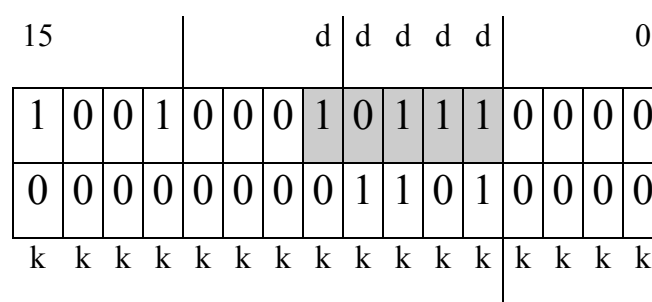
Exemplo: a instrução *RJMP vetor* que ocasiona um desvio incondicional da execução do programa para a posição de memória rotulada como vetor. O Código de máquina dessa instrução reserva 12 bits para especificar desvio relativo k (em destaque).



6.6.6 – Modo de endereçamento direto da memória de dados

A instrução vem acompanhada de uma palavra de 16 bits que contém o endereço de memória (SRAM, E/S, registrador), além de um campo Rd/Rr que contém o registro que será a fonte ou destino da informação. A Fig. 6.13 ilustra este modo de endereçamento.

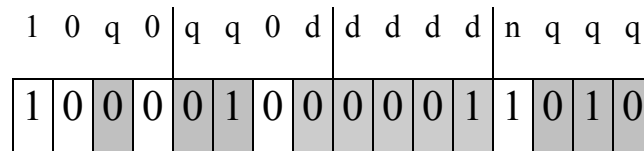
Exemplo: a instrução `LDS R23,$d0` que carrega o conteúdo da posição \$d0 no registrador R23. O Código de máquina dessa instrução reserva um campo de 5 bits (em destaque) para especificar o número do registrador endereçado e os n bits k indicam a posição de memória de dados endereçada.



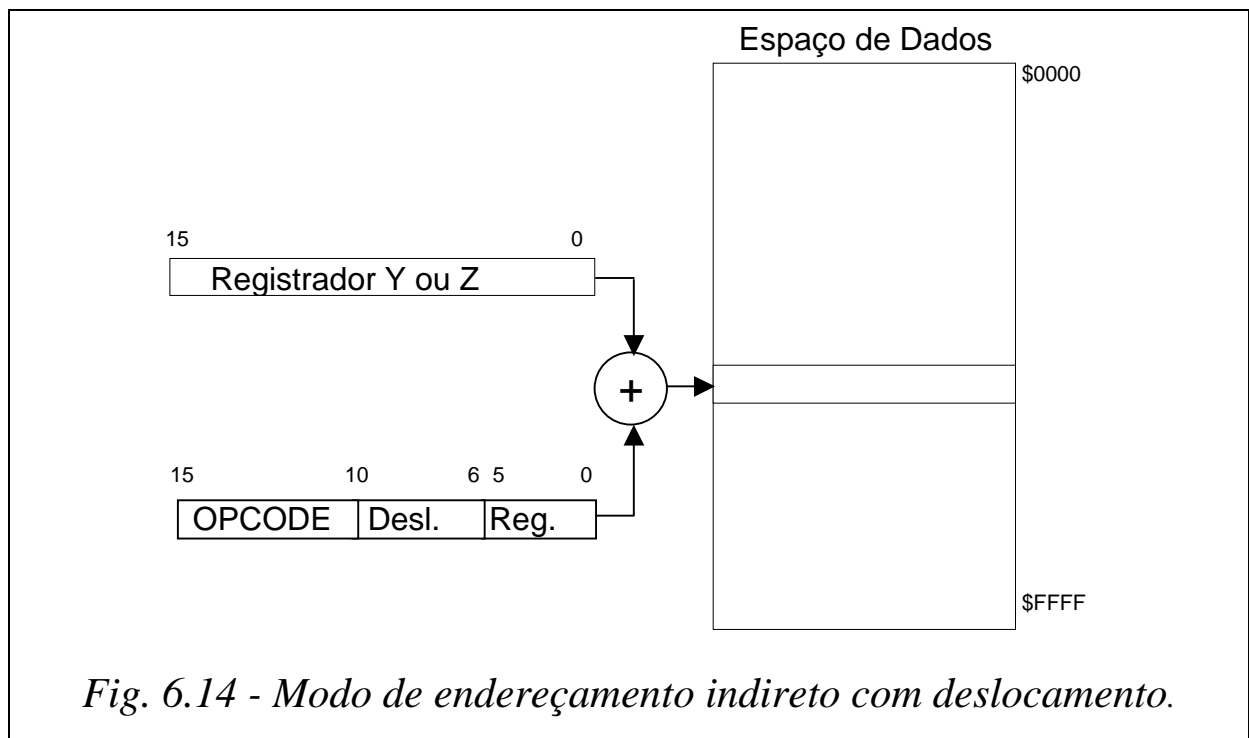
6.6.7 – Modo de endereçamento indireto com deslocamento

A instrução contém o deslocamento que será somado ao registrador Y ou Z para formar a posição onde se encontra o operando. A Fig. 6.14 ilustra este modo de endereçamento.

Exemplo: a instrução *LDR R1,Y+10* que carrega o dado que se encontra na posição de memória indicada por $Y+10$ no registrador R1. O Código de máquina dessa instrução é



Genericamente $0 < d < 31$ representa o registrador de destino, $0 < q < 63$ representa o deslocamento e n determina o registrador, Y para $n = 1$ e Z para $n = 0$.



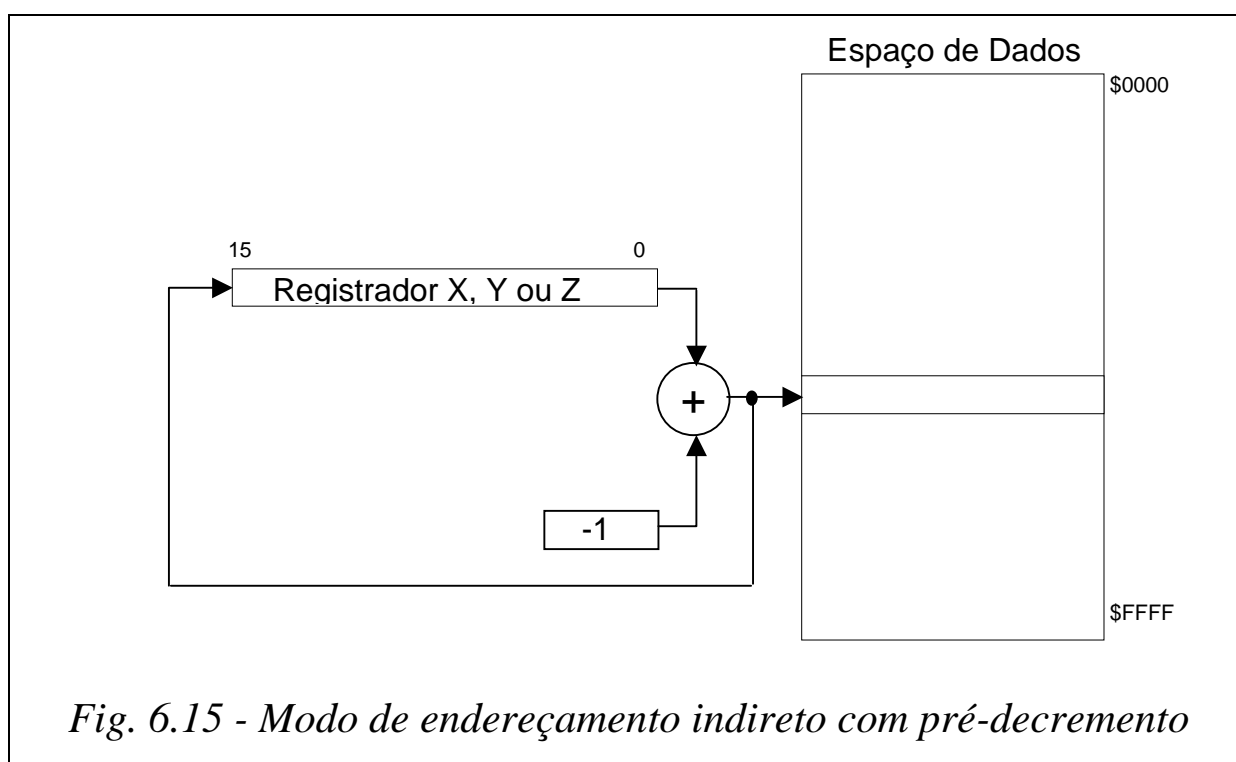
6.6.8 – Modo de endereçamento indireto com pré-decremento

A posição do operando é dada pelos registradores X,Y ou Z decrementados previamente de uma unidade. A Fig. 615 ilustra este modo de endereçamento.

Exemplo: a instrução *LD R15,-Y* que decrementa o registrador Y e carrega o dado que se encontra na posição de memória apontada por Y em R15. O Código de máquina dessa instrução é

15				d					d	d	d	d	n				0
1	0	0	1	0	0	0	0	1	1	1	1	1	1	0	1	0	

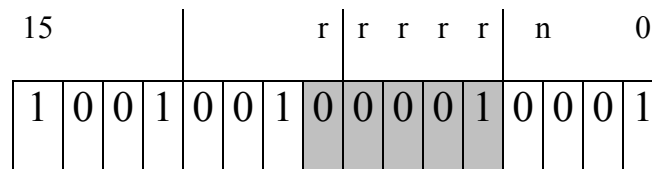
Genericamente $0 < d < 31$ representa o registrador de destino e n determina o registrador, X para n = 11b, Y para n = 10b e Z para n = 00b.



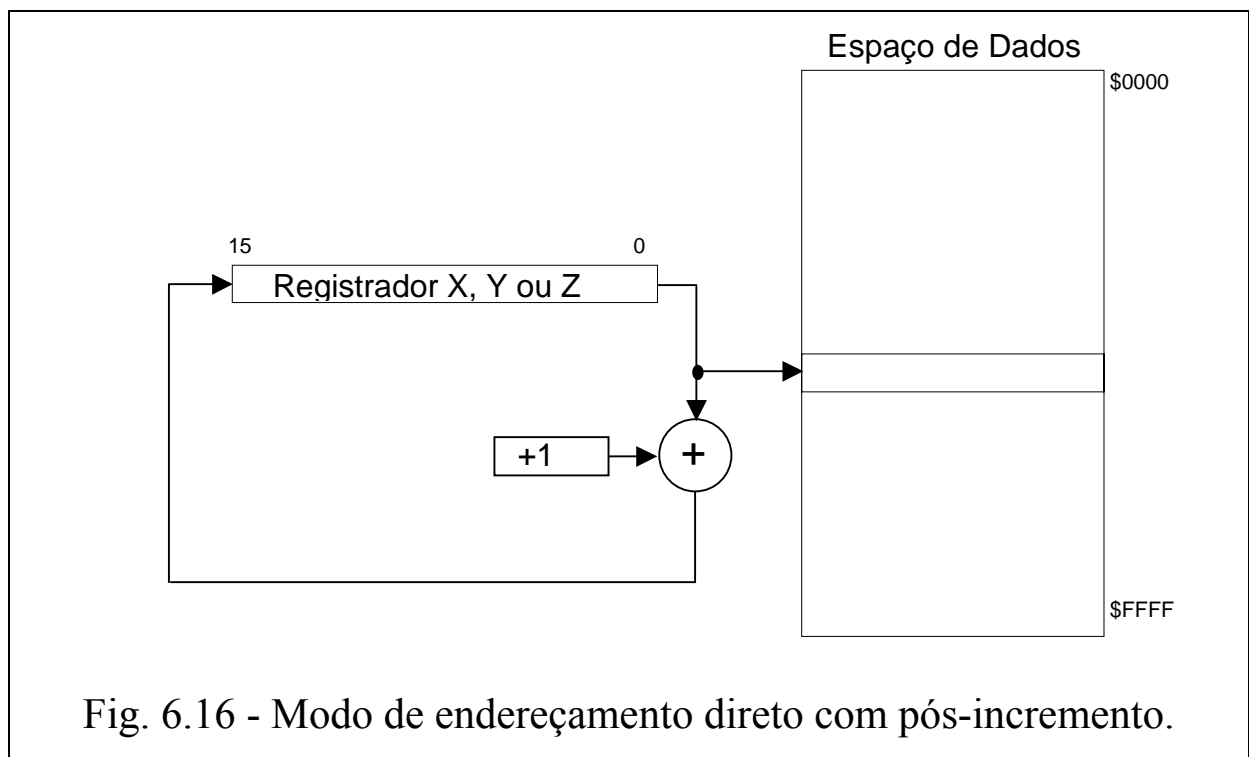
6.6.9 – Modo de endereçamento indireto com pós-incremento

A posição do operando é dada pelos registradores X,Y ou Z que no final da instrução são incrementados em uma unidade. A Fig. 6.16 ilustra este modo de endereçamento.

Exemplo: a instrução *ST Z+,R1* que armazena na posição de memória apontada pelo registrador Z o conteúdo de R1. Após, incrementa Z em uma unidade. O Código de máquina dessa instrução é



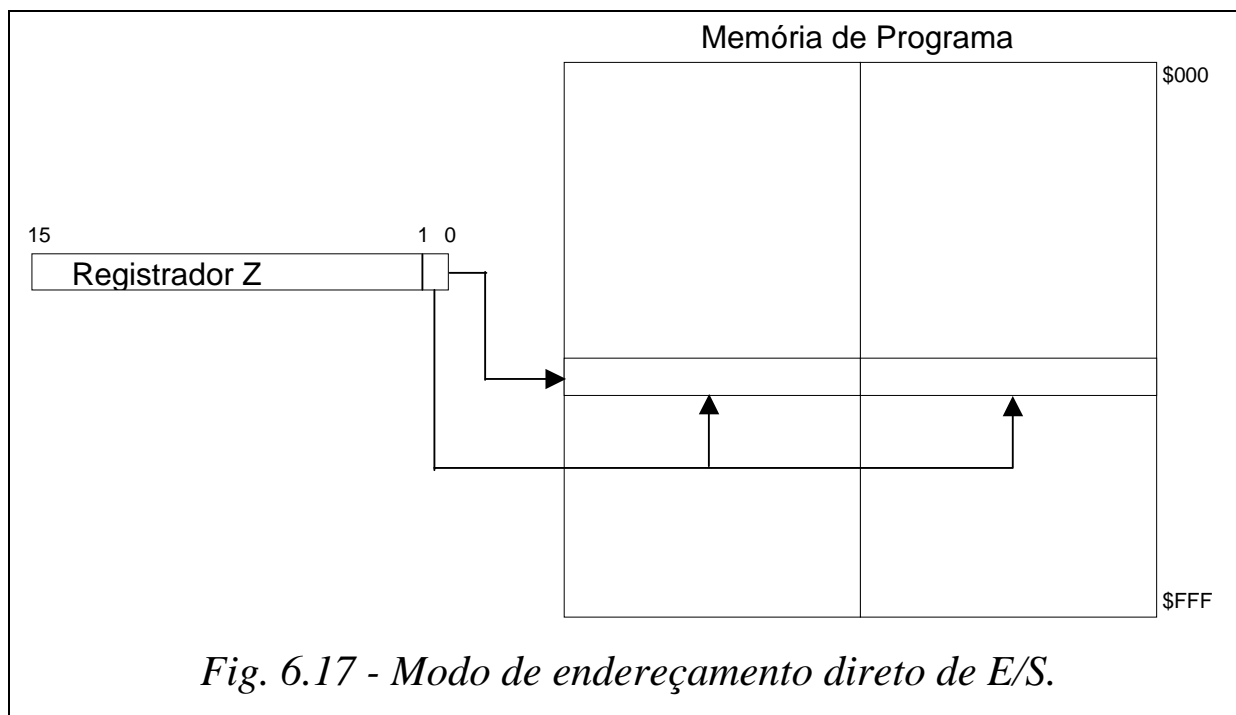
Genericamente $0 < r < 31$ representa o registrador fonte e n determina o registrador, X para n = 11b, Y para n = 10b e Z para n = 00b.



6.6.10 – Modo de endereçamento de constantes na memória de programa

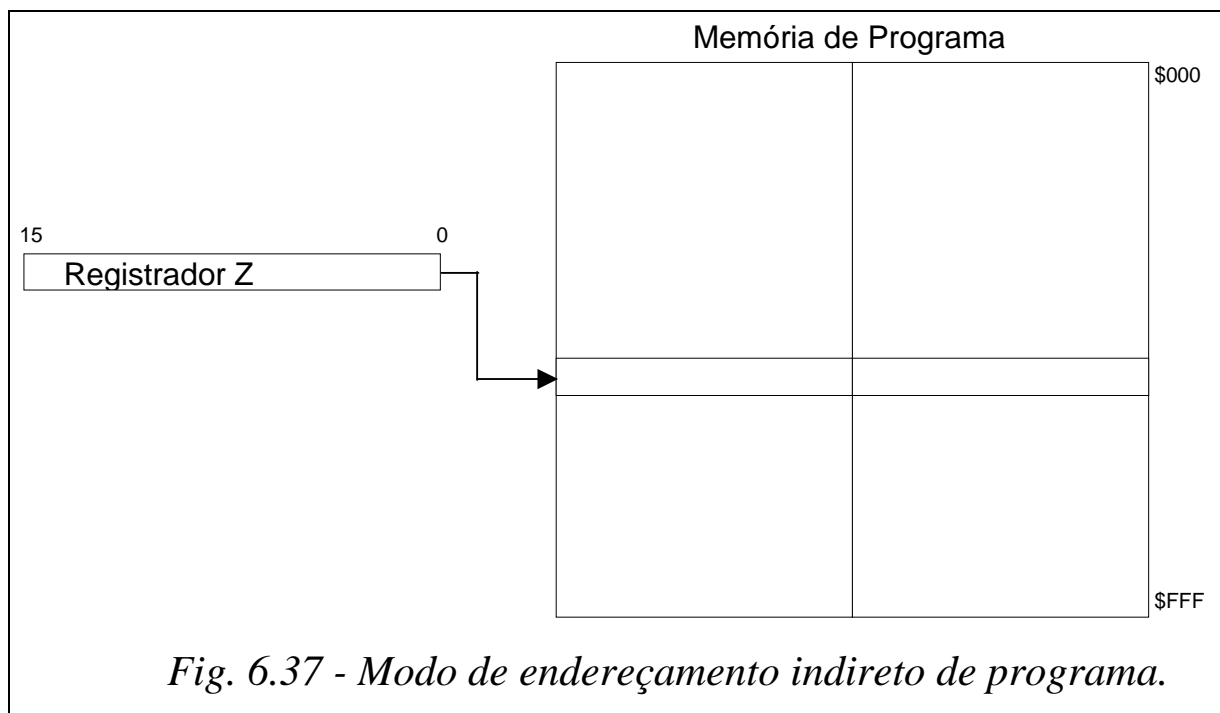
Como a memória de programa tem um tamanho de 16 bits e a memória de dados de 8 bits, o acesso a um de seus bytes requer um procedimento diferenciado. Os 15 bits mais significativos do

registrador Z especificam o endereço do dado (0 – 4k) e o bit 0 deste registrador é utilizado para seleccionar o byte alto (LSB = 1) e o byte baixo (LSB = 0) da constante. Com a execução da Instrução LPM o dado endereçado é copiado no registrador R0. A Fig. 6.17 ilustra este modo de endereçamento.



6.6.11 – Modo de endereçamento indireto de programa

As instruções IJUMP e ICALL permitem que a execução do programa continue na posição da memória de programa indicada pelo conteúdo do registrador Z, isto é, o PC é carregado com o conteúdo do registrador Z. A Fig. 6.18 ilustra este modo de endereçamento.



6.6.12 – Leitura e escrita na EEPROM

O tempo de acesso para uma operação de escrita na EEPROM está na faixa de 2,5 a 4 ms, dependendo do valor de V_{CC} . Entretanto, há uma função que permite que o software detecte quando o próximo byte pode ser escrito. Se o código de programa prevê acesso a EEPROM, algumas precauções devem ser seguidas. Fontes de alimentação fortemente filtradas atrasam a subida ou descida da tensão V_{CC} , fazendo com que o microcontrolador opere por um curto período de tempo com uma tensão mais baixa do que a especificada como mínima para a frequência de clock usada. A CPU operando sob estas condições pode executar desvios não intencionais e, eventualmente, executar instruções que acessam a EEPROM. Assim, para assegurar a integridade desta, na situação descrita, o usuário deve usar um circuito de reset externo para subtensões (detector *brown-out*).

Quando se lê ou se escreve na EEPROM, a CPU é interrompida por dois ciclos de relógio antes da próxima instrução ser executada. Os registradores para acesso a EEPROM, a saber, Registradores de Endereços da EEPROM EEARH e EEARL, Registrador de Dados da EEPROM (EEDR) e Registrador de Controle da EEPROM (EECR), são acessíveis no espaço de E/S. Eles são apresentados nas Figs. 6.19, 6.20 e 6.21.

Registradores EEARH, endereço \$1F(\$3F), e **EEARL** endereço \$1E(\$3E),
Valor na inicialização = 00000000b

bit	15	14	13	12	11	10	9	8
EEARH								EEAR8
EEARL	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
Bits 9 a 15	Reservados, sempre lidos como zero.							
EEAR8..0	Estes bits especificam o endereço da EEPROM no espaço de 512 bytes da mesma (\$000 a \$100).							

Fig. 6.19 – Registrador EEAR.

Registrador EEDR– endereço \$1D ; **Valor na inicialização** = 00000000b

	EEDR7	EEDR6	EEDR5	EEDR4	EEDR3	EEDR2	EEDR1	EEDR0
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
EEDR7..0	Estes bits contém o dado a ser escrito ou lido na EEPROM no endereço dado pelo registrador EEAR.							

Fig. 6.20 – Registrador EEDR.

Registrador EECR– endereço \$1C ; **Valor na inicialização** = 00000000b

	-	-	-	-	-	EEMWE	EEWE	EERE
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
Bits 7a 3	Reservados, sempre lidos como zero.							
EEMWE	Bit que permite, quando em 1 lógico, a escrita na EEPROM. Quando EEMWE está em 1 lógico, a ação de por em 1 lógico EEWE escreverá um dado no endereço selecionado da EEPROM. Quando EEMWE é posto em 1 lógico por software, o hardware leva-o para zero lógico após 4 ciclos de relógio.							
EEWE	<p>Bit de habilitação de escrita na EEPROM. Quando o endereço e os dados estão corretamente definidos, o bit EEWE deve ser posto em 1 lógico pra um valor ser escrito na EEPROM. Para ocorrer a escrita na EEPROM, o bit EEMWE deve estar em 1 lógico quando EEWE é posto em 1 lógico. O seguinte procedimento deverão ser seguidos quando da escrita na EEPROM:</p> <ol style="list-style-type: none"> 1. Aguardar até EEWE ser posto em zero lógico pelo hardware; 2. Escrever o endereço da EEPROM em EEARL e EEARH; 3. Escrever o dado da EEPROM em EEDR; 4. Escrever 1 lógico em EEMWE; 5. Dentro de 4 ciclos de relógio escrever 1 lógico em EEWE. <p>Quando o tempo de acesso transcorre, o bit EEWE é posto em 0 lógico pelo hardware. Assim, o software do usuário pode monitorar este bit e esperar por um zero antes de escrever o novo byte. Quando EEWE é posto em 1 lógico, a CPU espera por dois ciclos antes de executar a próxima instrução.</p>							
EERE	<p>Bit de habilitação de leitura na EEPROM. Quando o endereço correto é definido no registrador EEAR, o bit EERE deve ser posto em nível lógico 1. Quando este bit é posto em 0 lógico pelo hardware, o dado requisitado é encontrado no registrador EEDR. A leitura da EEPROM leva uma instrução e não há necessidade de monitorar o bit EERE. Quando o bit EERE é posto em 1 lógico, a CPU para por 4 ciclos antes de executar a próxima instrução.</p>							

Fig. 6.21 – Registrador EECR do AT90S8515.

O Código a seguir transfere o conteúdo de uma tabela da memória de programa para as primeiras posições da EEPROM.

```

; *****
; EXEMPLO DE UTILIZAÇÃO DA EEPROM E DE TABELAS EM
; MEMÓRIA. DE PROGRAMA
; *****
.include "8515def.inc"          ;Inclui todas as variáveis do AT90S8535
.def TEMP          =R17        ;Registro de armazenamento temporario
.def TEMP2         =R18        ;Registro de armazenamento temporario
;
; *****
;
; Definição dos Pinos de E/S
;
; *****
.cseg
.org 0x00                      ; RESET
                rjmp MAIN
; *****
;
; Interrupções
;
; *****
MAIN:
                ldi zh,high(tabela<<1)          ; posiciona z na tabela
                ldi zl,low(tabela<<1)
                ldi temp,$00                      ; inicializa temp
                mov r0,temp                      ; inicializa r0
                ldi temp2,$ff
pross:          out eearl,temp                    ; define 1a posição da EEPROM
                lpm                              ; código em r0
                out eedr,r0                      ; transfere código para eeprom
                adiw zh:zl,1                      ; z aponta proxima posição
                cp temp2,r0                      ; fim da tabela?
                breq fica                        ; se sim para execução
aguarda:        sbic eecr,eewe                   ; se não aguarda para gravar
                rjmp aguarda
                sbi eecr,eemwe                   ; seta master
                sbi eecr,eewe                    ; grava
                inc temp                         ; proxima posição da EEPROM
                rjmp pross                      ; continua
fica:           rjmp fica                      ; parada
;
tabela: .dw 0x0201,0x0403,0x0605,0xFF07

```

6.7 – Estrutura e operação das portas de E/S:

Todas as portas apresentam uma real funcionalidade lê-modifica-escreve quando usadas como portas de E/S. Isto significa que com o emprego das instruções *SBI* (seta bit do registrador de E/S) e *CBI* (limpa bit do registrador de E/S) a direção de um único pino da porta pode ser alterada sem alterar, inadvertidamente, qualquer outro. O mesmo se aplica a mudança da configuração do driver (se configurado como saída) ou a habilitação/desabilitação dos resistores de *pull-up* (se configurado como entrada).

O AT90S8515 apresenta 32 terminais bidirecionais de E/S agrupados nas portas A, B, C e D. Três posições da memória de E/S são associadas a cada uma delas, conforme indica a Tabela 6.4. O endereço dos pinos de entrada das portas ($PINX^{12}$) é somente de leitura, enquanto que os registradores de dados (*PORTX*) e de direção dos dados (*DDRX*) são de leitura/escrita. A Fig. 6.22 apresenta o esquema geral da arquitetura interna das linhas de E/S do AT90S8515. Cada bit dos registradores *PORTX* é representado como um flip-flop tipo D, o qual armazena um valor do barramento de dados em resposta a um sinal de escrita no *latch* (*WP* -write port) vindo da CPU. A saída Q do flip-flop é colocada no barramento em resposta a um sinal de leitura do *latch* (*RL* - read port latch) vindo da CPU, enquanto que o nível lógico do pino da porta é colocado no barramento em resposta a um sinal de leitura do pino (*RP* – read port pin).

¹² X – A, B, C ou D.

Todos os pinos das portas têm resistores de *pull-up* selecionáveis individualmente. Os *buffers* das portas podem drenar até 20 mA, o que permite o acionamento direto de LEDs. Quando os pinos das portas são usados como entradas e são externamente conectados a referência, eles fornecerão corrente se os resistores internos de *pull-up* estão ativados.

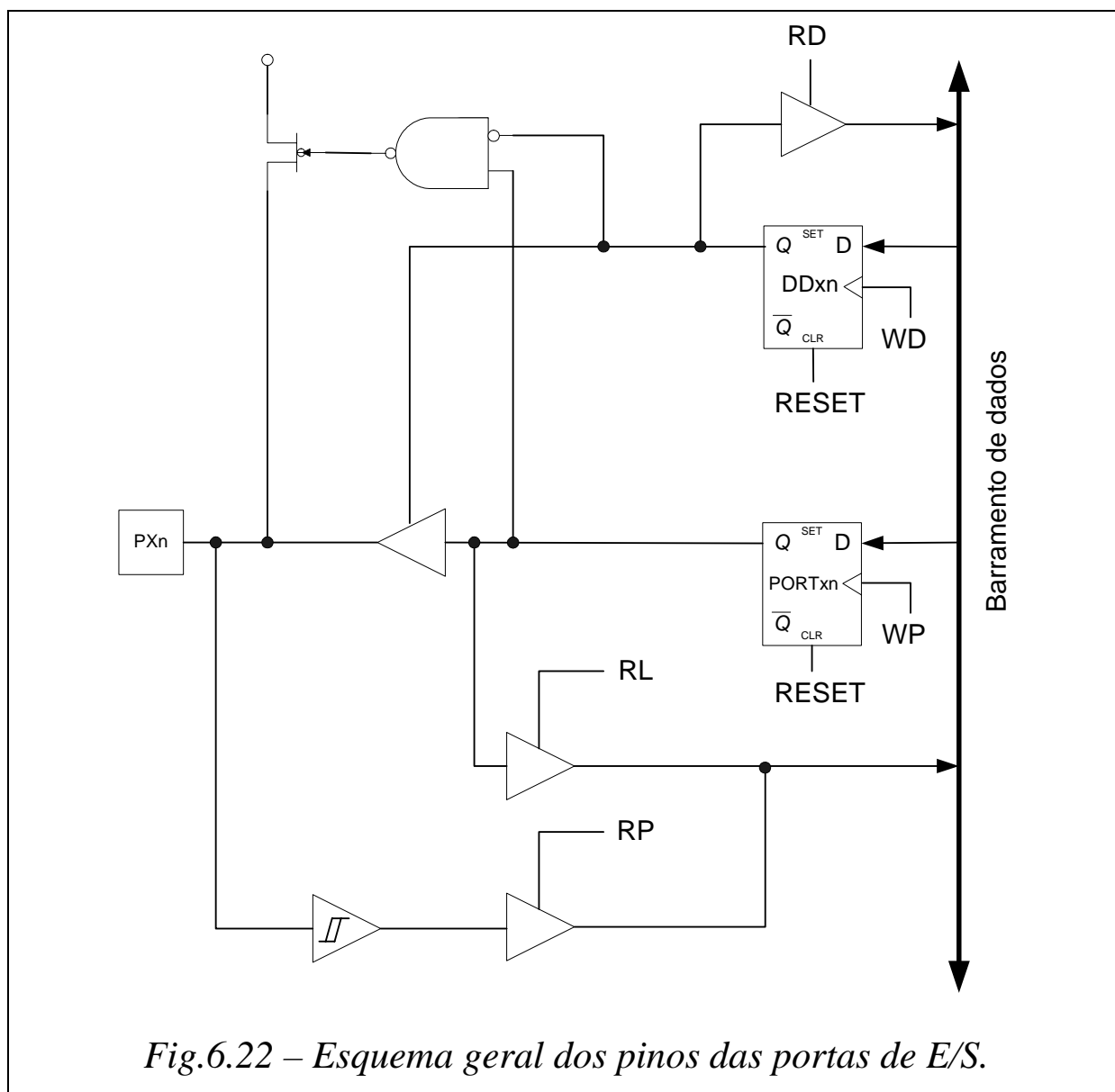


TABELA 6.4
REGISTRADORES ASSOCIADOS ÀS PORTAS E/S

Registrador de Dados da Porta A (PORTA) – Endereço: \$1B; Valor na inicialização: 00000000b								
Bit	7	6	5	4	3	2	1	0
Nome	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
Funcionalidade	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Registrador de Direção de Dados da Porta A (DDRA) – Endereço: \$1A; Valor na inicialização: 00000000b								
Bit	7	6	5	4	3	2	1	0
Nome	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
Funcionalidade	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Endereço do Pino de Entrada da Porta A (PINA) – Endereço: \$19								
Bit	7	6	5	4	3	2	1	0
Nome	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
Funcionalidade	R	R	R	R	R	R	R	R
Registrador de Dados da Porta B (PORTB) – Endereço: \$18; Valor na inicialização: 00000000b								
Bit	7	6	5	4	3	2	1	0
Nome	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
Funcionalidade	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Registrador de Direção de Dados da Porta B (DDRB) – Endereço: \$17; Valor na inicialização: 00000000b								
Bit	7	6	5	4	3	2	1	0
Nome	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
Funcionalidade	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Endereço do Pino de Entrada da Porta B (PINB) – Endereço: \$16								
Bit	7	6	5	4	3	2	1	0
Nome	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
Funcionalidade	R	R	R	R	R	R	R	R

TABELA 6.4
REGISTRADORES ASSOCIADOS ÀS PORTAS E/S

Registrador de Dados da Porta C (PORTC) – Endereço: \$15; Valor na inicialização: 00000000b								
Bit	7	6	5	4	3	2	1	0
Nome	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
Funcionalidade	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Registrador de Direção de Dados da Porta B (DDRC) – Endereço: \$14; Valor na inicialização: 00000000b								
Bit	7	6	5	4	3	2	1	0
Nome	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
Funcionalidade	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Endereço do Pino de Entrada da Porta C (PINC)– Endereço: \$13								
Bit	7	6	5	4	3	2	1	0
Nome	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
Funcionalidade	R	R	R	R	R	R	R	R
Registrador de Dados da Porta D (PORTD) – Endereço: \$12; Valor na inicialização: 00000000b								
Bit	7	6	5	4	3	2	1	0
Nome	PORTD6	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
Funcionalidade	-	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Registrador de Direção de Dados da Porta D (DDRD) – Endereço: \$11; Valor na inicialização: 00000000b								
Bit	7	6	5	4	3	2	1	0
Nome	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
Funcionalidade	-	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Endereço do Pino de Entrada da Porta D (PIND)– Endereço: \$10								
Bit	7	6	5	4	3	2	1	0
Nome	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
Funcionalidade	-	R	R	R	R	R	R	R

Os bits $DDXn^{13}$ dos registradores $DDRX$ selecionam a direção dos pinos de E/S, se DDx é colocado em nível lógico 1, o pino (PXn) é configurado como uma saída, se colocado em 0 lógico, o pino é configurado como entrada. Se $PORTXn$ é colocado em 1 lógico e o pino está configurado como um pino de entrada, o resistor MOS de pull-up está ativado. Para desativar o resistor de pull-up, $PORTXn$ tem que ser posto em nível lógico zero ou o pino tem ser configurado como um pino de saída. Os pinos das portas são colocados em *tri-state* quando ocorre um reset, mesmo que o relógio não esteja ativo. A Tabela 6.5 resume a discussão deste parágrafo.

Os pinos das portas podem desempenhar funções alternativas, conforme mostra a Tabela 6.6.

TABELA 6.5
CONFIGURAÇÃO DAS PORTAS DE E/S

DDXn	PORTXn	E/S	Pull-up	Condição
0	0	Entrada	Não	Tri-state
0	1	Entrada	Sim	O pino pode fornecer corrente
1	0	Saída	Não	Saída em 0 lógico
1	1	Saída	Não	Saída em 1 lógico

Nota: n (número do pino): 0, 1, ...7 para porta B e 0, 1, ...6 para porta D.

¹³ n: 0,1, ..7.

TABELA 6.6
FUNÇÕES ALTERNATIVAS DOS PINOS DAS PORTAS DO
AT90S8515

Pino	Função alternativa
PB0	T0 – Entrada externa do contador do Temporizador/Contador 0
PB1	T1 – Entrada externa do contador do Temporizador/Contador 1
PB2	AIN0 – Entrada não inversora do comparador analógico
PB3	AIN1 – Entrada inversora do comparador analógico
PB4	\overline{SS} – Entrada de seleção de escravo para o canal SPI
PB5	MOSI – Saída de dados do mestre / entrada de dados do escravo para o canal SPI
PB6	MISO – Entrada de dados do mestre / saída de dados do escravo para o canal SPI
PB4	SCK – Saída de relógio do mestre / entrada de relógio do escravo para o canal SPI
PD0	RXD – Entrada de dados da UART
PD1	TXD – Saída de dados da UART
PD2	INT0 – Entrada de interrupção externa 0
PD3	INT1 – Entrada de interrupção externa 1
PD5	OC1A – Saída da saída de Coincidência de Comparação A do Temporizador/Contador 1
PD6	\overline{WR} – Habilitação de escrita na memória externa
PD7	\overline{RD} – Habilitação de leitura na memória externa

6.8 – O conjunto de instruções do AT90S8515

6.8.1 – O Registrador de Estado (*Status Register* - SREG)

O registrador de estado SREG, apresentado na Fig. 6.23 contém bits que refletem a situação atual da CPU. Ele é composto pelos seguintes bits: os bits I de habilitação global de interrupções e T

de armazenamento/cópia, os flags de carry e half-carry, o bit de sinal e os flags de *overflow* no complemento de dois, de negativo e o de zero.

Registrador SREG – endereço \$3F ; **Valor na inicialização** = 00000000b

	I	T	H	S	V	N	Z	C
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
I	O Bit de habilitação global de interrupções deve ser colocado em 1 lógico para que as interrupções sejam habilitadas. O controle de habilitação individual de interrupções é realizado em registradores de controle separados. Se o bit de habilitação global de interrupções está em nível lógico zero, nenhuma interrupção está habilitada, independente do valor dos bits individuais de habilitação. O bit I é colocado em 0 lógico pelo hardware após o atendimento de uma interrupção e colocado em 1 lógico pela instrução <i>RETI</i> para habilitar futuras interrupções.							
T	As instruções de cópia de bit <i>BLD</i> (Bit Load) e <i>BST</i> (Bit Store) usam o bit T como fonte e destino em uma manipulação de bit. Um bit de um registrador pode ser copiado para o bit T pela instrução <i>BST</i> e o bit presente em T pode ser copiado para o bit de um registrador por <i>BLD</i> .							
H	O flag half-carry recebe o “vai um” do bit 3 do resultado das operações aritméticas.							
S	O flag de sinal S é sempre o resultado de um ou-exclusivo entre o flag de negativo e o de <i>overflow</i> no complemento de dois, ou seja, $S = N \oplus V$.							
V	O flag de overflow no complemento de dois destina-se a aritmética de complemento de dois.							
N	O Flag de negativo indica um resultado negativo após uma operação aritmética ou lógica.							
Z	O Flag de zero indica um resultado igual a zero após uma operação aritmética ou lógica.							
C	O Flag de carry recebe o “vai um” / “empresta um” resultante das operações aritméticas. É usado, também, em operações lógicas.							

Fig. 6.23 – Registrador SREG dos AVR.

6.8.2 – Instruções lógicas e aritméticas

Poucas instruções lógicas e aritméticas aceitam valores imediatos (SBCI, SUBI, CPI, ANDI e ORI) e as que aceitam trabalham apenas com a metade superior dos registradores (R16 – R31) do conjunto. A Tabela 6.7 mostra a lista completa de instruções lógicas e aritméticas.

TABELA 6.7
INSTRUÇÕES LÓGICAS E ARITMÉTICAS

Mnemônico	Operandos	Descrição
ADD	Rd,Rr	Soma dois registradores
ADC	Rd,Rr	Soma dois registradores com carry
ADIW	Rdh:Rdl,k	Soma uma palavra ao par de registradores
SUB	Rd,Rr	Subtrai dois registradores
SUBI	Rd,K	Subtrai uma constante do registrador
SBC	Rd,Rr	Subtrai dois registradores com carry
SBCI	Rd,K	Subtrai uma constante do registrador com carry
SBIW	Rdh:Rdl,k	Subtrai uma palavra do par de registradores
AND	Rd,Rr	Executa o “e” lógico entre dois registradores
ANDI	Rd,K	Executa o “e” lógico entre uma constante e um registrador
OR	Rd,Rr	Executa o “ou” lógico entre dois registradores
ORI	Rd,K	Executa o “ou” lógico entre uma constante e um registrador
EOR	Rd,Rr	Executa o “ou-exclusivo” entre dois registradores
COM	Rd	Faz o complemento de um do registrador
NEG	Rd	Faz o complemento de dois do registrador
SBR	Rd,K	Põe em 1 lógico bit(s) do registrador
CBR	Rd,K	Põe em 0 lógico bit(s) do registrador
INC	Rd	Soma um ao registrador
DEC	Rd	Diminui um do registrador
TST	Rd	Teste de zero ou negativo
CLR	Rd	Põe em 0 lógico todos os bits do registrador
SER	Rd	Põe em 1 lógico todos os bits do registrador

As instruções `ADIW Rdh:Rdl,k` e `SBIW Rdh:Rdl,k`, que operam, respectivamente, a soma e a subtração de uma palavra ao par de registradores indicados como destino. As instruções são válidas apenas para os quatros pares mais altos de registradores, ou seja, R25:R24, R27:R26, R29:R28 e R31:R30.

6.8.3 – Instruções de desvio

O AT90S8515 apresenta um rico conjunto de instruções de desvio condicional; para cada um dos oito *flags* do Registrador de Estado (SREG), ele apresenta duas opções. Com apenas 7 bits de offset, estas instruções podem desviar a execução apenas 64 instruções em ambas as direções. Para obter desvios relativos maiores, empregase a instrução `RJMP` que pode deslocar a execução em até 2 kbytes, o que é geralmente suficiente.

O trecho de programa a seguir, utiliza uma instrução de desvio para gerar um atraso de execução proporcional ao valor de R1.

delay:

```
DEC R1      ; R1 = R1 - 1
```

```
BRNE delay      ; enquanto R1 > 0,  
                  desvia para delay
```

No exemplo acima, enquanto o registrador R1 for diferente de zero, a execução do programa é desviada para a instrução rotulada por “delay”, gerando um laço de atraso. A instrução *BRNE* (*Branch if Not Equal*) testa o estado do flag Z, executando o desvio sempre que o

flag Z estiver em 0 lógico; quando o flag Z ir para nível lógico 1, condição que no nosso exemplo se dará quando R1 tornar-se igual a zero, a execução do programa prossegue para a instrução que segue *BRNE*. A instrução oposta a *BRNE* é *BREQ* (*Branch if EQual*). Além do flag zero, outros bits podem ser utilizados para criar desvios.

As instruções de desvio indireto *IJMP* e de chamada indireta de sub-rotina *ICALL* utilizam como ponteiro o registrador Z, o que permite desvios e chamadas de sub-rotina dentro dos 64k mais baixos da memória.

A Tabela 6.8 mostra a lista completa de instruções de desvio.

6.8.4 – Instruções de transferência dados

A Tabela 6.9 apresenta o conjunto de instruções disponíveis para mover dados dentro do espaço de memória do AT90S8515.

Por exemplo, a seqüência

```
LDI R16,150
```

```
MOV R15,R16
```

escreve o número 150 no registrador R16 e copia o conteúdo deste em R15. Observe que o primeiro registro é sempre o destino!

TABELA 6.8
INSTRUÇÕES DE DESVIO

Mnemônico	Operandos	Descrição
RJMP	k	Desvio relativo
RCALL	k	Chamada relativa de sub-rotina
RET		Retorno de sub-rotina
RETI		Retorno de interrupção
CPSE	Rd,Rr	Compara e pula se igual
CP	Rd,Rr	Compara
CPC	Rd,Rr	Compara com carry
CPI	Rd,K	Compara registrador com imediato
SBRC	Rr,b	Pula se bit no registrador é 0
SBRB	Rr,b	Pula se bit no registrador é 1
SBIC	P,b	Pula se bit em registrador de E/S é 0
SBIS	P,b	Pula se bit em registrador de E/S é 1
BRBS	s,k	Desvia se o flag de estado é 1
BRBC	s,k	Desvia se o flag de estado é 0
BREQ	k	Desvia se igual
BRNE	k	Desvia se diferente
BRCS	k	Desvia se o carry é 1
BRCC	k	Desvia se o carry é 0
BRSH	k	Desvia se maior ou igual
BRLO	k	Desvia se menor
BRMI	k	Desvia se negativo
BRPL	k	Desvia se positivo
BRGE	k	Desvia se maior ou igual, com flag de sinal
BRLT	k	Desvia menor, com flag de sinal
BRHS	k	Desvia se o half-carry é 1
BRHC	k	Desvia se o half-carry é 0
BRTS	k	Desvia se o flag T é 1
BRTC	k	Desvia se o flag T é 0
BRVS	k	Desvia se o flag de overflow é 1
BRVC	k	Desvia se o flag de overflow é 0
BRIE	k	Desvia se a interrupção está habilitada
BRID	k	Desvia se a interrupção está desabilitada

TABELA 6.9

INSTRUÇÕES DE TRANSFERÊNCIA DE DADOS DO AT90S8515

Mnemônico	Operandos	Descrição
MOV	Rd,Rr	Transferência de dados entre registradores
LDI	Rd,K	Carrega dado imediato no registrador
LD	Rd,X	Carrega registrador com o conteúdo da posição de memória apontada por X
LD	Rd,X+	Carrega registrador com o conteúdo da posição de memória apontada por X e incrementa X
LD	Rd,-X	Decrementa X e carrega registrador com o conteúdo da posição de memória apontada por X
LD	Rd,Y	Carrega registrador com o conteúdo da posição de memória apontada por Y
LD	Rd,Y+	Carrega registrador com o conteúdo da posição de memória apontada por Y e incrementa Y
LD	Rd,-Y	Decrementa Y e carrega registrador com o conteúdo da posição de memória apontada por Y
LDD	Rd,Y+q	Carrega registrador com o conteúdo da posição de memória apontada por Y+q
LD	Rd,Z	Carrega registrador com o conteúdo da posição de memória apontada por Z
LD	Rd,Z+	Carrega registrador com o conteúdo da posição de memória apontada por Z e incrementa Z
LD	Rd,-Z	Decrementa Z e carrega registrador com o conteúdo da posição de memória apontada por Z
LDD	Rd,Z+q	Carrega registrador com o conteúdo da posição de memória apontada por Z+q
LDS	Rd,k	Carrega registrador com o conteúdo da posição de memória k da SRAM
ST	X,Rr	Carrega na posição de memória apontada por X o valor de Rr
ST	X+,Rr	Carrega na posição de memória apontada por X o valor de Rr e incrementa X
ST	-X,Rr	Decrementa X e carrega na posição de memória apontada por X o valor de Rr
ST	Y,Rr	Carrega na posição de memória apontada por Y o valor de Rr
ST	Y+,Rr	Carrega na posição de memória apontada por Y o valor de Rr e incrementa Y
ST	-Y,Rr	Decrementa Y e carrega na posição de memória apontada por Y o valor de Rr

TABELA 6.9

INSTRUÇÕES DE TRANSFERÊNCIA DE DADOS DO AT90S8515

Mnemônico	Operandos	Descrição
STD	Y+q,Rr	Carrega na posição de memória apontada por Y+q o valor de Rr
ST	Z,Rr	Carrega na posição de memória apontada por Z o valor de Rr
ST	Z+,Rr	Carrega na posição de memória apontada por Z o valor de Rr e incrementa Z
ST	-Z,Rr	Decrementa Z e carrega na posição de memória apontada por Z o valor de Rr
STD	Z+q,Rr	Carrega na posição de memória apontada por Z+q o valor de Rr
STS	K,Rr	Carrega na posição de memória k da SRAM o valor de Rr
LPM		Carrega em R0 o conteúdo da posição de memória de programa apontada por Z
IN	Rd,P	Entrada de dados na porta
OUT	P,Rd	Saída de dados na porta
PUSH	Rr	Salva registrador na pilha e decrementa SP
POP	Rd	Carrega registrador com byte da pilha e incrementa SP

6.8.5 – Instruções de manipulação e teste de bits

A Tabela 6.10 apresenta o conjunto de instruções disponíveis para manipular e testar bits no AT90S8515. As instruções SBI P , b e CBI P , b trabalham apenas com os 32 primeiros registradores de E/S, ou seja, com os endereços \$0 a \$31.

TABELA 6.10
INSTRUÇÕES DE MANIPULAÇÃO E TESTE DE BITS

Mnemônico	Operandos	Descrição
SBI	P,b	Coloca em 1 lógico o bit b do registrador de E/S
CBI	P,b	Coloca em 0 lógico o bit b do registrador de E/S
LSL	Rd	Deslocamento lógico à esquerda
LSR	Rd	Deslocamento lógico à direita
ROL	Rd	Rotação à esquerda com carry
ROR	Rd	Rotação à direita com carry
ASR	Rd	Deslocamento aritmético à direita
SWAP	Rd	comuta nibbles do registrador
BSET	s	Coloca em 1 lógico o bit s do registrador de estado
BCLR	s	Coloca em 0 lógico o bit s do registrador de estado
BST	Rr,b	Armazena bit b do registrador em T
BLD	Rd,b	Carrega T no bit b do registrador
SEC		Coloca em 1 lógico o carry
CLC		Coloca em 0 lógico o carry
SEN		Coloca em 1 lógico o flag negativo
CLN		Coloca em 0 lógico o flag negativo
SEZ		Coloca em 1 lógico o flag zero
CLZ		Coloca em 0 lógico o flag zero
SEI		Habilita interrupção global
CLI		Desabilita interrupção global
SES		Coloca em 1 lógico o flag de sinal
CLS		Coloca em 0 lógico o flag de sinal
SEV		Coloca em 1 lógico o flag complemento de 2
CLV		Coloca em 0 lógico o flag complemento de 2
SET		Coloca em 1 lógico o bit T em SREG
CLT		Coloca em 0 lógico o bit T em SREG
SEH		Coloca em 1 lógico o half-carry
CLH		Coloca em 0 lógico o half-carry
NOP		Nenhuma operação
SLEEP		Dormir
WDR		Inicializa o Cão de Guarda

No AT90S8515 para realizar uma multiplicação por dois emprega-se a instrução `LSL Rd`. Esta instrução desloca todos os bits de um byte para a esquerda e escreve um zero no bit menos significativo, operação que é chamada de deslocamento lógico a esquerda. O bit mais significativo do byte será deslocado para o carry no registrador de estado `SREG`. A divisão por 2, por sua vez, é realizada pela instrução `LSR Rd`. Aqui o bit mais significativo é deslocado para o bit 6 e é preenchido com zero, enquanto o bit menos significativo é deslocado para o carry. O carry pode ser usado para arredondar o resultado (se em 1, adiciona um ao resultado). Por exemplo, a rotina abaixo é uma divisão por quatro com arredondamento:

```
LSR R1; divisão por 2

BRCC div2; desvia se C = 0

INC R1; arredonda

div2: LSR R1; mais uma divisão por 2

BRCC segue; desvia se C = 0

INC R1; arredonda

segue:
```

Se números inteiros com sinal são usados, o deslocamento lógico a direita sobrescreveria o bit de sinal no bit 7. A instrução de deslocamento aritmético a direita deixa o bit 7 intocado e desloca apenas os outros bits, inserindo um zero no bit 6. Com `ASR`, o bit 0 vai para o carry em `SREG`.

A seguinte seqüência multiplica por dois uma palavra de 16 bits:

```
LSL R1; deslocamento lógico à esquerda do  
byte menos significativo  
ROL R2; rotação à esquerda do byte mais  
significativo
```

O deslocamento lógico à esquerda na primeira instrução desloca o bit 7 para o carry, a instrução ROL rola-o para o bit zero do byte mais significativo. Após a segunda instrução o carry contém o último bit 7, podendo ser utilizado para indicar um estouro (se foi realizada um cálculo de 16 bits) ou ser rolado para um byte mais significativo (se for realizado um cálculo com mais de 16 bits).

O bit T pode ser colocado em 1 ou 0 lógico e o seu conteúdo pode ser copiado para qualquer bit de qualquer registrador:

```
SET; T = 1  
BST R2,2; copia o bit T para o bit 2 do  
registrador R2
```

Uma apresentação mais completa do Conjunto de Instruções pode ser encontrado em ATMEL (2002a).

6.9 – Como criar um programa em assembly para o AVR

Nesta seção, criaremos programas em assembly empregando a ferramenta de desenvolvimento gratuita AVR Studio da ATMEL. Ao iniciar o AVR Studio, aparece a janela “Welcome to AVR Studio”. Nessa janela, podemos criar um novo projeto ou abrir um

existente. Selecionando a primeira opção, aparecerá uma nova janela que contém uma caixa de texto onde nomearemos o projeto. A título de exemplo, criaremos um pequeno projeto chamado “ondaquad”. Após esta ação, selecionaremos o botão “Next >>”. Na janela que se abre, selecionaremos “AVR Simulator” e o dispositivo AT90S8515. Após, selecionamos “Finish”; na sequência, aparecerá uma janela onde poderemos editar o nosso código fonte em assembly.

A criação de um programa em assembly depende do conhecimento da arquitetura interna do microcontrolador. Na estruturação do programa, é necessário incluir um arquivo que contém todas as informações do AT90S8515 utilizando a diretiva `include`. Ao incluir este arquivo no código fonte em assembly, o programador pode usar todos os nomes dos registradores de E/S e dos bits destes registradores que aparecem na folha de dados. É necessário, também, lembrar que os primeiros endereços da memória de programa são reservados para os vetores de interrupção. Assim, o nosso programa começaria da seguinte forma:

```
.include "8515def.inc" ;Inclui todas as
                        ; variáveis do AT90S8515
.cseg
    rjmp MAIN
; espaço reservado p/ os vetores de interrupção
MAIN:
```

Nosso primeiro programa gerará uma onda quadrada de frequência 5 kHz ($T/2 = 100 \mu\text{s}$) na porta PDO de um AT90S8515 operando em 4 MHz ($T = 250 \text{ ns}$). Assim, será necessário configurar

este como pino como saída e inverter seu estado a cada 100 μ s. A sequência necessária de comandos é:

```
MAIN: ldi r16,0b0000001    ;carrega r16 com
                             ; 0x01
      out DDRD,r16          ;move o valor 0x01
                             ; para DDRD
                             ;configurando PD0
                             ; como saída
```

Para gerar a onda quadrada, seguiremos, a partir de agora, o fluxograma da Fig. 3.12, o que resultará no seguinte código:

```
repete:com r16
      out PORTD,r16          ;complementa PD0
      ldi r17,132            ;carrega r17 com 132
delay:
      dec r18                ;R17 = R17 - 1
      brne delay             ;enquanto R17>0, desvia
                             ;para delay
      rjmp repetec           ;desvia para repetec
```

Dentro do laço externo (repete) as três primeiras instruções são executadas, cada uma, em um período de relógio, totalizando 750 ns. As instruções `dec r18` e `brne delay` são executadas 132 vezes, sendo que a primeira é executada em um ciclo de relógio e a última em dois ciclos enquanto ocorrer o desvio (131 vezes) e em um ciclo quando não ocorrer o desvio, totalizando 98,75 μ s. Por sua vez, a última instrução é executada em 2 ciclos de relógio, ou seja, 500 ns. Assim o laço `repete` é executado em exatos 100 μ s, criando no pino PD0 uma onda quadrada de 5 kHz.

A ausência de instruções que complementam apenas um bit das portas de saída nos microcontroladores AVR e a estrutura de

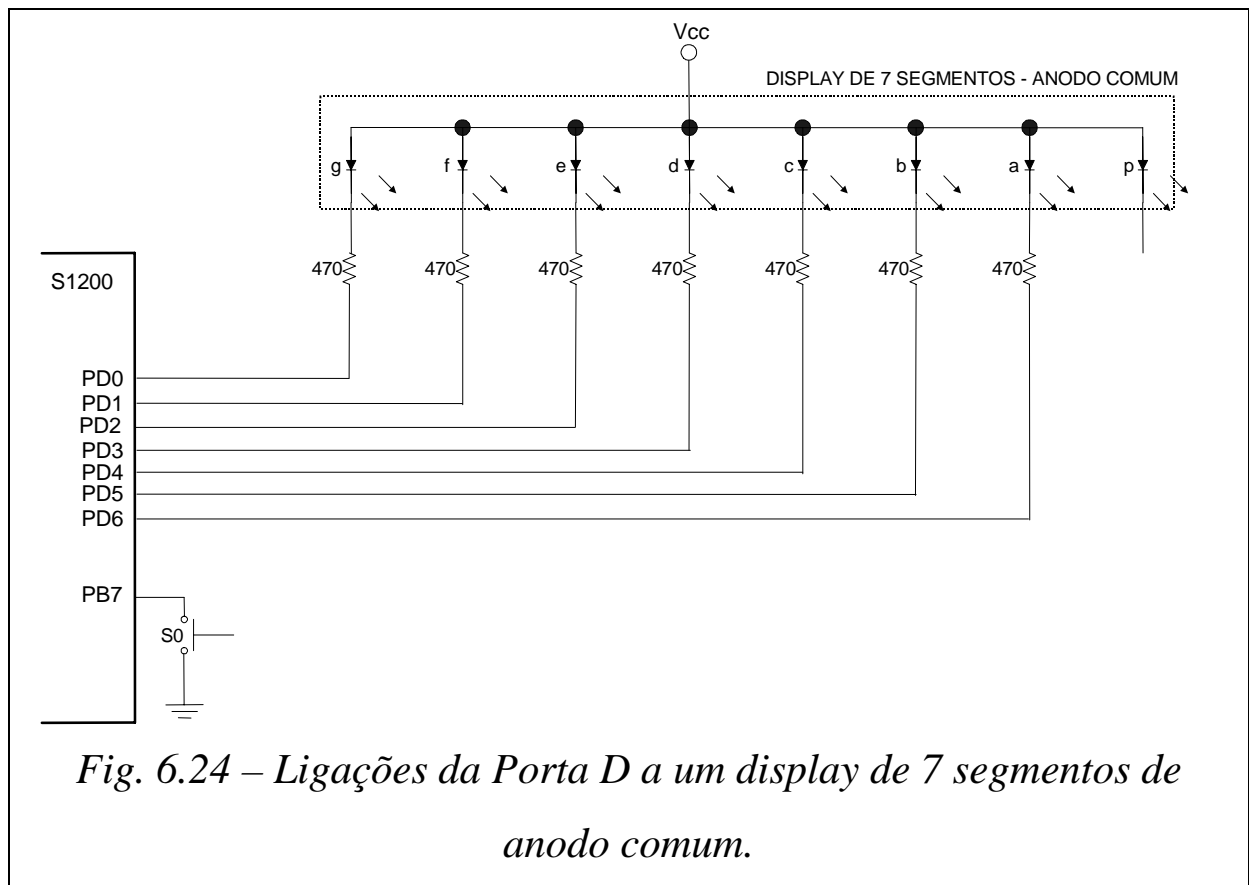
programação proposta pelo fluxograma da Fig. 3.12 geraram um programa que, na realidade, afeta todos os bits da Porta D. A seqüência a seguir aperfeiçoa o programa anterior (construa seu fluxograma), afetando apenas o bit PORTD0.

```
.include "8515def.inc" ;Inclui todas as
                        ;variáveis do AT90S8515
.def TEMP              =R16      ;Reg. de armazenamento
                        ; temporario
.def CONTA             =R17      ;Registro de contagem
.cseg
    rjmp MAIN
; espaço reservado p/ os vetores de interrupção
MAIN: ldi TEMP,0b0000001        ;carrega TEMP com
                                ; 0x01
    out DDRD,TEMP              ;move o valor 0x01
                                ; para DDRD
                                ;configurando PD0
                                ; como saída
repete:ldi CONTA,132            ; CONTA = 132
delay:
    dec CONTA                  ;CONTA = CONTA - 1
    brne delay                 ;enquanto CONTA > 0,
                                ; desvia para delay
    sbis PORTD,PD0             ;salta 1 instrução
                                ; se PD0=1
    rjmp seta                  ;desvia para seta
    cbi PORTD,PD0              ;Faz PD0 = 0
    rjmp repetec               ;desvia para repetec
seta: sbi PORTD,PD0             ;Faz PD0 = 1
    rjmp repetec               ;desvia para repetec
```

Após editar o programa acima, selecione “Build” no menu “Project” para compilar o arquivo fonte e link-editar o arquivo objeto gerado na compilação. Quando completar o processo o AVR Studio mostrará na janela “Output” a mensagem “Assembly complete with no errors”. A próxima ação é rodar o depurador. Para isso, selecione

“Start Debugging” no menu “Debug”. Você agora está no ambiente de simulação. Experimente-o!

O próximo programa testará o estado de um interruptor conectado do pino PB7 ao GND. Caso o interruptor esteja pressionado o microcontrolador colocará a letra ‘P’ em um display de 7 segmentos em anodo comum conectado a Porta D através de resistores de 470 Ω , conforme ilustra a Fig. 6.24. Caso o interruptor não esteja pressionado, a letra ‘S’ aparecerá no display.



O código fonte que executará essa função é apresentado a seguir.

```
.include "8515def.inc" ;Inclui todas as
                        ; variáveis do AT90S8515
.def TEMP              =R16      ;Reg. de temporário
.equ chave              =PB7
```

```
.cseg
    rjmp MAIN
; espaço reservado p/ os vetores de interrupção
MAIN: ldi TEMP,0b01111111
      out DDRD,TEMP          ; porta D como saída
      out PORTD,TEMP         ; em nível lógico 1
      ldi TEMP,0b00000000
      out DDRB,TEMP          ; porta B como
                              ; entrada
      ldi TEMP,0b10000000
      out PORTB,TEMP         ; PB7 com pull-up
REPETE:
      sbis PINB,CHAVE        ; Salta 1 instr. se a
                              ; Ch. estiver solta
      rjmp MOSTRA_P          ; Desvia para Mostra P
      ldi TEMP,0b00100100    ; Mostra "S" no
                              ; DISPLAY
      out PORTD,TEMP
      rjmp REPETE
MOSTRA_P:
      ldi TEMP,0b00011000    ; Mostra "P" no
                              ; DISPLAY
      out PORTD,TEMP
      rjmp repete
```

6.10 – Interrupções

6.10.1 – Introdução

O AT90S8515 manipula 13 fontes diferentes de interrupção, cada uma com o seu vetor associado. Cada interrupção possui um bit individual de habilitação que deve ser posto em 1 lógico juntamente com o bit I no registrador SREG para sua habilitação. Quando se produz uma interrupção qualquer de programa, o microcontrolador passa a executar as instruções que se situam a partir da posição em que se encontra o vetor associado a interrupção. É conveniente que na cabeça de cada vetor exista uma instrução de desvio incondicional,

RJMP, para uma posição de memória de programa (ou rótulo) onde, própria-mente, começará a rotina de interrupção. A lista completa de vetores é mostrada na Tabela 6.11. A lista determina, ainda, o nível de prioridade das diferentes interrupções, sendo o endereço mais baixo o de mais alta prioridade, ou seja, o RESET tem a mais alta prioridade, seguido de INT0, etc.

TABELA 6.11
VETORES DE INTERRUPÇÃO DO AT90S8515

Vetor n^o	Endereço de Programa	Fonte	Definição da Interrupção
1	\$000	RESET	Reset externo, Reset ao ligar e Reset por Cão de guarda
2	\$001	INT0	Requisição de Interrupção Externa 0
3	\$002	INT1	Requisição de Interrupção Externa 1
4	\$003	TIMER1 CAPT	Evento de Captura do Timer/Contador1
5	\$004	TIMER1 COMPA	Coincidência de Comparação A do Timer/Contador1
6	\$005	TIMER1 COMPB	Coincidência de Comparação B do Timer/Contador1
7	\$006	TIMER1 OVF	Estouro do Timer/Contador1
8	\$007	TIMER0, OVF	Estouro do Timer/Contador0
9	\$008	SPI, STC	Transferência Serial Completa
10	\$009	UART, RX	Recepção da UART Completa
11	\$010	UART, UDRE	Registro de Dados da UART Vazio
12	\$011	UART, TX	Transmissão da UART Completa
13	\$012	ANA_COMP	Comparador Analógico

Uma sugestão de programa de inicialização com os endereços dos vetores de interrupção é mostrada a seguir. Observe que o programa define o início da pilha no final da SRAM.

Endereço	Rótulo	Código	Comentário
\$000		RJMP main	; tratamento do RESET
\$001		RJMP EXT_INT0	; tratamento da IRQ0
\$002		RJMP EXT_INT1	; tratamento da IRQ1
\$003		RJMP TIM1_CAPT	; tratamento da captura do Timer1
\$004		RJMP TIM1_COMPA	; tratamento da comparação A do Timer1
\$005		RJMP TIM1_COMPB	; tratamento da comparação B do Timer1
\$006		RJMP TIM1_OVF	; tratamento do estouro de Timer1
\$007		RJMP TIM0_OVF	; tratamento do estouro de Timer0
\$008		RJMP SPI_STC	; tratamento da transferência SPI completa
\$009		RJMP UART_RXC	; tratamento da recepção da UART completa
\$00A		RJMP UART_DRE	; tratamento do reg. de dados da UART vazio
\$00B		RJMP UART_TXC	; tratamento da transmissão da UART completa
\$00C		RJMP ANA_COMP	; tratamento do comparador analógico
\$00D	main:	LDI r16, high(RAMEND)	; define pilha no final da SRAM
\$00E		OUT SPH, r16	
\$00F		LDI r16, low(RAMEND)	
\$010		OUT SPL, r16	
\$011		<instr> xxx	
...

6.10.2 – Tratamento de interrupções

O AT90S8515 apresenta dois registradores de controle de máscara de interrupção: GIMSK (General Interrupt MaSK register) no endereço \$3B do espaço de E/S e o TIMSK (Timer/Counter Interrupt MaSK register) no endereço \$39B do espaço de E/S.

Quando ocorre uma interrupção, o bit I de habilitação global de interrupção é colocado em 0 lógico e todas as interrupções são desabilitadas. O bit I é posto em 1 quando uma instrução de retorno de interrupção RETI é executada. Quando o contador de programa aponta para o vetor de interrupção para que ocorra a execução da rotina de tratamento da interrupção, o hardware limpa o flag que gerou a interrupção. Alguns dos flags de interrupção podem também ser limpos pela escrita de 1 lógico na posição do bit do flag a ser limpa.

Se a condição de interrupção ocorre quando o bit correspondente de habilitação de interrupção é 0 lógico, o flag de interrupção será posto em 1 e assim permanecerá até que a interrupção seja habilitada ou seja limpo por software. Se uma ou mais solicitações de interrupção ocorrerem quando o bit global de habilitação de interrupção está em 0, os flags correspondentes serão postos em 1 e assim permanecerão até que o bit global seja posto em 1, permitindo que as interrupções sejam executadas na ordem de prioridade.

Duas observações se fazem necessárias:

- O AT90S8515 disponibiliza para o usuário no registrador GIFR (General Interrupt Flag Register) flags para sinalizar o pedido de interrupção. O registrador GIFR é apresentado na Fig. 6.25.
- O registrador SREG não é automaticamente preservado quando se entra em uma rotina de interrupção e restaurado quando do seu retorno. Isto deve ser realizado pelo software. Na prática, salvá-lo é simples: a instrução `IN R0, SREG` salva-o e `OUT SREG, R0` o restaura.

Registrador GIFR – endereço \$3A ; **Valor na inicialização** = 00000000b

	INTF1	INTF0	-	-	-	-	-	-
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
INTF1	Flag da interrupção externa 1. Quando uma borda no terminal INT1 gatilha uma requisição de interrupção, o flag INTF1 vai para um lógico. Se o bit I do SREG e o bit INT1 do GIMSK estiverem em um lógico, o processamento desviará para o vetor de interrupção. Quando a rotina de interrupção for executada o flag INT1 vai para 0 lógico. O flag INT1, ainda, pode ser limpo escrevendo-se um lógico nele. Este flag é sempre limpo quando INT1 é configurada como interrupção por nível							
INTF0	Flag da interrupção externa 1. Quando uma borda no terminal INT0 gatilha uma requisição de interrupção, o flag INTF0 vai para um lógico. Se o bit I do SREG e o bit INT1 do GIMSK estiverem em um lógico, o processamento desviará para o vetor de interrupção. Quando a rotina de interrupção for executada o flag INT0 vai para 0 lógico. O flag INT0, ainda, pode ser limpo escrevendo-se um lógico nele. Este flag é sempre limpo quando INT0 é configurada como interrupção por nível.							
Bits 5, 4, 3, 2, 1 e 0	Reservados, sempre lidos como zero.							

Fig. 6.25 – Registrador GIFR.

6.10.3 – Interrupções Externas

As interrupções externas são controladas pelos terminais INT0 e INT1. Observe que quando habilitadas, as interrupções gatilharão mesmo que os terminais INT0/INT1 estejam configurados como saídas, característica que provê um modo de gerar interrupção por software. As interrupções podem ser ativadas na borda de descida ou subida ou, ainda, em nível baixo, dependendo da configuração dos bits ISCx1 e ISCx0 (Interrupt Sense Control x 1/0) no Registrador de Controle Geral da MCU (MCUCR), apresentado na Fig. 6.26. O nível e as bordas no terminal externo INTn que ativam a interrupção estão definidos na Tabela 6.12.

Quando controlada por nível, a interrupção fica pendente enquanto INTn se mantém em nível baixo. Note que a interrupção pode ser ativada mesmo se INTn for configurado como uma saída. Isto viabiliza a geração de interrupção por software.

O valor no terminal INTn é amostrado antes das bordas ocorrerem. Se a interrupção por borda é selecionada, pulsos com uma duração maior que um período de clock da CPU gerarão uma interrupção. Pulsos curtos não garantem a geração de uma interrupção. Se a interrupção por nível baixo é selecionada, para que ocorra uma interrupção, este nível deve ser mantido até que a instrução em execução encerre. Se habilitada, a ativação de interrupção por nível gerará um pedido de interrupção enquanto o terminal é mantido baixo.

O Registrador Geral da Máscara de Interrupção GIMSK é apresentado na Fig. 6.27.

Registrador MCUCR – endereço \$35; **Valor na inicialização** = 00000000b

	SRE	SRW	SE	SM	ISC11	ISC10	ISC01	ISC00
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
SRE	Bit de habilitação da SRAM externa . Quando o bit SER é posto em 1 lógico, a SRAM de dados externa é habilitada e são ativadas as funções alternativas dos terminais AD0 – 7 (Porta A), A8 – A15 (Porta C) \overline{WR} e \overline{RD} (Porta D). Quando o bit SER é posto em 0 lógico, a SRAM de dados externa é desabilitada e as configurações normais dos terminais são utilizadas.							
SRW	Bit de estado de espera da SRAM externa . Quando o bit SER é posto em 1 lógico, um estado de espera de um ciclo no ciclo é inserido no ciclo de acesso a SRAM externa. Quando o bit SER é posto em 0 lógico, o acesso a SRAM externa é executado com o esquema normal de três ciclos.							
SE	Bit de habilitação dos modos de redução de consumo (SLEEP) . O Bit SE deve ser posto em 1 lógico para fazer o microcontrolador entrar no modo sleep após a execução de uma instrução SLEEP. Para evitar a entrada indesejada no modo <i>sleep</i> , recomenda-se colocar o bit SE em 1 lógico um pouco antes da execução da instrução SLEEP.							
SM	O Bit modo de SLEEP seleciona entre dois dos modos de redução de consumo disponíveis. Quando SM é colocado em zero lógico, o modo de espera (<i>IDLE</i>) é selecionado; quando SM é colocado em um lógico, o modo de baixo consumo (<i>POWER-DOWN</i>) é selecionado. Os modos sleep serão vistos em detalhes na seção 6.12.							
ISC11 ISC10	Bits 0 e 1 de controle de sensoramento da Interrupção 1. A Interrupção Externa 1 é ativada pelo terminal externo INT1 se o flag I do SREG e a correspondente máscara de interrupção no registrador GIMSK estão em 1 lógico.							
ISC01 ISC00	Bits 0 e 1 de controle de sensoramento da Interrupção 0. A Interrupção Externa 0 é ativada pelo terminal externo INT0 se o flag I do SREG e a correspondente máscara de interrupção no registrador GIMSK estão em 1 lógico.							

Fig. 6.26 – Registrador MCUCR.

TABELA 6.12
CONTROLE DE SENSOREAMENTO DAS INTERRUPÇÕES
EXTERNAS

ISCn1	ISCn0	Descrição
0	0	O nível baixo de INT1 gera o pedido de interrupção
0	1	Reservado
1	0	A borda de descida de INT1 gera o pedido de interrupção
1	1	A borda de subida de INT1 gera o pedido de interrupção

onde n = 0 para Interrupção Externa 0 e n =1 para Interrupção Externa 1

Registrador GIMSK – endereço \$3B ; **Valor na inicialização** = 00000000b

	INT1	INT0	-	-	-	-	-	-
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
INT1	Bit de habilitação de atendimento da interrupção externa 1 . Quando os bits INT1 e I do SREG estão em 1 lógico, o terminal de interrupção externa está habilitado. Os bits ISC11 e ISC10 (Interrupt Sense Control 1 1/0) no Registrador de Controle Geral da MCU (MCUCR) define se a interrupção externa é ativada na borda de subida ou descida do sinal no terminal INT1 ou pela detecção de um nível baixo. A interrupção INT1 pode ser ativada mesmo se o pino é configurado como uma saída.							
INT0	Bit de habilitação de atendimento da interrupção externa 0 . Quando os bits INT0 e I do SREG estão em 1 lógico, o terminal de interrupção externa está habilitado. Os bits ISC01 e ISC00 (Interrupt Sense Control 0 1/0) no Registrador de Controle Geral da MCU (MCUCR) define se a interrupção externa é ativada na borda de subida ou descida do sinal no terminal INT0 ou pela detecção de um nível baixo. A interrupção INT0 pode ser ativada mesmo se o pino é configurado como uma saída.							
Bits 5, 4, 3, 2, 1 e 0	Reservados, sempre lidos como zero.							

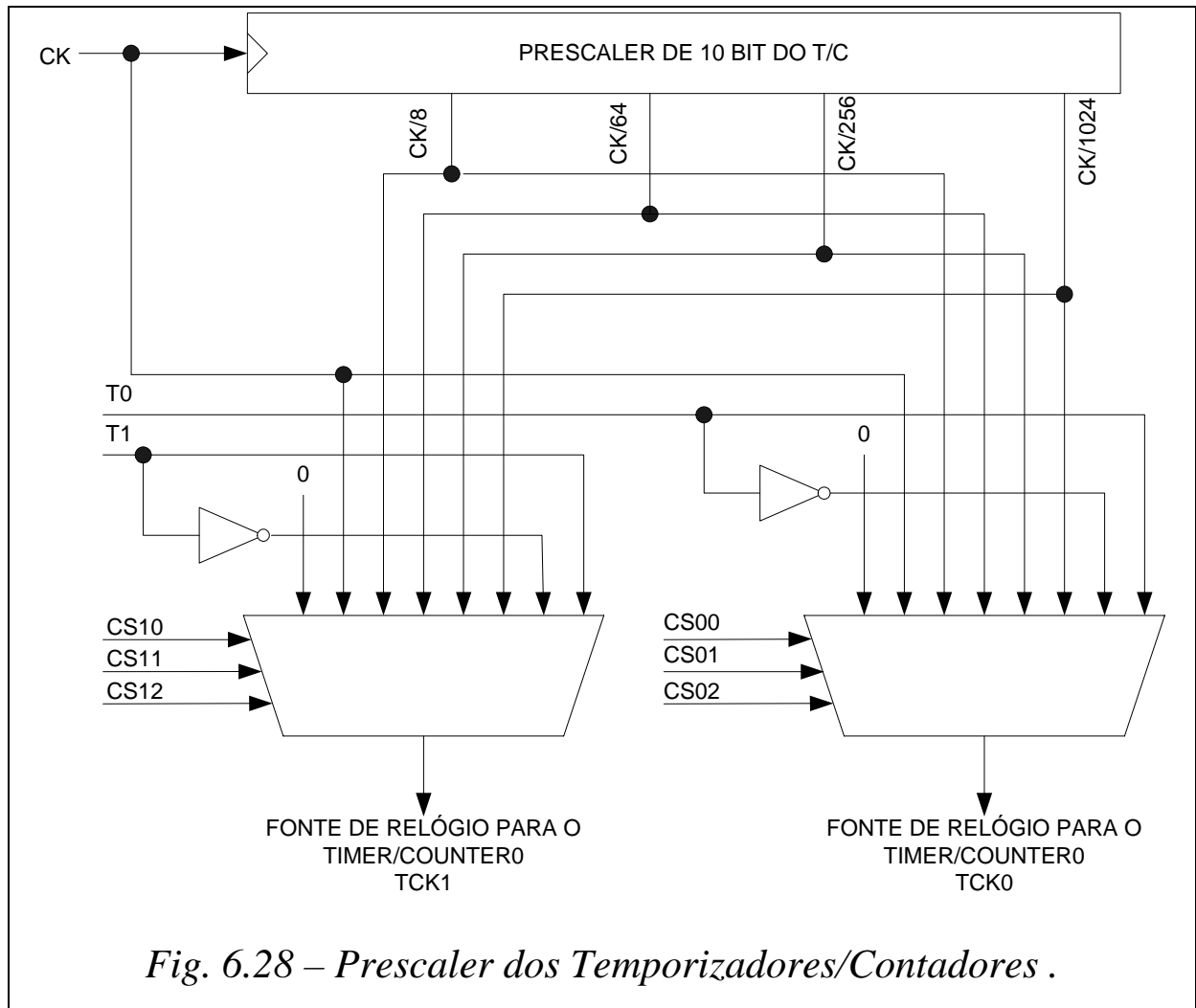
Fig. 6.27 – Registrador GIMSK.

A resposta de execução da interrupção para toda a interrupção habilitada do AVR é no mínimo 4 ciclos de relógio. Quatro ciclos após o flag de interrupção ser colocado em 1 lógico, o endereço do vetor de programa da rotina de tratamento da interrupção é executado. Durante este período de 4 ciclos, o contador de programa (9 bits) é enviado para o *Stack*. O vetor é, normalmente, um desvio relativo para a rotina de interrupção e este desvio leva dois ciclos de relógio. Se uma interrupção ocorrer durante a execução de uma instrução de múltiplos ciclos, esta instrução é completada antes da interrupção ser atendida.

6.11 – Temporizadores/Contadores

6.11.1 – Timer/counter0

O AT90S8515 possui um Temporizador/Contador de 8 bits de propósito geral – o Timer/counter0, que, por simplicidade, em nosso texto chamaremos apenas de **Timer 0**. A fonte do Timer 0 é um clock escalonado a partir de um temporizador com prescaler de 10 bits. Ele pode ser utilizado como um temporizador com uma base de tempo por clock interno ou como um contador com uma conexão externa, o qual sincroniza a contagem. A Fig. 6.28 ilustra o prescaler do Timer 0 e do segundo Timer, o Timer 1. São permitidas quatro seleções de prescaler – $CK/8$, $CK/64$, $CK/256$ e $CK/1024$, onde CK é a frequência do oscilador. São permitidas, ainda, como opções para fontes de clock CK , fonte externa e parada. A Fig. 6.29 mostra o diagrama de blocos para o Timer 0.

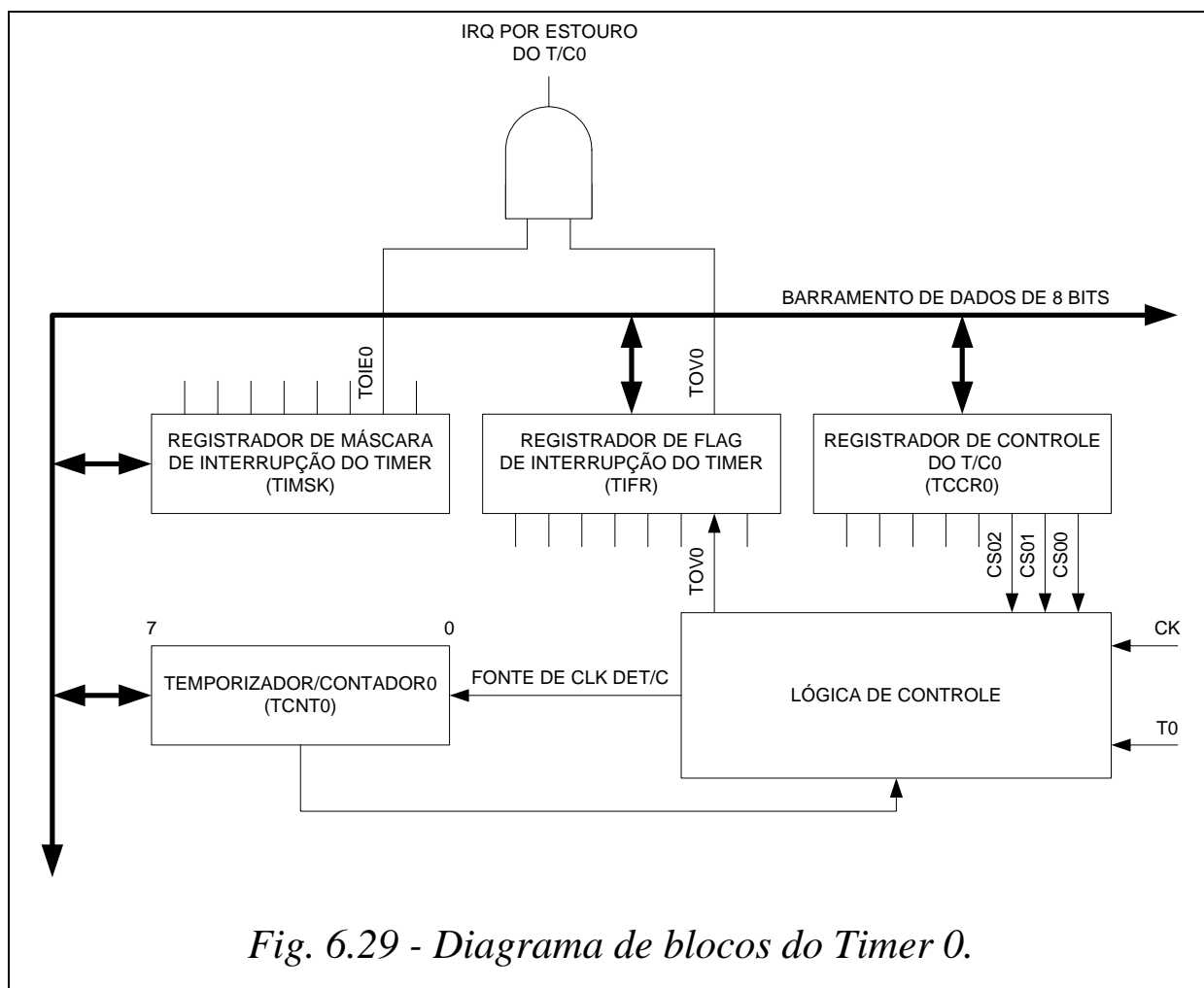


O Timer 0 pode selecionar como fonte de *clock* CK, CK escalonado ou terminal externo. Além disso, ele pode ser parado, conforme descreveremos mais diante.

O Timer 0 é implementado no registrador TCNT0 (\$32) como um contador crescente com acesso de leitura e escrita. Se ocorrer uma escrita no Timer 0 e uma fonte de *clock* está presente, ele continua a contagem no ciclo de relógio de temporização seguinte à operação de escrita. O Timer 0 pode gerar um pedido de interrupção quando o registrador TCNT0, que pode ser carregado por software, passa de FFh para 00h (estouro). O Flag de estado do estouro é

encontrado no Registrador de Flag de Interrupção por Temporizador/Contador (TIFR), apresentado na Fig. 6.30.

O registrador que permite configurar o sinal de relógio que controlará o Timer 0 é o Registrador de Controle do Temporizador/Contador0 (TCCR0), o qual é apresentado na Fig. 6.31. O bit de habilitação do Timer 0 é encontrado no Registrador de Máscara de Interrupção por Temporizador/Contador (TIMSK), o qual é apresentado na Fig. 6.32.



Registrador TIFR – endereço \$38 ; **Valor na inicialização** = 00000000b

	TOV1	OCF1A	OCF1A	-	ICF1	-	TOV0	-
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
TOV1	Flag de estouro do Timer 1. Este bit é posto em um lógico quando ocorre um estouro no Timer 1. Ele é limpo pelo hardware quando o vetor correspondente de interrupção é executado. Alternativamente, o bit TOV1 é limpo ao se escrever um lógico no mesmo. A interrupção por estouro do Timer 0 é executada quando os bits I do SREG, TOIE1 do TIMSK e o TOV1 estão em um lógico. No modo PWM, este bit é posto em um lógico quando o Timer 1 muda a direção de contagem em \$0000.							
OCF1A	Flag de Saída de Comparação 1A. Este bit vai a um lógico quando uma coincidência ocorre entre o Timer 1 e o dado no registrador de Saída de Comparação 1A - OCR1A. Ele é limpo pelo hardware quando for executado o vetor de tratamento de interrupção correspondente. Ainda, o bit OCF1A pode ser limpo escrevendo-se um lógico nele. Quando os bits I no SREG, OCIE1A no TIMSK e OCF1A estão em um lógico, a interrupção por Coincidência de Comparação A do Timer 1 é executada.							
OCF1B	Flag de Saída de Comparação 1B. Este bit vai a um lógico quando uma coincidência ocorre entre o Timer 1 e o dado no registrador de Saída de Comparação 1B - OCR1B. Ele é limpo pelo hardware quando for executado o vetor de tratamento de interrupção correspondente. Ainda, o bit OCF1B pode ser limpo escrevendo-se um lógico nele. Quando os bits I no SREG, OCIE1B no TIMSK e OCF1B estão em um lógico, a interrupção por coincidência de comparação B do Timer 1 é executada.							
Bits 4, 2 e 0	Reservados, sempre lidos como zero.							
ICF1	Flag de Entrada de Captura 1. O bit ICF1 vai par um lógico para sinalizar um evento de captura na entrada, indicando que o valor do Timer 1 foi transferido para o registrador de captura de entrada (ICR1). Ele é limpo pelo hardware quando for executado o vetor de tratamento de interrupção correspondente. Ainda, o bit ICF1 pode ser limpo escrevendo-se um lógico nele. Quando os bits I no SREG, TICIE1 no TIMSK e ICF1 estão em um lógico, a interrupção por Captura do Timer 1 é executada.							
TOV0	Flag de estouro do Timer 0. Este bit é posto em um lógico quando ocorre um estouro no Timer 0. Ele é limpo pelo hardware quando o vetor correspondente de interrupção é executado. Alternativamente, o bit TOV0 é limpo ao se escrever 1 lógico no mesmo. A interrupção por estouro do Timer 0 é executada quando os bits I do SREG, TOIE0 do TIMSK e o TOV0 estão em 1 lógico.							

Fig. 6.30– Registrador TIFR.

Registrador TCCR0 – endereço \$33 ; **Valor na inicialização** = 00000000b

	-	-	-	-	-	CS02	CS01	CS00
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
Bits 7, 6, 5, 4 e 3	Reservados, sempre lidos como zero.							
CS02 CS01 CS00	Bits 2, 1 e 0 de seleção do clock, os quais definem a fonte de escalonamento do Timer 0 como segue:							
	CS02	CS01	CS00	Fonte				
	0	0	0	Parada, o Timer 0 é parado.				
	0	0	1	CK				
	0	1	0	CK/8				
	0	1	1	CK/64				
	1	0	0	CK/256				
	1	0	1	CK/1024				
	1	1	0	Terminal externo T0, borda de descida.				
	1	1	1	Terminal externo T0, borda de subida.				

Fig. 6.31 – Registrador TCCR0.

Quando o Timer 0 é controlado externamente, o sinal externo é sincronizado com a frequência de oscilação da CPU. Para garantir uma amostragem apropriada do *clock* externo, o tempo mínimo entre duas transições externas deve ser de pelo menos um período de relógio interno da CPU. O *clock* externo é amostrado na borda de subida do relógio interno da CPU.

A condição de parada fornece uma função para habilitar/desabilitar. Se os modos externos são usados, transições em T0 controlarão o contador mesmo quando o terminal é configurado como saída.

Registrador TIMSK – endereço \$39; **Valor na inicialização** = 00000000b

	TOIE1	OCIE1A	OCIE1B	-	TICIE1	-	TOIE0	-
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
TOIE1	Habilitação de Interrupção por Estouro do Timer 1. Quando este bit e o bit I no SREG estão em um lógico, a interrupção por Estouro do Timer 1 está habilitada. A interrupção correspondente é executada no vetor \$0006 se um estouro no Timer 1 ocorre, isto é, quando o bit TOV1 no TIFR vai a um lógico.							
OCE1A	Habilitação de Interrupção da Saída de Coincidência de Comparação A do Timer 1. Quando este bit e o bit I no SREG estão em um lógico, a interrupção por Coincidência de Comparação A do Timer 1 está habilitada. A interrupção correspondente é executada no vetor \$0004 se uma Coincidência de Comparação A no Timer 1 ocorre, isto é, quando o bit OCF1A no TIFR vai a um lógico.							
OCE1B	Habilitação de Interrupção da Saída de Coincidência de Comparação B do Timer 1. Quando este bit e o bit I no SREG estão em um lógico, a interrupção por Coincidência de Comparação B do Timer 1 está habilitada. A interrupção correspondente é executada no vetor \$0004 se uma Coincidência de Comparação B no Timer 1 ocorre, isto é, quando o bit OCF1B no TIFR vai a um lógico.							
Bits 4, 2 e 0	Reservados, sempre lidos como zero.							
TICIE1	Habilitação de Interrupção da Entrada de Captura do Timer 1. Quando este bit e o bit I no SREG estão em um lógico, a interrupção da Entrada de Evento de Captura de de na do Timer 1 está habilitada. A interrupção correspondente é executada no vetor \$0003 se evento de gatilhamento-captura ocorre no pino 31 - ICP, isto é, quando o bit ICF1 no TIFR vai a um lógico.							
TOIE0	Habilitação de Interrupção por Estouro do Timer 0. Quando este bit e o bit I no SREG estão em um lógico, a interrupção por Estouro do Timer 0 está habilitada. A interrupção correspondente é executada no vetor \$0007 se um estouro no Timer 0 ocorre, isto é, quando o bit TOV0 no TIFR vai a um lógico.							

Fig. 6.32– Registrador TIFR.

Como exemplo de utilização do Timer 0 vamos gerar uma onda quadrada de 1 kHz no terminal PD0 utilizando o Timer 0. Para gerar uma onda de 1 kHz devemos inverter o estado do pino PD0 a cada 500 μ s. Para gerar o atraso de 500 μ s vamos incrementar o registrador TCNT0 de um valor inicial até o estouro. Vamos, ainda,

selecionar a frequência do relógio do Timer 0 como sendo a frequência do oscilador por 8, resultando em 4 MHz / 8, ou seja, 500 kHz. Assim, cada contagem do Timer 0 representará 2 μ s. Deste modo, para gerar um tempo de 500 μ s será preciso 250 contagens do Timer 0, ou seja, devemos carregar TCNT0 com 6 (256 – 250). Abaixo apresentamos, o código fonte necessário para o microcontrolador executar a função solicitada.

```
.include "8515def.inc" ;Inclui todas as
                        ; variáveis do AT90S8515
.equ val_inicial =0x06 ; val.inicial do Timer 0
.def TEMP          =R16    ;Reg. temporario
.cseg
    rjmp MAIN
;
TIM0_OVF:
    ldi TEMP,val_inicial
    out TCNT0,TEMP        ;reinicializa a
                        ; contagem
    sbis PORTD,PD0        ;salta 1 instrução
                        ; se PD0=1
    rjmp seta             ;desvia para seta
    cbi PORTD,PD0         ;Faz PD0 = 0
    reti                  ;desvia para repete
seta: sbi PORTD,PD0        ;Faz PD0 = 1
    reti                  ;desvia para repete
MAIN: ldi TEMP,0b0000001   ; TEMP = 0x01
    out DDRD,TEMP         ;DDRD = 0x01
                        ; PD0 é saída
    ldi TEMP,val_inicial
    out TCNT0,TEMP        ;inicializa a
                        ; contagem
    ldi TEMP,0b00000010   ;
    out TCCR0,TEMP        ;define prescaler /8
    ldi TEMP,0b00000010   ;habilita interrup.
    out TIMSK,TEMP        ; por Timer 0
    sei                   ; habilita interrupções
volta:
    rjmp volta            ; fica aqui
```

Para ilustrar o funcionamento das interrupções externas faremos modificações no programa anterior, tais que a cada transição de subida no terminal de interrupção externa o programa alterna entre as frequências de saída de 1 e 2 kHz. A rotina de interrupção externa utiliza o estado do bit T, complementado a cada pedido de interrupção, para determinar qual valor inicial de contagem será carregado no Timer 0, alternando desta forma a frequência de saída.

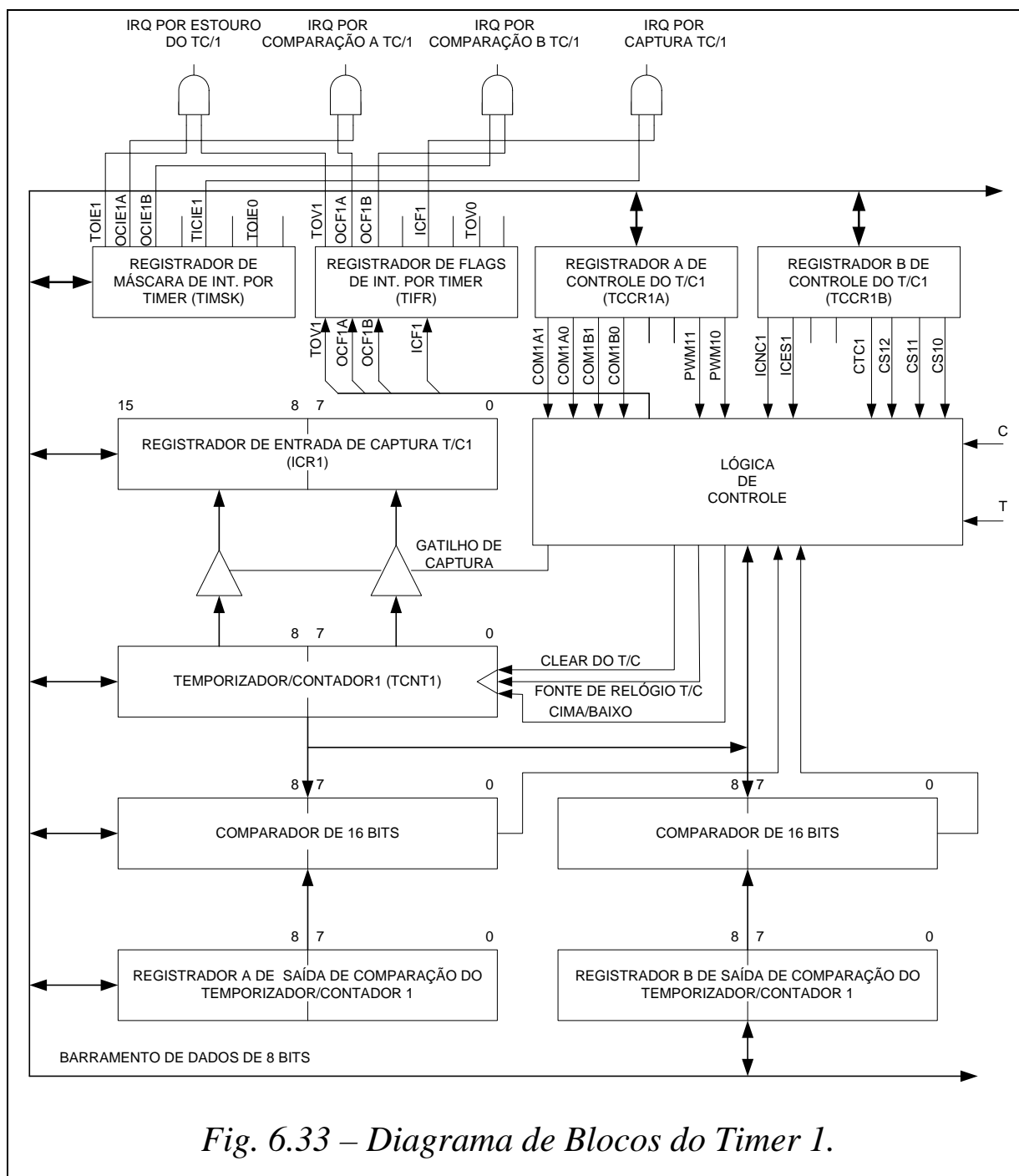
```
.include "8515def.inc" ;Inclui todas as
                        ;variáveis do AT90S8515
                        ;
.equ val_1k =0x06      ; valor inicial para 1 k
.equ val_2k =131       ; valor inicial para 2 k
.def TEMP             =R16 ;Reg. de armazenamento
                        ; temporario

.cseg
    rjmp MAIN
    rjmp EXT_INT0
    rjmp TIM0_OVF
EXT_INT0:
    brts limpa         ; desvia se T=1
    set                 ;Faz T = 1
    ldi TEMP,val_2k
    reti               ;retorna
limpa:    clt           ;Faz T=0
    ldi TEMP, val_1k
    reti
TIM0_OVF:
    out TCNT0,TEMP     ;reinicializa a contagem
    sbis PORTD,PD0     ;salta 1 instrução se
                        ;PD0=1
    rjmp seta          ;desvia para seta
    cbi PORTD,PD0      ;Faz PD0 = 0
    reti              ;desvia para repete
seta:    sbi PORTD,PD0  ;Faz PD0 = 1
    reti              ;desvia para repete
;
MAIN:    ldi TEMP,0b0000001 ; TEMP = 0x01
        out DDRD,TEMP      ; DDRD = 0x01
```

```
                                ; PD0 como saída
ldi TEMP,val_1k
out TCNT0,TEMP                ;inicializa contagem
ldi TEMP,0b00000010          ;
out TCCR0,TEMP                ;define prescaler /8
ldi TEMP,0b00000010          ;habilita interrup.
out TIMSK,TEMP                ; por Timer 0
                                ;
ldi TEMP,0b00000011          ;transição de subida
out MCUCR,TEMP                ;na interrupção
                                ;externa 0
ldi TEMP,0b01000000          ;habilita interrup.
out GIMSK,TEMP                ; por Timer 0
                                ;
sei                            ; habilita interrupções
volta:
rjmp volta                    ; fica aqui
```

6.11.2 – Temporizador/Contador 1 de 16 bits – Timer 1

A Fig. 6.33 mostra a estrutura do Temporizador/Contador 1 de 16 bits, doravante chamado de Timer 1. Do mesmo modo que o Timer 0, ele pode selecionar como fonte de relógio CK, CK escalonado ou um terminal externo. Seus diferentes flags de estado (estouro, coincidência de comparação e captura de evento) são encontrados no Registrador de Flags de Interrupção por Temporizador/Contador – TIFR. Os sinais de controle do Timer 1 encontram-se nos Registradores de Controle do Temporizador/Contador 1 – TCCR1A e TCCR1B. As opções para habilitar/desabilitar as interrupções por Timer 1 são encontradas no Registrador de Máscara de Interrupção por Temporizador/Contador – TIMSK.



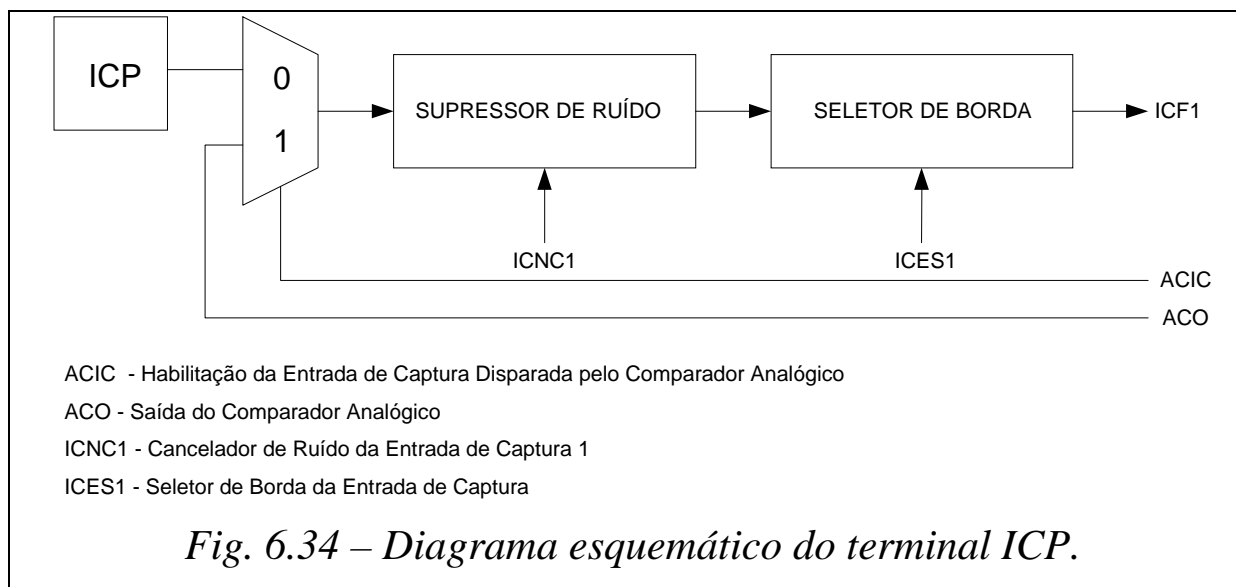
Quando o Timer 1 é controlado externamente, o sinal externo é sincronizado com a frequência de oscilação da CPU. Para garantir uma amostragem apropriada do relógio externo, o tempo mínimo entre duas transições externas deve ser de pelo menos um período de

relógio interno da CPU. O relógio externo é amostrado na borda de subida do relógio interno da CPU.

O Timer 1 possui duas saídas para as funções de comparação utilizando os Registradores 1A e B de Saída de Comparação (OCR1A e OCR1B) como fonte de dados para serem comparados ao conteúdo do Timer 1. As saídas das funções de comparação incluem uma limpeza opcional do contador quando o valor de comparação A for atingido e ações nos pinos de Saída de Comparação para ambos os valores de comparação (A e B).

O Timer 1 pode também ser usado como um Modulador por Largura de Pulso (PWM) de 8, 9 e 10 bits. Neste modo, o contador e os registradores OCR1A e OCR1B são utilizados como um PWM duplo, livre de falhas e autônomo.

A entrada da função de captura do Timer 1 fornece uma captura do conteúdo do Timer 1 pelo Registrador de Entrada de Captura (ICR1), gatilhada por um evento externo no pino de entrada de captura (ICP). As configurações para a captura de eventos são definidas pelo Registrador de Controle dos Temporizadores/Contadores. Complementarmente, o Comparador Analógico pode ser configurado para disparar um evento de captura. A lógica do terminal ICP é mostrada na Fig. 6.34. Se a função de cancelamento de ruído está habilitada, a condição de gatilhamento para o evento de captura é monitorada sobre quatro amostras e todas as quatro devem ser iguais para ativar o flag de captura (ICF1).



Registadores de Controle do Timer 1

As Figs. 6.35 e 6.36 apresentam os registradores de Controle do Timer 1 – TCCR1A e TCCR1B. O registrador TCCR1A contém os bits de controle das saídas de Comparação A e B e do modo PWM do Timer 1. Por sua vez, o registrador TCCR1B contém o bit que habilita a função de supressão de ruído na entrada de captura 1, o bit de seleção da borda da entrada de captura 1 e os bits que definem a fonte de escalonamento do Timer 1.

Registadores de Contagem do Timer 1

O registrador de 16 bits TCNTH1:TCNT1L, mostrado na Fig. 6.37, guarda o valor da contagem do Timer 1. Para garantir que os bytes alto e baixo sejam simultaneamente lidos ou escritos quando a CPU acessa o registrador, o acesso é realizado usando um registrador temporário de 8 bits (TEMP). Esse registrador temporário é usado, também, no acesso a OCR1A, OCR1B e ICR1. Se o programa principal e as rotinas de interrupção realizam o acesso a registradores

que usam TEMP, as interrupções devem ser desabilitadas no programa principal durante o acesso (e nas rotinas de interrupção caso interrupções sejam permitidas dentro das rotinas de interrupção).

Registrador TCCR1A – endereço \$2F ; **Valor na inicialização** = 00000000b

	COM1A1	COM1A0	COM1B1	COM1B0	-	-	PWM11	PWM10
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
Bits 7, 6	Bits de Modo da Saída de Comparação 1A. Os bits de controle COM1A1 e COM1A0 determinam qual ação de saída sobre o pino OC1A (Saída de Comparação A do Timer 1) segue a coincidência de comparação no Timer 1. É necessário, previamente, configurar o pino OC1A, uma função alternativa de PD5, como saída. Os tipos de ação que se produzem em OC1A são resumidas na Tabela 6.13.							
Bits 5, 4	Bits de Modo da Saída de Comparação 1B. Os bits de controle COM1B1 e COM1B0 determinam qual ação do pino de saída sobre o pino OC1A (Saída de Comparação B do Timer 1) segue a coincidência de comparação no Timer 1. Os tipos de ação que se produzem sobre esse pino são resumidas na Tabela 6.13.							
Bits 3, 2	Reservados, sempre lidos como zero.							
Bits 1, 0	Bits de Seleção do PWM. Esses bits selecionam a operação PWM do Timer 1 conforme mostra a Tabela 6.14.							

Fig. 6.35 – Registrador TCCR1A.

TABELA 6.13

SELEÇÃO DO MODO DE COMPARAÇÃO 1

COM1X1	COM1X0	Descrição
0	0	Timer 1 desconectado do pino de saída OC1X*
0	1	Pulsa a linha de saída OC1X
1	0	Baixa a linha de saída OC1X (para zero)
1	1	Eleva a linha de saída OC1X (para um)

onde X = A ou B. No modo PWM esses bits tem uma função diferente, conforme Tabela 6.23.

* O pino OC1B é dedicado, ele nunca é desconectado do Timer 1.

Registrador TCCR1B – endereço \$2E ; **Valor na inicialização** = 00000000b

	ICNC1	ICES1	-	-	CTC1	CS12	CS11	CS10
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
Bits 7	Supressor de Ruído da Entrada de Captura 1. Quando o bit ICNC1 está em zero lógico, a função supressora de ruído de disparo da entrada de captura está desabilitada. A entrada de captura é disparada na primeira borda de descida/subida amostrada pela entrada ICP. Quando o bit ICNC1 está em um lógico, haverá conteúdo do conteúdo do contador se quatro amostras sucessivas medidas em ICP forem altas ou baixas, conforme o estado lógico do bit ICES1.							
Bit 6	Bit Seletor da Borda da Entrada de Captura 1. Enquanto o bit ICES1 estiver em zero lógico, os valores dos registradores do Timer 1 são transferidos para o Registrador da Entrada de Captura (ICR1) na borda de descida do pino de entrada de captura (ICP). Enquanto o bit ICES1 estiver em um lógico, os valores são transferidos para o para ICR1 na borda de descida de ICP.							
Bits 5, 4	Reservados, sempre lidos como zero.							
Bit 3	Bit de Limpeza do Timer 1 após Coincidência na Comparação. Quando o bit CTC1 está em um lógico, o registrador do Timer 1 é posto em \$0000 no ciclo de relógio subsequente a uma coincidência de comparação A. Se CTC1 está em 0 lógico, o Timer 1 continua a contagem e não é afetado por coincidência. Já que a Coincidência de Comparação é detectada no ciclo de relógio da CPU subsequente a coincidência, esta função comporta-se-á de maneira diferente quando um escalonamento maior do que um for usado pelo temporizador. Quando um escalonamento de um é usado e o registrador de comparação A contém C, o temporizador contará como segue: ... C-2 C-1 C 0 1 ... Quando o prescaler é configurado para dividir por 8, o temporizador contará como segue: ... C-2, C-2, C-2, C-2, C-2, C-2, C-2, C-2 C-1, C-1, C-1, C-1, C-1, C-1, C-1, C-1 C, C, C, C, C, C, C, C 0, 0, 0, 0, 0, 0, 0, 0 ... No modo PWM este bit não tem efeito.							
CS12 CS11 CS10	Bits 2, 1 e 0 de seleção do clock, os quais definem a fonte de escalonamento do Timer 1 conforme mostra a Tabela 6.15.							

Fig. 6.36 – Registrador TCCR1B.

Quando a CPU escreve no byte alto TCNT1H, o dado escrito é colocado no registrador TEMP. Após, quando a CPU escreve no byte baixo TCNT1L, este byte de dado é combinado com o byte de dado no registrador TEMP e todos os 16 bits são escritos simultaneamente no registrador TCNT1 do Timer 1. Conseqüentemente, TCNT1H deve ser acessado primeiro para uma operação de escrita completa de 16 bits.

TABELA 6.14
SELEÇÃO DO MODO PWM

PWM11	PWM10	Descrição
0	0	Operação PWM do Timer 1 desabilitada
0	1	Timer 1 é um PWM de 8 bits
1	0	Timer 1 é um PWM de 9 bits
1	1	Timer 1 é um PWM de 10 bits

TABELA 6.15
SELEÇÃO DO PRESCALER PARA o TIMER 1

CS12	CS11	CS10	Fonte
0	0	0	Parada, o Timer 0 é parado.
0	0	1	CK
0	1	0	CK/8
0	1	1	CK/64
1	0	0	CK/256
1	0	1	CK/1024
1	1	0	Terminal externo T1, borda de descida.
1	1	1	Terminal externo T1, borda de subida.

Bit	15	14	13	12	11	10	9	8	
\$2D	MSB								TCNT1H
\$2C								LSB	TCNT1L
	7	6	5	4	3	2	1	0	

Fig. 6.37 – Par de Registradores TCNT1H:TCNT1L.

Quando a CPU lê o byte baixo TCNT1L, o dado do byte baixo é enviado para a CPU e o dado do byte alto TCNT1H é colocado no registrador TEMP. Quando a CPU lê o dado no byte alto TCNT1H, a CPU recebe o dado no registrador TEMP. Conseqüentemente, TCNT1L deve ser acessado primeiro para uma operação de leitura completa de 16 bits.

O Timer 1 é implementado como um contador crescente ou crescente/decrescente (no modo PWM) com acesso de leitura e escrita. Se for escrito no Timer 1 e uma fonte de relógio está selecionada, o Timer 1 continua a contagem no ciclo de clock de timer posterior a escrita do valor.

Registradores de Saída de Comparação

Os registradores de saída de comparação, apresentados na Fig. 6.38, são registradores de escrita/leitura de 16 bits.

Os registradores de Saída de Comparação do Timer 1 contêm os dados que são continuamente comparados com o Timer 1. As ações na coincidência de comparação são especificadas nos registradores de Controle do Timer 1 e de Estado. Uma coincidência de comparação só ocorre se o Timer 1 conta até o valor contido no OCR. Um software

que inicializa TCNT1 e OCR1A ou OCR1B com os mesmos valores não gera uma ação de coincidência de comparação.

Bit	15	14	13	12	11	10	9	8	
\$2B	MSB								OCR1AH OCR1AL
\$2A								LSB	
	7	6	5	4	3	2	1	0	

(a)

Bit	15	14	13	12	11	10	9	8	
\$29	MSB								OCR1BH OCR1BL
\$28								LSB	
	7	6	5	4	3	2	1	0	

(b)

Fig. 6.38 – (a) Par de Registradores OCR1AH:OCR1AL e (b) par de Registradores OCR1BH:OCR1BL.

Uma coincidência de comparação porá em nível lógico um o flag de interrupção de comparação no ciclo de clock seguinte ao evento de comparação.

Como os registradores de Saída de Comparação (OCR1A e OCR1B) são de 16 bits, para garantir que ambos os bytes sejam simultaneamente atualizados, o registrador temporário TEMP é usado ao se escrever em OCR1A/B. Quando a CPU escreve no byte alto OCR1AH ou OCR1BH, o dado escrito é armazenado temporariamente no registrador TEMP. Quando a CPU escreve no byte baixo OCR1AL ou OCR1BL, o registrador TEMP é simultaneamente copiado para OCR1AH ou OCR1BH. Conseqüentemente, o byte alto deve ser acessado primeiro para uma operação de escrita completa de 16 bits.

O exemplo a seguir gera uma base de tempo de um segundo para um seqüencial de LEDS utilizando o registrador de coincidência de OCR1A. Com um cristal de 3,6864 Mhz, utilizando-se um prescaler de 64, chega-se a uma frequência de operação do timer um de 57600 ciclos/s. Assim, ajustando-se o registrador de comparação em 57600 (\$E100) consegue-se uma base de tempo de um segundo.

```

;*****
;
;
; EXEMPLO DE EMPREGO DO REG. DE SAÍDA DE COMPARAÇÃO
;
;*****
;
;
.include "8515def.inc" ;Inclui todas as variáveis do AT90S8515
;
;*****
;
;
.def TEMP          =R17 ;Registro de armazenamento temporario
.def LEDS          =R18
;
;*****
;
;
; Definição dos Pinos de E/S
;
;*****
;
;
.cseg
.org 0x00                      ; RESET
                rjmp MAIN
;
;*****
;
;vetor de interrupção
.org 0x04
                rol LEDS
                out portb,LEDS
                reti
;
;*****

```

```

MAIN:    ldi temp,high(RAMEND)           ;inicializa stack
        out SPH,temp
        ldi temp,low(RAMEND)
        out SPL,temp
;
; Definição dos pinos de E/S
        ldi temp,$ff           ; PORTA B como saída
        out ddrb,temp
;
; Início do Programa Principal
;
        ldi LEDS,$ff
        ldi temp,$e1
        out ocr1Ah,temp         ;carrega valores para comparação
        ldi temp,$00
        out ocr1Al,temp
        sei
        ldi temp,1<<OCIE1A     ;habilita interrupção por
        out timsk,temp         ;coincidência de comparação A
        ldi temp,0b00001011    ;limpa timer1 após coincidência de
        out tccr1b,temp        ;comparação A e prescaler = 64
stop:    rjmp stop

```

Registradores de Entrada de Captura

Os registradores de entrada de captura, apresentado na Fig. 6.39, é um registrador de 16 bits somente de leitura.

Bit	15	14	13	12	11	10	9	8	
\$25	MSB								ICR1H
\$24								LSB	ICR1L
	7	6	5	4	3	2	1	0	

Fig. 6.39 – Par de Registradores ICR1H:ICR1L.

Quando uma borda de descida ou subida (de acordo com a configuração da borda da entrada de captura, bit ICES1) do sinal na entrada de captura (ICP), o conteúdo do par TCNTH1:TCNTL1 é copiado para o registrador de captura ICR1.

Como o registrador de Entrada de Captura (ICR1) é de 16 bits, para garantir que ambos os bytes sejam simultaneamente atualizados, o registrador temporário TEMP é usado quando ICR1 for lido. Quando a CPU lê o byte baixo ICR1L, o dado é enviado para a CPU e o byte alto ICR1H é colocado no registrador TEMP. Quando a CPU lê o dado do byte alto ICR1H ou OCR1BL, a CPU recebe o dado do registrador TEMP. Conseqüentemente, o byte baixo deve ser acessado primeiro para uma operação de escrita completa de 16 bits.

Modo de PWM do Timer 1

Quando o modo PWM é selecionado, o Timer 1, o registrador de Saída de Comparação 1A (OCR1A) e o registrador de Saída de Comparação 1B (OCR1B) formam um PWM duplo de 8, 9 ou 10 bits, livre de falhas e autônomo com saídas no pino PD5 (OC1A) e OC1B. O Timer 1 atua como contador crescente/decrescente, contando de \$0000 até o valor de topo (ver Tabela 6.16), quando ele retorna e conta para baixo até zero, antes de repetir o ciclo. Quando o valor de contagem coincide com o conteúdo dos 10 bits menos significativos de OCR1A ou OCR1B, os pinos PD5(OC1A)/OC1B serão postos em zero ou um lógico, de acordo com a configuração dos bits COM1A1/COM1A0 ou COM1B1/COM1B0 no registrador de Controle do Timer 1 (TCCR1A), conforme ilustra a Tabela 6.17.

TABELA 6.16

Valores de Topo do Timer e Frequência do PWM

Resolução do PWM (bits)	Valor de Topo do Timer	Frequência
8	\$00FF (255)	$f_{TCLK1} / 510$
9	\$01FF (511)	$f_{TCLK1} / 1022$
10	\$03FF (1023)	$f_{TCLK1} / 2046$

Note que no modo PWM, os 10 bits menos significativos de OCR1A/OCR1B, quando escritos, são transferidos para um endereço temporário. Eles são retidos quando o Timer 1 alcança o valor de topo. Isto previne a ocorrência de pulsos PWM de comprimento indesejáveis (falhas) na caso de uma operação de escrita não sincronizada em OCR1A/OCR1B, conforme ilustra a Fig. 6.40.

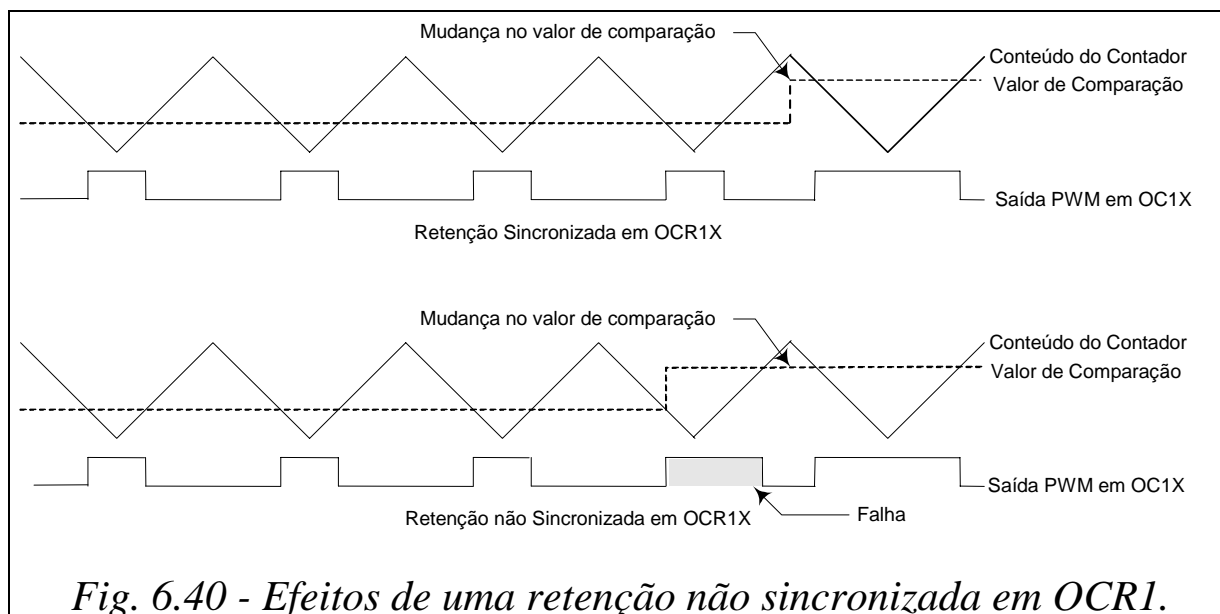
TABELA 6.17

SELEÇÃO DO MODO DE COMPARAÇÃO 1 NO MODO PWM

COM1X1	COM1X0	Descrição
0	0	Não conectado*
0	1	Não conectado*
1	0	Posto em 0 lógico na coincidência de comparação, contando para cima. Posto em 1 lógico na coincidência de comparação, contando para baixo. (PWM não inversor).
1	1	Posto em 0 lógico na coincidência de comparação, contando para baixo. Posto em 1 lógico na coincidência de comparação, contando para cima. (PWM inversor).

onde X = A ou B.

* O pino OC1B é dedicado, ele nunca é desconectado do Timer 1.



Durante o tempo entre a escrita e uma operação de retenção, uma leitura de OCR1A ou OCR1B lerá o conteúdo do endereço temporário. Isto significa que o valor mais recentemente escrito sempre será lido a partir de OCR1A/B

Quando OCR1 contém \$0000 ou o valor de topo, as saídas OC1A/OC1B são atualizadas para baixo ou alto na próxima coincidência de comparação, de acordo com as configurações dos bits COM1A1/COM1A0 ou COM1B1/COM1B0. Isto é mostrado na Tabela 6.18.

No modo PWM, o flag de estouro do Timer 1 (TOV1) é posto em um lógico quando o contador atinge \$0000. A interrupção por estouro do Timer 1 opera exatamente como no modo normal, isto é, é executada quando TOV1 vai a um lógico, desde que os bits de habilitação de interrupção por estouro do Timer 1 (TOIE1 do registrador TIMSK) e de interrupção global (I em SREG) estejam em

um lógico. Isto também se aplica aos flags e interrupções das Saídas de Comparação do Timer 1.

TABELA 6.18

SAÍDAS PWM com OCR1X = \$0000 ou valor de topo.

COM1X1	COM1X0	OCR1X	Saída OC1X
1	0	\$0000	Baixa
1	0	topo	Alta
1	1	\$0000	Alta
1	1	topo	Baixa

onde X = A ou B.

O Exemplo a seguir gera uma onda senoidal de frequência igual a 56,5 Hz no pino PD5. Para calcular esta frequência utiliza-se a expressão

$$f = \frac{f_{TCLK1}}{res \cdot n}$$

onde:

f_{TCLK1} - frequência de operação do timer 1;

res - 510, 1022 ou 2046, conforme a resolução do PWM;

n - número amostras um período.

```

;*****
;
;
; EXEMPLO DE UTILIZAÇÃO DA SAÍDA PWM
;
;*****
;
;
;include "8515def.inc"           ;Inclui todas as variáveis do AT90S8515
;
;*****
;
;
.def TEMP           =R17           ;Registro de armazenamento temporario

```

```

;
;
;*****
;
;
; Definição dos Pinos de E/S
;
;
;*****
.cseg
.org 0x00                                ; RESET
                rjmp MAIN
;
;
;*****
;vetores de interrupção
;
;Vetor do estouro do timer1
;
.org 0x06
                lpm                        ;descarrega tabela em r0
                mov temp,r0              ;se tabela conter
                cpi temp,$ff              ; ff volta ao inicio
                breq reload              ;da tabela
load:           out OCR1AL,r0            ;carrega PWM
                adiw zh:zl,1             ;atualiza ponteiro de tabela
                reti
reload:         ldi zh,high(tabela<<1)   ;inicializa pont. da tabela
                ldi zl,low(tabela<<1)
                lpm                      ;descarrega tabela em r0
                rjmp load
;
;
;*****
; PROGRAMA PRINCIPAL
;
;
MAIN:           ldi temp,high(RAMEND)     ;inicializa stack
                out SPH,temp              ;no topo da SRAM
                ldi temp,low(RAMEND)
                out SPL,temp
;
;
;*****
;
;
; Definição dos pinos de E/S
;
                ldi temp, 0b00100000     ; PD5 como saída
                Out DDRD, temp
;
;
;*****
;

```

```

; Início do Programa Principal
;
    ldi zh,high(tabela<<1)    ;inicializa pont. da tabela
    ldi zl,low(tabela<<1)
    ldi temp,1<<TOIE1        ; habilita int. p/ estouro de Timer 1
    out TIMSK,temp
    ldi temp,(1<<COM1A1)|(1<<PWM10);Saída de Comparação A
    out TCCR1A,temp          ;ligada a PD5 e PWM de 8 bits ativo
    ldi temp,1<<CS10         ; Timer 1 ligado com fonte clk
    out TCCR1B,temp
    sei                      ; Habilita interrupção global
stop:    rjmp stop           ; aguarda interrupção
;
;*****
;
;    SENÓIDE
;
;
tabela:    .dw 0x857F, 0x928B, 0x9E98, 0xAAA4, 0xB5B0, 0xC0BB
           .dw 0xCBC6, 0xD4D0, 0xDDD9, 0xE5E1, 0xECE9, 0xF2EF
           .dw 0xF7F4, 0xFAF9, 0xFDFC, 0xFEFD, 0xFEFE, 0xFDFD
           .dw 0xFAFC, 0xF7F9, 0xF2F4, 0xECEF, 0xE5E9, 0xDDE1
           .dw 0xD4D9, 0xCBD0, 0xC0C5, 0xB5BB, 0xAAB0, 0x9EA4
           .dw 0x9298, 0x858B, 0x797F, 0x6C72, 0x6066, 0x545A
           .dw 0x494E, 0x3E43, 0x3338, 0x2A2E, 0x2125, 0x191D
           .dw 0x1215, 0x0C0F, 0x070A, 0x0405, 0x0102, 0x0001
           .dw 0x0000, 0x0101, 0x0402, 0x0707, 0x0C0A, 0x120F
           .dw 0x1915, 0x211D, 0x2A25, 0x332F, 0x3E39, 0x4943
           .dw 0x544F, 0x605A, 0x6D66, 0x7973, 0x00FF

```

6.11.3 – O Timer Cão de Guarda (Watchdog)

O temporizador *watchdog* é inicializado pela instrução WDR e acionado a partir de um oscilador interno separado que funciona com 1 MHz para $V_{CC} = 5\text{ V}$. Controlando o *prescaler* do *watchdog*, o intervalo de *reset* por *watchdog* (WDR) pode ser ajustado conforme indica a Tabela 6.19, onde percebemos que oito diferentes períodos de relógio podem ser selecionados para determinar o período máximo

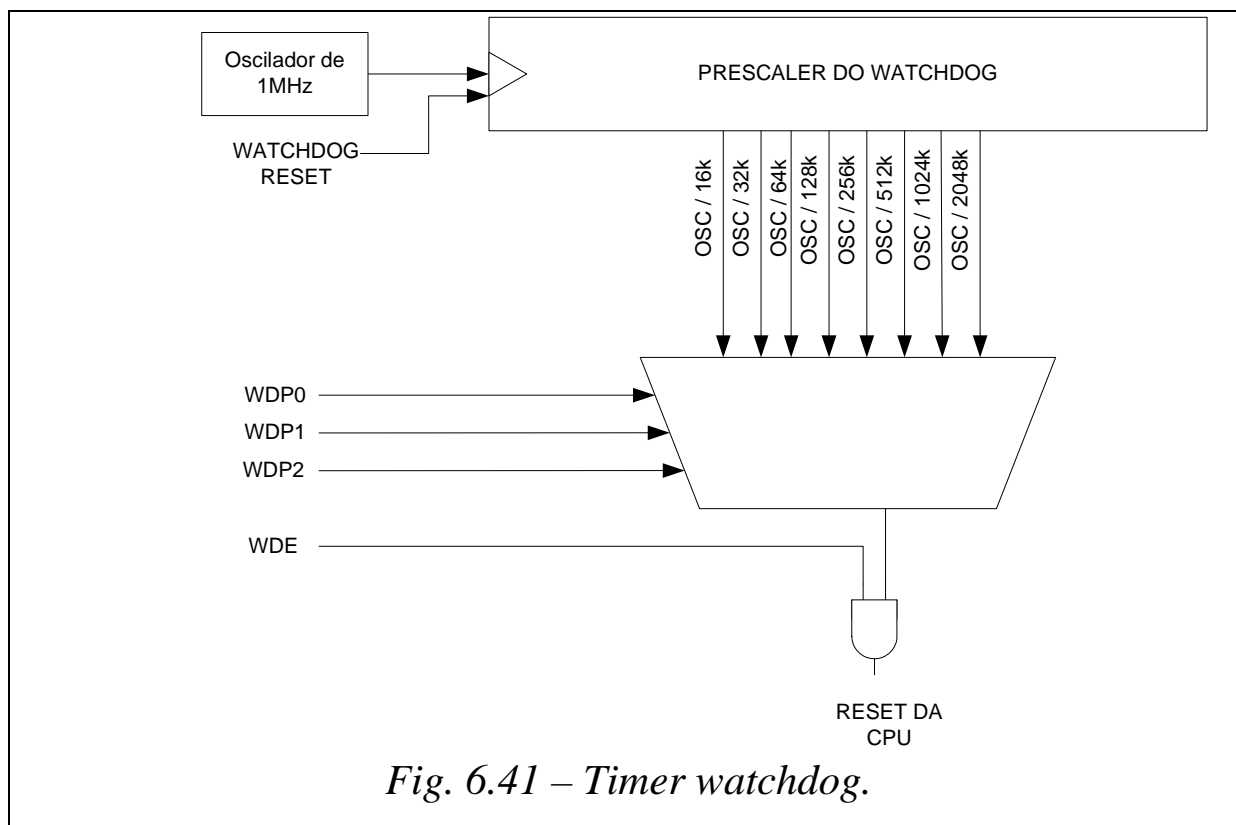
entre duas instruções WDR para prevenir o reset da CPU a partir do *watchdog*. Se o período de reset expira sem a execução de outra instrução WDR, o AT90S8515 reinicializa e inicia a execução a partir do vetor de reset. Para obter um período de reset em sintonia com o escalonamento selecionado, a instrução WDR deverá ser sempre executada antes do *watchdog* ser habilitado. A Fig. 6.41 apresenta o temporizador watchdog.

O AT90S8515 apresenta um bit para habilitar o desligamento do *Watchdog*, – o bit WDTOE . Este bit é o bit 4 do registrador WDTCR. O Registrador de Controle do Timer Watchdog (WDTCR) é apresentado na Fig. 6.42.

TABELA 6.19

SELEÇÃO DO PRESCALER DO TIMER WATCHDOG

WDP2	WDP1	WDP0	Número de ciclos de oscilador para WDR	Tempo típico de WDR com $V_{CC} = 5\text{ V}$
0	0	0	16 k	15 ms
0	0	1	32 k	30 ms
0	1	0	64 k	60 ms
0	1	1	128 k	0,12 s
1	0	0	256 k	0,24 s
1	0	1	512 k	0,49 s
1	1	0	1024 k	0,97 s
1	1	1	2048 k	1,90 s



6.12 – O comparador analógico

O comparador analógico compara os valores das entradas positiva (PB0/AIN0) e negativa (PB1/AIN1) e coloca em “1” a sua saída quando a tensão na entrada positiva é maior que a tensão na entrada negativa. A saída do comparador (ACO) pode ser configurada para acionar a interrupção do comparador analógico. O usuário pode seleccionar o acionamento da interrupção pela saída do comparador analógico por uma borda de subida ou descida ou, ainda, por um pulso. O Comparador Analógico do AT90S8515 pode ser configurado para acionar a função de Entrada de Captura do Timer 1. O diagrama de blocos do comparador analógico e sua lógica adjacente são mostrados na Fig. 6.43. O Registrador de Estado e de Controle do Comparador Analógico (ACSR) é apresentado na Fig. 6.44.

Registrador WDTCR– endereço \$21 ; **Valor na inicialização** = 00000000b

	-	-	-	-	WDE	WDP2	WDP1	WDP0
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
Bits 7, 6 e 4	Reservados, sempre lidos como zero.							
WDTOE	Bit de habilitação do desligamento do Watchdog. Este bit deve ser posto em um lógico quando WDE é posto em zero lógico, caso contrário o <i>watchdog</i> não será desabilitado. Uma vez em um lógico, o hardware colocará este bit em zero lógico após quatro ciclos de <i>clock</i> .							
WDE	Bit de Habilitação do Watchdog. Quando posto em 1 lógico habilita o timer <i>watchdog</i> ; se posto em 0 lógico desabilita-o. WDE só pode ser posto em zero lógico se o bit WDTOE estiver em um lógico. Para desabilitar um <i>Watchdog</i> habilitado, o seguinte procedimento deve ser seguido: <ol style="list-style-type: none">1. Na mesma operação, escrever um lógico em WDTOE e WDE. O um lógico deve ser escrito em WDE mesmo que ele esteja em um lógico antes da operação de desabilitação iniciar.2. Dentro de quatro ciclos de clock, escrever zero lógico em WDE.							
WDP2 ..0	Estes bits determinam o escalonamento do timer watchdog quando este é habilitado. Os valores diferentes de prescaler e seus correspondentes períodos de reinicialização foram mostrados na Tabela 6.19.							

Fig. 6.42 – Registrador WDTCR.

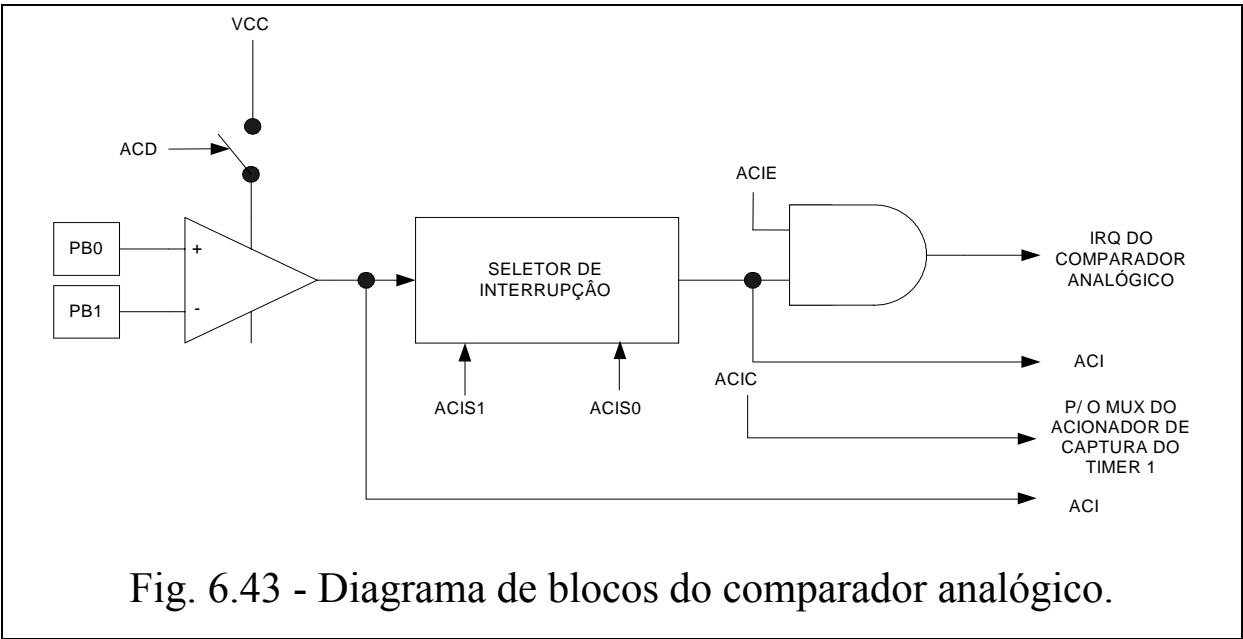


Fig. 6.43 - Diagrama de blocos do comparador analógico.

Registrador ACSR– endereço \$08 ; **Valor na inicialização** = 00φ00000b

	ACD	-	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
ACD	Bit de Desabilitação do Comparador Analógico. Com este bit em 1 lógico, a alimentação do comparador analógico está interrompida. Este bit pode ser posto em 1 lógico a qualquer momento, desligando o comparador, ação que reduzirá o consumo de energia nos modos Ativo e Idle. Ao se alterar o bit ACD, para evitar uma interrupção, a interrupção por comparador analógico deve ser desabilitada pondo em 0 lógico o bit ACIE, também do registrador ACSR.							
Bits 6	Reservado, sempre lido como zero.							
ACO	Saída do comparador analógico.							
ACI	Flag de Interrupção por Comparador Analógico. Este bit vai a 1 lógico quando um evento na saída do comparador analógico aciona o modo de interrupção definido por ACIS1 e ACIS0. A rotina de interrupção do comparador analógico é executada se o bit ACIE deste registrador e o bit I do SREG estão em 1 lógico, sendo que o flag ACI é posto em 0 lógico pelo hardware quando da execução da rotina. Alternativamente, o bit ACI pode ser posto em 0 lógico escrevendo-se um 1 lógico no flag. Observe, entretanto, que se outro bit deste registrador é modificado por uma instrução SBI ou CBI, o bit ACI será limpo se tornou-se um lógico antes da operação.							
ACIE	Bit de Habilitação da Interrupção por Comparador Analógico. Quando o bit ACIE e o bit I do SREG são postos em 1 lógico, a interrupção do comparador analógico é ativada.							
ACIC	Bit de Habilitação da Entrada de Captura por Comparação Analógica. Quando em um lógico, este bit habilita que a função de entrada de captura do Timer 1 seja acionada pelo Comparador Analógico. A saída do comparador é, neste caso, conectada diretamente a lógica de Entrada de Captura, conforme ilustra a Fig. 6.47. Quando em zero lógico, não há conexão entre o Comparador Analógico e a Entrada de Captura. Para fazer o comparador acionar a interrupção da Entrada de Captura do Timer 1, o bit TICIE1 no registrador TIMSK deve estar em um lógico.							
ACIS1 ACIS0	Estes bits determinam qual evento do comparador aciona a interrupção do comparador analógico, conforme Tabela 6.20.							

Fig. 6.44 – Registrador ACSR.

TABELA 6.20
CONFIGURAÇÃO DE ACIS1/ACIS0

ACIS1	ACIS0	Modo de Interrupção
0	0	Interrupção por pulso
0	1	Reservado
1	0	Interrupção por borda de descida
1	1	Interrupção por borda de subida

Observação: Ao alterar estes bits, para evitar uma interrupção, a interrupção do comparador analógico deve ser desabilitada colocando-se seu bit de habilitação da interrupção em 0 lógico.

6.13 – UART

O AT90S8515 possui uma UART (Transmissor e Receptor Assíncrono Universal) full-duplex com as características listadas abaixo. Os diagramas de blocos do transmissor e do receptor da UART são mostrados, respectivamente, nas Figs. 6.45 e 6.46.

- Gerador de taxa de comunicação com capacidade de gerar um grande número de taxas de comunicação;
- Altas taxas de comunicação com cristal de baixa frequência;
- Dados de 8 e 9 bits;
- Filtragem de ruído;
- Detecção de perda de dado;
- Detecção de erro de composição (falha na detecção do bit de parada);
- Detector de bit de início falso;

- Três interrupções separadas: transmissão completa, registro de dados vazio e recepção completa.

6.13.1 – Transmissão de dados

A transmissão é iniciada ao se escrever o dado a ser transmitido no Registrador de E/S de Dados da UART – UDR. O dado é transferido de UDR para o registrador de deslocamento de transmissão quando:

- Um novo caracter é escrito na UDR após o bit de parada do caracter anterior ter sido deslocado para a saída;
- Um novo caracter é escrito na UDR antes do bit de parada do caracter anterior ter sido deslocado para a saída. O registrador de deslocamento é carregado quando o bit de parada do caracter em transmissão for deslocado para a saída.

Se o registrador de deslocamento de 10(11) bits do transmissor está vazio, o dado é transferido do registrador UDR para o registrador de deslocamento. Neste momento o bit UDRE (Registrador de Dados da UART Vazio) no Registrador de Estado da UART (USR) é posto em um lógico. Quando esse bit está em um lógico, a UART está pronta para receber o próximo caracter. Ao mesmo tempo em que o dado é transferido de UDR para o registrador de deslocamento de 10(11) bits, o bit 0 (bit de início) do registrador de deslocamento é limpo e o bit 9 ou 10 (bit de parada) é posto em um

lógico. Se a palavra de 9 bits é selecionada (o bit CHR9 no Registrador de Controle da UART, UCR, está em um lógico), o bit TXB8 em UCR é transferido para o bit 9 do registrador de deslocamento de transmissão.

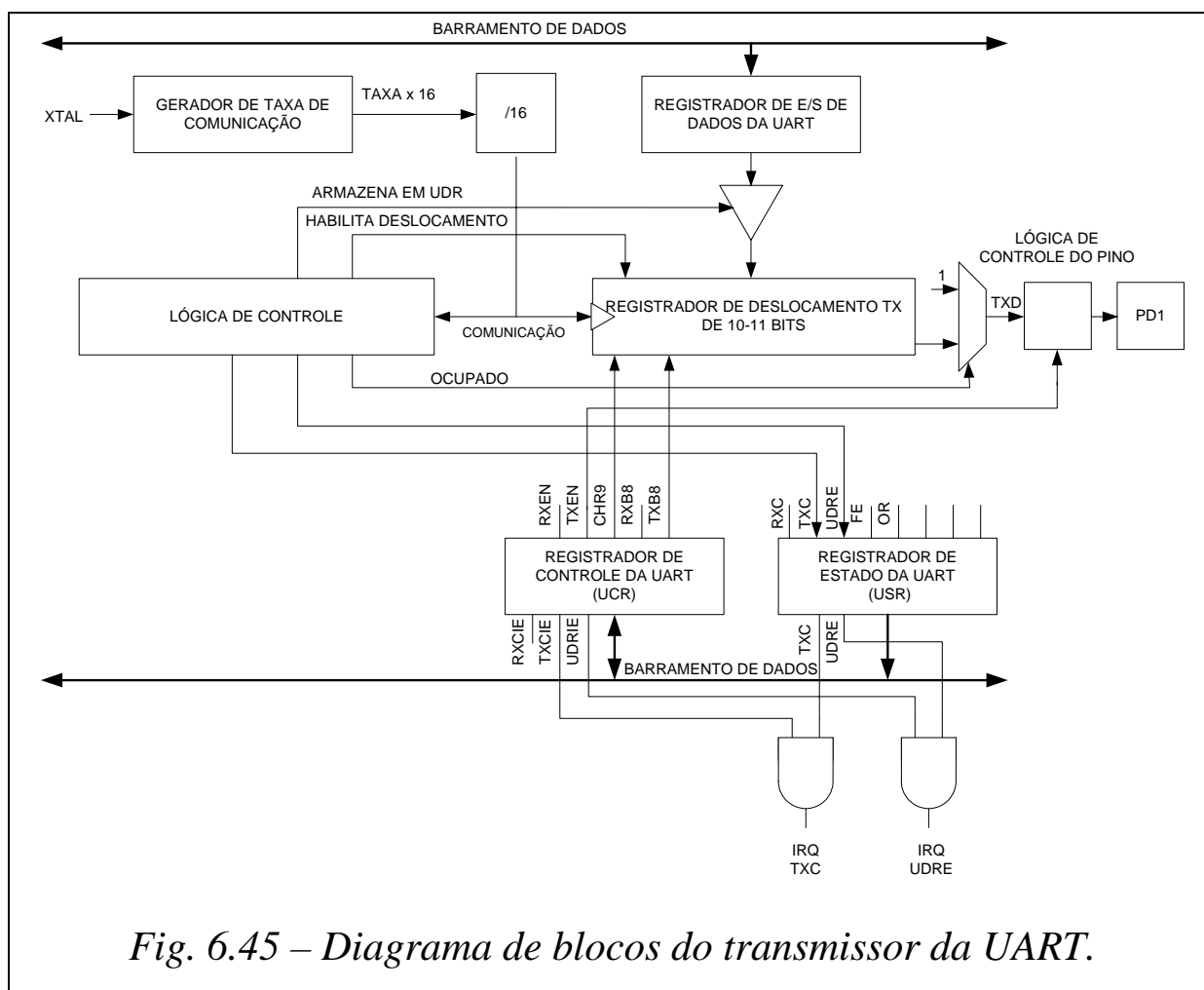
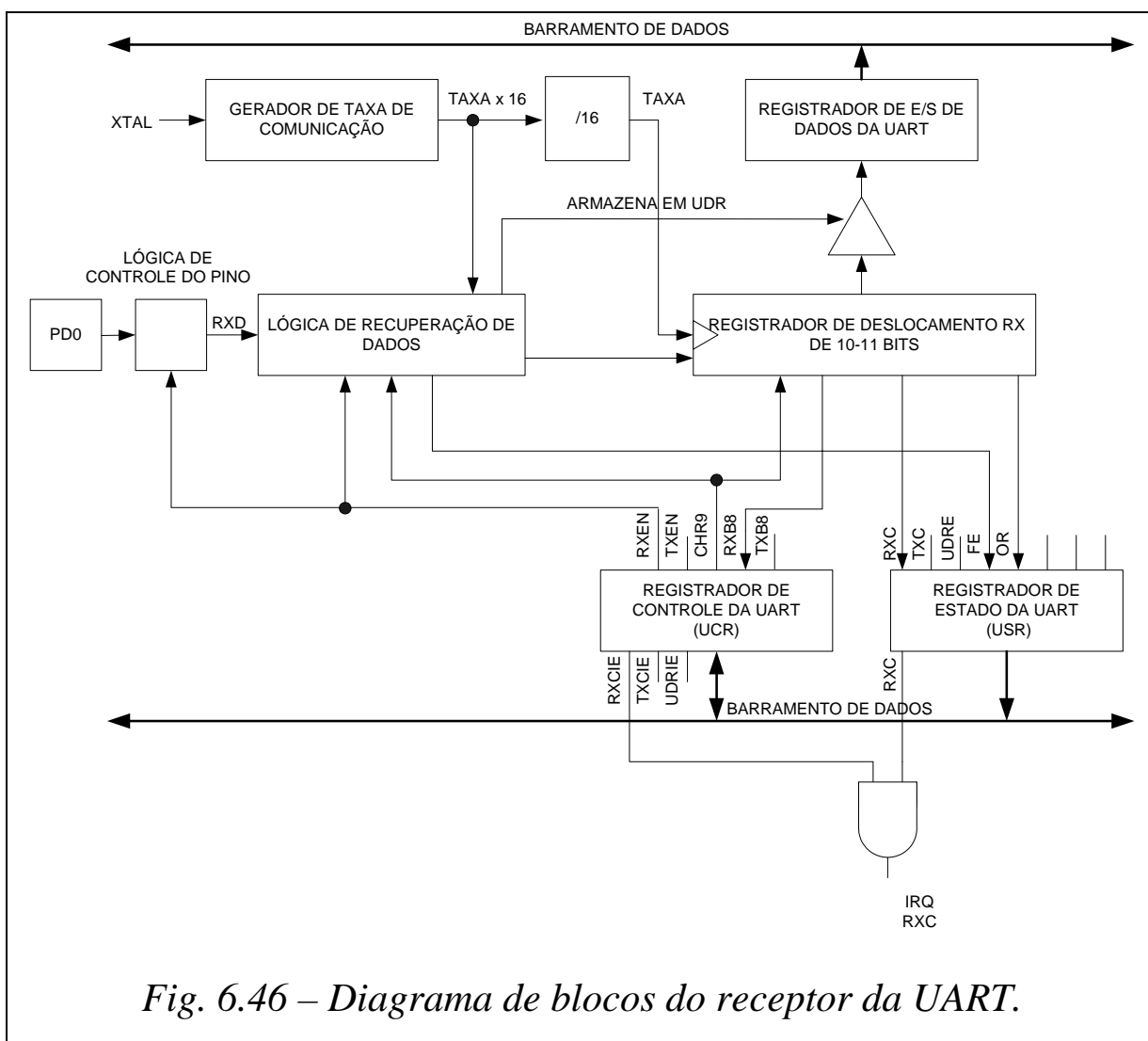


Fig. 6.45 – Diagrama de blocos do transmissor da UART.

No clock de taxa de comunicação seguinte a operação de transferência para o registrador de deslocamento, o bit de início é deslocado para o pino TXD, seguindo o dado com o LSB primeiro. Quando o bit de parada é deslocado para a saída, o registrador de deslocamento é carregado se algum novo dado foi escrito no UDR durante a transmissão. Durante a carga, UDRE é posto em um lógico. Se não há nenhum dado novo no registrador UDR para ser enviado

quando o bit de parada é deslocado, o flag UDRE permanecerá em um lógico até UDR ser escrito novamente. Se nenhum dado novo foi escrito é o bit de parada estiver presente em TXD pela largura de um bit, o flag de transmissão completa (TXC) em USR é posto em “1”. TXEN no registrador UCR habilita o transmissor da UART quando em um lógico. Se esse bit é posto em zero lógico, o pino PD1 pode ser usado para E/S de dados. Quando TXEN é posto em um lógico, o transmissor da UART é conectaqdo a PD1, o qual é forçado a ser um pino de saída independente da configuração do bit DDD1 em DDRD.



6.13.2 – Recepção de dados

O circuito lógico do receptor amostra o sinal no pino RXD em uma frequência 16 vezes maior que a taxa de comunicação. Enquanto a linha está inativa, uma única amostra em zero lógico será interpretada como a borda de descida de um bit de início e a sequência de detecção do bit de início é iniciada. Deixar de amostrar um significa o primeiro zero amostrado. Seguindo a transição de um para zero, o receptor amostra o pino RXD nas amostras 8,9 e 10. Se duas ou mais destas três amostras forem um lógico, o bit de início é rejeitado como um ruído e o receptor inicia a busca pela próxima transição de um para zero. Se, entretanto, um bit de início válido é detectado, é realizada a amostragem dos bits de dados seguintes ao bit de início. Esses bits são amostrados nas amostras 8, 9 e 10. O valor lógico encontrado em pelo menos duas das três amostras é tomado como o valor do bit. Todos os bits são deslocados para o registrador de deslocamento conforme eles são amostrados.

Quando o bit de parada entra no receptor, a maioria das três amostras deve ser um lógico para o bit de parada ser aceito. Se duas ou mais amostras são zero lógico, o flag de erro de composição (FE) no Registrador de Estado da UART é posto em um lógico. Antes de ler o registrador UDR o usuário deveria sempre checar o bit FE para detectar erros de composição.

Seja detectado ou não, ao fim de um ciclo de recepção de caracter, um bit válido de parada, o dado é transferido para o UDR e o flag RXC em USR é posto em um lógico. O registrador UDR é, na

realidade, dois registradores fisicamente separados, um para o dado transmitido e outro para o dado recebido. Quando o registrador UDR é lido, o registrador de recepção é acessado, e quando UDR é escrito o registrador de transmissão de dados é acessado. Se a palavra de dados de 9 bits é selecionada (o bit CHR9 do registrador UCR está em um lógico), o bit RXB8 do registrador UCR é carregado com o nono bit do registrador de deslocamento de transmissão quando o dado é transferido para UDR.

Se um caracter foi recebido e o registrador UDR não foi lido na recepção anterior, o flag de sobreposição (OR) em USR é posto em um lógico. Isto significa que o último byte de dados deslocado para o registrador de deslocamento não pôde ser transferido para UDR e foi perdido. O bit OR é mantido e é atualizado quando o byte de dados válido em UDR é lido. Então, o usuário deveria conferir sempre o bit OR antes de ler o registrador UDR para detectar alguma perda de dados por taxa de comunicação inadequada ou por CPU carregada.

Quando o bit RXEN no registrador UCR está em zero lógico, a recepção está desabilitada. Isto significa que o terminal PD0 pode ser usado como um pino de E/S. Quando RXEN está em um lógico, o receptor da UART está conectado a PD0, o que o força a ser uma entrada, independente da configuração do bit DDD0 em DDRD. Quando PD0 é forçado como entrada pela UART, o PORTD pode ainda ser usado para controlar o resistor de pull-up do pino.

Quando o bit CHR9 no registrador UCR está em um lógico os caracteres transmitidos e recebidos têm extensão de 9 bits, mais os

bits de início e parada. O nono bit a ser transmitido é o bit TXB8 do registrador UCR. Este bit deve ser posto no valor desejado antes que a transmissão seja iniciada pela escrita no registrador UDR. O nono bit recebido é o RXB8 do registrador UDR.

6.13.3 – Controle da UART

Conforme descrito anteriormente, o registrador UDR é, na realidade, dois registradores fisicamente distintos que compartilham o mesmo endereço de I/O (\$0C) Quando se escreve no registrador, escreve-se no registrador de transmissão de dados da UART; quando ele é lido, lê-se o registrador de recepção de dados da UART.

Os registradores de Estado (USR) e de Controle (UCR) da UART são apresentados nas Figs. 6.47 e 6.48.

6.13.4 – Gerador de Taxa de Comunicação

O gerador de taxa de comunicação é um divisor de frequência que produz taxas de acordo com a equação

$$TAXA = \frac{f_{ck}}{16(UBRR + 1)}$$

onde f_{ck} = frequência do cristal e UBRR é o conteúdo do Registrador de Taxa de Comunicação da UART, UBRR (\$09).

A Tabela 6.21 mostra os valores de UBRR que geram as taxas de comunicação mais utilizadas para um cristal de 3,6864 MHz..

Registrador USR– endereço \$0B ; **Valor na inicialização** = 00100000b

	RXC	TXC	UDRE	FE	OR	-	-	-
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
RXC	Recepção da UART Completa. Este bit é posto em um lógico quando um caractere recebido é transferido do registrador de deslocamento de recepção para UDR. Este bit é posto em um lógico independente da detecção de erros de composição. Quando o bit RXEN no USR está em um lógico, a interrupção de Recepção da UART Completa será executada enquanto RXC for um. O bit RXC é posto em zero lógico na leitura de UDR. Quando uma interrupção acionada pela recepção de dados é usada, a rotina de interrupção de Recepção da UART Completa deve ler UDR a fim de limpar RXC, se não o fizer, uma nova interrupção ocorrerá a cada vez que a rotina de interrupção terminar.							
TXC	Transmissão da UART Completa. Este bit é posto em um lógico quando um caractere completo (incluindo o bit de parada) no registrador de Deslocamento de Transmissão for transferido e nenhum dado foi escrito em UDR. Este flag é especialmente útil em interfaces de comunicação half-duplex, onde um aplicativo de transmissão deve entrar no modo de recepção e liberar o barramento de comunicação imediatamente após completar a transmissão. Quando o bit TXEN no USR está em 1 lógico, a interrupção por Transmissão da UART Completa será executada quando TXC for para um. O bit TXC é posto em zero lógico pelo hardware ao executar o vetor correspondente de tratamento de interrupção. Alternativamente, o bit TCX é limpo na escrita de 1 lógico no bit.							
UDRE	Registrador de Dados da UART Vazio. Este bit é posto em 1 lógico quando um caractere escrito no UDR é transferido para o registrador de Deslocamento de Transmissão. Um lógico neste bit indica que o transmissor está pronto para receber um novo caractere para transmissão. Quando o bit UDRIE no USR está em um lógico, a interrupção por Registrador de Dados da UART Vazio será executada enquanto UDRE estiver em um. O bit UDRE é posto em 0 lógico na leitura de UDR. Quando uma interrupção acionada pela transmissão de dados é usada, a rotina de interrupção por Registrador de Dados da UART Vazio deve escrever em UDR a fim de limpar UDRE, se não o fizer, uma nova interrupção ocorrerá a cada vez que a rotina de interrupção terminar. UDRE é posto em um lógico durante o reset para indicar que o transmissor está pronto.							
FE	Erro de Composição. Este bit é posto em um lógico se uma condição de erro de composição é detectada, isto é, quando o bit de parada de um caracter recebido é zero. FE é limpo quando o bit de parada do dado recebido é um.							
OR	Sobreposição. Este bit é posto em um lógico se uma condição de sobreposição é detectada, isto é, quando um caracter do registrador UDR não for lido antes do próximo caracter ter sido deslocado para o registrador de Deslocamento do Receptor. O bit OR é mantido, o que significa que ele manter-se-á em um lógico mesmo que o dado válido em UDR seja lido. Ele é limpo quando um dado é recebido e transferido para UDR.							
Bits 2..0	Reservados, sempre lidos como zero.							

Fig. 6.47 – Registrador USR do AT90S8515.

Registrador UCR– endereço \$0A ; **Valor na inicialização** = 00000000b

	RXC	TXC	UDRE	FE	OR	-	-	-
bit	7	6	5	4	3	2	1	0
Símbolo	Função							
RXCIE	Habilita Interrupção por Recepção da UART Completa. Quando este bit está em um lógico, a ida para um lógico de RXC em USR causará a execução da rotina de Interrupção por Recepção da UART Completa caso as interrupções estejam habilitadas (bit I em SREG em um lógico).							
TXCIE	Habilita Interrupção por Transmissão da UART Completa. Quando este bit está em um lógico, a ida para um lógico de TXC em USR causará a execução da rotina de Interrupção por Transmissão da UART Completa caso as interrupções estejam habilitadas.							
UDRIE	Habilita Interrupção por Registrador de Dados da UART Vazio. Quando este bit está em um lógico, a ida para um lógico de UDRE em USR causará a execução da rotina de Interrupção por Registrador de Dados da UART Vazio caso as interrupções estejam habilitadas.							
RXEN	Habilita Recepção. Este bit habilita o receptor da UART quando em um lógico. Quando o receptor está desabilitado, os flags de estado RXC, OR e FE não serão postos em um lógico. Caso eles estejam em um lógico, a desabilitação de RXEN não os colocará em zero lógico.							
TXEN	Habilita Transmissão. Este bit habilita o transmissor. Se o transmissor é desabilitado enquanto um caractere é transmitido, o transmissor não é desabilitado antes dos caracteres no registrador de deslocamento e no UDR terem sido completamente transmitidos.							
CHR9	Caracteres de 9 bits. Quando este bit está em um lógico os caracteres transmitidos e recebidos são de 9 bits mais os bits de início e parada. O nono bit é lido e escrito usando-se, respectivamente, os bits RXB8 e TXB8 em UCR.							
RXB8	Bit 8 Recebido. Quando CHR9 é um lógico, RXB8 é o nono bit do caracter recebido.							
TXB8	Bit 8 Transmitido. Quando CHR9 é um lógico, TXB8 é o nono bit do caracter transmitido.							

Fig. 6.48 – Registrador UCR do AT90S8515.

TABELA 6.21
VALORES DE UBRR PARA UM CRISTAL DE 3,6864 MHz

Taxa de Comunicação	UBRR
2400	95
4800	47
9600	23
14400	15
19200	11
28800	7
38400	5
57600	3
76800	2
115200	1

No código apresentado seguir, transmite-se o caracter anteriormente recebido pela UART acrescido de uma unidade. A taxa de comunicação é configurada para 2400 bps. O fluxograma da Fig. 6.49 representa o código a seguir.

```

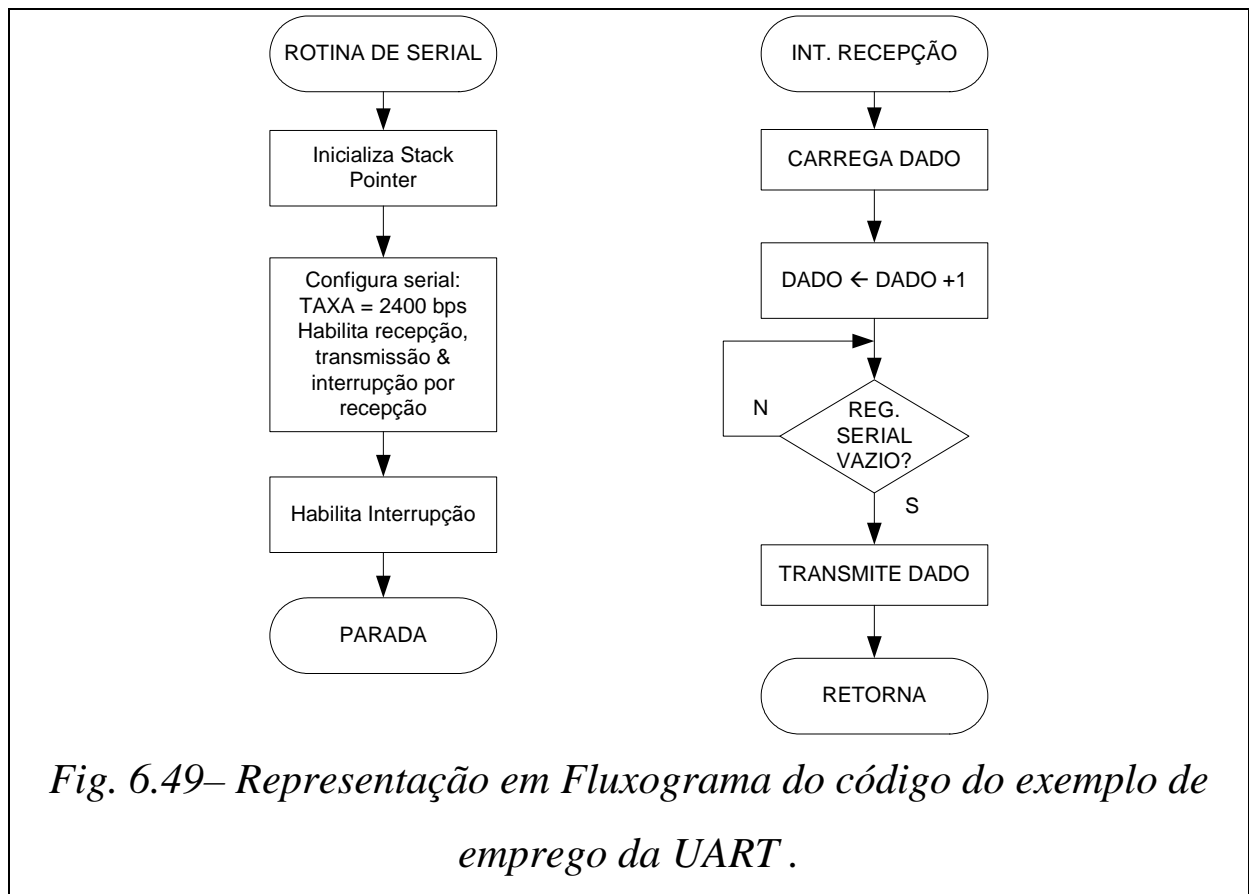
;*****
; EMPREGO DA UART
;*****
; DEFINIÇÕES
;
.include "8515def.inc"
.def temp      = r17
;*****
;
.cseg
.org 0x00
        rjmp main
;*****
;
; Vetores de Interrupção
.org 0x09
;
; tratamento da recepção serial

```

```

        in temp,udr
        inc temp
        aguarda:sbis usr,5           ;aguarda udr ser transferido
        rjmp aguarda                ;para reg. de deslocamento
        out udr,temp
        nop
        reti
;*****
;
; Programa Principal
;
main:    ldi temp,high(ramend)
        out sph,temp
        ldi temp,low(ramend)
        out spl,temp
        ldi temp,95                 ;define 2400 bps
        out ubrr,temp
        ldi temp,$98;habilita recepção/transmissão
        out ucr,temp;e int. p/ rec. completa
        sei                         ;hab. interrupções
stop:    rjmp stop
;*****

```



6.14 – Modos redução de consumo (sleep modes)

Para acessar os modos de redução de consumo, o bit SE no MCUCR (Fig. 6.26) deve ser posto em 1 lógico e uma instrução SLEEP deve ser executada. Se uma interrupção habilitada ocorrer enquanto a MCU está em um modo de redução de consumo, a MCU acorda, executa a rotina de interrupção e reinicia a execução a partir da instrução seguinte a SLEEP. Os conteúdos dos registradores e do espaço de memória de E/S não são alterados. Se um Reset ocorre durante um modo de redução de consumo, a MCU acorda e executa a partir do vetor de reset.

6.14.1 – Modo de espera (idle)

Quando o bit SM do MCUCR está em zero lógico, a instrução SLEEP faz a MCU entrar no modo de espera, parando a CPU, permitindo, porém, que o Timer 0, o timer watchdog e o sistema de interrupção continuem operando. Isto habilita a MCU acordar a partir de interrupções externas e internas. Se não é solicitado que a MCU acorde a partir da interrupção do comparador analógico, este pode ser desligado pondo em 1 lógico o bit ACD no registrador ACSR, reduzindo-se, desta forma, o consumo no modo de espera. Quando a MCU acorda a partir de um modo de espera, a CPU executa a execução do programa imediatamente.

6.14.2 – Modo de baixo consumo (power-down)

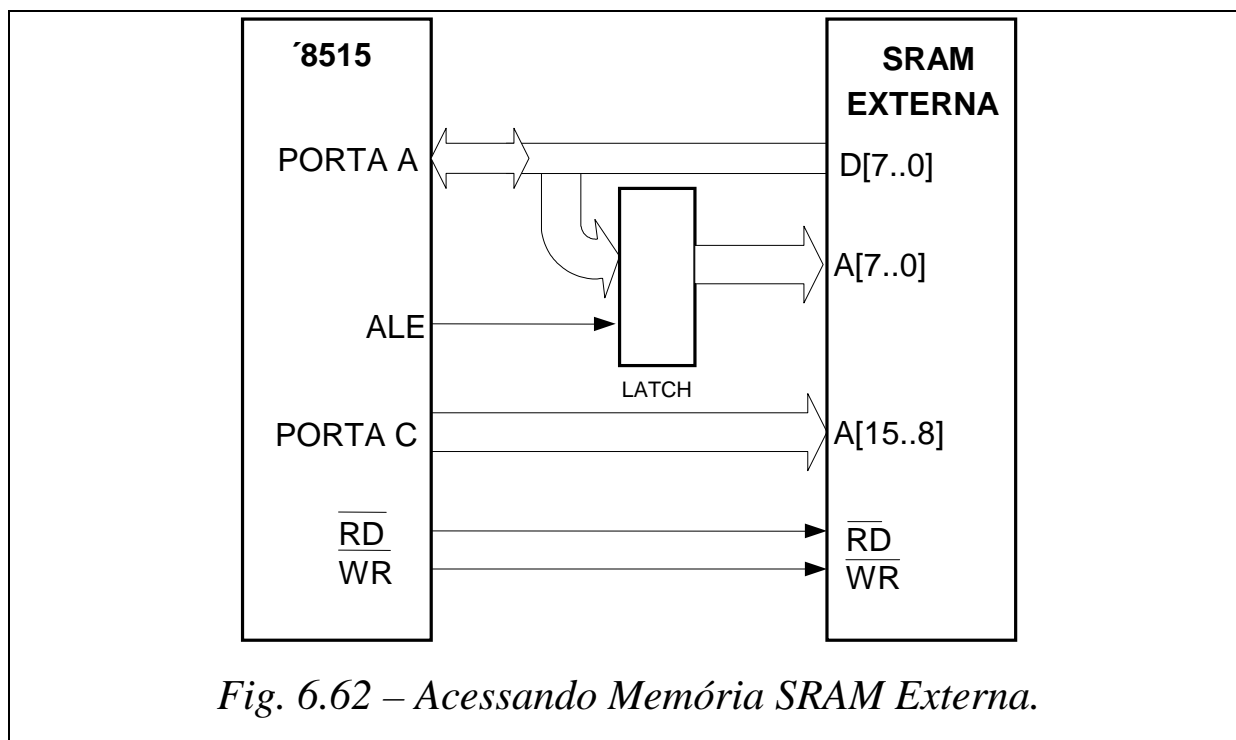
Quando o bit SM do MCUCR está em 1 lógico, a instrução SLEEP faz a MCU entrar no modo baixo consumo. Neste modo, o oscilador externo é desativado, enquanto as interrupções externas e o timer watchdog (se habilitado) continuam operando. Somente um reset externo ou por watchdog e uma interrupção externa por nível em INT0 podem acordar a MCU. A folha de dados chama a atenção que quando uma interrupção externa por nível é usada para acordar a MCU do modo de baixo consumo, o nível baixo deve permanecer por um tempo maior que o período time-out de atraso do reset, cerca de 16 ms para $V_{cc} = 5\text{ V}$.

6.15 – Acesso a Memória de Dados SRAM Externa

A configuração de Hardware necessária para acessar uma memória SRAM externa é apresentada na Fig. 6.50. A Porta A serve como um barramento multiplexado de dados e byte menos significativo de endereço. O sinal ALE (*Address Latch Enable*) trava este byte em um *latch* de endereços. Na borda de descida do sinal ALE, está disponível na porta A um endereço válido. Durante a transferência de dados ALE é baixo. A Porta C emite o byte mais significativo do endereço. Os sinais \overline{RD} (leitura) e \overline{WR} (escrita) são gerados pela CPU quando necessários.

O acesso a SRAM externa é habilitado pondo em um lógico o bit SRE do registrador MCUCR (Fig. 6.26), ação que anulará a

configuração dos registradores DDRA e DDRC. Quando o bit SER é posto em zero lógico, o acesso a SRAM externa é desabilitado e as configurações correntes dos pinos e da direção dos dados são usadas.



6.16 – Programação em C com o AVR AT90S8515

Para elaborar programas em C para o AT90S8515 utilizaremos o software de código aberto *WinAVR*, disponível no sítio www.sourceforge.net. Os programas fontes em C serão escritos utilizando-se a ferramenta *Programmers Notepad*, ferramenta incluída no pacote de distribuição do *WinAVR*.

Nosso primeiro exemplo será um sequencial de LEDS implementável em nossa placa de desenvolvimento. Para a elaborar um programa em C com o *WinAVR* que possa ser simulado no *AVRStudio* devem ser seguidos os seguintes passos:

1. Criar uma pasta “sequencial” para o exemplo;
2. Copiar o arquivo *makefile* do diretório *../WinAVR/Samples* para a pasta *../sequencial*;
3. Abrir o *Programmers Notepad*;
4. No *Programmers Notepad* abrir o menu *Tools* e selecionar *Options*;
5. Na caixa *Options* selecionar *Tools* e na sequência *Add*;
6. Dê a seguinte aparência a caixa *Propriedades de New Tool*:
 - Name : [WinAVR]Make Extcoff
 - Command: make.exe
 - Folder: %d
 - Parameters: extcoff
 - Shortcut: None
 - Save: Current File
7. Pressione OK duas vezes;
8. Selecione *File – New – C/C++*, nomeie o arquivo como *sequencial.c* e digite exatamente o que segue:

```
// sequencial.c
#include <avr/io.h>
#include <avr/delay.h>

int main (void)
{
    //porta D como saída
    DDRD = 0xFF;
    while (1)
    {
        for(int i = 1; i <= 128; i = i*2)
```

```
{
PORTD = i;
_delay_loop_2(65535);
// declarada em delay.h
}
}
return 1;
}
```

9. Selecione File – Save As... e no box *Nome do Arquivo*

digite sequencial.c;

10. Abra o arquivo *makefile* da pasta sequencial e

proceda as modificações em negrito:

```
# MCU name
MCU = at90s8515
```

```
# Processor frequency.
# This will define a symbol, F_CPU, in all source
code files equal to the
# processor frequency. You can then use this symbol
in your source code to
# calculate timings. Do NOT tack on a 'UL' at the
end, this will be done
# automatically to create a 32-bit value in your source
code.
F_CPU = 3686400
```

```
# Output format. (can be srec, ihex, binary)
FORMAT = ihex
```

```
# Target file name (without extension).
TARGET = sequencial
```

11. Salve as modificações de makefile;

12. Selecione *Tools – [WinAVR]Make All* para criar o arquivo sequencial.hex;

13. Selecione *Tools – [WinAVR]Make Extcoff* para criar o arquivo sequencial.coff;

Para simular o programa, abra o arquivo sequencial.c e salve-o em uma nova pasta, seqsim, com o nome seqsim.c. Após comente a linha `_delay_loop_2(65535)` e siga os procedimentos dos passos 10 a 13, observando que onde está escrito sequencial escrever-se-á seqsim. Ato contínuo, abra o *AVRStudio* e execute os seguintes passos:

1. No *AVRStudio* abra o arquivo seqsim.coff;
2. Selecione *AVR Simulator e AT90S8515*;
3. Na janela *AVRStudio Workspace* selecione *I/O AT90S8515 – PORTD*;
4. Na barra de ferramentas selecione o botão *AutoStep*;
5. Observe o sequenciamento dos bits da PORTA D.

Nosso segundo exemplo aperfeiçoa o sequencial de LEDS utilizando o timer 1 e sua saída de comparação A para gerar um intervalo de um segundo entre o acionamento de cada LED. A seguir é apresentado o código fonte em C para o nosso sequencial aperfeiçoado.

```
// timerlum

#include <avr/io.h> // contém as definições para o
dispositivo especificado
#include <avr/signal.h> // define algumas macros para
tratamento das interrupções
#include <avr/interrupt.h> // define macros para
habilitar/ desabilitar interrupções
```



```
//declara funções
void init_timer1(void);

int main(void)
{
    //porta D como saída
    DDRD = 0xFF;
    PORTD = 0x01;
    // inicialização do timer1
    init_timer1();
    //aguarda interrupção
    while (1)
    {}
    return 1;
}

// inicialização do timer1
void init_timer1(void)
{
    //carrega valores para comparação
    OCR1AH = 0xe1;
    OCR1AL = 0x00;
    //habilita interrupção por comparação A
    TIMSK = (1<<OCIE1A);
    //limpa timer1 após comparação A e prescaler = 64
    TCCR1B = (1<<CTC1)|(1<<CS11)|(1<<CS10);
    //Habilita interrupções
    sei();
}
//A interrupção ocorre a cada 1 segundo
SIGNAL(SIG_OUTPUT_COMPARE1A)
{
    PORTD = PORTD*2; // rotaciona PORTD
    if(PORTD==0)PORTD = 0x01;
}
```

Chamamos a atenção para os *includes* necessários para utilizar as interrupções, bem como para a função SIGNAL, utilizada para o tratamento das interrupções. Para tratar cada uma das

interrupções do AT90S8515 a função SIGNAL assume a forma apresentada na Tabela 6.22.

TABELA 6.22
DEFINIÇÃO DA FUNÇÃO SIGNAL PARA CADA INTERRUPÇÃO

<i>Interrupção</i>	<i>Função</i>
Externa 0	SIGNAL(SIG_INTERRUPT0)
Externa 1	SIGNAL(SIG_INTERRUPT1)
Captura do Timer/Contador1	SIGNAL(SIG_INPUT_CAPTURE1)
Coincidência de Comparação A Timer/ Contador1	SIGNAL(SIG_OUTPUT_COMPARE1A)
Coincidência de Comparação B Timer/ Contador1	SIGNAL(SIG_OUTPUT_COMPARE1B)
Estouro do Timer/ Contador1	SIGNAL(SIG_OVERFLOW1)
Estouro do Timer/ Contador0	SIGNAL(SIG_OVERFLOW0)
Transferência Serial Completa	SIGNAL(SIG_SPI)
Recepção da UART Completa	SIGNAL(SIG_UART_RECV)
Registro de Dados da UART Vazio	SIGNAL(SIG_UART_DATA)
Transmissão da UART Completa	SIGNAL(SIG_UART_TRANS)
Comparados Analógico	SIGNAL(SIG_COMPARATOR)

No WinAVR podem ser especificados arquivos fonte C adicionais para o uso em seu projeto. Por exemplo, você pode ter o arquivo `interfaces.c` que inclui as rotinas de interfaceamento acessadas no programa `main.c`. A seguir, apresenta-se as maneiras para incluir um arquivo ao *makefile*.

```
SRC = $(TARGET).c interfaces.c
```

ou

```
SRC = $(TARGET).c
```

```
SRC += interfaces.c
```

ou

```
SRC = $(TARGET).c \
```

```
interfaces.c
```

Se houver a necessidade de juntar arquivos em assembly, você pode fazê-lo da seguinte forma:

```
ASRC = assemblynome.S
```

onde o “S” da extensão deve ser maiúsculo.

Através da opção *OPT = nível* no makefile, o usuário pode alterar o nível de otimização do código gerado pelo compilador. Os níveis disponíveis são 0,1,2,3 e s. Cada nível realiza tarefas diferentes no código para torná-lo mais rápido ou menor, com exceção do nível 0, que não faz nada. O nível 3, por exemplo, cria um código maior, porém mais rápido. O código s é uma otimização para o tamanho do código, porém é um nível bem ?.

O código em C a seguir gera uma onda senoidal de frequência igual a 56,5 Hz no pino PD5. Devido ao tempo de processamento da função seno, optou-se por gerar uma tabela com a função `gera_tab()`, sendo que os elementos da tabela serão transferidos para o registrador OCR1AL a cada estouro do timer 1.

```
include <avr/io.h> // contém as definições para o
dispositivo especificado
#include <avr/signal.h> // define algumas macros para
tratamento das interrupções
```

```
#include <avr/interrupt.h> // define macros para
habilitar/ desabilitar interrupções
#include <math.h> // declara as funções matemáticas
básicas e funções

//declara funções
void init_timer1(void);
void gera_tab(void);
//
#define PI2X 6.28318
//
typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned long DWORD;
//
BYTE amostra = 0;
double ARG;
unsigned char pontos[128];

int main(void)
{
    // inicialização do timer1
    gera_tab();
    init_timer1();
    //aguarda interrupção
    while (1)
    {}
    return 1;
}

// inicialização do timer1
void init_timer1(void)
{
    //habilita interrupção por estouro do timer 1
    TIMSK = (1<<TOIE1);
    //saída de comp. A ligada a PD5 e PWM de 8 bits
ativo
    TCCR1A = (1<<COM1A1)|(1<<PWM10);
    //timer 1 ligado com fonte clk
    TCCR1B = (1<<CS10);
    //Habilita interrupções
    sei();
}
// gera tabela
```

```
void gera_tab(void)
{
    for(amostra=0;amostra<128;amostra++)
    {
        ARG = amostra * PI2X / 128;
        pontos[amostra]=(BYTE)(127UL + 127UL * sin(ARG));
    }
    amostra = 0;
}
//A interrupção ocorre a cada estouro do timer 1
SIGNAL(SIG_OVERFLOW1)
{
    OCR1AL = pontos[amostra];
    amostra++;
    if(amostra == 128)amostra = 0;
}
```

REFERÊNCIAS BIBLIOGRÁFICAS

ATMEL. **8 bit AVR Instruction set.** Documento disponível eletronicamente no site www.atmel.com.

ATMEL. **AT89 Series Hardware Description.** Documento disponível eletronicamente no site www.atmel.com.

ATMEL. **AT89S8252 Data Sheet.** Documento disponível eletronicamente no site www.atmel.com.

ATMEL. **AT89S8253 Data Sheet.** Documento disponível eletronicamente no site www.atmel.com.

ATMEL. **AVR Hardware design considerations.** Documento disponível eletronicamente no site www.atmel.com.

ATMEL. **C51 Family Programmer Guide and Instruction Set.** Documento disponível eletronicamente no site www.atmel.com.

ATMEL. **Flash Microcontroller - Architecture Overview.** Documento disponível eletronicamente no site www.atmel.com.

ATMEL. **Flash Microcontroller – Memory Organization.** Documento disponível eletronicamente no site www.atmel.com.

CATSOULIS, John. **Designing Embedded Hardware.** O'Reilly: USA, 2002.

TOCCI, Ronald. **Sistemas Digitais.** São Paulo: São Paulo: Pearson Education do Brasil, 2003.

ANEXO I

*CONJUNTO DE INSTRUÇÕES DA FAMÍLIA MCS-51™***Instruções que afetam os Flags**

Instrução	Flag			Instrução	Flag		
	C	OV	AC		C	OV	AC
ADD	?	?	?	CLR C	0		
ADDC	?	?	?	CPL C	?		
SUBB	?	?	?	ANL C,bit	?		
MUL	0	?		ANL C,/bit	?		
DIV	0	0 ¹⁴		ORL C,bit	?		
DA	?			ORL C,/bit	?		
RRC	?			MOV C,bit	?		
RLC	?			CJNE	?		
SETB C	1						

O Conjunto de Instruções e os modos de endereçamento

R_n	Registrador R0 – R7 do banco de registradores corrente
direto	Endereço da memória de dados interna de 8 bits. Pode ser um endereço da RAM de dados interna (0-127) ou um Registrador de Função Especial (128-255)
@R_i	Endereço da RAM de dados interna de 8 bits (0-255) endereçada indiretamente pelos registradores R ₀ ou R ₁
#dado	Constante de 8 bits incluída em uma instrução
#dado 16	Constante de 16 bits incluída em uma instrução
end 16	Endereço de 16 bits. Usado por LCALL e LJMP. O desvio pode ser para qualquer uma das posições da memória de programa
end 11	Endereço de 11 bits. Usado por ACALL e AJMP. O desvio será dentro do mesmo bloco de 2 kbytes da memória de programa ao qual pertence o primeiro

¹⁴ Exceção: Se B = 0, o flag de overflow vai para nível lógico 1

O Conjunto de Instruções e os modos de endereçamento

	byte da instrução seguinte
rel	Byte de offset de 8 bits em complemento de dois. Usado por SJMP e todos os desvios condicionais. A faixa de desvio é de -128 a +127 bytes relativos ao primeiro byte da instrução seguinte
bit	Bit endereçado diretamente na RAM de dados interna ou em um Registrador de Função Especial

Mnemônico		Descrição	Byte	Períodos de clock
OPERAÇÕES ARITMÉTICAS				
ADD	A,R _n	Soma registrador ao acumulador	1	12
ADD	A,direto	Soma posição de memória ao acumulador	2	12
ADD	A,@R _i	Soma posição de memória indicada por R _i ao acumulador	1	12
ADD	A,#dado	Soma dado imediato ao acumulador	2	12
ADDC	A,R _n	Soma registrador e o carry ao acumulador	1	12
ADDC	A,direto	Soma posição de memória e o carry ao acumulador	2	12
ADDC	A,@R _i	Soma posição de memória indicada por R _i e o carry ao acumulador	1	12
ADDC	A,#dado	Soma dado imediato e o carry ao acumulador	2	12
SUBB	A,R _n	Subtrai registrador e o borrow do acumulador	1	12
SUBB	A,direto	Subtrai posição de memória e o borrow do acumulador	2	12
SUBB	A,@R _i	Subtrai posição de memória indicada por R _i e o borrow do acumulador	1	12
SUBB	A,#dado	Subtrai dado imediato e o borrow o acumulador	2	12
INC	A	Soma um ao acumulador	1	12
INC	R _n	Soma um ao registrador	1	12
INC	direto	Soma um a posição de memória	2	12
INC	@R _i	Soma um à posição de memória indicada por R _i	1	12
DEC	A	Subtrai um do acumulador	1	12
DEC	R _n	Subtrai um do registrador	1	12
DEC	direto	Subtrai um da posição de memória	2	12
DEC	@R _i	Subtrai um da posição de memória indicada por R _i	1	12
INC	DPTR	Soma um ao DPTR	1	24

Mnemônico		Descrição	Byte	Períodos de clock
MUL	AB	Multiplica A e B	1	48
DIV	AB	Divide A por B	1	48
DA	A	Ajuste decimal do acumulador	1	12
OPERAÇÕES LÓGICAS				
ANL	A,R _n	Lógica E entre registrador e acumulador, com resultado no acumulador	1	12
ANL	A,direto	Lógica E entre posição de memória e acumulador, com resultado no acumulador	2	12
ANL	A,@R _i	Lógica E entre posição de memória indicada por R _i e acumulador, com resultado no acumulador	1	12
ANL	A,#dado	Lógica E entre dado imediato e acumulador, com resultado no acumulador	2	12
ANL	direto,A	Lógica E entre posição de memória e acumulador, com resultado na posição de memória	2	12
ANL	direto,#dado	Lógica E entre posição de memória e dado imediato, com resultado na posição de memória	3	24
ORL	A,R _n	Lógica OU entre registrador e acumulador, com resultado no acumulador	1	12
ORL	A,direto	Lógica OU entre posição de memória e acumulador, com resultado no acumulador	2	12
ORL	A,@R _i	Lógica OU entre posição de memória indicada por R _i e acumulador, com resultado no acumulador	1	12
ORL	A,#dado	Lógica OU entre dado imediato e acumulador, com resultado no acumulador	2	12
ORL	direto,A	Lógica OU entre posição de memória e acumulador, com resultado na posição de memória	2	12
ORL	direto,#dado	Lógica OU entre posição de memória e dado imediato, com resultado na posição de memória	3	24
XRL	A,R _n	Lógica OU-EXCLUSIVO entre registrador e acumulador, com resultado no acumulador	1	12
XRL	A,direto	Lógica OU-EXCLUSIVO entre posição de memória e acumulador, com resultado no acumulador	2	12
XRL	A,@R _i	Lógica OU-EXCLUSIVO entre posição de memória indicada por R _i e acumulador, com resultado no acumulador	1	12
XRL	A,#dado	Lógica OU-EXCLUSIVO entre dado imediato e acumulador, com resultado no acumulador	2	12

Mnemônico		Descrição	Byte	Períodos de clock
XRL	direto,A	Lógica OU-EXCLUSIVO entre posição de memória e acumulador, com resultado na posição de memória	2	12
XRL	direto,#dado	Lógica OU-EXCLUSIVO entre posição de memória e dado imediato, com resultado na posição de memória	3	24
CLR	A	Limpa o acumulador	1	12
CPL	A	Complementa o acumulador	1	12
RL	A	Rotaciona o acumulador à esquerda	1	12
RLC	A	Rotaciona o acumulador com o carry à esquerda	1	12
RR	A	Rotaciona o acumulador à direita	1	12
RRC	A	Rotaciona o acumulador com o carry à direita	1	12
SWAP	A	Permuta nibbles do acumulador	1	12
TRANSFERÊNCIA DE DADOS				
MOV	A,R _n	Move registrador para o acumulador	1	12
MOV	A,direto	Move posição de memória para o acumulador	2	12
MOV	A,@R _i	Move posição de memória indicada por R _i para o acumulador	1	12
MOV	A,#dado	Move dado imediato para o acumulador	2	12
MOV	R _n ,A	Move acumulador para o registrador	1	12
MOV	R _n ,direto	Move posição de memória para o registrador	2	12
MOV	R _n ,#dado	Move dado imediato para o registrador	2	12
MOV	direto,A	Move acumulador para posição de memória	2	12
MOV	direto,R _n	Move registrador para posição de memória	2	24
MOV	direto,direto	Move posição de memória para outra posição de memória	3	24
MOV	direto,@R _i	Move posição de memória indicada por R _i para posição de memória	2	24
MOV	direto,#dado	Move dado imediato para posição de memória	3	24
MOV	@R _i ,A	Move acumulador para posição de memória indicada por R _i	1	12
MOV	@R _i ,direto	Move posição de memória para posição de memória para @R _i	2	24
MOV	@R _i ,#dado	Move dado imediato para posição de memória indicada por @R _i	2	12
MOV	DPTR,#dado16	Move para DPTR constante de 2 bytes	3	24
MOVC	A,@A+DPTR	Move o conteúdo da posição de memória indicada pela soma do acumulador e do DPTR para o acumulador	1	24

Mnemônico		Descrição	Byte	Períodos de clock
MOVC	A,@PC+DPTR	Move o conteúdo da posição de memória indicada pela soma do acumulador e do PC para o acumulador	1	24
MOVX	A,@R _i	Move o conteúdo da posição de memória indicada por R _i para o acumulador	1	24
MOVX	A,@DPTR	Move o conteúdo da posição de memória indicada por DPTR para o acumulador	1	24
MOVX	@R _i ,A	Move acumulador para o conteúdo da posição de memória indicada por R _i	1	24
MOVX	@DPTR,A	Move acumulador para o conteúdo da posição de memória indicada por DPTR	1	24
PUSH	direto	Incrementa em um o <i>Stack Pointer</i> e move o conteúdo da posição de memória para a posição de memória indicada pelo <i>Stack Pointer</i>	1	24
POP	direto	Decrementa em um o <i>Stack Pointer</i> e move o conteúdo da posição de memória indicada por ele para a posição de memória	1	24
XCH	A,R _n	Faz a permuta entre o conteúdo do registrador e do acumulador	1	12
XCH	A,direto	Faz a permuta entre o conteúdo da posição de memória e do acumulador	2	12
XCH	A,@R _i	Faz a permuta entre o conteúdo da posição de memória indicada por R _i e do acumulador	1	12
XCHD	A,@R _i	Faz a permuta entre os <i>nibbles</i> baixos do conteúdo da posição de memória indicada por R _i e do acumulador	1	12
MANIPULAÇÃO DE VARIÁVEIS BOOLEANAS				
CLR	C	Coloca em nível lógico zero o bit de carry	1	12
CLR	bit	Coloca em nível lógico zero o bit indicado	2	12
SETB	C	Coloca em nível lógico um o bit de carry	1	12
SETB	bit	Coloca em nível lógico um o bit indicado	2	12
CPL	C	Complementa o bit de carry	1	12
CPL	bit	Complementa o bit indicado	2	12
ANL	C,bit	Lógica E entre o bit de carry e o bit indicado, com resultado no bit de carry	2	24
ANL	C,/bit	Lógica E entre o bit de carry e o complemento do bit indicado, com resultado no bit de carry	2	24
ORL	C,bit	Lógica OU entre o bit de carry e o bit indicado, com resultado no bit de carry	2	24
ORL	C,/bit	Lógica OU entre o bit de carry e o complemento do bit indicado, com resultado no bit de carry	2	24

Mnemônico		Descrição	Byte	Períodos de clock
MOV	C,bit	Move bit indicado para o bit de carry	2	12
MOV	bit,C	Move bit de carry para bit indicado	2	24
JC	rel	Desvia se o bit de carry for um	2	24
JNC	rel	Desvia se o bit de carry for zero	2	24
JB	bit,rel	Desvia se o bit indicado for um	3	24
JNB	bit,rel	Desvia se o bit indicado for zero	3	24
JBC	bit,rel	Desvia se o bit indicado for um e o coloca em nível lógico zero	3	24
RAMIFICAÇÃO DE PROGRAMA				
ACALL	end 11	Chama sub-rotina colocada numa faixa de 2 kbytes da posição atual	2	24
LCALL	end 16	Chama sub-rotina em qualquer posição da memória	3	24
RET		Retorno de sub-rotina	1	24
RETI		Retorno de rotina de atendimento de interrupção	1	24
AJMP	end 11	Desvia para posição de memória numa faixa de 2 kbytes da posição atual	2	24
LJMP	end 16	Desvia para qualquer posição de memória	3	24
SJMP	rel	Desvio curto relativo	2	24
JMP	@A+DPTR	Desvia para a posição de memória indicada pela soma do acumulador e do DPTR	1	24
JZ	rel	Desvia se o acumulador é zero	2	24
JNZ	rel	Desvia se o acumulador é diferente de zero	2	24
CJNE	A,direto,rel	Compara o acumulador com o conteúdo da posição de memória e desvia se eles forem diferentes	3	24
CJNE	A,#dado,rel	Compara o acumulador com dado imediato e desvia se eles forem diferentes	3	24
CJNE	R _n ,#dado,rel	Compara o registrador com dado imediato e desvia se eles forem diferentes	3	24
CJNE	@R _i ,#dado,rel	Compara conteúdo da posição de memória indicada por R _i com dado imediato e desvia se eles forem diferentes	3	24
DJNZ	R _n , rel	Decrementa o registrador e desvia se ele for diferente de zero	3	24
DJNZ	direto,rel	Decrementa o conteúdo da posição de memória e desvia se ela for diferente de zero	3	24
NOP		Nenhuma operação	1	12

ANEXO II

MACROS

Uma macro é um nome que você fixa para um ou mais comandos do assembly. O *Assembler A51* possui um processador de macro que habilita você a definir e usar macros em seus programas em *assembly*.

Quando se define uma macro, associa-se um texto (um código *assembly*) a um nome de macro. Para incluir o texto da macro em seu programa *assembly*, você escreve o nome da macro e o *assembler A51* substitui o nome da macro pelo texto especificado na definição da macro. Usar macros em programas *assembly* apresenta as seguintes vantagens:

- O uso freqüente de macros reduz erros provocados pela digitação. Uma macro permite definir seqüências de instruções que são usadas repetidamente no programa. O subsequente uso da macro produz fielmente o mesmo resultado a cada vez. É claro, a introdução de um erro em uma macro fará que o erro seja repetido sempre que a macro for usada;
- O conjunto de símbolos usados na macro está limitado a macro. Não é necessário se preocupar com símbolos usados previamente;

- Macros são convenientes para a criação de tabelas de código simples. A produção dessas tabelas a mão é tediosa e sujeita a erros.

Uma macro pode ser pensada como uma sub-rotina, com a diferença que o código que estaria na sub-rotina está incluído no ponto de chamada da macro. Entretanto, as macros não devem ser usadas para substituir sub-rotinas. A chamada da sub-rotina só adiciona o código de chamada desta. Já a chamada de uma macro faz com que o código *assembly* associado com esta seja incluído no programa *assembly*. Isto provoca um crescimento rápido do programa quando uma grande macro é freqüentemente usada. Em programas onde o tempo de execução é crítico, as macros aceleram a execução de algoritmos ou comandos freqüentemente usados, pois eliminam o demorado procedimento de chamada de sub-rotina. Assim, ao decidir entre o emprego de macros ou sub-rotinas, siga as seguintes orientações:

- Sub-rotinas são melhor empregadas quando alguns procedimentos são executados freqüentemente ou quando há pouco espaço de memória disponível;
- Macros devem ser usadas quando se requer velocidade máxima de processamento e quando há muito espaço de memória disponível

Para chamar uma macro em um programa *assembly*, você deve primeiro definir a macro. Por exemplo, a definição seguinte:

```
ATRASSO MACRO A1, A2 ;definição da macro
```

```
MOV R1, #A1
```

```
MOV R2, #A2
```

```
DJNZ R1, $
```

```
DJNZ R2, $-2
```

```
ENDM
```

define uma macro chamada “atraso” que aceita os argumento A1 e A2. Caso A1 for igual a 3 e A2 for igual a 4, o código emitido para as duas primeiras linhas será:

```
MOV R1, #3
```

```
MOV R2, #4
```

O programa que segue mostra como você pode chamar a macro “atraso” a partir de um programa *assembly*.

```
ORG 00h
```

```
LOOP:MOV P0, #0 ;limpa PORTA 0
```

```
ATRASSO 4, 5;
```

```
MOV P0, #0FFh ;seta PORTA 0
```

```
ATRASSO 4, 5;
```

```
SJMP LOOP ;repete
```

ANEXO III

DIRETIVAS DO ASSEMBLER

O *Assembler A51* suporta um número de diretivas que permitem definir símbolos, reservar e inicializar espaço de memória e especificar locação de memória para código objeto gerado. As diretivas não devem ser confundidas com as instruções do 8051. Com exceção de DB e DW, diretivas não produzem código de objeto. As diretivas do *assembler* podem ser divididas em categorias. Há diretivas para definir símbolos, reservar espaço de memória, inicializar dados, controlar o estado do *assembler*, selecionar segmentos e definir macros. A seguir, apresentamos as diretivas separadas por categorias, descrevemos brevemente o efeito da diretiva e apresentamos um exemplo.

Diretiva	Definição	Exemplos
Diretivas para Definição de Símbolos		
BIT	Define um símbolo que remete ao endereço de um bit da memória de dados interna.	LED BIT P1.0 FLAG BIT 20h
CODE	Define um símbolo que remete a um endereço de código.	REINICIO CODE 00H INTE0 CODE REINICIO + 3
DATA	Define um símbolo que remete a um endereço de dados interno.	SOMA DATA 40H PORT1 DATA 90H
EQU	Relaciona um valor numérico ou um registrador a um símbolo.	CONTA EQU R5 LEDS EQU P1
IDATA	Define um símbolo que remete a um endereço de dados interno endereçável indiretamente.	BUF IDATA 60h BUF_FIM IDATA BUF + 0Fh

Diretiva	Definição	Exemplos
SEGMENT	Declara um segmento relocável ¹⁵ dando-lhe um nome, o tipo de segmento e a informação de relocação (opcional).	STACK SEGMENT IDATA
SET	Relaciona um valor numérico ou um registrador a um símbolo. O símbolo pode ser mudado na sequência usando outro comando SET.	CONT SET R1 TEMP SET CONT
XDATA	Define um símbolo que remete a um endereço de dados externo.	TEMPO XDATA 100h
Diretivas que Reservam e Inicializam Memória		
DB	Inicializa um endereço da memória de programa (código) com um valor de oito bits	FRASE : db 'BEM VINDO' TABELA:db 00h, 03h, 07h
DBIT	Reserva espaço em um segmento de bit	ON DBIT 1 ;reserva 1 bit OFF DBIT 1
DS	Reserva espaço no segmento corrente	TEMPO DS 8
DW	Inicializa um endereço da memória de programa (código) com um valor de 16 bits	TAB: DW TAB, TAB + 10,
Diretivas para <i>Link</i>-edição do Programa		
EXTRN	Especifica símbolos que são declarados em outros módulos objeto.	EXTRN CODE (TABELA), EXTRN DATA (SOMA)
NAME	Especifica o nome do módulo objeto.	NAME MODULO TECLADO
PUBLIC	Especifica símbolos que podem ser usados em outros módulos objeto.	PUBLIC TABELA, SOMA

¹⁵ Segmentos relocáveis tem um nome, um tipo e outros atributos. Segmentos relocáveis com o mesmo nome mas de módulos objeto diferentes são considerados partes do mesmo segmento e chamados de segmentos parciais. Eles podem ser combinados na *link*-edição pelo *L51*.

Diretiva	Definição	Exemplos
Diretivas de Controle do Estado do Assembler		
END	Assinala o fim de um módulo em <i>Assembly</i>	
ORG	Especifica qual o endereço do próximo comando do <i>Assembly</i>	ORG 100h
USING	Indica o banco de registradores R1 a R7 em uso no trecho subsequente do programa	USING 3 PUSH AR2 ;Coloca R2 do banco 3 na pilha
Diretivas de Seleção de Segmento		
BSEG	Seleciona um segmento absoluto ¹⁶ dentro do espaço endereçável por bit	BSEG AT 30h ; segmento de bit absoluto @ 30h DEC_FLAG: DBIT 1 ; bit absoluto INC_FLAG: DBIT 1
CSEG	Seleciona um segmento absoluto na memória de programa	CSEG AT 1000h ; segmento de código absoluto @ 1000h PARITY_TAB: DB 00h ; paridade para 00h DB 01h ;paridade para 01h DB 01h ;paridade para 02h . . . DB 00h ;paridade para FFh
DSEG	Seleciona um segmento absoluto na memória de dados interna	DSEG AT 40h ; segmento de dados absoluto @ 40h TMP_A: DS 2 ; palavra de dados absoluta TMP_B: DS 4

¹⁶ Segmentos absolutos residem em uma localização de memória fixa. Eles não pode, ser movidos pelo *link*-editor.

Diretiva	Definição	Exemplos
ISEG	Seleciona um segmento absoluto na memória de dados interna de acesso indireto	ISEG AT 40 h ; seg indireto de dados abs @ 40h TMP_IA: DS 2 TMP_IB: DS 4
RSEG	Seleciona um segmento relocável que foi declarado previamente com a diretiva segmento. ,	. . . PROG SEGMENT CODE ; declara um segmento RSEG PROG ; seleciona o segmento MOV A, #0 MOV P0, A . . .
XSEG	Seleciona um segmento absoluto na memória de dados externa	XSEG AT 2000h ; seg de dados externos abs @ 2000h OEMNOME:DS 25 ; dados externos absolutos PRDNOME: DS 5 VERSÃO: DS 25
Diretivas de Definição de Macros		
ENDM	Fim da definição da macro	
EXITM	Termina imediatamente uma expansão de macro. Se uma diretiva EXITM é encontrada enquanto a macro expande, o processador de macro ignora as linhas subsequentes até uma diretiva ENDM ser encontrada.	ENSAIO MACRO X, Y, Z ; definição de macro . . . IF X = 0 ; Termina a expansão se X é zero EXITM ENDIF . . . ENDM; fim de macro

Diretiva	Definição	Exemplos
IRP	Repete um bloco de texto uma vez para cada argumento de uma lista. O parâmetro especificado no bloco de texto é substituído por cada argumento. O comando IRP tem a seguinte formação: IRP parâmetro, <argumento, argumento, ... >	; Esta macro zera R1,R2,R3 e R4 quando chamada LIMPREGS MACRO ;define macro IRP RN, <R0,R1,R2,R4,> MOV RN, #0 ENDM ; fim de IRP ENDM ; fim de macro
IRPC	Repete um bloco de texto uma vez para cada caractere do argumento especificado. O Parâmetro especificado é substituído no bloco de texto por cada caractere. O comando IRPC tem a seguinte formação: IRP parâmetro, <argumento>	; Esta macro transmite a mensagem teste quando chamada TRANS MACRO ;define macro IRPC CHR, <TEST> JNB TI, \$; espera fim de transmissão CLRTI MOV A, #'CHR' MOV SBUF, A ; transmite CHR ENDM ; fim de IRPC ENDM ;fim de macro
LOCAL	Usada dentro de uma definição de macro para listar até 16 símbolos locais que são usados dentro de uma macro. Símbolos locais são visíveis somente dentro de uma macro.	; Esta macro apaga o bloco de memória com início em “INI” e de dimensão “DIM” APAGMEM MACRO INI, DIM ;define macro LOCAL LOOP ;define rótulo local MOV R7, #DIM MOV R0, #INI MOV A, #0 LOOP: MOV @R0, A INC R0 DJNZ R7, LOOP ENDM

Diretiva	Definição	Exemplos
MACRO	Usada para declarar uma macro.	DELAY MACRO A1 ;define macro
REPT	Especifica um fator de repetição para um bloco de texto.	; Esta macro insere um atraso de A1 ciclos de máquina ATRASSO MACRO A1 ;define macro REPT A1 ;insere a instrução NOP A1 vezes NOP ENDM ;fim do bloco REPT ENDM ;fim da definição de macro,