

Introdução ao Microcontrolador 8051

Prof. Raimes Moraes

Departamento de Engenharia Elétrica e Eletrônica

Universidade Federal de Santa Catarina

– 2014 –

1. Introdução: Microprocessadores e Microcontroladores

A Unidade Central de Processamento (UCP) é uma máquina de estados projetada para realizar a leitura de instruções da memória de programa, bem como decodificá-las e executá-las. Em microcomputadores, a UCP encontra-se encapsulada em um único circuito integrado denominado microprocessador. A UCP possui dois módulos: Unidade de Controle (UC) e Unidade Lógica Aritmética (ULA).

A UC tem a tarefa de realizar a leitura, decodificação e execução das instruções. Sinais de controle são gerados pela UC para viabilizar estas tarefas. A execução das instruções pode compreender a alteração do fluxo de execução do programa, leitura e/ou escrita de dados em registradores ou em memória de dados, bem como o acionamento da ULA. A ULA realiza as operações lógicas (*and*, *or* e outras) e aritméticas requisitadas pelas instruções.

Em aplicações práticas, o microprocessador deve ser utilizado em conjunto com, pelo menos, memória de programa (geralmente EEPROM - *electrically erasable programmable read-only memory*) e de dados (RAM - *random access memory*). Diferentes periféricos podem também ser acrescentados ao sistema para aumentar suas funcionalidades de maneira a construir, por exemplo, um microcomputador. Para que o microprocessador possa se comunicar com as memórias e periféricos, o mesmo conta com conjunto de pinos (denominados portas) de entrada e/ou saída que são interligados aos pinos dos outros dispositivos. Como ilustrado pela Figura 1, estes pinos têm a tarefa de suprir os

endereços (barramento de endereços), sinais de controle (barramento de controle) e receber ou escrever dados (barramento de dados). Estes barramentos são constituídos por conjunto de trilhas na placa de circuito impresso para conectar os pinos de entrada/saída do microprocessador aos periféricos. O número de trilhas em cada barramento depende do microprocessador e periféricos utilizados.

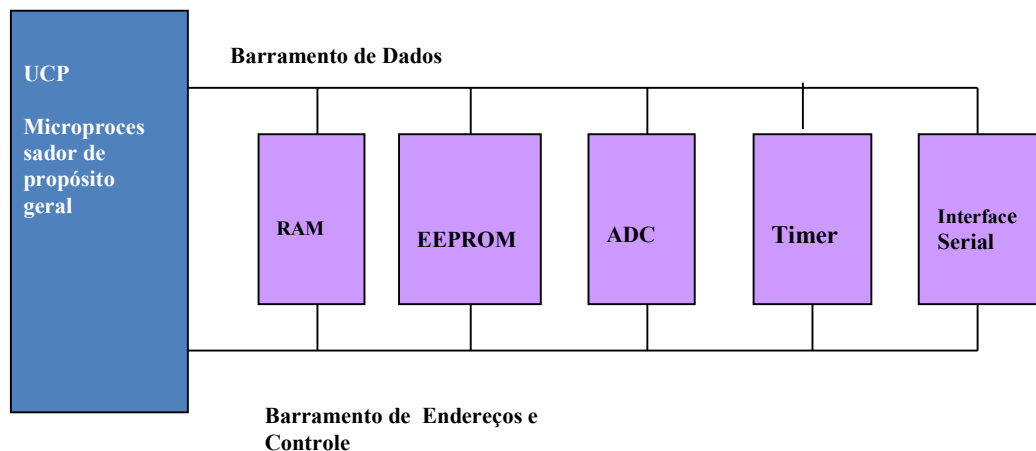


Figura 1 – Diagrama de microprocessador (arquitetura Von Neuman) conectado à memória de dados (RAM), memória de programa (EEPROM) e periféricos (Conversor Analógico Digital – ADC, Timer e Interface Serial). Cada um destes barramentos contém número de trilhas que depende do tipo de microprocessador e da capacidade de armazenamento das memórias.

Microcontroladores, por sua vez, contêm, em um mesmo circuito integrado, UCP, memória de programa e memória de dados, incorporando ainda, outros periféricos, tais como temporizadores/contadores e interface de comunicação serial.

Os microcontroladores possibilitam a implementação de sistemas mais compactos e de menor custo do que aqueles baseados em microprocessadores. Por outro lado, as UCPs dos microcontroladores têm um conjunto de instruções menos versátil e, geralmente, com desempenho inferior ao dos microprocessadores.

2. Fundamentos da Organização do Microcontrolador 8051

O 8051 é um microcontrolador de 8 bits (registradores internos de 8 bits) lançado comercialmente pela Intel em 1981, sendo que versões do mesmo são produzidos até hoje pelos maiores fabricantes de dispositivos semicondutores; há também, versões comerciais em HDL para gravação em dispositivos lógicos programáveis.

A Figura 2 apresenta o encapsulamento original do microcontrolador 8051 e as funções de seus pinos. Para energizar o 8051, Vcc (pino 40) é conectado a 5V e GND (pino 20), a 0V. Aos pinos XTAL1 e XTAL2, conecta-se cristal que estabelece o período de referência (sinal de clock) para a execução das atividades da UCP. O fabricante informa na folha de dados do microcontrolador qual a faixa de frequência de funcionamento suportada pelo mesmo de tal forma a orientar a escolha do cristal a ser utilizado; no caso do 8051, o limite superior é 12 MHz. O 8051 possui 4 portas (P0, P1, P2 e P3) bidirecionais (entrada ou saída) que podem ser utilizadas para comunicação de dados ou para gerar sinais de controle, sendo que as portas P0, P2 e P3 podem ainda executar outras funções (discutidas ao longo do texto). A opção pela utilização das portas para uma ou outra função irá depender das características do projeto no qual o 8051 é empregado.

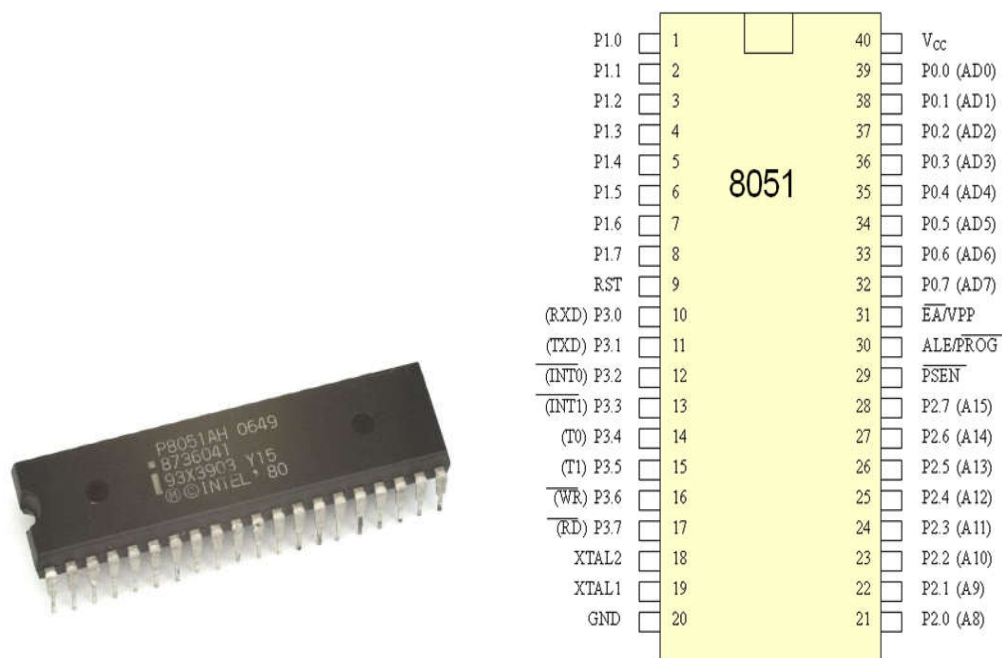


Figura 2 - Foto do 8051 no seu encapsulamento original e funções de seus pinos.

O 8051 possui UCP, periféricos, memória de programa e de dados, sendo:

- ❑ 4 KiB (4 x 1024 bytes) de memória ROM interna para programas;
- ❑ 128 bytes de memória RAM interna para dados.

A UC do 8051 pode acessar cada byte destas memórias, informando o endereço da posição de memória e gerando sinais de controle. Uma representação simplificada da organização da UCP do 8051 é mostrada na Figura 3.

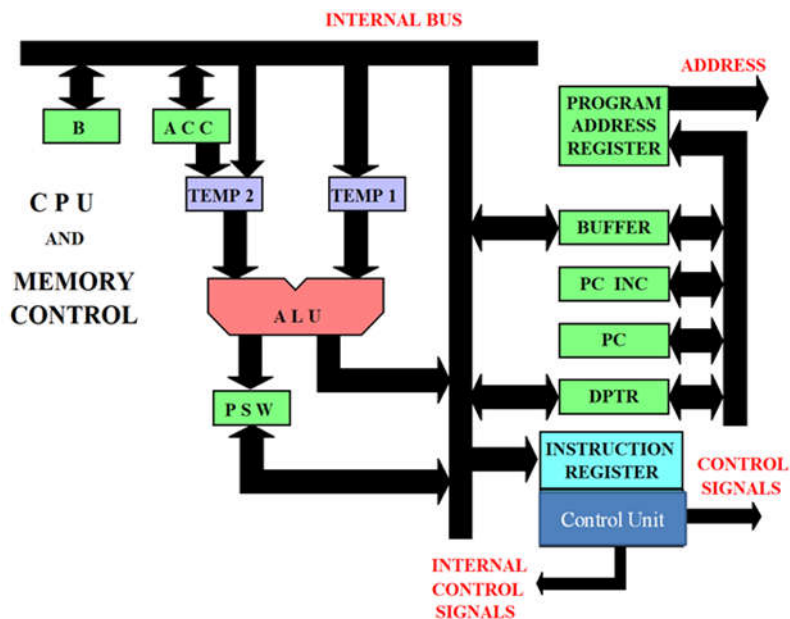


Figura 3 – Representação simplificada da arquitetura da UCP do 8051

O endereço da memória de programa do qual o 8051 deve buscar a instrução a ser executada é fornecido pelo registrador denominado Contador de Programa (PC – *Program Counter*), sendo que este possui 16 flip-flops; permite, portanto, acessar até 64 KiB ($2^{16} = 65536$) de memória de programa (endereços de 0000H a FFFFH). Ao ser energizado, este registrador é inicializado com o valor 0000H. O conteúdo do PC é disponibilizado no barramento de endereços para especificar o endereço da memória de programa de onde o código da instrução a ser executada deve ser lido; após a leitura do primeiro byte da instrução, a

UC incrementa o PC (PC = 0001H). O conteúdo contido no endereço 0000H da memória de programa é recebido e carregado no registrador de instrução (*Instruction Register* – IR) onde é decodificado, demandando que UC gere todos os sinais de controle necessários à execução desta instrução. O 8051 possui instruções com mais de um byte. Se o conteúdo contido no endereço 0000H corresponde à parte de instrução (portanto, de uma instrução com mais de um byte), o 8051 lê o byte do próximo endereço da memória de programa (contido em PC = 0001H; este é, então, novamente incrementado: PC = 0002H). Quando todos os bytes que compõem a instrução forem lidos de endereços consecutivos da memória de programa (tal que o 8051 disponha de todas as informações necessárias para a execução da tarefa correspondente à instrução), a tarefa solicitada é executada.

Há instruções do 8051 que solicitam a alteração do fluxo de execução do programa (SJMP, ACALL e outras); ou seja, o código da próxima instrução a ser executada não está sendo apontada pelo atual valor contido no PC. Neste caso, a execução destas instruções faz com que haja alteração do conteúdo do PC. Assim, quando a máquina de estado da UC colocar o novo conteúdo do PC (modificado, por exemplo, pela instrução ACALL) no barramento de endereços, o código lido da memória de programa será o código do novo endereço contido no PC. Após, a leitura deste conteúdo, o PC é incrementado, assim como descrito anteriormente.

O que vem a ser esta decodificação realizada pela UC? O projetista do processador cria um conjunto de códigos binários, sendo que cada código demanda a execução de diferentes ações por parte do processador. Por exemplo, o código 11101000B (E8H - B corresponde a binário; H a hexadecimal. Embora, o microprocessador sempre trabalhe com valores binários, é usual representar estes valores em base hexadecimal por questão de compactação) faz com que a UC gere sinais internos no 8051 para copiar o conteúdo do registrador R0 no Acumulador (registrador A - Seção 2.1). O código E5H, 20H (instrução com 2 bytes, portanto) faz com que a UC atue para transferir o conteúdo da posição 20H da memória de dados interna para o registrador A. Existem diversos outros códigos, como por exemplo, para

solicitar que o processador execute operações aritméticas, lógicas, altere o fluxo de execução do código do programa. Estas instruções serão abordadas ao longo do texto.

Como o usuário do processador toma conhecimento destes registradores e códigos? O fabricante do processador publica a folha de dados do mesmo explicando o seu modo de funcionamento (arquitetura, registradores e outros), bem como, os códigos.

As tarefas a serem executadas pelo 8051 são estabelecidas por conjunto de instruções armazenado na memória de programa tal que o processador, ao realizar a sua leitura (uma a uma) e executá-las (uma a uma), realize as atividades desejadas pelo programador (por exemplo, reproduzir uma música armazenada em formato mp3).

Em sistemas utilizando o 8051, as instruções são organizadas na memória de programa em palavras de 8 bits. As Tabelas 1a e 1b mostram exemplo de programa carregado na memória em formato binário e hexadecimal. Como digitar dados e instruções de 8 bits é trabalhoso (Tabela 1a), normalmente estes dados são escritos, por conveniência, de forma mais compacta, em representação hexadecimal (Tabela 1b). Em ambas as tabelas, os endereços da memória de programa já foram representados, por conveniência, em formato hexadecimal. Os endereços são supridos pela UC do 8051 (após leitura do conteúdo do registrador PC) à memória de programa através de 16 linhas de endereçamento (barramento de endereços).

2.1 - Registradores do 8051

Além do PC, utilizado para suprir o endereço da instrução a ser lida da memória de programa, o 8051 conta com outros registradores internos para a execução das tarefas (Figura 4).

O registrador DPTR, constituído por DPH (8 flip-flops mais significativo do registrador) e DPL (8 flip-flops menos significativo do registrador), também

possui 16 flip-flops; os demais registradores do 8051 possuem 8 flip-flops. O DPTR armazena endereço de 16 bits para acessar dados na memória de programa, bem como ler ou escrever dados em memória RAM externa (Seção 2.2). Assim como o PC, este registrador tem 16 flip-flops para que possa endereçar uma faixa de 64KiB (0000H a FFFFH).

Tabela 1 – Exemplo de memória de programa com o conteúdo escrito em formato binário (1a) e hexadecimal (1b)

(1a)		(1b)	
ENDEREÇOS (HEXADECIMAL)	DADOS (BINÁRIO)	ENDEREÇOS (HEXADECIMAL)	DADOS (HEXADECIMAL)
0000	0111 0101	0000	75
0001	1010 1000	0001	A8
0002	1001 0000	0002	90
0003	1011 0110	0003	B6
0004	0000 0001	0004	01
0005	1111 1101	0005	FD
0006	1011 1001	0006	B9
....	

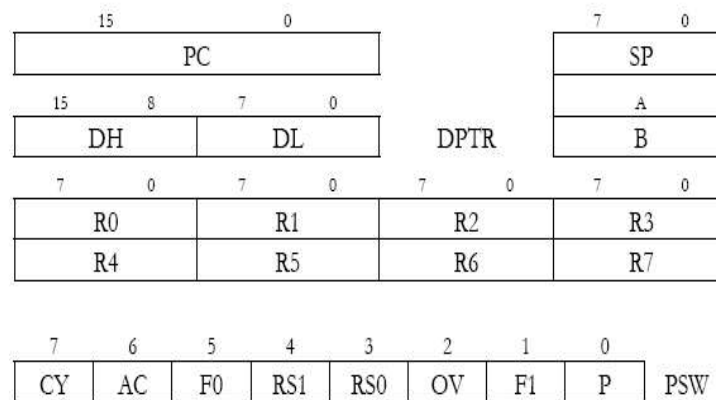


Figura 4 – Registradores da UCP do 8051

O registrador A (referenciado como ACC em algumas instruções) é denominado acumulador, pois armazena o resultado de operações lógicas e aritméticas.

O registrador B é utilizado em operações de multiplicação e divisão, podendo ser também empregado como registrador de propósito geral. Os registradores R0 a R7, de propósito geral, são utilizados no código escrito pelo programador para a manipulação de dados. Exemplos serão apresentados futuramente.

O registrador PSW contém informações sobre o resultado de operações executadas utilizando a ULA (CY, AC e OV), informação sobre dado carregado no acumulador (P), flags (F1 e F0) para uso do programador e flip-flops para especificar o banco de registradores a ser manipulado pelo processador (Seção 2.4). A Tabela 2 apresenta a função de cada um dos flip-flops que o compõe.

Tabela 2 – Descrição da função dos flip-flops do registrador PSW

Nome	Localização	Descrição
CY	PSW.7	Sinaliza vai-um do bit 7
AC	PSW.6	Sinaliza vai-um auxiliar (vai-um do bit 3 para 4)
F0	PSW.5	Sinalização definida pelo programador
RS1	PSW.4	Flip-flop 1 do seletor do banco de registradores
RS0	PSW.3	Flip-flop 0 do seletor do banco de registradores
OV	PSW.2	Sinaliza <i>overflow</i> (excede capacidade)
F1	PSW.1	Sinalização definida pelo programador
P	PSW.0	Sinaliza paridade ímpar

2.2 - Conexão do 8051 às memórias externas de programa e de dados

Recapitulando, o 8051 possui UCP, periféricos, memória de programa e de dados, sendo:

- ☐ 4 KiB de memória ROM interna para programas;
- ☐ 128 bytes de memória RAM interna para dados.

Em algumas aplicações, este volume de memória pode ser insuficiente para armazenar o programa ou para a manipulação de dados pelo programa. O 8051 pode acessar, externamente, até:

- ☐ 64 KiB de memória de programa;
- ☐ 64 KiB de memória de dados.

O registrador PC tem 16 flip-flops para acessar até 64 KiB ($2^{16}=65536$) de memória de programa (endereços de 0000H a FFFFH). Assim que um byte é lido da memória de programa (interna ou externa), o PC é incrementado para que passe a conter o endereço do próximo byte a ser lido da memória de programa.

Existe um pino de entrada do 8051, o /EA (*External Access*) que especifica a maneira de acessar a memória de programa (Figura 2 – Pino 31). Quando o nível lógico presente neste pino for alto (/EA = '1'), o 8051 acessa a memória de programa interna e externa; neste modo, caso o conteúdo do PC exceda o maior endereço da memória de programa interna (0FFFFH; ou seja, endereço correspondente a 4 KiB), o 8051 passa a buscar, automaticamente (gerando os sinais de controle necessários para tal), instruções da memória de programa externa (Figura 5). Opcionalmente, o programa pode ser lido apenas da memória de programa externa na faixa de endereçamento de 0000H a FFFFH, colocando-se o pino /EA em nível lógico baixo (/EA = '0').

Quando o 8051 acessa a memória externa, as portas P0 e P2 ficam comprometidas com esta tarefa. A porta P0 (função AD7-AD0) supre o byte menos significativo (LSB – *Least Significant Byte*) do endereço da memória externa. A porta P2 (função A15-A8) supre o byte mais significativo (MSB – *Most Significant Byte*) do endereço.

Esta função da porta P0 de suprir o LSB do endereço é multiplexada, durante parte do ciclo de leitura (ou escrita), com a tarefa de receber (ou escrever) o byte de dado da memória; ou seja, a porta P0 cumpre as tarefas de conectar o 8051 com o barramento de endereços (8 linhas menos significativas de endereçamento) e com o barramento de dados da memória. O objetivo é reduzir o número de pinos do circuito integrado. Deve-se observar, contudo, que, durante todo o ciclo de leitura (ou escrita), o endereço do qual o dado está sendo lido (ou escrito) deve ser mantido no barramento de endereço da memória sendo acessada. Para compatibilizar estas duas funções (suprir endereço, receber/escrever dado), sistemas com o 8051 que acessam memória externa utilizam, geralmente, o *latch* 74373.

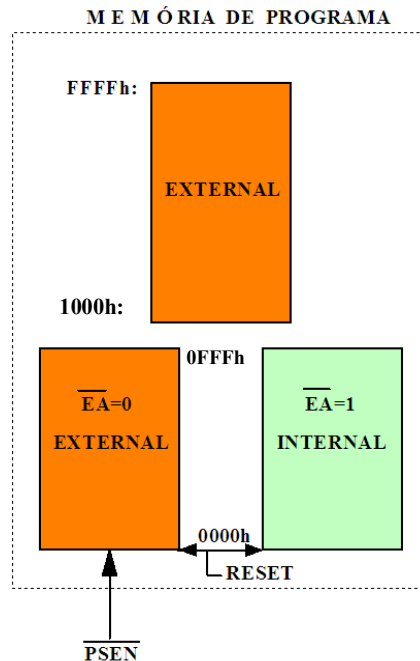


Figura 5 – Para $\overline{EA}=1$, o 8051 acessa os endereços de 0000H a 0FFFH na memória de programa interna; para endereços a partir de 1000H, o 8051 passa a acessar memória de dados externa. Caso \overline{EA} seja '0', o 8051 acessa apenas a memória de programa externa (0000H a FFFFH).

A Figura 6 mostra como o 8051 deve ser conectado à memória de programa externa. Para acessar apenas memória de programa externa, o pino \overline{EA} deve ser colocado em nível lógico baixo ('0').

O componente 74373 (Figura 7a) armazena o LSB do endereço da memória externa a ser acessado. Neste componente, o pino *Output Enable* (\overline{IOEN}) é fixado em nível lógico baixo. Desta forma, como mostrado na tabela verdade (Figura 7b), as saídas do componente acompanham as entradas quando seu pino G está em nível lógico alto (sendo este sinal gerado pelo 8051 através do pino ALE). Quando o pino ALE (*Address Latch Enable*) retorna ao nível lógico baixo, o 74373 retém os valores anteriormente atribuídos (Q_0).

Durante acesso à memória de programa externa, o 8051 coloca o MSB do endereço contido no PC na porta P2, o LSB do PC na porta P0 e o pino ALE em nível lógico alto (Figura 8). Posteriormente, o pino ALE é colocado pela UC do 8051 em nível lógico baixo, disponibilizando a porta P0 para receber byte de instrução proveniente da memória. Em seguida, o sinal \overline{PSEN} (*Program Store*

Enable) é colocado em nível lógico baixo indicando que a UC está pronta para ler um byte de instrução. Com *Output Enable* (/OE – Figura 6) em nível lógico baixo, a memória disponibiliza o byte de instrução em P0. O byte é armazenado pelo 8051 no registrador de instrução quando /PSEN retornar ao nível lógico alto.

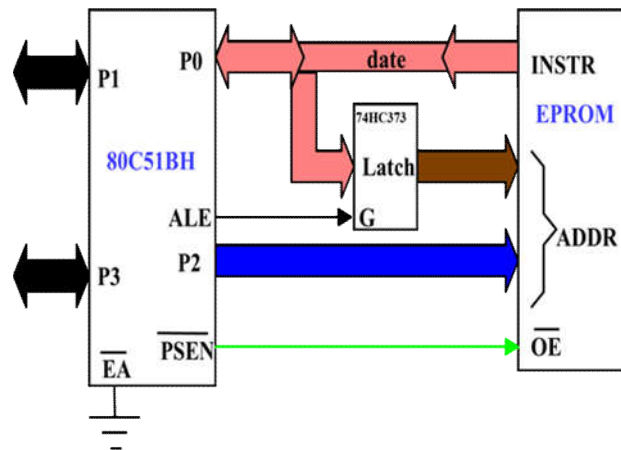


Figura 6 – Conexão do 8051 à memória de programa externa. O componente 74373 é necessário, pois o endereço a ser acessado na memória deve ser mantido nos pinos de endereçamento da memória durante todo o ciclo de leitura; no entanto, P0 cumpre a função de suprir o LSB do endereço (armazenado no 74373) e receber o byte da instrução no mesmo ciclo de leitura.

As conexões entre o 8051 e as memórias são realizadas por trilhas em placa de circuito impresso (PCB - *Printed Circuit Board*). Por exemplo, cada pino de saída de dados da memória de programa é conectado a um pino de entrada de dados do 8051 (porta P0). O bit menos significativo do byte de dados deve estar conectado ao pino P0.0; o mais significativo, ao P0.7. Exemplo de conexão do 8051 a memória externa de 16KiB (2^{14}) é apresentado na Figura 9; como são necessários apenas 14 pinos, A14 e A15 (pinos P2.6 e P2.7) não são conectados à memória.

A Figura 10 mostra o 8051 conectado à memória externa de programa e memória externa de dados. Os sinais de controle /RD (P3.7) e /WR (P3.6) especificam se a operação na memória externa de dados é de leitura ou de escrita, respectivamente; um destes pinos (P3.7 ou P3.6) será automaticamente colocado em nível lógico baixo pela UC ao executar instrução

de leitura ou escrita da memória externa de dados. A diferenciação entre leitura da memória externa de programa e da memória externa de dados em um mesmo endereço se dá através dos sinais de controle /PSEN (leitura da memória de programa) e /RD (leitura da memória de dados) que nunca são colocados em nível lógico baixo simultaneamente pelo 8051. Esta diferenciação é necessária, pois caso o sistema projetado possua, externamente, 64KiB de memória de programa e 64KiB de memória de dados, a faixa de endereçamento de ambas é a mesma: 0000H a FFFFH.

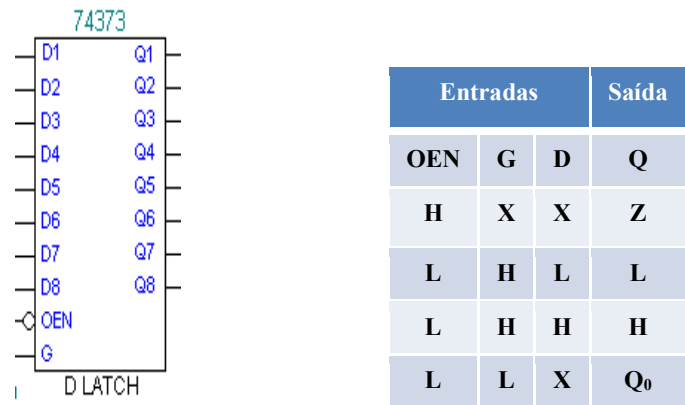


Figura 7 – a) Pinagem e b) tabela verdade do latch D 74373 onde: H = nível lógico alto; L = nível lógico baixo; X = nível lógico qualquer; Z = alta impedância; Q₀= saída anterior.

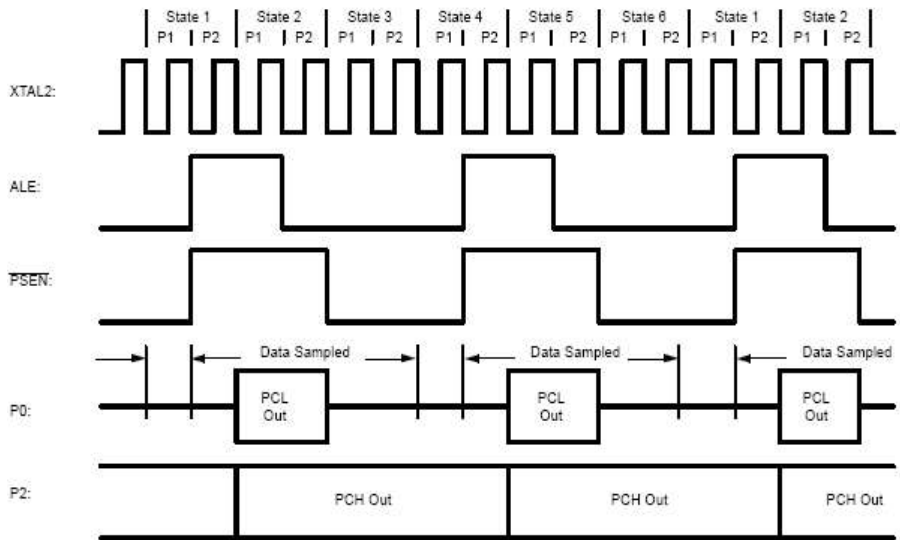


Figura 8 – Exemplo dos sinais de controle supridos pelo 8051 para a leitura de instrução de 3 bytes de memória de programa externa.

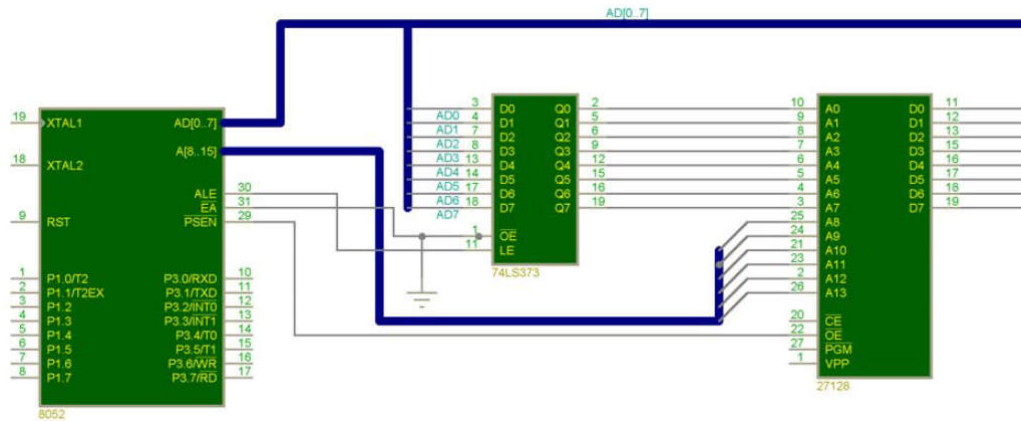


Figura 9 – Conexão do 8051 à memória de programa externa de 16KiB (Fonte: www.hitex.co.uk).

Na Figura 10, o decodificador de endereços foi incluído apenas para efeitos ilustrativos; o mesmo seria útil se houvessem diferentes circuitos integrados (CIs) de memória externa de programa e de dados ocupando diferentes faixas de endereço (tal contexto é mais comum em sistemas microprocessados onde não há diferença no hardware em relação ao acesso destes dois tipos de memória; assim, as mesmas são colocadas em diferentes faixas de endereços). O decodificador de endereços ativa (nível lógico baixo) uma de suas saídas para uma dada faixa de endereços em sua entrada; esta saída é conectada ao *Chip Enable* das memórias para especificar com qual CI o processador deseja se comunicar nesta dada faixa de endereços.

Quando da escrita de dado na memória externa pela instrução `MOVX @DPTR,A`, o 8051 coloca o MSB do endereço (MSB do DPTR – DPH) na porta P2, o LSB do endereço (LSB do DPTR – DPL) na porta P0 e o pino ALE em nível lógico alto (Figura 11). Posteriormente, o pino ALE é colocado pela UC do 8051 em nível lógico baixo; neste instante, o *latch* retém o valor LSB deixando de ser influenciado pelo dado presente nos pinos de P0. Dois ciclos de *clock* (Xtal2) após este evento, o dado a ser escrito é colocado na porta P0; após atraso de mais um ciclo de *clock*, o pino *WR* é colocado em nível lógico baixo para que o dado seja escrito na memória quando de seu retorno a nível lógico alto. O procedimento de leitura funciona de forma similar, sendo que, neste

caso, o pino $\overline{\text{RD}}$ é colocado em nível lógico baixo (sendo que $\overline{\text{WR}}$ fica em nível lógico alto) e dado é recebido pela porta P0.

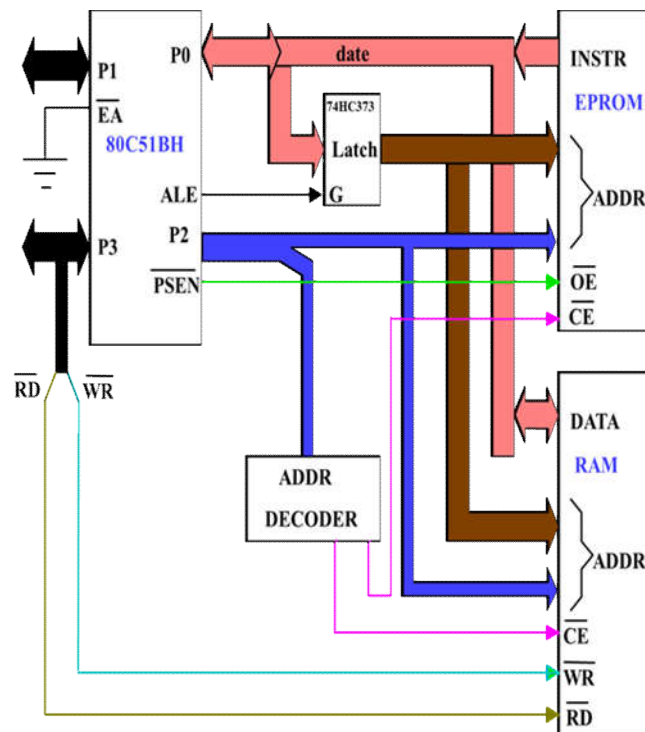


Figura 10 – Conexão do 8051 às memórias externas de programa e de dados. O componente 74373 é necessário, pois o endereço a ser acessado nas memórias deve ser mantido nos pinos de endereçamento da memória durante todo o ciclo de leitura/escrita. O decodificador de endereços é mais usual em sistemas com microprocessador, não sendo necessário seu uso com o 8051, haja vista que o $\overline{\text{PSEN}}$ e $\overline{\text{RD}}$ não são colocados simultaneamente em nível lógico baixo pela sua UC.

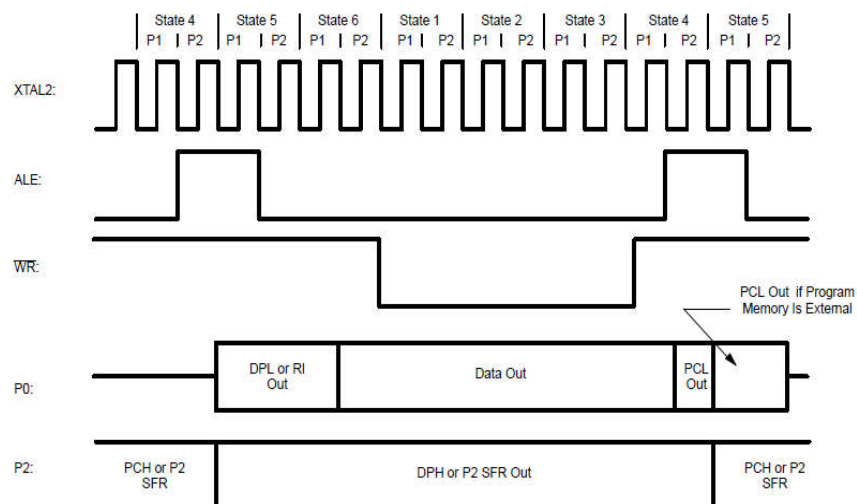


Figura 11 – Exemplo dos sinais de controle gerados pelo 8051 para a escrita de dado em memória externa.

2.3 – Instruções e Mnemônicos do 8051

Conforme já mencionado, as instruções do processador são compostas por conjuntos de bits (Tabela 1). Contudo, é difícil memorizar estes conjuntos visando a elaboração de programas. Assim, estas sequências de bits são associadas a mnemônicos (cadeias de caracteres associadas às tarefas que se deseja do processador) utilizados para escrever os programas em arquivo texto (ASCII). Posteriormente, compiladores são utilizados para converter estes programas-fonte escritos com os mnemônicos nas correspondentes sequências de bits que são gravadas na memória de programa. A Tabela 3 apresenta exemplos de mnemônicos do conjunto de instruções do 8051, bem como flags que são alterados durante a execução das mesmas. Observar que nem todas as instruções alteram os flags. A Tabela 4 exemplifica como o código binário é obtido a partir dos mnemônicos. O Apêndice A traz o conjunto de mnemônicos das instruções do 8051, bem como os flags alterados pelas mesmas. A alteração do flags é útil para a tomada de decisões ao longo do programa (Seção 4).

Tabela 3 - Exemplos de mnemônicos de instruções do 8051 (Rn deve ser substituído por um dos registradores de R0 a R7)

Mnemônicos	Descrição	Operação / Exemplo	Flags
MOV A,Rn	A = Rn	MOV A,R3	
ADD A,Rn	A = A + Rn	ADD A,R7	C, OV, AC, P.
DA A	O conteúdo de A é convertido para nro decimal de 2 dígitos	Se $[A_{3-0} > 9 \text{ ou } AC = 1]$ então $(A_{3-0} + 6)$ Se $[A_{7-4} > 9 \text{ ou } C = 1]$ então $(A_{7-4} + 6)$	C
RL A	Rotaciona o conteúdo do acumulador para a esquerda. O bit 7 é carregado com bit 0.	$(A_{n+1}) \leq (A_n)$ $(A_0) \leq (A_7)$	

Ao se acrescentar ao conjunto de menmônicos, um conjunto de regras de sintaxe de programas-fonte e um conjunto de instruções para o compilador (diretivas de compilação), tem-se a linguagem Assembly.

O Apêndice B apresenta outras características da linguagem Assembly.

Tabela 4 – Exemplos de obtenção de código binário a partir de mnemônicos contidos da folha de dados do 8051. A Tabela 4.a faz uso do campo RRR que deve ser substituído pelo valor binário especificado na Tabela 4.c. A Tabela 4.d mostra a conversão de mnemônicos em código binário.

Instrução MOV A,Rn (a)

5 bits Código Instrução	3 bits Operando
1 1 1 0	1 R R R

Instrução ADD A,direct (b)

8 bits Código Instrução	8 bits
0 0 1 0 0 1 0 1	endereço

REGISTRADOR	RRR
R0	000
R1	001
R2	010
R3	011
R4	100
R5	101
R6	110
R7	111

(c)

Mnemônico	Código Binário	Código Hexadecimal
MOV A,R2	1110 1010	EA
ADD A,32H	0010 0101 0011 0010	25 , 32

(d)

2.4 - Organização da memória de dados interna (iRAM)

Recapitulando, o 8051 possui:

- ☐ 4 KiB de memória ROM interna para programas;
- ☐ 128 bytes de memória RAM interna para dados;

podendo acessar até:

- ☐ 64 KiB de memória externa de programa;
- ☐ 64 KiB de memória externa de dados.

Mostrou-se, anteriormente, como o 8051 é conectado à memória de programa externa; esta pode conter todo o programa (/EA='0') ou conter parte do programa enquanto a memória de programa interna contém outra parte (/EA=1). A memória de programa (ROM) não pode ser modificada; o programa deve ser gravado durante sua fabricação (OBS: o CI AT89C51 é uma variante do 8051 que possui memória EEPROM em vez de ROM, permitindo que este seja programado em laboratório).

A memória de dados do tipo RAM (interna ou externa) pode ser utilizada para criar variáveis; como o nome sugere, estas são alteradas durante a execução do programa para proporcionar, por exemplo, a realização de uma contagem.

Caso a quantidade de memória RAM interna seja suficiente para a aplicação desejada (128 bytes – faixa de endereçamento hexadecimal: 00 a 7FH), não é necessário conectar o 8051 à memória de dados externa.

Os 128 bytes da memória RAM interna podem ser utilizados pelo processador (para guardar dados temporários) ou pelo programador (para armazenar variáveis e dados). No entanto, a mesma possui certas particularidades.

Os registradores R0 a R7 referenciados nas instruções do 8051 (Figura 4 e Tabela 3) pertencem à memória interna de dados, estando alocados entre os endereços 00H a 1FH. Outra peculiaridade da organização da memória interna de dados é que cada um destes registradores está associado a 4 endereços distintos da memória interna de dados. Por exemplo, o registrador R1 corresponde aos endereços de memória interna 01H, 09H, 11H, 19H. Para que se saiba qual posição de memória está sendo alterada ao se manipular o registrador R1 (por exemplo, MOV R1,A) deve-se observar o valor dos flip-flops RS1 e RS0 do registrador PSW (Tabela 2). Estes flip-flops especificam o banco de memória referenciado pelos registradores R0 a R7. A Tabela 5 mostra a faixa de endereços dos bancos especificados pelos flip-flops RS1 e RS0 (registrador PSW) aos quais os registradores de R0 a R7 estão associados; a Tabela 6 mostra a posição de memória do registrador R1 para diferentes bancos de memória (diferentes valores de RS1 e RS0). Ao longo do programa,

para transportar dado para registrador de um dado banco, o programador deve alterar os valores de RS1 e RS0 (registrador PSW) para especificar o banco com o qual deseja trabalhar.

Tabela 5 – Faixa de endereços do banco de registradores R0 a R7 apontado por RS1_RS0

	RS0=0	RS0=1
RS1=0	Banco 0: 00 a 07H	Banco 1: 08 a 0FH
RS1=1	Banco 2: 10 a 17H	Banco 3: 18 a 1FH

Tabela 6 – Endereço do registrador R1 dentro do banco apontado por RS1_RS0

	RS0=0	RS0=1
RS1=0	R1=01H (Banco 0)	R1=09H (Banco 1)
RS1=1	R1=11H (Banco 2)	R1=19H (Banco 3)

Outra particularidade da memória de dados interna do 8051 é que os seus registradores alocados entre os endereços 20 a 2FH podem ter o conteúdo de seus flip-flops manipulados de forma independente dos demais. A Figura 12 apresenta, parcialmente, os endereços (00 a 7FH) para manipulação individual de 128 flip-flops alocados nestas posições de memória interna (20 a 2FH), bem como os 4 bancos de registradores (00 a 1FH). Observe, portanto, que nas instruções de manipulação de bits, os endereços utilizados (0 a 7FH) correspondem a endereços de flip-flops; não são portanto, endereços de registradores completos. A UC identifica se o endereço referenciado é de flip-flop ou de registrador completo através da interpretação do código da instrução sendo executada. Nas instruções do 8051, um dado flip-flop pertencente a esta região de memória (20H a 2FH) pode ser opcionalmente endereçada, por exemplo, como 21H.6 (sexto flip-flop do registrador 21H ou flip-flop 0EH); portanto, as instruções SETB 21H.6 e SETB 0EH ao serem compiladas geram um mesmo código (observe o conjunto de instruções para a manipulação de bits no Apêndice A). Para enfatizar, o flip-flop 4 do registrador 2FH pode ser colocado em nível lógico baixo pela instrução CLR 7CH ou, opcionalmente, por CLR 2FH.4.

Os registradores da memória RAM interna de 30 a 7FH não possuem características peculiares.

7F	7E	7D	7C	7B	7A	79	78	2FH
...								...
0F	0E	0D	0C	0B	0A	09	08	21H
07	06	05	04	03	02	01	00	20H
R0 - R7								1FH
Banco 3								18H
R0 - R7								17H
Banco 2								10H
R0 - R7								0FH
Banco 1								08H
R0 - R7								07H
Banco 0								00H

Figura 12 – Organização da memória de dados interna (iRAM) do 8051. Os 4 bancos de registradores R0 a R7 estão alocados entre 00 e 1FH. Os registradores de 20 a 2FH podem ter seus flip-flops manipulados individualmente através dos endereços de 00 a 7FH.

Existem outros 128 endereços de flip-flops alocados na região de memória disponibilizadas para os registradores de funções especiais.

2.5 - Registadores de Funções Especiais (*Special Function Registers* – SFR)

Foi mencionado na seção anterior que o registrador R1 do Banco 2 tem o endereço 11H; assim, uma instrução que transporte do conteúdo da posição de memória interna de endereço 11H para o acumulador pode ser realizado de forma equivalente, por:

MOV A,R1 (encontrando-se RS1='1',RS0='0') ou MOV A,11H.


Existem outras posições de memória interna de dados alocadas entre 80 e FFH que cumprem funções específicas na estrutura do 8051; são os registradores de funções especiais (*Special Function Registers* – SFR). Através da Tabela 7, toma-se conhecimento dos endereços destes registradores.

Por exemplo, o acumulador encontra-se alocado no endereço E0H. Portanto, a instrução MOV Acc,11H é equivalente a MOV 0E0H,11H (Ver observação no Apêndice A). Contudo, observe que a interpretação da primeira declaração é muito mais prontamente realizada, não requerendo consulta à Tabela 7. Um

segundo exemplo é que, ao enviar um dado para a porta P1 (MOV P1,A), o dado transferido é armazenado em registrador cujo endereço é 90H.

Tabela 7 – Endereços dos registradores de função especial (*Special Function Registers*). Destes registradores, aqueles com endereço de final 0 ou 8H (coluna em verde) podem ter seus flip-flops manipulados individualmente.

F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8									CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	P0	SP	DPL	DPH				PCON	87


 Bit-addressable Registers

Nesta região de memória (80 a FFH), encontram-se também, os registradores B, PSW e DPTR. A bem da verdade, o DPTR de 16 flip-flops é constituído por 2 registradores de 8 flip-flops (DPH e DPL), sendo que DPH corresponde ao MSB e DPL ao LSB. Outros registradores são utilizados para especificar o modo de funcionamento de periféricos contidos no 8051 e serão estudados posteriormente (Seções 7, 8, 9 e 10).

É possível também observar que nem todos os endereços de 80 a FFH foram implementados no 8051; por exemplo, não existe registrador no endereço 91H. Estes endereços são reservados para acesso a periféricos que foram, posteriormente, utilizados em *upgrades* do 8051, como por exemplo, para acessar conversor digital-analógico.

Outra particularidade dos SFR é que aqueles com endereço de final 0 ou 8H (coluna verde da Tabela 7) podem ter seus flip-flops manipulados individualmente; por exemplo, SETB P2.1. Tal arranjo faz com que, além dos 128 flip-flops alocados na RAM interna (iRAM) que podem ser manipulados

individualmente (0 a 7FH), existem outros 128 endereços de flip-flops disponibilizados na região de SFR (80 a FFH). Assim, o endereço 80H corresponde ao flip-flop 0 da porta P0; o endereço 81H corresponde ao flip-flop 1 da porta P0; o endereço 87H corresponde ao flip-flop 7 da porta P0; o endereço 88H corresponde ao flip-flop 0 do registrador TCON; o endereço F7H corresponde ao flip-flop 8 do registrador B. Nem todos estes endereços correspondem a flip-flops efetivamente implementados (alguns destes endereços são disponibilizados para *upgrades* do 8051). Deve ser observado que os endereços em instruções que manipulam bits (SETB, CLR e outras) constituem-se nos endereços de flip-flops mencionados neste parágrafo (além dos 128 flip-flops da iRAM); endereços em instruções que manipulam bytes (por exemplo, MOV Acc,11h, equivalente a MOV 0E0h,11h) são de registradores de 8 flip-flops. Assim, embora a faixa de endereçamento da iRAM + SFR seja a mesma dos flip-flops (0 a FFh), a unidade de controle os diferencia pelo código da instrução.

Os registradores SFR podem ser acessados apenas por endereçamento direto (Seção 3). Isto se deve ao fato de que *upgrades* do 8051 possuem memória de dados interna de propósito geral entre os endereços 80 e FFH (possuem portanto, iRAM de 256 bytes). Para diferenciar acesso a esta distinta região de memória no mesmo endereço dos SFR, os projetistas do 8051 arbitraram que a memória de dados interna de 80 a FFH é acessada com endereçamento indireto e os SFR, com endereçamento direto. Assim, torna-se necessário apresentar os modos de endereçamento do 8051.

3. Modos de endereçamento do 8051

O 8051 possui diferentes modos de endereçamento para tornar mais eficiente a manipulação de dados em diferentes contextos, sendo eles: imediato, direto, indireto, via registrador, indexado e de registradores específicos.

Endereçamento Imediato: O valor do dado manipulado pela instrução é declarado na mesma, passando a compor o código da instrução a ser

carregado na memória de programa. O valor do dado deve ser precedido pelo caractere '#'.

Exemplo: MOV A,#0A0H; o registrador A recebe o valor 0AH. Caso o primeiro caractere do byte de dado esteja entre A e F, o compilador exige a colocação do valor zero na frente, embora este não componha o código da instrução após a compilação.

Endereçamento Direto: O dado a ser manipulado é o conteúdo de uma posição de memória cujo endereço de 8 *bits* compõe o código da instrução a ser carregado na memória de programa. O endereçamento direto é utilizado para acessar a memória RAM interna (00 a 7FH) e os registradores de funções especiais (SFR: 80 a FFH).

Exemplo: MOV R4,0A0H; o registrador R4 passa ter conteúdo da posição de memória A0h. Supondo que o conteúdo de A0h é 38H ([A0H]=38H), tem-se após a execução da instrução [R4]=38H. Observar diferença em relação ao exemplo anterior.

Endereçamento Indireto: A instrução referencia registrador (R0, R1 ou DPTR precedido por '@') cujo conteúdo é o endereço da posição de memória que contém o dado a ser manipulado. A posição de memória apontada pelo registrador referenciado na instrução pode ser da memória de dados interna (MOV) ou externa (MOVX); exceção feita aos SFR que não podem ser acessados por endereçamento indireto. Deve-se observar que para acessar a memória de dados externa utilizando R0 ou R1 (MOVX A,@R1), a porta P2 deve conter o MSB do endereço da memória externa; opcionalmente, pode-se utilizar MOVX A,@DPTR.

Exemplo: Carregar o acumulador com o conteúdo posição de memória de dados externa 0A0BH

```
MOV DPTR,#0A0BH
MOVX A,@DPTR ; Ou opcionalmente
```

```
MOV P2,#0AH
MOV R1,#0BH
MOVX A,@R1
```

Endereçamento via Registrador: O valor manipulado pela instrução está contido no registrador (R0 a R7) do banco de registradores apontado por PSW.

Exemplo: MOV A,R4; o acumulador recebe conteúdo do registrador R4 do banco indicado por PSW.

Endereçamento Indexado: A instrução referencia registrador (A precedido por '@') cujo conteúdo, somado a um endereço base (conteúdo de PC ou DPTR), compõe endereço da posição de memória do programa que contém o dado a ser lido. Utilizado apenas para a leitura de dados da memória de programa (MOVC).

Exemplo: Para o código da subrotina abaixo, um valor entre 1 e 4 no Acumulador fará com que a rotina retorne um dos quatro valores de TABELA.

```
REL_PC:    INC A
           MOVC A,@A+PC
           RET
TABELA:    DB 66H, 77H, 88H, 99H
```

Endereçamento específico: Há instruções que atuam apenas em um registrador específico (por exemplo, Acumulador ou DPTR). Assim sendo, não há necessidade de endereçá-lo; o endereço está implícito no código da instrução.

Exemplo: RL A ; rotaciona o conteúdo do acumulador para a esquerda

4. Instruções de desvio incondicional e condicional

O registrador PC é sempre incrementado ao término da leitura de um byte da memória de programa de forma a conter o endereço do próximo byte a ser lido da memória de programa; assim, as instruções são sequencialmente executadas. No entanto, durante a execução do programa, é comum que a UC

do 8051 altere o fluxo normal de execução do código (em resposta à comparações, por exemplo) para executar trecho de código que se encontra em outra região da memória de programa. Para tal, o conteúdo do registrador PC deve ser modificado para conter o endereço desta outra região de memória, deixando de executar as instruções de forma sequencial. Para que isto seja possível, o 8051 conta com instruções de desvio incondicional e de desvio condicional do fluxo de execução do programa.

Na notação do Assembly, a instrução `JMP <rótulo>` irá sobrescrever o PC com o endereço da instrução indicada pelo <rótulo>, sendo denominada de instrução de desvio incondicional.

Observação: A bem da verdade, existem três instruções do 8051 para desvio incondicional. São elas:

LJMP: Especifica endereço de 16 bits. A instrução possui 3 bytes (opcode + 16 bits de endereço).

AJMP: Especifica endereço de 11 bits. A instrução possui 2 bytes (opcode + 11 bits de endereço).

SJMP: Especifica *off-set* (-128 to +127) a ser somado ao PC para compor o endereço da instrução. A instrução possui 2 bytes (opcode + *off-set*).

Contudo, o programador pode utilizar a instrução `JMP` (que não existe no conjunto de instruções do 8051) e deixar que o compilador a substitua pelo código da instrução mais adequada. A instrução `LJMP` (3 bytes) é utilizada quando o desvio ocorrer para um trecho de código distante (além de 2048 bytes) da instrução `JMP`, pois ocupa 1 byte a mais da memória de programa do que as instruções `AJMP` e `SJMP`.

Na instrução `SJMP`, o campo endereço relativo (<rel addr>) especifica que o rótulo da instrução (no código fonte em Assembly) para a qual se deseja alterar o fluxo do programa será substituído pelo compilador por um *off-set* (-128 a +127) a ser somado ao byte LSB do PC, sendo que o resultado obtido substitui o valor do byte LSB do PC, ocasionando a alteração do fluxo de execução do programa.

A Tabela 8 apresenta instruções que realizam a alteração do fluxo de execução do programa, sobrescrevendo o PC com o endereço da instrução indicada pelo

<rótulo> quando uma dada condição é satisfeita; são portanto, denominadas de instruções de desvio condicional.

Tabela 8 – Instruções do 8051 para o desvio condicional do fluxo de execução do programa.

MNEMÔNICO	DESCRIÇÃO
JZ <rel addr>	Salta se A = 0
JNZ <rel addr>	Salta se A != 0
JC <rel addr>	Salta se C = 1
JNC <rel addr>	Salta se C != 1
JB <bit>, <rel addr>	Salta se bit = 1
JNB <bit>,<rel addr>	Salta se bit != 1
JBC <bit>, <rel addr>	Salta se bit =1, limpa bit
CJNE A, direct, <rel addr>	Compara A e memória. Salta se não igual
CJNE A, #data, <rel addr>	Compara conteúdo de A e dado. Salta se não igual
CJNE Rn, #data, <rel addr>	Compara conteúdo de Rn e dado. Salta se não igual
CJNE @Ri, #data, <rel addr>	Compara conteúdo da posição de memória interna apontada Ri e dado. Salta se não igual
DJNZ Rn, <rel addr>	Decrementa Rn e salta se não zero
DJNZ direct, <rel addr>	Decrementa memória e salta se não zero

5. Subrotinas

Quando um conjunto de linhas de código é muito utilizado ao longo de um programa, é usual disponibilizá-lo como uma subrotina de forma a melhorar a legibilidade do código, bem como compactar o código, visando a menor ocupação da memória de programa. A Figura 13 ilustra como isto é realizado.

Neste exemplo, se o número de instruções (Ninst) repetidas for 40, por exemplo, a implementação do programa com rotina é reduzido de, aproximadamente, 80 linhas de instrução. Tal redução de tamanho do código original obtido com o uso de subrotina pode proporcionar que o código resultante possa ser comportado na memória de programa interna sem a necessidade de acrescentar memória externa ao sistema. Tem-se também ganhos em relação à legibilidade do código, pois uma vez que a função da subrotina tenha sido compreendida, outra menção à mesma no programa não requer a sua nova análise. Como desvantagem, tem-se que a alteração do fluxo de execução do programa consome algum tempo (execução de procedimentos adicionais pela UC do 8051); pode portanto, não ser recomendável quando se requer maior velocidade na execução de tarefas pelo processador.

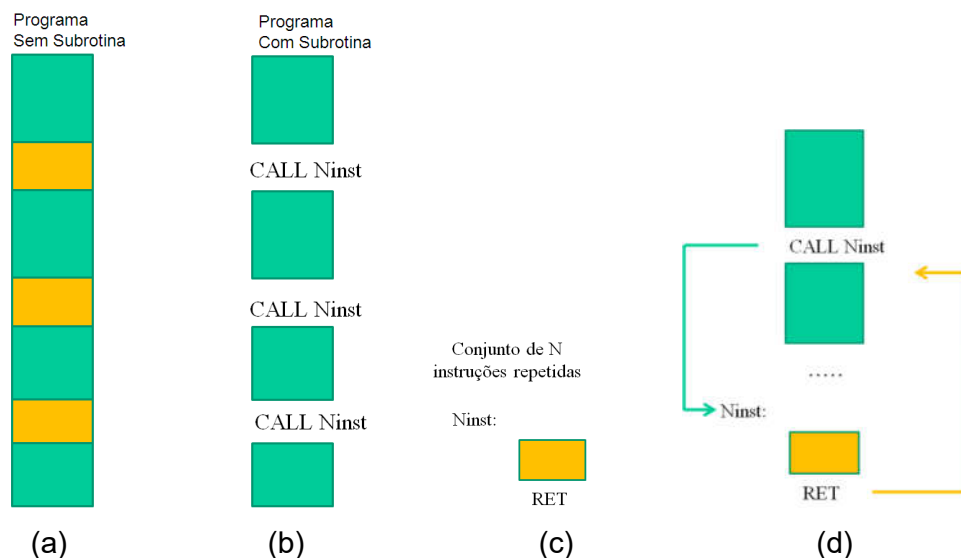


Figura 13 – (a) Esquema de programa com trechos de código que se repetem (laranja); (b) Esquema de programa onde os trechos de código que se repetiam (c) foram substituídos por instrução (CALL e endereço da subrotina) para a execução do (c) código da subrotina; (d) esquema da alteração do fluxo de execução do programa quando do uso de subrotinas.

Como já mencionado, o 8051 coloca o atual valor do PC no barramento de endereços e realiza a leitura de byte de instrução da memória de programa; terminada a leitura deste byte, o registrador PC é incrementado tal que o PC sempre possua o endereço de byte de instrução a ser lido da memória de programa.

Como ilustrado na Figura 13, o trecho de código da subrotina é retirado de sua localização original no programa e colocado em outra região da memória de programa. Assim, há a necessidade de alterar o fluxo de execução do programa tal que a próxima instrução a ser executada pelo 8051 após a chamada da subrotina (CALL) não seja aquela que se segue à instrução CALL, mas sim, as instruções pertencentes à subrotina. Enfatizando, a função da instrução CALL é alterar o fluxo de execução do código do 8051 tal que, após a sua execução, o próximo byte de instrução seja lido da subrotina.

A instrução CALL é acompanhada do endereço da memória de programa na qual o código da subrotina se encontra, sendo este o endereço para o qual o fluxo de programa deve ser desviado. O código da subrotina é encerrado com a instrução RET (*return*) que demanda que o 8051 volte a executar o programa principal a partir da instrução que se segue a instrução CALL.

Observação: A bem da verdade, existem duas instruções do 8051 para solicitar a execução de subrotina. São elas:

LCALL seguida de endereço de 16 bits. A instrução possui 3 bytes (opcode + 16 bits de endereço). A subrotina pode se encontrar em qualquer lugar da memória de programa (64 kiB).

ACALL seguida de endereço de 11 bits. A instrução possui 2 bytes (opcode + 11 bits de endereço). A subrotina deve se localizar até 2048 bytes (2^{11}) distante da instrução.

Contudo, o programador pode utilizar a instrução CALL (que não existe no conjunto de instruções do 8051) e deixar que o compilador a substitua pelo código da instrução mais adequada. A instrução LCALL (3 bytes) é utilizada quando a rotina estiver distante (além de 2048 bytes) da instrução CALL, pois ocupa 1 byte a mais da memória de programa.

Observe no esquema da Figura 13 que, embora a instrução que solicita a execução da rotina seja a mesma ao longo de todo o código (CALL e endereço da subrotina), cada vez que a subrotina é executada, o 8051 passa a executar a instrução do código principal que se segue a última chamada da subrotina.

Como o processador sabe para qual endereço retornar após a execução da subrotina, haja vista que há diferentes chamadas da mesma subrotina ao longo do programa principal? Para saber para onde retornar, o processador faz uso da pilha.

6. Pilha

A pilha é uma região da memória interna de dados (iRAM) utilizada pelo processador e programador ao longo da execução do programa para armazenar dados temporariamente. O processador a utiliza para armazenar endereços quando da alteração do fluxo de execução do programa (subrotina e interrupção). O programador a utiliza para armazenar dados temporários, como por exemplo, para passar parâmetros do programa principal para o código da subrotina.

O registrador denominado ponteiro de pilha (*Stack Pointer* – SP) indica onde dados serão alocados e retirados da pilha. Quando o 8051 é energizado ou resetado, o registrador SP é inicializado com o valor 07H.

A Figura 14 mostra um exemplo de utilização da pilha quando da chamada de subrotina. Após a leitura dos 3 bytes que compõem a instrução LCALL 2028H, o PC passa a conter o endereço da próxima instrução a ser lida do programa principal: 000DH. Porém, a execução da instrução LCALL 2028H faz com que o 8051 sobrescreva o PC com o valor 2028H (ou seja, o endereço do primeiro byte do código da subrotina). Para que o 8051 saiba para qual endereço retornar no programa principal após a execução da subrotina, antes de sobrescrever o PC com 2028H, o atual conteúdo do PC (neste exemplo, 000DH) é salvo na pilha. Para tal, o SP é incrementado e o byte LSB do PC (0DH) é armazenado no endereço apontado pelo SP (08H). Em seguida, o SP é incrementado uma vez mais e o byte MSB do PC (00H) é armazenado no endereço apontado pelo SP (09H). O conteúdo do PC é então sobrescrito com o endereço da subrotina (2028H). O 8051 passa, então, a ler e executar as instruções contidas na memória de programa a partir deste endereço.

END.	CÓDIGO	PILHA	END.	DADO
000A	LCALL 2028H	SP	07H	
000D	MOV A,B	SP+1	08H	0DH [PC LSB]
		SP+1	09H	00H [PC MSB]

Figura 14 – Exemplo de colocação do endereço de retorno da subrotina na memória de dados interna apontada pelo registrador SP.

Sucintamente, os eventos que ocorrem durante execução de LCALL são:

$[PC] \leftarrow [PC] + 3$; PC é incrementado durante a leitura de cada um dos 3 bytes que compõem a instrução LCALL 2028H; a execução da instrução é iniciada:

$[SP] \leftarrow [SP] + 1$; Conteúdo do registrador SP é incrementado;

$[[SP]] \leftarrow [PC_{LSB}]$; LSB do PC é armazenado no endereço de memória apontado pelo conteúdo de SP;

$[SP] \leftarrow [SP] + 1$; Conteúdo do registrador SP é incrementado;

$[[SP]] \leftarrow [PC_{MSB}]$; MSB do PC é armazenado no endereço de memória apontado por SP;

$[PC] \leftarrow \text{addr}_{15-0}$; PC é sobrescrito com o endereço da subrotina.

Onde se adotou a seguinte notação: [] corresponde ao conteúdo do registrador; { } corresponde ao endereço de memória apontado por

Pelo fato do processador guardar na pilha o endereço da instrução que se segue àquela que solicita a alteração do fluxo de execução do programa (LCALL neste exemplo), o processador consegue recuperar o endereço da instrução do programa principal a ser executada após as tarefas da subrotina. A instrução RET (*return*) encerra o código de sub-rotinas; ao ler o código desta instrução (que possui um byte), o 8051 incrementa o PC e executa a instrução. A instrução RET faz com que o 8051 leia o conteúdo da posição de memória

apontada pelo SP, carregue-o no registrador MSB do PC e decremente o SP; em seguida, copia o conteúdo da posição de memória apontada pelo SP no registrador LSB do PC e decrementa o SP.

Enfatizando, quando do retorno da subrotina para o programa principal (execução da instrução RET), os seguintes eventos ocorrem:

$[PC] \leftarrow [PC] + 1$; PC é incrementado após a leitura do byte de instrução correspondente a instrução RET; a execução da instrução é iniciada:

$[PC_{MSB}] \leftarrow [[SP]]$; Conteúdo da posição de memória apontada por SP é carregado no MSB do PC;

$[SP] \leftarrow [SP] - 1$; Conteúdo do registrador SP é decrementado;

$[PC_{LSB}] \leftarrow [[SP]]$; Conteúdo da posição de memória apontada por SP é carregado no LSB do PC;

$[SP] \leftarrow [SP] - 1$; Conteúdo do registrador SP é decrementado.

Para o exemplo da Figura 14, após o término da execução da instrução RET contida no código da subrotina, tem-se: $[PC] = 000DH$ e $[SP] = 07H$. Assim, o 8051 volta a executar a instrução que se encontra após a chamada da subrotina no programa principal.

Como já comentado, o registrador SP é inicializado com o valor 07H quando o 8051 é energizado ou resetado. Assim, caso seu programa utilize o banco de registradores 1, o registrador SP deve ser carregado com novo valor que corresponda a endereço de memória de dados interna não utilizado pelo programa; geralmente, isto é realizado no início do programa. Deve-se também tomar cuidado com a pilha para que esta não cresça excessivamente tal que invada região da memória utilizada para armazenar outros dados necessários para a execução do programa ou exceda o seu limite (7FH).

Caso haja uma chamada de subrotina dentro da execução do código de uma outra subrotina, o endereço de retorno para a primeira subrotina é colocado na pilha sobre o endereço de retorno para o programa principal. Como os bytes são colocados uns sobre os outros na memória apontada por SP (sob a perspectiva dos endereços mais significativos para os menos significativos), esta região de memória é denominada de pilha. Neste caso (chamada de

subrotina dentro de uma subrotina), a primeira ocorrência de instrução RET retira os últimos 2 bytes que se encontram na pilha de tal forma a sobrescrever o PC com o endereço para o qual retornar na primeira subrotina; a segunda ocorrência de RET retira os outros dois bytes na pilha, fazendo com que o fluxo de execução do programa retorne ao programa principal.

Uma situação que pode ocorrer é o código da subrotina utilizar um registrador que está sendo utilizado pelo código do programa principal, como por exemplo, o registrador A. Neste contexto, o programador pode transferir o conteúdo do registrador A para uma posição de memória de dados e depois, recuperar este conteúdo de memória para A. Tem-se, portanto, de reservar uma posição de memória para esta finalidade. Uma alternativa é guardar o dado na pilha antes de se chamar a subrotina. Após o retorno ao programa principal, o dado deve ser recuperado na pilha. Esta abordagem evita que uma posição da memória fique comprometida apenas com a finalidade de armazenar dado temporariamente, não podendo ser utilizada para outra finalidade. As instruções do 8051 para colocar e retirar dados da pilha são, respectivamente, PUSH e POP. A Figura 15 mostra o código e as alterações na pilha após a execução destas instruções, assumindo (como exemplo) os seguintes conteúdos iniciais para os registradores SP=[2FH] e A=[CAH].

END. (HEX.)	Mnemônico		PILHA	END.	DADO
010E	PUSH ACC		SP	2F	
0200	LCALL 34C2H	PUSH ACC	SP+1	30	CA
0203	...	LCALL 34C2	SP+1	31	03
			SP+1	32	02

Figura 15 – Exemplo de colocação do conteúdo de A (PUSH ACC) e de endereço de retorno da subrotina na pilha (LCALL 34C2H). O registrador A é referenciado por ACC para que este símbolo seja substituído pelo seu endereço da região dos SFRs pelo compilador.

Ao retornar ao programa principal, o conteúdo do acumulador é recuperado da pilha pela instrução POP ACC (Figura 16). Quando então, os seguintes eventos ocorrem:

$[A] \leftarrow [[SP]]$; ; Conteúdo da posição de memória apontada por SP é carregado no acumulador
 $[SP] \leftarrow [SP] - 1$; Conteúdo do registrador SP é decrementado

Assim, os conteúdos finais dos registradores SP e A passam a ser iguais aos seus conteúdos iniciais $SP=[2FH]$ e $A=[CAH]$.

A retirada de dados da pilha não precisa ser realizada através do mesmo registrador cujo conteúdo foi copiado para a pilha; por exemplo, na Figura 16 o conteúdo poderia ser retirado para o registrador B (POP B) em vez de se utilizar duas instruções, POP ACC e MOV B,A.

END.	Mnemônico
010E	PUSH ACC
0200	LCALL 34C2H
0203	POP ACC
0205	MOV B,A

Figura 16 – Exemplo de recuperação de dado colocado na pilha pela instrução PUSH. Ao término da execução desta instrução, o conteúdo de SP será idêntico ao anterior à execução deste código.

Ao gerar o código, o compilador não analisa o mesmo para saber qual banco de registradores está sendo utilizado pelo programador. Por esta razão, não é possível referenciar o nome de registradores em instrução de PUSH e POP; estas instruções utilizam endereço direto. Assim, nestas instruções, deve-se especificar o endereço sendo utilizado pelo registrador; ou seja, para receber dado armazenado na pilha no registrador R1 do banco 2, a instrução é POP 11H. Utiliza-se ACC em vez de A nas instruções de PUSH e POP para referenciar o acumulador; assim, o compilador substitui ACC pelo endereço que este ocupa na memória SFR. No Assembly, instruções que utilizam A em vez de ACC não utilizam o endereço do acumulador, mas o uso deste fica implícito no código binário da instrução.

Caso o programa principal esteja utilizando os registradores R0 a R7 do banco de registradores 0 e chame uma subrotina que também empregue estes mesmo registradores (R0 a R7) para execução de tarefa não relacionada a atividade do programa principal, haveria necessidade de salvar todos os registradores na pilha através do código da subrotina (instruções PUSH) e recuperá-los posteriormente (instruções POP). Ver primeira versão da subrotina Ninst no quadro a seguir. Outra alternativa é estabelecer que a subrotina trabalhe com um banco de registradores diferente daquele utilizado pelo programa principal; isto evita o uso de múltiplas instruções PUSH e POP, gerando código de mais rápida execução e que ocupa menor espaço na memória de programa. No quadro mostrado a seguir, apresenta-se código alternativo, supondo que o programa principal trabalhe com registradores do banco 0; estabelece-se que a rotina trabalhe com o banco 1.

; SUBROTINA Ninst			
Ninst:	PUSH	00H	
	PUSH	01H	
	;....		(mais PUSHes)
	PUSH	07H	
	;....		(CÓDIGO DA SUBROTINA)
	POP	07H	
	;....		(mais POPs)
	POP	01H	
	POP	00H	
	RET		
; NOVA VERSÃO - SUBROTINA Ninst usando Banco 1			
Ninst:	SETB	RS0	
	;....		(CÓDIGO DA SUBROTINA)
	CLR	RS0	
	RET		

7. *Polling* e Interrupção

Suponha que você tenha um semáforo controlado por um processador responsável por alternar o acendimento das lâmpadas verde, amarelo e vermelho. Este processador pode ainda ter que atuar em resposta a sinais apresentados em seus pinos de entrada por dispositivos externos, como por exemplo, sinal gerado por radar (para informar que carro excedeu limite de velocidade permitido) e/ou sinal gerado por botoeira para solicitar abertura do semáforo para pedestre.

Assim como esta, existem diversas outras situações práticas em que o processador deve executar tarefas ocasionais solicitadas por dispositivos externos (ou mesmo, eventos internos), abandonando momentaneamente, a execução das tarefas principais para retomá-las posteriormente.

Há duas estratégias comumente adotadas para verificar se um dispositivo demanda atuação por parte do processador: *Polling* e Interrupção.

Polling corresponde ao teste periódico do nível lógico presente nos pinos de entrada (ou de algum registrador em determinados contextos) para verificar se algum dispositivo demanda atuação do processador. Esta estratégia faz com que o desempenho do processador decaia na execução das tarefas do programa principal.

Outra estratégia é a interrupção quando o hardware do processador desvia o fluxo de execução do programa principal para realizar outra tarefa apenas quando outro dispositivo demandar a sua execução. O código executado em resposta à solicitação da interrupção é denominado de tratador de interrupção. A Figura 17 ilustra tal evento. Deve-se observar que a interrupção pode ocorrer durante a execução de qualquer instrução do programa principal, tendo em vista que não é possível saber quando uma interrupção será solicitada. Por esta razão, o código do tratador de interrupção não deve alterar o conteúdo de registradores de propósito geral para evitar que o programa principal deixe de

funcionar adequadamente. Exemplificando, se o programa principal estiver utilizando o registrador A e ocorrer interrupção habilitada, o tratador será executado; se o tratador modificar o conteúdo de A, ao retornar para o programa principal, o código do programa principal terá um comportamento inesperado em função desta alteração de conteúdo de A.

Mais genericamente, se ocorre interrupção habilitada, o processador:

1. Conclui a leitura e execução da instrução sendo processada, atualizando o PC para apontar para a próxima instrução do programa principal;
2. Salva contexto (alguns processadores salvam *flags* e certos registradores) e endereço do atual valor do PC na pilha. No 8051, salva-se na pilha, apenas o atual conteúdo do PC. Para tal, o 8051 incrementa o registrador SP e armazena no endereço da pilha apontado pelo SP, o byte menos significativo (LSB) do PC; incrementa o registrador SP uma vez mais e armazena no endereço da pilha apontado pelo SP, o byte mais significativo (MSB) do PC;
3. Carrega o endereço do tratador de interrupção no PC;
4. Executa o tratador de interrupção;
5. Recupera da pilha o endereço da instrução posterior àquela que estava sendo executada quando a interrupção foi solicitada; recupera contexto do programa principal (*flags* e registradores) e continua execução do programa principal. No 8051, ao executar a instrução RETI, este carrega no byte mais significativo do PC, o conteúdo da pilha cujo endereço é apontado pelo SP, decrementa o registrador SP; carrega no byte menos significativo do PC o conteúdo da pilha cujo endereço é apontado pelo SP, decrementa o registrador SP.

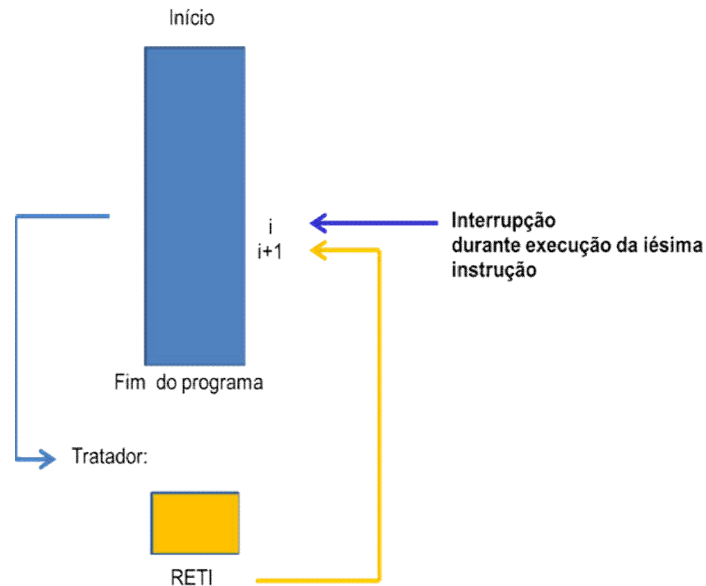


Figura 17 – Representação do desvio do fluxo de execução de programa decorrente da solicitação de interrupção. A i -ésima instrução do programa principal estava sendo executada. Após o retorno do tratador de interrupção, o processador volta a executar as instruções do programa principal a partir a $(i+1)$ instrução.

Deve-se salientar que, em muitos processadores (como por exemplo, o 8051), o endereço do tratador de interrupção carregado no PC (passo 3 acima) é definido pelos projetistas do processador, sendo o mesmo informado na folha de dados do processador. Assim, o programador deve ter o cuidado de alocar o tratador (ou instrução de desvio para o tratador) no endereço especificado na folha de dados do processador.

Outros processadores podem possuir registradores associados a cada fonte de interrupção que devem ser inicializados pelo programador com o endereço onde o tratador está alocado na memória de programa.

Ao tratar a interrupção, o 8051 não salva (e, portanto, não recupera), automaticamente, o valor de qualquer outro registrador na pilha que não seja o PC; no entanto, o programador pode fazê-lo no código do tratador. A instrução RETI, de forma idêntica a instrução RET, recupera da pilha o endereço para o qual o 8051 deve retornar no programa principal (carrega tal endereço no PC); além disto, esta instrução limpa *flag* interno da UC para informar que o tratador foi executado e que, portanto, outra interrupção pode ser atendida. Assim,

deve-se tomar cuidado em utilizar a instrução RETI e não RET para retornar do tratador de interrupção. Caso contrário, novas solicitações de interrupções com mesmo nível de prioridade não serão atendidas, pois a máquina de estados do 8051 não é informada de que a execução do tratador foi concluída.

O 8051 atende interrupções externas em resposta a colocação do pino P3.2 (/INT0 - Interrupção Externa 0) e P3.3 (/INT1 - Interrupção Externa 1) em nível lógico baixo, por borda ou nível (Figura 20). Além destas duas fontes de interrupção externa, existem outras três fontes de interrupção devido a dispositivos internos. A Tabela 9 informa quais são estas fontes de interrupção e o endereço da memória de programa para o qual o fluxo de execução do programa é desviado em resposta à ocorrência de cada uma delas.

Tabela 9 – Fontes de interrupção do 8051 e endereços nos quais os tratadores devem ser alocados na memória de programa.

INTERRUPÇÃO	Endereço do Tratador (Hex)
Externa 0	0003
Timer 0	000B
Externa 1	0013
Timer 1	001B
Serial	0023

Exemplificando, caso a interrupção externa 1 do 8051 seja solicitada, o PC será carregado com o endereço 0013H. O programador deve, portanto, alocar o tratador de interrupção neste endereço.

Durante a execução de certos trechos do programa principal, o atendimento de interrupção pode ser indesejável por comprometer o desempenho da tarefa em andamento. O hardware possui mecanismos para contornar tal situação, permitindo que o programador habilite/desabilite fontes de interrupção.

Para habilitar o atendimento das interrupções do 8051, o programador deve atuar sobre o registrador *Interrupt Enable* (IE). O flip flop 7 (*Enable All*) do registrador IE (Figura 18) funciona como uma chave geral para as interrupções do 8051; ou seja, se ele estiver em '0', nenhuma interrupção é atendida. Além de colocar o flip-flop EA em '1', o flip-flop correspondente à interrupção que se deseja habilitar (Figura 18) deve ser também colocado em nível lógico alto. Por exemplo, a habilitação da Interrupção Externa 1 pode ser realizada através das seguintes instruções: SETB EX1, SETB EA.

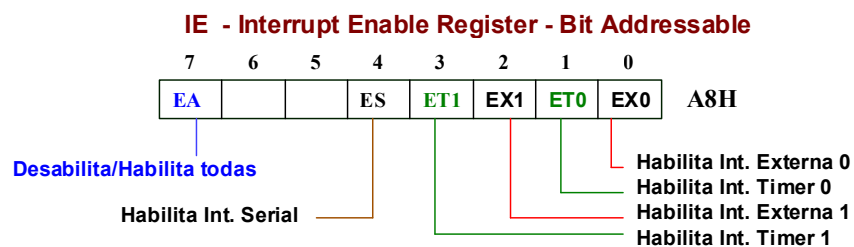


Figura 18 – Registrador *Interrupt Enable*. Deve-se colocar '1' em cada flip-flop para se obter a função descrita.

Para que se desabilite uma interrupção específica, o flip-flop correspondente a interrupção que se deseja desabilitar deve ser colocado em nível lógico baixo. Por exemplo, a Interrupção Externa 0 pode ser desabilitada através da seguinte instrução: CLR EX0; opcionalmente, pode-se executar a instrução CLR EA, desabilitando conjuntamente o atendimento de todas as interrupções.

Caso todas as interrupções estejam habilitadas, existe a possibilidade de mais de uma fonte de interrupção solicitar, simultaneamente, a atenção da Unidade de Controle (UC). Os projetistas do 8051 estabeleceram um nível de prioridade entre as interrupções para arbitrar qual interrupção será primeiramente atendida. A Tabela 10 mostra a ordem *default* de atendimento das interrupções, caso todas elas estejam habilitadas e sejam simultaneamente solicitadas. Os flip-flops dos registradores TCON e SCON (Figura 20 e Seção 9) que indicam solicitação de execução dos respectivos tratadores são informados na segunda coluna da Tabela 10.

Tabela 10 – Ordem *default* de prioridade de atendimento das interrupções do 8051 e flip-flops dos registradores TCON e SCON que indicam (colocados em nível lógico alto) que ocorreu solicitação de execução dos respectivos tratadores.

Fontes de Interrupção	Flip-flop que solicita execução do tratador	Nível de Prioridade
Externa 0	IE0	Mais Alta
Timer 0	TF0	
Externa 1	IE1	
Timer 1	TF1	
Serial	TI e/ou RI	Mais Baixa

Contudo, a ordem de prioridade de atendimento das interrupções apresentadas na Tabela 10 pode ser inadequada em um dado projeto. O registrador *Interrupt Priority* (IP) apresentado na Figura 19 permite alterar a prioridade default de atendimento das interrupções (todas elas possuem nível de prioridade 0 quando o 8051 é resetado). Ao setar um dos flip-flops, a interrupção correspondente passa a ter maior prioridade de atendimento (nível de prioridade 1). Se dois flip-flops estiverem setados no registrador IP, as interrupções correspondentes têm maior prioridade (nível de prioridade 1) que as de prioridade 0; entre as interrupções de nível de prioridade 1, a maior prioridade de atendimento obedece a ordem da Tabela 10. Por exemplo, caso os flip-flops PX1 e PT1 do registrador IP estejam setados (os demais flip-flops em '0') e todas as interrupções estejam habilitadas, a simultânea solicitação de todas as interrupções terá como ordem de execução dos tratadores: Externa 1, Timer 1, Externa 0, Timer 0, Serial.

Outra característica é que interrupção com nível de prioridade 1 interrompe a execução de tratadores com nível de prioridade 0, caso a de nível 1 seja solicitada durante a execução do tratador de interrupção com nível de prioridade 0. Interrupções não interrompem a execução de tratadores com mesmo nível de prioridade (1 ou 0). Devido a esta característica, o programador deve ter o cuidado de escrever tratadores curtos tal que outra

fonte de interrupção com o mesmo nível de prioridade não tenha que aguardar um longo tempo para ser atendida. Caso contrário, o tempo de resposta do processador para executar uma dada tarefa pode não ser adequado para o projeto em questão.

Como geralmente não se sabe quando o tratador de interrupção será executado, o código do tratador não deve alterar o conteúdo dos registradores de propósito geral. A razão é que, em código com alguma complexidade, estes registradores devem estar sendo utilizados no programa principal ou em alguma subrotina; se o valor de algum destes registradores fosse alterado pelo tratador de interrupção, a execução do código interrompido que faz uso deste registrador irá se comportar de forma inesperada, pois o valor do registrador modificado será, obviamente, diferente daquele antes da execução do tratador. Portanto, adota-se como boa prática de programação, que o código de tratadores de interrupção não deve alterar o conteúdo de registradores de propósito geral. É possível utilizar tais registradores no tratador, mas se deve salvar o conteúdo destes na pilha e depois, recuperá-los tal que os valores originais, ao término da execução do tratador, sejam os mesmos.

Caso instruções do tratador alterem os flags, deve-se salvar o PSW antes de executar estas instruções e recuperá-lo pouco antes da instrução RETI.

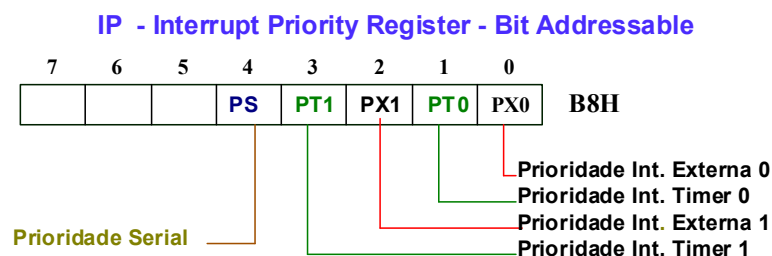


Figura 19 – Registrador *Interrupt Priority* (IP). Deve-se colocar o flip-flop em '1' para aumentar o nível de prioridade da fonte de interrupção correspondente.

O registrador TCON (Figura 20) além de registrar pendência de interrupções a serem atendidas pela UC da UCP (faça correspondência do mesmo com a Tabela 10), especifica se as interrupções externas 1 e 0 devem ser atendidas por borda de descida (IT1='1'; IT0='1') ou por nível lógico baixo (IT1='0';

IT0='0'). Neste segundo caso, o dispositivo externo que solicita interrupção deve manter o nível lógico baixo do pino /INT1 (ou /INT0) durante 12 ciclos de clock para que o flip-flop IE1 (ou IE0) seja setado, demandando a execução do tratador (caso a interrupção correspondente tenha sido habilitada). Os flip-flops TR1 e TR0 serão discutidos posteriormente (Seção 8).

TCON - Timer Control Register – Bit Addressable

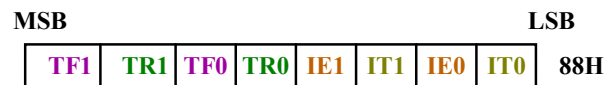


Figura 20 – Registrador *Timer Control* (TCON). Os flip-flops TF1, TF0, IE1, IE0 em nível lógico '1' indicam pendência de atendimento de interrupção (consultar Tabela 10). Os flip-flops IT1 e IT0 em nível lógico '1' especificam que as interrupções externas 1 e 0 (respectivamente) são solicitadas por borda de descida. Os flip-flops TR1 e TR0 estão relacionados à habilitação da contagem dos Timers 1 e 0, respectivamente (Seção 8).

Para exemplificar os aspectos discutidos nesta seção, apresenta-se a implementação de programa que aceita a Interrupção Externa 0 (int0) acionada por borda de descida. A cada ocorrência da int0, escreve-se na porta P1, caractere por caractere, a cadeia de 16 caracteres 'Microcontrolador' até que toda ela seja enviada.

Este exemplo reforça importantes aspectos em relação à escrita de um tratador de interrupção:

- ❖ Observar o endereço da memória de programa no qual o tratador deve ser alocado para cada fonte de interrupção;
- ❖ Código do tratador deve ser de rápida execução para não haver tempo de resposta inadequado para outras fontes de interrupção;
- ❖ Ao retornar do tratador para o programa principal, os registradores de propósito geral devem ter valores iguais àqueles que possuíam antes de executar o tratador; portanto, o tratador

não deverá alterar o conteúdo de registradores de manipulação de dados.

- ❖ Ao término de sua execução, o tratador não deverá ter alterado o ponteiro de pilha: SP (ao executar RETI, SP deve ser ter valor igual àquele com o qual iniciou a execução do tratador);

RESET	EQU	00H	
LTEXT0	EQU	03H ; LOCAL TRATADOR	
STATE	EQU	20H	
	ORG	RESET	;pc=0 depois de reset
	JMP	INICIO	
	ORG	LTEXT0	
INICIO:	JMP	HANDLER	
	MOV	IE,#10000001B	; habilita interrupção
	MOV	TCON,#00000001B	; interrupção por borda
	MOV	STATE,#0H	; inicialização de variável
	MOV	R0,# STATE	
	MOV	DPTR,#TABELA	
	MOV	R1,#0	
VOLTA:	CJNE	@R0,#1,VOLTA	
	MOV	STATE,#0H	
	MOV	A,R1	
	MOVC	A,@A+DPTR	
	MOV	P1,A	
	INC	R1	
	CJNE	R1,#16,VOLTA	
	JMP	\$	
HANDLER:	MOV	STATE,#1H	; informa ocorrência de interrupção
	RETI		
TABELA:	DB	'Microcontrolador'	
	END		

8. Temporizadores e Contadores do 8051

São características do hardware do 8051 representado na Figura 21:

- CPU de 8 bits;
- 4 portas de entrada e saída (8 pinos cada);
- 4 KiB de memória ROM interna para programas;
- 128 bytes de memória RAM interna para dados;
- endereça 64 KiB de memória de programa externa;
- endereça 64 KiB de memória de dados externa;
- 5 vetores de interrupção com 2 níveis de prioridade:
 - 2 interrupções externas
 - 2 temporizadores / contadores
 - 1 interface serial

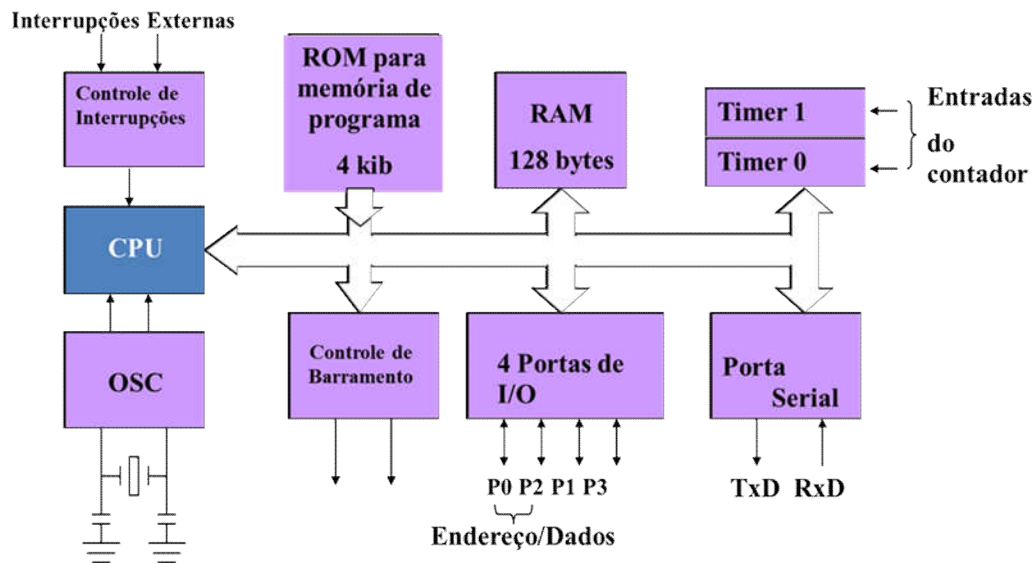


Figura 21 – Diagrama dos diferentes módulos compreendidos pelo 8051.

Falta, portanto, ainda descrever o funcionamento dos contadores /temporizadores e da interface serial do 8051 para que se tenha uma completa visão do potencial deste microcontrolador.

Contador/Temporizador é um módulo do hardware projetado para contar eventos externos (por exemplo, número de pessoas que passam pela catraca de um terminal de ônibus) e intervalos de tempo (por exemplo, para cronometrar o tempo que um atleta demora para percorrer 100m). Esta seção descreve os contadores/temporizadores do 8051; a partir desta descrição, você pode imaginar como utilizá-los em projetos, bem como os circuitos eletrônicos complementares. Outros microcontroladores possuem tais módulos com características bem mais aprimoradas. No entanto, a partir da compreensão deste módulo do 8051, o aprendizado do funcionamento de outros tornar-se-á mais intuitivo.

O 8051 tem dois contadores/temporizadores de 16 bits (*Timer 0* e *Timer 1*) sendo que cada um deles pode operar em 4 modos diferentes. A seleção da forma de funcionamento (temporizador ou contador), bem como do modo de operação é especificada pelo registrador TMOD (Figura 22). Os 4 flip-flops mais significativos do registrador TMOD controlam o funcionamento do contador/temporizador 1; os 4 flip-flops menos significativos deste registrador controlam o contador/temporizador 0. Para ambos os contadores/temporizadores, o flip-flop C/T define a forma de funcionamento: contador (C/T= '1') ou temporizador (C/T= '0'). Através dos flip-flops M1 e M0, especifica-se um dos 4 modos de operação para cada contador/temporizador. Nos modos de 0 a 2, os dois contadores/temporizadores funcionam de forma similar; no modo 3, os funcionamentos são diferentes.

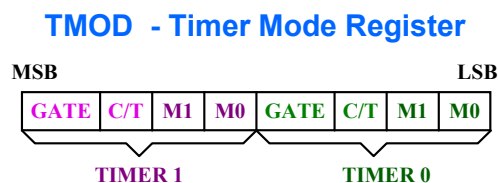


Figura 22 - TMOD: registrador de controle dos contadores / temporizadores.

A Figura 23 apresenta o diagrama em blocos dos 2 temporizadores/contadores quando funcionando nos modos 0 e 1. Nesta figura, a letra **x** que compõe a designação dos flip-flops/registadores deve ser substituída por 0, quando se deseja trabalhar com o contador/temporizador 0, ou por 1, quando se deseja trabalhar com o contador/temporizador 1.

A seguir, irá se trabalhar com o **TIMER1** para evitar qualquer confusão por parte do leitor. Os mesmos procedimentos devem ser adotados para o **TIMER0** para os modos de 0 a 2, substituindo-se 1 por 0 na designação dos flip-flops/registadores.

No modo 1 ($M1='0'$; $M0='1'$), os registradores **TL1** (menos significativo) e **TH1** (mais significativo) são vistos como um contador de 16 bits (**TH1-TL1**). O flip-flop **C/T** de **TMOD** seleciona a fonte de *clock* para **TH1-TL1**.

Na função temporizador ($C/T= '0'$), o contador de 16 bits é incrementado a cada ciclo de instrução. Como cada ciclo de instrução dura 12 períodos de clock gerados pelo oscilador, os temporizadores contam pulsos de clock a uma taxa de 1/12 da frequência do cristal.

Na função contador ($C/T= '1'$), o contador de 16 bits é incrementado em resposta a uma transição de '1' para '0' em pino de entrada **T1** (Figura 23) pertencente à Porta 3 (Figura 2). Os níveis lógicos presentes nestes pinos são amostrados a cada ciclo de instrução, sendo o contador de 16 bits incrementado apenas no ciclo seguinte. Portanto, a frequência máxima de contagem da forma de onda aplicada a **T1** é de 1/24 da frequência do cristal.

O sinal de *clock* é apresentado para a contagem em **TH1-TL1** quando o sinal **CONTROL** (Figura 23) for igual a '1'. Para tal, o flip-flop **TR1** do registrador **TCON** deve ser colocado em '1' pelo programador através da instrução **SETB TR1**. Além disto, a outra entrada da porta **AND** (Figura 23) deve também estar em '1'. Isto pode ser realizado por software, colocando o flip-flop **GATE** de **TMOD** em '0'. Caso **GATE** seja igual a '1', a contagem irá acontecer apenas quando forma de onda apresentada ao pino **/INT1** estiver em nível lógico '1';

este modo é utilizado para medir largura de pulso em nível lógico '1' apresentado ao pino /INT1.

Sendo o sinal de *clock* apresentado ao contador TH1-TL1, a contagem é realizada até FFFFH. Quando do próximo incremento, os registradores TH1-TL1 vão para zero e o *flag* TF1 é colocado em nível lógico alto, solicitando execução de tratador de interrupção devido ao TIMER1 (Tabela 10). Para que o tratador seja executado, tal interrupção deverá ter sido habilitada no registrador IE (Figura 18).

Portanto, para se gerar uma interrupção (passagem de FFFFH para 0 em TH1-TL1) ao término de uma contagem de 20 ciclos de instrução, deve-se inicializar TH1-TL1 com o fundo de escala subtraído de 20, ou seja, $2^{16}-20=65516=FFECH$ (MOV TH1,#0FFH; MOV TL1,#0ECH).

O modo 0 (M1='0'; M0='0') tem funcionamento similar ao modo 1. A diferença é que o registrador TH1 (8 bits) é incrementado apenas quando TL1 excede uma contagem de 5 bits (ou seja, quando TL1 está 11111B e recebe novo sinal de *clock*, indo para zero). Desta forma, TL1 pode ser visto como um divisor do sinal de *clock* por 32 (2^5). Assim, TH1 e TL1 constituem um contador de 13 bits: 5 bits menos significativos de TL1 e os 8 bits mais significativos de TH1. Quando de um próximo incremento, o registrador TH1 for de FFH para zero, o *flag* TF1 é colocado em nível lógico alto, solicitando execução do tratador de interrupção devido ao TIMER1. Esse modo tem por objetivo manter compatibilidade com a família anterior de microcontroladores da Intel (MCS48).

Resumindo, os passos para gerar uma interrupção pelo contador/temporizador, compreendem a observação dos seguintes passos:

- 1) Alocar tratador para o contador/temporizador no seu devido endereço;
- 2) Habilitar interrupção do contador/temporizador no registrador IE (EA, ETx);
- 3) Especificar o modo de funcionamento do contador/temporizador (TMOD);
- 4) Especificar intervalo de contagem (THx e TLx);
- 5) Iniciar contagem (setb TRx);
- 6) Ao término da contagem, realizar recarga nos modos: 0, 1 e 3.

OBS: Caso se deseje medir largura de pulso de sinal aplicado em /INT1, fazer GATE='1'. A contagem ocorrerá durante intervalo de tempo no qual o pino /INT1 estiver em nível lógico alto.

No modo 2 ($M1=1$; $M0=0$), o registrador TL1 funciona como um contador de 8 bits sendo que, ao término da contagem (ou seja, quando o conteúdo for igual a FFH e receber novo sinal de *clock*), o mesmo é carregado com o conteúdo do registrador TH1 (Figura 24) e o *flag* TF1 é colocado em '1', demandando execução do tratador de interrupção associado ao TIMER1. TL1 continua a contagem a partir do valor carregado de TH1, sendo que TH1 não é alterado. Assim, como nos modos 0 e 1, o funcionamento é o mesmo para *clock* oriundo do cristal/12 (temporizador) ou do pino T1 (contador).

Quando o Temporizador/Contador 0 é colocado no modo 3 ($M1=1$; $M0=1$), os registradores TH0 e TL0 funcionam como dois contadores independentes de 8 bits (Figura 25). Os flip-flops TR1 e TF1 do registrador TCON passam a atuar no controle da contagem realizada pelo registrador TH0; TR1='1' habilita a contagem por parte de TH0 e TF1='1' demanda execução do tratador do TIMER1 agora associado ao *overflow* de TH0 (FFH para 00H).

Quando colocado no Modo 3, os registradores TH1-TL1 param de contar. Se programado em outro modo, quando o contador/temporizador 0 estiver no modo 3, os registradores TH1-TL1 continuam funcionando; porém, não é possível desabilitar sua contagem através de TR1, nem requerer execução de tratador de interrupção.

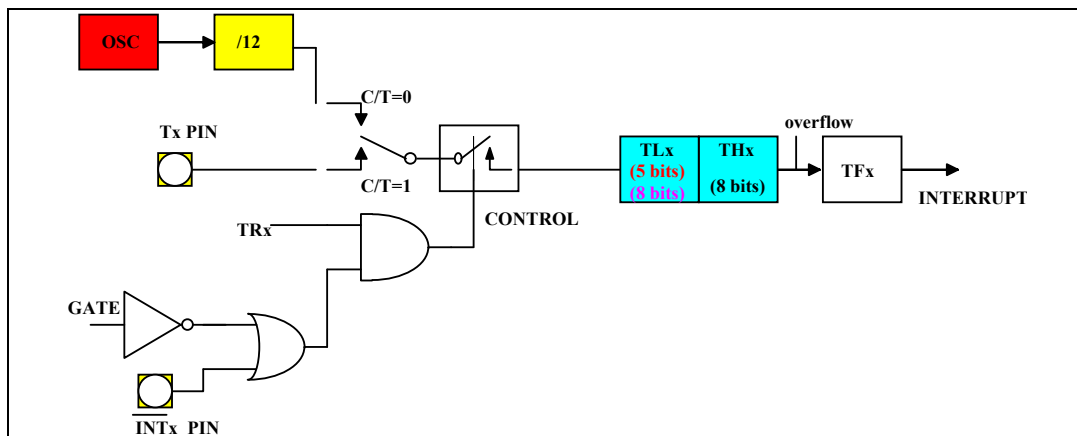


Figura 23 - Esquema de funcionamento dos Contadores/Temporizadores nos modos 0 e 1.

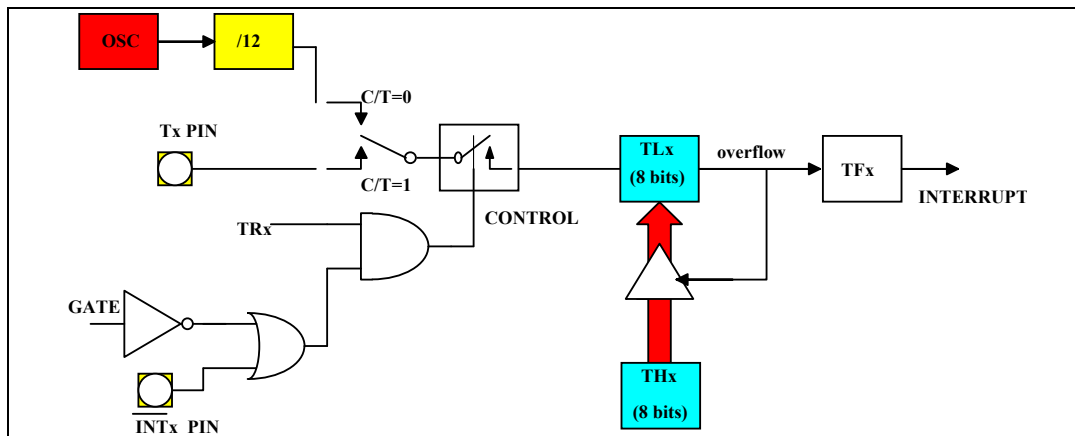


Figura 24 - Esquema de funcionamento dos Contadores/Temporizadores no modo 2.

Nos exemplos apresentados, enfatizou-se a geração de interrupção por parte dos Contadores/Temporizadores. Contudo, em programas mais dedicados pode-se, em vez de gerar interrupção, testar frequentemente (*polling* dos *flags* TFx) se houve encerramento da contagem, ou seja, se TFx='1'. Neste caso, o programador deve limpar o correspondente *flag* TFx para que se possa identificar corretamente o encerramento de uma nova contagem.

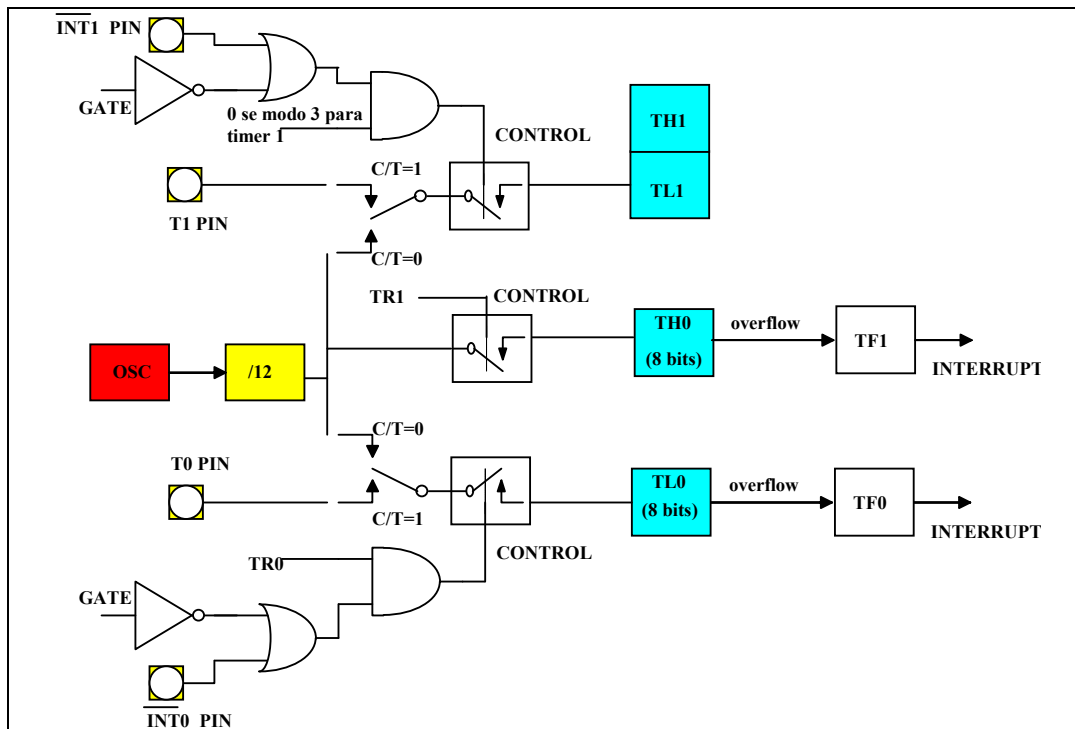


Figura 25 - Esquema de funcionamento dos Contadores/Temporizadores no modo 3.

9. Interface Serial

Em diferentes projetos, torna-se necessário estabelecer comunicação com dispositivos periféricos, por exemplo, transdutor de temperatura. Existem diferentes tipos de transdutores no mercado que já disponibilizam suas medidas em formato binário. Para que a comunicação não demande a utilização de muitos pinos do microprocessador, a transmissão é realizada de forma serial. Existem muitos protocolos para estabelecer a comunicação serial entre microprocessadores e periféricos. O 8051 possui máquina de estado para implementar o protocolo EIA-RS232C. Como exemplo de sua utilidade, foi comentado anteriormente que é possível projetar sistema eletrônico com o 8051 para realizar a contagem do número de pessoas que passam pela catraca de um terminal de ônibus. No entanto, esta informação deve ser transmitida para um computador central para, por exemplo, se obter informações sobre o número de usuários em horários de pico de forma a aprimorar a prestação de serviço. A interface de comunicação serial do 8051 pode estabelecer comunicação com módulo GPRS (telefonia celular) para que este transmita a informação para um servidor.

O 8051 tem uma máquina de estados para transmitir/receber serialmente uma palavra de 8 ou 9 bits, sem a necessidade de participação da UC. A interface serial do 8051 permite a comunicação serial síncrona (Modo 0) e assíncrona (Modos 1, 2 e 3). O modo de funcionamento da interface serial é programado nos dois flip-flops mais significativos do registrador SCON: SM0 e SM1 (Figura 26). A diferença de funcionamento dos diferentes modos é sumarizada na Tabela 11.

A Figura 27 mostra os diagramas de tempo para a transmissão serial síncrona (Modo 0: SM0='0'; SM1='0') e assíncrona. O bit menos significativo é transmitido primeiro.

Nos modos síncrono e assíncrono, o byte a ser transmitido deve ser carregado no registrador SBUF pelo programa sendo que a máquina de estado do 8051 transmite automaticamente o conteúdo de SBUF.

No modo síncrono, o byte contido em SBUF será transmitido através do pino Rx (segunda função do pino P3.0 – Figura 2) enquanto o pino Tx (segunda função do pino P3.1 – Figura 2) supre o sinal de sincronismo (*clock*). Nesse modo, são transmitidos 8 bits de dados. O sinal de sincronismo permite que o dispositivo que recebe o dado identifique o instante em que o bit transmitido deve ser lido, tal que a informação seja recebida corretamente. Por esta razão, o sinal de sincronismo deve ser suprido pelo dispositivo que envia o dado.

No modo assíncrono, o byte contido em SBUF é transmitido serialmente através do pino Tx. Caso se deseje transmitir um nono bit (Modo 2 ou Modo 3), seu valor deve ser carregado no flip-flop TB8 do registrador SCON. É comum que o mesmo seja utilizado para transmitir bit de paridade empregado pelo receptor para verificar que o dado não foi corrompido durante a transmissão. Neste modo, os dispositivos que se comunicam devem ser informados a respeito da taxa de transmissão tal que o bit de dado possa ser corretamente recebido. Uma transição de '1' para '0' (*start bit*) identifica o início da transmissão do dado. Ao final da transmissão de um dado, o nível lógico deve ser colocado em '1' pelo menos, durante o período de transmissão de um bit na taxa de transmissão programada (*stop bit*).

Deve-se observar que, no modo assíncrono, alguns processadores possuem máquinas de estado para identificar, automaticamente, a taxa de comunicação de dados.

Na transmissão assíncrona, um nono bit (conteúdo do flip-flop TB8 – Figura 26) pode ser transmitido (Modo 2 (SM0='1'; SM1='0') ou Modo 3 (SM0='1'; SM1='1')) ou não (Modo 1: SM0='0'; SM1='1'). Como no 8051 a transmissão assíncrona requer a transmissão de 1 *start* bit e de 1 *stop* bit em conjunto com cada byte transmitido, a Tabela 11 os computa na coluna de bits transmitidos, sendo que nos Modos 2 e 3, transmite-se um nono bit que é o valor contido em TB8 (Figura 26).

Após a configuração da interface serial (através dos registradores SCON e PCON e contador/temporizador 1), o programa deve carregar o byte a ser

transmitido no registrador SBUF. Quando a transmissão for concluída, o flip-flop TI é setado pelo hardware; caso a interrupção serial esteja habilitada, o fluxo de execução do programa é desviado (PC é carregado com o valor 0023h) para que o programador identifique que a interface serial pode ser carregada com novo dado a ser transmitido. O flip-flop TI deve ser resetado pelo programador (CLR TI). O hardware não reseta o flip-flop para permitir que o programador identifique se a solicitação de execução do tratador deve-se a buffer de transmissão vazio ou não. Opcionalmente, o programador pode identificar este evento por *polling* (sem habilitar de interrupção), devendo também limpar o flag TI.

Para que o 8051 receba dados, o flip-flop REN do registrador SCON (Figura 26) deve ser colocado em nível lógico alto. Quando um byte for recebido em SBUF, o flip-flop RI é setado pelo hardware; caso a interrupção serial esteja habilitada, o fluxo de execução do programa é desviado (PC é carregado com o valor 0023h) para que o programador identifique que há dado para ser lido na interface serial. O flip-flop RI deve ser resetado pelo programador (CLR RI). O hardware não reseta o flip-flop para permitir que o programador identifique se a solicitação de execução do tratador deve-se a buffer de recepção cheio (RI='1') ou buffer de transmissão vazio (TI='1'). Opcionalmente, o programador pode identificar a ocorrência de recepção de dado (RI='1') por *polling* (sem habilitar de interrupção), devendo também limpar o flag RI.

Observe, portanto, que quando a recepção serial está habilitada (REN='1'), cabe ao código escrito pelo programador identificar qual evento solicitou a execução do código do tratador da interrupção serial: buffer de recepção cheio (RI='1') ou buffer de transmissão vazio (TI='1'). Assim, a ação correspondente a origem da interrupção poderá ser corretamente realizada.

Após a configuração da interface serial (através dos registradores SCON e PCON e contador/temporizador 1), o 8051 recebe byte através do registrador SBUF. O registrador de recepção e transmissão tem o mesmo nome, pois compartilham um mesmo endereço nos SFR. Contudo, ao se escrever um dado em SBUF, este é carregado em um buffer de transmissão; ao se ler um

dado de SBUF, este é lido do buffer de recepção. Sinal de controle interno do 8051 permite que os dois buffers compartilhem o mesmo endereço nos SFR.

No modo síncrono, o byte será recebido através do pino Rx e armazenado no registrador SBUF, enquanto o pino Tx supre o sinal de sincronismo (*clock*). Nesse modo, são recebidos 8 bits de dados. Comparando com a transmissão no modo síncrono, é possível observar que, neste modo, não é possível receber e transmitir dados simultaneamente. No jargão técnico, diz-se que o modo síncrono é *half-duplex* (HD).

No modo assíncrono, o byte recebido serialmente através do pino Rx é armazenado no registrador SBUF; caso se receba um nono bit (Modo 2 ou Modo 3), este bit é armazenado no flip-flop RB8 do registrador SCON (Figura 26). Neste modo, é possível receber e transmitir dados simultaneamente. No jargão técnico, diz-se que o modo assíncrono é *full-duplex* (FD).

A Tabela 11 também mostra as diferentes taxas de transmissão possíveis para cada um dos 4 modos de transmissão serial.

No modo síncrono (Modo 0), a taxa de transmissão é fixa em 1/12 da frequência do cristal (Fclock) conectado nos pinos XTAL1 e XTAL2 (Figura 2).

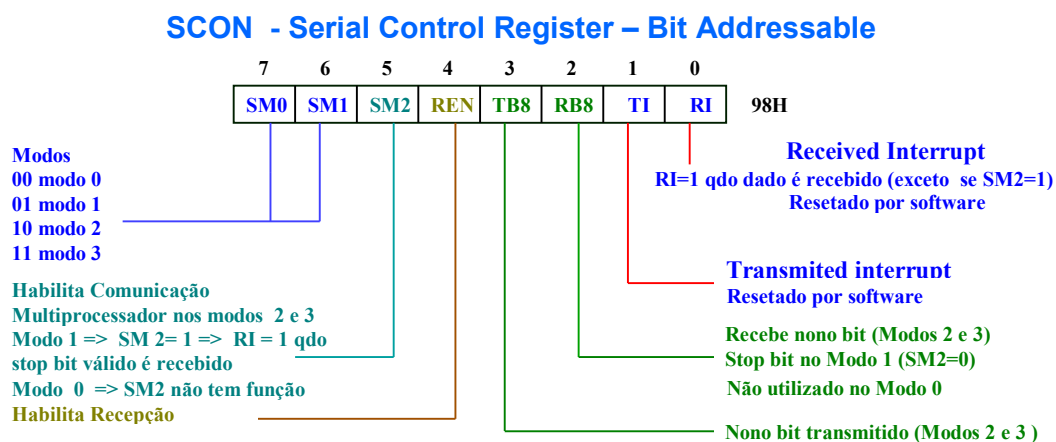


Figura 26 – Registrador *Serial Control* (SCON). Os flip-flops SM0, SM1 e SM2 programam o modo de funcionamento da interface serial. O flip-flop REN habilita a recepção serial. Os flip-flops TB8 e RB8 correspondem ao nono bit transmitido e recebido, respectivamente. Os flip-flops TI e RI em nível lógico '1' solicitam a execução do tratador da interrupção serial. No modo 1, RB8 recebe o stop bit para SM2=0; no modo 0, não tem função.

Tabela 11 – Modos de operação da interface serial definidos por meio da programação dos flip-flops SM0 e SM1 do registrador SCON. O protocolo implementado em cada modo de operação, o número de bits transmitidos e as taxas de transmissão serial possíveis são também apresentados. HD: *Half-Duplex*; FD: *Full-Duplex*.

SM0	SM1	MODO	PROTOCOLO	BITS	TAXA
0	0	0	SÍNCRONO - HD	8	Fclock/12
0	1	1	ASSÍNCRONO - FD	10	Variável
1	0	2	ASSÍNCRONO - FD	11	Fclock/32 ou /64
1	1	3	ASSÍNCRONO - FD	11	Variável

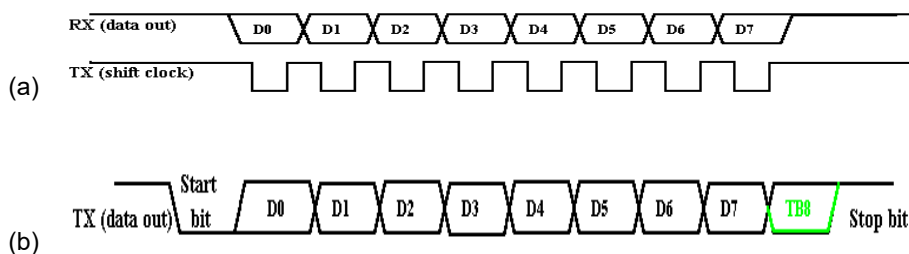


Figura 27 – Diagramas de tempo da transmissão serial no modo síncrono (a) e assíncrono (b). No modo síncrono, o sinal de *clock* transmitido pelo Tx informa o instante em que o receptor deve realizar a leitura do nível lógico presente em seu pino de entrada. Nos modos assíncronos, o receptor deve ter conhecimento da taxa de transmissão; o *start* bit possibilita sincronizar a recepção. O nono bit transmitido nos Modos 2 e 3 (modos assíncronos) corresponde ao valor contido no flip-flop TB8.

No Modo 2, a taxa de transmissão serial de dados (*Baud Rate* – BR) é calculado pela seguinte equação:

$$BR = [(2^{SMOD}) / 64] \times F_{clock}$$

SMOD é um flip-flop do registrador PCON (Seção 10). Assim, para SMOD= '0', BR = Fclock/64; para SMOD= '1', BR = Fclock/32. Isto corresponde aos valores mostrados na Tabela 11.

Nos Modos 1 e 3, o Timer 1 é utilizado para estabelecer a *baud rate* (BR), de acordo com a equação:

$$BR = [(2^{SMOD}) / 32] \times \{ F_{clock} / [12 \times (256 - TH1)] \}$$

Observe que esta equação possui 3 variáveis, SMOD, Fclock e TH1. SMOD pode assumir 2 valores ('0' ou '1'), Fclock pode ser qualquer valor comercial de cristal dentro da faixa de funcionamento do 8051 e TH1 pode assumir 256 diferentes valores. Para simplificar o trabalho do programador, a Tabela 12 apresenta os valores destas 3 variáveis, bem como o modo de programação do Contador/Temporizador 1 para que se obtenha algum dos *baud rates* mais frequentemente utilizados.

Comunicação Multiprocessadores: Os modos 2 e 3 podem ser utilizados para permitir, por exemplo, que um computador se comunique com múltiplos 8051 que compartilham um mesmo barramento de comunicação serial, denominado sistema multiprocessadores.

Como comentado, nestes dois modos, um nono bit recebido é armazenado em RB8. Se o flip-flop SM2 do registrador SCON estiver em nível lógico alto, RI será colocado em '1' (solicitando assim, a execução do tratador serial), apenas se o nível lógico recebido em RB8 for alto ('1').

Suponha, portanto, que os 8051 que compartilham o mesmo barramento de comunicação serial estejam programados no modo 2 ou 3 com SM2='1'.

Tabela 12 – Valores de Fclk, SMOD e Contador/Temporizador 1 para se obter a *baud rate* especificada na primeira coluna.

----- Timer 1 -----					
Baud Rate	Fclk	SMOD	C/T	Mode	Reload
Mode 0 Max: 1 MHz	12 MHz	X	X	X	X
Mode 2 Max: 375K	12 MHz	1	X	X	X
Modes 1, 3: 62.5K	12 MHz	1	0	2	FFH
19.2K	11.059 MHz	1	0	2	FDH
9.6K	11.059 MHz	0	0	2	FDH
4.8K	11.059 MHz	0	0	2	FAH
2.4K	11.059 MHz	0	0	2	F4H
1.2K	11.059 MHz	0	0	2	E8H
137.5K	11.986 MHz	0	0	2	1DH
110K	6 MHz	0	0	2	72H
110K	12 MHz	0	0	1	FEEBH

Quando o computador quiser enviar um bloco de dados para um dos 8051, este envia, primeiramente, um byte de endereço que identifica o 8051 com o qual deseja se comunicar. Este byte de endereço distingue-se dos bytes de dados por meio do nono bit; o mesmo é '1' para endereços e '0' para dados.

Ao enviar um byte de endereço (com seu correspondente TB8='1'), todos os 8051 que compartilham o mesmo barramento de comunicação solicitarão a execução de seus respectivos tratadores de interrupção. Dentro do tratador, cada 8051 irá verificar se o byte de endereço corresponde ao seu; quando for o caso, o tratador do 8051 que se reconhece no endereço, limpa o flip-flop SM2 de seu registrador SCON.

O computador passa então a enviar bytes de dados com o nono bit igual a '0'. Apenas o 8051 previamente endereçado (agora com SM2='0') irá requerer a execução de seu tratador de interrupção da serial de forma a receber os dados enviados pelo computador. Os demais 8051 (SM2='1') não tomam conhecimento do envio destes dados pelo computador.

Quando o computador quiser enviar um bloco de dados para um dos 8051, este envia, primeiramente, um byte de endereço que identifica o 8051 com o qual deseja se comunicar. Este byte de endereço distingue-se dos bytes de dados por meio do nono bit; o mesmo é '1' para endereços e '0' para dados.

Ao enviar um byte de endereço (com seu correspondente TB8='1'), todos os 8051 que compartilham o mesmo barramento de comunicação solicitarão a execução de seus respectivos tratadores de interrupção. Dentro do tratador, cada 8051 irá verificar se o byte de endereço corresponde ao seu; quando for o caso, o tratador do 8051 que se reconhece no endereço, limpa o flip-flop SM2 de seu registrador SCON.

O computador passa então a enviar bytes de dados com o nono bit igual a '0'. Apenas o 8051 previamente endereçado (agora com SM2='0') irá requerer a execução de seu tratador de interrupção da serial de forma a receber os dados

enviados pelo computador. Os demais 8051 (SM2='1') não tomam conhecimento do envio destes dados pelo computador.

Ao término da recepção do bloco de dados, o código do 8051 que se comunicou com o computador deve fazer SM2='1'.

No exemplo apresentado, é possível que o papel do computador seja realizado por um 8051, denominado de mestre.

Observações:

No Modo 0, SM2 não exerce qualquer função.
No Modo 1, se SM2='1', a interrupção não é solicitada enquanto um *stop* bit válido não for recebido.

10. Gerenciamento do consumo de energia do 8051

Como já mencionado, o flip-flop SMOD pertence ao registrador PCON (*Power Control*). Conforme o nome sugere, o registrador PCON está relacionado ao gerenciamento de consumo de energia, sendo bastante relevante em aplicações em que o 8051 é alimentado por baterias. No entanto, em muitos processadores, quando flip-flops de um registrador encontram-se sem uso, estes são utilizados em outros módulos do hardware como é o caso do flip-flop SMOD, utilizado para estabelecer o *baud rate* da interface serial. Discute-se agora, o papel dos demais flip-flops do registrador PCON (Figura 28).

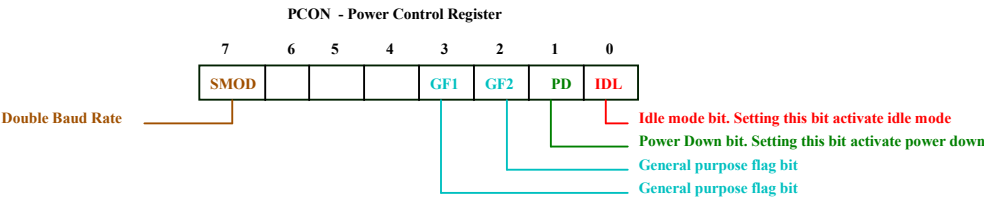


Figura 28 – Registrador *Power Control* (PCON). O flip-flop SMOD é utilizado pela interface serial. Os flip-flops GF1 e GF2 estão disponíveis para o programador sinalizar eventos relacionados ao gerenciamento do consumo de energia. Os flip-flops PD e IDL atuam para reduzir o consumo do 8051.

Ao se fazer $IDL=1$ por meio de instrução do programa, a UCP entra no modo *Idle*. Nesse modo (Figura 29), o sinal de *clock* da UCP é inibido, chegando, contudo, aos contadores/temporizadores, controladores de interrupções externas e interface serial. Assim, os demais registradores não relacionados a estes módulos são ‘congelados’ (como por exemplo, PC, PSW, Acc, SP). Os sinais ALE e PSEN vão para nível lógico alto.

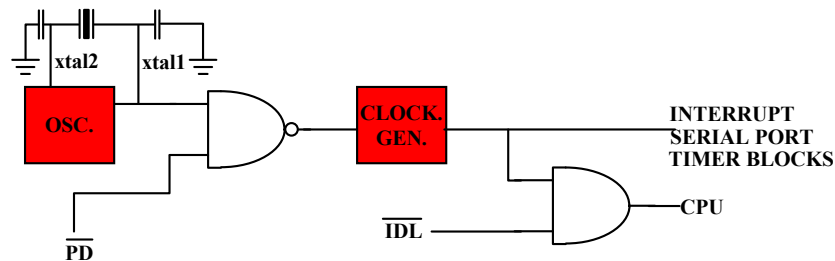


Figura 29 – Diagrama funcional da unidade de controle de consumo. O flip-flop IDL do registrador SCON inibe a chegada do *clock* na UCP. O flip-flop PD do registrador SCON inibe a chegada do *clock* na UCP e demais módulos do 8051.

Existem duas formas de sair do modo *idle*: ativação de uma interrupção habilitada ou por *Reset*; em ambos os casos, IDL é colocado em ‘0’.

Quando uma interrupção habilitada é solicitada, o *clock* da UCP é restabelecido e o tratador de interrupção é executado. Quando da execução da instrução de RETI do tratador, o 8051 volta a executar a instrução posterior àquela que o colocou em *Idle*.

Quando ocorre *Reset*, o *clock* da CPU é restabelecido e o 8051 executa a instrução posterior àquela que o colocou em *Idle*.

Observação:

Após a instrução que colocou o 8051 em *Idle*, o fabricante recomenda: inserir de 3 instruções de NOP (*no operation*) e que a primeira instrução após *Idle* não escreva nas portas ou na memória de dados externa.

Ao se fazer $PD=1$ por meio de instrução do programa, a UCP entra no modo *Power Down*. Nesse modo (Figura 29), o sinal de *clock* é cortado. A RAM interna e SFR são ‘congelados’, bem como os níveis lógicos presentes nos pinos das portas. Os sinais ALE e PSEN vão para nível ‘0’.

Para sair do modo *Power Down*, reseta-se o 8051. O Reset apenas não altera o conteúdo da RAM interna (00 a 7FH).

Observação:

Nesse modo, a alimentação da pastilha pode ser reduzida a 2 Volts. Todavia o valor de Vcc não pode ser reduzido antes que a pastilha esteja em *Power Down*. Ao sair de *Power Down*, Vcc já deve ter sido restabelecido para 5V por tempo suficiente para que o oscilador entre em funcionamento normal.

APÊNDICE A

CONJUNTO DE INSTRUÇÕES DO 8051

A.1 Convenções

IRAM	- memória de dados RAM interna;
XRAM	- memória de dados RAM externa;
Rn	- registrador R0 a R7 do banco selecionado por RS1_RS0 (PSW).
@Ri	- endereçamento indireto a uma posição de memória RAM interna Ri=R0 ou R1
#data	- endereçamento imediato; <i>data</i> é constante de 8 <i>bits</i> incluída na instrução.
#data16	- endereçamento imediato; data16 é constante de 16 <i>bits</i> incluída na instrução.
direct	- endereçamento direto; <i>direct</i> é o endereço de posição de IRAM.
bit	- endereçamento direto de bit na IRAM ou SFR. Exemplo: P3.2 ou 21h.3.
addr16	- endereço com tamanho de 16 bits. Usado nas instruções LJMP e LCALL.
addr11	- endereço com tamanho de 11 bits. A faixa de endereçamento é 1024 bytes anteriores ou 1023 bytes posteriores à posição atual do PC.
rel	- deslocamento de 128 bytes anteriores ou 127 bytes posteriores à posição atual de PC. Usado por SJMP e todos os demais JMPs condicionais.

OBS: Em instruções com registrador específico (*REGISTER SPECIFIC INSTRUCTIONS*), o acumulador é referenciado como A; em instruções com endereçamento direto (*DIRECT ADDRESSING*), o acumulador é referenciado como Acc, sendo substituído pelo seu endereço E0h.

A.2 Instruções para Transferência de Dados

Mnemônico	Descrição	Operação / Exemplo	Flags
MOV A, Rn	Acumulador recebe o conteúdo do registrador	(A) ← (Rn) MOV A, R3	

MOV A, direct	Acumulador recebe um byte da memória RAM interna cujo endereço é <i>direct</i>	(A)← (direct) MOV A,40h	
MOV A, @Ri	Acumulador recebe um dado da memória RAM interna endereçado pelo conteúdo de R0 ou R1	(A)←((Ri)) MOV A,@R1	
MOV A,#data	Acumulador recebe data	(A)←data MOV A,#04h	
MOV Rn,A	Registrador recebe o conteúdo de A	(Rn)← (A) MOV R5,A	
MOV Rn, direct	Registrador recebe um byte da memória RAM interna cujo endereço é <i>direct</i>	(Rn)←(direct) MOV R5,40h	
MOV Rn,#data	Registrador recebe <i>data</i>	(Rn)←data MOV R6,#40h	
MOV direct,A	O conteúdo de A é copiado numa posição da memória RAM interna cujo endereço é <i>direct</i>	(direct)←(A) MOV 45h, A	
MOV direct,Rn	O conteúdo de Rn é copiado numa posição de memória RAM interna cujo endereço é <i>direct</i>	(direct)←(Rn) MOV 55h, R4	
MOV direct1,direct2	O conteúdo da posição de memória RAM interna com endereço <i>direct2</i> é copiado para outra posição da memória interna cujo endereço é <i>direct1</i>	(direct1)←(direct2) MOV 45h,40h	
MOV direct,@Ri	Uma posição de memória RAM interna, cujo endereço é <i>direct</i> , recebe dado endereçado pelo conteúdo de R0 ou R1	(direct)←((Ri)) MOV 45h,@R1	
MOV direct,#data	Conteúdo de memória RAM interna, cujo endereço é <i>direct</i> , recebe <i>data</i> .	(direct)←#30h MOV 45h,40h	
MOV @Ri,A	O conteúdo do acumulador é copiado para uma posição de memória RAM interna cujo endereço é o conteúdo de R0 ou R1	((Ri))←A MOV @R0, A	

MOV @Ri,direct	Conteúdo de memória RAM interna, cujo endereço é <i>direct</i> , é copiado para a posição de memória endereçado pelo conteúdo de R0 ou R1	(direct)←((Ri)) MOV @R1,SBUF	
MOV @Ri,#data	<i>data</i> é copiado para uma posição de memória RAM interna cujo endereço é o conteúdo de R0 ou R1	((Ri))←data MOV @R1,#40h	
MOV DPTR,#data16	Carrega o registrador DPTR com o valor data16	(DPTR)←data16 MOV DPTR,#4010h	
MOVC A,@A+DPTR	O conteúdo do acumulador é somado ao de DPTR formando o endereço de um dado na memória de programa sendo que tal dado é carregado no acumulador	(A)←((A)+(DPTR)) MOV A, @A+DPTR	
MOVC A,@A+PC	O conteúdo do acumulador é somado ao do PC formando o endereço de um dado na memória de programa sendo que tal dado é carregado no acumulador	(A)←((A)+(PC)) MOV A, @A+PC	
MOVX A,@DPTR	O conteúdo de DPTR é o endereço de um dado da RAM externa que é carregado no acumulador	(A)←((DPTR)) MOV A, @DPTR	
MOVX @Ri,A	O conteúdo do acumulador é copiado na posição de memória da RAM externa cujo endereço é dado pelo conteúdo de Ri. A parte mais significativa do endereço deve ser especificada na porta P2 (MOV P2,#ADDR_MSB).	((Ri))←(A) MOV @R1,A	
MOVX @DPTR,A	O conteúdo do acumulador é copiado na posição de memória RAM externa cujo endereço é dado pelo conteúdo de DPTR	((DPTR))←(A) MOV @DPTR,A	
PUSH direct	Coloca na pilha o conteúdo da posição de memória RAM interna cujo endereço é <i>direct</i> .	(SP)←(SP)+1 ((SP))←(direct) PUSH 40h	
POP direct	retira da pilha um dado e armazena na posição de memória RAM interna cujo endereço é <i>direct</i>	(direct)←((SP)) (SP)←(SP)-1 POP 40h	

XCH A,Rn	Troca o conteúdo do acumulador com o conteúdo do registrador	$(A) \longleftrightarrow (Rn)$ XCH A, R7	
XCH A,direct	Troca o conteúdo do acumulador com o conteúdo de uma posição de memória RAM interna endereçada por <i>direct</i>	$(A) \longleftrightarrow (direct)$ XCH A, 50h	
XCH A,@Ri	Troca o conteúdo do acumulador com o conteúdo de uma posição de memória cujo endereço é o conteúdo de Ri.	$(A) \longleftrightarrow ((Ri))$ XCH A, @R1	
XCHD A, @Ri	Troca os 4 <i>bits</i> menos significativos do conteúdo de A com os 4 <i>bits</i> menos significativos do conteúdo da posição de memória cujo endereço é o conteúdo de Ri. Os outros bits ficam inalterados.	$(A)_{0-3} \longleftrightarrow ((Ri))_{0-3}$ XCH A, R7	

A.3. Instruções Aritméticas

Mnemônico	Descrição	Operação / Exemplo	Flags
ADD A, Rn	Ao conteúdo do A, soma-se o conteúdo do registrador.	$(A) \leftarrow (A) + (Rn)$ ADD A, R7	C, OV, AC
ADD A, direct	Ao conteúdo de A, soma-se o conteúdo de uma posição de memória RAM interna cujo endereço é <i>direct</i>	$(A) \leftarrow (A) + (direct)$ ADD A, 40h	C, OV, AC
ADD A,@Ri	Ao conteúdo de A, soma-se o conteúdo de uma posição de memória RAM interna endereçada pelo conteúdo de Ri	$(A) \leftarrow (A) + ((Ri))$ ADD A, @R1	C, OV, AC
ADD A,#data	Ao conteúdo de A, soma-se o valor <i>data</i>	$(A) \leftarrow (A) + data$ ADD A, #30h	C, OV, AC
ADDC A,Rn	Ao conteúdo de A, somam-se o conteúdo do registrador e o conteúdo do <i>flag</i> de <i>carry</i> .	$(A) \leftarrow (A) + (Rn) + (C)$ ADDC A, R7	C, OV, AC

ADDC A,direct	Ao conteúdo de A, somam-se o conteúdo de uma posição de memória RAM interna cujo endereço é direct, e o conteúdo do <i>flag</i> de <i>carry</i>	$(A) \leftarrow (A) + (\text{direct}) + (C)$ ADDC A, 40h	C, OV, AC
ADDC A,@Ri	Ao conteúdo de A, somam-se o conteúdo de uma posição de memória RAM interna cujo endereço é o conteúdo de Ri, e o conteúdo do <i>flag</i> de <i>carry</i>	$(A) \leftarrow (A) + ((Ri)) + (C)$ ADDC A, @R1	C, OV, AC
ADDC A,#data	Ao conteúdo de A, somam-se o valor data e o conteúdo do <i>flag</i> de <i>carry</i>	$(A) \leftarrow (A) + \text{data}$ ADD A, #30h	C, OV, AC
SUBB A, Rn	Do conteúdo de A, subtraem-se o conteúdo do registrador e do <i>flag</i> de <i>carry</i>	$(A) \leftarrow (A) - (Rn) - (C)$ SUBB A, R7	C, OV, AC
SUBB A,direct	Do conteúdo de A, subtraem-se o conteúdo de uma posição de memória RAM interna cujo endereço é <i>direct</i> e o <i>flag</i> de <i>carry</i>	$(A) \leftarrow (A) - (\text{direct}) - (C)$ SUBB A, 40h	C, OV, AC
SUBB A,@Ri	Do conteúdo de A, subtraem-se o conteúdo de uma posição de memória RAM interna cujo endereço é o conteúdo de Ri e o <i>flag</i> de <i>carry</i>	$(A) \leftarrow (A) - ((Ri)) - (C)$ SUBB A, @R1	C, OV, AC
SUBB A,#data	Do conteúdo de A, subtraem-se o valor data e o <i>carry flag</i> .	$(A) \leftarrow (A) - \text{data} - (C)$ SUBB A, #30h	C, OV, AC
INC A	Ao conteúdo de A, soma-se 1.	$(A) \leftarrow (A) + 1$ INC A	
INC Rn	Ao conteúdo do registrador, soma-se 1.	$(Rn) \leftarrow (Rn) + 1$ INC R3	
INC direct	Ao conteúdo da posição de memória RAM interna cujo endereço é direct, soma-se 1.	$(\text{direct}) \leftarrow (\text{direct}) + 1$ INC 40h	
INC @Ri	Ao conteúdo da posição de memória RAM interna, cujo endereço é o conteúdo de Ri, soma-se 1.	$((Ri)) \leftarrow ((Ri)) + 1$ INC @R1	
DEC A	Do conteúdo do acumulador, subtrai-se 1.	$(A) \leftarrow (A) - 1$ DEC A	

DEC Rn	Do conteúdo do registrador , subtrai-se 1.	$(Rn) \leftarrow (Rn) - 1$ DEC R3	
DEC direct	Do conteúdo da posição de memória RAM interna cujo endereço é <i>direct</i> , subtrai-se 1.	$(direct) \leftarrow (direct) - 1$ DEC 40h	
DEC @Ri	Do conteúdo da posição de memória RAM interna, cujo endereço é o conteúdo de Ri, subtrai-se 1.	$((Ri)) \leftarrow ((Ri)) - 1$ DEC @R1	
INC DPTR	Ao conteúdo do registrador DPTR, soma-se 1.	$(DPTR) \leftarrow (DPTR) + 1$ INC DPTR	
MUL AB	O conteúdo do acumulador e do registrador B são multiplicados sem sinal. Da palavra de 16 <i>bits</i> resultante, os 8 <i>bits</i> LSB são armazenados em A ; os 8 <i>bits</i> MSB são armazenados em B.	$(A) \leftarrow [(A) * (B)]_{0-7}$ $(B) \leftarrow [(A) * (B)]_{8-15}$ MUL A, B	C=0, OV
DIV AB	O conteúdo do acumulador é dividido (sem sinal) pelo conteúdo do registrador B. Em A, é armazenado a parte inteira do quociente da divisão; em B, o resto	$(A) \leftarrow \text{int}[(A)/(B)]$ $(B) \leftarrow \text{resto}[(A)/(B)]$ DIV A, B	C=0, OV
DA A	O conteúdo de A é convertido para um número decimal de dois dígitos de quatro bits cada que são armazenados em A	If $[(A)_{3-0} > 9 \text{ or } (AC)=1]$ then $(A)_{3-0} \leftarrow (A)_{3-0} + 6$ If $[(A)_{7-4} > 9 \text{ or } (C)=1]$ then $(A)_{7-4} \leftarrow (A)_{7-4} + 6$	C

A.4. Instruções para Operações Lógicas

Mnemonic	Descrição	Operação / Exemplo	Flags
ANL A,Rn	Lógica E entre o conteúdo do acumulador e o conteúdo do registrador	$(A) \leftarrow (A) \wedge (Rn)$ ANL A, R7	
ANL A,direct	Lógica E entre o conteúdo do acumulador e o conteúdo de uma posição de memória RAM interna cujo endereço é direct	$(A) \leftarrow (A) \wedge (direct)$ ANL A, 40h	

ANL A,@Ri	Lógica E entre o conteúdo do acumulador e uma posição de memória RAM interna cujo endereço é o conteúdo de Ri	$(A) \leftarrow (A) \wedge ((Ri))$ ANL A, @R1	
ANL direct,A	Lógica E entre o conteúdo do acumulador e o conteúdo de uma posição de memória RAM interna cujo endereço é <i>direct</i>	$(direct) \leftarrow (direct) \wedge (A)$ ANL 60h, A	
ANL direct,#data	Lógica E entre <i>data</i> e o conteúdo de uma posição de memória RAM interna cujo endereço é <i>direct</i>	$(direct) \leftarrow (direct) \wedge data$ ANL 23h, #20h	
OR A, Rn	Lógica OU entre o conteúdo do acumulador e o conteúdo do registrador.	$(A) \leftarrow (A) \vee (Rn)$ OR A, R7	
ORL A,direct	Lógica OU entre o conteúdo do acumulador e o conteúdo de uma posição de memória RAM interna cujo endereço é <i>direct</i>	$(A) \leftarrow (A) \vee (direct)$ OR A, 40h	
ORL A,@Ri	Lógica OU entre o conteúdo do acumulador e uma posição de memória RAM interna cujo endereço é o conteúdo de Ri	$(A) \leftarrow (A) \vee ((Ri))$ OR A, @R1	
ORL direct,A	Lógica OU entre o conteúdo de uma posição de memória RAM interna cujo endereço é <i>direct</i> e o conteúdo do acumulador	$(direct) \leftarrow (direct) \vee (A)$ OR 30h, A	
ORL direct,#data	Lógica OU Exclusivo entre o conteúdo de uma posição de memória RAM interna cujo endereço é <i>direct</i> , e <i>data</i>	$(direct) \leftarrow (direct) \vee data$ OR 32h, #20h	
XRL A, Rn	Lógica OU Exclusivo entre o conteúdo do acumulador e o conteúdo do registrador	$(A) \leftarrow (A) \oplus (Rn)$ XRL A, R7	
XRL A,direct	Lógica OU Exclusivo entre o conteúdo do acumulador e o conteúdo de uma posição de memória RAM interna cujo endereço é <i>direct</i>	$(A) \leftarrow (A) \oplus (direct)$ XRL A, 40h	
XRL A,@Ri	Lógica OU Exclusivo entre o conteúdo do acumulador e uma posição de memória RAM interna cujo endereço é o conteúdo de Ri	$(A) \leftarrow (A) \oplus ((Ri))$ XRL A, @R1	

XRL direct,A	Lógica OU Exclusivo entre o conteúdo do acumulador e o conteúdo de uma posição de memória RAM interna indicada por <i>direct</i>	$(\text{direct}) \leftarrow (\text{direct}) \oplus (A)$ XRL 90h, A	
XRL direct,#data	Lógica OU Exclusivo entre <i>data</i> e o conteúdo de uma posição de memória RAM interna cujo endereço é <i>direct</i> .	$(\text{direct}) \leftarrow (\text{direct}) \oplus \text{data}$ XRL 73h, #20h	
CLR A	Zera o conteúdo do acumulador	$(A) \leftarrow 0$ CLR A	
CPL A	complementa o conteúdo do acumulador	$(A) \leftarrow \neg(A)$	
RL A	rotaciona o conteúdo do acumulador para a esquerda. O bit 7 é carregado no bit 0.	$(A_{n+1}) \leftarrow (A_n)$ $(A_0) \leftarrow (A_7)$ RL A	
RLC A	rotaciona o conteúdo do acumulador para a esquerda através do <i>carry</i>	$(A_{n+1}) \leftarrow (A_n)$ $(A_0) \leftarrow (C)$ $(C) \leftarrow (A_7)$ RLC A	C
RR A	rotaciona o conteúdo do acumulador para a direita	$(A_n) \leftarrow (A_{n+1})$ $(A_7) \leftarrow (A_0)$ RR A	
RRC A	rotaciona o conteúdo do acumulador para a direita através do <i>carry</i>	$(A_n) \leftarrow (A_{n+1})$ $(A_7) \leftarrow (C)$ $(C) \leftarrow (A_0)$ RRC A	C
SWAP A	Troca os 4 bits menos significativos do conteúdo de A com o 4 bits mais significativos	$(A_{3-0}) \longleftrightarrow (A_{7-4})$ SWAP A	

A.5. Instruções para Manipulação de *Bits*

Por possuir estas instruções que controlam flip-flops da iRAM e SFR (incluindo P0, P1, P2 e P3), o 8051 é classificado como processador booleano.

Mnemonico	Descrição	Operação / Exemplo	Flags

CLR C	Zera o <i>flag carry</i> em PSW.	$(C) \leftarrow 0$ CLR C	C=0
CLR bit	Zera o bit da posição de memória RAM interna indicada. Atua na iRAM de 20H a 2FH e em alguns SFR.	$(bit) \leftarrow 0$ CLR 29.5 CLR P3.2	
SETB C	Coloca em 1, o <i>flag carry</i> de PSW.	$(C) \leftarrow 1$ SETB C	
SETB bit	Coloca em 1, o bit da posição de memória RAM interna indicada. Atua na iRAM de 20H a 2FH e em alguns SFR.	$(bit) \leftarrow 1$ SETB 29.5 SETB P3.2	
CPL C	Complementa o <i>flag carry</i> em PSW.	$(C) \leftarrow (C)/$ CPL C	C
CPL bit	Complementa o bit em uma posição de memória RAM interna com <i>bits</i> endereçáveis.	$(bit) \leftarrow (bit)/$ CPL 29.5 CPL P3.2	
ANL C,bit	Lógica E entre o <i>flag carry</i> e o bit em uma posição de memória RAM interna com <i>bits</i> endereçáveis.	$(C) \leftarrow (C) \wedge (bit)$ ANL C, 29.5	C
ANL C, /bit	Lógica E entre o <i>flag carry</i> e o complemento do bit em uma posição de memória RAM interna com <i>bits</i> endereçáveis.	$(C) \leftarrow (C) \wedge [(bit)/]$ ANL C, /29.5	C
ORL C,bit	Lógica OU entre o <i>flag carry</i> e o bit em uma posição de memória RAM interna com <i>bits</i> endereçáveis.	$(C) \leftarrow (C) \vee (bit)$ ORL C, 29.5	C
ORL C, /bit	Lógica OU entre o <i>flag carry</i> e o complemento do bit em uma posição de memória RAM interna com <i>bits</i> endereçáveis.	$(C) \leftarrow (C) \vee [(bit)/]$ ORL C, /29.5	C
MOV C,bit	Acerta o <i>flag carry</i> com o valor do bit de uma posição de memória RAM interna com <i>bits</i> endereçáveis.	$(C) \leftarrow (bit)$ MOV C, 29.5 MOV C, P3.5	C

MOV bit,C	Acerta o bit de uma posição de memória RAM interna com <i>bits</i> endereçáveis com o <i>flag carry</i>	(bit) \leftarrow (C) MOV 29.5,C MOV P3.5,C	
JC rel	Desvia se o <i>flag carry</i> estiver em 1	If (C)=1 then (PC) \leftarrow (PC)+rel	
JNC rel	Desvia se o <i>flag carry</i> estiver em 0	If (C)=0 then (PC) \leftarrow (PC)+rel	
JB bit,rel	Desvia se o bit estiver em 1	If (bit)=1 then (PC) \leftarrow (PC)+rel	
JNB bit,rel	Desvia se o bit estiver em 0	If (bit)=0 then (PC) \leftarrow (PC)+rel	
JBC bit,rel	Desvia se o bit estiver em 1 e automaticamente zera o bit	If (bit)=1 then (PC) \leftarrow (PC)+rel (bit) \leftarrow 0	

A.6. Instruções para desvio do fluxo de execução do programa

Mnemonico	Descrição	Operação / Exemplo	Flags
ACALL addr11	Desvia para uma subrotina. Essa subrotina deve estar no máximo a 1KiB de distância do ponto de chamada	(PC) \leftarrow (PC)+2 (SP) \leftarrow (SP)+1 ((SP)) \leftarrow (PC ₇₋₀) (SP) \leftarrow (SP)+1 ((SP)) \leftarrow (PC ₁₅₋₈) (PC ₁₀₋₀) \leftarrow addr11	
LCALL addr16	Desvia para uma subrotina.	(PC ₁₅₋₀) \leftarrow addr16	
RET	Retorna de uma subrotina.	(PC ₁₅₋₈) \leftarrow ((SP)) (SP) \leftarrow (SP)-1 (PC ₁₀₋₀) \leftarrow ((SP)) (SP) \leftarrow (SP)-1	

RETI	Retorna do tratador de interrupção, habilitando atendimento de outras interrupções com mesmo nível de prioridade.	$(PC_{15-8}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$ $(PC_{10-0}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$	
AJMP addr11	Desvia fluxo de execução do programa para a instrução endereçada. A instrução endereçada deve estar, no máximo, em uma região de 1KiB desta.	$(PC) \leftarrow (PC) + 2$ $(PC_{10-0}) \leftarrow \text{addr11}$	
LJMP addr16	Desvia fluxo de execução do programa para a instrução endereçada.	$(PC_{15-0}) \leftarrow \text{addr16}$	
SJMP rel	Desvia fluxo de execução do programa para a instrução endereçada. A instrução endereçada deve estar, no máximo, em uma região de 1KiB desta.	$(PC) \leftarrow (PC) + 2$ $(PC) \leftarrow (PC) + \text{rel}$	
JMP @A + DPTR	Desvia fluxo de execução do programa para a instrução endereçada pela soma do conteúdo de DPTR com o conteúdo A.	$(PC) \leftarrow (PC) + (A)$	
JZ rel	Desvia fluxo de execução do programa para a instrução endereçada se o acumulador for zero.	if (A)=0 then $(PC) \leftarrow (PC) + \text{rel}$	
JNZ rel	Desvia fluxo de execução do programa para a instrução endereçada se o acumulador for diferente de zero.	if (A)<>0 then $(PC) \leftarrow (PC) + \text{rel}$	
CJNE A,direct,rel	Desvia fluxo de execução do programa para a instrução endereçada se o conteúdo de A e <i>direct</i> forem diferentes.	if (A)<>(direct) then $(PC) \leftarrow (PC) + \text{rel}$	C
CJNE A,#data,rel	Desvia fluxo de execução do programa para a instrução endereçada se o conteúdo de A não for igual ao dado.	if (A)<>data then $(PC) \leftarrow (PC) + \text{rel}$	C

CJNE @Ri,#data,rel	Desvia fluxo de execução do programa para a instrução endereçada se o conteúdo de Rn for diferente do conteúdo da posição de memória cujo endereço é apontado por R1 ou R0.	if (Rn)<>(Ri) then (PC)←(PC)+rel	C
CJNE Rn,#data,rel	Desvia fluxo de execução do programa para a instrução endereçada se o conteúdo de Rn e <i>data</i> forem diferentes.	if (Rn)<>data then (PC)←(PC)+rel	C
DJNZ Rn,rel	Decrementa o registrador e desvia fluxo de execução do programa para a instrução endereçada se o acumulador for diferente de zero.	(Rn)←(Rn)-1 if (Rn)<>0 then (PC)←(PC)+rel	
DJNZ direct,rel	Decrementa o conteúdo da posição de memória dada por <i>direct</i> . Se o resultado não for nulo ocorre um desvio; caso contrário; a instrução seguinte é executada.	(direct)←(direct)-1 if (direct)<>0 then (PC)←(PC)+rel	
NOP	Não executa qualquer atividade.	(PC)←(PC)+1	

APÊNDICE B

LINGUAGEM ASSEMBLY

O programa-fonte em Assembly consiste-se de uma série de linhas de texto podendo possuir até quatro campos, separados por caractere ASCII de espaço ou tabulação:

<Rótulo> <Instrução> e/ou <Diretiva de compilação> <;Comentário>

Como por exemplo:

LOOP: MOV A,R2 ; transfere conteúdo de R2 para acumulador

Comentários devem ser precedidos de ponto e vírgula, sendo opcionais.

Os rótulos (ou *labels*) são símbolos que representam endereços. Os endereços referenciados ao longo programa por meio dos rótulos são substituídos pelos respectivos endereços (ou *off-sets*) durante a compilação. Geralmente, os compiladores requerem que os rótulos sejam colocados na primeira coluna do arquivo-fonte e que o primeiro caractere do rótulo deva ser uma letra.

No programa fonte, os rótulos, menemônicos e diretivas de compilação podem ser escritos em letra maiúscula ou minúscula; porém, não devem ser escritos parcialmente em maiúsculas ou minúsculas.

Diretivas de compilação

As diretivas de compilação do Assembler não geram código de instruções a serem executadas pelo processador, mas especificam, por exemplo, onde o compilador deve alocar dados, seções do código e outros. A seguir, comenta-se as diretivas mais utilizadas do Assembler.

ORG (Origin)

Especifica o endereço a partir do qual os dados ou instruções do programa devem ser armazenados na memória de programa. Caso a mesma esteja ausente no programa, o programa e dados são alocados de forma consecutiva a partir do endereço 0H.

Formato:

ORG <Endereço da memória> <;comentário>

EQU (Equate)

Associa um nome simbólico a uma constante numérica. O compilador substitui o nome simbólico pelo valor correspondente. Similar ao uso da diretiva `#define` na linguagem C. O uso do nome simbólicos criados pela diretiva ao longo do programa auxilia a melhor compreensão dos mesmos (pois constantes podem ser associados a sua funcionalidade no código) e também, facilita o teste de alterações desta constante; isto é, para alterar o valor de uma constante utilizada diversas vezes no programa, basta modificar o valor associado à diretiva EQU.

Formato:

<rótulo> EQU <constante> <;comentário>

DB (Define Byte)

Instrui compilador para armazenar constantes e tabelas na memória. Várias constantes (até 256 bytes) podem ser criadas com uma única diretiva DB; para tal, as constantes devem se encontrar separadas por vírgulas ou declaradas como *strings* (conjuntos de caracteres ASCII entre aspas).

Formato:

<Rótulo> DB <valor ou *string*>,< valor ou *string* > <comentário>

END

Informa fim do programa fonte ao compilador. Linhas de código abaixo da mesma são ignoradas.

Formato:

END <;comentário>

Exemplo de emprego das diretivas de compilação:

```
reset      EQU  0H      ; endereço inicial do programa
tratador   EQU  03H     ; endereço para alocar trecho de código
flag       EQU  05H     ; endereço de memória referenciado no programa
dados      EQU  30H     ; endereço para alocar constantes no código

                ORG      reset      ; programa se inicia aqui
                JMP      inicio

                ORG      tratador ; colocar código a partir de 03H
                MOV      flag,#1
                RETI

inicio:      MOV      DPTR,#ctes2 ; carrega endereço em dptr
            .....
            (código)
            .....
final:      JMP      final

                ORG      dados ; dados alocados a partir de 30H

ctes1:  DB  45H,0ABH,0E2H
ctes2:  DB  "EEL",70H,30H ; na memória => 45, 45, 4B, 70,30

                END      ;texto abaixo (se houver) é ignorado pelo compilador
```

Referências

Atmel (2004), Atmel 8051 Microcontrollers Hardware Manual.

Morh H. (2005), A família de microcontroladores 8051, apostila.

Ziller R. M. (2000) Microprocessadores conceitos importantes, segunda edição, edição do autor, Florianópolis, SC.

Wisbeck J. O. (1998), Microcontroladores - O 80C51BH da Família MCS51, apostila.