

1	ARQUITETURA EXTERNA E INTERNA DOS MICROPROCESSADORES	2
1.1	Arquitetura de um microcomputador.....	3
1.2	Execução das instruções no microcomputador: Fases.....	8
1.3	Arquitetura do microprocessador/microcontrolador 8051	11
1.4	Organização da memória	15
1.5	Conexão de chips externos de memória	20
2	PROGRAMAÇÃO DOS MICROPROCESSADORES.....	29
2.1	Modos de endereçamento dos dados	29
2.2	Repertório de instruções	35
2.3	Ferramentas para o trabalho com Microprocessadores.....	38
2.4	Programação em linguagem <i>assembler</i>	40
2.5	Exemplos de programas em linguagem <i>assembler</i>	52
2.6	Programação em linguagem C usando o compilador SDCC	70
3	ATENÇÃO A DISPOSITIVOS DE ENTRADA-SAÍDA	85
3.1	Portas paralelas no 8051.....	85
3.2	Atenção a teclados.....	89
3.3	Atenção a displays.....	99
3.4	Atenção a displays LCD.....	106
3.5	Temporizadores y contadores.....	124
3.6	Temporizadores no microprocessador 8051	126
3.7	Interrupções.....	142
3.8	Interrupções na família 8051	144
3.9	Porta serial	157
3.10	A Porta Serial do 8051.....	164
4	INTRODUÇÃO À SIMULAÇÃO COM PROTEUS (ANEXOS)	176
4.1	Passeio por o PROTEUS ISIS.....	177
5	LABORATORIOS	190
5.1	LABORATORIO 1: KIT DE DESENVOLVIMENTO E MANUSEIO DE LEDS.....	190
5.2	LABORATORIO 2: ATENÇÃO A DISPLAY DE SETE SEGMENTOS.....	197
5.3	LABORATORIO 3: ATENÇÃO A DISPLAY LCD.....	202
5.4	LABORATORIO 4: USO DAS INTERRUPÇÕES EXTERNAS.....	206
5.5	LABORATORIO 5: USO DOS TEMPORIZADORES NO 8051.....	210

1 ARQUITETURA EXTERNA E INTERNA DOS MICROPROCESSADORES

Os microcontroladores são essencialmente, microcomputadores num chip. Neles se combinam os recursos fundamentais de um microcomputador, ou seja, o microprocessador, a memória RAM e ROM / EPROM / EEPROM e as facilidades de entrada saída, como são a gestão de interrupções, portas paralelas, seriais e temporizadores, tudo em um único circuito integrado. Podem incluir também entradas e saídas analógicas associadas a conversores A/D e D/A, circuito do Watchdog, etc.

Historicamente, o desenvolvimento dos microprocessadores pode resumir-se em duas grandes e importantes vertentes mais ou menos paralelas:

1. Os microprocessadores propriamente ditos, de 8, 16 e 32 bits, nos que se faz ênfase na potência de cálculo, a quantidade de memória endereçável, etc. Estes microprocessadores estão orientados fundamentalmente ao mercado dos computadores pessoais e as estações de trabalho.
2. Os microcontroladores ou microcomputadores num chip, orientados ao desenvolvimento de aplicações industriais, de comunicações, domésticas e de todo tipo. trata-se de ter no menor espaço possível e a um custo razoável, os elementos essenciais para desenvolver este tipo de aplicação. Faz-se ênfase nas facilidades de entrada saída do dispositivo (manejo de bits individuais, interrupções, sinais analógicos, etc.) Nesta disciplina estudaremos a família de microprocessadores MCS51/52 e especificamente o subconjunto da Família 8051.

A família MCS 51/52 da Intel tem vários membros. Os principais são:

O 8051 com memória ROM interna para armazenar o programa.

O 8031: um 8051 sem memória ROM interna.

O 8751: um 8051 com memória EPROM interna.

O 8052: um 8051 “melhorado”.

O 8032: um 8052 sem memória ROM interna.

O 8752: um 8052 com memória EPROM interna.

De cada um destes membros há diferentes versões (para baixo consumo, etc.). Na atualidade existem várias empresas (SIEMENS, Philips, SIGNETICS, etc) que estão autorizadas a fabricar versões a partir do núcleo básico do 8051 (CPU), por isso as possibilidades destas versões são muito variadas e portanto o usuário pode encontrar um microcontrolador que se adapte o mais possível a sua aplicação.

1.1 Arquitetura de um microcomputador

Um microcomputador é um sistema que inclui um microprocessador ou Unidade Central de Processamento (CPU, *Central Processor Unit*), memória (de programa e de dados) e diversos dispositivos de Entrada-Saída (E/S). A implementação prática de um microcomputador conter vários chips e dispositivos (teclado, monitor impressora, discos, disquetes, exploratório) como ocorre em um computador pessoal (PC), ou pode estar integrada em um simple chip de silício, como é o caso de um microcontrolador.

A arquitetura geral de um microcomputador se mostra na figura 1.1, e se chama de “programa armazenado” devido a seu funcionamento é governado por um conjunto de instruções (programa de aplicação) que reside na memória de programa, e que é procurado, interpretado e executado pela CPU.

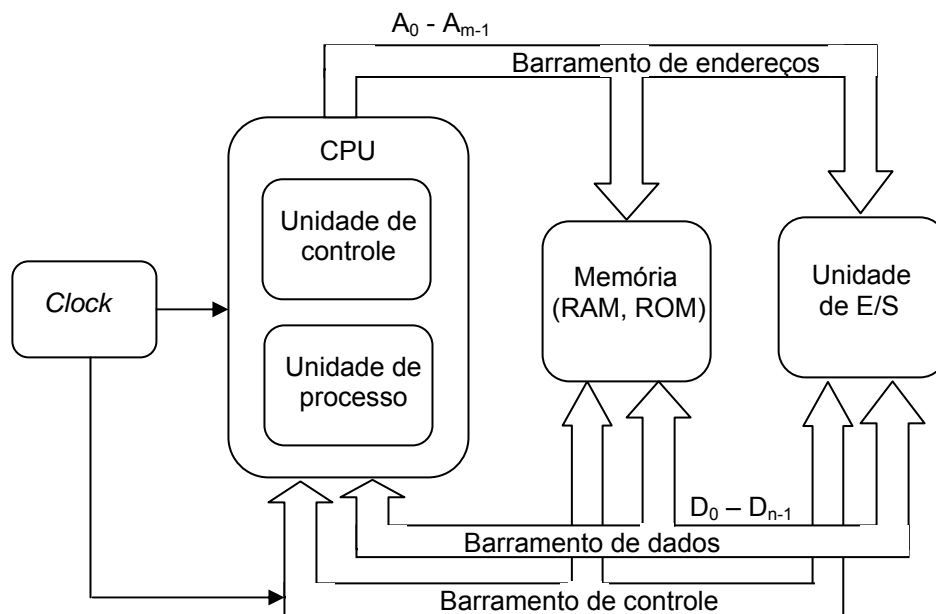


Figura 1.1 Diagrama de blocos general de um microcomputador.

Todos os módulos estão interconectados entre si mediante linhas elétricas denominadas barramentos, por isso às vezes, lhe conhece como Arquitetura de barramentos. Os barramentos podem ser de endereços (se transportarem direções de memória ou do E/S), de dados (se transportarem instruções ou dados) e de controle (se transportarem sinais de controle). As linhas do barramentos de endereços são unidirecionais, ou seja, transportam os endereços desde a CPU até o módulo. Cada módulo abrange uma fila de endereços e pode estar composto por

vários chips. As linhas do barramento de dados são bidirecionais: a informação vai para a CPU quando se realiza uma operação de leitura ou para o módulo, quando se realiza uma operação de escrita.

Cada sinal individual do barramento de controle é unidirecional, embora de forma global, existem sinais que entram na CPU ou saem desta. Por exemplo, o sinal de controle de reset entra na CPU, e serve para levar a mesma a um estado inicial predeterminado. O sinal de controle de leitura saliente da CPU indicará ao módulo em questão, que a CPU deseja receber (ler) os dados existentes na localização de memória (ou do E/S) cujo endereço é especificado em suas linhas de endereços.

O número de linhas de cada barramento pode ser variável. A CPU que se estudará nesta disciplina (Família MCS51) tem 16 linhas de endereços e 8 linhas de dados, por isso pode processar instruções e transferir dados de 8 bits, sendo classificada como uma CPU de 8 bits.

Em cada instante de tempo só pode estar ativa a CPU e um só módulo (chips de memória ou do E/S). A transferência direta de dados entre dois módulos do sistema, sem intervenção da CPU, consegue-se usando um mecanismo chamado Acesso Direto a Memória ou DMA (do inglês *Direct Memory Access*) que não será tratado nesta disciplina.

Cada sinal individual do barramento de controle é unidirecional, embora de forma global, existem sinais que entram na CPU ou saem desta. Por exemplo, o sinal de controle de RESET entra na CPU, e serve para levar a mesma a um estado inicial predeterminado. O sinal de controle de leitura saliente da CPU indicará ao módulo em questão, que a CPU deseja receber (ler) os dados existentes na localização de memória (ou do E/S) cujo endereço é especificado em suas linhas de endereços.

A seguir descreveremos mais detalhado as características dos três tipos de barramentos de um microprocessador.

1.1.1 Barramento de endereços: $A_0 - A_{m-1}$

É o empregado pela CPU para selecionar a endereço de memória ou o dispositivo do E/S com o qual vai intercambiar informação. É portanto unidirecional e seu tamanho, ou número de condutores que o constituem, determina a capacidade de endereçamento da CPU, que é o máximo número de posições de memória e

dispositivos E/S aos que a CPU pode acessar. Para m linhas a capacidade de endereçamento será: 2^m .

Exemplos:

- Microprocessadores 6502, Z80, 8051: $m=16$ Capacidade de endereçamento = $2^{16} = 65\,536$ posições (64 k)
- 8086: $m=20$ Capacidade de endereços = $2^{20} = 1.048.576$ posições (1 Mega localizações)

1.1.2 Barramentos de dados: $D_0 - D_{n-1}$

O barramentos de dados é o conjunto de condutores através do qual o microprocessador troca informação com a unidade de memória ou E/S selecionada mediante o barramentos de endereços. Características:

- Bidirecional: a informação pode viajar nos dois sentidos.
- Número de linhas (n): representa a quantidade de bits que se podem transmitir simultaneamente. Também é chamado Comprimento da Palavra do microprocessador.
- Tri-estado: as linhas do barramentos de dados devem ser tri-estado. As linhas tri-estado são aquelas que são capazes de ter três estados:
 - Estado alto (*High*, H).
 - Estado baixo (*Low*, L).
 - Estado de alta impedância (*High Impedance*, Hz).

Na figura 1.2 se mostra a transferência de dados desde um dispositivo de E/S até o microprocessador (chamada como operação de leitura de E/S), onde somente o dispositivo E/S1 envia informação (nível H neste caso) para o microprocessador enquanto o outro dispositivo E/S fica em estado de alta impedância e por tanto não afeta à transferência. A Unidade de Controle da CPU é quem define o que elemento vai se trocar informação com o microprocessador enviando seu endereço por o barramento de endereços. A seleção final de um dispositivo é feita no próprio chip, através de um terminal físico chamado como *Chip Select* ou Seleção de Chip, que quando é habilitado o dispositivo está pronto para trocar informação (enviar ou receber) e quando não, então cai no estado de alta impedância.

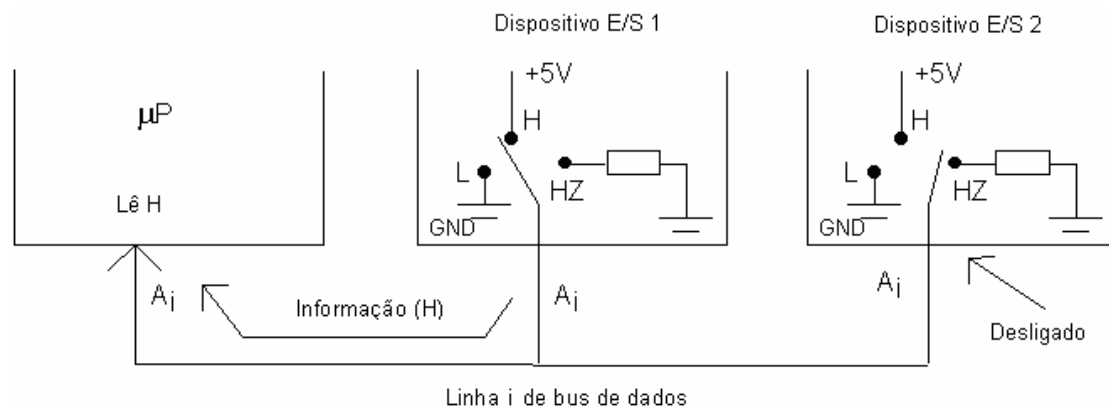


Figura 1.2 Exemplo de uso de barramento de dados

1.1.3 Barramentos de controle.

Está formado por um conjunto de linhas pelas que circulam os sinais auxiliares de governo e sincronização do sistema. As linhas existentes dependem do fabricante do μP e das funções que deseje lhe implementar. Alguns sinais típicos em todos os sistemas são:

- Sinal de relógio de sincronização
- Sinal do RESET ou inicialização
- Sinal de leitura/escritura em memória ou em portas, entre outras.

1.1.4 Outros barramentos: Barramento I2C

Para explorar todos os benefícios dos sistemas e dispositivos eletrônicos, os engenheiros e projetistas visam melhorar à eficiência do hardware e minimizar a complexidade dos circuitos. Para facilitar esta árdua tarefa surgiu o protocolo de comunicação I2C. O protocolo de comunicação em 2 sinais I2C foi originalmente desenvolvido pela Philips em meados de 1996. Atualmente, este protocolo está amplamente difundido e interconecta uma ampla gama de dispositivos eletrônicos. Dentre estes encontramos vários dispositivos de controle inteligente, normalmente microcontroladores e microprocessadores assim como outros circuitos de uso geral, como *drivers* LCD, portas de I/O, memórias RAM e EEPROM ou conversores de dados. Muitas vantagens podem ser atribuídas ao protocolo I2C. Destacam-se entre elas:

- Organização funcional em blocos, providenciando um simples diagrama esquemático final.

- Não há necessidade dos projetistas desenvolverem interfaces. Todos os dispositivos integram as interfaces "on-chip", o que aumenta a agilidade no desenvolvimento.
- Endereçamento e protocolo de transferência de dados totalmente definido via software.
- Possibilidade de inclusão ou exclusão de dispositivos no barramento sem afetá-lo ou outros dispositivos conectados a este.
- Diagnóstico de falhas extremamente simples. O mau funcionamento é imediatamente detectado.
- Desenvolvimento simplificado do software através do uso de bibliotecas e módulos de software reutilizáveis.
- Facilidade no desenvolvimento de placas de circuito impresso, devido à quantidade de interconexões.
- Adicionalmente, utilizando as vantagens da tecnologia CMOS na fabricação dos dispositivos, temos:
 - Baixo consumo de corrente.
 - Alta imunidade a ruídos.
 - Ampla faixa de tensões para alimentação.
 - Ampla faixa de temperatura para operação.

Características Gerais do Barramento I2C

- Suporta qualquer tecnologia de produção.
- Duas vias de comunicação: *serial data* (SDA) e *serial clock* (SCL), ambas bidirecionais, conectadas ao positivo da fonte de alimentação através de um resistor de *pull-up*. Enquanto o barramento está livre, ambas as linhas ficam em nível lógico alto.
- A taxa de transferência máxima é de 100kbit/s no modo padrão (*standard*), ou 400kbit/s no modo rápido (*fastmode*).
- Informação de *carry* entre dispositivos conectados.
- Todo dispositivo possui um endereço único no barramento, independente de sua natureza.
- Qualquer dispositivo conectado pode operar com transmissor ou receptor. Claro que isso depende da natureza do dispositivo - um LCD não vai operar como transmissor, assim como um teclado não operará como receptor. Independente disto, qualquer dispositivo endereçado é chamado de escravo (*slave*).

- O número de interfaces conectadas fica dependente da capacitância máxima do barramento, que é de 400pF – “master bus” incluindo detecção de colisão e arbitração

Definições no barramento I2C

- Transmissor (*Transmitter*): dispositivo que envia dados através do barramento.
- Receptor (*Receiver*): dispositivo que recebe dados através do barramento.
- Maestro (*Máster*): dispositivo que inicia a comunicação gera o sinal de clock e encerra a comunicação.
- *Multi-master*: vários dispositivos podem controlar o barramento, mesmo sem comprometer a mensagem. Quando isto ocorre temos vários dispositivos operando em modo máster.
- Arbitro: procedimento p/ o controle do barramento em modo multi-master. Visa não corromper a transmissão dos dados e perder a sincronismo do clock.
- Sincronização: procedimento p/ sincronizar o clock de um ou mais dispositivos.

Exemplo genérico de um barramento I2C

Um exemplo genérico de um barramento I2C para ser utilizado nos microcontroladores é mostrado na figura 1.3

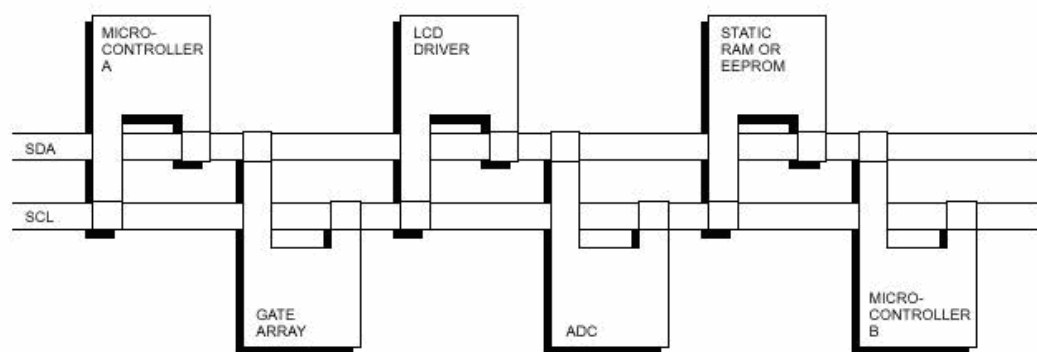


Figura 1.3 Exemplo genérico de um barramento I2C

1.2 Execução das instruções no microcomputador: Fases.

O programa de aplicação, residente em memória de programa, está composto por várias instruções que são executadas sequencialmente pela CPU. Por exemplo, a caixa registradora eletrônica de um supermercado é um microcomputador cujo programa principal realiza várias funções associadas a sua aplicação, uma delas,

por exemplo, é calcular o preço de um produto qualquer, multiplicando sua tarifa (preço/peso) e seu peso.

A execução de uma instrução se realiza em três fases: busca, decodificação (ou interpretação) e execução. Cada CPU tem diferentes tipos de instruções: de transferência de dados, aritméticas, lógicas, manejo de bits, chamadas sub-rotinas, de controle, entre outras.

A seguir se descreverão as três etapas anteriores, considerando que uma CPU genérica de 8 bits deve executar a instrução genérica simbolizada como MOV A, R0, cuja função é mover os 8 bits de dados de um registro interno à CPU chamado R0 até outro registro interno chamado Acumulador (A). Esta instrução de transferência interna é típica em quase todas as CPU existentes. Consideremos que a instrução anterior é previamente armazenada (durante a gravação) no endereço 0 da memória de programa, com o código 1110 1000B (E8H). Este código identifica o tipo de operação (MOV A, R0) que deverá executar a CPU e se denomina **Código de Operação**. Qualquer linguagem de programação (C, BASIC, Pascal, ou ASSEMBLER) é traduzido finalmente a códigos binários que sejam interpretáveis pela CPU.

Quando a CPU genérica é reiniciada, um registro de 16 bits denominado Contador de Programa (ou PC, *Program Counter*, em inglês) carrega-se com o valor 0000H, indicando o primeiro endereço da memória de programa, onde se buscará a instrução a executar. A seguir se mostram as operações internas e externas que realiza a CPU nas diferentes etapas de execução da instrução MOV A,R0.

1.2.1 Busca do código de operação.

A CPU saca o valor do PC (0000H) pelas linhas do barramento de endereços. Apesar de que todas as linhas de endereços estão a 0 lógico, o decodificador de endereços (um dispositivo lógico combinacional que recebe as linhas de endereços e tem diversas saídas de seleção: uma para cada dispositivo de memória ou E/S) só ativa a entrada de seleção da memória de programa, garantindo o acesso à localização 0000H da memória de programa. Logo, a CPU ativa seu sinal de controle de “Leitura em Memória de Programa”, permitindo a transferência, através do barramento de dados, do código E8H armazenado na endereço 0000H, ao registro de instrução (RI), interno na CPU, cuja função é armazenar o código de operação da instrução (ver figura 1.4).

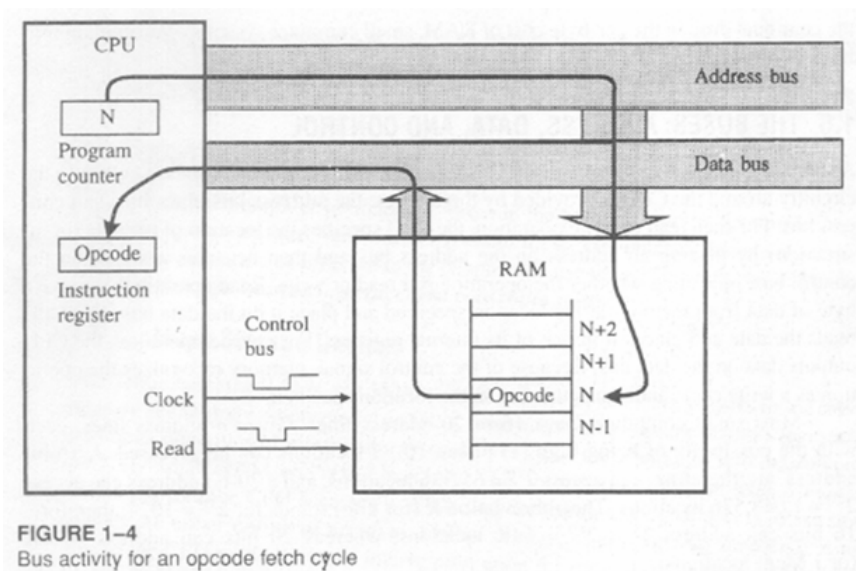


FIGURE 1-4
Bus activity for an opcode fetch cycle

Figura 1.4 Operações nos barramentos de um microcomputador durante as etapas de execução de uma instrução.

1.2.2 Decodificação e Execução da instrução.

Nestas etapas, a unidade de controle (interna à CPU) transfere o conteúdo do RI ao decodificador de instrução, que é um bloco lógico projetado para interpretar todos os possíveis códigos de instruções da CPU em questão, e em consequência, gerar diversos sinais de controle internas e externas, para completar a execução da instrução. Os sinais de controle internos podem ser ordens à unidade lógica aritmética (ALU) para realizar operações aritméticas ou lógicas; operações de incremento, decremento ou transferência com registros internos (ou memória interna); comandos para habilitar/desabilitar mecanismos da CPU (interrupções, DMA), entre outras. Os sinais externos podem ser de leitura de novos dados requeridos pela instrução, ou de escrita em memória ou em dispositivos do E/S, entre outras. No exemplo analisado, o decodificador de instrução recebe o código E8H, decifra que se deve copiar o conteúdo do registrador R0 no registro A, provocando que a unidade de controle ative os sinais internos para conectar a saída do registrador R0 com a entrada do registrador A, via barramento de dados interno, realizando-se assim a transferência.

Os microprocessadores usam um oscilador de pulsos (externo ou interno) para sincronizar todas suas operações internas e externas. O número de pulsos utilizados pelo microprocessador para acessar à memória ou os dispositivos de E/S, chama-se **Ciclo de Máquina** ou **Ciclo de barramento**. O número total de pulsos

utilizados para procurar, decodificar e executar uma instrução, denomina-se **Ciclo de Instrução**. O oscilador pode implementar-se com portas lógicas e redes RC internas ou externas (para aplicações de pouca exatidão e baixo custo) ou com cristais de quartzo externos (para aplicações de maior exatidão).

Um exemplo do ciclo de execução é mostrado na figura 1.5

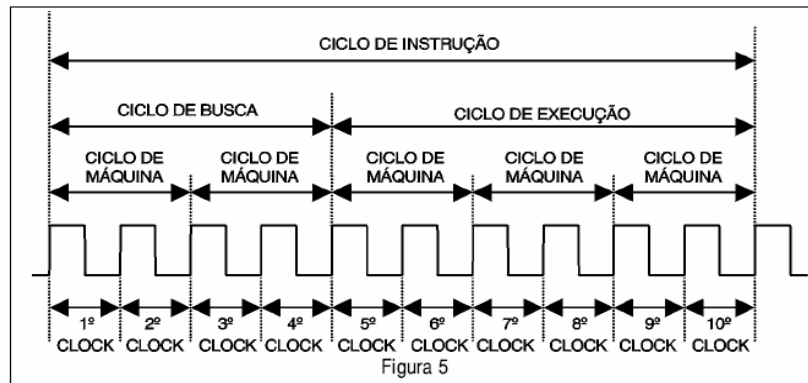


Figura 1.5 Ciclo típico de execução de uma instrução

1.3 Arquitetura do microprocessador/microcontrolador 8051

Na figura 1.6 se mostra um diagrama em blocos com a arquitetura interna do microprocessador/microcontrolador 8051. Nele pode ver-se que este microcontrolador tem:

- CPU de 8 bits com um amplo repertório de instruções que inclui o manuseio de bits individualmente, inclui decodificação de instruções, uma unidade lógico aritmética (ULA) que permite operações de soma, subtração, multiplicação e divisão em 8 bits.
- Memória EPROM interna de 4 KB, para o armazenamento do programa.
- Memória RAM interna de 128 Bytes para armazenar dados.

Além disso, permite endereçar 64KB de memória de Programa e 64KB de memória de Dados, o que permite duplicar o espaço de memória endereçável. Quatro portas de 8 bits cada uma, com linhas quase-bidirecionais. Além de sua função principal algumas destas linhas podem desempenhar funções auxiliares.

- Dois contadores/temporizadores de 16 bits.
- Uma porta serial full duplex, síncrona - assíncrona.
- Cinco fontes de interrupções vetorizadas (3 internas e 2 externas).
- Oscilador de relógio incorporado.

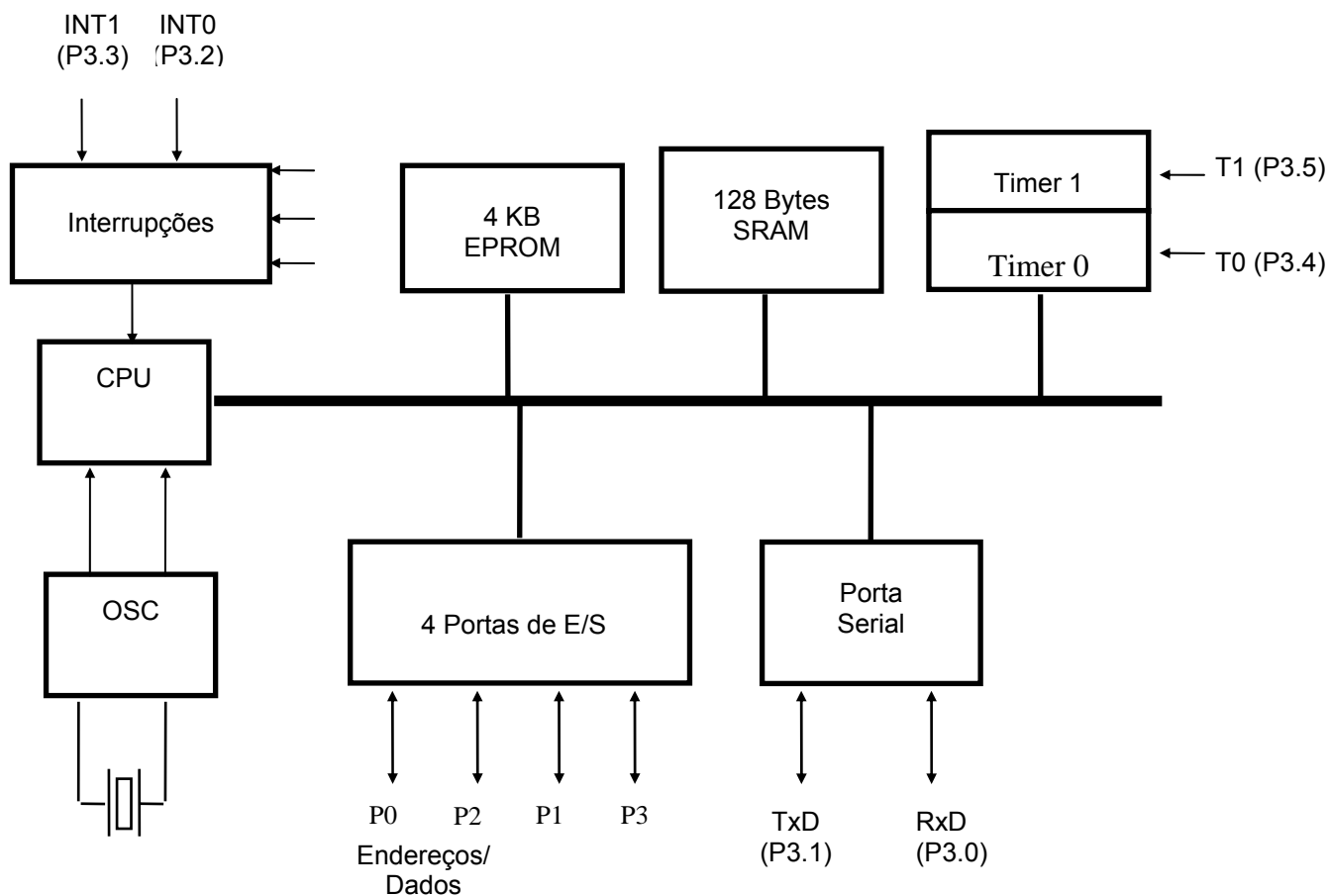


Figura 1.6 Diagrama em blocos do 8051

1.3.1 Terminais e sinais do microprocessador 8051

Na figura 1.7 se mostra a pinagem do microprocessador 8051, em dois tipos de encapsulamento diferentes DIL de 40 pinos e PLCC.

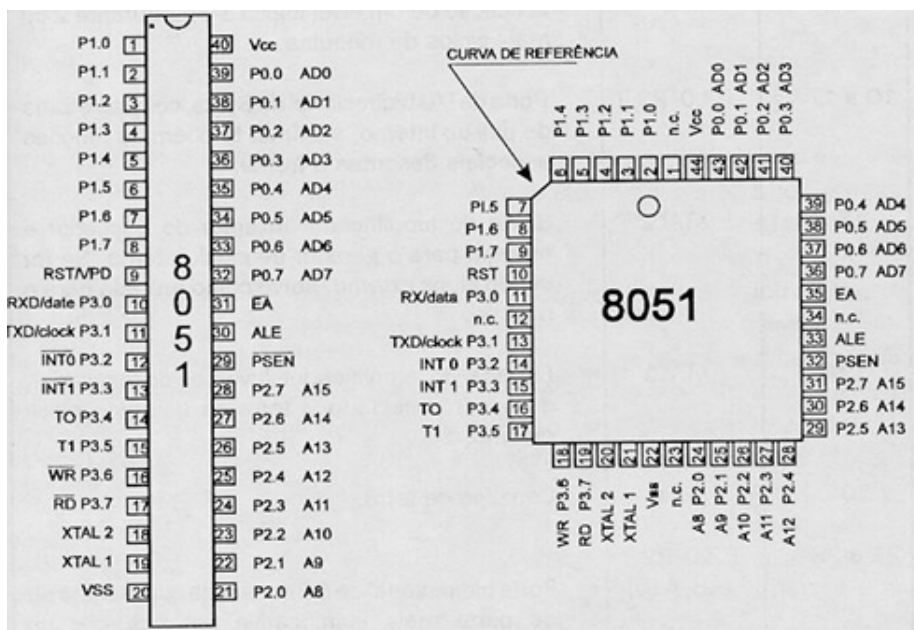


Figura 1.7 Pinagem do 8051 para o encapsulamento: (a) DIL e (b) PLCC

Na figura 1.8 se mostra o diagrama funcional do microprocessador 8051,

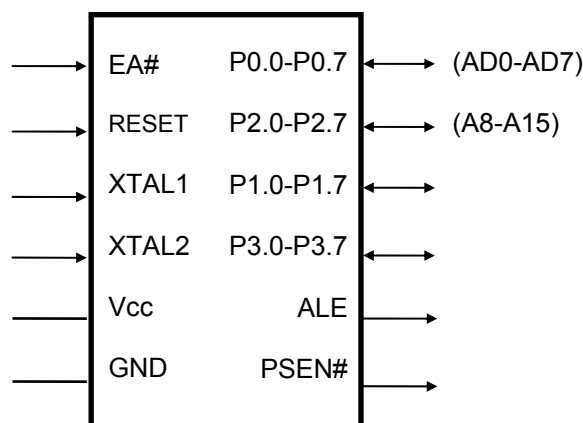


Figura 1.8 Terminais do Microprocessador/Microcontrolador 8051

A seguir, uma descrição breve dos terminais do 8051:

- P0.0 a P0.7: Porta P0. Porta de entrada / saída de uso geral. Quando se emprega memória externa, estes terminais passam a ser um barramento multiplexado de dados e endereços: AD0 a AD7.
- P2.0 a P2.7: Porta P2. Porta de entrada / saída de uso geral. Quando se emprega memória externa, estes terminais são de endereço: A8 a A15.
- P1.0 a P1.7 :Porta P1. Porta de entrada / saída de uso geral.
- P3.0 a P3.7 :Porta P3. Porta de entrada / saída de uso geral. Estes terminais também servem para transportar sinais da porta serial, dos temporizadores, das interrupções e como terminais de controle da memória de dados externa, segundo indica a tabela seguinte:

Bit de P3	Sinal	Função que realiza
P3.0	RxD	Entrada de dados à porto serial
P3.1	TxD	Saída de dados da porto serial
P3.2	INT0#	Entrada da interrupção externa 0
P3.3	INT1#	Entrada da interrupção externa 1
P3.4	T0	Entrada de contagem do temporizador 0
P3.5	T1	Entrada de contagem do temporizador 1
P3.6	WR#	Sinal de escrita em memória de dados externa
P3.7	RD#	Sinal de leitura em memória de dados externa

EA#: *External Access*. Entrada que se utiliza para lhe indicar ao microcontrolador se os primeiros 4 KB de memória de programa são internos (EA# = Vcc) ou externos (EA# = GND):

Endereço	EA# = Vcc	EA# = GND
0 a 0FFFh (4 KB)	ROM / EPROM interna	ROM / EPROM externa
1000h a FFFFh (60 KB)	ROM / EPROM externa	

No microcontrolador 8031 o fabricante conecta este terminal a terra internamente, pois só endereçará memória externa. Para versões com EPROM (8751) esse terminal receberá a tensão de programação (V_{pp}) quando se programa essa memória.

PSEN#: (*Program Store Enable*), Saída: Sinal de controle de leitura da memória de programa externa. Deve ser usada como sinal de seleção de Memória de Programa Externa. PSEN# = 0, o microcontrolador acessa à Memória de Programa Externa.

ALE: (*Address Latch Enable*). ALE = 1 indica que pela porta P0 está saindo a parte baixa (A0 - A7) de um endereço de memória (de programa ou de dados). Podem-se capturar estes bits de endereço usando um *latch octal* sincronizado com o sinal ALE, como se mostra na figura 1.9. Este sinal só se emprega quando há memória externa (de programa ou de dados).

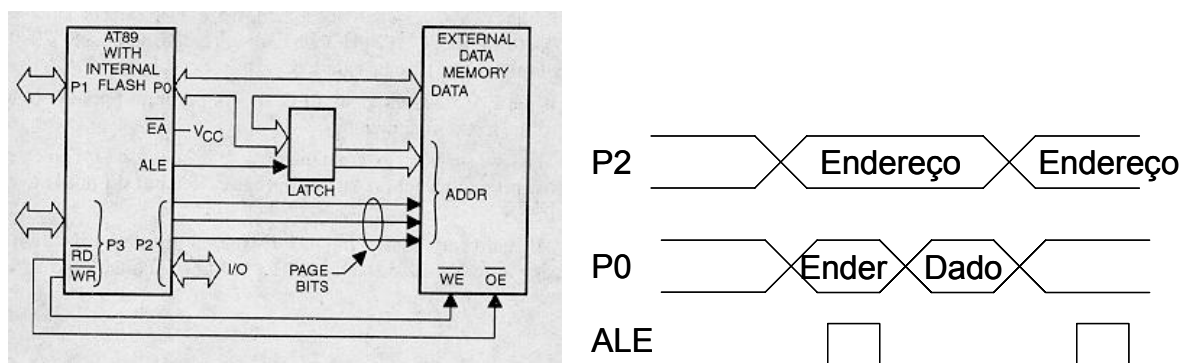


Figura 1.9 De-multiplexado de dados e endereço usando o sinal ALE e um latch 74LS373

RESET: O *reset* do microcontrolador se produz mantendo este sinal em nível alto ao menos 24 períodos do oscilador. Por exemplo, se o oscilador for de 12 MHz, o reset deve durar pelo menos 2 μ s. Ao finalizar o sinal de reset, o microcontrolador “salta” ao endereço 0 da memória de programa (interna ou externa segundo EA=1 ou 0, respectivamente). Também alguns registradores de funções especiais (SFR) tomam determinados valores:

PC = 0000, PSW = 00, ACC = 00, SP = 07, B = 00, DPTR = 0000.

1.4 Organização da memória

A diferença dos microprocessadores da família xx86 da Intel, que utilizam um único espaço de memória (Arquitetura de Von Newman), os microcontroladores 8051/52 empregam dois espaços separados de memória (Arquitetura Harvard):

Memória de Programa. (só leitura, ROM /EPROM).
Memória de Dados. (Leitura / escrita, sRAM).

Não existe um espaço independente de entrada / saída: as portas são endereçáveis dentro da memória de dados.

1.4.1 A memória de programa

A memória de programa pode ser interna e/ou externa, como se ilustra a seguir:

- Interna: 4KB
- Interna e externa: 4KB + 60KB
- Externa: 64KB

A figura 1.10 ilustra estas configurações possíveis da memória de programa:

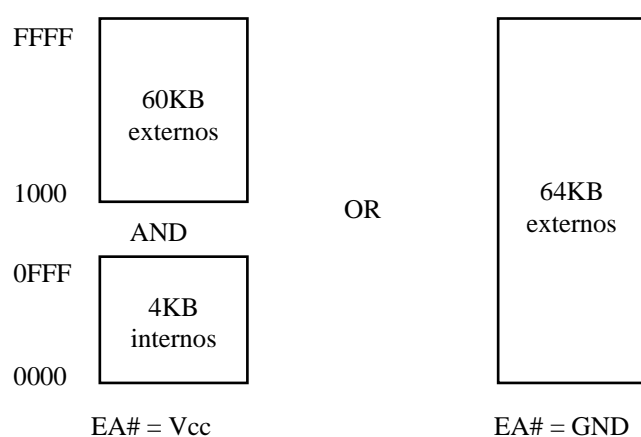


Figura 1.10 Configurações possíveis da memória de programa.

Depois de um reset, o microcontrolador “salta” ao endereço 0 da memória de programa interna se $\#EA = V_{cc}$ ou da memória externa se $\#EA = GND$.

A Memória de Programa só é acessível em leitura e quando é externa. O sinal de controle empregado para validar a leitura é PSEN (*Program Store Enable*, em português: sinal de habilitação de programa armazenado). Essa memória sempre se endereça com 16 bits.

1.4.2 A memória de dados

A memória de dados está dividida em três partes:

- A memória RAM interna de 128 bytes.
- Os registros de funções especiais (SFR), em 128 bytes adicionais.
- A memória RAM externa, que pode chegar até 64KB.

Memória de Dados:

- Interna: 128 bytes + SFR

- Interna y externa: 128 bytes + SFR + 64 KB

O uso da memória de dados externa não inabilita a RAM interna, ou seja, a memória de dados externa acrescenta à memória de dados interna, não a substitui. A figura 1.11 ilustra esta situação.

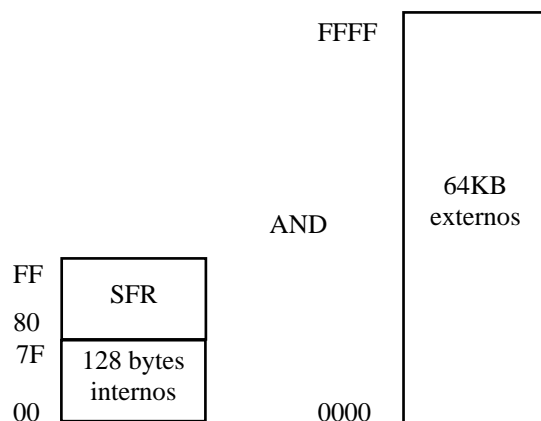


Figura 1.11 Configurações possíveis da memória de dados.

A RAM interna

A RAM interna de 128 bytes está estruturada da seguinte forma:

- 32 bytes: constituem um espaço endereçável por bytes (endereços de 00h até 1Fh). Este espaço também se organiza em 4 bancos de 8 registradores de 8 bits. Os registradores em cada banco se denominam R0 a R7.
- 16 bytes: constituem um espaço endereçável por bytes (endereços 20h a 2Fh) ou por bits (128 bits, endereços 00h a 7Fh).
- 80 bytes: constituem um espaço endereçável por bytes (endereços 30h a 7Fh).

A figura 1.12 ilustra a estrutura da memória RAM interna:

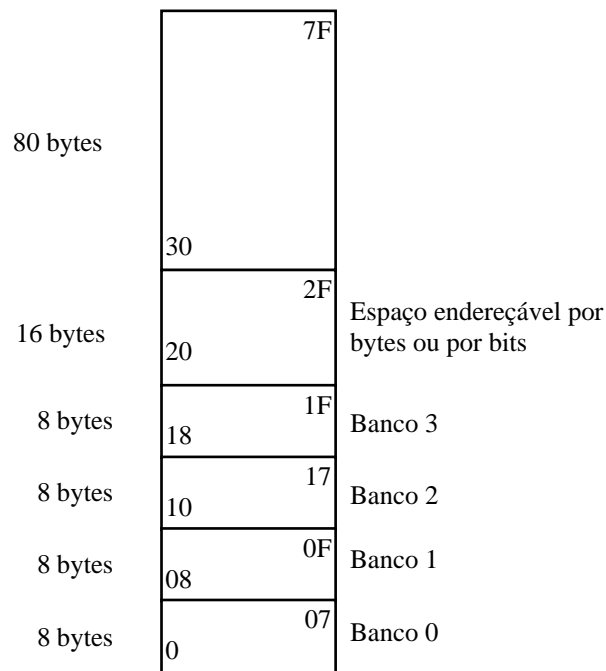


Figura 1.12 Organização da RAM interna.

Os Registradores de Funções Especiais (SFR)

Os endereços 80h - FFh da memória de dados interna estão reservados para os chamados registradores de funções especiais (SFR: *Special Function Registers*). Aqui ficam os registradores convencionais de um microprocessador (o acumulador ACC, o registrador B, o registrador de estado PSW e o ponteiro da pilha SP), assim como as portas do microcontrolador e seus registradores associados. Também encontramos o registro DPTR que serve como ponteiro de dados na RAM externa. Um detalhe importante é que os registradores cujos endereços são múltiplo de 8, ou seja, a primeira coluna da tabela, são também endereçáveis por bits, ocupando os endereços desde 80h até FFh. Observe então que o acumulador, o registrador de estado PSW e as portas paralelas são endereçáveis por bits.

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

PSW	<i>Program Status Word.</i>
ACC	Acumulador (registrador A).
B	Registrador B (usado em operações de multiplicação e divisão)
SP	Ponteiro de pilha. Depois de um reset, SP=07.
DPTR (DPH, DPL)	Ponteiro de dados na RAM externa.
P0, P1, P2, P3	Registradores associados às portas paralelas.
SCON, SBUF	Registradores associados à porta serial.
TH0, TL0, TH1, TL1 TCON, TMOD	Registradores associados aos <u>temporizadores</u> .
IE, IP	Registradores associados às interrupções.
PCON	<i>Power Control Register</i>

O registrador de estado ou PSW (em inglês) indica o estado de vários bits que mudam em dependência de algumas instruções.

PSW (Program Status Word)

Endereço: 0D0h; bits 0D0h a 0D7h

7	6	5	4	3	2	1	0
C	AC	F0	RS1	RS0	OV	-	P

bit	Significado	
0	P	Paridade. Se P=0 há paridade par. Se P=1 há paridade ímpar.
1		Reservado
2	OV	Transbordamento (<i>Overflow</i>).
3, 4	RS1, RS0	Seleção do banco de registros.
5	F0	Bandeira disponível para o usuário.
6	AC	Transporte auxiliar
7	C	Transporte

1.4.3 Uso de memória externa

A conexão de uma memória externa cancela o uso das portas paralelas P0 e P2, que agora passam a ser barramentos externos do microcontrolador, que neste caso se comporta como microprocessador:

- P0.0, ..., P0.7 = AD0, ..., AD7 (Linhas de dados D0...D7 multiplexadas com as linhas de endereço A0...A7)
- P2.0, ..., P2.7 = A8, ..., A15 (Linhas de endereço diretas A8...A15)

Para conectar uma memória padrão se necessita um *latch octal* (74LS373) e o sinal ALE#, numa estrutura que permite separar ou demultiplexar os sinais do barramento de dados com os sinais da parte baixa do barramento de endereços.

A memória externa pode ser conectada de duas formas: **segregada** ou **combinada**:

Memória segregada: estão separados os espaços de memória de programa e dados. A memória de programa se lê com o sinal PSEN# e a de dados com o sinal RD# (P3.7). Cada espaço pode ter até 64 KB, em total pode haver até 128 KB.

A memória de dados se pode organizar em páginas de 256 bytes. Neste caso, o porto P2 serve para endereçar as páginas (Figura 1.12).

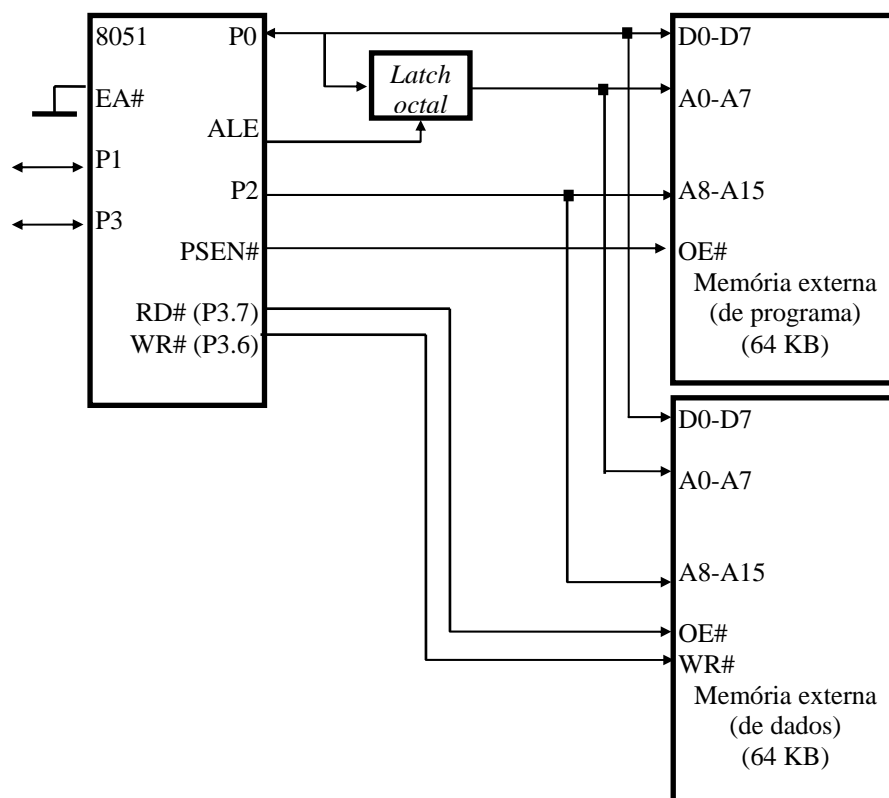


Figura 1.12 Conexão da memória externa segregada.

Observe que PSEN# é um sinal de leitura, só da memória de programa externa.

Memória combinada: há um único espaço de memória externa de até 64 KB. Consegue-se combinando em um só sinal de leitura ao PSEN# e RD# mediante uma porta AND. Com este tipo de conexão se trabalha de forma análoga a como se faz na arquitetura *Von-Neuman* (Figura 1.13).

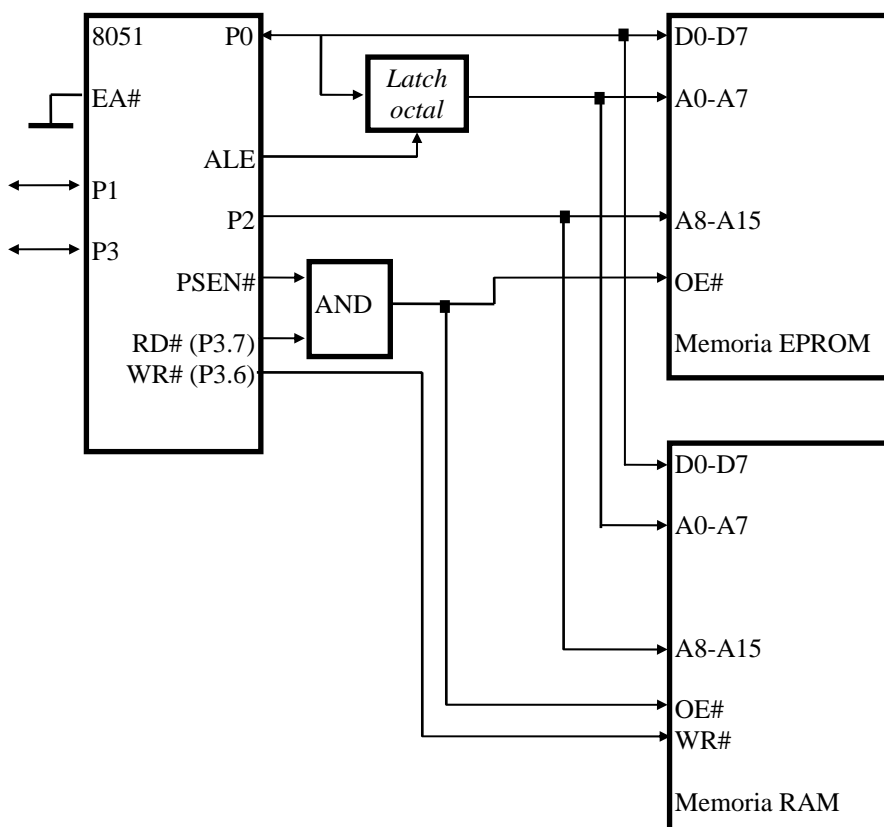


Figura 1.13 Conexão da memória externa combinada.

1.5 Conexão de chips externos de memória

Como foi estudado anteriormente, a um 8051 pode ser adicionado chips de memória externa de programa e/ou de dados. Continuando, veremos quais são os terminais típicos de algumas memórias com o objetivo final de estudar como é que se conectam os diversos chips de memória a um sistema com microprocessadores, e em especial ao microprocessador 8051. Como é conhecido os chips de memórias se classificam em: (a) Memórias de só leitura ou memórias ROM, entre as quais temos as memórias propriamente ROM (só leitura), PROM (programáveis uma vez), EPROM (programáveis eletricamente e apagáveis com luz ultravioleta), EEPROM e FLASH (programáveis e apagáveis eletricamente), e (b) Memórias de leitura/escrita ou RAM, classificadas em RAM estáticas (sRAM) e dinâmicas (dRAM).

1.5.1 Terminais típicos de uma memória.

Não existem nomes universais para os terminais das memórias, embora tampouco existem nomes muito diferentes. De forma genérica, por exemplo, para uma memória de 2 K x 8 estes terminais são:

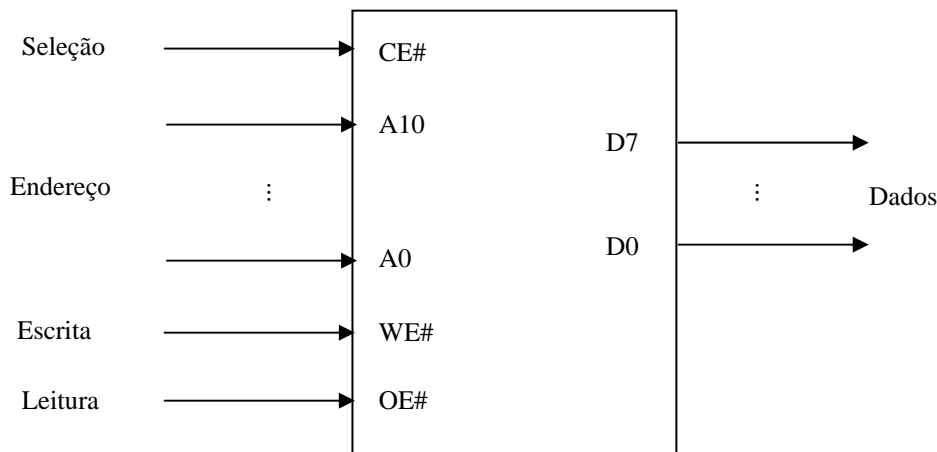


Figura 1.14 Terminais típicos de uma memória genérica de 4 kb

Uma memória de 2K x 8 é um conjunto de localizações de 8 bits que têm os endereços internos que vão da 0 a 7FFh.

A seguinte tabela-função mostra as funções dos terminais:

CE#	OE#	WE#	Operação
1	x	x	D0 - D7 em 3ero. estado
0	0	1	Leitura: D0 - D7 em saída
	1	0	Escrita: D0 - D7 em entrada
	1	1	D0 - D7 em 3ero. estado

1.5.2 Decodificação de endereços.

A memória externa num microprocessador ocupa uma faixa de endereços. O mesmo acontece com os dispositivos de entrada saída externos (portas paralelas, temporizadores, controladores de teclado e display, etc.). Por exemplo, a memória externa de dados no 8051 ocupa desde o endereço externo 0000H até o endereço final FFFFH. A memória de programa pode ocupar desde o endereço 1000H (se #EA = 1) ou o 0000H (se #EA = 0) até FFFFH. Frequentemente se usam diferentes chips de memória. Dado que em um instante de tempo somente pode estar conectado o microprocessador e um dispositivo de memória (ou de entrada/saída),

então é necessário uma lógica de seleção implementada com um chip, chamado **decodificador de endereços**, que selecione cada um dos diferentes chips de memória (o de E/S) em dependência da faixa de endereços, que o usuário atribuía a cada um deles dentro dos endereços de memória globais do sistema.

Outras vezes, se quer acrescentar um chip de memória ou de E/S a um sistema pronto, que só tem livre uma faixa de endereços para expansão (caso de um PC) e temos que usar uma lógica de seleção para evitar conflitos nos barramentos, ou seja, que esse chip somente envie ou receba dados na faixa de endereços que ele tenha atribuído.

A lógica de seleção ou decodificador de endereços é um circuito combinacional que se pode realizar utilizando diferentes circuitos integrados:

- Circuitos integrados de pequena e media escala de integração, tais como portas AND, OR, NAND, etc., decodificadores binários 3 a 8 (74LS138) ou de 4 a 16 (74154), comparadores binários (74LS688), entre outros.
- Memórias PROM bipolares (são muito rápidas) como, por exemplo, a 82S147 de 512 x 8.
- Dispositivos lógicos programáveis (PLD) de diversos tipos: PLA, PAL, Gal, PALCE.

Na figura 1.15 se mostra a filosofia da lógica de seleção para o caso típico em que se deseje colocar um chip de memória numa faixa de endereços atribuída, num sistema dado.

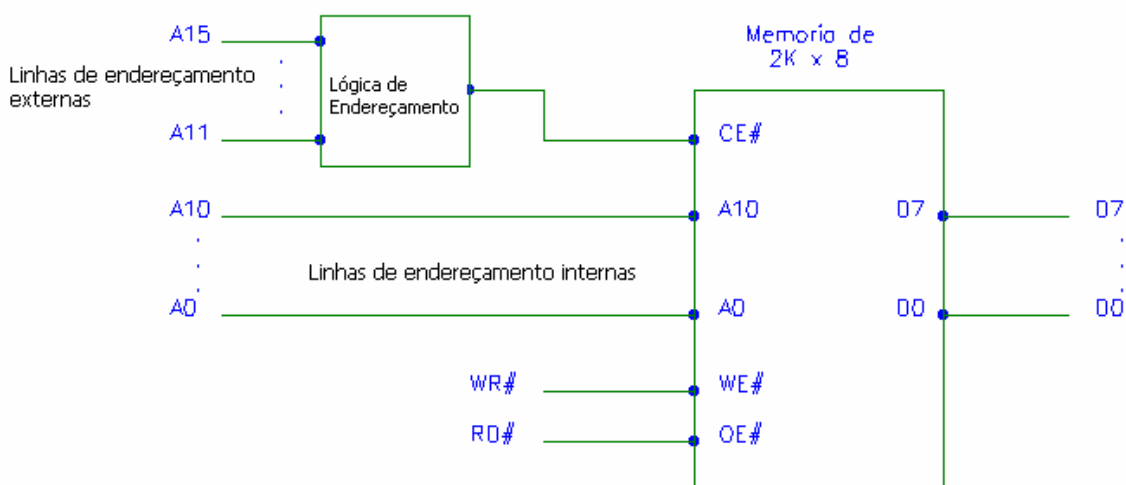


Figura 1.15 Definição das linhas de endereçamento internas e externas para uma memória RAM de 2048 localizações de 8 bits (ou 2 kb).

Neste caso, a memória tem uma capacidade de 2 kb, pelo que usa 11 linhas de endereço para acessar a cada uma das 2048 células de 8 bits internas. Denominemos a essas linhas como **linhas de endereçamento internas**, pois definem o endereço interno (ou local) que vai ser acedido. O primeiro endereço local seria aquele que tem todos seus bits em zero (endereço local 0) e o último endereço será aquele que tem todos seus bits em um (endereço local 2047). Quando a memória é conectada a um sistema que tem vários chips, então esse chip de memória ocupará uma faixa de endereços globais dentro do sistema. Por exemplo, se o endereço inicial atribuído é o 1000H, que é o primeiro endereço externo na memória de programa de um 8051 quando se usa memória de programa interna então o endereço final dessa memória será $1000H + 7FFH = 17FFH$. Então a lógica de endereçamento (decodificador de endereços) deverá garantir que nessa faixa (1000H-17FFH) só fluam pelos barramentos os dados entre o microprocessador e essa memória de programa externa, e nenhum outro dispositivo fique conectado aos barramentos. As linhas de endereço que são usadas para diferenciar entre diversos chips no sistema se denominam **linhas de endereçamento externas**, e por tanto, serão as linhas que serão conectadas ao decodificador de endereços (nas figuras 1.15 e 1.16 serão as linhas A11 até A15).

Exemplo

Endereçar uma memória de 2K x 8 a partir da direção A000h.

Endereço inicial: A000 == 1010 0000 0000 0000

Endereço final: A7FF == 1010 0111 1111 1111

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	A000H
1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	A7FFH

linhas de endereço externo linhas de endereço interno

Como pode observar-se, as linhas de endereçamento externo (bits que identificam a zona de endereços de A000h a A7FFh) são A11 a A15. O padrão de bits a decodificar é 0100.

A lógica de seleção poderia ser:

$$CE\# = A15\# + A14 + A13\# + A12 + A11 = (A15 + A14\# + A13 + A12\# + A11\#)\#$$

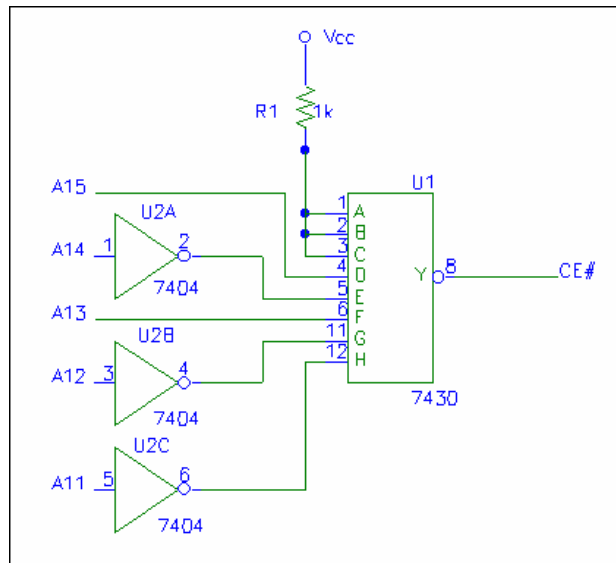
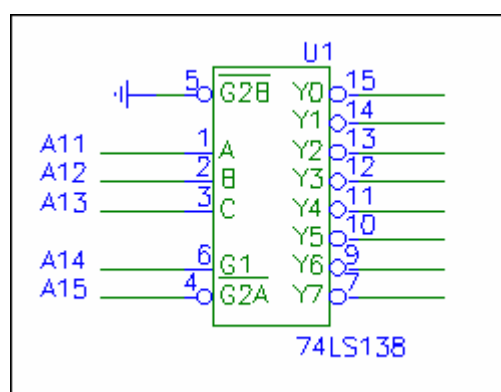


Figura 1.16 Decodificador de endereço do exemplo

A pergunta chave para obter um projeto eficiente é: Pode-se reduzir o número de circuitos integrados ? . Obviamente, na medida em que o número de CIs seja menor, o espaço físico que ocupará a lógica será menor, a confiabilidade será maior, o consumo de energia será menor e provavelmente o custo será menor.

Exemplo

Determinar os endereços que selecionam cada um dos terminais do seguinte decodificador de endereços.



Este é um exemplo típico quando um usuário se enfrenta à problemática de pesquisar um decodificador de endereços que funciona em alguma placa pronta e que se quer reusar para outra aplicação: Por tanto, o projetista deverá ter certeza das diversas faixas de endereço que ocupa cada saída do decodificador, cada uma das quais estarão conectados os diversos chips da placa. A solução, para este

exemplo, se mostra na seguinte tabela, onde a primeira linha corresponde com a saída Y0 e a ultima linha à saída Y7:

Linhas de endereços (A15 ...A0)																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Endereços
0	1	0	0	0	X	X	X	X	X	X	X	X	X	X	X	4000-47FF
		0	0	1	X	X	X	X	X	X	X	X	X	X	X	4800-4FFF
		0	1	0	X	X	X	X	X	X	X	X	X	X	X	5000-57FF
		0	1	1	X	X	X	X	X	X	X	X	X	X	X	5800-5FFF
		1	0	0	X	X	X	X	X	X	X	X	X	X	X	6000-67FF
		1	0	1	X	X	X	X	X	X	X	X	X	X	X	6800-6FFF
		1	1	0	X	X	X	X	X	X	X	X	X	X	X	7000-77FF
		1	1	1	X	X	X	X	X	X	X	X	X	X	X	7800-7FFF

1.5.3 Teste de um decodificador de endereços.

Na posta em funcionamento de um sistema pratico com microprocessadores, uma das primeiras tarefas de um projetista deve ser testar o decodificador de endereço, pois se ele não funciona adequadamente, o sistema não vai trabalhar devido a que não vai poder aceder aos diversos chips do sistema. O seguinte exercício mostra como isto pode ser realizado, neste caso, usando o simulador Professional PROTEUS ISIS/VSM. No anexo, o estudante pode ver os fundamentos de trabalho com o software de simulação PROTEUS ISIS/VSM.

Exercício Resolvido

Projetar um decodificador de endereço com um SN74LS138 para conectar 8 bancos de memórias de 1 kb cada um a partir do endereço 8000 H da memória de dados. Realizar um programa para ler e escrever em cada faixa de endereços e testar com o osciloscópio do programa PROTEUS ISIS/VSM.

Solução:

Neste caso, cada memória tem 1 kb pelo que as linhas de endereço internas são de A0 até A9 e as linhas de endereço externo serão as restantes linhas: A10 até A15.

2. Editar e compilar o programa de teste.

O programa de teste deve ser muito simples (para testar o hardware) e realizará a leitura e a escrita num endereço de memória externa da faixa desejada. Desta forma o usuário deveria observar no osciloscópio a geração do pulso de seleção para cada terminal de seleção (#CS) das memórias conectadas em cada saída do decodificador de endereços. As ações a realizar serão:

2.1 Editar o programa de teste: Menu Source --- Add/Remove Source File ---- Selecionar a ASEM51 em *Code Generation Tool* ---- New --- buscar a pasta de trabalho (C:\8051\EX8051) --- Digitar o nome do arquivo de teste (MEMO.ASM)--- Abrir---Responder SIM ante a pergunta de “se deseja criá-lo” ---Digitar OK --- menu Source ----MEMO.ASM (se abrirá um editor de texto) ---Digitar o seguinte programa

```
ORG 0000H
                                MOV DPTR,#83FFH
AGAIN:  MOVX A,@DPTR
                                MOVX @DPTR,A
                                JMP AGAIN
END
```

Finalmente salvar com a opção **Save** dentro do menu *File*.

2.2 Compilar o programa MEMO.ASM a seu código executável MEMO.HEX: Menu Source --- Build All

A compilação com sucesso se comunica a traves da seguinte janela (Figura 1.18):

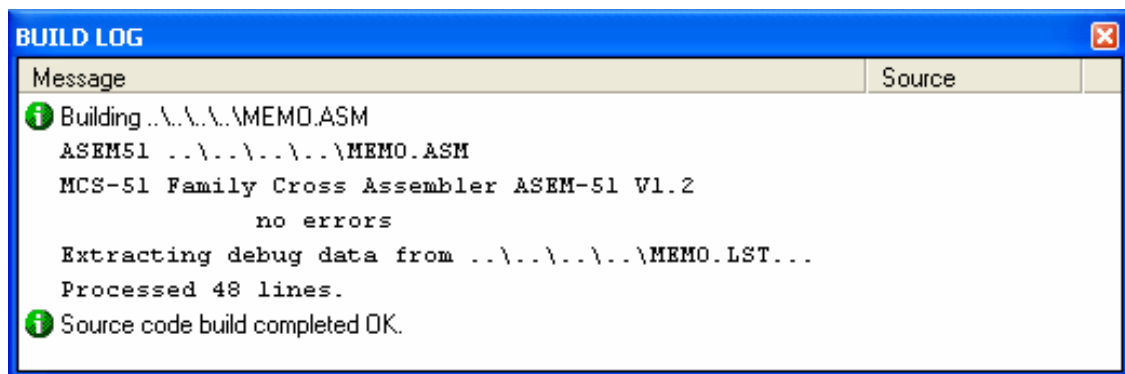
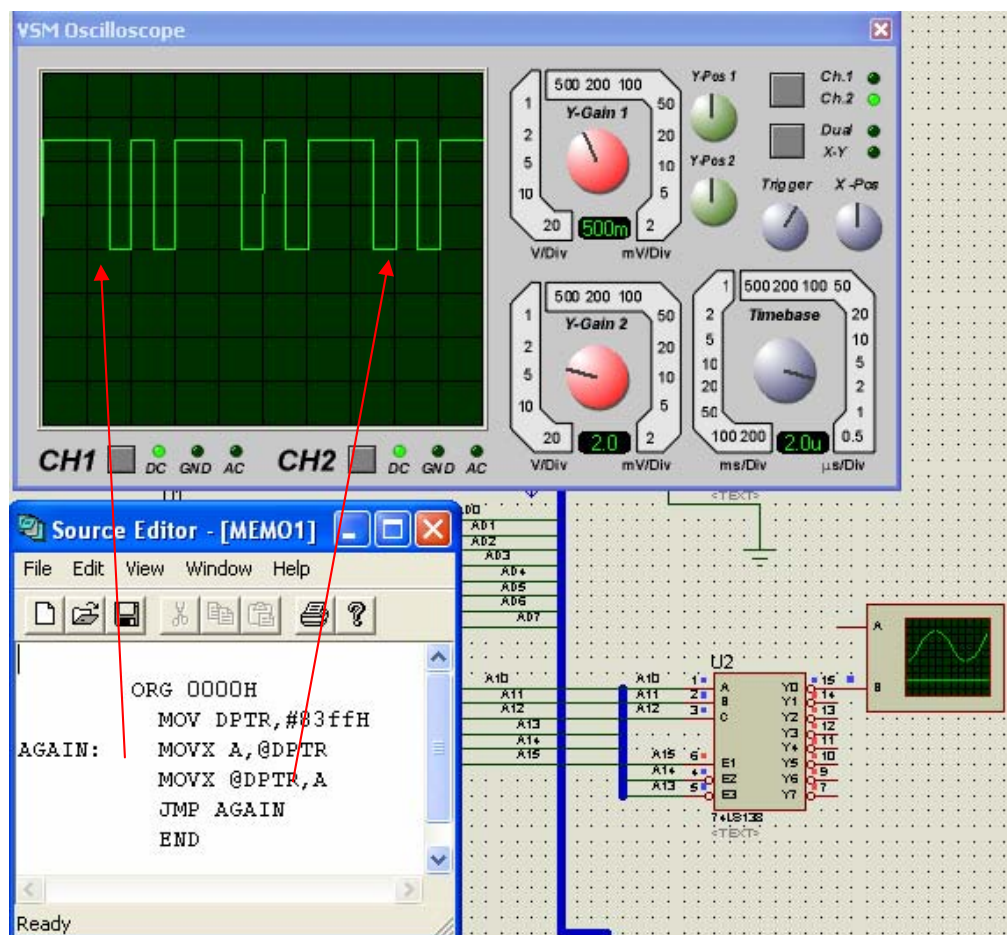


Figura 1.18 Janela de comunicação de compilação com sucesso

2.3 Carregar o arquivo **MEMO.HEX** gerado, no microprocessador 8051: Duplo clique acima do 8051 e buscar o arquivo **MEMO.HEX** na pasta de trabalho (C:\8051\EX8051).

2.4 Executar o programa de teste: Pressionar o botão **PLAY**. Na figura seguinte se mostra à forma de onda na saída Y0 (canal B do osciloscópio) quando se executa o programa de teste mostrado. Observar que o endereço que aponta o registrador

DPTR é 83FFh que é o ultimo endereço correspondente a essa saída Y0 (faixa de endereços de 8000H – 83FFH). Qualquer endereço pertencente a essa faixa ativa a saída Y0. Também se observa que aparecem dois pulsos consecutivos: o primeiro é o sinal de seleção causado pela execução da instrução **MOVX A,@DPTR** que habilita a memória quando realiza uma operação de leitura nela (habilita o #CS da memória conectada à saída #Y0) e o segundo pulso é o sinal de seleção causado pela execução da instrução **MOVX @DPTR,A** que ativa a memória quando se realiza uma operação de escrita nela. O estudante pode carregar no registrador DPTR outro endereço, por exemplo, 8400 H e observar que a saída Y0 não será ativada. Nesse caso se ativará a saída Y1 que corresponde à faixa de endereços 8400H - 87FFH segundo a tabela anterior.



(b)

Figura 1.19. Decodificador de endereços: (a) Circuito eletrônico (b) Programa de teste e sinais de habilitação na saída Y0 durante a leitura (MOVX A, @DPTR) e a escrita (MOVX @DPTR, A)

2 PROGRAMAÇÃO DOS MICROPROCESSADORES

Os microcontroladores se programam tradicionalmente empregando a linguagem *assembler* do dispositivo particular. Apesar de que um programa em linguagem *assembler* se executa rápido, este tem várias desvantagens. Um programa em *assembler* está composto por nemónicos e é difícil sua aprendizagem e manutenção. Além disso, os microcontroladores fabricados por diversas empresas têm diferentes linguagens *assembler* e o usuário deve aprender uma nova linguagem cada vez que empregue um novo microcontrolador. Os microcontroladores podem programar-se também empregando uma linguagem de alto nível, como BASIC, Pascal, e C. As linguagens de alto nível têm a vantagem de que são muito mais fáceis de aprender que o *assembler*. Do mesmo modo, o uso de uma linguagem de alto nível permite desenvolver programas grandes e complexos com maior facilidade, com o custo de usar mais memória (atualmente não é problema). Os programas de alto nível permitem inserir instruções em linguagem *assembler* pelo que se pode aproveitar as vantagens das duas linguagens. Nesta apostila se aprenderá a programação do microcontrolador 8051 empregando as linguagens *assembler* e C. Em ambos casos se usará o compilador de código livre SDCC (*Small Device C Compiler*), disponível na internet.

2.1 Modos de endereçamento dos dados

Os programas estão formados por instruções. Cada instrução tem um código de operação que a identifica, e pode usar (no caso de 8051) de zero até 3 operandos. Os programas comandos ou instruções do microprocessador têm o seguinte formato:

Código de operação [operando1], [operando2], [operando3]

O código de operação identifica a operação que realiza a instrução. Os operandos definem os dados que a operação necessita.

A seguir alguns exemplos de instruções, organizados segundo o formato anterior:

Nº Op.	COD. OP.	OPER. 1	OPER. 2	OPER. 3	Função
0	RET				Retornar de sub-rotina
1	CLR	A			Zerar acumulador
2	MOV	A	B		Mover de registrador B para o A
3	CJNE	A	#55H	Endereço	Comparar A com dado 55H e se são diferentes, saltar para um endereço

Cada uma das instruções anteriores está escrita em linguagem *assembler*, que é uma linguagem de baixo nível, mais o menos compreensível pelo usuário. Na verdade o microprocessador só trabalha com códigos binários, pelo que cada instrução deve ser traduzida a um conjunto de 0s e 1s, que o chamado código de máquina. Regularmente, o número de operandos da uma ideia do número de bytes associado a cada instrução. No 8051 as instruções podem ocupar desde 1 byte até 3 bytes. O decodificador de instrução (interno na CPU) é projetado para interpretar todas as instruções do microprocessador. O primeiro byte de cada instrução sempre é o código de operação, embora às vezes também contem informação de algum operando, como por exemplo, a instrução **MOV A, Rn** (Rn pode ser qualquer registrador desde R0 a R7) é codificada em binário como 11101RRR (RRR = 000 para R0, RRR = 111 para R7). Os projetistas de CPU fazem isto para ganhar em eficiência na codificação das instruções e por tanto do uso da memória.

O **modo de endereçamento** é a forma em que o microprocessador acessa a um dado que requer para executar uma instrução ou para armazenar o dado resultante da execução de uma instrução (refletidos na instrução através do operando), assim existem modos de endereçamento:

No 8051, ao igual a em um sistema apoiado em um microprocessador, os dados podem dar-se formando parte das instruções, podem estar em um registrador ou na memória. Daqui surgem 4 modos básicos de endereçar um dado:

- Endereçamento imediato.
- Endereçamento por registrador.
- Endereçamento direto.
- Endereçamento indireto por registrador.
- Endereçamento indireto indexado por registrador.

2.1.1 Endereçamento imediato

O dado aparece especificado na própria instrução (no 2do. Byte da instrução, depois do código de operação). Na programação em assembler é usado o símbolo # para indicar este tipo de endereçamento.

Exemplos:

MOV A, #30h ; Carregar o ACC com 30h

MOV A, #CTE ; Carregar o ACC com o valor atribuído ao símbolo CTE

2.1.2 Endereçamento por registrador

O dado está em um dos registradores R0 a R7 do banco de registradores selecionado.

Exemplo:

MOV A, R1; Carregar ACC com o conteúdo de R1

Observe que neste caso R0 a R7 não são tratados como parte da memória de dados interna senão como registradores propriamente ditos. Ao codificar a instrução se utilizam alguns bits do código de operação para indicar o registrador usado (3 bits pois há 8 registradores).

Quando o dado está em memória (de dados ou de programa, interna ou externa) empregam-se os endereçamentos direto e indireto.

2.1.3 Endereçamento direto

O endereço do dado se dá no segundo byte da instrução. Este modo de endereçamento só é aplicável aos dados na memória interna (RAM interna de 128 bytes, SFR) e ao espaço endereçável por bits.

Exemplos:

MOV 20H, A ; Carregar o endereço 20h na RAM interna com o conteúdo do ACC.

MOV P1, #35h ; Carregar o valor 35h no endereço P1 (o valor do símbolo P1 se estabeleceu previamente). Se P1 = 90h, então essa instrução está indicando que se escreverá 35h no porto P1.

2.1.4 Endereçamento indireto por registrador

O endereço do dado está num registrador (R0, R1 ou DPTR). O endereçamento indireto não é aplicável a dados nos SFR ou no espaço endereçável por bits. Em *assembler*, indica-se com o símbolo @, ou seja, @R0, @R1, @DPTR.

Exemplos:

MOV @R0, #20h ; Carrega 20h na RAM interna endereçada por R0.

MOVBX @R0, A ;Carrega A na RAM externa endereçada por R0 (a RAM externa está paginada).

MOVBX @DPTR, A ; Carrega o conteúdo de A na RAM externa endereçada pelo DPTR.

2.1.5 Endereçamento indireto indexado por registrador

Somente a memória do programa pode ser acessada mediante este modo de endereçamento e só em leituras. Este modo de Endereçamento é utilizado nas leituras de tabelas da memória do programa ou dados que se encontram como constantes. Um registro de 16 bits (DPTR ou PC) aponta ao endereço base (endereço de início) da tabela e mediante o Acumulador se estabelece o número da entrada da tabela (índice dentro da tabela). O endereço da entrada da tabela na memória do programa está formado pela soma do Acumulador e o Ponteiro Base (DPTR ou PC). Outro tipo do Endereçamento indexado é usando a instrução `JMP @A+DPTR`. Neste caso o endereço do destino do salto é calculado como a soma do ponteiro base (DPTR) mais o Acumulador (A).

Exemplo:

`MOVC A,@A+DPTR`: Move uma constante que se encontra na memória do programa. O Acumulador é carregado com o dado que se encontra pontado pelo endereço formado pela soma do Acumulador A e o Apontador de Dados.

`MOVC A,@A+PC`: O Acumulador é carregado com dado que se encontra na endereço formada pela soma do mesmo Acumulador A e o Contador do Programa (PC).

Exemplo 1:

Programa	Sub-rotina
Principal	Tabela: <code>INC A</code> ;Para compensar à instrução <code>RET</code>
.....	<code>MOVC A,@A+PC</code>
.....	<code>RET</code>
<code>CALL Tabela</code>	<code>DB 66H</code>
.....	<code>DB 77H</code>
.....	<code>DB 88H</code>
.....	<code>DB 99H</code>

Se a Sub_rotina Tabela é chamada com o acumulador (A)=1; quando retornar (A)=77H.

Exemplo 2:

	Em Memória
.....	Tabela: <code>DB 30H</code>	
<code>MOV DPTR,#Tabela</code>	Tabela+1: <code>DB 31H</code>	
<code>MOV A,#3</code>	Tabela+2: <code>DB 32H</code>	
<code>MOVC A,@A+DPTR</code>	Tabela+3: <code>DB 33H</code>	
.....	Tabela+4: <code>DB 34H</code>	

Depois da instrução em A aparece o valor 33H.

2.1.6 Endereçamento de bit

Os endereços dos bits a acessar se encontram nos SFR endereçáveis bit a bit e na RAM interna. Como só se emprega um byte para especificar o endereço o número de bits endereçáveis está limitado a 256.

Exemplos:

MOV C, 97h; Levar ao *carry* C o valor do bit 97h do espaço endereçável por bits.

Neste caso, lê-se o bit 7 da porta P1: pode-se escrever também MOV C, P1.7

ANL C,17H ; Levar ao *carry* C o valor do AND de C com o valor do bit 17h do espaço endereçável por bits. Neste caso, quando se opera com o bit com endereço 17H, lê-se o bit 7 do byte da RAM interna cuja direção é 22H .

A seguir se mostra uma tabela onde se resumem os modos de endereçamento empregados para acessar a dados situados na memória.

Tabela resumo dos modos de endereçamento de dados na memória.

Dados em:	Endereçamento		Se for indireto, o endereço fica em:
	Direto	Indireto	
RAM interna de 128 bytes	X	X	R0, R1
SFR	X		
RAM externa		X	R0, R1 se endereço é de 8 bits DPTR se endereço é de 16 bits Só vale a instrução MOVX
Espaço endereçável por bits	X		
Memória de programa		X	DPTR Só vale a instrução MOVC

Na figura 2.1 se mostra na área da RAM interna (DRAM) endereçável por bit

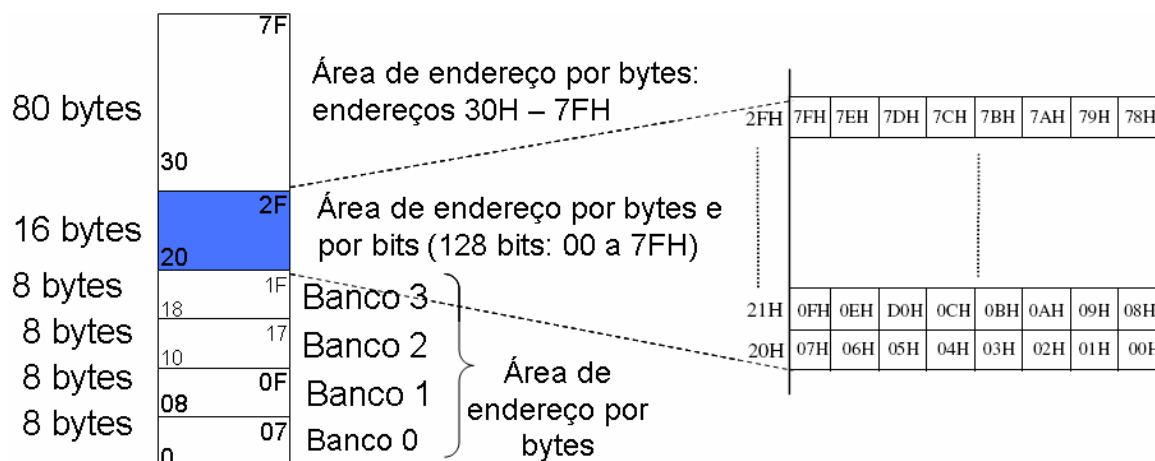


Figura 2.1 Área da RAM interna (DRAM) endereçável por bit

Os registradores de funções especiais (SFR) cujos endereços terminam em 0 ou 8h são também endereçáveis por bits, com os endereços 80h a FFh. Na figura 2.2 se mostra na área de Registradores de Funções Especiais (SFR) endereçável por bit.

8 Bytes							
F8							FF
F0	B						F7
E8							EF
E0	ACC						E7
D8							DF
D0	PSW*						D7
C8	T2CON**	T2MOD*	RCAP2L*	RCAP2H*	TL2*	TH2*	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			87
							PCON*

↑
Bit Addressable

Figura 2.2 Área dos Registradores de Funções Especiais (SFR) endereçável por bit

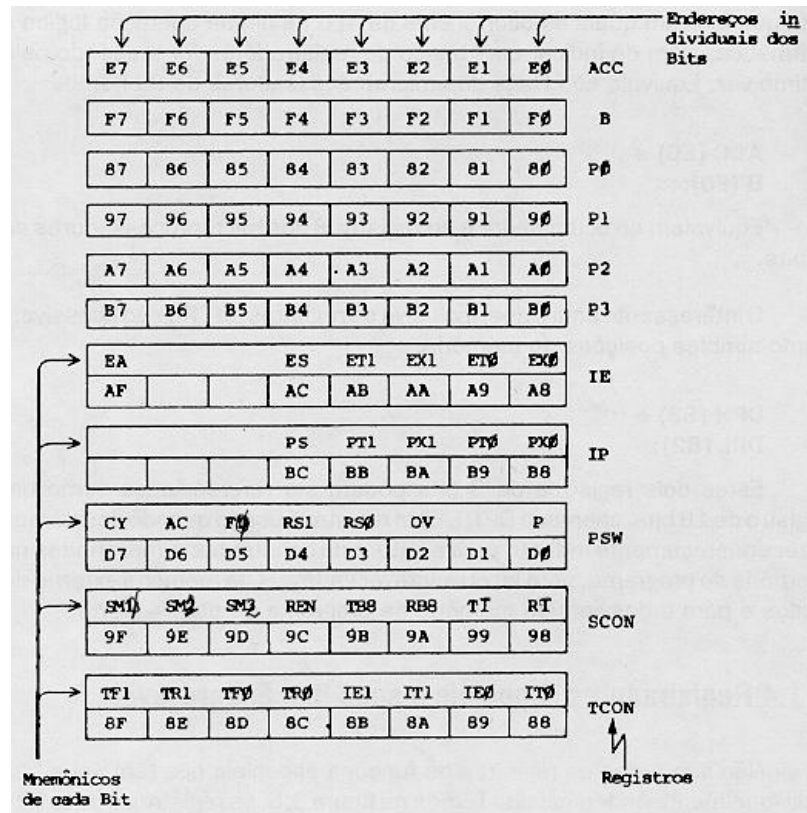


Figura 2.3 Endereços de cada bit da Área SFR

2.2 Repertório de instruções

O repertório de instruções no microprocessador 8051 pode dividir-se em:

- Instruções de transferência de dados
- Instruções aritméticas
- Instruções lógicas
- Instruções de salto
- Instruções booleanas ou de manuseio de bits.

Nas tabelas aparece o término <byte> que deve entender-se de diferente forma segundo o tipo de endereçamento empregado:

<byte> = Endereço de memória se endereçamento é direto.
 @R0 ó @r1 se endereçamento é indireto.
 R0, ... , R7 se endereçamento é por registrador.
 #dato8: dados de 8 bits se endereçamento é imediato.

Também:

Ri = Registradores R0 a R7

2.2.1 Instruções de transferência de dados

Nemónico	Operação	Dir	Ind	Reg	Inm
MOV A, <byte>	A = <byte>	X	X	X	X
MOV <byte>, A	<byte> = A	X	X	X	
MOV <byte>, <byte>	<byte> = <byte>	X	X	X	X
MOV DPTR, dato16	DPTR = dato16				X
PUSH <byte>	pilha <= <byte>	X			
POP <byte>	<byte> <= pila	X			
XCH A, <byte>	Intercambiar A com <byte>	X	X	X	
XCHD A, @Ri	Intercambiar <i>nibbles</i> baixos de A e @Ri		X		
MOVX A, @Ri	Ler em memória de dados externa		X		
MOVX @Ri, A	Escrever em memória de dados externa		X		
MOVX A, @DPTR	Ler em memória de dados externa		X		
MOVX @DPTR, A	Escrever em memória de dados externa		X		
MOVC A, @A + DPTR	Ler em memória de programa		X		
MOVC A, @A + PC	Ler em memória de programa		X		

2.2.2 Instruções aritméticas

Nemónico	Operação	Dir	Ind	Reg	Inm	C	OV	AC
ADD A, <byte>	$A = A + \text{<byte>}$	X	X	X	X	X	X	X
ADDC A, <byte>	$A = A + \text{<byte>} + C$	X	X	X	X	X	X	X
INC A	$A = A + 1$							
INC <byte>	$\text{<byte>} = \text{<byte>} + 1$	X	X	X				
INC DPTR	$\text{DPTR} = \text{DPTR} + 1$							
DEC A	$A = A - 1$							
DEC <byte>	$\text{<byte>} = \text{<byte>} - 1$	X	X	X				
MUL AB	$B:A = B \times A$					0	X	
DIV AB	$A = \text{int}(A/B) \quad B = \text{mod}(A/B)$					0	X	
DA A	Ajuste decimal					X		

2.2.3 Instruções lógicas

Nemónico	Operação	Dir	Ind	Reg	Inm	C	OV	AC
ANL A, <byte>	$A = A \text{ and } \text{<byte>}$	X	X	X	X	X	X	X
ANL <byte>, A	$\text{<byte>} = \text{<byte>} \text{ and } A$	X						
ANL<byte>,#dat8	$\text{<byte>} = \text{<byte>} \text{ and } \text{dad8}$	X						
ORL A, <byte>	$A = A \text{ or } \text{<byte>}$	X	X	X	X	X	X	X
ORL <byte>, A	$\text{<byte>} = \text{<byte>} \text{ or } A$	X						
ORL<byte>,dato8	$\text{<byte>} = \text{<byte>} \text{ or } \text{dad8}$	X						
XRL A, <byte>	$A = A \text{ xor } \text{<byte>}$	X	X	X	X	X	X	X
XRL <byte>, A	$\text{<byte>} = \text{<byte>} \text{ xor } A$	X						
XRL<byte>,dato8	$\text{<byte>} = \text{<byte>} \text{ xor } \text{dad8}$	X						
CLR A	$A = 0$							
CPL A	$A = \text{not } A$							
RL A	Rodar A 1 bit izq.							
RLC A	Rodar A e C 1 bit izq.							
RR A	Rodar A 1 bit der.							
RRC A	Rodar A e C 1 bit der.							
SWAP A	Intercamb. nibbles de A							

2.2.4 Instruções de salto

Nemónico	Operação	Observações
JMP label	Saltar à instrução marcada com label: (PC <= label)	SJMP rel: salto curto LJMP end16: salto longo AJMP end11: salto absoluto
JMP @A + DPTR	Saltar a @A + DPTR	
JZ label	Saltar se A = 0	É um salto curto condicional
JNZ label	Saltar se A ≠ 0	É um salto curto condicional
DJNZ <byte>, label	Decrementar <byte> e saltar se não ser zero	É um salto curto condicional Endereçamento direto ou por registrador
CJNE A, <byte>, label	Saltar se A ≠ 0	É um salto curto condicional. Endereçamento direto ou imediato.
CJNE <byte>, #dat8, rel	Saltar se <byte> ≠ #dat8	É um salto curto condicional. Endereçamento indireto ou por registrador. Se afeta C.
CALL label	pilha <= PC PC <= label	LCALL dir16: chamada longa ACALL dir11: chamada absoluta
RET	Retornar de sub-rotina	
RETI	Retornar de sub-rotina de interrupção	
NOP	Não operação	

2.2.5 Instruções booleanas ou de Manuseio de Bits

Estas instruções utilizam endereçamento direto dentro do espaço da memória de dados interna endereçável por bits. Na tabela, bit é a direção do bit.

Nemónico	Operação	C	OV	AC
MOV C, bit	C = bit	X		
MOV bit, C	bit = C			
ANL C, bit	C = C and bit	X		
ANL C, /bit	C = C and not bit	X		
ORL C, bit	C = C or bit	X		
ORL C, /bit	C = C or not bit	X		
CLR C	C = 0	0		
CLR bit	bit = 0			
SETB C	C = 1	1		
SETB bit	bit = 1			

Nemónico	Operação	C	OV	AC
SETB bit	bit = 1			
CPL C	C = not C	X		
CPL bit	bit = not bit			
JC label	Salta curto se C = 1			
JNC label	Salto curto se C = 0			
JB bit, label	Salto curto se bit = 1			
JNB bit, label	Salto curto se bit = 0			
JBC bit, label	Se bit = 1, salta e faz bit = 0			

2.3 Ferramentas para o trabalho com Microprocessadores

Um microprocessador por seu só é inútil a menos que se programe y se use adequadamente em um circuito eletrônico. As ferramentas para o trabalho com microprocessadores se classificam em: (a) ferramentas de hardware e (b) ferramentas de software.

Ferramentas de hardware mínimas

Para desenvolver um projeto com microprocesadores ou microcontroladores, usualmente se necessitam as ferramentas de hardware seguintes:

- Ordenador pessoal (PC) de mesa ou portátil
- Dispositivo programador de microprocessadores ou microcontroladores
- Placa de provas (*breadboard*) ou algo similar
- Chips de microprocessadores ou microcontroladores e seus componentes de suporte.
- Fonte de alimentação.

Ferramentas de software requeridas

Os microcontroladores necessitam de um programa ou software, para seu funcionamento. Este software se desenvolve e se prova pelo especialista (ou usuário). Durante a fase de desenvolvimento de um projeto com microcontroladores se requerem as ferramentas de software seguintes:

- Editor de texto
- Compilador: para programas feitos em uma linguagem de alto nível, por exemplo, C, e *assembler* para programas realizados em linguagem *assembler* ou de baixo nível. Na verdade, o compilador faz a tradução da linguagem de alto nível

para linguagem de baixo nível já que o próprio compilador tem integrado seu próprio programa *assembler*.

- Software do dispositivo programador de microcontroladores

Na figura 2.4 se mostra o diagrama de fluxo para usar as ferramentas de software com qualquer microprocessador e em particular com os microprocessadores da Família 8051.

- **EDITOR DE TEXTO.** Utiliza-se para criar o arquivo *.ASM com o programa fonte em linguagem *assembler*. Serve qualquer editor de textos em modo de não documento.

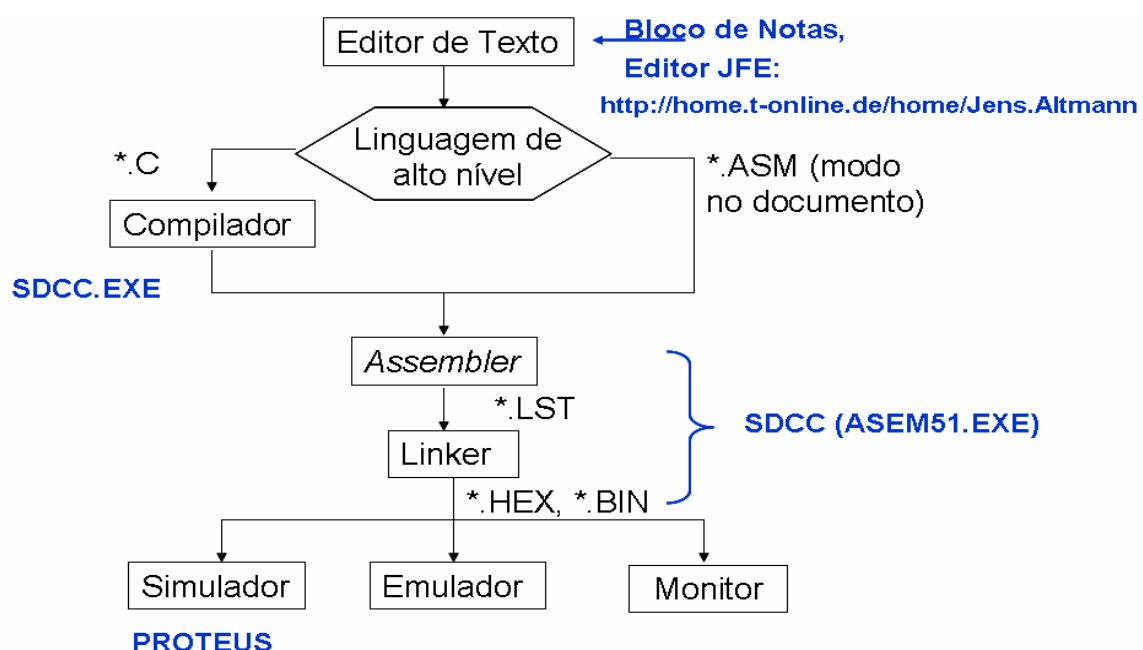


Figura 2.4 Diagrama de fluxo com as ferramentas de software usadas para o trabalho com o microprocessador/microcontrolador 8051.

- **ASSEMBLER** (ASEM51.EXE). A partir do programa em linguagem *assembler* (arquivo *.ASM), ele constrói um arquivo codificado objeto (arquivo *.OBJ), assim como uma lista da codificação em um arquivo texto com extensão *.LST.
- **LINKER** (neste caso, integrado no ASEM51). A partir do arquivo objeto (arquivo *.OBJ), cria um arquivo executável *.HEX, que é o adequado para enviar ao cartão de desenvolvimento (kit de treinamento) com o 8051.
- **SIMULADOR (PROTEUS ou JSIM do pacote SDCC).** Trata-se de um programa no computador pessoal que permite simular o comportamento de um 8051. Dispõe de comandos que facilitam a depuração de programas. Utiliza arquivos executáveis *.HEX como entrada. Existem simuladores que aceitam

arquivos com outras extensões (por exemplo, *.COFF nos PIC) que são mais adequados para simular programas escritos em linguagens de alto nível (por exemplo, C) pois fornecem informações sobre símbolos. O arquivo *.HEX nesse sentido está limitado mas é mais usado na maioria dos simuladores, entre eles o SDCC.

- **MONITOR.** É uma ferramenta de depuração e posta a ponto quando se dispõe de uma placa de desenvolvimento com um 8051. Consta de duas partes: uma reside no computador pessoal e a outra na memória EPROM da placa referida. A placa de desenvolvimento e o computador pessoal se comunicam via serial (mais usada) ou paralela. É possível, mediante os comandos apropriados, transferir o programa do usuário, no formato *.HEX, à placa e estando ali, executar os programas de forma local ou desde o computador pessoal.

2.4 Programação em linguagem *assembler*

Começaremos estudando como desenvolver e testar programas em linguagem *assembler*, cujo diagrama de fluxo se mostra na figura 2.5.

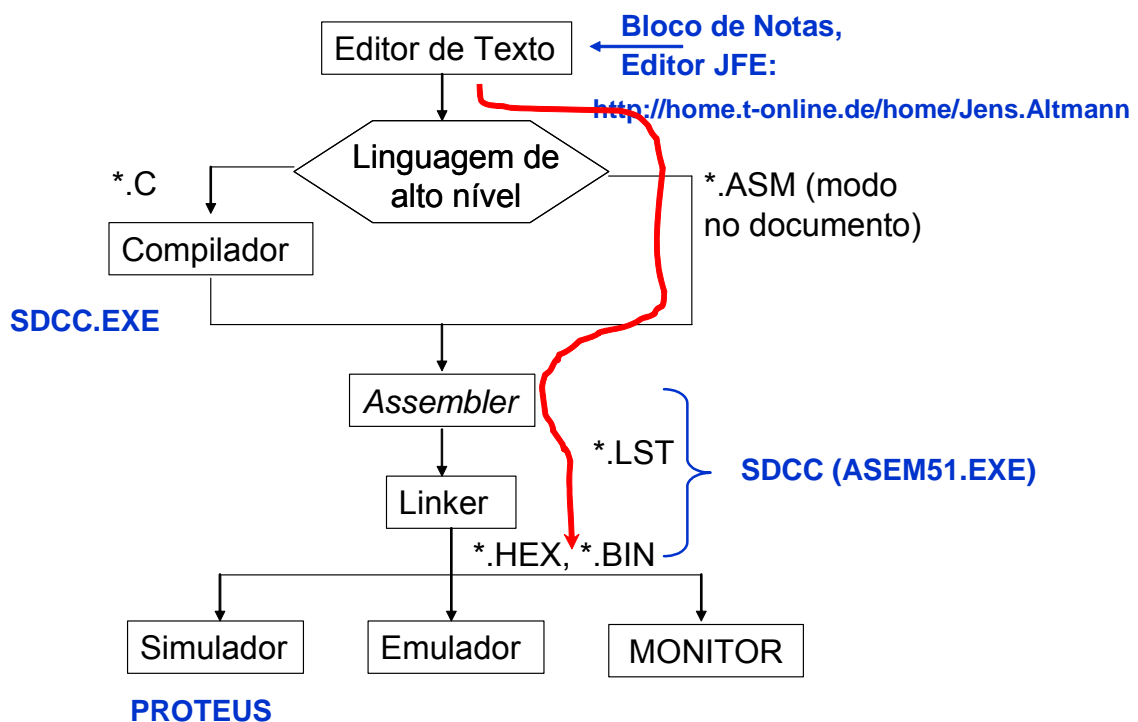


Figura 2.5 Diagrama de fluxo para o trabalho em linguagem *assembler* com o microprocessador/microcontrolador 8051.

O programa *assembler* converte um arquivo de origem em linguagem assembly (criado com um editor de textos) a um arquivo objeto em linguagem de

máquina. Este processo de tradução é feito em dois passos de tradução sobre o arquivo origem. Durante o primeiro passo, o assembler constrói uma tabela de símbolos dos símbolos e rótulos usados no arquivo origem. É durante o segundo passo o *assembler* traduz o arquivo origem no arquivo objeto de linguagem de máquina. Também é durante o segundo passo é gerado um arquivo listado com os códigos traduzidos.

Um programa em linguagem *assembler* é uma seqüência de linhas de texto, cada uma das quais pode ser:

- uma instrução do repertório do 8051,
- uma diretiva para o *linker*,
- uma macro instrução,
- uma opção do *linker* (indicada com o símbolo \$ colocado na primeira coluna da linha),
- uma linha de comentário (indicado com “ponto e vírgula “;”).

O estilo de escrever as instruções e as diretivas em linguagem assembler esta baseada em distribuir as seqüências de texto seguindo a seguinte estrutura:

[etiqueta] operação [operandos] [; comentário]

As etiquetas devem começar na primeira coluna.

A operação da instrução (chamado também como nemo-técnico ou nemónico) não pode começar na primeira coluna.

Nos operandos podem aparecer expressões formadas por símbolos e constantes enlaçados por operadores.

2.4.1 Símbolos e etiquetas

Os símbolos são representações alfanuméricas de constantes numéricas, endereços, macros, etc. Os caracteres permitidos para símbolos são: letras maiúsculas e minúsculas, números decimais e os caracteres especiais: sinal de interrogação (?) e sublinha (_). Para que o *assembler* possa distinguir entre um símbolo e um número, todos os símbolos têm que começar com uma letra ou os caracteres especiais: ? ou _. O seguinte é exemplos de símbolos legais:

PI

Tabela_Multip

LOC_4096

?_?_?

São significativos os primeiros 32 caracteres. Não se podem repetir dois símbolos no mesmo programa. Durante processo de tradução, o assembler converte todas as letras para maiúscula, e portanto, o assembler não faz nenhuma distinção entre letras maiúsculas e minúsculas. Por exemplo, os dois símbolos seguintes são os mesmos para o programa assembler:

Porta_Serial

PORTA_SERIAL

Existem alguns símbolos que não podem ser usados pois são reservados como diretivas, códigos de instruções (nemotécnicos), símbolos de operandos implícitos (por exemplo, os nomes de registradores A, B, DPTR, PC, R0...), operadores em tempo de compilação (EQ, GE, MOD, ver operadores em epígrafe 2.4.5). A diretiva EQU permite definir os símbolos quando se trabalha em linguagem *assembler*, como se estudará posteriormente no epígrafe das diretivas.

Exemplos de símbolos não permitidos:

1ST_VARIABLE ; Os símbolos não pode começar com um numero

ALPHA# ; Caracter não permitido "#"

MOV ; Instrução do 8051

LOW ; Operador da linguagem *assembler*

DATA ; Diretiva da linguagem *assembler*

As **etiquetas** ou **rótulos** são casos especiais de símbolos. “As etiquetas são usadas diante das sentencias (instruções ou diretivas) em um programa e devem terminar com o caracter “:”. Durante compilação do programa a cada etiqueta é atribuído um endereço, segundo a posição que ocupe dentro do programa. A seguir alguns exemplos de etiquetas validas:

TABELA_DE_CONSTANTES:

DB 0,1,2,3,4,5 (Diretiva de armazenamento de dados)

MINSAGEM: DB 'HELP'

VARIABLES: DS 10 (Reserva de dados)

BIT_VARIABLES: DBIT 16 (Reserva de dados)

START: MOV A, #23 (Instrução em linguagem *assembler*)

2.4.2 Constantes numéricas

Podem ser:

- Decimais (d, D): Ej: 10, 10d, 10D.
- Hexadecimais (\$, h, H): Ej: \$0A, 0Ah, 0AH.
- Octais (@, q, Q): Ej: @12, 12q, 12Q.
- Binárias (% , b, B): Ej: %1100, 1100b, 1100B.

2.4.3 Operadores em linguagem *assembler*

Operador	Uso	Resultado
+	x+y	x+y
-	x-y	x-y
*	x*y	x*y (sem sinal)
/	x/y	x/y (sem sinal)
MOD	x MOD y	Resto da divisão inteira sem sinal
SHL	x SHL y	x deslocado y lugares à esquerda
SHR	x SHR y	x deslocado y lugares à direita
HIGH	HIGH x	Byte mais significativo de x
LOW	LOW x	Byte menos significativo de x
.	byte.2	Bit 2 do byte
NOT	NOT x	Complemento 1 de x
AND	x AND y	Operação AND entre x e y
OR	x OR y	Operação OR entre x e y
XOR	XOR	Operação XOR entre x e y
LT	x LT y	-1 se x > e, se não 0
LE	x LE y	-1 se x <= y, se não 0
EQ	x EQ y	-1 se x = y, se não 0
NE	x NE y	-1 se x != y, se não 0
GE	x GE y	-1 se x >= y, se não 0
GT	x GT y	-1 se x > y, se não 0

Nota: -1 = FFFF (todo em 1)

2.4.4 Símbolos predefinidos

Os nomes dos SFR são símbolos predefinidos que ao usar-se, representam o endereço do SFR correspondente. Por exemplo, **P0** equivale ao endereço 80h.

Os bits de alguns SFR endereçados por bits têm nomes predefinidos, os quais podem utilizar-se nas instruções que endereçam bits. Por exemplo, **CY** equivale a **PSW.7** e é o bit cujo endereço é 0D7h.

O símbolo **#** se usa para indicar endereçamento imediato e **@** para o indireto.

Existe um arquivo na biblioteca do assembler chamado MOD8051.LIB onde é definido cada um dos símbolos usados para o microprocessador 8051. Isto simplifica o trabalho do programador, pois só tem que incluir esse arquivo no *header* de qualquer programa desenvolvido e desta forma pode usar símbolos em lugar de

endereços, que não são fáceis de lembrar pelo usuário. No SDCC a diretiva em *assembler* \$MOD51 inclui ao arquivo chamado MOD51 cujo conteúdo (parcial) se mostra a seguir:

```
; REV. 1.0 MAY 23, 1984

P0 DATA 080H ;PORT 0

SP DATA 081H ;STACK POINTER

DPL DATA 082H ;DATA POINTER - LOW BYTE

DPH DATA 083H ;DATA POINTER - HIGH BYTE

PCON DATA 087H ;POWER CONTROL

TCON DATA 088H ;TIMER CONTROL

TMOD DATA 089H ;TIMER MODE

TL0 DATA 08AH ;TIMER 0 - LOW BYTE

TL1 DATA 08BH ;TIMER 1 - LOW BYTE

TH0 DATA 08CH ;TIMER 0 - HIGH BYTE

TH1 DATA 08DH ;TIMER 1 - HIGH BYTE

P1 DATA 090H ;PORT 1

SCON DATA 098H ;SERIAL PORT CONTROL

SBUF DATA 099H ;SERIAL PORT BUFFER

P2 DATA 0A0H ;PORT 2

IE DATA 0A8H ;INTERRUPT ENABLE

P3 DATA 0B0H ;PORT 3

IP DATA 0B8H ;INTERRUPT PRIORITY

PSW DATA 0D0H ;PROGRAM STATUS WORD

ACC DATA 0E0H ;ACCUMULATOR

B DATA 0F0H ;MULTIPLICATION REGISTER

IT0 BIT 088H ;TCON.0 - EXT. INTERRUPT 0 TYPE

IE0 BIT 089H ;TCON.1 - EXT. INTERRUPT 0 EDGE FLAG
```

2.4.5 Diretivas

As diretivas são comandos do programa *assembler* para indicar seu modo de atuação durante o processo de tradução. Estas são usadas para definir símbolos,

reservar espaço de memória, armazenar valores na memória de programa, configurar o contador de localizações, identificarem o fim de arquivos, entre outras. Só é permitida uma diretiva por linha, porém podem ser incluídos comentários. A parte restante deste epígrafe detalha as diretivas mais usadas.

2.4.5.1 Diretivas de definição de símbolos

- **EQU: Definir símbolo (só uma vez)**

Atribui um valor a um símbolo

símbolo EQU expressão

Exemplos:

TEN EQU 10 ;Símbolo atribuído a um número

CONTADOR EQU R7 ;Símbolo definido pelo usuário para o símbolo do
;operando implícito R7. CONTADOR pode agora
;ser usado no mesmo lugar que corresponde a R7
;Por exemplo a instrução
;INC R7 pode ser escrita como INC COUNTER.

ASCII_D EQU 'D' ;Símbolo atribuído a uma constante ou literal ASCII

- **SET: Definir ou redefinir símbolo**

Atribui o re-atribui um valor a um símbolo.

Símbolo SET expressão

Esta diretiva é similar a EQU, com a diferença de que na EQU o símbolo só pode ser definido uma vez, enquanto com SET, o símbolo pode ser redefinido sem limites.

Exemplos:

PONTEIRO SET R0 ;Símbolo atribuído ao registrador 0

PONTEIRO SET R1 ;POINTER redefinido para o registrador 1

CONTADOR SET 1 ;Símbolo iniciado a 1

CONTADOR SET CONTADOR +1 ;Incrementando o símbolo

- **BIT: Definir bit da RAM interna**

Atribui um endereço direto de bit da RAM interna a um símbolo. Se o endereço estiver entre 0 e 127 então corresponde a um bit da RAM interna. Se estiver entre 128 e 255 então corresponde a um bit da área SFR.

símbolo EQU expressão

Exemplos:

```
CF          BIT    0D7H      ;Bit da bandeira de Carry no PSW  
FLAG1       BIT    6         ;Endereço de memória de uma bandeira (flag)  
FLAG2       BIT    FLAG1+1   ;O próximo bit é outro flag
```

- **CODE: Define um símbolo para a memória de código.**

Símbolo CODE expressão, números o símbolos.

O valor numérico do endereço não pode ultrapassar 65535

Exemplo:

```
RESET CODE    0  
EXTIO CODE    RESET + (1024/16)
```

- **DATA: Define um símbolo para a RAM interna com endereçamento direto.**

Símbolo DATA expressão, números o símbolos.

Atribui um endereço direto na memória RAM interna a um símbolo. Endereços entre 0 e 127 correspondem à RAM interna de dados, e entre 128 e 255 correspondem aos SFR. O valor numérico do endereço não pode ultrapassar 255

Exemplo:

```
PSW DATA     0D0H      ;Definindo o endereço do PSW  
BUFFER DATA   32       ;Endereço da RAM interna  
ESP_LIVRE DATA BUFFER+16 ;Expressão aritmética
```

- **IDATA: Define um símbolo para a RAM interna com endereçamento indireto.**

Símbolo IDATA expressão, números o símbolos.

Atribui um endereço indireto na memória RAM interna a um símbolo. O valor numérico do endereço não pode ultrapassar 255

Exemplo:

```
TOKEN IDATA    60  
BYTE_CNT IDATA TOKEN + 1  
ADDR IDATA     TOKEN + 2
```

- **XDATA: Define um símbolo para a RAM externa.**

Símbolo XDATA expressão, números o símbolos.

Atribui um endereço localizado na memória externa de dados (XRAM) a um símbolo. O valor numérico do endereço não pode ultrapassar 65535

Exemplo:

```
USER_BASE XDATA 2048  
HOST_BASE XDATA USER_BASE + 1000H
```

2.4.5.2 Diretivas de seleção de segmentos

- **CSEG, BSEG, DSEG, ISEG, XSEG: Definição de segmentos de memória**

Cada um dos espaços de memória no 8051 leva associado uma diretiva de definição de segmento de memória, como aparece a seguir:

CSEG: memória de programa

BSEG: memória de dados endereçável por bits

DSEG: memória de dados interna endereçável de forma direta

ISEG: memória de dados interna endereçável de forma indireta

XSEG: memória de dados externa

Cada segmento tem seu próprio contador de endereços que é zerado durante a inicialização do programa *Assembler*. Os conteúdos do contador de endereços podem ser alterados usando o comando AT depois de selecionar o segmento. O segmento assumido pelo assembler durante o assembly é o de memória de programa (diretiva CSEG). O valor depois do comando AT, deve ser um número, uma expressão aritmética, ou um símbolo definido.

Exemplos:

DSEG ;Selecionar o uso do contador atual de endereços para o segmento
; de dados com endereços diretos.

BSEG AT 32 ;Selecionar segmento de dados de bit e forçar o contador de
; de endereços ao valor 32 em decimal.

XSEG AT (USER_BASE * 5) MOD 16 ; Podem ser usadas expressões aritméticas para especificar uma localização específica.

2.4.5.3 Diretivas de armazenamento e de reserva de memória

- **DB: Definir byte**

Armazena um byte na memória de programa.

[etiqueta] DB dado8 ; dado8 pode ser: 17h, “uma cadeia”, ‘outra cadeia’

Exemplos:

COPYRGHT_MSG: DB '(c) Copyright, 1984' ;ASCII Literal

CONSTANTES:

DB 127,13,54,0,99 ;Tabela de constantes

DB 17,32,239,163,49 ;a etiqueta é opcional

MIXED: DB 2*8,'MPG',2*16,'abc' ;podem ser misturados constantes e nao.

- **DW: Definir Word (Palavra de 16 bis)**

Armazena uma constante de 16 bits na memória de programa.

[etiqueta] DW dado16 ;dado16 pode ser 17h, 1234h

Exemplos:

TABELA_SALTOS: DW RESET,START,END ;Tabela de endereços

RADIX: DW 'H',1000H

;o primeiro byte contem 0, segundo byte contem 48H (ASCII da letra H)

;o terceiro byte contem 10H e o quarto byte contem 0

- **DS: definir espaço**

[etiqueta] DS número

- Reserva o número de bytes indicado. O contador de localizações avança esse número.

Exemplos:

DSEG ;Selecionar o segmento de dados

DS 32 ; Reservar 32 bytes, a etiqueta é opcional

- **DBIT: definir espaço na area de bit**

[etiqueta] DBIT número

- Reserva o número de bits indicado dentro do segmento de bits (segmento BIT) . O contador de localizações avança esse número. Se deve ter cuidado com que o contador de localizações não ultrapasse o limite do segmento.

Exemplo:

```
BSEG          ;Selecionar o segmento de bit
DBIT    16    ;a etiqueta é opcional
BUFF_ES:    DBIT    32    ;Reservar um buffer de bits para E/S
```

2.4.5.4 Outras Diretivas

- **ORG: Iniciar contador de localizações ou contador de endereços.**

ORG expressão

Põe no contador de localizações no valor da expressão.

Exemplo:

```
ORG 10          ; A próxima instrução se localiza no endereço 10
```

- **END: terminar programa.**

END

- **USING: cria um segmento para um dos bancos de registros.**

Especifica qual banco de registradores vai ser usado quando se deseja gerar endereços diretos usando símbolos de registros. Esta diretiva simplifica o endereçamento direto de um banco de registradores especificados. Por exemplo, o registro R0 do banco 1 geraria o endereço direto 08H.

```
USING numero ; numero pode ser 0,1, 2 ou 3
```

Exemplo:

```
USING    0    ; endereços diretos são gerados para o banco 0
```

```
USING 1+1+1    ; endereços gerados para o banco 3
```

- **INCLUDE: inserir conteúdo de um arquivo.**

```
INCLUDE filename    ; filename no diretório atual
```

```
INCLUDE <filename>    ; filename no path
```

```
INCLUDE "filename"    ; filename no diretório atual
```

2.4.5.5 Comandos de controle do assembler

Todos os comandos de controle do *assembler* começam com o sinal dólar (\$) seguido de um comando de controle. Não pode existir espaço entre o sinal de dólar e o comando. Só é permitido um controle por linha, embora é permitido comentários nessa mesma linha.

Existem vários tipos de controles. Nesta disciplina o maior interesse é o seguinte comando:

`$MOD51`

Ele reconhece os símbolos dos registradores SFR predefinidos no programa fonte. Isto faz que o usuário não tenha que definir esses símbolos porque já estão definidos no arquivo MOD especificado. O comando `$NOMOD` desabilita esse comando e portanto a função de reconhecimento. O programa *assembler* só busca o arquivo especificado no comando `$MOD` no drive assumido.

Exemplo:

`$MOD51 ; reconhece símbolos para os microprocessadores seguintes: 8051,
; 8751, 8031, 80C51, 80C31, 87C51, 9761, 8053`

- **EXTERN: declarar símbolos externos.**

`EXTERN símbolo`

Declara um símbolo que será importado desde outro módulo, no qual foi definido como PUBLIC ou GLOBAL.

- **PUBLIC: declarar símbolos públicos.**

`PUBLIC símbolo [, símbolo ...]`

Declara um símbolo como público, de modo que pode ser usado em outro módulo (onde aparece como externo).

- **GLOBAL: declarar símbolos públicos ou externos.**

`GLOBAL símbolo [, símbolo ...]`

Declara um símbolo como público ou externo, segundo o caso.

Exemplo: Somar os conteúdos das localizações X e Y da RAM interna, o resultado depositá-lo na localização S1 da RAM interna e S2 da RAM externa.

Solução:

```
X EQU 40h
Y EQU 41h
S1 EQU 42h
S2 EQU 1000h
SOMA:
    MOV A, X
    ADD A, Y
    MOV S1, A
    MOV DPTR, #S2
    MOVX @DPTR, A
    END
```

Este programa se localiza no endereço 0 da memória ROM.

O arquivo .ASM:

; Programa que soma dois números de 8 bits armazenados na memória

;Rótulos operação [operando] comentários

```
X EQU 40h ;Primeiro número = Endereço 40 H da RAM interna
Y EQU 41h ;Segundo número = Endereço 41 H da RAM interna
S1 EQU 42h ;Resultado na RAM interna
S2 EQU 1000h ;Resultado na RAM externa
    ORG 0 ;Programa inicia em endereço 0000 H
```

```
SOMA: MOV A, X ;Trazer primeiro operando
      ADD A, Y ;Trazer segundo operando
      MOV S1,A ;Armazenar soma na RAM interna
      MOV DPTR, #S2 ;Aponta para endereço na RAM externa
      MOVX @DPTR, A ;Escrever no endereço 8000H em XRAM
      END
```

O arquivo .LST (Observar em negrito os códigos binários de cada instrução):

ASEM-51 V1.3 Copyright (c) 2002 by W.W. Heinz
PAGE 1

MCS-51 Family Macro Assembler A S E M - 5 1 V 1.3

=====

```
Source File: Ejemplo1UEA.ASM
Object File: Ejemplo1UEA.hex
List File: Ejemplo1UEA.lst
```

Line	I	Addr	Code	Source
1:				; Programa que soma dois números de 8 bits armazenados na memória
2:				
3:				;Rótulos operação [operando] comentários
4:				
5:		N	0040	X EQU 40h ;Primeiro número = Endereço 40
6:		N	0041	Y EQU 41h ;Segundo número = Endereço 41
7:		N	0042	S1 EQU 42h ;Resultado na RAM interna
8:		N	1000	S2 EQU 1000h ;Resultado na RAM externa
9:				
10:		N	0000	ORG 0 ;Programa inicia em endereço
11:				
12:				
13:				
14:				
15:				
16:				
17:				
18:				

0000H

11:

12: 0000 **E5 40** SUMA: MOV A, X ;Trazer primeiro operando

13: 0002 **25 41** ADD A, Y ;Trazer segundo operando

14: 0004 **F5 42** MOV S1,A ;Armazenar soma na RAM

interna

15: 0006 **90 10 00** MOV DPTR, #S2 ;Aponta para endereço na

RAM externa

16: 0009 **F0** MOVX @DPTR, A ;Escrever no endereço

8000H en XRAM

17:

18: END

register banks used: ---

no errors

O arquivo .HEX:

```
:0A000000E5402541F542901000F0A4
:00000001FF
```

2.5 Exemplos de programas em linguagem assembler

A seguir se mostram programas de exemplos desenvolvidos por nos e outros que tem sido tomados dos exemplos do autor Denys E C Nicolosi.

Exercício 1

```
;PROGR0: TREINO DE UTILIZACAO DO INICIALIZADOR DE ENDEREÇOS ORG (ORiGin)
```

```
;LABEL INSTR OPERANDO ; COMENTARIOS
```

```
.*****
;

```

```
ORG 0000h ;começa o programa na linha 0000h na EPROM
```

```

mov    A,#0FFh    ;escrevo '1111 1111'b no acumulador A (ACC)

org    000Fh        ;começa o programa na linha 000F na EPROM

mov    A,#00h ;escrevo '0000 0000'b no ACC

END

```

; OBS: Este programa no simulador irá ter instruções NOP (NO OPERATION)

; entre os endereços 0000h e 000Fh pois não existe elo entre 0000h e 000fh

; no programa aqui realizado.

; Para verificar no simulador o segundo ORG operando, basta ir no menu

; superior intitulado "Cpu" optar por "Animate" e pressionar a tecla 0

; para executar seu programa passo-a-passo ou tecla 1 para executá-lo

; automaticamente até o PC (*program counter*) chegar no

; valor 000Fh, que assim o programa começará a rodar neste

; endereço.

Exercício 2

; PROGR1 : TREINO DOS VÁRIOS MODOS DE ENDEREÇO DIRETO

; LABEL INSTR OPERANDO ; COMENTÁRIOS

```

.*****
;

org    0        ;começa o programa na linha 0000h na EPROM

mov    A,#0h    ;escrevo '0000 0000'b no acumulador A (ACC)

mov    A,#0ffh  ;escrevo '1111 1111'b no ACC

mov    B,#0fh   ;escrevo '0000 1111'b no registrador B

mov    R0,#0f0h ;escrevo '1111 0000'b no registrador R0

mov    A,#15    ;escrevo '0000 1111'b ou 0fh no ACC,por
                ; decimal

mov    A,#10101010b ;escrevo '1010 1010'b ou AAh no ACC, por
                ; binário

mov    A,#HIGH(0fh) ;escrevo a( parte alta de 0F) em A

mov    B,#LOW(0fh) ; " (a parte baixa de 0F) em B

mov    A,#HIGH(65535);" (a parte alta de 65535)=FFh em A

```

```

mov    B,#LOW(65535) ;" (a parte baixa de 65535)=FFh em B
mov    A,#(255-250) ;" a diferença da conta 255-250 = 5 =05h
        ; em A
mov    A,#HIGH(255-240) ;" a parte alta da conta (15=0Fh),
        ;que e'00h em A
mov    B,#low(255-240) ;" a parte baixa da conta (15=0Fh),
        ;que e'0Fh em B
mov    A,#'A'        ;escrevo em A o "código ASCII" da letra A,
        ;que 'e 65 em decimal, ou 41h(em hexa).
mov    A,#'B'        ;escrevo em A o "código ASCII" da letra B,
        ; que 'e 66 em decimal ou 42h.

```

END

;OBS1 : 65535 utilizado e' equivalente a FFFFh (16 bits).

;OBS2 : Quando o numero e' hexa na instrução e este começar com
;os índices A,B,.F, voce deve escrever o numero hexa com zero na
; frente, p.ex. :f3h = 0f3h , senao o compilador nao aceitara' o
;comando gerando mensagem de erro a ser visto no arquivo ".prn",
;que neste caso e'o arquivo "progr1.prn" .Altere o segundo "mov"
;deste programa para nao ter o "0" na frente do numero e compile
;e veja o que acontecel!.

Exercício 3

```

; Progr2 :      INTRODUZ O RECURSO DE "LABEL"
;label instr  operando      ;comentarios
.***** ***** *****
;
;
        org 0          ;começa em 0
        mov    A,#00h ;escrevi o valor 00h no acumulador (=A)
volta: inc    A      ;incremento A ,por 1
        mov    P1,A  ;movo o valor de A para P1
        sjmp   volta
END

```

;O recurso de Label e' muito importante pois voce nao precisa contar
 ;cada instrucao executada para saber para onde ,p.ex., o "JUMP" da ul-
 ;tima instrucao, neste ex., em endereco absoluto, o mesmo vai retornar.
 ;Ao inves disso, eu chamo a posicao de volta desejada por um nome (label)
 ;e utilizo este label dentro da instrucao, e o proprio compilador ira'
 ;fazer as contas do endereco absoluto para nos!

Exercício 4

; PROGR3 : TREINO DE ALTERAR OS BANCOS B0,B1,B2 E B3
 ; DOS REGISTRADORES R0,..R7;
 ; UTILIZAREMOS O REG. "PSW" PARA FAZER A ALTERACAO DOS BANCOS:
 ; O registrador PSW foi criado para ajudar a dar informações sobre
 ; o conteúdo de A(acumulador).Ele indica ,sobre o resultado da ultima
 ; operação realizada no Acumulador.Ele indica :
 ; 1- Se houve "Carry" na operacao (Bit CY ou simplesmente C)
 ; 2- Se houve "Auxiliary Carry" na operacao (Bit AC)
 ; 3- Se houve "Paridade par" na operacao (Bit P)
 ; 4- Se houve "Over-flow" na operacao (Bit OV)
 ; O Bit "D1" nao tem funcao e o Bit" D5" e' um Flag auxiliar para
 ; uso geral.Os Bits RS1 e RS0 sao os que se muda de Banco de
 ; Registradores,como sera' visto:
 ; PSW(NOME) = CY AC F0 RS1 RS0 OV XX P (E'O NOME DE CADA BIT)
 ; PSW(END.) = D7 D6 D5 D4 D3 D2 D1 D0 (E' O ENDERECO EM HEXA)
 ; NESTE MOMENTO,DO PSW SO' NOS INTERESSA OS BITS "RS0 E RS1" :
 ; RS1 RS0 BANCO
 ; 0 0 0 =B0
 ; 0 1 1 =B1
 ; 1 0 2 =B2
 ; 1 1 3 =B3
 ; POSSO ACESSAR ESTES BITS PELO ENDERECO D3h e D2h

```

; (COM INSTRUCAO SETB,CLRB) OU PELO BYTE INTEIRO DO PSW
; (COM INSTRUCAO MOV PSW, "XX")
; NAO SE ESQUECA QUE NO RESET O PSW JA' AJUSTA OS BITS PARA O
; BANCO 0
; (B0),NA VERDADE O RESET ZERA TODO O PSW, I.E'. PSW = 00h
; LABEL INSTR  OPERANDO ; COMENTARIOS
.*****
;
mov    R0,#00h      ;movo para o R0 do banco B0 o valor 00h
                    ;note que pelo reset o banco ja' e' B0
mov    psw,#00001000b ;movo para o psw "0 e 1" para os bits
                    ;"RS1 e RS0", que e' o banco B1
mov    psw,#08h      ;mesma instrucao acima so' que com codi-
                    ;go em hexa para fazer RS1=0 e RS0=1 no
                    ;psw.
mov    R0,#01h      ;movo para o R0 do banco B1 o valor 01h
setb   rs1           ;faco diretamente o bit RS1 =1; como o
                    ;bit RS0 ja' era 1,teremos agora os
                    ;dois em 1, logo estou no banco B3
mov    R0,#03h      ;movo entao o valor 03h para o reg. R0
                    ;do B3
end
;  Verifique se realmente o dado foi para R0 do banco 0,depois se o segundo
;  dado foi para R0 do banco 1,etc.

```

Exercício 5

```

; Progr4 :  Introduz o recurso de "JUMP INCONDICIONAL" e suas variações
;
;  Os jumps (desviar,pular) são muito utilizados para se desviar um
;  fluxo de programa. Eles se dividem em :
;
;  - Condicionais . ( O desvio e' condicionado a algum teste prévio)

```



```

.*****
;
;zero EQU 00h ; e' o mesmo escrever "zero" ou 00
somador EQU 0E0h ; e' o endereço de A;e'o mesmo que A.
porta1 EQU p1 ;e' o mesmo que o port P1
    org 0 ;começa em 0
    mov somador,#zero ;escrevi o valor 00h no somador (=A)
volta: inc somador ;incremento A ,por 1
    mov porta1,somador ;movo o valor de A para P1
    sjmp volta ;retorno o software para a posicao com
           ;nome "volta".

END

```

;O recurso de"Equate" (igualar) e' muito importante pois você pode se utilizar
;de nomes fáceis de guardar ou relacionar, com posições de memória ou com
;valores numéricos. Este programa e' o mesmo que o intitulado "progr2" ,agora
;com nomes atribuídos a certas variáveis e posições de memória.Compare com o
;progr2 para notar as diferenças ,teste os dois e veja se apos a compilação
;existe alguma diferença de codificação na EPROM(pelo simulador) entre os dois.

Exercício 7

; Progr8 : Introduz o recurso do "JUMPS CONDICIONAIS"

; Os JUMPS CONDICIONAIS desviam o programa corrente para uma posição.
; definida,(que e' o mesmo que alterar o PC para a nova posição definida)
; se uma certa condição testada nesta mesma instrução seja obedecida.
; Por ex. ,se queremos testar o Bit de CARRY,conhecido como "C" ,e desviar
; para outra parte do programa se este Bit for "1", podemos utilizar a
; instrução: "JC ENDERECO", que testara' o bit de Carry e desviara' o programa
; para" ENDERECO" se ele for 1.

; Outras instruções importantíssimas do 8051 são:

; CJNE <byte> , <byte> , <ENDERECO> ,ex: cjne A,#FFh, Desvio
; Aqui a instrução testa se o Acumulador tem conteúdo diferente de FF.
; Se tiver, então desvia para o endereço aqui com nome simbólico "Desvio".

; Se o conteúdo for igual, ela vai para a próxima instrução que apareça
 ; apos a mesma.
 ; DJNZ <byte> , <ENDERECO> ,ex: djne R0, Desvio
 ; Aqui a instrução decrementa o conteúdo de R0, depois testa se este
 ; conteúdo e'zero; se não e' zero ela desvia o programa para o endereço
 ; simbólico "Desvio".Se for zero, ela vai para a próxima instrução que
 ; apareça apos a mesma.
 ; Existe extensa opção de instruções com Jumps Condicionais.
 ; Vide Set de Instruções em livro para se atualizar de todas elas.
 ; Exemplo :

```
;label instr  operando;comentários
.*****  *****  *****
,
      org 0      ;começa em 0
inicio: mov  A,#0  ;carrego inicialmente A com zero.
      mov  r0,#9  ;carrego R0 com 9
      djnz  r0,$   ;decremento R0 ,comparo com zero e se não for zero,
                      ;desvio para este mesmo ponto de endereço do software.
      mov  A,#0FFh ;se a contagem chegou ao fim carrego A com FFh.
      sjmp  inicio ; volto para o inicio.
      END
```

; Observe que eu gastei" tempo" entre fazer A = 0 e A = FF, que e' uma das
 ; funções desta instrução.Se R0 e' carregado com numero maior,gastarei
 ; mais tempo entre os dois estados, fazendo um" pisca-pisca".

Exercício 8

; Progr9 : Introduz o recurso do "CALL" e "RETURN"
 ; As rotinas CALL e RETURN se diferem dos JUMPS porque, p.ex., um CALL
 ; guarda (PUSH) o PC corrente na pilha antes de se desviar para o novo
 ; endereço.
 ; A rotina RETURN ,por sua vez, traz de volta (POP) o endereço guardado

; anteriormente na pilha. Logo você não precisa se preocupar em anotar onde
 ; estava para realizar a volta `a rotina principal.
 ; Temos as rotinas:
 ; ACALL, que permite trabalhar com endereços de ate' 11bits (2K de desvio).
 ; LCALL, que permite ate' 16bits de desvio.
 ; RET, que retorna da pilha o ultimo endereço salvado (gerado pelo CALL).
 ; RETI, que só se usa quando o desvio foi causado por interrupção, pois
 ; ela, alem de retornar ao endereço anterior, ela "reseta" a interrupção
 ; criada.(vide o "Progrm" de exercício de interrupções para entendimento.
 ; Exemplo :

```
;label instr operando;comentários
```

```
.*****  

,

```

```

    org 0          ;começa em 0
inicio: mov  A,#0FFh  ; carrega acc com ff(1111 1111)
          lcall compl  ; chama rotina de complementar o acumulador
          sjmp  inicio  ; volta para o inicio

;-----Sub-rotina de complementação de A-----
compl: cpl  A      ;complementa o valor de A
       ret
       END

```

; OBS: As sub-rotinas são importantíssimas quando se quer criar um soft-
 ; ware estruturado,i.e'. ,quando utilizo os recursos de uma rotina chamar
 ; outras,de modo encadeado.

Exercício 9

; PROGR10: INTRODUZ O RECURSO DE "SOMA E SUBTRACAO"
 ; Existem duas instruções de adição: a soma simples (ADD) e a soma que leva em
 ; consideração o conteúdo do "carry" (ADDC), que e o bit mais significativo do
 ; registrador PSW, indicado a seguir:
 ; Registrador PSW:
 ; Nome do bit = | CY | AC | F0 | RS1 | RS0 | OV | XX | P |

; Endereço em hexa = | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

; O bit de "carry" (C) indica se houve "estouro" (ou "vai-um") do bit 7 após a

; realização da soma

; Lembrar que no PSW existe também o Carry Auxiliar (AC), que indica se houve

; "vai-um" do nibble menos significativo para o mais significativo.

; LABEL INSTR OPERANDO COMENTARIO

.*****

```

ORG      0      ; começa em 0000
MOV      A, #0Fh ; move para A o valor 0Fh=15d
MOV      R0, #08 ; move para R0 o valor 8
ADD      A, R0   ; soma o conteúdo de R0 com A e guarda em A
                ; (A = 0Fh + 8h = 17h = 23d)

```

; Observe que o bit de "carry" permanece em 0, pois não houve "estouro" do bit

; 7. Já o "carry auxiliar" (AC) vai para 1, pois houve "vai-um" do nibble menos

; significativo para o mais significativo

; 0000 1111 = 0Fh = 15d

; + 0000 1000 = 08

; -----

; 0001 0111 = 17h = 23d

; LABEL INSTR OPERANDO COMENTARIO

.*****

```

ADD      A, #0F0h ; soma o valor 0F0h ao conteúdo de A e guarda em
A
                ; (A = 17h + F0h = 07)

```

; Neste exemplo, o bit de "carry" vai para um, pois existe "vai-um" do bit 7.

; O "carry auxiliar" (AC) vai para zero, pois não há "vai-um" do nibble menos

; significativo para o mais significativo

; 0001 0111 = 17h = 23d

; + 1111 0000 = F0h = 240d

; -----

; C->1 0000 0111 = 07

; Observe que, ao realizarmos a operação na base decimal, e
 ; considerando números
 ; variando de 0 a 255, o resultado excede 255 ($23+240=263$) e, por isso, o bit de
 ; "carry" vai para 1.

;LABEL INSTR OPERANDO COMENTARIO

.*****

ADDC A,#04 ; neste caso, deveria ser somado ao conteúdo de A
 (que

; contem 7) o valor 4, resultando 11. Observe, porem,

; que o resultado obtido foi $A = 0Ch = 12d$. Isto ocorre

; pois a instrução utilizada soma também o valor atual

; do "carry" ($C=1$).

; Portanto, $A = 07 + 04 + 01 = 0Ch = 12d$

; Utilizando a operação de soma (ADD) também e' possível realizar operações de

; subtração, se trabalharmos utilizando a notação de complemento de 2. Neste caso,

; considerando-se 8 bits, tem-se números na faixa de -128 a +127.

;LABEL INSTR OPERANDO COMENTARIO

.*****

MOV R0,#40h ; move para R0 o valor 40h=64d

MOV A,#23h ; move para o acumulador o valor 23h=35d

CPL A ; complementa A bit a bit

INC A ; incrementa em um o conteúdo de A

; Nestas duas etapas, obtivemos o complemento de 2 do

; conteúdo de A (ou seja, seu conteúdo com sinal invertido)

ADD A,R0 ; Finalmente, basta somar os números para que seja realizada

; a subtração ($A = 40h + (-23h) = 1Dh$ ou $A = 64d + (-35d) = 29d$)

; Iremos agora estudar a função do bit de "overflow" (OV), que indica se

; houve ou não "estouro" da operação. O entendimento de OV fica bastante simples

; quando a análise e' feita raciocinando em complemento de 2.

; se somarmos dois números positivos e o resultado for negativo, $OV=1$

; se somarmos dois números negativos e o resultado for positivo, $OV=1$

; nos demais casos OV=0

;LABEL INSTR OPERANDO COMENTARIO

.*****

MOV A,#00110001b ;move 00110001b=31h=49d para o acumulador

ADD A,#01110000b ;soma 01110000b=70h=112d ao conteúdo do acumulador e
; guarda o resultado no mesmo

;Neste caso, o resultado obtido e' 10100001b=A1h=161d. Admitindo números variando de

; 0 a 255d (8 bits), o resultado e' coerente. Porem, raciocinando em

; complemento de 2,

;somamos dois números positivos (pois MSB=0 para ambos) e obtivemos um numero negativo

; (MSB=1 no resultado). Portanto, OV=1.

;LABEL INSTR OPERANDO COMENTARIO

.*****

MOV A,#10110001b ;move 10110001b=B1h=177d para o acumulador
;em complemento de 2, move -79d para o acumulador

ADD A,#11110000b ;soma 11110000b=F0h=240d ao conteúdo
;do acumulador e guarda
;o resultado no mesmo
;em complemento de 2, soma -16d ao conteúdo
; do acumulador

; Neste caso, o resultado obtido e' 10100001b=A1h=161d. Admitindo números variando de

; 0 a 255d (8 bits), o resultado não e' coerente, pois 177d+240d=417d. Porem, observe

; que C=1, indicando que houve "vai-um" ou "estouro" do bit 7, como já havíamos
; visto anteriormente.

; Porem, raciocinando em complemento de 2, somamos dois números negativos

; (pois MSB=1 para ambos) e obtivemos um numero negativo (MSB=1 no resultado).

; Portanto, OV=0. Na base decimal (-79d)+(-16d)=(-95d), resultado totalmente

; coerente.

;LABEL INSTR OPERANDO COMENTARIO

.*****

MOVA,#10110001b ;move 10110001b=B1h=177d para o acumulador

;em complemento de 2, move -79d para o acumulador

ADD A,#10110000b ;soma 10110000b=B0h=176d ao conteúdo do acumulador e

; guarda o resultado no mesmo

;em complemento de 2, soma -80d ao conteúdo do

;acumulador

; Neste caso, o resultado obtido e' 01100001b=61h=97d. Admitindo números variando de

; 0 a 255d (8 bits), o resultado não e' coerente, pois $177d+176d=353d$. Porem, observe

; que C=1, indicando que houve "vai-um" ou "estouro" do bit 7, como já havíamos

; visto anteriormente.

; Porem, raciocinando em complemento de 2, somamos dois números negativos

; (pois MSB=1 para ambos) e obtivemos um numero positivo (MSB=0 no resultado).

; Portanto, OV=1.

; Na base decimal, deveríamos ter $(-79d)+(-80d)=(-159d)$, resultado que esta fora

; da faixa valida para algarismos com sinal, o que também revela a incoerência

; do resultado obtido.

; Verifica-se portanto que, juntos, os bits de "carry" (C) e "overflow" (OV)

; podem fornecer muitas informações sobre o status final da operação de adição.

; A operação de subtração também pode ser realizada utilizando-se a instrução.

; SUBB, que evita a necessidade de conversão dos números para complemento de 2.

;LABEL INSTR OPERANDO COMENTARIO

.*****

CLR C ;limpa o "carry"

MOV A,#0Ah ;move o valor 0Ah=10d para o acumulador

SUBB A,#02 ;subtrai o valor 2 mais o conteúdo do carry de A,
;deixando o resultado no acumulador.

; (A=0Ah-2-0=8) ou (A=10d-2-0=8)

; Verifique que, no caso da instrução SUBB, o valor do "carry" sempre será
; levado em consideração na operação, ou seja, será subtraído do conteúdo do
; acumulador juntamente com o operando da instrução. Veja o caso a seguir:

; LABEL INSTR OPERANDO COMENTARIO

.*****
;

SETB C ;faz C=1

MOV A,#0Ah ;move o valor 0Ah=10d para o acumulador

SUBB A,#02 ;subtrai o valor 2 mais o conteúdo do "carry" de A,
;deixando o resultado no acumulador.

; (A=0Ah-2-1=7) ou (A=10d-2-1=7)

; No caso da subtração, os valores finais de C e OV também trazem informações
; sobre o status final da operação.

; LABEL INSTR OPERANDO COMENTARIO

.*****
;

CLR C ;limpa o "carry"

MOV A,#3Ch ;move o valor 3Ch=60d para o acumulador

SUBB A,#32h ;subtrai o valor 32h=50d mais o conteúdo do "carry"
;de A, deixando o resultado no acumulador
; (A=3Ch-32h-0=0Ah) ou (A=60d-50d=10d)

; Neste caso, o "carry" permanece em 0, pois não foi necessária a operação de

; "empréstimo" para o bit 7, apenas para o bit 1. Observe que o bit 2

; "empréstimo" para o bit 1 com peso 2. Isto é equivalente a 10b, em binário.

; Neste caso, 10b-1b=1b. Na subtração seguinte, isto é, com o bit 2, não

; se pode esquecer que já foi emprestado 1, logo o resultado da subtração

; neste bit é 0b-0b=0b (Pense como na subtração em base decimal, observando

; que o "empréstimo" em decimal representava 10 em decimal, e em base binária,

; representa o empréstimo de 2 em decimal ou 10 em binário)

; Observe também que OV=0 (será analisado a seguir)

; empresta 1

; 0 0 1 1 1 1->10 0

; - 0 0 1 1 0 0 1 0

; -----

; 0 0 0 0 1 0 1 0

; LABEL INSTR OPERANDO COMENTARIO

.*****

;

CLR C ;limpa o "carry"

MOV A,#32h ;move o valor 32h=50d para o acumulador

SUBB A,#3Ch ;subtrai o valor 3Ch=60d mais o conteúdo do "carry"

 ;de A, deixando o resultado no acumulador

 ;(A=32h-3Ch-0=F6h)

; Neste caso, o "carry" vai para 1, pois foi necessaria a operacao de

; "empresta-um" para o bit 7 (bem como para os bits 2, 3, 4, 5 e 6, como

; indicado no diagrama a seguir).

; Alem disso, pensando em complemento de 2, verificamos que o resultado

; e' coerente, ou seja, A=50d-60d=-10d (11110110b)

; Observe também que OV=0 (será analisado a seguir)

;

; 1-> 0 ->0 ->1 ->1 ->0 ->0 1 0

; - 0 0 1 1 1 1 0 0

; -----

; 1 1 1 1 0 1 1 0

; Como regra geral, podemos assumir que ao executarmos a instrução SUBB

; C=0 implica em resultado positivo da subtração e C=1 implica em resultado

; negativo da subtração.

; No caso do bit de "overflow" (OV), e' mais simples analisar cada

; caso admitindo-se complemento de 2 e tem-se como regra:

; Se um valor negativo e' subtraído de um valor positivo e o resultado e' negativo,

; então OV=1

; Se um valor positivo e' subtraído de um valor negativo e o resultado e' positivo,

; então OV=1

; Nos demais casos, OV=0, como visto nos dois casos anteriores.

;LABEL INSTR OPERANDO COMENTARIO

.*****

CLR C ;limpa o "carry"

MOV A,#00101100b ;move para A o valor 2Ch=44d

SUBB A,#10010000b ;subtrai o valor 90h=144d mais o conteúdo do "carry"
;de A, deixando o resultado no acumulador

; Neste caso, o resultado obtido é 10011100b. Pensando em complemento de 2

; subtraímos um valor negativo de outro positivo e obtivemos um valor

; negativo. Portanto OV=1. Como na operação existe "empresta-um" para o

; bit 7 tem-se C=1.

;LABEL INSTR OPERANDO COMENTARIO

.*****

CLR C ;limpa o "carry"

CLR OV ;limpa o bit de "overflow"

MOV A,#10010000b ;move para A o valor 90h=144d

SUBB A,#01010000b ;subtrai o valor 50h=80d mais o conteúdo do "carry"
;de A, deixando o resultado no acumulador

END

; Neste caso, o resultado obtido é 01000000b. Pensando em complemento de 2

; subtraímos um valor positivo de outro negativo e obtivemos um valor

; positivo. Portanto OV=1. Como na operação não existe "empresta-um" para o

; bit 7 tem-se C=0.

; Verifica-se portanto que, juntos, os bits de "carry" (C) e "overflow" (OV)

; também podem fornecer muitas informações sobre o status final da operação

; de subtração.

; Finalizando este exemplo, pode-se dizer que o Flag de Carry (C) além de

; indicar o status da operação aritmética, pode ser usado na própria.

; operação, enquanto que o Flag de OverFlow (OV) nos fornece apenas um

; status.

2.5.1 Cálculo aproximado de retardos em *assembler*.

Realizar uma demora por hardware, utilizando temporizadores é muito mais eficiente e preciso, além disso, o microprocessador não tem que dedicar-se somente a executar a demora, mas sim pode ir executando outras tarefas e manter o controle dos periféricos, mas os temporizadores serão estudados posteriormente nesta disciplina.

O retardo por software, se logra mediante vários registradores aninhados, com mais registradores quanto maior seja o número de contagem ou demora. O decremento sucessivo dos mesmos produza este tempo de espera.

Para calcular este tempo, deve-se conhecer a quantidade de ciclos de máquina que empregam todas as instruções que intervêm no programa de demora e quantas vezes se executam elas, e é obvio terá que conhecer o tempo de duração de cada ciclo de máquina.

Considerando que a $f_{xtal} = 12 \text{ MHz}$ e que um ciclo de máquina na família MCS 51 consta de 6 estados e cada estado contém 2 períodos de relógio, pode-se dizer que cada ciclo de máquina equivale a 12 períodos de relógio. Daí que:

$$T = 1/f \quad T = 1/12 \text{ MHz} \quad T_{\text{ciclo}} = 1/12 \text{ MHz} * 12 = 1 \mu\text{s}$$

Calculemos o tempo de duração da sub-rotina,

Exemplo 1:

DEM: MOV R7,#100	;1 ciclo = 1 μs
LAZO: DJNZ R7,LAZO	;2 ciclos = 2 μs
RET	;2 ciclos = 2 μs

Para isto adicionemos os tempos de duração das instruções

$$T_{\text{sub-rotina}} = T_{\text{ciclo}} + R7 (2 T_{\text{ciclo}}) + 2 T_{\text{ciclo}}$$

$$R7 = 100$$

$$T_{\text{sub-rotina}} = 1 \mu\text{s} + 100 * 2 * 1 \mu\text{s} + 2 \mu\text{s} = 203 \mu\text{s}$$

Observa-se que a instrução DJNZ se repete 100 vezes e como dura 2 μs , contribui 200 μs à demora.

Aqui, far-se-á um cálculo aproximado e só se terá em conta a quantidade de ciclos de máquina que contribui a instrução DJNZ, o resto se despreza e portanto se comete um ligeiro engano que é desprezível. Deixa-se como exercício proposto, ao

estudante, o cálculo exato tendo em conta todos os ciclos de máquinas, de cada uma das instruções.

Exemplo 2:

Necessita-se obter uma demora de 10 s.

O primeiro que se deve fazer é converter as unidades de segundos a microssegundos.

$$1 \text{ s} = 10\,000\,000 \mu\text{s}.$$

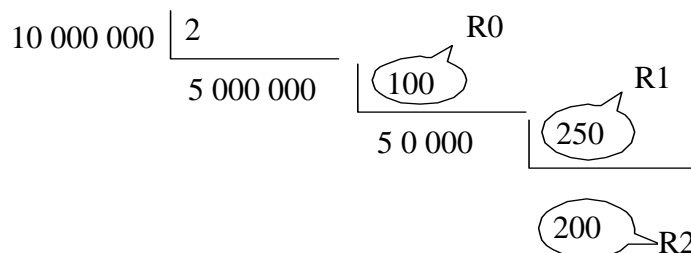
Como a demora é relativamente grande é necessário aninhar vários registradores.

```
MOV R0,#100
AQUI3: MOV R1,#250
AQUI2: MOV R2,#200
AQUI1: DJNZ R2,AQUI1
        DJNZ R1,AQUI2
        DJNZ R0,AQUI3
```

Observem a estrutura de laços aninhados e como, para diminuir uma vez o registrador R1, devem diminuir-se 200 vezes o registro R2 e como, para diminuir uma vez R0, devem-se diminuir 250 vezes a R1.

Como calcular os valores a carregar nos registros?

Primeiro se toma o # máximo a contar (10 000 000) e se divide entre 2 (ciclos do DJNZ), se o resultado fora menor que 255 que for o # máximo que se pode carregar em um registro de 8 bits, só se usaria um registro como no exemplo 1, mas não é o caso, portanto se segue dividindo. O divisor deve ser um múltiplo do dividendo, para obter que o quociente sempre seja um # inteiro. A divisão se continua até que o quociente seja um # menor ou igual a 255.



Para comprovar que o resultado está correto, multiplica-se:

$$100 * 250 * 200 * 2 = 10\,000\,000$$

2.6 Programação em linguagem C usando o compilador SDCC

A variante de programação em linguagem C se mostra na figura 2.6. Todo programa em C é uma coleção de funções. As funções recebem argumentos e retornam valores, mas existem funções que não recebem argumentos ou que não retornam valores. Toda declaração (*statement*) em C deve terminar com ponto e vírgula (“;”).

Em C, faz-se distinção entre letras maiúsculas e minúsculas (valor e Valor são interpretados diferentes).

As chaves “{” e “}” são empregadas para indicar agrupamentos dentro do programa. As variáveis declaradas dentro de um agrupamento {...} só possuem o escopo dentro de este agrupamento.

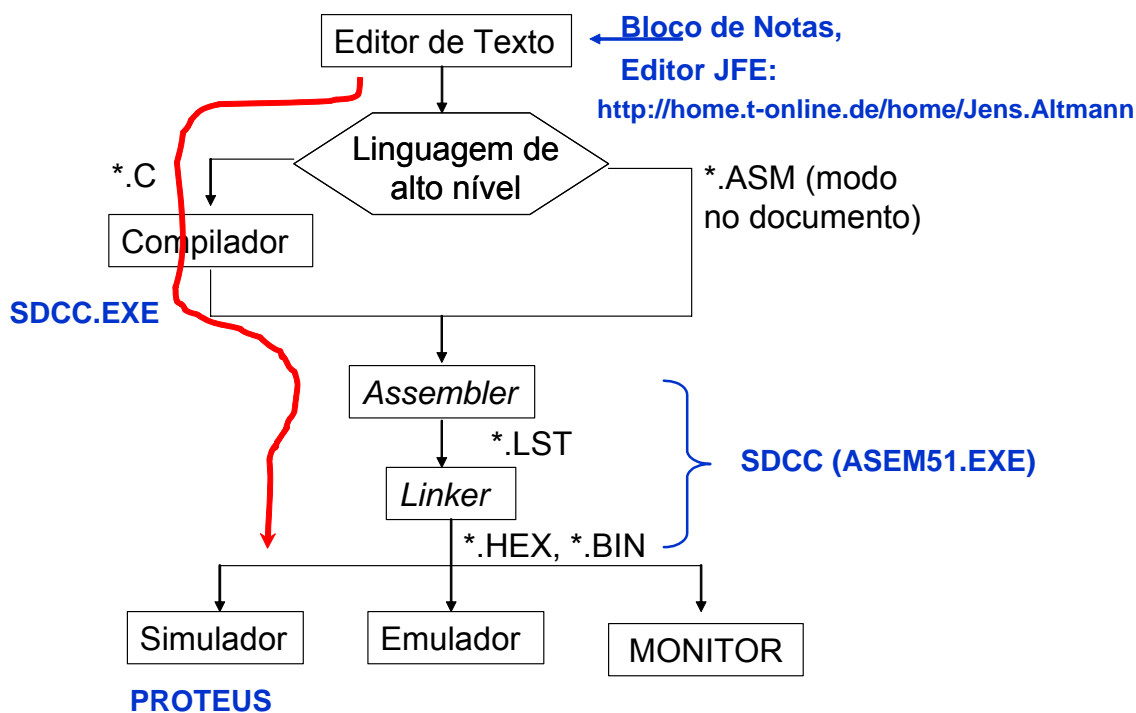


Figura 2.6 Diagrama de fluxo para o trabalho em linguagem C com o microprocessador/microcontrolador 8051.

A estrutura de um programa em C para computadores é a mesma para um programa de microcontroladores. Nem todos os comando da linguagem C estão disponíveis para microcontroladores. É possível trabalhar com múltiplas bibliotecas.

Para executar o código é necessário que o programador possua algum simulador de microcontroladores, emulador ou que grave em um microcontrolador e teste em um hardware.

O arquivo fonte na linguagem C é uma seqüência de linhas de texto, que podem ter (Figura 2.7):

- Linhas de códigos de pré-processamento.
- Funções.
- Declaração de variáveis.
- Linhas de código *assembly*.
- Linhas de comentário ("*//* ou */* ... */*").

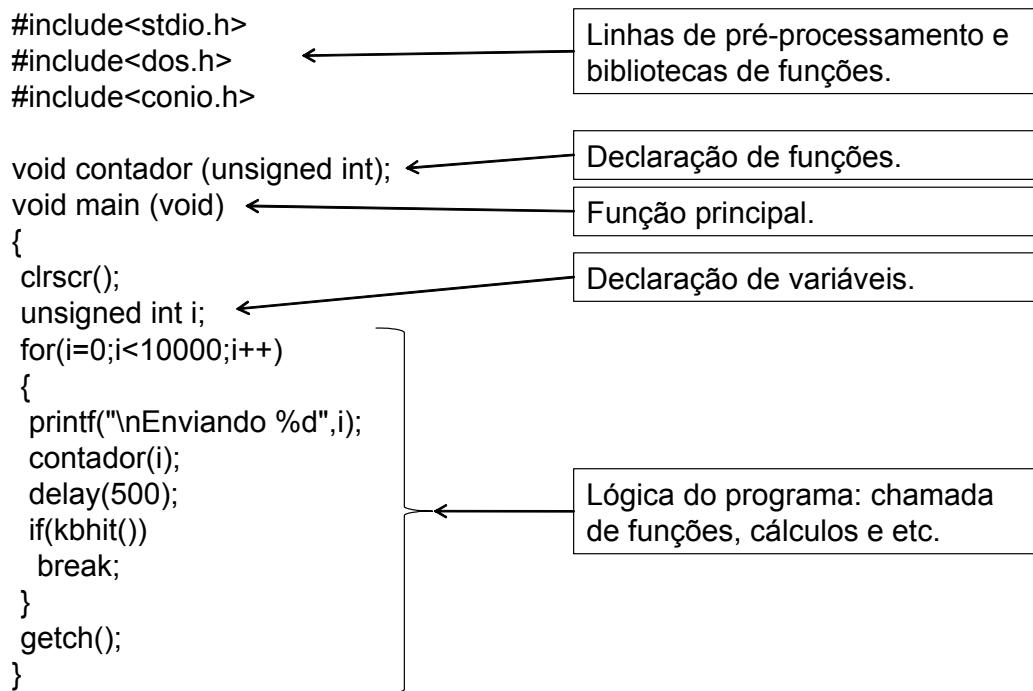


Figura 2.7 Estrutura de um programa em linguagem C

As linhas de pré-processamento são úteis para colocar definições de portas, constantes e etc, da mesma forma que se faz testes em relação ao sistema de processamento.

As bibliotecas de funções podem ser arquivos `*.c` ou mesmo `*.h`. São inseridos com `#include <*. *>`.

Quando deseja-se realizar cálculos, conversões, escritas em portas, leituras de memória, etc., fora da função *main*, é necessário declarar as funções e os argumentos que irão receber e retornar (quando é aplicável) antes da função *main*. Para não ter que ser feito isso, o programador pode escrever as funções antes da função *main*, assim não é necessário fazer essas declarações.

A declaração de variáveis, vetores e matrizes deve sempre ser feita antes de que se comece a escrever as linhas de código do programa, do contrário, o compilador acusará erro.

Na lógica de programa entra todas as funções já conhecidas do C e cálculos do programa. Os comandos de pre-processors continuam disponíveis nos microcontroladores.

Alguns compiladores (por exemplo o CCS para microcontroladores PIC) já possuem certas funções implementadas em seu código, facilitando o esforço de programação: `i2c_read()`, `read_adc()`, etc.

As funções implementadas são fornecidas pelas empresas desenvolvedoras dos compiladores. Certos compiladores permitem que códigos em assembly sejam inseridos dentro do código C.

A seguir se mostra um exemplo de programa:

```
/* Exemplo de um programa */
int calc_quadrado (int x)
{
    int y;    //declara variável y
    y = x*x;  //calcula o quadrado
    return (y); //retorna o quadrado
}
void main (void)
{
    int i, qdr;
    i = 5;
    qdr = calc_quadrado(i);
}
```

A palavra reservada VOID significa que a função *main* não recebe nem retorna parâmetros. A maioria dos programas para microcontroladores ficam eternamente na memória, presos num laço infinito, raramente recebem ou retornam valores.

Nesta disciplina se estudará o compilador SDCC (*Small Device C Compiler*) que é de código livre e muito simples de usar. Esse compilador é conveniente, para usar em projetos como o microprocessador ou qualquer membro de sua família.

Tipos de variáveis para o compilador SDCC.

O compilador SDCC tem apenas os tipos de variáveis mostradas na tabela seguinte:

Nome	Tipo	Bits	Faixa sem sinal (<i>unsigned</i>)	Faixa com sinal (<i>signed</i>)
Caráter	char	8	0-255	-128-127
Inteiro	int	16	0-65535	-32.768-32.767
Inteiro curto	short	16	0-65535	-32.768-32.767
Inteiro longo	long	32	0-4.294.967.295	-2.147.483.648 - 2.147.483.647
Flutuante	float	32	3,4×10 ⁻³⁸ a 3,4×10 ³⁸	-3,4×10 ³⁸ a 3,4×10 ³⁸

2.6.1 Definição de Constantes: DEFINE

#define IDENTIFICADOR *valor*

- O compilador substituirá o IDENTIFICADOR pelo valor, quando o encontrar no arquivo fonte.
- Permite também definir macros, nas quais se substitui seus parâmetros pelos do ponto em que a invoca.

Exemplos:

```
#define TRUE 1
#define END_LEDS 0x8000 //endereço externo
#define CR 0x13 //ASCII do carriage return
#define pi 3.14159
#define LETRA_A 'A' //Código ASCII da letra A
```

Tem o mesmo papel do EQU em assembly. É costume empregar letras maiúsculas para tais declarações.

2.6.2 Cadeias de caracteres (string)

- São *arranjos* de 0 ou mais caracteres que são sempre delimitados por aspas duplas (“ e ”).
- Em C, toda string é indicada por seu endereço de início e seu final é marcado pelo byte 0 (zero, ‘\0’) o qual é inserido automaticamente. Isto simplifica bastante porque com um simples endereço faz-se referência à string.

Exemplos:

- char vetor[20] = “ABCDEF” equivale a
- vetor[0] = 0x41 (ASCII de A), vetor[1]=0x42 (ASCII de B)...vetor[6]=0
- vetor[7...19]= não definidos

- A constante "" é formada apenas por NULL '\0';
- A constante "A" é uma cadeia com os literais 'A' e '\0';
- A constante "texto" é uma cadeia com os literais 't', 'e', 'x', 't', 'o' e '\0';
 - Constantes literais e constantes de texto são armazenadas de forma diferente e, portanto, NUNCA podem ser comparadas ou confundidas.
"A" ≠ 'A'

2.6.3 Operadores em C

Operadores aritméticos

Operador	Uso	Resultado
+	x+y	x+y (Soma)
-	x-y	x-y (Subtração)
*	x*y	x*y (Multiplicação sem signo)
/	x/y	x/y (Divisão)
%	x%y	Módulo (Resto da divisão)

O Tipo de variável impõe algumas restrições. Por exemplo:

- Se forem empregadas variáveis inteiras: $7/3 = 2$
- Deve-se tomar cuidado para não extrapolar a faixa de cada tipo:
char i = 100, j = 200;
i = i * j; //20000 não pode ser armazenado em um byte

Operadores em C: Relacionais e Lógicos

Operador	Uso	Operação realizada
>	x>y	Maior que
>=	x>=y	Maior que ou igual a
<	x<y	Menor que
<=	x<=y	Menor que ou igual a
==	x == y	Igual
!=	x != y	Diferente
&&	x && y	x AND y
	x y	x OR y
!	! y	NOT (NÃO)

Operadores em C: De bits

Operador	Uso	Resultado
&	x&y	AND <i>bitwise</i> (bit a bit)
	x y	OR bitwise (bit a bit)
^	x^y	XOR (bit a bit)
~	~x	NOT x (bit a bit)
>>	x >> N	Deslocamento para direita N posições
<<	x << N	Deslocamento para esquerda N posições

Operadores de incremento (++) e decremento (--). Ao invés de empregarmos $x = x + 1$; ou $y = y - 1$; é mais eficiente escrevermos $x++$ e $y--$; respectivamente.

Exemplo 1: $z \&= 0xFF00$; //Faz: $z \leftarrow z \& 0xFF00$

Exemplo 2:

`int x,y,z` //declaração de três variáveis

`z = x = y = 5;`

`z+=10;` // $z = z + 10$

`x = (z++) + 20;` // $x = z + 20$ e depois incrementa z

`x = (++z) + 20;` //faz o incremento de z e depois $x = z + 20$.

`y = z >> 8;` //desloca 8 vezes para a direita o conteúdo de z e o atribui a y

Operadores em C: De Endereço

Operador	Uso	Significado
*	*variável	Operador de conteúdo apontado ou operador de não-endereçamento
&	&variável	Operador de endereço da variável

Exemplo:

```
{ int *ponteiro, a=25, b;
```

```
    ponteiro = &a; //coloco o endereço de (a) no ponteiro
```

```
    b = *ponteiro; //atribuo a (b) o conteúdo do endereço à qual aponta o ponteiro, o  
    seja, (25)
```

```
    b = ponteiro; /*atribuo a (b) o endereço à qual aponta o ponteiro, o seja o  
    endereço de (a)*/
```

```
}
```

2.6.4 Conversiones de tipo implícitas y explícitas (*casting*)

Mais adiante se comentarão as conversões implícitas de tipo que ocorrem quando em uma expressão se mesclam variáveis de distintos tipos. Por exemplo, para poder somar duas variáveis faz falta que ambas sejam do mesmo tipo. Se uma for *int* e outra *float*, a primeira se converte a *float* (ou seja, a variável do tipo de menor range se converte ao tipo de maior range, antes de realizar a operação). A esta conversão automática e implícita de tipo (o programador não precisa intervir, embora sim conhecer suas regras), lhe denomina *promoción*, pois a variável de menor range é promovida ao range da outra.

Asim, quando dois tipos diferentes de constantes e/ou variáveis aparecem em uma mesma expressão, relacionadas por um operador, o compilador converte os dois operandos ao mesmo tipo de acordo com os ranges, que de major a menor se ordenam do seguinte modo: *long double > double > float > unsigned long > long > unsigned int > int > char*.

Outra classe de conversão implícita tem lugar quando o resultado de uma expressão é atribuído a uma variável, pois dito resultado se converte ao tipo da variável (neste caso, esta pode ser de menor range que a expressão, por isso esta conversão pode perder informação e ser perigosa). Por exemplo, se *i* e *j* são variáveis inteiras e *x* é *double*, $x = i*j - j + 1$;

En C existe también la posibilidad de realizar *conversiones explícitas de tipo* (llamadas **casting**, en la literatura inglesa). El casting es pues una conversión de tipo, forzada por el programador. Para ello basta preceder la constante, variable o expresión que se desea convertir por el tipo al que se desea convertir, encerrado entre paréntesis. En el siguiente ejemplo, $k = (int) 1.7 + (int) masa$; la variable **masa** es convertida a tipo *int*, y la constante **1.7** (que es de tipo *double*) también. El **casting** se aplica con frecuencia a los valores de retorno de las funciones.

Em C existe também a possibilidade de realizar *conversões explícitas de tipo* (chamadas **casting**, na literatura inglesa). O *casting* é uma conversão de tipo, forçada por o programador. Para isto é necessário anteceder à constante, variável ou expressão que se deseja converter por tipo ao que se deseja converter, encerrado entre parêntesis. No seguinte exemplo, $k = (int) 1.7 + (int) masa$; a variável **masa** é convertida a tipo *int*, e a constante **1.7** (que é de tipo *double*) também. O **casting** se aplica com frequência aos valores de retorno das funções.

2.6.5 Estruturas para o controle de fluxo

As sentenças de um programa em C se executam seqüencialmente, isto é, cada uma a seguir da anterior começando pela primeira e acabando pela última. A linguagem C dispõe de varias sentenças para modificar este fluxo seqüencial da execução.

As mais utilizadas se agrupam em duas famílias:

- **os trechos de programa:** que permitem executar repetidamente um conjunto de instruções tantas vezes como se deseja, trocando ou atualizando certos valores.
- **as bifurcações:** que permitem escolher entre duas ou mais opções segundo certas condições, e

2.6.5.1 Trechos de programa condicionais: WHILE

Executa trechos de programa enquanto uma condição for válida

```
while (condição) {  
Bloco de instruções para executar  
}
```

/*Exemplo: Função de cálculo da media dos 16 últimos dados lidos de um conversor A/D ligado a porta P1 (Para reduzir efeito de ruído). */

```
char media(void){  
    int i,soma;  
    soma = i = 0;  
    while (i < 16) {  
        soma += P1;  
        i++;  
    }  
    return soma>>4; //Divisão por 16 }
```

Algumas vezes é preciso colocar o programa num laço infinito.

/*Exemplo2: O emprego de *break*, faz possível sair do laço infinito. */

```
while (1) { /* Laço infinito */  
    declarações
```

```

...
    if (condição de saída) break;
...
    declarações
}

```

2.6.5.2 Trechos de programa condicionais: FOR

```

for (valor inicial ; condição de permanência ; incremento)
{
    Conjunto de instruções para executar;
}

```

```

/*Exemplo: Meia de 16 valores lidos da porta 1 implementado com FOR */
char mediaP1(void) {
    char i, soma;
    soma = 0;
    for (i = 0; i < 16; i++) soma+= P1;
    return soma>>4
}

```

2.6.5.3 Bifurcações: IF e variantes

```

if (condição)
{
    Instruções para executar
}

```

```

if ( condição )
{
    Bloco para executar se condição é certa
}
else
{
    Bloco para executar se condição não é certa
}

```

```
if (condição) declaração;
else if (condição) {declarações;}
else if (condição) {declarações;}
.....
else {declarações;}
```

Exemplos:

```
1. if (x>5)  z = 10;
    else z = 20;
```

```
2. if (x>5)
```

```
{
    z = 10;
    y = 20;
```

```
}
```

```
else
```

```
{
    z = 20;
    y = 10;
```

```
}
```

```
3. if (x <= 10) {
```

```
    x++
```

```
    z = 10;
```

```
}
```

2.6.5.4 Bifurcações: Switch - Case

```
switch (variável )
```

```
{ case constante 1:
```

```
    código a executar se a variável = constante 1
```

```
    break;
```

```
    case constante2:
```

```
        código a executar se a variável = constante 2
```

```
    break;
```

```

default:
    código a executar se a variável é diferente aos anteriores
break;
}

```

É importante que cada trecho finalize com um *break*. Caso contrario, o processamento seguiria deste ponto em diante

BREAK fornece uma maneira de se sair antecipadamente de laços tipo while, for, do ou switch. O BREAK sempre faz a quebra do laço mais interno

Exemplo: Função que retorna o código ASCII do nibble recebida (**nib**)

```

char nib_asc (char nib) {
    char resp; /* variavel para receber a resposta */
    nib &= 0xF; /*(nib=nib&0xF) garantir faixa de 0 a 15*/
    switch (nib) {
        case 0: resp = '0' ; break;
        case 1: resp = '1' ; break;
        case 2: resp = '2' ; break;
        case 3: resp = '3' ; break;
        case 4: resp = '4' ; break;
        case 5: resp = '5' ; break;
        case 6: resp = '6' ; break;
        case 7: resp = '7' ; break;
        case 8: resp = '8' ; break;
        case 9: resp = '9' ; break;
        case 10: resp = 'A' ; break;
        case 11: resp = 'B' ; break;
        case 12: resp = 'C' ; break;
        case 13: resp = 'D' ; break;
        case 14: resp = 'E' ; break;
        default: resp = 'F' ; break; }
    return resp; }

```

Alternativa usando string: A seguir, uma maneira mais eficiente de fazer tal conversão:


```
char tabela[ ] = "0123456789ABCDEF" ;
char nib_asc (char nib) {
    nib &= 0xF; /*garantir faixa de 0 a 15*/
    return tabela[nib]
}
```

Observações:

- Não foi necessário indicar seu tamanho, pois ela já é inicializada como uma *string*.
- Tal declaração foi feita externamente à função nib_asc, o que a torna uma variável global.

2.6.5.5 Sentencias de salto: GOTO

A declaração **GOTO** permite saltar (desviar) até uma *label* determinada.

Formalmente o GOTO não deveria ser usado, entretanto existem situações onde GOTO pode ajudar:

- Para abandonar os laços DO, WHILE, ou FOR
- Para saltar imediatamente um trecho específico de programa.

```
while (...) {
    for (...)
        if (erro) goto deu_erro;
}
....
deu_erro:
    rotina para tratar o erro
```

2.6.6 Particularidades do SDCC para 8051

Quais alterações são feitas para que linguagem C fosse adaptada ao universo dos microcontroladores?

2.6.6.1 Modelos de memória

Small: As variáveis são alocadas na RAM interna. Modelo default do compilador. Gera códigos mais eficientes.

Large: As variáveis são alocadas na XRAM incluindo todos parâmetros e variáveis locais. Quando se emprega este modelo, diversas otimizações são desabilitadas

A escolha do modelo depende do número de variáveis envolvidas no software e da disponibilidade de XRAM

Módulos com diferentes modelos de memória nunca devem ser misturados, tanto é que existem dois conjuntos de bibliotecas, um para cada modelo.

2.6.6.2 Classes de Armazenamento

Disponibilizar o sofisticado ambiente de programação do 8051 aos programadores C é um bom desafio. NO caso de SDCC, o compilador permite que o usuário defina a classe de armazenamento para suas variáveis.

Valido para todos os tipos de variáveis: char, int, short, long e float. Definem onde e como são alocadas a variáveis.

data/Near

xdata/Far

idata

pdata

code

bit

sfr/bit

1) data/near: Variáveis declaradas com esta classe são alocadas na RAM interna nos endereços de 0 a 127.

Exemplo:

```
data unsigned char teste;
```

Uma atribuição do valor 1 equivale a `MOV _teste,#1`

As variáveis declaradas em C são referenciadas através de seu rótulo (`_teste` neste caso)

Se nada for dito, esta é classe de armazenamento empregada pelo compilador

2) xdata/far: Variáveis declaradas com esta classe são alocadas na RAM externa nos endereços de 0 a 127.

Exemplo:

```
xdata unsigned char xteste;
```

Uma atribuição do valor 2 resulta nas instruções *assembly*:

```
MOV DPTR, #_xteste ;DPTR ← endereço da variável xteste
```

```
MOV A, #2           ; Dado imediato a ser escrito
```

```
MOVX @DPTR,A       ; Altera o conteúdo da variável xteste
```

Se nada for dito, esta é classe de armazenamento default do modelo *large* (quando empregado pelo compilador).

3) idata: Variáveis declaradas com esta classe são alocadas na RAM interna dentro da área endereçável indiretamente (128 *Lower* + 128 *Uppper*). A primeira metade desta área (endereços 0 a 127) também trabalha com endereçamento direto. Alguns membros da família não possuem endereços de 128 a 255.

Exemplo:

```
idata unsigned char iteste;
```

Uma atribuição do valor 3 resulta nas instruções *assembly*:

```
MOV R0, #_iteste    ;R0 ← endereço da variável iteste
```

```
MOV A, #3           ; Dado imediato a ser escrito
```

```
MOVX @R0,A         ; Altera o conteúdo da variável xteste
```

Se nada for dito, esta é classe de armazenamento default do modelo *large* (quando empregado pelo compilador).

4) pdata: Permite uma paginação muito simples na memória de dados externa. Faz uso das instruções MOVX A, @R0 e MOVX @R0, sendo que o endereço é previamente armazenado na porta P2 ⇒ endereços só de 0 a 255.

Exemplo:

```
pdata unsigned char *pteste; //ponteiro do tipo pdata
```

```
pteste_pt = (pdata *) 0xFE;  //ponteiro = endereço 0xFE
```

```
*pteste_pt = 4;              //escreve 4
```

Em assembly:

```
MOV _pteste_pt,#0FEH
MOV R0, _pteste_pt    ; R0 ← conteúdo do ponteiro
MOV A, #4              ; ACC ← valor a ser escrito
MOVX @R0, A           ; Altera posição para 4
```

5) code: Variáveis declaradas com esta classe são alocadas na memória de programa.

Exemplo:

```
code unsigned char cteste;
```

A leitura desta variável resulta nas instruções assembly

```
MOV DPTR,#_cteste    ;DPTR ← endereço da variável cteste
CLR A                ; zera acumulador
MOVC A,@A+DPTR       ; lê a variável
```

6) bit: Variáveis declaradas com esta classe são alocadas da área endereçável bit-a-bit na memória de dados.

Exemplo:

```
bit bit_teste;
```

A escrita do valor 1 nesta variável resulta na instrução assembly

```
SETB _bit_teste      ;ativa o bit
```

Apenas se usam os endereços de bit de 0 a 127H (bytes 20H a 27H) da RAM interna. Os demais são considerados como bits de registradores de funções especiais.

2.6.7 Ponteiros para os diversos espaços de memória

Se nada for dito, o compilador cria um ponteiro genérico que pode ser empregado com qualquer área de memória.

O ponteiro genérico implica numa maior carga de programa e, por isso, sempre que possível, o programador deve explicitar o ponteiro que está criando.

Exemplos:

- `xdata unsigned char * data pt; /*ponteiro na RAM interna apontando a XRAM*/`
- `data unsigned char * xdata pt; /*ponteiro na XRAM apontando para RAM interna*/`
- `xdata unsigned char * code pt; /*ponteiro na memo. de prog. apontando a XRAM*/`
- `code unsigned char * code pt; /*ponteiro na memo. de programa apontando para memo. de programa*/`

3 ATENÇÃO A DISPOSITIVOS DE ENTRADA-SAÍDA

Neste tema serão estudados os diversos dispositivos de entrada saída que tem o 8051: portas paralelas, temporizadores, interrupções e porta serial. Para cada caso serão descritos seu hardware e os registradores de controle e dados associados a eles. Posteriormente serão projetadas aplicações simples, mostrando o hardware e seu programa (em assembler e/ou C) de manuseio. Se começara pelo estúdio das portas paralelas do 8051 que são as mais simples e as mais usadas.

3.1 Portas paralelas no 8051.

3.1.1 Estrutura e operação

As portas P0, P1, P2 e P3 são bidirecionais. Cada bit destas portas está composta por:

- Um *latch* (corresponde aos SFR P0 a P3).
- Um *driver* de saída.
- Um buffer de entrada.

Na figura 3.1 mostra um esquema simplificado de um bit de E/S, que ajuda a compreender o funcionamento da porta. Na realidade, os circuitos são ligeiramente

diferentes para cada uma das portas, e mais complexos do que o apresentado aqui, mas para fins de entendimento este diagrama básico é suficiente.

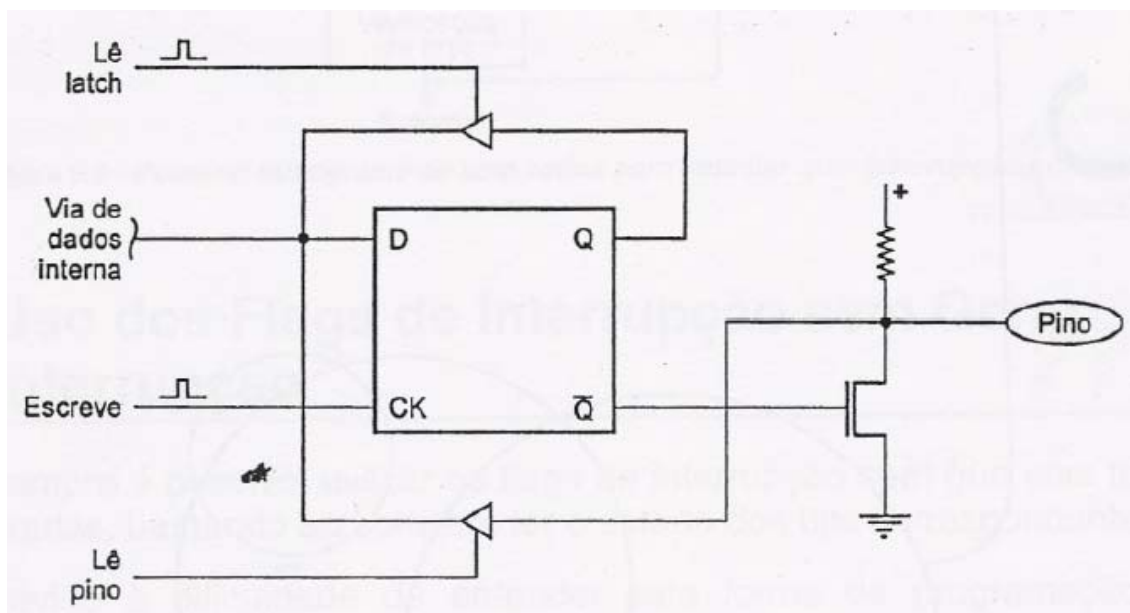


Figura 3.1. Circuito simplificado de uma linha de porta de Entrada-Saída.

Quando escrevemos um bit na porta, ele “caminha” pela via interna até a entrada D do flip-flop, e então um pulso de *clock* em CK coloca esse bit em #Q, que pela eletrônica associada leva esse nível ao pino externo.

Já no caso da leitura, algumas instruções lêem o pino enquanto outras lêem o estado do *latch*, em Q. As instruções que lêem o *latch* são aquelas que costumam ler, modificar e escrever o bit, a saber: INC, DEC, CPL, JBC, DJNZ, ANL, ORL, XRL, MOV Px.n, C; CLR Px.n e ainda SETB Px.n.

As demais instruções de acesso às portas lêem o estado presente nos pinos.

Qual é a diferença?

Vamos supor que temos a base de um transistor NPN sendo acionada pelo pino de E/S. Quando escrevermos 1 nesse pino, o transistor entrará em condução, e se a CPU ler o estado desse pino para eventual confirmação, lerá a tensão na base do transistor e interpretará como nível ZERO, então neste caso deve ser lido o valor do *latch*, que estará corretamente indicando nível UM.

As portas 1, 2 e 3 têm resistores de *pull-ups* internos. Para usar estas portas como entrada, no *latch* deve ser armazenado previamente um “1”, o qual curta ao FET de saída e o pino correspondente é posto a “1” pelo *pull-up* interno. Na figura 3.2 se mostra a estrutura eletrônica interna real da porta paralela P0.

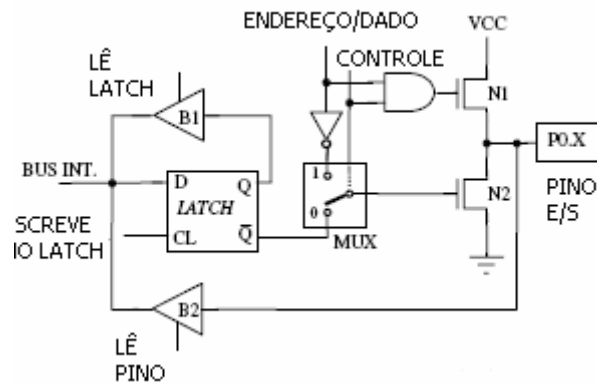


Figura 3.2. Estrutura interna de porta paralelo P0 no 8051.

A porta 0 não tem *pull-up* interno. O FET que atua como *pull-up* só trabalha quando a porta está emitindo um “1” durante os acessos à memória externa; em qualquer outra ocasião ele está cortado. Portanto, em saída os bits desta porta trabalham como coletor aberto: escrever um “1” no latch põe flutuante ao pino correspondente. Portanto, ao ter um “1” no *latch*, pode-se utilizar o pino correspondente como uma entrada de alta impedância. Recomenda-se utilizar um *pull-up* externo de 10 K Ω na porta P0 quando opera como porta de saída.

Em resumo: para ler os pinos das portas paralelas no microprocessador 8051 é aconselhável escrever previamente um “1” no *latch* correspondente ao bit, e depois se faz a leitura deste. Esse tipo de estrutura de saída se conhece como quase-bidirecional.

Porque os sinais “lê latch” e “lê pin” ? . Ao ler uma porta, pode ser que leiamos o valor situado no pino ou o valor armazenado no *latch*. Esclarecemos:

Quando a instrução que lê o porto tem como destino um lugar diferente ao porto, lê-se o pino.

Exemplo: ANL A, P0 lê os pinos de P0.

Quando a instrução modifica o dado existente na porta, ou seja, o operando destino é a porta, lê-se o *latch* (e não o pino) e se escreve o novo valor no *latch*.

Exemplo: ANL P0, A lê o *latch* de P0, faz um AND com o ACC e o resultado o escreve no *latch* (e no pino) de P0.

3.1.2 Características elétricas em DC no Microprocessador 8051

As características de DC da família 8051 aparecem no seguinte fragmento do *datasheet*. É muito importante ter em conta os baixos valores de corrente de cada

uma dos pinos ($I_{OL} = 3,2 \text{ mA}$ para a porta 0, e $I_{OL} = 1,6 \text{ mA}$ para os restantes portas). Desta forma não é possível conectar um LED a um pino de saída de qualquer porta.

Operating Conditions: T_A (Under Bias) = 0°C to $+70^\circ\text{C}$; $V_{CC} = 5\text{V} \pm 10\%$; $V_{SS} = 0\text{V}$

D.C. CHARACTERISTICS (Under Operating Conditions)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage (Except \overline{EA} Pin of 8751H & 8751H-8)	-0.5	0.8	V	
V_{IL1}	Input Low Voltage to \overline{EA} Pin of 8751H & 8751H-8	0	0.7	V	
V_{IH}	Input High Voltage (Except XTAL2, RST)	2.0	$V_{CC} + 0.5$	V	
V_{IH1}	Input High Voltage to XTAL2, RST	2.5	$V_{CC} + 0.5$	V	$XTAL1 = V_{SS}$
V_{OL}	Output Low Voltage (Ports 1, 2, 3)*		0.45	V	$I_{OL} = 1.6 \text{ mA}$
V_{OL1}	Output Low Voltage (Port 0, ALE, \overline{PSEN})*				
	8751H, 8751H-8		0.60	V	$I_{OL} = 3.2 \text{ mA}$
			0.45	V	$I_{OL} = 2.4 \text{ mA}$
	All Others		0.45	V	$I_{OL} = 3.2 \text{ mA}$
V_{OH}	Output High Voltage (Ports 1, 2, 3, ALE, \overline{PSEN})	2.4		V	$I_{OH} = -80 \mu\text{A}$
V_{OH1}	Output High Voltage (Port 0 in External Bus Mode)	2.4		V	$I_{OH} = -400 \mu\text{A}$
I_{IL}	Logical 0 Input Current (Ports 1, 2, 3, RST) 8032AH, 8052AH All Others		-800 -500	μA μA	$V_{IN} = 0.45\text{V}$ $V_{IN} = 0.45\text{V}$
I_{IL1}	Logical 0 Input Current to \overline{EA} Pin of 8751H & 8751H-8 Only		-15	mA	$V_{IN} = 0.45\text{V}$
I_{IL2}	Logical 0 Input Current (XTAL2)		-3.2	mA	$V_{IN} = 0.45\text{V}$
I_{LI}	Input Leakage Current (Port 0) 8751H & 8751H-8 All Others		± 100 ± 10	μA μA	$0.45 \leq V_{IN} \leq V_{CC}$ $0.45 \leq V_{IN} \leq V_{CC}$
I_{IH}	Logical 1 Input Current to \overline{EA} Pin of 8751H & 8751H-8		500	μA	$V_{IN} = 2.4\text{V}$
I_{IH1}	Input Current to RST to Activate Reset		500	μA	$V_{IN} < (V_{CC} - 1.5\text{V})$
I_{CC}	Power Supply Current: 8031/8051 8031AH/8051AH 8032AH/8052AH 8751H/8751H-8		160 125 175 250	mA mA mA mA	All Outputs Disconnected; $\overline{EA} = V_{CC}$
C_{IO}	Pin Capacitance		10	pF	Test freq = 1 MHz

Fator de Carga:

Portas 1, 2 y 3	4 cargas TTL LS
Porta 0	Em modo barramento externo: 8 cargas TTL LS. Como porta: utilizar <i>pull-up</i> externo de 10K.

Um exemplo simples de conexão das linhas seria um sistema (ver figura 3.3) que em função de certo bit, por exemplo, o bit 0 da porta 0 (escrito como P0.0 em *assembler* ou como P0_0 em C), fará com que os LEDs colocados nos 8 pinos da porta 1 sejam acesos sequencialmente, conforme o estado daquele bit.

para ir realizando o processo de análise das seqüências em função de todas as possíveis. Por tanto, em neste curso, usaremos a linguagem C para atingir este objetivo.

Os teclados podem se de vários tipos:

- Teclados não matriciais: As teclas se acoplam diretamente às portas ou através de alguma lógica. No caso do 8051 deverá se usar um resistor de *pull-up* externo para o caso da porta P0. As outras portas têm o resistor de *pull-up* integrado (Figura 3.4). A vantagem de este tipo de teclado é o programa para atingir o objetivo 1 é fácil e simples. A desvantagem está dada pelo fato de usar uma linha da porta por cada tecla usada, o qual pode ser proibitivo quando se quer usar as portas paralelas para conexão a outros periféricos (sensores digitais, conversores A/D ou D/A, displays, etc) como é normalmente típico.

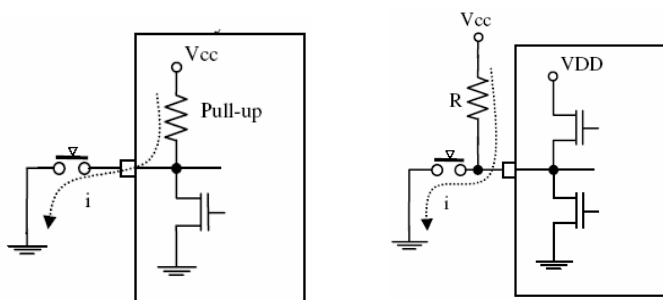


Figura 3.4 (a) Conexão de uma tecla a qualquer pino das portas P1, P2 e P3 do 8051 (O resistor de pull-up está integrado) (b) Conexão de uma tecla a um pino da porta P0 usando um resistor de pull-up externo.

- Teclados matriciais: As teclas são conectadas em forma matricial: organizadas em linhas e colunas

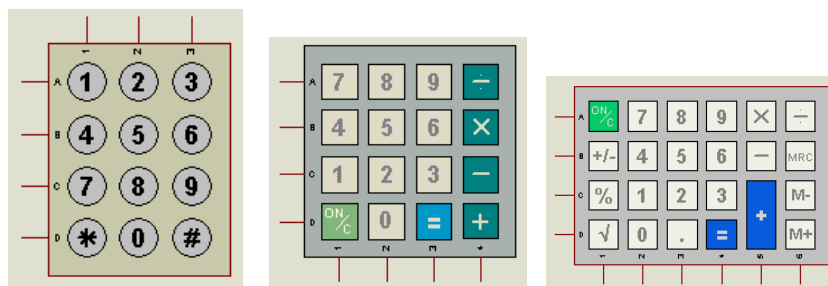


Figura 3.5 Exemplos de teclados matriciais disponíveis no simulador PROTEUS

O uso dos teclados matriciais se justifica quando o número de teclas é maior que 8, como se observa na tabela a seguir:

Tamanho do Teclado	# linhas (Teclas indep.)	# linhas (Matricial)
--------------------	--------------------------	----------------------

3 * 3	9	6
4 * 4	16	8
4 * 6	24	10

- Teclados inteligentes: São os que geram um código automaticamente por cada tecla que se pressione. Este código geralmente é o código ASCII. O mais provável é que estejam formados por um microprocessador que é o encarregado de obter o código, por tanto, a estrutura (matricial o não) dependerá das características do teclado projetado. Um exemplo típico deste tipo de teclado é o teclado do computador pessoal.

3.2.1 Atenção a teclados não matriciais.

Para entender a forma de atenção a esse tipo de teclado, consideremos o circuito mostrado na figura 3.6. Aqui se quer acender a 6 LEDs segundo se pressiona seis teclas: a tecla T1, quando é pressionada, acenderá a o LED conectado ao pino P1.2 e a tecla T6, quando é pressionada, acenderá o LED P1.7.

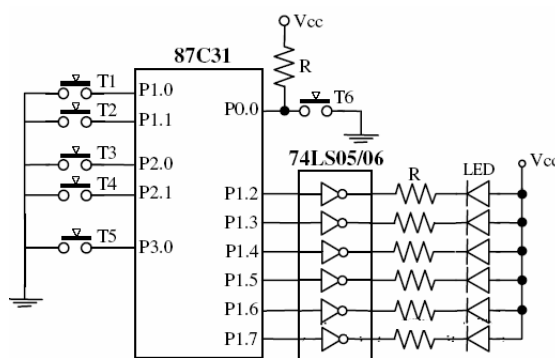


Figura 3.6 Cada inversor SN74LS05 ($I_{OL} = 25 \text{ mA}$) se usa para excitar aos LEDs devido a que a corrente suportada pela porta 1 é pequena. Também pode ser usado o CI SN 74LS06 que tem seis *buffers-drivers* (I_{OL} : 30 – 40 mA) com saídas de alta tensão.

O programa deve ler o estado das teclas e acender ao diodo LED associado a cada tecla pulsada. O diodo LED ficará aceso enquanto a tecla permaneça pulsada. O programa em linguagem *assembler* 8051 aparece a seguir:

; Programa para ler as teclas do circuito anterior.

```

ORG 0H
MOV P1, #0000 0011b ;Leds apagados. P1.0 e P1.1 como entradas
LJMP Principal
; Rotina Principal (Teste de teclas e aceso/apagado de leds)
ORG 0100H

```

Principal:

```
JNB P1.0, Tecla_T1    ; Verifica se pulsou-se T1
JNB P1.1, Tecla_T2    ; Verifica se pulsou-se T2
JNB P2.0, Tecla_T3    ; Verifica se pulsou-se T3
JNB P2.1, Tecla_T4    ; Verifica se pulsou-se T4
JNB P3.0, Tecla_T5    ; Verifica se pulsou-se T5
JNB P0.0, Tecla_T6    ; Verifica se pulsou-se T6

CLR P1.2              ; Apaga led ligado a P1.2
CLR P1.3              ; Apaga led ligado a P1.3
CLR P1.4              ; Apaga led ligado a P1.4
CLR P1.5              ; Apaga led ligado a P1.5
CLR P1.6              ; Apaga led ligado a P1.6
CLR P1.7              ; Apaga led ligado a P1.7
```

; A instrução MOV P1, #0000 0011b pode substituir a CLR P1.X,
SJMP Principal

Tecla_T1:

```
MOV P1, #0000 0111b    ; Acende led de P1.2 e apaga o resto
SJMP Principal          ; Salta a Principal
```

Tecla_T2:

```
MOV P1, #0000 1011b    ; Acende led de P1.3 e apaga o resto
SJMP Principal          ; Salta a Principal
```

Tecla_T3:

```
MOV P1, #0001 0011b    ; Acende led de P1.4 e apaga o resto
SJMP Principal          ; Salta a Principal
```

Tecla_T4:

```
MOV P1, #0010 0011b    ; Acende led de P1.5 e apaga o resto
SJMP Principal          ; Salta a Principal
```

Tecla_T5:

```
MOV P1, #0100 0011b    ; Acende led de P1.6 e apaga o resto
SJMP Principal          ; Salta a Principal
```

Tecla_T6:

```
MOV P1, #1000 0011b    ; Acende led de P1.7 e apaga o resto
SJMP Principal          ; Salta a Principal
```

3.2.2 Atenção a teclados matriciais.

Os conjuntos de teclas que constituem o teclado se interconectam entre si formando uma matriz. As linhas e colunas desta matriz se conectam às linhas das portas paralelas, como se mostra na seguinte figura 3.7. Cada tecla, quando é pressionada, provoca o curto-circuito entre a linha e a coluna onde está conectada. Tipicamente, pelas linhas são enviados os códigos de exploração (níveis lógicos) e

pelas colunas entra um grupo de bits que dependerá se a tecla está pressionada (o nível lógico que sai pela linha entrará pela coluna) o se estar liberada (entra um nível ALTO garantido pelos resistores de *pull-up* conectados a VCC). Os teclados matriciais economizam o Hardware (Diminuem os custos) a troco de aumentar o tamanho do software (Diminui a velocidade). A vezes se conectam diodos às linhas para evitar o curto circuito provocado entre duas linhas quando estejam pressionadas duas teclas da mesma coluna conectadas a duas saídas diferentes. Se a porta tem saída quase-bidirecional (caso do 8051) então não é necessário usar esses diodos.

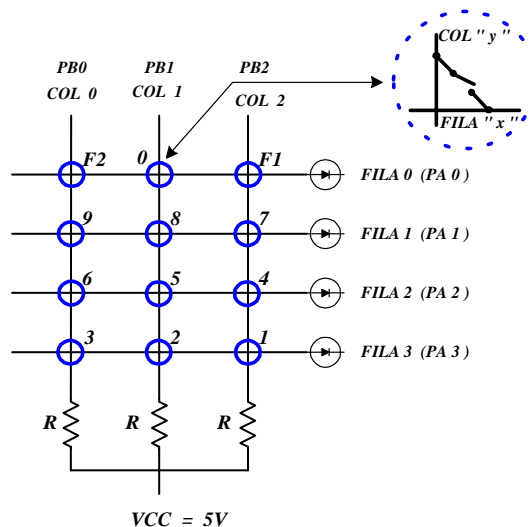


Figura 3.7 Estrutura e conexão de um teclado matricial.

A matriz conectada à porta paralela se “explora” da seguinte forma: enviam-se sinais para as linhas da matriz e se lê a informação das colunas (também denominadas, linhas de retorno). Com a informação escrita para a matriz e a lida, é possível conformar um código (código de exploração) para a tecla que foi pulsada. Neste análise, por simplicidade, se considera que é pressionada apenas uma tecla.

O mecanismo empregado para a atenção de teclados tem vários passos:

1. Esperar pela liberação do teclado (devido à pulsação da tecla anterior à atual).
2. Detectar que há tecla pulsada.
3. Se houver esperar que termine a trepidação (*debounce*). Isto se obtém com uma simples demora de 20 ms.
4. Explorar a matriz do teclado, gerando um código padrão para a tecla pulsada, por exemplo, o código ASCII.

Utilizam-se duas formas para explorar a matriz do teclado (ponto 4.):

- Envia-se um “0” pela primeira linha e se lêem todas as colunas. Se a leitura não contiver nenhum “0” (todas as colunas estão em “1”), a tecla pulsada não está nessa linha. Então se envia o “0” pela seguinte linha e se lêem todas as colunas. Repete-se o processo até encontrar uma leitura das colunas diferente do 0FFh. Conhece-se então em que fila e coluna, fica a tecla pulsada e com isto se pode conformar um código para a tecla.
- Envia-se “0” a todas as filas e se lêem todas as colunas, com o que se detecta em que coluna está a tecla pulsada. Agora se investe o processo: envia-se “0” a todas as colunas e se lêem todas as filas, detectando-se assim a fila em que está a tecla pulsada. Assim, se pode pesquisar a linha e a coluna da tecla pressionada e portanto, se pode conformar um código para essa tecla.

O mecanismo descrito, quando se pulsarem simultaneamente várias teclas, detecta só uma e se conhece na literatura inglesa como *2-key lockout*. Existem outros (conhecidos como *N-key rollover*) que permitem detectar várias teclas pulsadas simultaneamente.

Trepidação de contatos: Quando se fecha uma tecla (chave), os contatos não se detêm imediatamente. Em virtude da inércia da parte móvel da chave, ocorre uma oscilação mecânica ao redor da posição de contato o qual se conhece como Trepidação de contatos.

A frequência destas trepidações e o tempo de amortecimento variam em função das características próprias do tipo de tecla considerada. Este tempo oscila entre 0,5 ms e 1 ms. Para eliminar essas trepidações, tipicamente se introduz um retardo no sinal de validação de 10 ms a 20 ms.

Exercício Resolvido: Atenção a um teclado de 16 teclas pelo porto P1.

Se dispõe de um sistema com um microprocessador 80C31 que só tem livre a porta 1 devido a que as portas 0 e 2 são os barramentos externos que neste caso são imprescindíveis para (pelo menos) conectar a memória de programa externa e a porta 3 é usada para usar seus sinais de controle (o mais provável é que os outros pinos estejam sendo usados com suas funções específicas: Timer e interrupções). Projetar um hardware para conectar um teclado matricial de 4 linhas e 3 colunas, e que permita que as linhas da porta P1 possam ser compartilhadas para ser conectadas a um *latch* octal com o objetivo de acender oito LEDs (cujo programa não será abordado pois não é objetivo deste exercício). As 12 teclas estão rotuladas como os 10 números decimais mais duas teclas adicionais com os símbolos # e *).

a) Hardware:

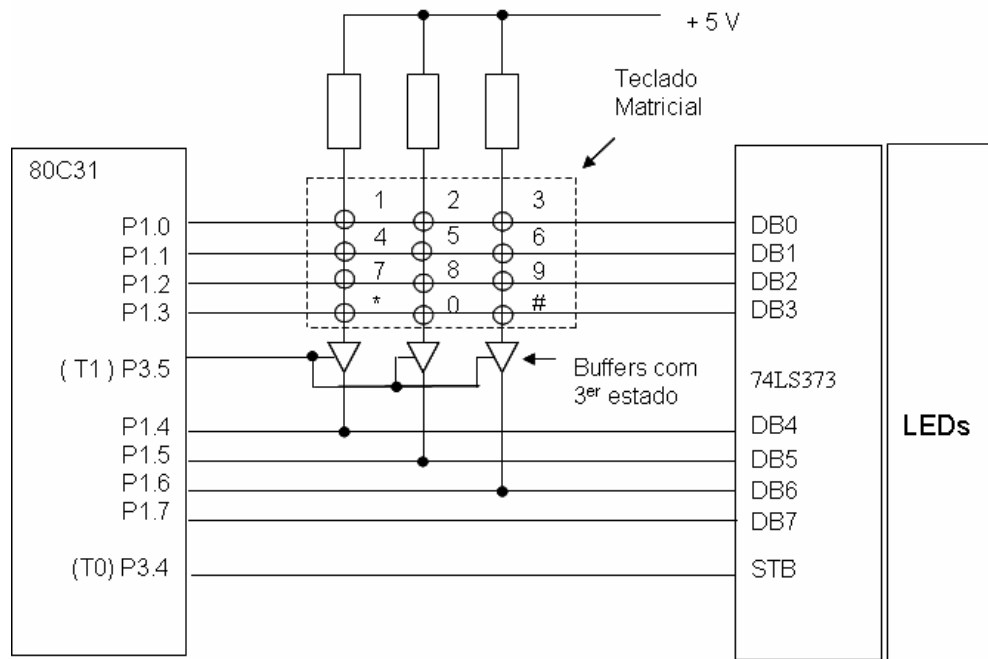
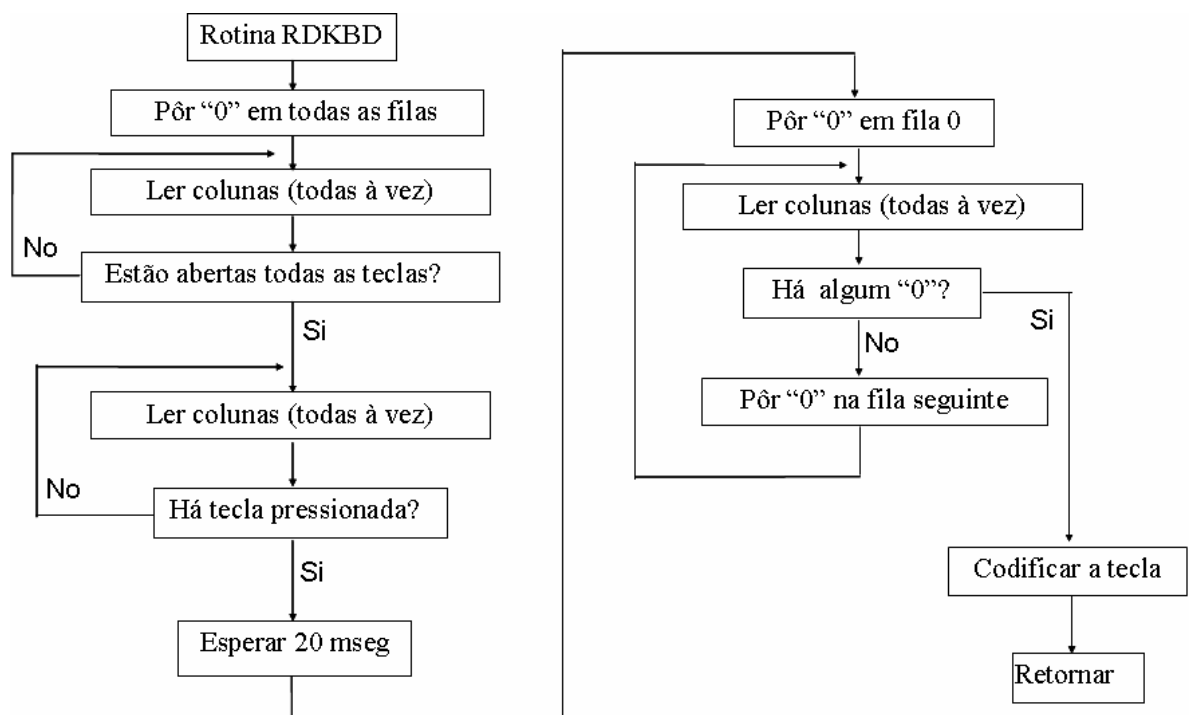


Figura 3.8 Uso simultâneo da porta 1 para atender a um teclado matricial e oito LEDs através de um LATCH 74LS373. Quando se usa memórias externas. A única porta disponível para conectar periféricos externos é a porta 1.

b) Software:



O bloco "Codificar a tecla" pode, por exemplo, devolver um "código de exploração" para cada tecla, que pode ser, por exemplo:

0	0	0	0	C	C	F	F
				Coluna		Linha	

Tecla	Código de exploração									Código ASCII
	D7	D6	D5	D4	D3	D2	D1	D0	HEX	
1	x			0				0	EEh	31h
2	x		0					0	DEh	32h
3	x	0						0	BEh	33h
4	x			0			0		EDh	34h
5	x		0				0		DDh	35h
6	x	0					0		BDh	36h
7	x			0		0			EBh	37h
8	x		0			0			DBh	38h
9	x	0				0			BBh	39h
*				0	0				E7h	2Ah
0			0		0				D7h	30h
#		0			0				B7h	23h

A seguinte rotina realiza a exploração do teclado, cada vez que ela seja chamada desde um programa principal, mediante a instrução **CALL rdkbd**.

;rdkbd: Esta rotina espera por a pulsação correta de uma tecla y volta
;o código ASCII da mesma no ACC. Esta rotina trabalha com o esquema
;do teclado de 16 teclas conectado à porta P1.

rdkbd:

push 0

rdkbd5:

mov p1, #0f0h

setb p3.5

mov a, p1

clr p3.5

cjne a, #0f0h, rdkbd5

kbd10:

setb p3.5

mov a, p1

clr p3.5

cjne a, #0f0h, kbd20

jmp kbd10

kbd20:

call demora

mov a, #0feh

kbd30:

```
mov p1, a
setb p3.5
mov r0, p1
clr p3.5
orl 0, #80h ; força bit 7 a 1
cjne a, 0, kbd40 ; em r0 o código de exploração
rl a
cjne a, #0efh, kbd30 ; saltar pois falta por explorar
jmp rdkbd5 ; saltar pois há pulsação falsa
```

kbd40:

```
mov a, r0
call vertab1 ;explorar a tabela de códigos de exploração
cjne a, #0ffh, kbd50
jmp rdkbd5 ;saltar se pulsação múltiplo
```

kbd50:

```
call vertab2 ;explorar tabela de códigos ASCII
pop 0
ret
```

;vertab1: esta rotina busca em uma tabela por conteúdo e volta o número
;de ordem do elemento. A tabela termina em 00. Se o dado não se encontra
;na tabela, se volta 0FFh.

;entradas: (ACC): dado a encontrar

;saídas: (ACC): número de ordem do dado, se é procurado. Em caso de não
;procurar-se, então ACC=FFh.

vertab1:

```
push 0 ;Salva R0
push 1 ;Salva R1
push dpl ;Salva DP
push dph
mov r0, a
mov r1, #0ffh ; r1 é o contador de elementos
mov dptr, #tabla1
```

vert01:

```
inc r1
mov a, r1
movc a, @a+dptr
jz vert02 ; Se encontrar um 0, sair
cjne a, 0, vert01 ; Compara com código de entrada
mov a, r1
```

```

    jmp vert03
vert02:
    mov a, #0ffh
vert03:
    pop dph
    pop dpl
    pop 1
    pop 0
    ret
;Tabela de códigos de exploração:
tabla1:
    db 0eeh, 0deh, 0beh, 0edh, 0ddh, 0bdh
    db 0ebh, 0dbh, 0bbh, 0e7h, 0d7h, 0b7h
    db 0h
;vertab2: Esta rotina procura em uma tabela de bytes por o numero de ordem
;do elemento, voltando seu valor.
;entradas: (ACC): numero de ordem do elemento.
;saídas: (ACC): valor do elemento.
vertab2:
    push dpl
    push dph
    mov dptr, #tabla2
    movc a, @a + dptr
    pop dph
    pop dpl
    ret
;Tabela de códigos ASCII:
tabla2:
    db 31h, 32h, 33h
    db 34h, 35h, 36h
    db 37h, 38h, 39h
    db 2ah, 30h, 23h
;Rotina de demora
demora:
    mov r0, #0
demora1:
    djnz r0, demora1
    ret
end

```

3.3 Atenção a displays

Os displays usados nos painéis dos instrumentos podem ser classificados nos seguintes tipos:

1.- **Anunciadores:** luzes on/off, por exemplo diodos LEDs de varias cores.

2.- **Numéricos:** Concebidos para mostrar números, por exemplo, displays de 7 segmentos (Figura 3.9).

Os mostradores de informação numérica se ofertam de diferentes formas mas as mais típicas som Ânodo comum e Catodo comum. Normalmente têm um ponto decimal à esquerda e/ou à direita de cada dígito. Ofertam-se mostradores onde se integram o LEDs junto com seu decodificador.

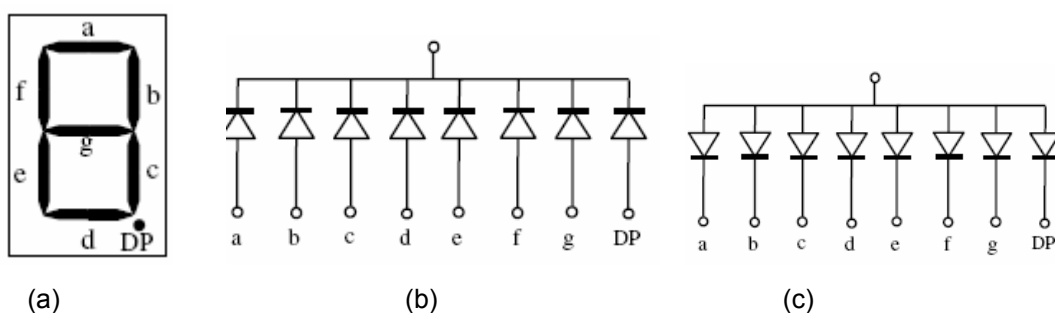


Figura 3.9 (a) Display numérico de 7 segmentos e Tipos: (b) Cátodo comum (c) Ânodo comum.

3.- **Alfanuméricos:** Concebidos para mostrar números e letras (Figura 3.10). Entre os exemplos deste tipo de displays, displays de 16 ou 18 segmentos e os displays de cristal líquido (LCD).

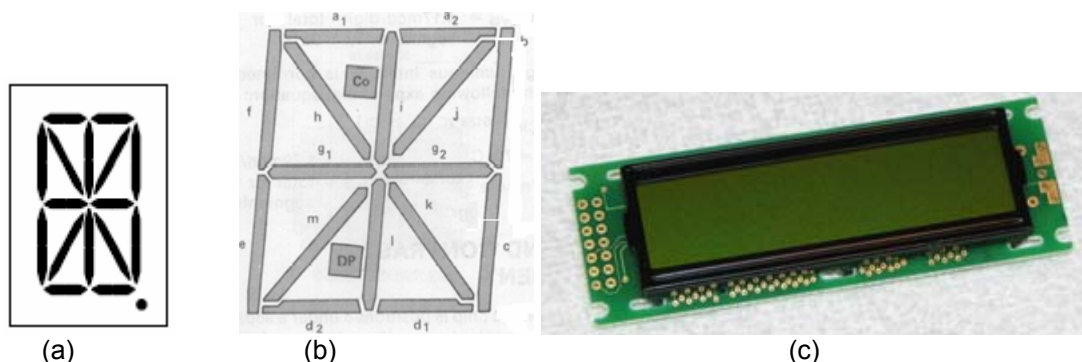


Figura 3.10 Tipos de displays alfanuméricos: (a) 16 segmentos (b) 18 segmentos e (c) LCDs.

4.- **Gráficos:** Concebidos para mostrar números, letras, e desenhos gráficos de certa complexidade. (Figura 3.11).

- Matrizes alfanuméricas
- Displays LCD gráficos.
- Telas de TV (com LCD ou com Tubo de Raios Catódicos)

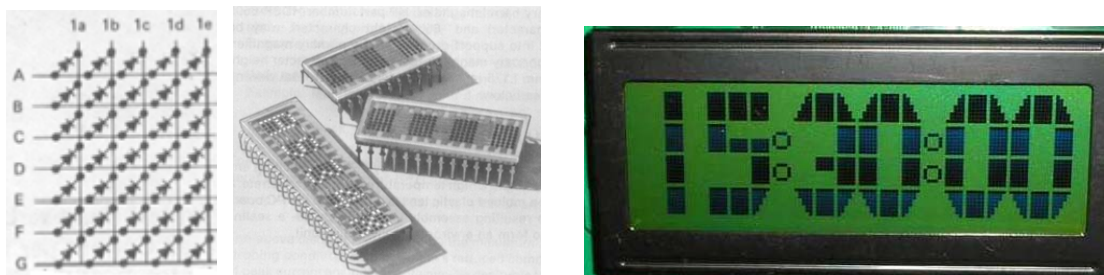


Figura 3.11 Matriz gráfica com LEDs (a) Estrutura (b) Display gráfico de 5 × 7 a estado sólido (c) Display gráfico a cristal líquido.

O ponto de operação típico de um diodo LED é:

- Tensão de operação (V_{LED}) entre 1,6V y 2,4V
- Corrente de operação (I_{LED}): entre 3 mA (diodos de alta eficiência) y 10 mA.

Os valores típicos para nossos projetos corresponderão aos do diodo genérico seguinte: $V_{LED} = 2V$ e $I_{LED} = 10\text{ mA}$.

O processo de atenção a display, também chamado refrescamento de display, consiste em transferir a informação de uma memória de display, chamada RAM de display (DDRAM) ao mostrador, como se mostra na figura 3.12. Dependendo do tipo de display usado, assim será o formato do código escrito na DDRAM. Na Figura 3.12 se mostra um possível formato quando se trata de displays de 7 segmentos. Cada bit armazena o estado aceso/apagado para cada segmento do display. O nível (ALTO ou BAIXO) usado para acender o segmento depende do hardware empregado. Normalmente, se usa um bit para armazenar o estado do ponto decimal (**dp**: *decimal point*, em inglês).

Como a maioria dos mostradores estão concebidos para escrever informação, em muita literatura são chamados como memória de só escrita (Dispositivos W.O.M: *Write Only Memory*, em inglês). Isto não é uma lei, pois a maioria dos displays LCD tem a possibilidade de ser lidos também.

A transferência pode ser a traves de alguns dos mecanismos de Entrada Saída dos Microprocessadores que são:

- Atenção diretamente durante a execução do programa principal (OBS: O programa principal é o que executa o processador depois de um RESET).
- Interrupção
- Acesso direto a memória (DMA)

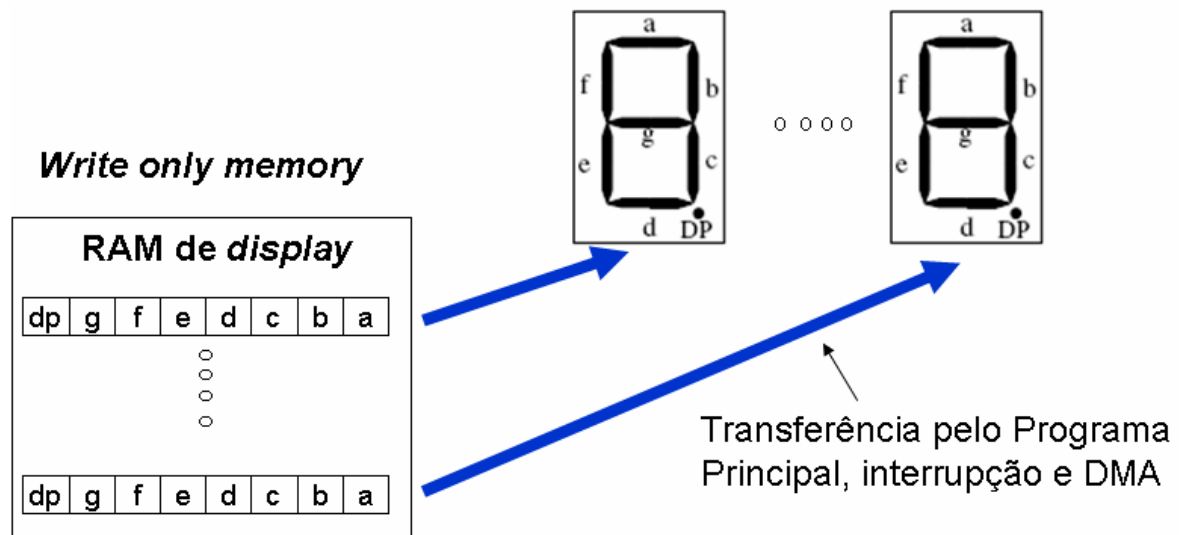


Figura 3.12 Processo de refresco do display: transferência de informação desde a RAM de display até os dispositivos usados como displays.

O processo de transferência para o display se realiza mediante um suporte de hardware (interfase de display) e software (programas usados para: (a) converter do formato de entrada (Exemplo: BCD, ASCII, etc.) ao formato de saída para o display (Exemplo: código de 7 segmentos, sinais de vídeo, etc.). A seguir estudaremos esses métodos para dois casos particulares: displays numéricos de estado sólido (7, 16 e 18 segmentos) e display de cristal líquido (próximo apartado).

3.3.1 Atenção a displays de 7 segmentos: Hardware:

As interfases de conexão dos displays com as portas dos microprocessadores podem ser:

- Não multiplexada, continua ou a DC: Se usa uma linha de porta por cada segmento a acender. A informação de saída (no formato adequado) para o display é mantida na porta até que deva mudar a informação a ser mostrada. **Vantagem:** o programa é muito simples. **Desvantagem:** Requerem-se muitas linhas de portas (uma linha por segmento a mostrar) quando aumenta o número de displays. Com decodificadores (SN7447, SN7448) se podem economizar linhas, com o preço de aumentar o número de chips no circuito. Na figura 3.13 se mostra um exemplo com um display de 7 segmentos mais o ponto decimal.

Os níveis lógicos na porta de saída que provocam o aceso e apagado dos segmentos são terminais da porta de saída por onde sai a

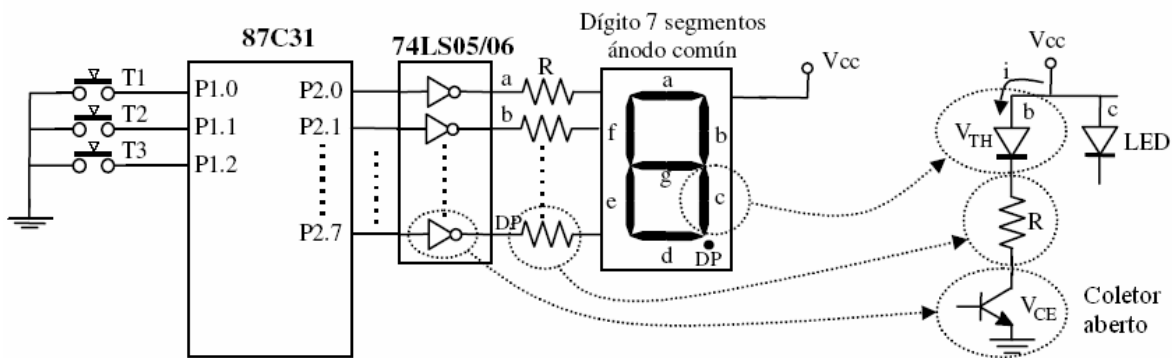


Figura 3.13 Exemplo de Interface não multiplexada com um microcontrolador 87C31. Os inversores são de coletor aberto para absorver a corrente de cada LED.

- Multiplexada ou descontínua: Se compartilham as mesmas linhas de uma porta para segmentos homólogos de diversos displays (barramento de segmentos). A informação a ser mostrada num display dado, é mantida só durante o tempo em que se habilita a circulação de corrente por o terminal comum desse display garantindo que o resto de displays fique apagado. Posteriormente o processo se repete para outro display. Em um instante de tempo dado, apenas um display (do conjunto) fica aceso e o resto fica apagado. **Vantagem:** economiza linhas de portas. **Desvantagem:** A complexidade do programa é maior.

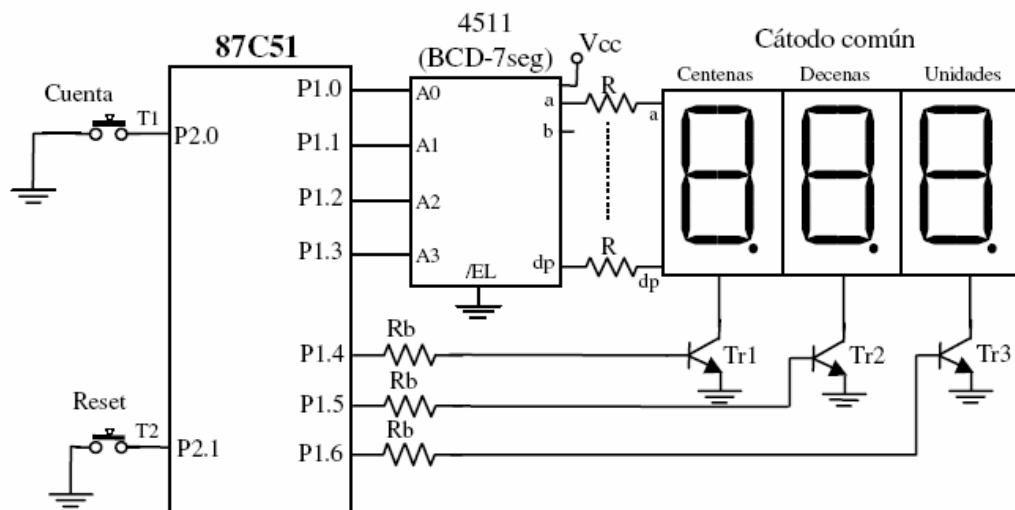


Figura 3.14 Exemplo de Interface multiplexada com um microcontrolador 8751. Os transistores garantem o de-multiplexado da informação.

Na seguinte figura 3.15 se mostra o diagrama de tempo durante dos ciclos de completos de refrescamento dos três displays considerando que vai-se realizar por interrupção. Quando chega a primeira solicitação de interrupção (pulso 1), se escreve nos bits P1.0 - P1.3, o código BCD do dígito a ser mostrado no display das centenas e posteriormente se envia um nível ALTO pelo terminal P1.4 para permitir a circulação de corrente por o transistor Tr1. Neste intervalo, os transistores Tr2 e

Tr3 ficam cortados (são enviados níveis BAIXO pelas linhas P1.5 e P1.6). Quando chega a segunda solicitação de interrupção se repetem as ações, só que neste caso o código BCD é enviado para o display das dezenas ao mesmo tempo em que só é habilitado o transistor Tr2. Na frequência de refrescamento f_R (igual ao inverso do período de refrescamento T_R) deverá estar entre 40 e 200 Hz, para evitar piscos. O período de interrupção é igual ao período de refrescamento dividido pelo número de displays usados (3 neste caso).

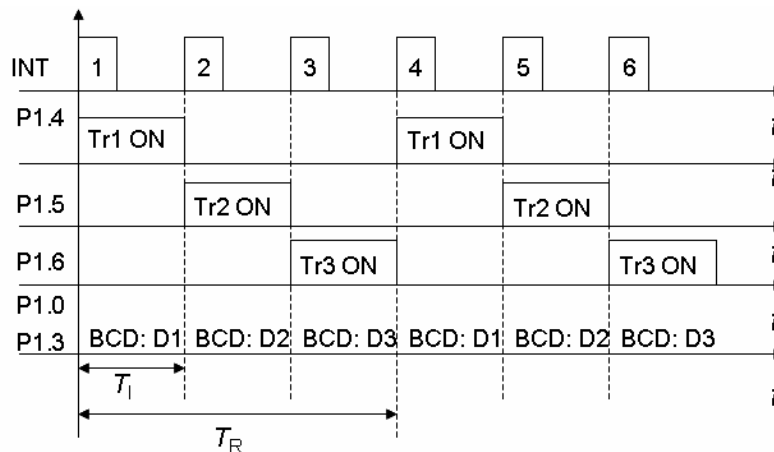


Figura 3.15 Formas de ondas nos terminais da porta de saída. T_I é o Período do sinal que solicita a interrupção e T_R é o período de refrescamento que garante que a informação não pisque no display.

A diferencia do método não multiplexado, onde a corrente por cada segmento é constante, no método multiplexado, a corrente por cada segmento tem forma de pulso de corrente (ver figura 3.16). Por tanto, para garantir um nível de corrente médio $I_{AVG} = 10 \text{ mA}$, então a amplitude do pulso de corrente deverá ser maior. Calculemos matematicamente a relação entre a corrente media e a corrente pico por segmento I_{SEG} .

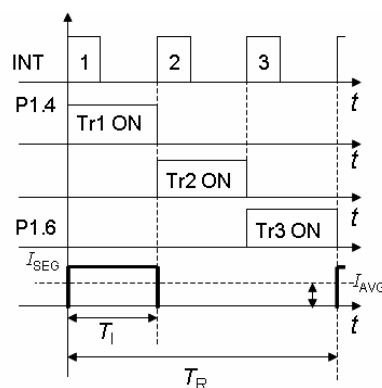


Figura 3.16 Formas de ondas de tensões nos terminais da porta e de corrente nos segmentos do display durante o refrescamento multiplexado.

O valor médio da corrente (valor de DC) de um pulso quadrado de duração T_I e período T_R , por definição é:

$$I_{AVG} = \frac{1}{T_R} \int_0^{T_R} i dt = \frac{1}{T_R} \int_0^{T_I} I_{SEG} dt = I_{SEG} \frac{T_I}{T_R}$$

Por tanto, $I_{SEG} = I_{AVG} \frac{T_R}{T_I}$. Neste caso, temos que: $I_{SEG} = 3 \times I_{AVG} = 30 \text{ mA}$

Devido à propriedade básica do GaAsP com que se constroem os LEDs que compõem os segmentos do mostrador de 7 ou mais segmentos, a eficiência luminosa (intensidade luminosa por unidade de corrente) incrementa-se com a amplitude dos pulsos de corrente que se aplicam a um segmento dado quando se trabalha de forma descontínua, isto é um dos fatores positivos deste tipo de excitação de forma descontínua ou multiplexada com respeito à contínua ou a DC. Do ponto de vista prático se demonstrou que excitando a um LED de forma multiplexada se obtém igual intensidade luminosa com um valor de corrente médio que seja aproximadamente igual ao 80% da corrente (I_{LED}) constante (média ou DC) que circularia por este se permanecesse sempre iluminado.

Segundo o anteriormente expressado, a corrente média $I_{AVG} = 0.8 I_{LED}$.

Neste exemplo, em particular, $I_{SEG} = 3 \times I_{AVG} = 3 \times 0,8 \times I_{AVG} = 24 \text{ mA}$, ou seja, com um nível de corrente de 24 mA (6 mA menos) se conseguiu a mesma intensidade luminosa. Com o valor calculado de I_{SEG} pode se obter o valor do resistor R que limita a corrente num segmento (figura 3.17).

Para o circuito de saída do decodificador, segundo LKV, temos:

$$V_{CC} - V_{OH} - I_{SEG} - V_{LED} - V_{OL} = 0$$

Para o cálculo do transistor saturado, temos que:

$$I_C = 8 I_{SEG}, V_{BE(SAT)} = 0,75V, I_B = \frac{I_C}{\sigma h_{FE}} (\sigma = 0,1 - 0,2)$$

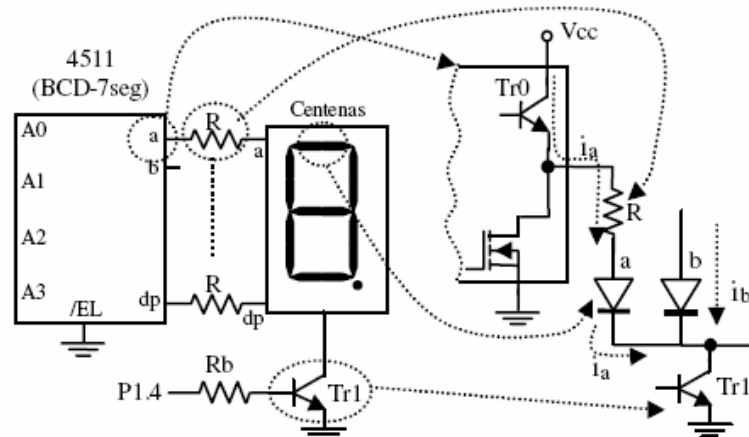


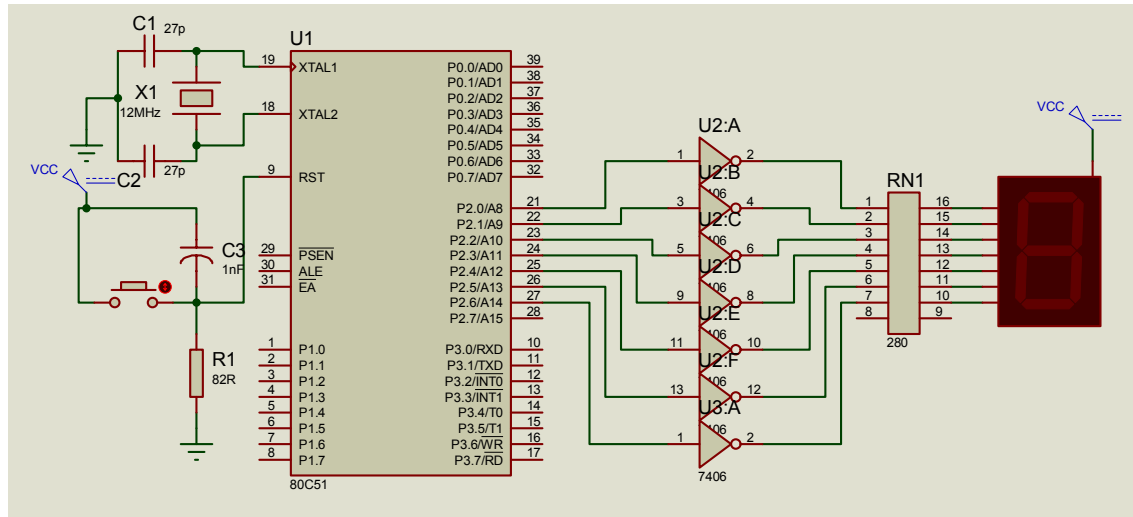
Figura 3.17 Circuito equivalente de saída do decodificador BCD-7segmentos 4511.

$$V_{OH} = 4,12 \text{ V @ } I_{OH} = 10 \text{ mA.}$$

Exemplo:

Projetar o hardware e o software (na linguagem C) para fazer um contador de 0 a 9 empregando uma interfase não multiplexada.

Hardware:

**Software:**

```
//Contador 0 a 9
#include <8051.h>

unsigned char converte(unsigned char i)
{
    unsigned char b;

    if (i==0)
        b = 0x3f; // 0

    if (i==1)
        b = 0x06; // 1

    if (i==2)
        b = 0x5b; // 2

    if (i==3)
        b = 0x4f; // 3

    if (i==4)
        b = 0x66; // 4

    if (i==5)
        b = 0x6d; // 5

    if (i==6)
        b = 0x7c; // 6

    if (i==7)
        b = 0x07; // 7
}
```

```

    if(i==8)
        b = 0x7f; //8
    if(i==9)
        b = 0x67; //9
    return b;
}
void delay(void){
    unsigned int x;
    for(x=0;x<60000;x++);
    for(x=0;x<60000;x++);
    for(x=0;x<60000;x++);
}
//Aqui começa o programa principal
void main (void)
{
    unsigned char mask = 1;
    unsigned char i,x;
    for(;;){
        x=0;
        for(i=0;i < 10 ;i++)
        {
            P2 = converte(i);
            delay();
        }
    }
}

```

3.4 Atenção a displays LCD

Os módulos LCD são interfaces de saída muito útil em sistemas microprocessados. Estes módulos podem ser alfanuméricos e gráficos.

Os módulos de mostradores LCD alfanuméricos se compõem de uma tela de cristal líquido (LCD) que consta de uma matriz de 16, 32, 40 ou 80 caracteres de 5 x 7 pixels e um controlador interno para sua excitação e controle. O controlador libera ao microprocessador de realizar essas funções o qual aumenta a eficiência que ele tem na realização de suas funções. Por exemplo, no modelo Hitachi o controlador é o HD44780, que é o mais utilizado para este fim. Entre as telas LCD que usam este controlador se encontram os modelos LM054, LM016L, etc.

Os módulos LCD gráficos são encontrados com resoluções de 122x32, 128x64, 240x64 e 240x128 dots pixel, e geralmente estão disponíveis com 20 pinos

para conexão. Os LCD comuns (tipo caracter) são especificados em número de linhas por colunas.

As características gerais de um módulo LCD são:

- Tela de caracteres ASCII, además dos caracteres Kanji e griegos.
- Deslocamento dos caracteres para a esquerda o direita.
- Proporciona o endereço da posição absoluta ou relativa do caracter.
- Memória de 40 caracteres por linha da tela.
- Movimento do cursor e mudança de seu forma.
- Permite que o usuario possa programar oito caracteres.
- Conexão a um processador usando uma interface de 4 ou 8 bits.

3.4.1 Funcionamento dos displays LCD

Os dispositivos de cristal líquido funcionam segundo as características seguintes (figura 3.18):

- Requerem uma tensão entre 2 e 3 volts entre terminais.
- Não se podem alimentar com corrente direta, pois se deterioram com uma componente de DC major de 50 mV.
- A freqüência de trabalho está entre 30 e 150 Hz.
- Utilizar dispositivos CMOS para excitá-los:

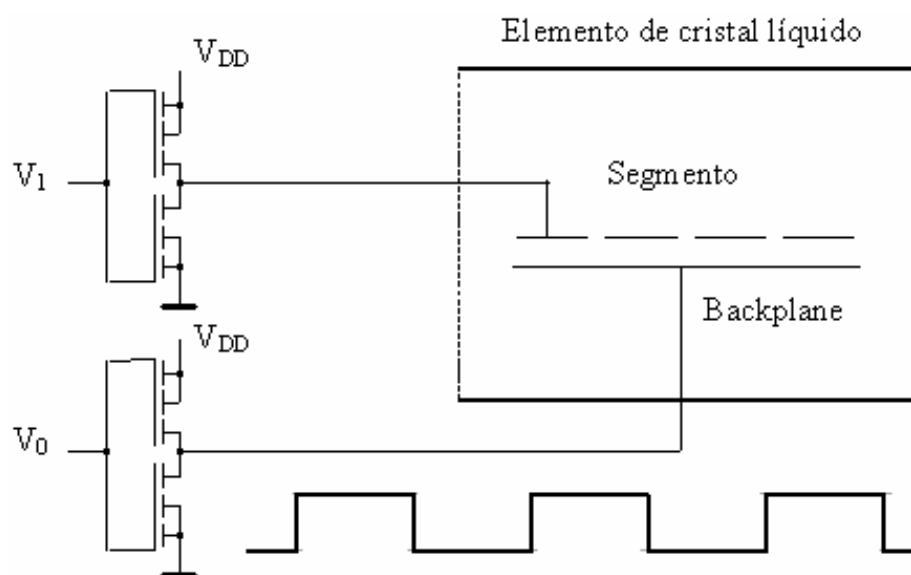


Figura 3.18 Excitação de um display LCD

Observe que:

- Se $V_1 = V_0$, o segmento é invisível.
- Se $V_1 = \# V_0$ (ou seja, $V_1 = \text{not } V_0$), o segmento é visível.
- Sempre se cumpre que a componente de DC é zero.

O esquema que garante a excitação (em fase ou contra fase) no circuito anterior usa uma porta OR exclusivo como se mostra na figura 3.19.

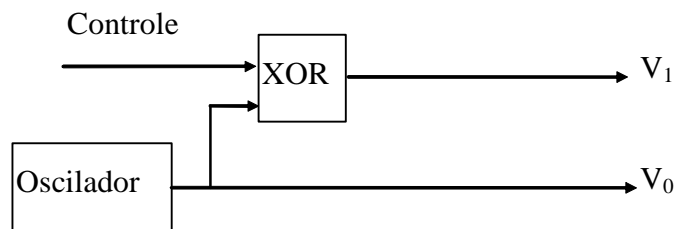


Figura 3.19 Esquema para garantir a excitação em fase e em contra-fase

3.4.1 O módulo de cristal líquido (LCM, *Liquid Crystal Module*) LM016L.

3.4.1.1 Características gerais e estrutura interna

O LCM tem as seguintes características gerais:

- Possibilidade de acoplar-se a microprocessadores de 4 ou 8 bits.
- Display LCD com 2 linhas de até 16 caracteres cada uma.
- Armazenamento de 80 caracteres em uma RAM interna de 80 bytes.
- Gerador de caracteres em ROM interna com:
 - ⇒ 160 caracteres do tipo 5 * 7 pontos.
 - ⇒ 32 caracteres do tipo 5 * 10 pontos.
- Gerador de caracteres em RAM interna, para definir novos caracteres pelo usuário, com:
 - ⇒ 8 caracteres do tipo 5 * 7 pontos.
 - ⇒ 4 caracteres do tipo 5 * 10 pontos.
- Muito baixo consumo (tecnologia CMOS). Pode alimentar-se com baterias.
- Circuito do RESET interno automático no aceso.
- Amplo repertório de comandos:
 - ⇒ apagado de display,
 - ⇒ piscada de caracteres e do cursor,

- ⇒ deslocamentos do cursor e do display,
- ⇒ apagado/aceso do cursor e o display,
- ⇒ outros

O LCM LM016L (Figura 3.20) consta dos seguintes elementos:

- Display de cristal líquido (LCD). Permite mostrar 2 linhas de 16 caracteres cada uma. Cada caracter se representa em uma matriz de 8 x 5.
- Um *driver* HD44100 (do Hitachi). Realiza as funções inerentes à conversão de níveis de voltagens adequadas para a excitação do LCD, multiplexado, temporização, etc. Pode dirigir 8 caracteres de um LCD de uma linha ou 16 caracteres em um LCD de 2 linhas.
- Controlador HD44780 (Hitachi). Este controlador do LCD pode excitar 80 caracteres alfanuméricos. Para isso possui uma RAM interna de dados de 80 bytes, assim como um gerador de caracteres em ROM e outro em RAM, este último para a definição pelo usuário, de outros caracteres que não estejam na ROM interna. Executa uma série de comandos para a manipulação do LCD.
-

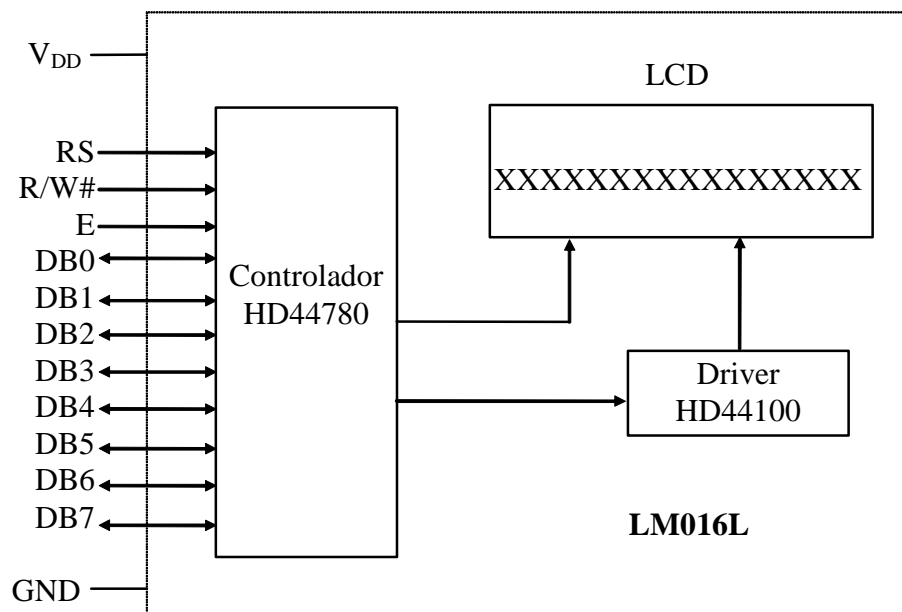


Figura 3.20 Estrutura interna do modulo de cristal liquido de Hitachi LM016

O HD44780 pode excitar diretamente (faz as vezes de *driver*) 8 caracteres em uma linha ou 16 caracteres em 2 linhas. O *driver* HD44100 pode excitar a mesma quantidade de caracteres em igual número de linhas. Assim, podem analisar-se diferentes combinações:

- Um LCD de 8 x 1 caracteres necessita um controlador HD44780 e não requer de *driver* HD44100.

- Um LCD de 16 x 2 caracteres necessita um controlador HD44780 e um driver HD44100 (é o caso do LM016L).
- Um LCD de 40 x 2 caracteres necessita um controlador HD44780 e 4 drivers HD44100.

3.4.1.2 Terminais do módulo de cristal líquido

RS: *Register Select.* (Control/#Dados): Através desta linha se indica ao LCD se o byte que se enviar por DB0-DB7 é de Controle (Nível ALTO) ou de Dados (Nível BAIXO).

R/W#: *Read/Write.* (Leitura /#Escrita). Um "1" indica leitura e um "0" escrita.

E: *Enable.* (Habilitação). Linha de habilitação do dispositivo: Um "1" habilita e um "0" inabilita o dispositivo.

DB0 a DB7: Barramento de dados. Por estas linhas transitam os comandos e dados em ambas as direções (do microprocessador ao LCM e viceversa).

Formas de onda

Para escrever no LCM devem ser gerados a seqüência de sinais mostrados na carta de tempo da figura 3.21.

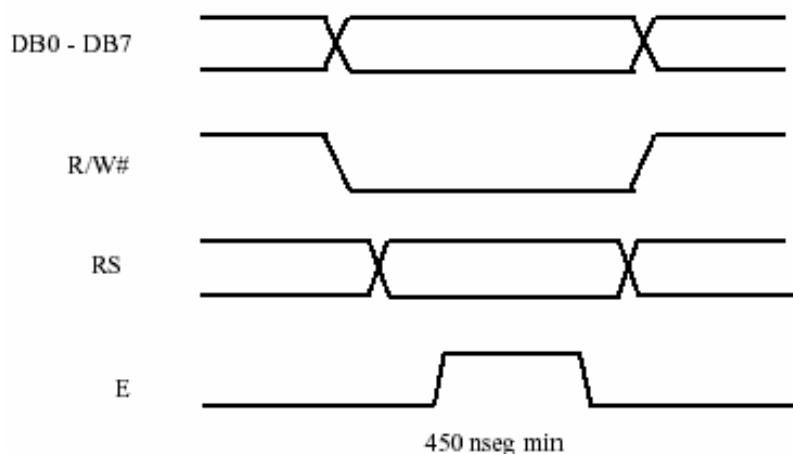


Figura 3.21 Diagrama de tempos para la escritura en el Módulo de Cristal Líquido.

Assim como em um rádio relógio todo módulo LCD permite um ajuste na intensidade da luz emitida ou ajuste de contraste, isto é possível variando-se a tensão no pino 3. A Figura 3.22 mostra um circuito típico e recomendado pela

maioria dos fabricantes para efetuar este ajuste. Alguns fabricantes recomenda o uso de um resistor de 4K7 em série com o potenciômetro de 10K.

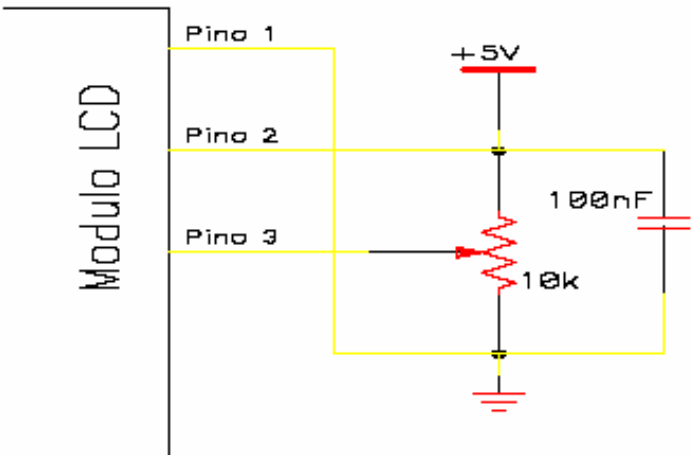


Figura 3.22 Detalhe do controle de contraste do módulo LCD

3.4.1.3 Conexão do LCM ao microprocessador 8051

Para conectar um LCM ao microprocessador 8051 podem-se utilizar as portas paralelas do 8051. Mediante o software adequado, geram-se os sinais na sequência correta. A figura 3.23 mostra uma possível conexão:

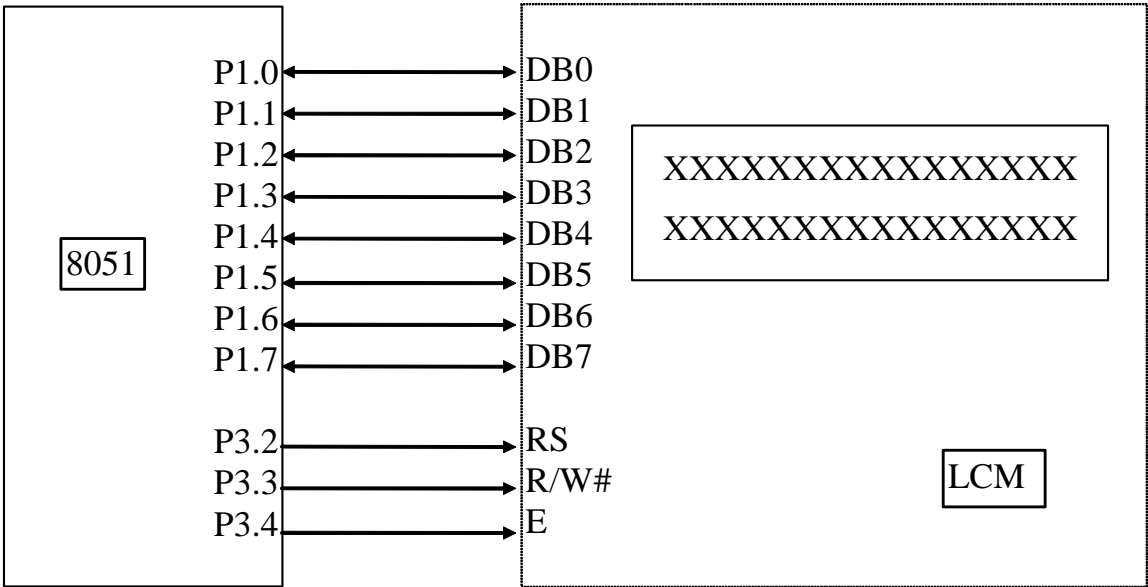


Figura 3.23 Interfase direta de 8 bits do modulo LM016 com duas portas do 8051

3.4.1.4 Comandos de programação do LCM

Comandos	R S	R/W	D B 7	D B 6	D B 5	D B 4	D B 3	D B 2	D B 1	D B 0	DESCRIÇÃO
Limpa o Display	0	0	0	0	0	0	0	0	0	1	- Limpa todo o display e retorna o cursor para a primeira posição da primeira linha (endereço 0)
Cursor a posição inicial	0	0	0	0	0	0	0	0	1	X	Retorna o cursor para a 1ª coluna da 1ª Linha A DD RAM não se afeita.
Definir modo de entrada	0	0	0	0	0	0	0	1	I/ D	S	- Estabelece o sentido de deslocamento do cursor (I/D=0 p/ esquerda, I/D=1 p/ direita) - Estabelece se a mensagem deve ou não ser deslocada com a entrada de um novo caracter (S=1 SIM, I/D=1 p/ direita) - Esta instrução tem efeito somente durante a leitura e escrita de dados.
Controle do display	0	0	0	0	0	0	1	D	C	B	-Liga (D=1) ou desliga display (D=0) -Liga(C=1) ou desliga cursor (C=0) -Cursor Piscante (B=1) se C=1
Deslocar cursor ou mensagem	0	0	0	0	0	1	C	R	X	X	-Desloca o cursor (C=0) ou a mensagem (C=1) para a Direita se (R=1) ou esquerda se (R=0). - Desloca sem alterar o conteúdo da DDRAM
Seleção de Função	0	0	0	0	1	L	N	F	X	X	-Comunicação do módulo com 8 bits(L=1) ou 4 bits(L=0) -Número de linhas: 1 (N=0) e 2 ou mais (N=1) -Matriz do caracter: 5x7(F=0) ou 5x10 (F=1) - Esta instrução deve ser ativada durante a inicialização
Especificar endereço para a CGRAM	0	0	0	1	Endereço da CGRAM						-Fixa o endereço na CGRAM para posteriormente enviar ou ler o dado (byte) Depois deste comando os dados escritos ó lidos vão parar ou provêm desta RAM.
Especificar endereço para a DDRAM	0	0	1	Endereço da DDRAM						-Fixa o enderço na DDRAM para posteriormente enviar ou ler o dado (byte) Selecciona la dirección de la DD RAM. Depois deste comando os dados escritos ó lidos vão parar ou provêm desta RAM.	
Leitura do <i>Flag Busy</i> e do contador de endereços	0	1	B F	Conteúdo do contador de endereços (AC)						-Lê o conteúdo do contador de endereços (AC) e o BF. O BF (bit 7) indica se a última operação foi concluída (BF=0 concluída) ou está em execução (BF=1).	
Escrita de dados na RAM (DD ó CG)	1	0	Dado a ser gravado no LCD						- Grava o byte presente nos pinos de dados no local apontado pelo contador de endereços (posição do cursor). Depende de qual foi a último endereço selecionada, se a DD RAM ou a CG RAM.		
Leitura de dados da RAM (DD ó CG)	1	1	Dado lido do módulo.						Idem - Lê o byte no local apontado pelo contador de endereços (posição do cursor)		

O comando de deslocamento da mensagem se implementa deslocando o conteúdo das localizações da RAM de display da forma mostrada na seguinte figura. Quando se trabalha com um modulo LM016 em que apenas se mostra uma janela

de 16 caracteres em cada linha então na tela se observará os conteúdos da RAM que fiquem dentro desta janela deslocante.

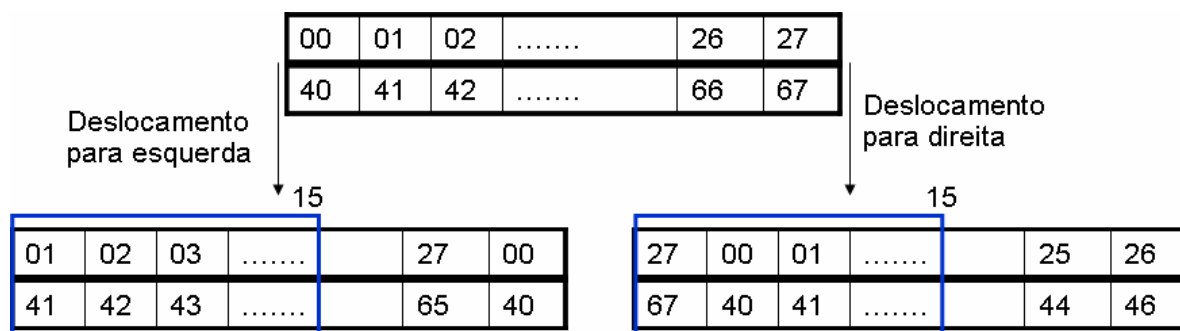


Figura 3.24 Ilustração do efeito do comando de deslocamento da mensagem no caso particular do modulo LM016 que mostra apenas 16 caracteres.

A seguir alguns dos comandos mais comuns:

Seleção de Função Instrução

1 linha 5×7 (8 bits)	30 H
2 linha 5×7 (8 bits)	38 H
1 linha 5×10 (8 bits)	34 H
1 linha 5×7 (4 bits)	20 H
2 linha 5×7 (4 bits)	28 H
1 linha 5×10 (4 bits)	24 H

Display aceso c/ cursor fixo	0E H
Display aceso c/ cursor intermitente	0F H
Display aceso sem cursor	0C H
Display apagado	08 H

Modo de Operação

Instrução

Escreve deslocando a mensagem para esquerda (cursos fixo)	07 H
Escreve deslocando a mensagem para direita (cursos fixo)	0F H
Escreve deslocando o cursor para a direita	06 H
Escreve deslocando o cursor para a esquerda	04 H

Modo de Operação

Instrução

Limpa display e retorna o cursor para o início	01 H
Retorna o cursor para o início (sem alterar a DDRAM)	02 H
Desloca somente o cursor para a direita	14 H
Desloca somente o cursor para a esquerda	10 H
Desloca o cursor + mensagem para a direita	1C H
Desloca o cursor + mensagem para a esquerda	18 H
Desloca o cursor para posição inicial da segunda linha	C0 H
Desloca o cursor para posição inicial da primeira linha	80 H

CGRAM (caracteres especiais)

Instrução

Endereço inicial para construir caracteres especiais	40 H
Para escrever o primeiro caracter (previamente construídos)	00 H
Para escrever o último caracter (previamente construídos)	07 H

3.4.2 Inicialização do módulo LCM LM016.

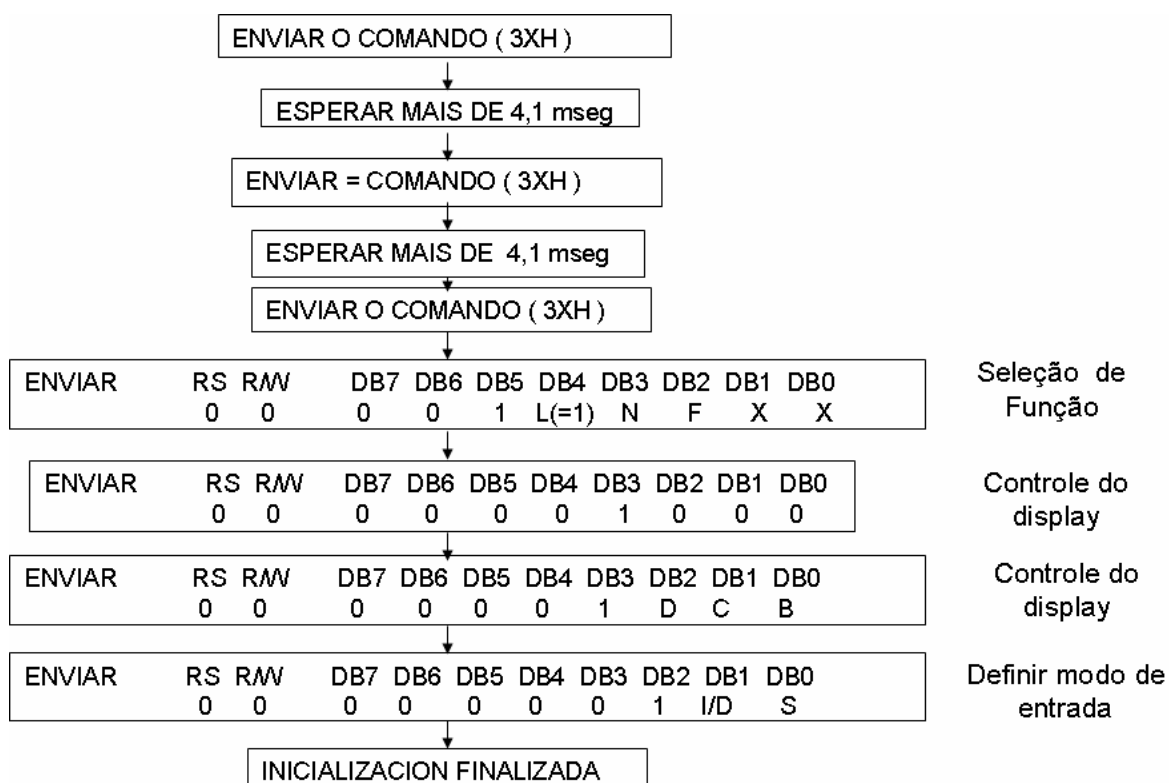
Toda vez que alimentamos o módulo LCD deve ser executado o procedimento de inicialização, que consiste no envio de uma seqüência de instruções para configurar o modo de mostrar no display durante uma aplicação determinada. Em muitos display este procedimento ocorre automaticamente, dentro de condições específicas que envolvem temporizações mínimas referente a transição do nível lógico 0 para 1, quando ligamos a fonte. Em caso de dúvidas, recomendamos o envio destas instruções após o reset do sistema.

Inicialização para sistemas 8 bits de dados (5 instruções)

Entre as duas primeiras instruções recomendamos um delay de 15 mS. As demais instruções podem ser escritas após checar o Busy Flag.

MÓDULO LCD	Instruções em Hexadecimal (8 bits)									
	1		2		3		4		5	
1 linha - Matriz 7x5 e 8x5	30	15mS	30	15mS	06	BF	0E	BF	01	
1 linha - Matriz 10x5	34	15mS	34	15mS	06	BF	0E	BF	01	
2 linha - Matriz 7x5 e 8x5	38	15mS	38	15mS	06	BF	0E	BF	01	

O fabricante HITACHI sugere seguir a seguinte seqüência de comandos.



Inicialização para sistemas 4 bits de dados (5 instruções)

Entre as quatro primeiras instruções recomendamos um delay de 15 ms. As demais instruções podem ser escritas após checar o Busy Flag. Estes bits (*nibble*) devem estar conectados aos pinos 11, 12, 13 e 14.

MÓDULO LCD	Instruções em Hexadecimal (4 bits)											
	1	2	3	4	5	6	7	8	9	10	11	12
1 linha - Matriz 7x5 e 8x5	3	3	3	2	2	0	0	8	0	1	0	1
1 linha - Matriz 10x5	3	3	3	2	2	4	0	8	0	1	0	1
2 linha - Matriz 7x5 e 8x5	3	3	3	2	2	8	0	8	0	1	0	1

3.4.3 Exemplos em *assembler* e em Linguagem C com o módulo LM016.

Exercício Resolvido

Utilizando o esquema eletrônico da figura XXX2, realizar o programa necessário para mostrar o cartel "BENVINDO A UEA" no modulo com LCD LM016.

Solução:

```
.*****
;
; Programa de teste do LCD *
.*****
;
; Comandos do LCD
DISP_APAGA EQU 01H; Comando de apagar display
DISP_INICIA EQU 02H ; Comando posição inicial
DISP_FORMATO EQU 38H ; Comando de seleção de função:
; DL = 1 (COMPRIMENTO DE DADOS DE 8 BITS)
; N = 1 (NUMERO DE LINHAS = 2)
; F = 0 (FONT (TIPO) DE CARACTER DE 5 * 7 PONTOS)
DISP_DESLOCA EQU 06H ; Comando de Modo do display.
; I/D = 1 (CURSOR INCREMENTA SUA POSIÇÃO)
; S = 0 (O DISPLAY NAO SE DESLOCA)
DISP_ATIVA EQU 0CH; Comando Display Aceso/Apagado
; D = 1 (DISPLAY ACESO)
; C = 0 (CURSOR APAGADO)
; B = 0 (CURSOR QUETO O NAO PISCANTE)
DESPLAZA_CURSOR EQU 14H
; S/C = 0 (DESLOCA O CURSOR)
; R/L = 1 (DESPLAZA À DIREITA)
DEFINE_CARACTER EQU 40H
VISUALIZA_CARACTER EQU 80H ; Comando de seleção do
;endereço da DRAM
;endereço 0.
```

```

; Sinais de controle para o LCD
RW    EQU    T0
RS     EQU    INT0
E      EQU    INT1
PORTA EQU    P1
RETARDO EQU    10H
CODIGO EQU    R3
CANT_CARACT EQU    R4
CARACTER EQU    R5
INCREMENTO EQU    R0

; Texto a visualizar
TABELA EQU    9000H
ORG    9000H
DB    0EH          ;Quantidade de caracteres a visualizar
DB    'BENVINDO A UEA' ; Cartel a visualizar
;*****
;
;  PROGRAMA PRINCIPAL.                *
;*****
;
ORG    8000H
    MOV    PORTA,#0FFH
    MOV    INCREMENTO,#00H    ;Carregar 00 a R0
    LCALL  Disp_inicializa    ; Inicializar controlador
    MOV    DPTR,#TABELA      ;Apontar a tabela dados
    MOV    A,INCREMENTO      ;Traer variable índice
    MOVC   A,@A+DPTR         ;Sacar longitud de cadena
    MOV    CANT_CARACT,A      ;Resguardar
Laço: INC    INCREMENTO      ;Incrementar indice
    MOV    A,INCREMENTO
    MOVC   A,@A+DPTR         ;Sacar caracter de tabela
    MOV    CHARACTER,A       ;Resguardar
    LCALL  Visualizar         ;Enviar caracter a controlador
    DJNZ   CANT_CARACT,Laço   ;Laço = longitud de cadena
Here: SJMP Here              ;Laço infinito inactivo
;*****
;
;  SUBROUTINA Disp_inicializa.        *
;*****
;
Disp_inicializa:
    MOV    CODIGO, #DISP_APAGA
    LCALL  Control
    LCALL  Control

```

```

    LCALL Control
    LCALL Control
    MOV CODIGO, #DISP_INICIA
    LCALL Control
    MOV CODIGO, #DISP_FORMATO
    LCALL Control
    LCALL Control
    LCALL Control
    LCALL Control
    MOV CODIGO, #DISP_DESLOCA
    LCALL Control
    MOV CODIGO, #DISP_ATIVA
    LCALL Control
    RET

.*****
;
; SUBROUTINA Control. *
.*****
;
Control:
    LCALL Demora
    MOV PORTA, CODIGO
    CLR RW
    CLR RS
    SETB E
    CLR E
    CLR RS
    SETB RW
    LCALL Demora
    RET

.*****
;
; SUBROUTINA Visualizar. *
.*****
;
Visualizar:
    LCALL Demora
    MOV PORTA,CARACTER
    CLR RW
    SETB RS
    SETB E
    CLR E
    CLR RS
    LCALL Demora
    RET

.*****
;

```

```

; SUBROTINA Demora.
;
;*****
;
Demora:
    MOV     R6, #20H
Laço_1: DJNZ  R6, Laço_1
    RET
END

```

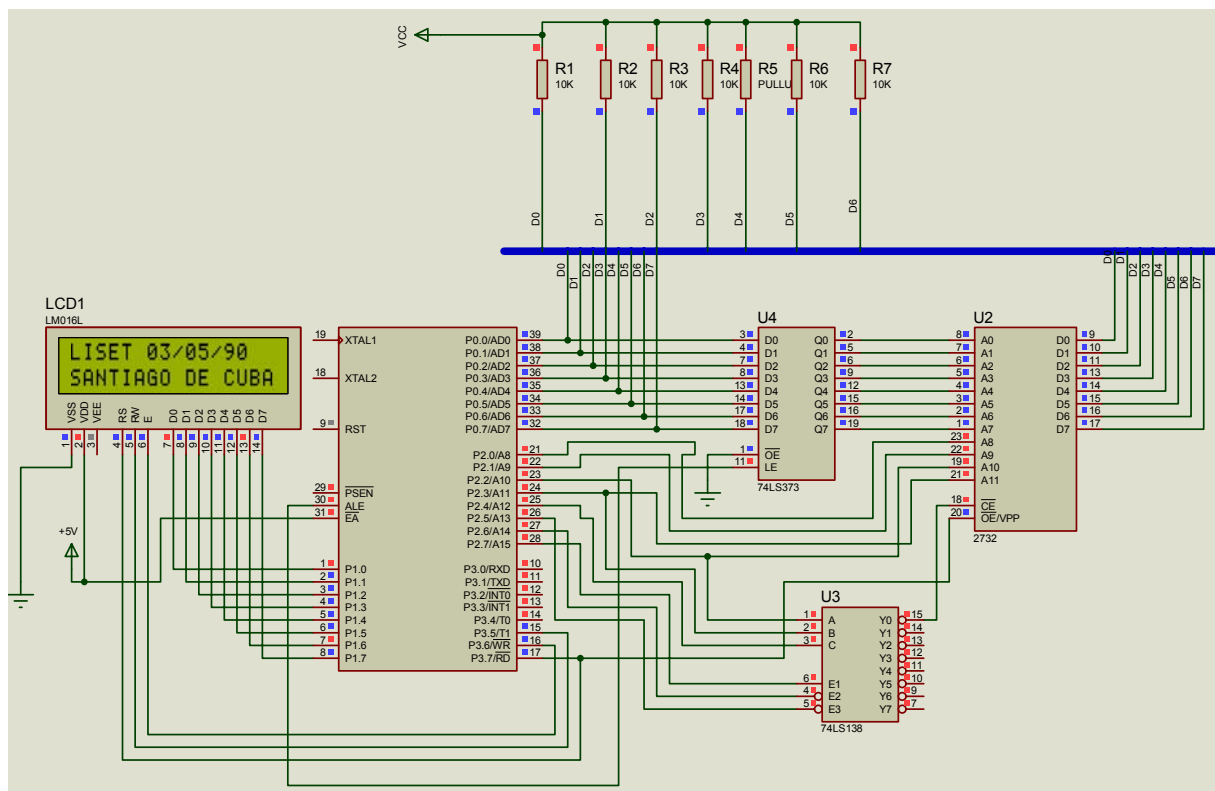
Exercício Resolvido

Usando o simulador PROTEUS, realizar o hardware usando um microprocessador acoplado a um modulo LM016, e o software necessário (na linguagem ASSEMBLER) para garantir o cartel mostrado embaixo que será armazenado numa memória externa INTEL2732 (4 kb) cujo endereço de início é 8000H.

LISET 03/05/90
SANTIAGO DE CUBA

Solução:

O hardware e a execução do programa se mostra na seguinte figura:



O software se armazena em 2 memórias diferentes: (a) o programa de visualização se armazena na memória de programa interna do microprocessador 8051 e (b) cartel se armazena na memória EPROM externa.

a) PROGRAMA PRINCIPAL

```
; Programa de teste do LCD *
;*****
; Comandos do LCD
DISP_APAGA EQU 01H; Comando de apagar display
DISP_INICIA EQU 02H; Comando posição inicial
DISP_FORMATO EQU 38H; Comando de seleção de função

; DL = 1 (COMPRIMENTO DE DADOS DE 8 BITS)
; N = 1 (NUMERO DE LINHAS = 2)
; F = 0 (FONT (FONDO) DE CARACTER DE 5 * 7 PONTOS)

DISP_DESLOCA EQU 06H; Comando de Modo do display.
; I/D = 1 (CURSOR INCREMENTA SUA POSICAO)
; S = 0 (O DISPLAY NAO SE DESLOCA)

DISP_ACTIVIA EQU 0CH; Comando Display Aceso./Apagado
; D = 1 (DISPLAY ACESO)
; C = 0 (CURSOR APAGADO)
; B = 0 (CURSOR QUIETO O NAO PISCANTE)

DESLOCA_CURSOR EQU 14H
; S/C = 0 (DESLOCA O CURSOR)
; R/L = 1 (DESPLAZA A LA DERECHA)

DEFINE_CHARACTER EQU 40H
VISUALIZA_CHARACTER EQU 80H; Comando de seleção do endereço da DDRAM
;endereço 0.
VISUALIZA_CHARACTER2 EQU 0C0H; 2da Linha
; Sinais de controle para o LCD
RW EQU P3.5
RS EQU P3.7
E EQU P3.6

PORTA EQU P1
RETARDO EQU 0FFH
```

```

; Texto a visualizar
TABELA EQU 8000H

.*****
;
; PROGRAMA PRINCIPAL. *
;
.*****
;
ORG 0H

MOV PORTA,#0FFH
MOV R0,#00H ;Llevar 00 a R0
LCALL Disp_inicializa ;Inicializar controlador
MOV DPTR,#TABELA ;Apontar a tabela dados
MOVBX A,@DPTR ;Trazer variável índice do nome
MOV R4,A ;GUARDAR
Lazo: INC DPL ;Incrementar índice, Apontar a nome
MOVBX A,@DPTR ;Trazer caracter da memória
MOV R5,A ;Resguardar
LCALL Visualizar ;Enviar caracter a controlador
DJNZ R4,Lazo ;Lazo = comprimento da cadeia

; Sacar dados de nascimento
MOV DPTR,#TABELA+10 ;Apontar a tabela dados
MOVBX A,@DPTR ;Trazer variável e índice dos dados
MOV R4,A ;GUARDAR
Lazo1: INC DPL ;Incrementar índice
MOVBX A,@DPTR ;Trazer caracter da memória
MOV R5,A ;Guardar
LCALL Visualizar ;Enviar caracter a controlador
DJNZ R4,Lazo1 ;Lazo = comprimento da cadeia

; Enviar ao display um comando de troca de linha

MOV R3, #VISUALIZA_CHARACTER2
LCALL Controle
LCALL Controle
LCALL Controle
LCALL Controle

; Sacar cidade de nascimento
MOV DPTR,#TABELA+20 ;Apontar a tabela dados
MOVBX A,@DPTR ;Trazer variável cidade
MOV R4,A ;GUARDAR

```



```

Lazo2: INC   DPL           ;Incrementar índice
        MOVX  A,@DPTR      ;Trazer caracter da memória
        MOV   R5,A         ;Guardar
        LCALL Visualizar   ;Enviar caracter a controlador
        DJNZ  R4,Lazo2     ;Lazo = comprimento da cadeia
Here:   SJMP  Here         ;Lazo infinito inativo

```

```

.*****
;
;  SUBROTINA Disp_inicializa.          *
.*****
;

```

Disp_inicializa:

```

        MOV   R3,#DISP_APAGA
        LCALL Controle
        LCALL Controle
        LCALL Controle
        LCALL Controle
        MOV   R3,#DISP_INICIA
        LCALL Controle
        MOV   R3,#DISP_FORMATO
        LCALL Controle
        LCALL Controle
        LCALL Controle
        LCALL Controle
        MOV   R3,#DISP_DESLOCA
        LCALL Controle
        MOV   R3,#DISP_ACTIVIA
        LCALL Controle
        RET

```

```

.*****
;
;  SUBROTINA Controle.                *
.*****
;

```

Controle:

```

        LCALL Demora
        MOV   PORTA, R3
        CLR   RW
        CLR   RS
        SETB  E
        CLR   E
        CLR   RS
        SETB  RW
        LCALL Demora
        RET

```

```

.*****
;
; SUBROTINA Visualizar.          *
.*****
;
Visualizar:
    LCALL Demora
    MOV  PORTA,R5
    CLR  RW
    SETB RS
    SETB E
    CLR  E
    CLR  RS
    LCALL Demora
    RET

.*****
;
; SUBROTINA Demora.             *
.*****
;
Demora:
    MOV  R7, #10
    MOV  R6, #0FFH
Lazo_1: DJNZ R6, Lazo_1
    DJNZ R7, Lazo_1
    RET
END

```

b) CARTEL NA MEMÓRIA EXTERNA

```

    ORG 0                ;Endereço inicial local da memória 2732

DB  06H                 ;Quantidade de caracteres a visualizar

DB  'LISET '            ;Nome do Cartel a visualizar

    ORG 10

DB  08

DB  '03/05/90'          ;Data do nascimento do Cartel a visualizar

    ORG 20

DB  16

DB  'SANTIAGO DE CUBA'   ;Cidade do nascimento do Cartel a visualizar

    END

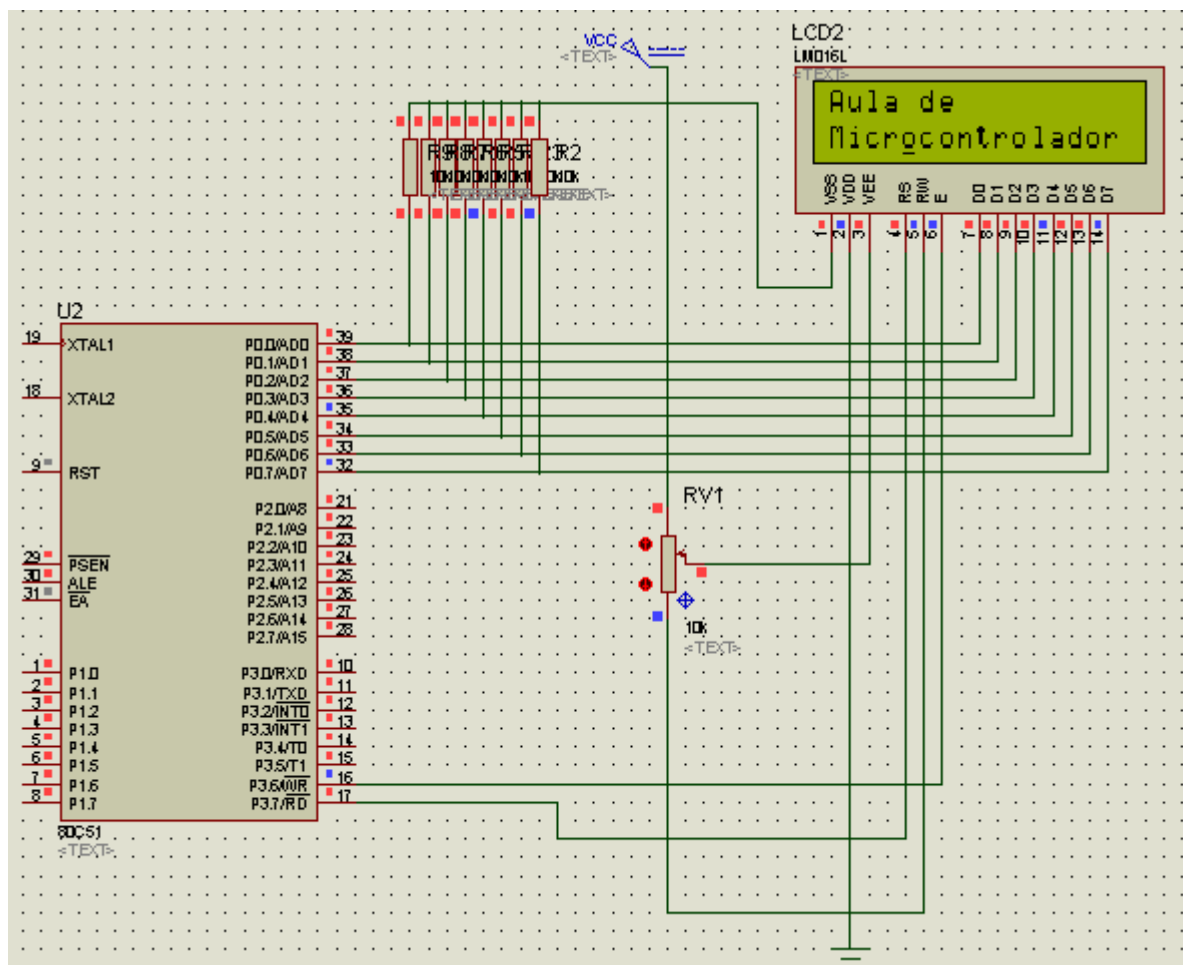
```

Exercício Resolvido

Usando o simulador PROTEUS, realizar o hardware com o modulo LM016, e o software necessário (em linguagem C) para mostrar (em duas linhas) o seguinte cartel:

Aula de
Microcontrolador

Solução: O hardware em pleno funcionamento se mostra na figura seguinte, onde se tem omitido, por simplicidade para a simulação, a conexão de alguns pinos, mas que na vida real são imprescindíveis.



O software em linguagem C se mostra a seguir:

```
#include <at89s8252.h>

void cmlcd(unsigned char c, char cd)    // Rotina para enviar comandos ao LCD
{ int t;
  P0 = c;
  if (cd==0) P3_7=0; else P3_7=1; //cd = 0 controle, cd = 1 dados
```

```

P3_6=1; //gerar pulso de Habilitação de 1 a 0
P3_6=0;
for (t=0; t < 8;t++);
}
void initlcd(void) // função para inicializar o display LCD
{
    cmlcd (0x38,0); //Seleção de Função: 2 linhas de 5 * 7
    cmlcd (0x06,0); //Modo de operação: Escreve deslocando o cursos para direita
    cmlcd (0x0e,0); //Controle do display: Display aceso com cursor fixo
    cmlcd (0x01,0); //Modo de operação: Limpa e retorna cursos para o principio
}
void write(char *c) // função para escrever no LCD
{
    for (; *c!=0; c++) cmlcd(*c,1);
}
#include <at89s8252.h>

void main() // o programa começa aqui
{ initlcd(); // chama a função para inicializar o lcd

    for(;;){
        cmlcd (0x080,0); // posiciona o LCD para imprimir na primeira linha
        write ("Aula de"); // imprime mensagem na linha 2
        cmlcd (0x0C0,0); // posiciona o lcd para imprimir na segunda linha
        write ("Microcontroladores"); // imprime mensagem na linha 2
    }
    for(;;); // o programa fica aqui ate que aconteça algum reset ou interrupção
}

```

3.5 Temporizadores y contadores

Os temporizadores são um elemento importante em qualquer microcontrolador. Um temporizador é basicamente um contador o qual se excita com um sinal de relógio interna (modo Timer) ou externa (modo Contador) ao microcontrolador. O temporizador pode ser de 8 bits ou de 16 bits. Por programa,

pode-se carregar o valor de contagem do temporizador, assim como arrancar e deter ele. A maioria dos temporizadores pode configurar-se para gerar uma interrupção quando se alcançar certo número de contagem (usualmente quando se transbordam). O programa de atenção à interrupção pode empregar-se para levar a cabo operações de temporização exatas dentro do microcontrolador.

Os microcontroladores revistam ter ao menos um temporizador. Alguns microcontroladores podem ter dois, três, ou mais temporizadores e alguns destes podem conectar-se em cascata para realizar contagens maiores.

Aplicações dos Temporizadores

- Medir intervalos de tempo
- Enviar pulsos com duração definida
- Gerar sinais digitais como por exemplo: Sinal modulado por largura de pulso ou PWM (figura 3.25)
- Contar eventos.

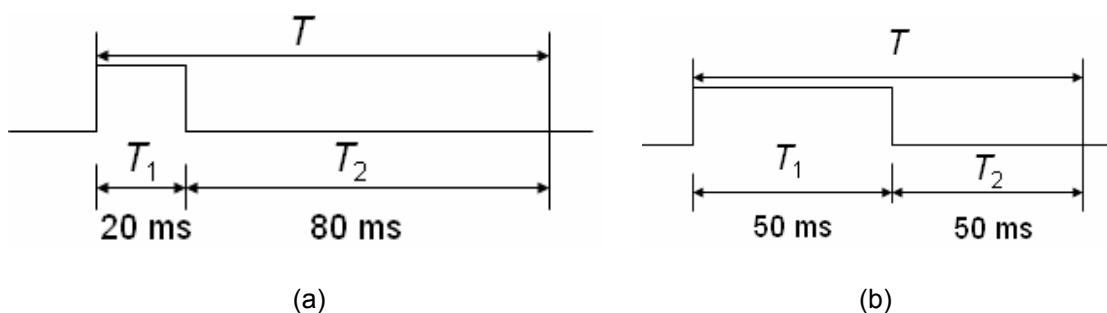


Figura 3.25 Modulação por largura de pulso (**PWM**): T_1 , T_2 trocam, mas T = constante. Existe outra modulação chamada por Freqüência de Pulso (**PFM**) onde T_1 (ou T_2) são constantes mas T_2 (ou T_1) trocam, e por tanto o período T não é constante.

Para detalhar um pouco mais no funcionamento de um temporizador, consideremos o circuito da Figura 3.26, que representa a um timer elementar de apenas 3 bits, com três saídas: Q0 (bit menos significativo), Q1 e Q2 (bit mais significativo). A estrutura desse temporizador é basicamente a de um contador assíncrono ascendente, com a novidade que neste caso, aparece um biestável tipo D ao final da cadeia de contagem, que indica quando a contagem ultrapassou o máximo números de pulsos possíveis a contar, procedentes do sinal de entrada indicado como sinal de Clock. Quando isso acontece o biestável (chamado no contexto de microprocessadores como Bit de *Flag* ou bandeira) vai a nível ALTO (se seta) para indicar o transbordamento do contador quando a contagem ultrapassa de

111 a 000. Nos microcontroladores tem integrado estruturas como estas, mas com um maior número de bits (8, 16, 24 bits), com possibilidades de usar contadores programáveis (e não um contador de módulo fixo como na figura) e com a possibilidade de ler o flag por programa, ou de que o flag gere uma interrupção à CPU quando transbordar.

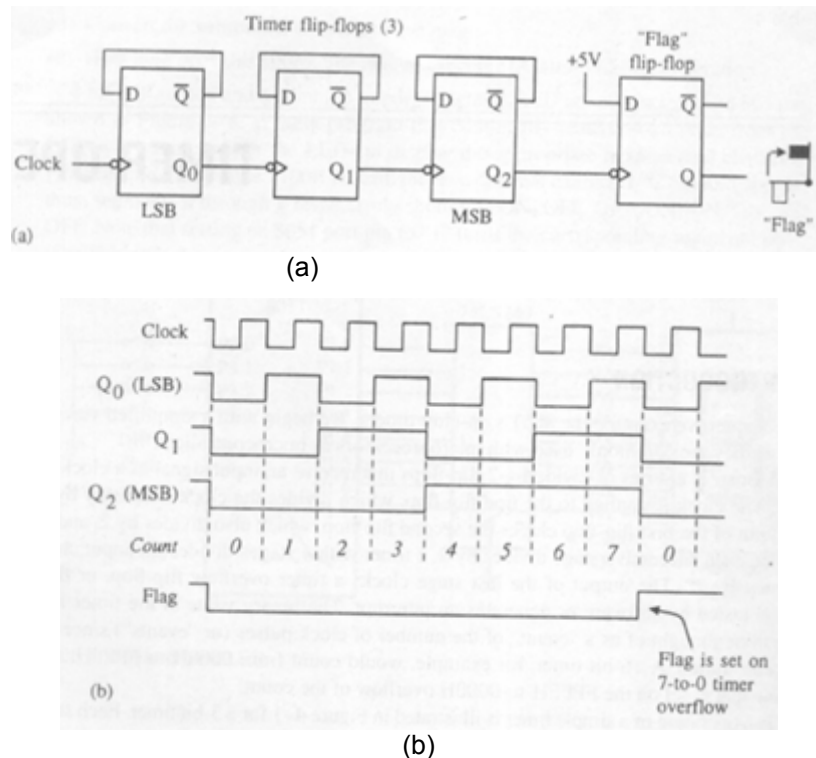


Figura 3.26 Estrutura e funcionamento de um temporizador elementar de 3 bits.

Um timer de 16 bits conta de 0000H a FFFFH. A *flag de overflow* é ativada cada vez que o contador ultrapassa de FFFFH a 0000H.

3.6 Temporizadores no microprocessador 8051

O 8051 tem dois registros contadores - temporizadores, Timer 0 e Timer 1, que podem configurar-se para que operem como temporizadores ou como contadores:

- Como temporizadores (timers), os registros contam ciclos de máquina (um ciclo de máquina é igual a 12 pulsos de relógio).
- Como **contadores** (counters), os registros contam as transições de 1 a 0 (flanco de descida) que se produzem nos pinos T0 e T1.

Em ambos os casos, a contagem é sempre ascendente.

Os temporizadores e contadores utilizam os registradores de funções especiais (SFR) seguintes:

SFR	Uso
TH0, TL0	Valor da contagem do timer/counter 0.
TH1, TL1	Valor da contagem do timer/counter 1.
TMOD	Para programar o modo de operação como timer ou counter.
TCON	Alguns bits servem para habilitar a contagem; outros estão relacionados com as interrupções.

Cada timer/counter se pode programar, como *timer* ou como *counter*, em 4 modos diferentes:

Modo	Operação
0	Temporizador / Contador de 13 bits
1	Temporizador / Contador de 16 bits
2	Temporizador/Contador de 8 bits con autocarregamento
3	Vários contadores

Os modos 0 e 1 são iguais, exceto pelo número de bits.

Os modos 0, 1 e 2 são igualmente aplicáveis a ambos os timers. O modo 3 no timer 0 é diferente ao modo 3 no timer 1 e introduz algumas limitantes ao timer 1 ainda quando este se encontre programado em outro modo.

TMOD							
Endereço: 89h							
7	6	5	4	3	2	1	0
GATE	C/T#	M1	M0	GATE	C/T#	M1	M0
Timer 1				Timer 0			

M1, M0	Modo: 00, 01, 10, 11
C/T#	0: Temporizador 1: Contador
GATE	1: Controle por hardware 0: Controle por software

TCON (Timer/Counter Control Register)							
Endereço: 88h; bits 88 a 08Fh							
7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Timer 1		Timer0		INT1#		IINT0#	

TR	0: Timer inabilitado 1: Timer habilitado
TF	Bandeira de <i>overflow</i> do timer. Vai a 1 quando o timer passa do número máximo (tudo na 0). Repõe-se (vai a 0) automaticamente ao atendê-la interrupção.

OBS:

Se $f_{OSC} = 12 \text{ MHz}$, $T_{C. MAQ.} = 1 \mu\text{seg}$ e o tempo máximo dos contadores é:

Se 8 bits: 0,256 mseg ($f = 3906,25 \text{ Hz}$)

Se 13 bits: 8,192 mseg ($f = 122,07031 \text{ Hz}$)

Se 16 bits: 65,536 mseg ($f = 15,258789 \text{ Hz}$)

Modos 0 e 1

Os modos 0 e 1 são aplicáveis indistintamente a ambos os timers, os quais ficam configurados como ilustra na figura 3.27.

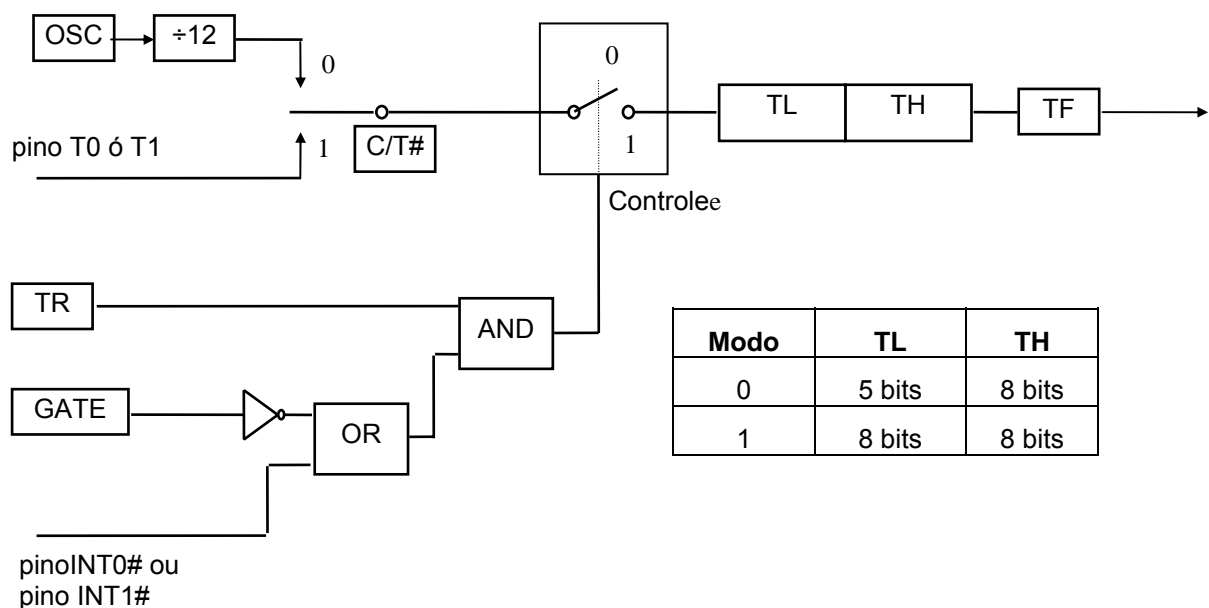


Figura 3.27 Estrutura interna de um temporizador no 8051 em modos 0 e 1.

Observe que:

- No modo 0 os contadores/temporizadores são de 13 bits. No modo 1 são de 16 bits.
- A contagem é ascendente. Quando o contador dá uma volta (vai a 0), se ativa a bandeira TF do timer correspondente.
- O bit C/T# do SFR TMOD seleciona a operação como temporizador ou como contador. Neste último caso se contam os pulsos que chegam por T0 ou T1, conforme seja o timer 0 ou 1.

O bit GATE do SFR TMOD atua como seletor do tipo de controle de habilitação do timer. Se GATE=1, o controle é governado pelo bit TR (controle por software); se GATE=0 e TR=1, o controle é governado pelo pino INT0# ou INT1# (controle por hardware).

Modo 2

O modo 2 é aplicável indistintamente a ambos os timers/counters, os quais ficam configurados como ilustra na figura 3.28:

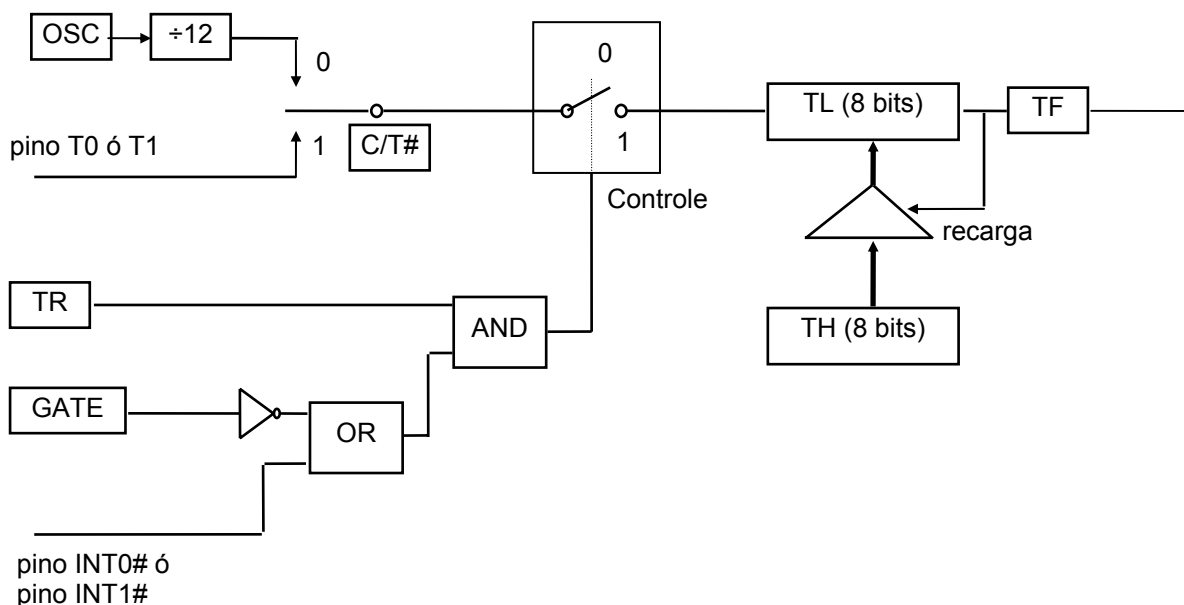


Figura 3.28 Estrutura interna de um temporizador no 8051 em modo 2.

Observe que:

- Este modo é muito parecido aos modos 0 e 1, só que agora cada timer utiliza um contador ascendente de 8 bits (TL) o qual, quando dá uma volta (vai a 0) faz que se recarregue no TL o valor existente no TH.
- O resto do esquema funciona igual a nos modos 0 e 1.

Modo 3

O modo 3 no timer 0 é diferente ao modo 3 no timer 1.

Quando o timer 1 se programa em modo 3, simplesmente se detém o contagem; o efeito é similar a fazer TR1=0.

O timer 0 programado em modo 3 se desdobra em dois contadores de 8 bits cada um: o primeiro pode ser timer/counter e o segundo trabalha como timer. A figura 3.29 ilustra a estrutura que toma o timer 0 em modo 3:

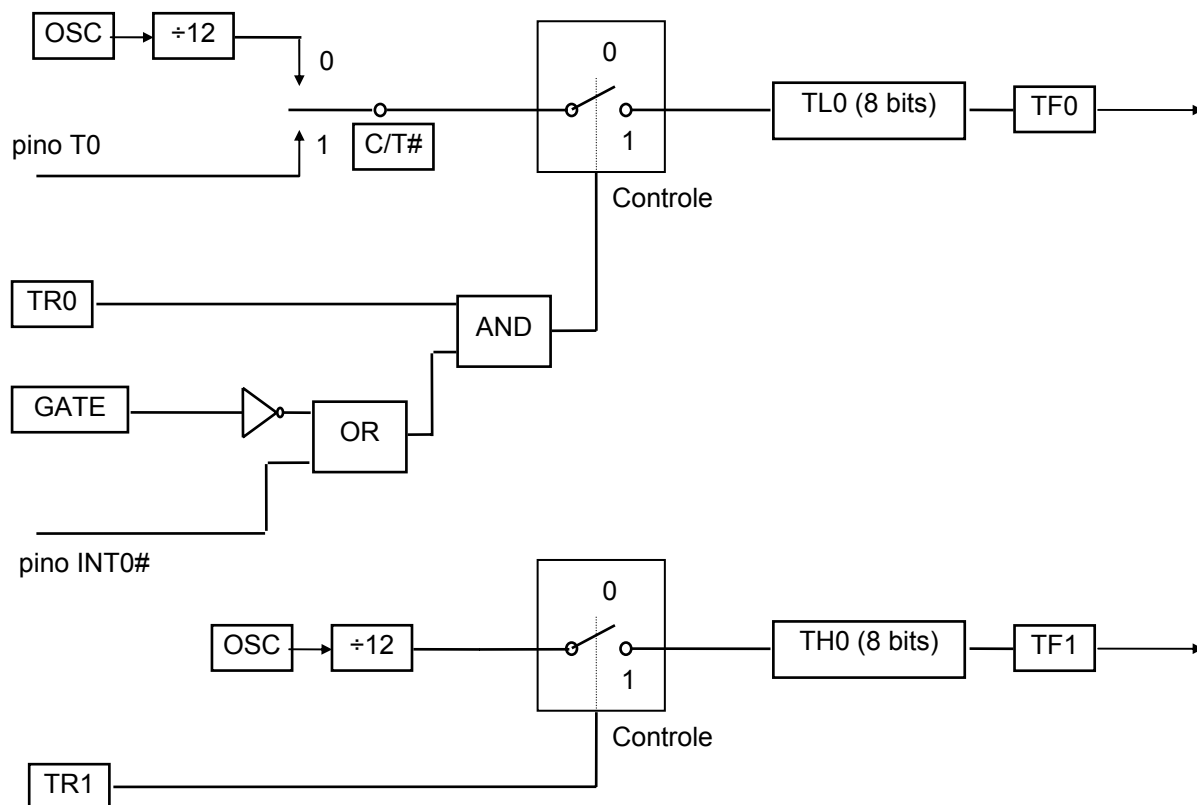


Figura 3.29 Estrutura interna de um temporizador no 8051 em modo 3.

Observe que neste modo o timer 0 utiliza alguns bits (TR1, TF1) que pertencem ao timer 1, então o timer 1 pode utilizar-se em aplicações que não necessitem interrupções provenientes deste timer.

3.6.1 Criação de uma base de tempo e sincronização de eventos à mesma.

Pode-se criar uma base de tempo (relógio de tempo real) no microcontrolador utilizando a interrupção de um dos timers. Pode-se, por exemplo, programar o timer 0 em modo 0 ou 1 e assim obter uma interrupção periódica cada certo número de milissegundos. Dentro da rotina que atende esta interrupção, se seta uma bandeira que pode ser consultada pelo programa. Se a bandeira estiver em 1 significa que transcorreu o tempo previsto e então se pode executar o evento que se deseja sincronizar. Uma vez realizado o evento se zera a bandeira e continua a consulta.

O seguinte exemplo ilustra esta maneira de proceder.

Exemplo:

Tem-se um 8051 com um cristal de 12 MHz. Deseja-se formar uma base de tempo de 1 s e sincronizada a ela, alternar o valor da porta P1 entre 00 e FFh.

Pode calcular-se que se o cristal for de 12 MHz e o timer 0 se programa em modo 0, a interrupção ocorre a cada 8,19 ms (122 Hz) aproximadamente. Para alcançar um segundo terá que contar 122 interrupções (*tics*) do contador. Para isso se pode utilizar o registrador R7. Como bandeira indicadora do segundo se utilizou o bit PSW.5 (F0), a qual é setada cada um segundo no programa da interrupção do timer e, consultada e posta a zero no programa principal.

```
;
; Exemplo: P1 pisca cada 1 s, sincronizado a uma base de tempo.
; File: RELOGIO.ASM
;
    ORG    0
    JMP    INÍCIO
;
    ORG    0Bh                ; Endereço da interrupção TIMER0
    JMP    TIMER_0
;
; Inicialização do sistema:
;
INÍCIO:
    CLR    EA                ; Inabilitar todas as interrupções.
    MOV     TMOD, #00        ; Programar timer 0 como temporizador,
                                ; modo 0, controle por software.

    MOV     TH0, #0
    MOV     TL0, #0          ; Pôr valor do contador em 0.
    MOV     R7, #122        ; Contador de tics de relógio (127 tics= 1 s).
    CLR     F0              ; PSW.5 (F0 indicador do segundo).
;
    SETB    TR0             ; Habilitar contagem do timer 0
    MOV     IE, #82H        ; Habilitar só a interrupção do timer 0
;
; Programa principal:
;
PROG:
    JNB     F0, SIGUE        ; Consulta-se F0: se F0=0 IR SEGUE,
    MOV     A, P1            ; se F0=1, realizar evento sincronizado.
```

```

CPL    A
MOV    P1, A
CLR    F0                      ;Resetar F0
SEGUE:
    JMP    PROG
    ;
    ; Interrupção do timer 0:
    ;
TIMER_0:
    DJNZ   R7, FIN_0           ; Se R7 não for 0, retornar,
    MOV    R7, #122           ;si R7=0, recarregar R7.
    SETB   F0                  ; Setar bandeira do segundo.
FIM_0:
    RETI                       ;Retornar.
    ;
END

```

Exercício Resolvido

```

; Progr14 :  Introduz o recurso de "Programar os TIMERS"
;
;   Para entender bem os "timers" e' necessário ler o livro no capítulo de
;   timers.Aqui voce so' terá um resumo da aplicação.Com duas palavras de
;   controle se comanda os timers.Elas são TMOD e TCON. Em TMOD eu programo
;   como o timer ira' funcionar :modo 0,1,2,ou 3;ainda defino nesta palavra
;   TMOD se o timer ira' contar pulsos externos ou contara' da base de tempo
;   interna.Defino finalmente nesta,se o timer ficara' sujeito a disparo
;   por interrupção.São quatro bits para cada timer de programação com fim
;   a se utilizar dois bits para determinar o modo de operação(M1,M0);um
;   bit de definição de interrupção (GATE) e um bit para se definir se o
;   timer ira' se basear em base de tempo interna ou contagem de eventos ex-
;   ternos(C/T).Como temos dois timers disponíveis ,temos quatro bits para
;   cada timer totalizando os oito bits disponíveis na palavra. Em TCON eu
;   posso ligar o timer,como também verificar se a contagem já chegou ao ma-
;   ximo.São dois bits para cada timer para estas duas funções(TR=liga,
;   TF=chegou ao fim).Os outros quatro bits desta palavra tem outra função
;   que será visto no programa de interrupção:
;
;   TMOD
;   .....
;
;   GATE1  C/T1  M1.1  M0.1   GATE0  C/T0  M1.0  M0.0

```

```

; -----
;
;          TIMER 1          TIMER 0
;
; TCON
;
; .....
;
; TF1   TR1   TF0   TR0   XX   XX   XX   XX
;
; -----
;
;   TIMER1   TIMER0   NAO TEM APLICACAO NO TIMER
;
; Os modos de contagem sao :
;
;
;
;       M1  M0
;
; MODO 0   0  0 ,conta ate' 13 bits
; MODO 1   0  1 ,conta ate' 16 bits
; MODO 2   1  0 ,conta ate' 8 bits com recarga automática do numero
; MODO 3   1  1 ,conta" evento externo" por "tempo interno",utilizando
;
;           neste caso parte do timer 1 para esta contagem "even-
;
;           tos por tempo", alem de usar o timer 0 todo.
;
; Para se carregar os valores de contagem se utiliza dos registradores cha-
;
; mados por TH,"Timer High" e TL,"Timer Low", um par deles para cada timer,
;
; isto e'. TH1/TL1 e TH0/TL0.E' neles que eu carrego o valor inicial de
;
; contagem desejada ,onde o contador ira' incrementando seu valor ate' che-
;
; gar no Maximo de contagem,onde ai' o timer nos avisara' fazendo o bit
;
; TF=1 que fica na palavra TMOD.
;
; Em resumo: cada timer tem sua programação em TMOD,seu controle em TCON
;
; e sua carga inicial de contagem em THigh e TLow, que sao dois registra-
;
; dores para cada timer ter seu valor inicial de contagem.
;
; Na contagem da base de tempo interna, o timer pega a frequencia do clock
;
; do Micro e divide por 12 antes de contar.(Ex clock 12MHz => 1microseg.)
;
; Cada vez que a contagem chega ao Maximo eu tenho que recarregar os valo-
;
; res de TH eTL de novo.So' no modo 2 e' que eu não preciso fazer isto,
;
; pois a recarga e' automática.
;
; Ex. Programar timer 0 como contador de 16 bits para contagem de um total
;
; de 20000 pulsos do clock interno do Micro .O cristal deste Micro roda a
;
; 12MHz.Isto e', eu vou contar um total de 20 Milisegundos.(Certo?).Neste
;
; exemplo eu nao uso interrupção para avisar o Micro que ja' chegou no
;
; fim de contagem,i.e',eu faco o Micro ficar "esperando" o contador termi-
;
; nar a contagem,que não e' a melhor opção.No segundo ex. vai se interrom-
;
; per o Micro pelo Timer,que e' o ideal,pois o Micro ficara' livre para
;
; fazer outras tarefas enquanto o Timer conta.
;
; Ex1: Timer não interrompe o Micro.OBS: Como este programa se inicia em
;
;   000Fh,você deve forçar o "PC" ir para este ponto de memória,senão
;
;   não se conseguira' simular o exercício.
;label instr  operando;comentários

```

```

.***** ***** ***** *****
;
    org    000Fh        ;começo este software depois da interrupção
                        ;que se vai gerar no segundo ex.(EX2) para nao
                        ;ter conflito de compilação entre este programa
                        ;e o que vem a seguir, que usa a interr. do
                        ;Timer 0 que esta' em 000Bh.

    mov     ie,#00000000b ; Sem interrupção alguma.
    mov     TMOD,#10h     ;programo TMOD,TIMER1,para modo 1
                        ;Isto e',TMOD=0000 0001,ou M1=0,M0=1
                        ;e ainda contagem interna,pois C/T=0
    mov     TH1,#high(65535-20000) ;Ao inves de fazer as contas na
                        ;mao,deixo o compilador fazer
                        ;para mim.
    mov     TL1,#low(65535-20000) ;Nestas duas instruções eu carreguei
                        ;a parte alta do numero 45535 em THigh
                        ;e a parte baixa em TLow,criando um
                        ;contador de 16 bits que contara' de
                        ;45535 ate' 65535,dando o total de
                        ;20000 contagens
    setb     TR1          ;Ligo o timer 1.E' um bit da palavra TCON
                        ;que no caso acionei via "Bit" ao inves de
                        ;via "Byte".(Veja a "palavra" acima no texto).

    JNB      TF1,$        ;Aqui o Micro fica "esperando" a contagem fi-
                        ;nalizar,através do teste do Bit TF1, que e'
                        ;o indicador de "Contagem Estourada".
    CLR      TR1          ;O contador ja' contou,então desligo-o através
                        ;do bit TR1.
    CLR      TF1          ;Tenho que tomar cuidado par "desligar " o Flag
                        ;de aviso de estouro de contagem,senao na pro-
                        ;xima vez de usa-lo ele ja' estara' setado e fa-
                        ;ra'confusão,pois voce ira' testa-lo se ele foi
                        ;para "1" e ele ja'esta' em "1".

;    Ex2: Timer interrompe o Micro .Comece analisando o software pelo
;    "org 100h" para poder estuda-lo.(Faca PC=100h)

;label instr operando;comentarios
.***** ***** ***** *****
;
;_____
    ORG     0BH          ;TIMER0 (ET0) -Serviço de interrupção do TIMER0
    CLR     TR0          ;O contador ja' contou,então desligo-o através

```

```

;do bit TR0.Observe que eu não fiz aqui a rotina
;"Clear TF0",pois a interrupção automaticamente faz
;isto para mim.(So' no caso de Interrupção).
reti      ;Retorno de Interrupção
;
org 100h      ;comecei o programa em 100h
inicio: mov  ie,#10000010B ;TIMER0 (ET0) pode pedir interrupção.
mov  ip,#2      ;TIMER0 max prioridade
mov  TMOD,#01h  ;programoTMOD,TIMER0,para modo 1
               ;Isto e',TMOD=0000 0001,ou M1=0,M0=1
               ;e ainda contagem interna,pois C/T=0
mov  TH0,#high(65535-20000) ;Ao inves de fazer as contas na
               ;mao,deixo o compilador fazer
               ;para mim.
mov  TL0,#low(65535-20000) ;Nestas duas instruções eu carreguei
               ;a parte alta do numero 45535 em THigh
               ;e a parte baixa em TLow,criando um
               ;contador de 16 bits que contara' de
               ;45535 ate' 65535,dando o total de
               ;20000 contagens
setb  TR0      ;Ligo o timer 0.E' um bit da palavra TCON
               ;que no caso acionei via "Bit" ao inves de
               ;via "Byte".(Veja a "palavra" acima no texto).
Mov  A,#00      ; Zera A
loop: INC  A      ;Incrementa A enquanto o Timer fica contando.
sjmp  loop      ;Veja que eu nao estou "esperando" o Timer
               ; contar eu estou fazendo outra coisa (Incre-
               ;mentando "A" neste exemplo) ate' que o Timer
               ;estoure a conta. Ai' eu atendo a sua inter-
               ;rupcao, e depois, eu retorno da interrupção
               ;(RETI),voltando ao meu antigo serviço que era
               ;incrementar" A".

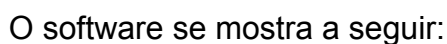
END

```

Exercício Resolvido

Realizar hardware e software (em linguagem C) para contar o número de pulsações de uma tecla conectada à entrada externa do Timer 0 (#T0). Mostrar a contagem na primeira linha de um modulo LCD LM016 depois de um cartel CONTAGEM, e na segunda linha mostrar o seguinte cartel: Modo Contador.

O hardware em funcionamento se mostra na seguinte figura:



```
void cmlcd(unsigned char c, char cd) // Rotina para enviar comandos ao LCD
```

}

 $\{$

}


```

void write(char *c)                                // funcao para escrever no LCD
{
    for (; *c!=0; c++) cmlcd(*c,1);    //c es un string y passa o endereço de primer
    elemento
}
void mostreLcd(unsigned int conta)
{
    unsigned char aux;
    cmlcd(0x8A,0);    //Começar a escrever depois do cartel CONTAGEM
    aux = conta/10000;
    aux = aux + 0x30;
    cmlcd(aux,1);
    conta = conta % 10000;

    aux = conta / 1000;
    aux = aux + 0x30;
    cmlcd(aux,1);
    conta = conta % 1000;

    aux = conta / 100;
    aux = aux + 0x30;
    cmlcd(aux,1);
    conta = conta % 100;

    aux = conta / 10;
    aux = aux + 0x30;
    cmlcd(aux,1);
    conta = conta % 10;

    aux = conta / 1;
    aux = aux + 0x30;
    cmlcd(aux,1);
    conta = conta % 1;
}
void timer_2(void)

```

```

{
unsigned int cont;
    TMOD = 0x05;    //01 para timer, 05 como contador
    TH0 = 0x00;//~ (60000/256); // TH0 = -36; // (60000/256);//
    TL0 = 0x00;//~ (60000%256);
    TR0 = 1;
    TF0 = 0;
    while(!TF0)
    {
        cont = (TH0 << 8)| TL0;
        mostreLcd(cont);
    }
}

void main(void)                // o programa começa aqui
{
    initLcd();                  // chama a funcao para inicializar o lcd
    for(;;){
        cmlcd (0x080,0);        // posiciona o lcd para imprimir na segunda linha
        write (" CONTAGEM ");   // imprime mensagem na linha 2
        cmlcd (0x0C0,0);        // posiciona o lcd para imprimir na segunda linha
        write ("Modo Contador"); // imprime mensagem na linha 2
        timer_2();
    }

    // o programa fica aqui ate que aconteca algum reset ou
    interrupcao
}

```

Exercício Resolvido

Realizar hardware e software (em linguagem C) para contar o número de ciclos de máquina de um 8051 ($f_{XTAL} = 12 \text{ MHz}$) e mostrar a contagem na primeira linha de um modulo LCD LM016 depois de um cartel CONTAGEM, e na segunda linha mostrar o seguinte cartel: Modo Timer. Inicialmente realizar um controle da contagem por software, e posteriormente o controle por hardware (mediante uma tecla conectada ao pino #INT0).

Solução:

O hardware em funcionamento se mostra na seguinte figura.


```

void initlcd(void)                // funcao para inicializar o display LCD
{
    cmlcd (0x38,0);  //2 linha de 5+7
    cmlcd (0x06,0);  //Escreve deslocando o cursos
    cmlcd (0x0e,0);  //Display aceso com cursor fixo
    cmlcd (0x01,0);  //Limpa display e retorna cursos para inicio.
}

void write(char *c)                // funcao para escrever no LCD
{
    for (; *c!=0; c++) cmlcd(*c,1);    //c es un string y passa o endereço de primer
    elemento
}

void mostreLcd(unsigned int conta)
{
    unsigned char aux;
    cmlcd(0x8A,0);    //Começar a escrever depois do cartel CONTAGEM
    aux = conta/10000;
    aux = aux + 0x30;
    cmlcd(aux,1);
    conta = conta % 10000;

    aux = conta / 1000;
    aux = aux + 0x30;
    cmlcd(aux,1);
    conta = conta % 1000;

    aux = conta / 100;
    aux = aux + 0x30;
    cmlcd(aux,1);
    conta = conta % 100;

    aux = conta / 10;
    aux = aux + 0x30;
    cmlcd(aux,1);
    conta = conta % 10;
}

```

```

    aux = conta / 1;
    aux = aux + 0x30;
    cmlcd(aux,1);
    conta = conta % 1;
}
void timer_2(void)
{
    unsigned int cont;
    TMOD = 0x09;    //01: timer controlado por software, 09 por hardware
    TH0 = 0x00;//~ (60000/256); // TH0 = -36; // (60000/256);//
    TL0 = 0x00;//~ (60000%256);
    TR0 = 1;
    TF0 = 0;
    while(!TF0)
    {
        cont = (TH0 << 8) | TL0;
        mostreLcd(cont);
    }
}
void main(void)                // o programa começa aqui
{
    initlcd();                  // chama a função para inicializar o lcd
    for(;;){
        cmlcd (0x080,0);        // posiciona o lcd para imprimir na segunda linha
        write (" CONTAGEM ");   // imprime mensagem na linha 2
        cmlcd (0x0C0,0);        // posiciona o lcd para imprimir na segunda linha
        write ("Modo Timer");    // imprime mensagem na linha 2
        timer_2();
    }

    // o programa fica aqui ate que aconteça algum reset ou
    // interrupção
}

```

3.7 Interrupções

3.7.1 Mecanismos de Entrada Saída dos microprocessadores

Os mecanismos de Entrada Saída com os periféricos são os seguintes:

- a) Espera ou Pesquisa (*polling*): O programa principal do microprocessador (ou seja, o programa que começa depois de um sinal de RESET) testa sistematicamente os terminais das portas conectados ao periférico para conhecer o estado do periférico. Pode ser estabelecido algum tipo de protocolo de conversação (*handshake*) entre o periférico e a CPU, tal como se mostra na figura 3.30.

As desvantagens deste método são:

- Terá que interrogar os bits de consulta periodicamente.
- Ao periférico lhe atende depois de realizar a pesquisa e não quando solicita a intervenção da CPU.

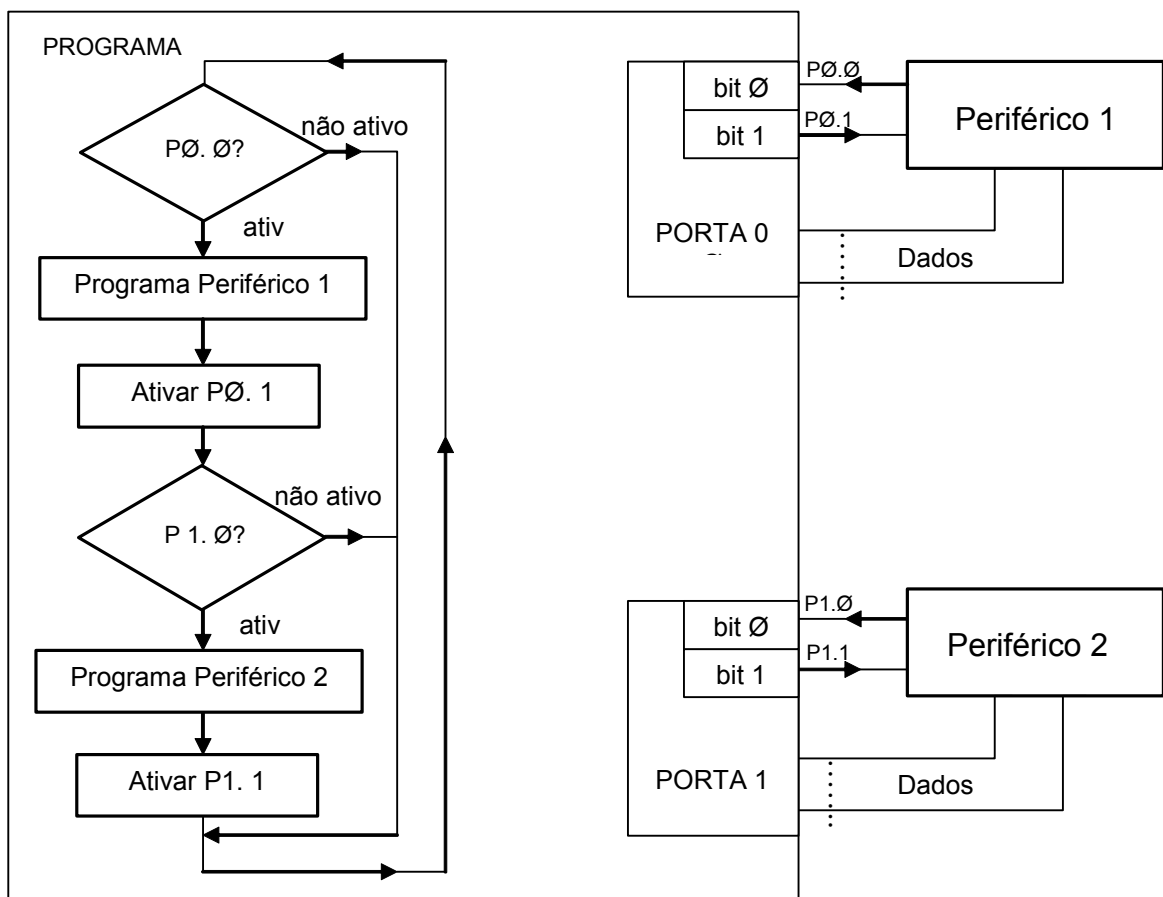


Figura 3.30 Comunicação com periféricos por consulta

- b) Interrupção: Serviço direto entre periféricos e CPU, sempre que o primeiro solicite estabelecer o diálogo. A figura 3.31 mostra o serviço de

interrupção. Este serviço tem a característica de atendimento imediato, podem eliminar-se total ou parcialmente os ciclos de espera ou pesquisa e permite inibir a interrupção quando se considera que é “inoportuna” e, portanto, prejudicial para a marcha do processo. Esta forma de trabalho é inerente ao controle de processos em tempo real.

Obviamente as interrupções permitem atender a eventos em tempo real, e se ganha em eficiência devido a que não é preciso perder tempo em pesquisar o estado de um evento externo ou interno. A importância das interrupções nasce da necessidade de executar um sub-processo no instante preciso, portanto se considera sua intervenção “urgente”. Quando termina a execução deste sub-processo, a CPU volta para programa principal, continuando sua tarefa cíclica justo onde se deixou.

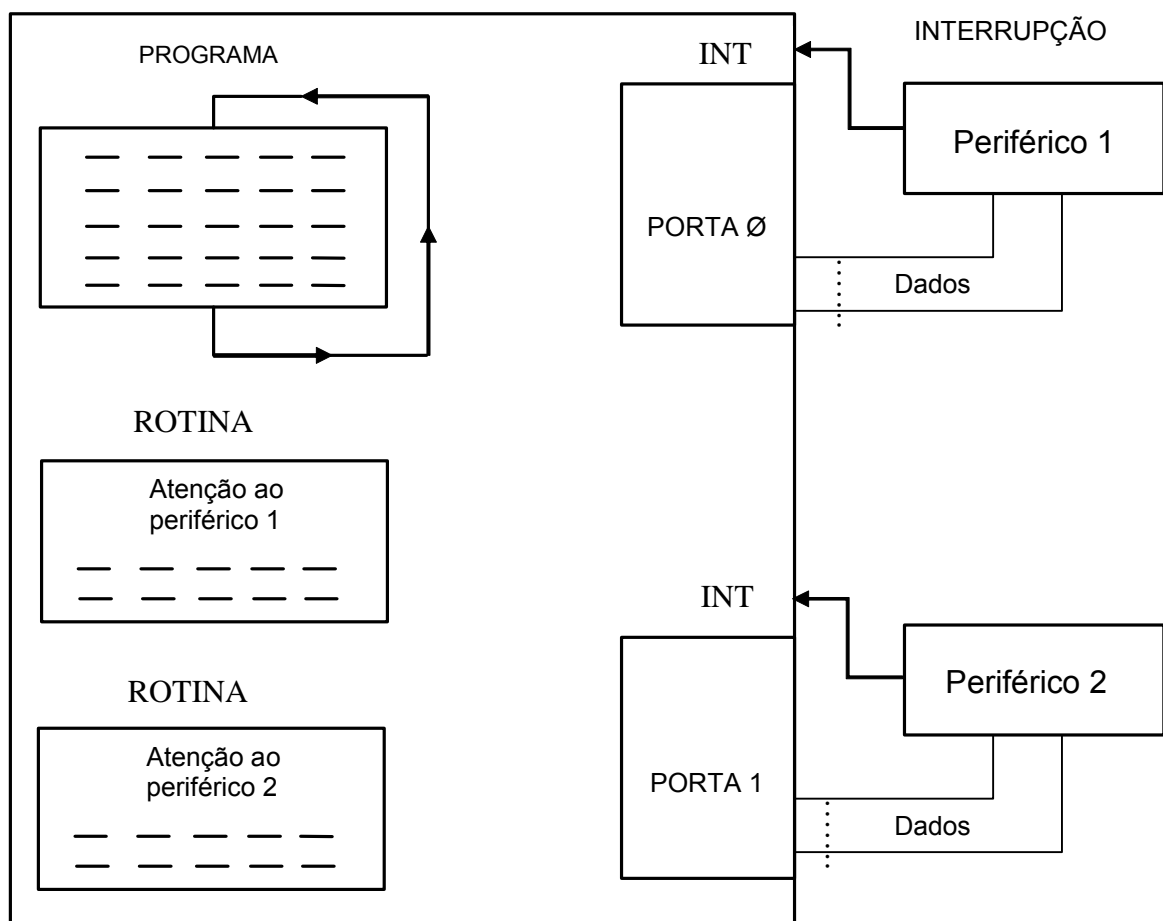


Figura 3.31 Serviço de periféricos por interrupção.

Na figura 3.32 se mostram diversas classificações das interrupções segundo seu origem, o endereço de salto (vetor de interrupção) e sua tipo de resposta.

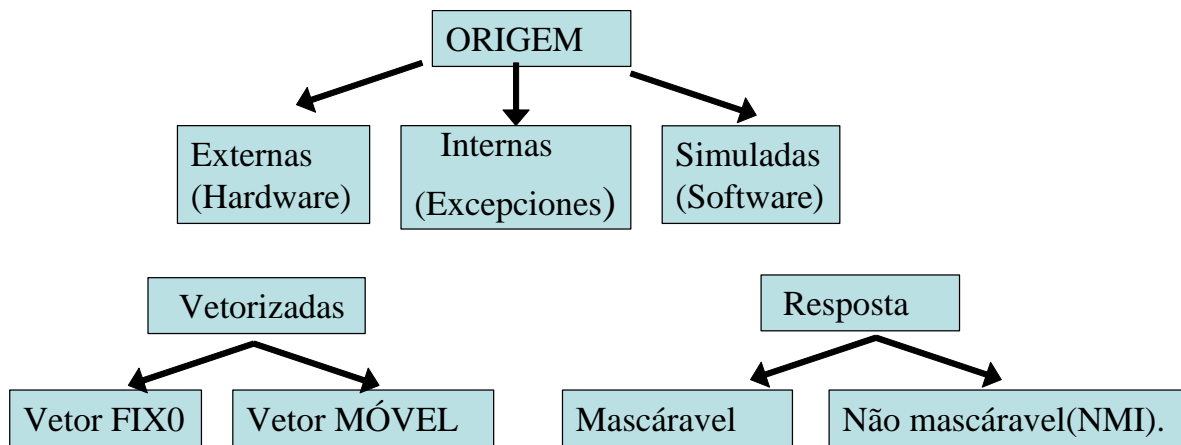


Figura 3.32 Serviço de periféricos por interrupção.

As excepciones pertencem a interrupções mais avançadas típicas de microprocessadores usados para aplicações de computo intenso como acontece em os computadores pessoais (Ex: Interrupção por divisão por zero)

As interrupções simuladas NO são verdadeiras interrupções, a pesar de que na bibliografia isto é um erro freqüente.

Uma interrupção no microcontrolador pode ser:

- Externa: solicitada por um periférico através de pinos específicos.
- Interna: solicitada por um dispositivo integrado no próprio chip do micro controlador.

Um sistema pode atender diversas solicitações de interrupções procedentes de diversas fontes. Cada uma delas deve conduzir à execução de um subprograma específico, e por tanto cada um destes subprogramas deve ter um endereço de começo que se indica normalmente em posições de memória em forma de uma tabela, que recebe o nome de tabela de vetores de interrupções.

E importante não abusar no uso das interrupções pois pode tornar ineficiente ao sistema, devido a que se o microprocessador tem que atender constantemente diferentes solicitações de interrupções então não terá tempo de executar seu programa principal.

3.8 Interrupções na família 8051

O microcontrolador 8051 tem cinco fontes de interrupções:

- 2 externas (INT0# y INT1#)
- 3 internas (Timer 0, Timer 1 y Porta Serial)

Estas interrupções são fixas. Ao aparecer uma delas, o microcontrolador guarda o conteúdo do Contador de programa (PC) na pilha e salta a um dos seguintes endereços:

INT0#	0003 h
TIMER0	000B h
INT1#	0013 h
TIMER1	001B h
Porta Serial	0023 h

O programador deve colocar nestes endereços uma instrução JMP Endereço apontando à rotina de atenção à interrupção.

Na verdade a interrupção com a mais alta ordem é o RESET o qual não pode ser inabilitada (ou mascada). Quando o RESET ocorre o programa começa a partir do endereço 0000H do programa.

Quando uma interrupção é produzida o Contador do Programa (PC) armazena seu conteúdo temporalmente dentro do *stack* ou pilha e se carrega com a endereço da localidade onde se encontra a rotina de serviço da interrupção correspondente. Uma vez posicionado nessa localidade deverá começar a execução da rotina de serviço, até que encontre a instrução RETI, que lhe permitirá ao PC recuperar novamente seu valor original armazenado no *stack*, e continuar com o programa anterior à interrupção.

Por exemplo, à interrupção 0 lhe atribui a localidade 0003H, se a interrupção não se utilizar, esta localidade pode utilizar-se para propósitos gerais do programa, se a interrupção tiver sido permitida, (estabelecendo o bit correspondente dentro do registro de controle IE), no momento que exista uma ativação da interrupção (estado desço na linha INT0) o PC se carregará com 0003 e saltará a essa localidade para começar a executar a rotina de serviço.

Estas localizações de memória dos serviços de interrupção estão separadas em intervalos de 8 bytes, entre si. Quando um serviço de interrupção é curto, este pode estar contido nos 8 bytes. No caso de que fosse comprido se pode executar um salto a outra localidade de memória para continuar com a sequência de interrupção. O termo do serviço de interrupção deverá realizar-se mediante a execução da instrução da instrução RETI.

Os *flags* ou indicadores de interrupção são amostrados no estado 5, fase 2 (S5P2) de cada ciclo de máquina. As amostras são escrutinadas durante o seguinte ciclo de máquina. O sistema de interrupções do micro controlador gera um LCALL ao apropriado vetor de interrupções. Esta situação se produz salvo que:

- Uma interrupção de igual ou maior nível de prioridade esteja nesse momento em processo.
- Não tenha finalizado a instrução que nesse momento está processando-se.
- A instrução em processo é uma RETI (retorno de interrupção) ou se esteja produzindo um acesso aos registros IE ou IP; então assegura que uma instrução mais, ao menos, será executada antes que a interrupção seja vetorizada.

Pode acontecer que um *flag* de interrupção seja ativado e não possa responder o sistema por encontrar-se em uma das situações de bloqueio comentadas anteriormente.

Nestas condições pode acontecer o seguinte:

- a) Que desapareça a situação de bloqueio e que o *flag* siga ativado. Neste caso a interrupção será tratada pelo micro controlador.
- b) Que desapareça a situação de bloqueio, mas que nesse intervalo o *flag* de solicitude de interrupção se desativou. Nesse caso a interrupção será denegada e deverá ser solicitada novamente.

Com relação ao apagado dos *flags*, uma vez servida a rotina de interrupção, tem-se que dizer que, em alguns casos, o apagado é automático, aspecto denominado como apagado por hardware; no outro caso, é o programador o que deve estar pendente de seu apagado, modalidade de apagado por software.

Não se produz apagado por hardware nas interrupções do porta serie e o timer 2.

Os *flags* das interrupções exteriores INTO e INT1 são apagados por hardware quando se ativam por flanco. Se estiverem ativadas por nível, este deve voltar para seu estado de não ativação para que saia da rotina de interrupção.

Uma vez válida a interrupção e produzido o salto à rotina de serviço, a CPU guarda no *stack* o conteúdo do contador de programa PC e não salva o registro de estado PSW.

A rotina de interrupção deve finalizar com a instrução RETI que informa à CPU de que a rotina finalizou. A continuação se recupera do *stack* os dois bytes de endereço do programa principal, instrução seguinte a que em processo recebeu a interrupção e que concluiu antes de passar ao processo de tratamento. Estes dois bytes carregados no PC permitem que se siga executando o programa principal, sem mais que sofrer uma relativa pequena perda de tempo.

As interrupções são controladas mediante a escritura nos registradores IE (*Interrupt Enable*) e IP (*Interrupt Priority*)

3.8.1 Registradores associados com as interrupções

Em princípio, as interrupções trabalham com os registradores SFR IE e TCON:

IE (*Interrupt Enable Register*). Para habilitar individualmente as interrupções.

IE (<i>Interrupt Enable Register</i>). Endereço: 0A8h; bits 0A8 a 0AFh							
7	6	5	4	3	2	1	0
EA	-	ET2	ES	ET1	EX1	ET0	EX0
Todas		Timer2 (8052)	PSerial	Timer1	INT1#	Timer0	INT0#

EA, ET, ES, EX	1: Habilita 0 : Inabilita
----------------	------------------------------

TCON (*Timer/Counter Control Register*).

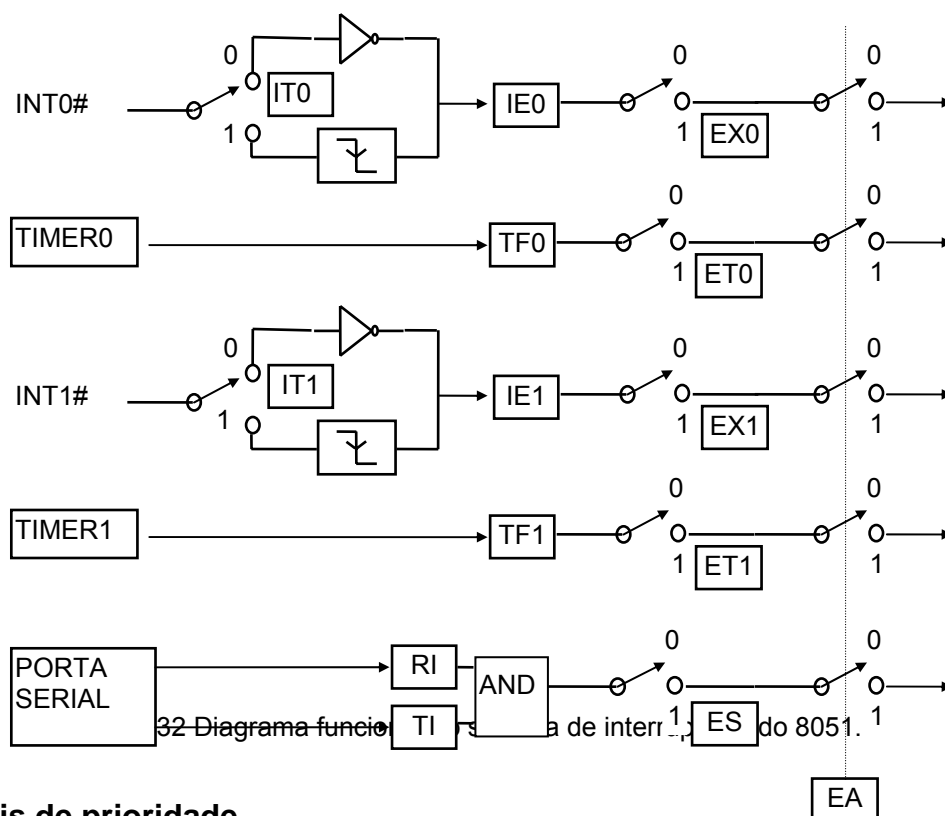
A solicitude de interrupção seta diferentes bandeiras (*flags*) que se encontram nos registradores associados a cada subsistema. Por exemplo, no caso das interrupções externas, INT0 e INT1, encontram-se no registro TCON, que corresponde ao subsistema de contagem e temporização.

O registrador de controle do Contador/temporizador é endereçável por bit, para ativar ou desativar cada uma de suas bandeiras.

TCON (<i>Timer/Counter Control Register</i>) Endereço: 88h; bits 88 a 08Fh							
7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Timer 1		Timer0		INT1#		INT0#	

IT	0: Interrupção ativa por nível 1: Interrupção ativa por flanco
TF, IE	Vão a 1 se houver solicitude de interrupção. Se a interrupção externa for por flanco, IE1, IE0 vão a 0 automaticamente quando é atendida a interrupção. Se a interrupção externa for por nível, terá que levar IE1, IE0 a 0 por software. TF1, TF0 vão a 0 por hardware.

A figura seguinte ilustra a relação entre as fontes de interrupção e as diferentes *flags*:



3.8.2 Níveis de prioridade

Cada fonte de interrupção pode programar-se em um de dois níveis de prioridade (**alto** e **baixo**) através do SFR IP.

IP (<i>Interrupt Priority Register</i>)							
Endereço: 0B8h; bits 0B8h a 0BFh							
7	6	5	4	3	2	1	0
-	-	PT2	PS	PT1	PX1	PT0	PX0
		Timer2 (8052)	PSerial	Timer1	INT1#	Timer0	INT0#
PT2, PS, PT1, PX1, PT0, PX0				1: Alta prioridade 0 : Baixa prioridade			

As regras de prioridade são (figura 3.33):

- Uma interrupção de alta prioridade pode interromper a uma de baixa prioridade, mas não é interrompida por uma de baixa prioridade.
- As interrupções de um mesmo nível de prioridade (alto ou baixo) são atendidas segundo a seguinte escala:

(Maior prioridade) **INT0#** **TIMER0** **INT1#** **TIMER1** **P. SERIE** (Menor prioridade)

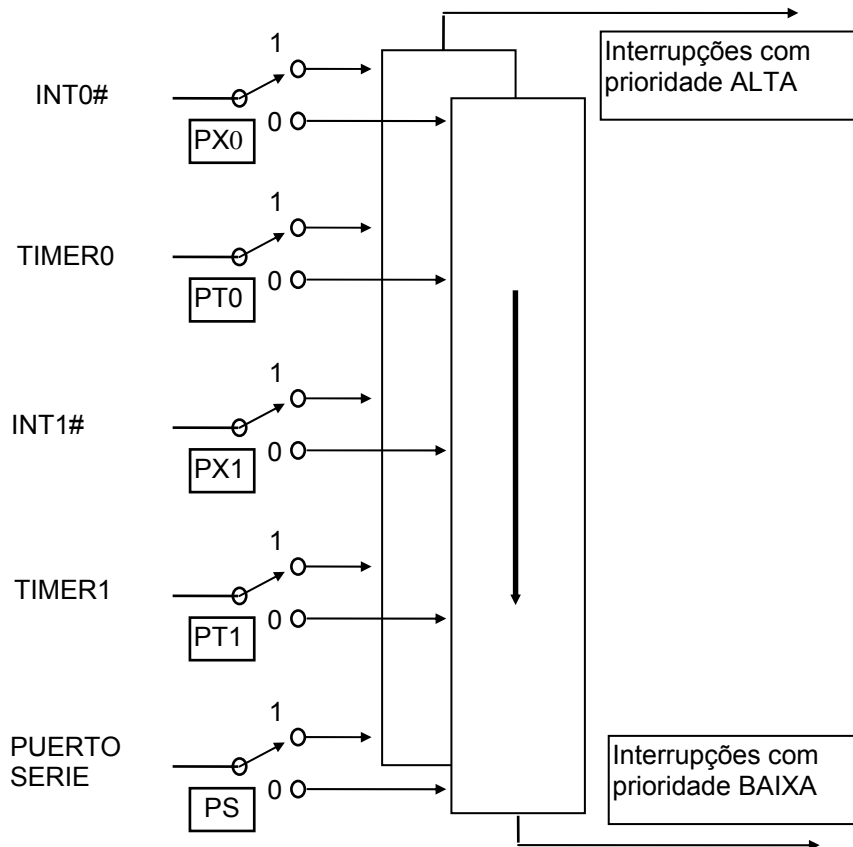


Figura 3.33 Diagrama funcional dos níveis de prioridades do sistema de interrupções do 8051.

Exercício Resolvido

```

; Progr20 :  Introduz o recurso de "Programar as INTERRUPCOES"
;
;  Uma interrupcao aceita pelo Micro faz com que o mesmo se desvie
;  para um endereco pre-definido, onde neste endereco se ira' tratar
;  a interrupcao solicitada.Isto se chama "Servico de Interrupcao".
;
;  As interrupcoes possiveis nesta maquina sao:
;
;  -INT0\,que e' fisicamente o pino-port P3.2
;
;  -INT1\,que e' fisicamente o pino-port P3.3
;
;  -TIMER0,que e' o periferico"Timer 0",que interrompe o micro
;  internamente.
;
;  -TIMER1,que e' o periferico"Timer 1",que interrompe o micro
;  internamente.
;
;  -SERIAL,que e' o periferico "Saida Serial",que interrrompe
;  o micro internamente.
;
;  Os enderecos de desvio de cada interrupcao sao:
;
;  INT0\ - endereco 0003h
;
;  INT1\ - endereco 0013h
;
;  TIMER0 - endereco 000Bh
;
;  TIMER1 - endereco 001Bh

```

```

; SERIAL - endereco 0023h
;
;
; As palavras de comando das interrupcoes sao :
; IE - Interrupt Enable
;
; .....
;
; EA    XX    XX    ES    ET1    EX1    ET0    EX0
;
; -----
;
; ^          ^      ^      ^      ^      ^
;
; |          |      |      |      |      |
;
; Habilita Geral      |      |      |      |
;
; Habilita Int. Serial __|      |      |      |
;
; Habilita Int. TIMER 1_____|      |      |
;
; Habilita Int. INT1\ _____|      |      |
;
; Habilita Int. TIMER 0_____|      |
;
; Habilita Int. INT0\ _____|
;
; obs : "1" =ENABLE
;
; IP - Interrupt Priority
;
; .....
;
; XX    XX    XX    PS    PT1    PX1    PT0    PX0
;
; -----^-----^-----^-----^-----
;
; SERIAL_____|      |      |      |
;
; TIMER 1_____|      |      |      |
;
; INT1\ _____|      |      |
;
; TIMER 0 _____|      |
;
; INT 0\ _____|
;
; obs : "1" = Prioridade Alta "0" = Prioridade Baixa (Vide livro
; no capitulo "Interrupcao" para maiores detalhes,pois esta pro-
; gramacao nao e' tao simples)
;
;
; TCON -( Parte desta palavra se usa para Interrupcao)
;
; .....
;
; XX    XX    XX    XX    IE1    IT1    IE0    IT0
;
; -----^-----^-----^-----
;
; NAO TEM APLICACAO AQUI      |      |      |      |
;
; (So' usado nos Timers)      |      |      |
;
;          |      |      |
;
; FLAG de interrupcao INT1\_____|      |      |
;
; OPCA0 de acionar INT1 por Borda/Nivel____|      |
;
; FLAG de interrupcao INT0\_____|
;
; OPCA0 de acionar INT0 por Borda/Nivel_____|

```

```
; obs : "1" = Borda "0" = Nivel
; Nao da' para explicar tudo aqui .Vide Capitulo de
; " interrupcao" no livro para maiores detalhes.

; Exemplo :
; Gerar onda quadrada no Port P1.0 (90H) atraves do Timer1 gerando
; tempo de 200+200 Microssegundos(Com Clock de 12MHz) atraves de in-
; terrupcao do Micro pelo Timer1.Enquanto o Timer1 conta, o Micro
; fica incrementando "A", no sentido teorico de mostrar que o Micro
; faz outra coisa independente do Timer ,ate' este Timer interrompe
; -lo.E' claro que o Timer1 precisa ,neste caso, ter sua interrupcao
; liberada na palavra de controle de interrupcoes.Tambem como exemplo
; libera-se o Pino INT1\ para poder interromper o Micro.Neste caso a
; subrotina desta interrupcao faz apenas zerar o "A" que estava em pro-
; cesso de incremento,mostrando que varias interrupcoes podem ocorrer
; no Micro ,cada uma com certa prioridade, e cada uma com sua subrotina
; diferente ,i.e',cada uma com seu "Servico de Interrupcao" diferente.
```

```
;label instr operando;comentarios
```

```
.***** ***** *****
;
```

```
org 0 ;Comeca em 0 (RESET)
ljmp inicio ;Vai para a rotina de incremento de "A",no
;fim do programa(Vide parte final)
```

```
;
ORG 3 ;INT0 (EX0) - Servico de interrupcao de INT0
nop ;Nao faz nada neste exemplo
reti ;Retorno de Interrupcao
```

```
;
ORG 0BH ;TIMER0 (ET0) -Servico de interrupcao do TIMER0
CPL 90h ; Neste exemplo pulsamos todo o Port1.0 atraves da
;instrucao CPL(complementa)
reti ;Retorno de Interrupcao
```

```
;
ORG 13H ;INT1 (EX1)
CLR A ;Zera "A" caso o Pino INT1\ seja acionado,
;mostrando que esta interrupcao alterou a
;rotina normal do Micro, que era incrementar "A".
;Cada vez que voce de proposito fizer o Pino INT1\
;ser zero, o Micro te atendera' e fara' "A" ser zerado
;independente de onde estava a contagem.
reti ;Retorno de Interrupcao
```

```

;
;-----
ORG    1BH    ;TIMER 1 (ET1)

nop          ;Nao faz nada neste exemplo
reti        ;Retorno de Interrupcao

```

```

;
;-----
ORG    23H    ;Serial (ES)

nop          ;Nao faz nada neste exemplo

reti        ;Retorno de Interrupcao

```

```

;.....

```

inicio: mov ie,#10000110B ;TIMER0(ET0) e INT1(EX1) podem pedir interr.

```

mov    ip,#4    ;INT1 max prioridade

mov    tmod,#2    ;TIMER0 modo 2(Contagem de 8 bits c/auto-reload)
mov    tcon,#00000100b ;EX1 sensivel a Borda (Isto e'o Pino INT1)
mov    TH0,#55    ;Move para THigh #55
mov    TL0,#55    ;Move para TLow #55
setb   TR0        ; Liga TIMER0
                ;Quando Timer contar 200,ele "estoura",
                ;pois 200 + 55 = FFh; Ai' o Bit TF0 vai
                ;para "1" e como a interrupcao deste timer
                ;esta' liberada,ele interrompe o Micro,fa-
                ;zendo o Micro ir para o endereco desta in-
                ;terrupcao ,que e'obrigatoriamente 000Bh.
                ;Neste ponto do Software eu devo tratar esta
                ;interrupcao.

mov    A,#00      ; Zera A
loop:  INC    A    ;Incrementa A enquanto o Timer fica contando.
sjmp   loop       ;Veja que eu nao estou "esperando" o Timer
                ; contar eu estou fazendo outra coisa (Incre-
                ;mentando "A" neste exemplo) ate' que o Timer es-
                ;toure a conta. Ai' eu atendo a sua interrupcao,
                ;que no caso e' piscar o Port P1.0 (90h) atraves
                ;da instrucao CPL P1.0,e apos ter complementado
                ;P1.0, eu retorno da interrupcao (RETI),voltando
                ;ao meu antigo servico que era incrementar" A".
                ;Se eu acionar o pino INT0,via simulacao(faca
                ;isto),ele vai atender esta segunda e mais prio-

```


END

- 2 teclas conectadas às entradas #INT0 e #INT1
- Quando se pressiona a tecla associada a cada pino, se escreve em cada linha de display LM016 um cartel com o nome do pino (INT0 na primeira linha 0 e INT1 na segunda linha).
- Cada tecla pressionada apaga o cartel da tecla anterior.
- Quando ambas as teclas fiquem presionadas, mostrar a ultima que foi pressionada.
- $f_{XTAL} = 12 \text{ MHz}$.

O software em linguagem C se mostra a seguir:

```
#include <at89s8252.h>
void External0_ISR(void) interrupt 0; // INT0
void Timer0_ISR(void) interrupt 1; // Timer0
void External1_ISR(void) interrupt 2; // INT1
//void Timer1_ISR(void) interrupt 3; // Timer1
//void Serial_ISR(void) interrupt 4; // serial RX
void delay(void)
{
    int i;
    for(i=0;i<8;i++);
}
void delay2(void)
{
    unsigned int i;
    for(i=0;i<6000;i++);
    // for(i=0;i<60000;i++);
}
void Dados_instrucao(unsigned char comando,unsigned char dados)
{
    P0 = dados;
    if (comando == 0)
        P3_7 = 0; //instrucao
    else
        P3_7 = 1; //dado
    //Enable do LCD
    P3_6 =1;
    P3_6 =0;
    delay();
}
void Inicializa_LCD(void)
{
    //0 - Instrucao
    //1 - Dados
    Dados_instrucao(0,0x38);//Instrucao para informar ao LCD qual o numero de linhas
    Dados_instrucao(0,0x06);// Desloca o curso para Direita
```

```

    Dados_instrucao(0,0x0e);//Liga o cursor
    Dados_instrucao(0,0x01);//Limpa o display
}
void EscreveLcd(char *msg)
{
    for(;*msg!=0;msg++)
        Dados_instrucao(1,*msg);
}
void converte(int valor)
{
    int aux;
    unsigned char envia;

    aux = valor; //contador
    aux = aux / 1000;
    envia = (unsigned char)aux + 0x30;
    Dados_instrucao(1,envia); //primeiro digito
    aux = valor % 1000;

    valor = aux; //centena
    aux = aux / 100;
    envia = (unsigned char)aux + 0x30;
    Dados_instrucao(1,envia); //segundo digito
    aux = valor % 100;

    valor = aux; //dezena
    aux = aux / 10;
    envia = (unsigned char)aux + 0x30;
    Dados_instrucao(1,envia); //terceiro digito
    aux = valor % 10;

    envia = (unsigned char)aux + 0x30;
    Dados_instrucao(1,envia); //quarto digito
}
void tempo(void)
{
    TMOD = 0x05; // Modo 1 funcionando como contador

```

```

//TH0 = 0xd8;
// TL0 = 0xf0;

TH0 = 0x00;
TL0 = 0x00;

TR0 = 1;//inicio da contagem
TF0 = 0;
while (TF0 == 0)
{
    Dados_instrucao(0,0xc7);//posiciona na segunda linha
    converte(TL0);
}
}

void External0_ISR(void) interrupt 0
{
    i Dados_instrucao(0,0x80);
    EscreveLcd("INT 0");
    Dados_instrucao(0,0xc0);
    EscreveLcd("  ");
}

void Timer0_ISR(void) interrupt 1 // Timer0
{
}

void External1_ISR(void) interrupt 2 // INT1
{
    Dados_instrucao(0,0xc0);
    EscreveLcd("INT 1");
    Dados_instrucao(0,0x80);
    EscreveLcd("  ");
}

void main(void)
{
    int i;
    unsigned char x;

```

```

TMOD=0x11; // Temporizador T0 e T1 no modo 1
ET0=1;    // Habilita interrupcao do timer 0
TR0=1;    // Liga timer 0

EX1=1;    // Habilita interrupcao Externa 1 INT1
IT1=1;    // interrupcao externa 1 (INT1) ativada por nivel 0 - ativada por nivel
           //                                     1 - ativada por borda
EX0=1;    //Habilita interrupcao Externa 0 INT0
IT0=1;    // interrupcao externa 0 (INT0) ativada por nivel 0 - ativada por nivel
           //                                     1 - ativada por borda

EA=1;     //Habilita todas as interrupcoes

// IT0=0;   // interrupcao externa 0 (INT0) ativada por nivel 0 - ativada por nivel
           //                                     1 - ativada por borda

TF0 = 1;

Inicializa_LCD();

for(;;){

    }

}

```

3.9 Porta serial

3.9.1 Introdução

A transmissão série de informação binária consiste no envio, um a um, dos bits de uma palavra. Assim, por exemplo, o dado B2h = 10110010b pode ser irradiado mediante um sinal que represente o 0 com um nível de voltagem baixo (V_L) e o 1 com um nível de voltagem alta (V_H) e onde a duração de um 1 ou um 0 seja constante, como se mostra na figura 3.34.

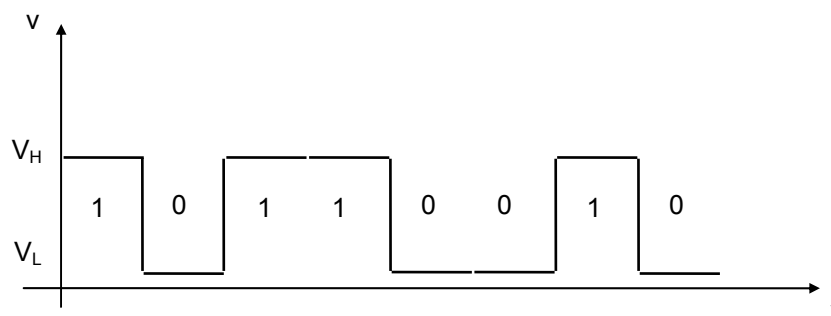


Figura 3.34 Transmissão serial (bit a bit) de informação.

Uma forma de caracterizar este sinal é mediante o parâmetro Velocidade de Transmissão (V_T), definida como o inverso da duração de um bit. Se cada bit dura τ segundos, a velocidade de transmissão é então:

$$V_T = \frac{1}{\tau} \text{ Baudios (Bd)}$$

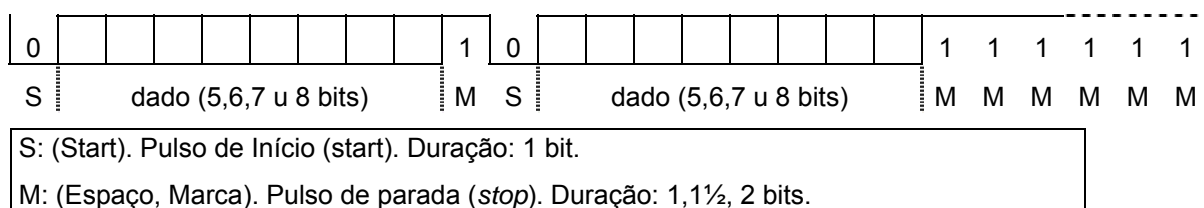
É evidente que entre o transmissor e o receptor deve existir certa sincronização de modo que a informação transmitida possa ser interpretada corretamente pelo receptor. Esta sincronização deve lhe permitir ao receptor conhecer a duração de cada bit e o momento em que começa cada palavra transmitida. A primeira destas condições pode obter-se se o transmissor e o receptor têm um relógio de referência de igual frequência. A segunda condição, ou seja, o conhecimento por parte do receptor do momento em que começa uma nova palavra pode obter-se de duas formas diferentes conhecidas como:

1. Comunicação assíncronica.
2. Comunicação síncrona.

Em ambas as formas se introduz certa informação redundante aos efeitos de obter a sincronização necessária entre transmissor e receptor.

3.9.2 Comunicação assíncronica

Consiste em introduzir um elemento de sincronização por cada dado irradiado, que não é outra coisa que um bit (0) para indicar o começo de cada palavra e outro bit (1) para indicar o final. Quando o transmissor não tem palavras que transmitir, envia uma sequência fixa de 1's. O sinal tem a seguinte forma:



Observe como a sincronização do receptor tem lugar em cada dado transmitido. As velocidades mais utilizadas na transmissão assíncrona são: 50, 75, 110, 150, 300, 600, 1200, 2400, 4800 e 9600 bd.

Conexão entre equipes: interfase RS-232C

O estabelecimento de uma comunicação a distância requer a participação de vários equipamentos que podem agrupar do seguinte modo:

1. Equipamentos que produzem o sinal de dados ou som os receptores finais do sinal de dados.
2. Equipamentos que adequam o sinal de dados ao meio de transmissão utilizado ou recebem este sinal do meio de transmissão oferecendo a de forma apropriada ao receptor final.

Dentro dos primeiros equipamentos podemos emoldurar, por exemplo, ao microcomputador que gera o sinal de dados no formato assíncrono visto anteriormente. Se este sinal tiver que transmitir-se a outro microcomputador através de um canal telefónico, então é necessário utilizar alguma equipe intermédia para adequar o sinal de dados ao canal telefónico no lado transmissor e vice-versa no lado receptor. Esta equipe é conhecida pelo MODEM (modulador - demodulador) e estará emoldurado dentro das equipes da segunda categoria indicada anteriormente.

O CCITT denominou a todas as equipes da primeira categoria com o nome de Equipes Terminais de Dados (DTE: *Data Terminal Equipments*) e os da segunda como Equipes de Comunicação de Dados (DCE: *Data Communication Equipments*), de modo que o sistema de comunicação (de dados) pode ficar como se ilustra na figura 3.35.

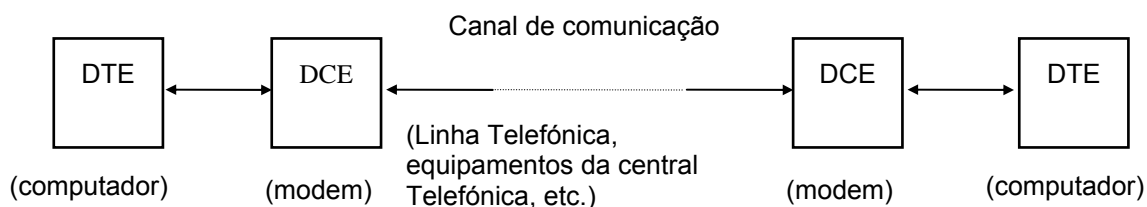


Figura 3.35 Modelo de sistema de comunicação segundo CCITT

A conexão entre o DTE e o DCE se encontra normalizada pelo CCITT, existindo várias normas ao respeito. Uma das mais utilizadas é a relativa à comunicação em modo assíncrono para velocidades baixa e meias (até 9600 bd) dada na Recomendação V.24 do Livro Branco de 1969. Esta norma é também conhecida como RS-232C, pois foi “proposta” originalmente pela EIA (*Electronic*

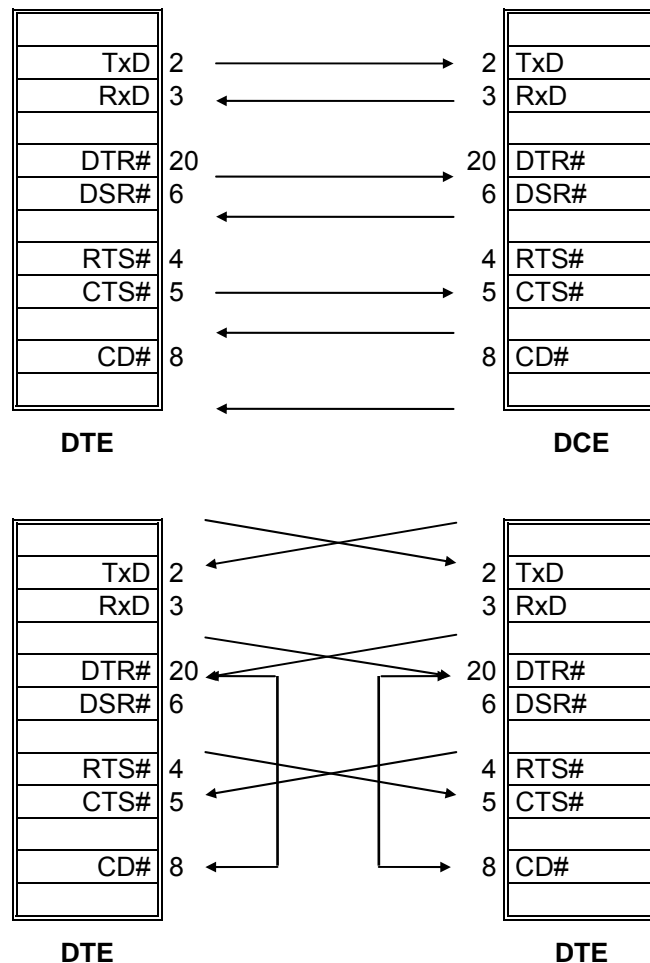
Industries Association, USA) para a conexão entre equipes de dados a pequena distância (< 50 pés (16,4 m)) em um ambiente ruidoso. Esta norma se popularizou de tal forma que virtualmente tudo os microcomputadores pessoais trazem um interfase RS-232C e seu conector para a comunicação com modems e periféricos. A norma especifica 25 sinais. Os níveis do 0 e o 1 são:

0	Entre +3 e +15 volts com carga, até +25 volts sem carga. Normalmente se utiliza +12 volts.
1	Entre -3 e -15 volts com carga, até -25 volts sem carga. Normalmente se utiliza -12 volts.

Sinais da interface RS-232C				
Conector		Sinal	From DCE	To DCE
25D	9D			
1		Protective Ground (GND)		
2	3	Transmitted Data (TxD)		x
3	2	Received Data (RxD)	x	
4	7	Request to Send (RTS)		x
5	8	Clear to Send (CTS)	x	
6	6	Data Set Ready (DSR)	x	
7	5	Signal Ground (SG)		
8	1	Rcvd Line Signal Detect (DCD)*	x	
9		-		
10		-		
11		Select Standby		x
12		-		
13		-		
14		-		
15		Transmit Signal Element Timing	x	
16		-		
17		Receiver Signal Element Timing	x	
18		Test	x	
19		-		
20	4	Data Terminal Ready (DTR)		x
21		-		
22	9	Ring Indicator (RI)	x	
23		Speed Select		x
24		-		
25		-		

* Data Carrier Detect

Conexão DTE-DCE e DTE-DTE mediante o interfaz RS-232C:



3.9.3 Comunicação síncrona

Na comunicação assíncrona, ao realizá-la sincronização caracter a caracter (bits *start-stop*) perde-se 20% do tempo de transmissão (10 bits transmitidos por cada 8 bits de dados). Por outro lado, a comunicação síncrona é mais eficiente, pois uma vez sincronizados o transmissor e o receptor, envia-se um bloco de dados. Quando o transmissor termina de enviar um bloco de dados e não há mais dados que enviar, a linha se mantém em um estado determinado (em 1) (*marking condition*). Para indicar o início da transmissão de um bloco, enviam-se caracteres de sincronização (*sync characters*) ou um padrão único de bits (flag) em dependência do sistema utilizado.

Tanto para a comunicação assíncrona como síncrona pela linha telefônica usando modems, necessita-se um conjunto de sinais para o *handshake* básico. Além deste *handshaking*, requer-se um nível maior de coordenação ou protocolo entre

transmissor e receptor. Neste caso o protocolo é um conjunto de regras acordadas que asseguram a transferência ordenada dos dados.

Existem:

- Protocolos orientados a bytes (*byte oriented*):

BISYNC: *IBM Binary Synchronous Communications Protocol*.

- Protocolos orientados a bit (*bit oriented*):

HDLC: *High -level Data LinControl Protocol*.

SDLC: *Synchronous Data Link Control Protocol*. (IBM).

- Protocolos para comunicação semi-duplex:

BISYNC.

- Protocolos para comunicação full-duplex:

HDLC/SDLC.

- Protocolos para comunicação multiponto:

Em sistemas hierárquicos:

HDLC/SDLC.

Em sistemas não hierárquicos:

CSMA/CD: *Carrier Sense, Multiple Access with Collision Detection*. (IEEE 802.3: *Ethernet Network Standard*)

Há diferentes circuitos VLSI para suportar estes protocolos:

8251A: *Universal Synchronous/Asynchronous Receiver/ Transmitter (USART)*: BISYNC.

8273: *High level peripheral for HDLC/SDLC protocol support*: HDLC/SDLC.

8274: Multi-protocol serial controller (MPSC): Assíncrono, BISYNC, HDLC/SDLC.

82530: *Serial Communicatios Controller (SCC)*: Assíncrono, BISYNC, HDLC/SDLC.

Protocolo BISYNC

BISYNC é um protocolo orientado a bytes (*byte-controlled protocol BCP*) pois utiliza caracteres ASCII ou EBCDIC para indicar o início da mensagem e intercambiar handshake entre transmissor e receptor.

O formato de uma mensagem no protocolo é o seguinte:

SYNC	SYNC	SOH	HEADER	STX	TEXTO	ETX	BCC
------	------	-----	--------	-----	-------	-----	-----

SYNC: Caráter de sincronização 16h).

SOH: Start of Header (01h).

HEADER:

STX: Start of Text (02h).

TEXTO: 128 ó 256 caracteres ASCII o EBCDIC.

ETX: End of Text ().

BCC: Block Check Characters (1ó 2 caracteres).

Como se efectúa a comunicação:

TX: Envia ENQ (05h) [enquiry].

RX: Se não estar preparado para receber, responde com NAK (15h) [negative acknowledge].

Se estiver preparado para receber, responde com o ACK (06h) [affirmative acknowledge].

TX: Se tiver recebido um NAK, espera e envia um novo ENQ.

Se tiver recebido um ACK, envia a mensagem.

RX: Se a mensagem foi recebido incorretamente, envia um NAK ao TX e este repete a mensagem.

Se a mensagem foi recebido corretamente, envia um ACK ao TX e este envia a seguinte mensagem.

Protocolo HDLC/SDLC

HDLC foi proposto pela ISO. SDLC foi desenvolvido pela IBM. Ambos os protocolos são muito parecidos.

HDLC é um protocolo orientado a bits (*bit-oriented protocol BOP*). As mensagens são um grupo de bits em lugar de ser um grupo de caracteres (bytes). O grupo de bits que conformam uma mensagem se denomina bloco (*frame*). Há três tipos de blocos:

1. Bloco I (*Information*).
2. Bloco S (*Supervisory control sequencies*).
3. Bloco U (*Command / response*).

Um bloco se divide em campos (fields).

O formato de um bloco I é o seguinte:

FLAG (7Eh)	Direção (8 bits)	Controle (8 bits)	Informação (qualquer número de bits)	Verificação (16 bits)	FLAG (7Eh)
---------------	---------------------	----------------------	---	--------------------------	---------------

Observe que o campo FLAG contém uma seqüência de seis uns (7E=01111110). Para evitar que uma seqüência qualquer de 6 uns presente nos dados a transmitir se confunda com um campo FLAG, o transmissor previamente acrescenta um zero a toda seqüência de 5 uns e o receptor, tira-o.

HDLC se usa para comunicações entre 2 ou mais sistemas em um enlace de dados. Uma das estações deve ser o controlador do enlace ou estação primária; as outras são estações secundárias.

Suponhamos que a estação primária quer enviar vários blocos de informação a uma estação secundária.

A estação primária começa por enviar um bloco S com a direção da estação secundária e uma palavra de controle perguntando se a estação está lista para receber. A estação secundária responde com um bloco S com sua direção e uma palavra de controle indicando que está lista para receber. A estação primária envia então a seqüência de blocos de informação (quadros I).

3.10 A Porta Serial do 8051

Comunicação serial

Um sistema microprocessador não tem por que ser um sistema isolado, mas sim pode comunicar-se com o exterior. Existem dois tipos básicos de comunicação de dados: paralelo e serial. Na comunicação paralela se transmite informação simultaneamente por múltiplas linhas e na comunicação serie se transmite informação por uma só linha, sendo esta última a que nos ocupa a seguir.

A comunicação serial se realiza entre dois dispositivos distantes, que não têm por que ser igual. Devido a isto tem que existir um acordo sobre como se realize a comunicação, existe uma normativa que define os meios de transmissão, os tipos de transmissão, os protocolos.

Tipos de comunicação serial:

- Sincrônica:
 - Rx/Tx.
 - Relógio.
- Assíncrono (UART):
 - Rx.
 - TX.

O mais utilizado é o modo assíncrono.

A recepção, no modo assíncrono, de bits individuais de cada caractere está "sincronizada" mediante um bit adicional aos bits do caractere chamado bit do START (bit de arranque). O bit do START é um bit de sincronização justo antes do primeiro bit de cada caractere, e é de igual natureza que os bits do caractere. Depois do último bit de cada caractere existe um novo bit de sincronismo chamado bit do STOP, que identifica o último bit do caractere. Os bits do START e STOP isolam o caractere de seus homólogos circundantes.

Existe uma fórmula de detecção de erros que se realiza mediante a adição de um bit de paridade (este bit se transmite depois do último bit do caractere e antes do bit do STOP). Gera-se o bit de paridade mediante a regra em que todos os caracteres têm um número par (ou ímpar) de bits a "1". Daí surge as opções paridade par ou ímpar. Se escolher paridade par, por exemplo, o bit de paridade será 0 se o número de uns do caractere é número par e será 1 em caso contrário. De igual modo se calcula este bit com paridade ímpar, para manter ímpar o número de uns transmitidos (figura 3.18).

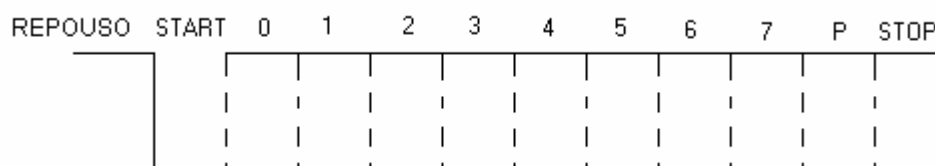


Figura 3.36 Formato do Tx/RX em modo assíncrono.

Quando não se transmitem bits se mantém a linha em estado lógico alto. Na transmissão série se começa pelo bit de menor peso e o último em ser transmitido é o bit de maior peso.

3.6.2 A comunicação através do RS-232

As portas de comunicação serial utilizados nos ordenadores PC/AT se ajustam ao padrão EIA Rs-232-C e também ao padrão de V.24 do CCITT (são normas quase idênticas).

O protocolo RS-232-C responde definição "Interface entre um dispositivo terminal de dados e um dispositivo de comunicação de dados empregando intercâmbio de dados binários em série". O dispositivo terminal de dados (DTE) é o termo genérico utilizado para catalogar ao computador e o dispositivo de comunicação de dados (DCE) referência ao modem (**MO**dulador-**DE**Modulador). A norma RS-232-C define três campos necessários para a comunicação DTE/DCE:

descrição mecânica dos circuitos de interface, descrição funcional dos circuitos de intercâmbio e características dos sinais elétricos.

O padrão RS-232 define um nível lógico bipolar, na definição existem tanto os níveis de tensões como sua polaridade.

Definem-se quatro níveis lógicos. Entrada-las têm distintas definições que as saídas e os sinais TD e RD têm distintas funções que os sinais de controle.

Os níveis binários de entrada vão de +3 V a +15 V e de -3 V a -15 V. Tensões entre -3 V e +3 V estão indefinidos.

As tensões utilizadas na RS-232 não são os níveis que se utilizam nos ordenadores, por isso se precisa uma conversão de níveis. Esta conversão se faz com circuitos inversores, com o qual o sinal tem que investir-se na transmissão e na recepção. Exemplo: MAX232

3.6.3 Comunicação serial no microprocessador 8051

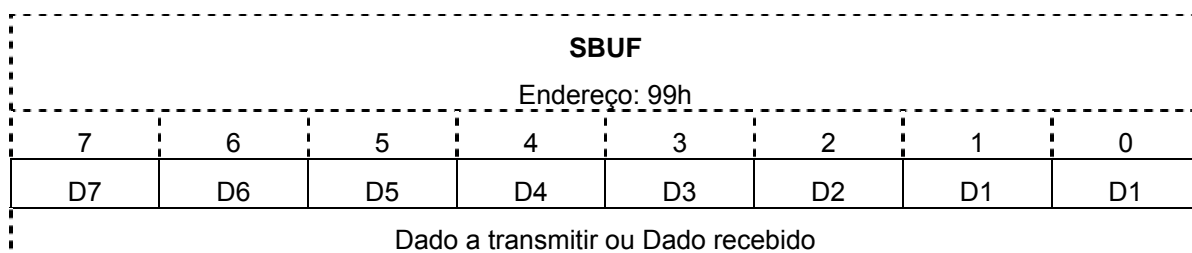
Por outra parte a comunicação serie no microprocessador é tratada de uma forma mais simples e se realiza através dos pinos 10 e 11 (Rx e Tx) que correspondem a sua vez com os bits 0 e 1 do porta 3 (P3.0 e P3.1)

Esta porta serie trabalha no modo *full duplex*, o que significa que pode transmitir e receber simultaneamente. Como receptor tem um *buffer* que lhe permite receber um segundo byte, antes que o byte previamente recebido tenha sido lido pelo registro receptor. Se a leitura do primeiro byte não se realizar no que acima o segundo, um dos dois se perde. Aos "registros" receptor e transmissor das portas serie se acessa por um único registro denominado SBUF e que esta situado na endereço 99H do SFR (*Special Function Register*). Escrevendo no SBUF se carrega o byte a transmitir e lendo do SBUF se acessa ao byte recebido.

A porta serial do 8051 permite comunicação *full-duplex*, ou seja, pode-se transmitir e receber simultaneamente.

O SFR SBUF atua como registrador de entrada/saída da porta serial: o dado a transmitir se escreve previamente no SBUF e o dado recebido se lê no SBUF, uma vez completada sua recepção. Pode-se estar recebendo um dado ainda quando o dado recebido anteriormente não tenha sido lido do SBUF, entretanto, uma vez completada a recepção do novo dado, o anterior se pede se não se leu SBUF.

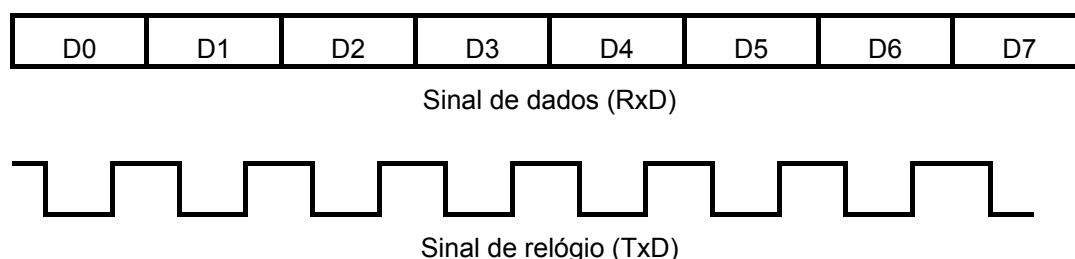
Em transmissão ou recepção, o primeiro bit que se envia ou recebe é D0 (o bit menos significativo do dado).



A porta serial pode ser operada em um de quatro modos diferentes.

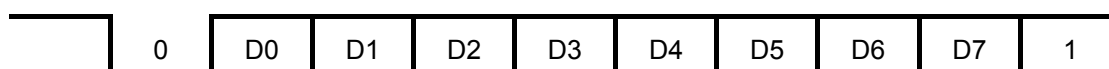
Modo 0 (Registro de deslocamento)

A porta trabalha como um transmissor ou receptor de dados de 8 bits pelo pin RxD. O pino TxD é saída do sinal de relógio (um pulso por cada bit de saída). A velocidade de transmissão é constante e igual a $F_{OSC}/12$.



Modo 1 (UART de 8 bits)

A porta trabalha como um transmissor (pelo pino TxD) ou receptor (pelo pino RxD) de dados em modo assíncrono (1 bit de start (0), 8 bits de dados, 1 bit de stop (1)). A velocidade de transmissão é variável e se obtém mediante o Timer 1.

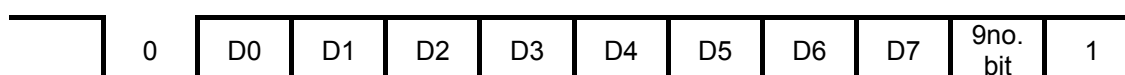


Modos 2 e 3 (UART de 9 bits)

A porta trabalha como um transmissor (pelo pino TxD) ou receptor (pelo pino RxD) de dados em modo assíncrono (1 bit de start (0), 8 bits de dados, 1 bit programável (9no. bit), 1 bit de stop (1)). A velocidade de transmissão é programável:

Em modo 2, $V_T = F_{OSC} / 32$ ó $F_{OSC} / 64$.

Em modo 3, VT se obtém mediante o Timer 1.



Em transmissão, o 9no. bit é o bit TB8 do SFR SCON.

Em recepção, o 9no. bit se pode ler no bit RB8 do SCON. Além disso, pode-se programar que se gere uma interrupção ao receber um dados com RB8 = 1, o qual é útil para implementar a comunicação entre microcontroladores em um sistema multiprocessador com um amo e vários escravos. Em resumo, os modos 2 e 3 estão desenhados para facilitar a operação em um ambiente multiprocessador.

Controle da porta serial

O controle da porta serial é feito através do registrados SFR SCON:

SCON							
Endereço: 98h, bits 98h a 9Fh							
7	6	5	4	3	2	1	0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI
Modos 0, 1, 2 e 3			Habilitar recepção	9no. bit		Interrupção	

SM0	SM1	Modo	Descrição	V _T
0	0	0	Registrador de deslocamento	F _{OSC} / 12
0	1	1	UART de 8 bits	Variável (Timer 1)
1	0	2	UART de 9 bits	F _{OSC} / 12 ou F _{OSC} / 64
1	1	3	UART de 9 bits	Variável (Timer 1)

SM2	Em modo 2 ou 3: Se SM2=1, então se solicita interrupção por recepção (o bit RI se ativa) quando se receber o 9no. bit em um (RB8=1). Se SM2=0, não se solicitará interrupção por recepção (o bit RI não se ativa) quando se receber o 9no. bit em um (RB8=1) Em modo 1: Se SM2=1, então se solicita interrupção por recepção (RI se ativa) ao receber um bit de stop válido. Se SM2=0, não se solicita interrupção ao receber um bit de stop válido. Em modo 0, SM2 deve estar em 0.
REN	Habilitar recepção por software: Se REN=1, recepção habilitada; se REN=0, recepção inabilitada.
TB8	Aqui se coloca o valor do 9no. bit que se transmitirá nos modos 2 e 3.
RB8	Nos modos 2 e 3, é o 9no. bit recebido. Em modo 1, se SM2=1, é o bit de stop recebido. Em modo 0 não tem significado.
TI	Bandeira de interrupção em transmissão: fica em 1 por hardware ao completá-la transmissão de um dado. Terá que pô-la em 0 por software.
RI	Bandeira de interrupção em recepção: fica em 1 por hardware ao completá-la recepção de um dado. Terá que pô-la em 0 por software.

Obtenção da velocidade de transmissão

No modo 0, a velocidade de transmissão é fixa e igual a:

$$V_T (\text{modo 0}) = F_{OSC} / 12$$

No modo 2, a velocidade de transmissão pode ser $V_T = F_{OSC} / 32$ ou $F_{OSC} / 64$.

A seleção se realiza mediante o bit SMOD do SFR PCON. (PCON: Power Control Register, direção 87h; SMOD é o bit 7 do PCON). Se SMOD=0 (valor por defeito), então o divisor da frequência do oscilador é 64; se SMOD=1, o divisor é 32. pode-se escrever:

$$V_T (\text{modo 2}) = 2^{SMOD} \cdot F_{OSC} / 64$$

Nos modos 1 e 3, a velocidade de transmissão está determinada pela frequência de transbordamento do timer 1 según:

$$V_T (\text{modos 1 e 3}) = (2^{SMOD} / 32) \cdot (\text{frec. desbord. timer 1})$$

Para esta aplicação, a interrupção do timer 1 pode ser inabilitada e o timer se trabalha como temporizador (contador de ciclos de máquina). Se o timer 1 se operar em modo 2 (timer de 8 bits com recarga automática), com o número carregado em TH1 se podem obter diferentes velocidades de transmissão segundo a expressão:

$$V_T (\text{modos 1 e 3}) = (2^{SMOD} / 32) \cdot (F_{OSC} / (12 \cdot (2^8 - TH1)))$$

A seguir se apresenta uma tabela com alguns valores de V_T , F_{OSC} , SMOD e TH1:

V_T (modos 1 e 3)	F_{OSC}	SMOD	TH1
19200 Bd	11,059 MHz	1	FDh
9600 Bd	11,059 MHz	0	FDh
4800 Bd	11,059 MHz	0	FAh
2400 Bd	11,059 MHz	0	F4h
1200 Bd	11,059 MHz	0	E8h
110 Bd	6,0 MHz	0	72h

3.6.4 Exemplos de programas de comunicação serial

Exemplo 1: Modo 0

Neste programa vai-se comprovar a transmissão em modo-0 e dentro dela a base de tempo que aparece no pin 11 (RXD) e a velocidade de dados pino 10 (TXD). Neste modo os 8 bits de dados são transmitidos e se recebe pelo pino RX,

começando pelo bit menos significativo. A velocidade é fixada 1/12 da frequência de oscilação.

Listado do programa

```

0000                                ORG 0000H
0000 020030                        LJMP MAIN
0030                                ORG 0030H

0030 759800      MAIN:  MOV SCON, #00H      ; em modo 0
0033 C29         CLR TI                    ; Preparado para transmitir

0035 7599AA      LOOP:  MOV SBUF,#0AAH      ; Dado a transmitir
                                JNB TI,$      ; Espera pelo fim do TX
                                CLR TI        ; Limpa a bandeira para
                                                ; voltar a transmitir

003D80F6                                JMP LOOP
                                END

```

Exemplo 2: Modo 1

No seguinte programa se pretende comprovar a transmissão em modo-1 calculando a velocidade de bits pelo pino 11(TXD) a qual depende do dado carregado em TH1, neste caso F0H.

Listado do programa

```

0000                                ORG 0000H
0000 020030                        LJMP MAIN
0030                                ORG 0030H

0030 759840      MAIN:  MOV SCON, #40H      ; Configuração modo 1
0033 58920         MOV TMOD,#20H          ; fica o Timer-1 em autocarga
                                                ; de 8 bits

0036 758DF0         MOV TH1, #0F0H        ; Configura-se a velocidade
0039 758840         MOV TCON,#40H        ; Se arranca o Timer-1
003C C299         CLR TI
003E 75992F      LOOP:  MOV SBUF, #02FH    ; Se transmite e se espera o fim
0041 3099FD        JNB TI, $              ; Início da transmissão
0044 C299         CLR TI
0044 C299         JMP LOOP
                                END

```

Exemplo 2: Modo 2

Neste exemplo se realiza uma transmissão em modo 2 onde se escolhe uma relação para a velocidade de 1/64 da frequência do relógio do sistema e se acrescenta um bit 11 configurável pelo usuário o qual é um espelho do bit TB8 do registro SCON e usualmente é usado como bit de paridade. Note-se que este se encontra ao final dos 8 bits de dados e antes do de parada, neste caso sempre permanecerá em zero.

Listado do programa

```
0000                                ORG 0000H
0000 020030                        LJMP MAIN

0030                                ORG 0030H
0030 759880      MAIN:  MOV SCON, #80H      ; Configuração do modo-2
0033 C299        LOOP:   CLR TI              ; Início da transmissão
0035 7599AA                        MOV SBUF, #0AAH      ; Dado a transmitir
0038 3099FD                        JNB TI, $            ; Espera até finalizar a Tx
003B 80F6                        JMP LOOP
                                END
```

Para terminar de comprovar todos os modos se propõe um quinto programa, o qual trabalhará em modo-3 e onde sua única diferença com o modo-1 consiste em inserir um bit configurável pelo usuário similar ao exemplo anterior dando como resultado 11bits a transmitir ou a receber. Neste caso o resto do programa ficaria similar ao exemplo 3 e este bit sempre seria zero.

Listado do programa

```
0000                                ORG 0000H
0000 020030                        JMP MAIN

                                ; Outro segmento

0030                                ORG 0030H
0030 7598C0      MAIN:  MOV SCON, #0C0H      ; Configura o modo-3
0033 758920                        MOV TMOD, #20H      ; Fica o Timer-1 em autocarga.
                                ; de 8 bits
0036 758DDD                        MOV TH1, #0DDH      ; Se configura a velocidade.
0039 758840                        MOV TCON, #40H      ; Se arranca o Timer-1
003C C299        LOOP:   CLR TI              ; Início da transmissão.
003E 74AA                        MOV A, #0AAH
```

0040 A2D0	MOV C, P	
0042 929B	MOV TB8, C	; Se envia o bit configurável.
		; como bit de paridade.
0044 F599	MOV SBUF, A	
0046 3099FD	JNB TI, \$	
0049 80F1	JMP LOOP	
	END	

3.6.5 Comunicação entre vários microcontroladores 8051

Os modos 2 e 3 permitem interligar vários 8051, sendo um mestre e vários escravos (figura 3.37).

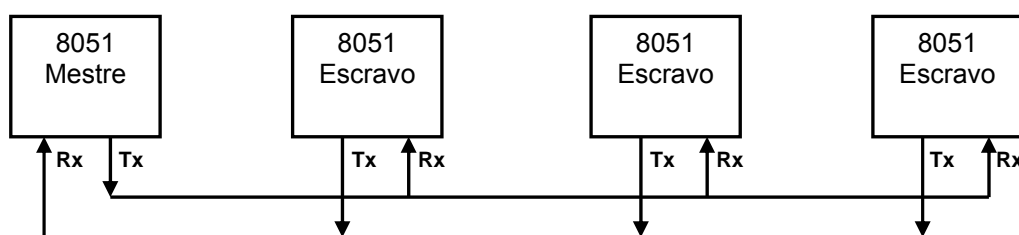


Figura 3.37 Esquema de comunicação entre vários 8051

Nestes modos temos:

- 1 start bit;
- 8 bits de dados;
- Um nono bit que vai para o bit RB8 (na recepção) ou pode ser escolhido 0 ou 1 na transmissão escrevendo-se em TB8;
- 1 stop bit

Note que se SM2 = 1 e RB8 = 1, a interrupção da serial será atendida.

O algoritmo de comunicação consiste em:

- 1) No início, todos os escravos estão com SM2 = 1
- 2) Quando o mestre quiser enviar dados para algum escravo, ele escreverá 1 em seu bit TB8 e então enviará serialmente o endereço do escravo desejado, e como teremos todos os bits RB8 em 1, todos escravos serão interrompidos para verificar se é seu o endereço enviado.
- 3) O escravo selecionado zerará o seu bit SM2 e estará preparado para receber os dados, os quais terão agora o nono bit (RB8) em 0.
- 4) Os demais escravos permanecerão com SM2 em 1 e, dessa forma,

não serão mais interrompidos pois os dados têm RB8 = 0.

- 5) Em resumo, o mestre envia endereços com o nono bit em 1 e os dados com o nono bit em 0, portanto, se o mestre desejar se comunicar com outro escravo basta enviar o novo endereço com o nono bit em 1.

3.6.6 Comunicação serial entre o microcontrolador 8051 e o PC

Devido à diferença de tensão entre os níveis TTL e os usados no padrão RS-232C (PC), é necessário usar circuitos conversores para interfacear o microcontrolador com a porta serial do PC.

A conversão pode ser feita através de circuitos transistorizados discretos ou através de circuitos integrados como o 1488 (conversor de TTL para RS-232C) e o 1489 (conversor de RS-232C para TTL). A seguir estão os esquemas dos dois circuitos integrados muito usados para conversão de níveis de tensão TTL a níveis RS232 ($\pm 12V$) e o esquema básico de interligação entre o 8051 e a serial do PC.

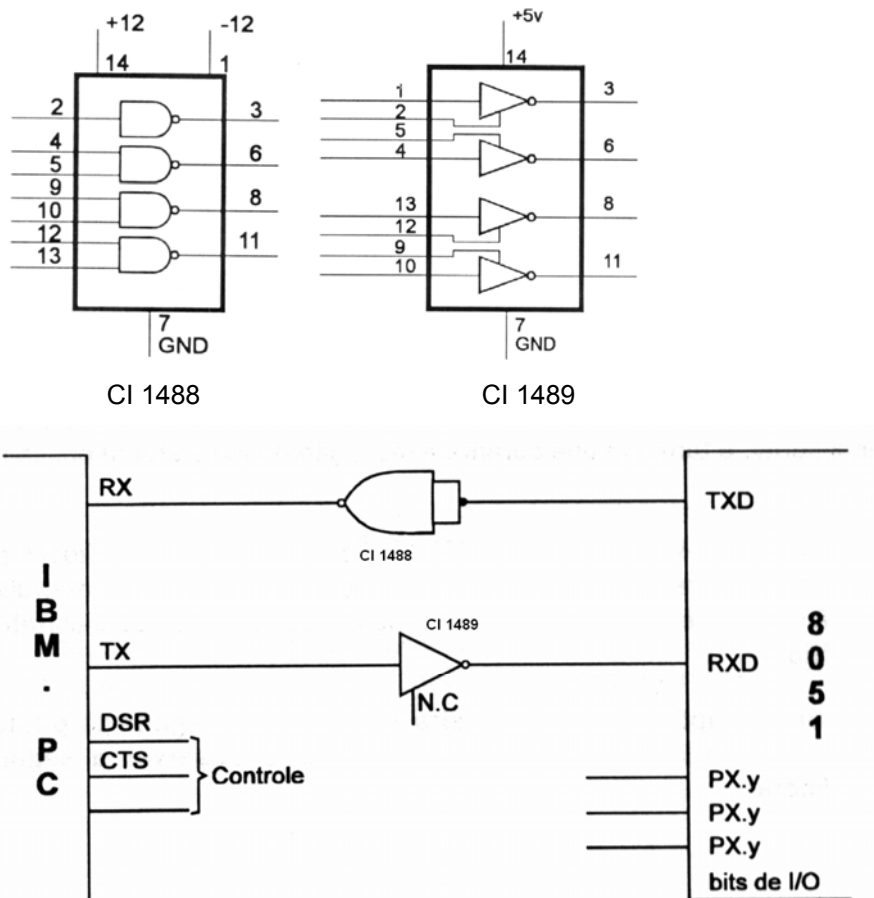


Figura 3.38 Esquema de interconexão do 8051 e uma PC

;Progr21 Introdução conceito de transmissão SERIAL

; Existem dois tipos de comunicação entre equipamentos: paralela
;
; e serial.
;
; A transmissão paralela , aquela em que cada bit de uma palavra
;
; , transportada em uma via de dados, ou seja, se trabalharmos
;
; com 8 bits , necessário 8 vias para comunicação, al,m de uma
;
; via para sinal de referencia. Este tipo de comunicação , vi vel
;
; para curtas distancias e em condições que necessário alta velo-
;
; cidade.
;
; A transmissão serial esta sendo aplicada em larga escala em to-
;
; do mundo. A transmissão serial consiste em efetuar uma comuni-
;
; cação entre dois, ou mais, equipamentos utilizando apenas uma
;
; via de dados, al,m da via de referencia. Um bom exemplo de
;
; aplicação deste tipo de comunicação , a INTERNET. Este tipo de
;
; comunicação minimiza muito o custo pois necessita de menos fios
;
; e podemos aproveitar a própria linha telefônica para efetuar este
;
; tipo de comunicação.
;
; No 8051 podemos trabalhar nos modos síncrono e assíncrono e tipo
;
; de comunicação *full duplex*.
;
; Existem apenas duas palavras para efetuarmos a interface serial:
;
; SBUF (byte por onde chegam ou são transmitidos os dados)
;
; SCON (byte de controle da interface serial)
;
; OBS: na verdade temos dois bytes para SBUF, um para transmissão
;
; e outro para recepção, mas para acessarmos utilizamos o mesmo
;
; endereço.
;
; PALAVRA SCON
;
; SM0=0 e SM1=0 Modo 0 de operação
;
; Modo síncrono e taxa de transmissão (freq. clock)/12
;
; Pino RXD usado para dados Pino TXD usado para clock
;
; SM0=0 e SM1=1 Modo 1 de operação
;
; Modo assíncrono e taxa de transmissão variável, controlado pelo
;
; bit SMOD da palavra PCON e pelo T/C1 (contador 1), obedecendo a
;
; seguinte equação: smod
;
; taxa = $\frac{2^x}{32}$ (taxa de overflow do T/C1)
;
; 32

```

; Este modo , composto por 01 bit start, 08bits de dados e 01 bit
; stop. Pino RXD usado para recepção e TXD para comunicação.
;
; SM0=1 e SM1=0 Modo 2 de operação
; Modo assíncrono e taxa de transmissão dada apenas pelo bit SMOD:
; SMOD=0 (freq.clok)/64 SMOD=1 (freq.clock)/32
; Este modo , composto por 01 bit start, 09 bits de dados e 01 bit
; stop. O nono bit , dado pelos bits TB8 para transmissão e RB8 para
; recepção. Pinos RXD e TXD idem modo 01.

; SM0=1 e SM1=1 Modo 3 de operação
; Modo de funcionamento , idêntico ao modo 2 e taxa de transmissão
; , idêntica ao modo 1.
; As flags TI e RI servem como interrupções caso alguma transmissão
; seja realizada.
;label instr operando ;comentários
;*****
;
; org 0 ;começa em 0
; Ser visto nos exemplos abaixo algumas rotinas para a utilização
; em programas que utilizam a transmissão de dados seriais.
; mov C,P ;mova o dado de paridade para carry
; mov TB8,C ;mova o dado de C pra o bit TB8
; mov SBUF,A ;mova o dado do acc para o SBUF
; Esta rotina faz com que o bit TB8 carregue o sinal de paridade usa-
; para conferência do dado transmitido.
WAIT: jnb ri,wait ;verifica se RI foi setado
; mov A,sbuf ;mova o dado de sbuf para acc.
; clr ri ;limpa RI
; Aqui termos uma rotina de verificação de recepção de mensagem.
WAIT2: jnb ti,wait2 ;verifica se ti foi setado
; clr ti ;limpa TI
; mov A,sbuf ;move dado a ser transmitido do acc para sbuf
; Agora a rotina apresentada , para transmissão de dados.
; mov A,pcon ;mova a palavra existente em pcon para acc

```

```

setb Acc.7    ;setar o bit 7 do acc
mov pcon,A    ;retornar dado para pcon. Repare que bit 7 do
              ;pcon , SMOD
;    Esta , uma rotina para manipulação do valor SMOD
mov scon,#52h ;move para scon 52h = (01010010)b. Isto deixar
              ;o port serial no modo 1, habilitar recepção, e
              ;setar bit TI.
mov TMOD,#20h ;move para TMOD 20h = (00100000)b. Isto habili-
              ;ta o timer 1 no modo 2.
mov TH1,#-13  ;programa T/C1
setb TR1      ;inicia contagem
Este , um exemplo para manipular a taxa de transmissão serial.
END

```

4 INTRODUÇÃO À SIMULAÇÃO COM PROTEUS (ANEXOS)

O Programa PROTEUS é uma aplicação CAD que se compõe de três módulos básicos:

- ISIS (*"Intelligent Schematic Input System"*): Módulo de captura de esquemas.
- VSM (*"Virtual System Modelling"*): Módulo de simulação, incluindo PROSPICE.
- ARES (*"Advanced Routing Modelling"*): Módulo para realização de circuitos impressos (PCB).

Nesta epígrafe se brinda uma guia básica para o emprego dos módulos ISIS e VSM, mostrando exemplos do procedimento a seguir para realizar de simulação de circuitos, especialmente digitais, que são os que abrangem esta disciplina.

O PROTEUS é uma ferramenta de CAD eletrônico. Suporta-se no PSPICE, tem uma biblioteca com mais de 6500 componentes analógicos y digitais, que é atualizada constantemente. Foi o primeiro programa simulador de microcontroladores com ambiente gráfico e até hoje, isto há despertado muito interesse atualmente. Suporta o trabalho com linguagens de baixo nível y de alto nível C (*Basic* o C). Permite a simulação interativa: pode-se, por exemplo, abrir e fechar interruptores durante a simulação, usar elementos hardware do computador (alto-falante, teclados, TRC, portas série e paralelo, etc). Ele possui diversos instrumentos virtuais (multímetros, geradores de sinal, osciloscópios, etc), e inclui modelos de diversos elementos reais: conversores analógicos digitais (CAD),

conversores digitais analógicos (CDA), displays reais, sensores de temperatura, motores, entre outros.

Por tanto, o PROTEUS permite idear projetos eletrônicos de uma forma muito rápida e serve de suporte prévio à construção de protótipos.

4.1 Passeio por o PROTEUS ISIS.

O objetivo deste epígrafe é introduzir rapidamente ao estudante em como criar um circuito simplex através do PROTEUS ISIS. Começar-se-á com a captura esquemática e conexão dos componentes, empregando um circuito singelo. Posteriormente se verá sua simulação, ou seja, como comprovar que seu funcionamento é correto. Assume-se que já se instalou o programa PROTEUS, e que o diretório atual é o predefinido normalmente pelo programa de instalação, ou seja: **C:\Arquivos de programas\Labcenter Electronics\Proteus 7 Professional**.

Para começar a trabalhar com o ISIS: Início → Programas → Proteus 7 Professional → ISIS 7 Professional. Então o PROTEUS ISIS se carregará e executará. Neste momento se mostrará a interfase gráfica principal (Figura 4.1). Digite por enquanto “**Não**”. Para nossa explicação ao **botão esquerdo** do mouse chamaremos **botão normal**, e ao **direito**, chamaremos **botão contrário**.

A área maior da tela (Figura 4.2) se chama a **Janela de Edição** (delimitada por um marco de cor azul), e atua como uma janela de desenho, aqui é onde se colocam e se interconectam os componentes. A área menor à esquerda do topo da tela se chama **Janela de vista global** ou **janela global** (“*Overview Window*”), a qual mostra uma visão global do esquema elétrico inteiro. O marco azul mostra o bordo da folha atual e o marco verde da janela global mostra a área da folha desdobrada atualmente na Janela de Edição. Entretanto, quando se toma um novo componente (com o *mouse*) da **janela de dispositivos** (DEVICES), localizada embaixo da janela global, este aparecerá na Janela global, e estará preparado para pô-lo na janela de edição.

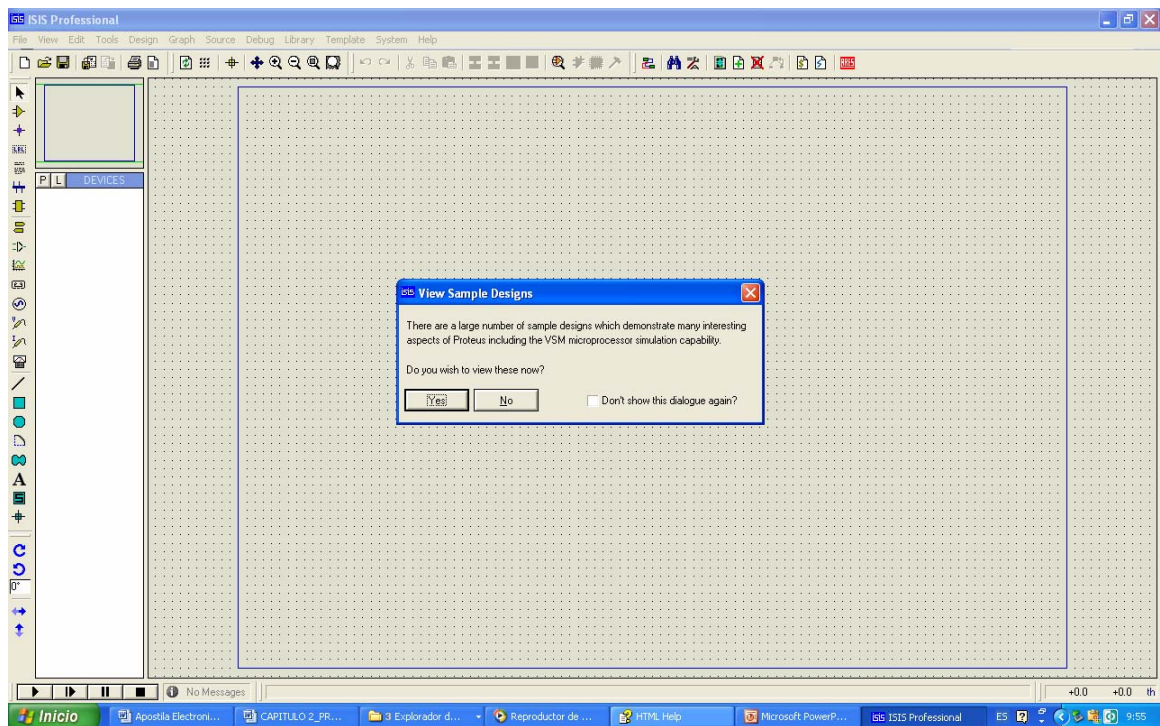


Figura 4.1 Interface gráfica inicial do PROTEUS ISIS

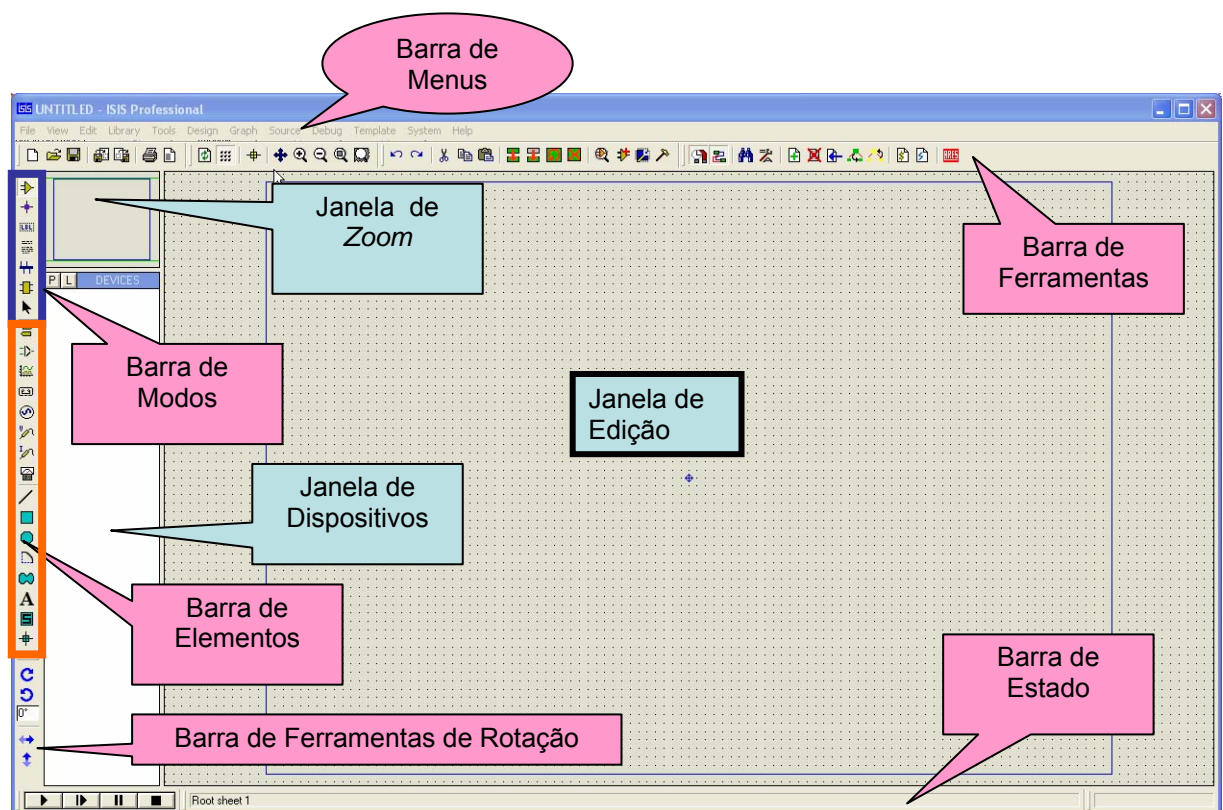







Figura 4.2 Janelas, barras e ferramentas de trabalho da Interface gráfica do PROTEUS ISIS.

Você pode ajustar a área do desenho desdobrada na Janela de Edição de diferentes maneiras:

- A Janela de Edição poderá mover-se para, abaixo, acima, esquerda ou direita com o indicador do *mouse* em cima da parte desejada da Janela de Edição e conjuntamente com isto se aperta à tecla F5 (equivale a *View* → F5).
- Mantendo pressionado o botão normal em cima da janela de *zoom*, pode passear de forma rápida por toda a janela de edição. Para sair deste modo deve pressionar o botão normal uma vez.

Para ajustar a escala do desenho na Janela de Edição se pode:

- Apontar com o mouse no lugar onde se deseja aumentar ou diminuir e logo oprimir a tecla F6 (, *Zoom In*) o F7 (, *Zoom Out*), respectivamente. Isto também se pode realizar com ajuda da roda do *mouse*: para diante (aproximar) e para trás (afastar).
- Quando se oprimir a tecla F8 () se desdobra todo o desenho.
- Oprimindo a tecla SHIFT e tirando um quadro ao redor de uma área determinada, esta será aproximada e centrada na Janela de Edição.
- Outra opção é usando os ícones da barra de tarefa: "*Zoom In*", "*Zoom Out*", "*Zoom All*" ou "*Zoom Area*". Os primeiros já se explicaram. O *zoom* a uma área permite mostrar uma área que o usuário selecione. Para isto pressione encima do ícone  é marque um área em forma de caixa para visualizar. Para sair deste modo, clique uma vez.

Este sistema possui na janela de edição um ralo de pontos o qual pode desdobrar-se de diferentes formas, (a) usando o comando "*Grid*", situado na barra de menu na opção "*View*", (b) pressionando a tecla 'G' e (c) pulsando o ícone do ralo na barra de ferramentas(). Esta ajuda a alinhar os componentes e facilita a ligação dos componentes.

4.1.1 Escolher, pôr e conectar os componentes.

Nesta epígrafe descreveremos os passos a seguir para realizar o desenho de um circuito simplex. O circuito objeto de estudo (Figura 4.3) é o seguinte: uma porta AND de 2 entradas, com 2 resistores (chamados resistores de *pull-up*) que garantem um nível de 1 lógico seguro quando os 2 interruptores (de tipo *push-button*) não estejam pressionados, devido a que estão alimentados com uma fonte de alimentação simbolizada pelo Terminal VCC, que está referenciado a terra (GND). O multímetro permite medir a tensão de saída.

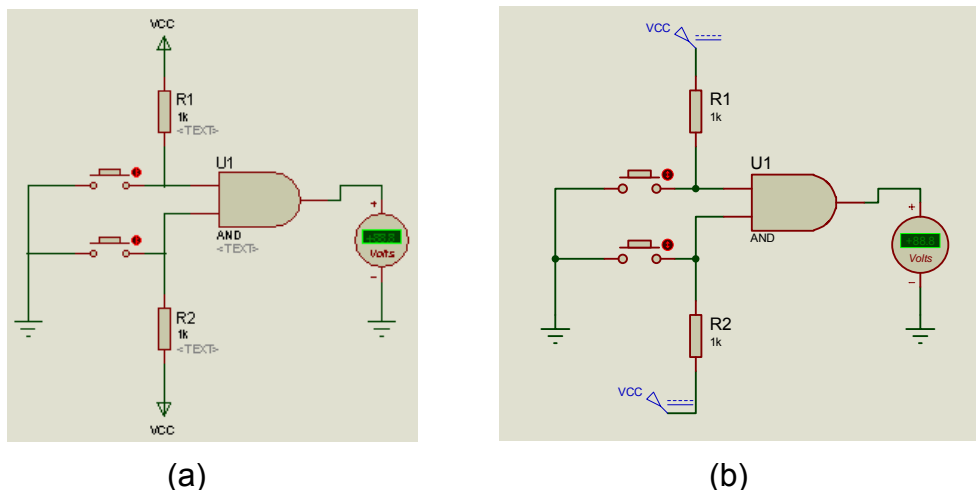





Figura 4.3 Circuito de exemplo. O circuito é alimentado (a) usando a fonte VCC do ícone  e (b) usando um gerador de tensão contínua no ícone  (ler texto na epígrafe 4.1.4).

Antes de começar é importante garantir que este selecionado o **Modo Componente** () da barra de Modo.

Primeiro se posicionará o indicador do mouse no botão “P” se localizado no extremo esquerdo do Seleccionador de Objeto (Figura 4.4) e logo se faz clique esquerdo obtendo como resultado a aparição da janela do **Seletor de Livraria e Dispositivos** (*Pick Devices*). Uma vez desdobrada esta janela você deverá digitar os identificadores (Exemplo: AND_2, Resistor...) dos diversos componentes ou elementos do circuito desejado.



Figura 4.4 Janelas, barras e ferramentas de trabalho da Interfase gráfica do PROTEUS ISIS

Uma vez desdobrada a janela “*Pick Devices*” a forma mais rápida de encontrar o componente é digitar seu nome. Senão conhece este, pode inclusive, digitar algumas palavras alegóricas ao mesmo, que com certeza, vai se mostrar algum resultado. Neste caso o identificador da porta AND de 2 entradas é **and_2** (também pudesse ser digitado o identificador **7408** que é um exemplo de porta AND da família TTL). Em qualquer caso, você deve selecionar, à componente desejado, e ter certeza de que este possui um modelo de simulação PSPICE disponível no programa PROTEUS (aparece na janela do extremo superior direito como “*Digital*

Primitive”). Por último, dando dobro clique na linha conveniente dos resultados mostrados, pode escolher o componente desejado: AND_2 (Figura 4.4) e permanecerá aberta para continuar digitando outros componentes. Na medida em que vão selecionando, os componentes devem ir aparecendo na janela de dispositivos.

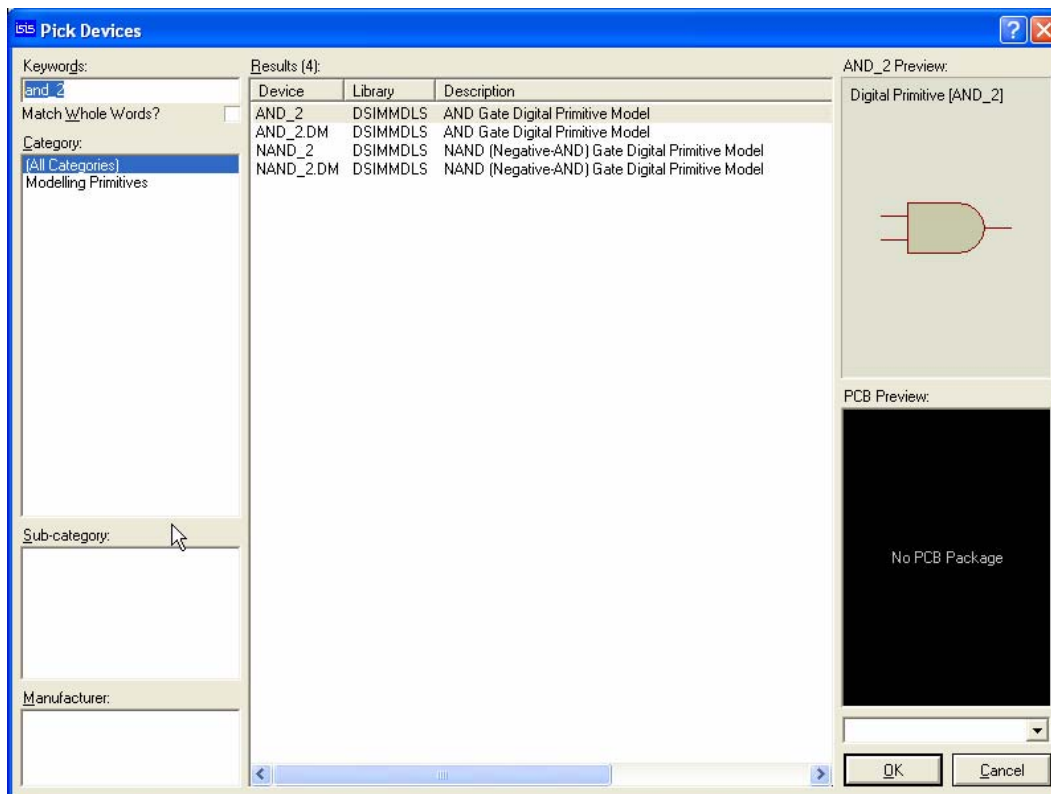


Figura 4.5 Seleção dos componentes digitando seu nome na parte superior esquerda donde fala *Keywords*, neste caso a seleção é o componente chamado AND_2 (porta AND de duas entradas).

4.1.2 Colocação de componentes no esquemático.

Uma vez selecionados os componentes, o próximo passo é colocá-los na área de desenho (Janela de Edição), (Figura 4.6). Faz-se clique no componente desejado ('AND') na janela do *Devices*, provocando que este apareça na Janela do *Zoom*. Logo se posiciona o cursor na posição onde se deseja colocar o componente na janela de edição (Figura 4.6) e finalmente se faz um clique. Sucessivos clicem em outras zonas de esta janela de edição, ocasionassem a repetição do componente selecionado.

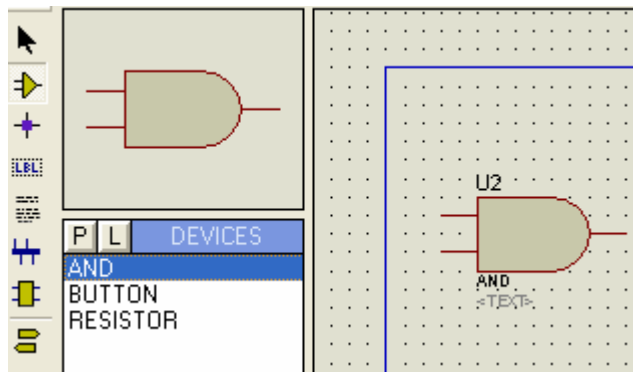


Figura 4.6 O objeto é selecionado na janela de DEVICES, aparece na janela de zoom e finalmente se faz um clique para colocar a ele na janela de edição (direita)

Para rotar o componente, colocar-se acima de ele e apertar botão “CONTRARIO” do mouse. Isto pode ser feito de outra forma, ou seja, antes de colocar o componente na janela de edição, se pode selecionar o mesmo, e usar nas barras de ferramentas de rotação.



Figura 4.7 Botões de rotação e espelho dos componentes

Repetir a mesma operação para outros componentes até conseguir a colocação de todos eles. Para arrastar o componente para outra posição da janela de edição, se clica uma vez (ele fica na cor vermelho) e logo, se clica mantendo pressionado até colocar o componente no lugar desejado.

Para editar qualquer valor do componente, se clica duas vezes acima do componente com o botão “NORMAL”.

Para apagar o componente, se clica duas vezes acima do componente com o botão “CONTRARIO”.

Quando se desejar de-selecionar o componente, ou seja, para que o componente não fique na cor vermelho, se clica fora de ele.


4.1.3 Interconexão ou Ligação dos componentes.



ISIS possui um sistema inteligente para interconectar componentes, chamado WAR (*Wire Auto Router*, em inglês). Para fazer um fio de conexão entre terminais de 2 componentes diferentes, se aponta com o mouse para um de estes terminais, automaticamente aparece uma caixinha vermelha acima, indicando que existe a

possibilidade, se clica o botão NORMAL (não precisa manter pressionado) e se faz o mesmo no terminal do segundo componente. O ISIS procura um jeito de fazer o fio, em caso contrário, desloque de posição a um dos componentes.

Se logo depois de desenhar os fios, move-se um componente, o WAR volta para conectar o fio de acordo ao movimento efetuado.

Para mover o cabo manualmente, pressiona-se clique CONTRÁRIO para selecioná-lo, e apertando (e mantendo apertado) o clique NORMAL se arrasta até a nova posição onde se deseja colocá-lo. Para conectar o fio manualmente, simplesmente se aperta clique NORMAL no terminal, e logo clique NORMAL cada vez que se alcance uma esquina ou um ponto de inflexão na trajetória do cabo.

Para colocar o voltímetro de contínua, selecionar na barra de elementos o ícone de instrumentos virtuais () e logo pegar ao *DC VOLTMETER*.

Para completar o esquema eléctrico, se necesitan colocar y conectar la fuente de alimentación y la tierra. Para esto, seleccionar en la barra de modos, el icono Terminal (). Seleccione y coloque la tierra (GND) y la fuente (POWER). Em cima do Terminal POWER, faça clique duas vezes e digite (ou selecione) a string VCC. Outra forma de colocar a fonte de alimentação é usando um gerador de tensão contínua (ícone ) que está barra de elementos (Figura 4.8).

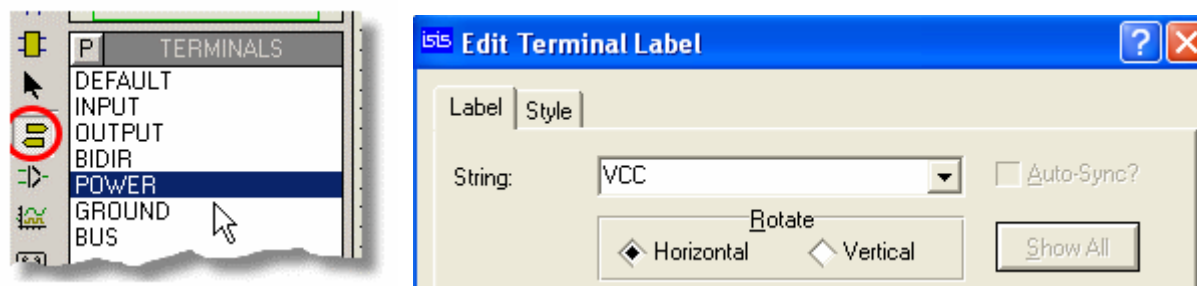


Figura 4.8 Uso do Ícone de terminais (*Terminals Mode*) para colocar a terra (GND) e a fonte de alimentação (POWER)

Para operar sobre uma área do circuito (apagar, mover, apagar, rotar), mantendo pressionado o botão NORMAL encerrar a área desejada. Depois fazer clique no botão CONTRARIO (o use os ícones de bloco na barra de ferramentas) e selecione a operação desejada.

4.1.4 Simulação do circuito e salvar o circuito

Simular significa comprovar de forma virtual o funcionamento do circuito. Em este caso, interessa medir a indicação do voltímetro de saída para cada uma das combinações dos níveis lógicos à entrada. Portanto, ativando ou não aos interruptores de entrada, a tensão de saída deve trocar. Como a porta usada é ideal, então as tensões de saída correspondentes aos níveis lógicos serão ideais (0 V para 0 lógico e $V_{CC} = 5\text{ V}$ para 1 lógico).

Para iniciar a simulação e manter esta de forma contínua, pressionar o botão PLAY (▶) situado na parte inferior da tela. Neste modo, o efeito de cada mudança realizada em algum dos dois interruptores à entrada, é observado imediatamente no voltímetro de saída. Para realizar a simulação passo a passo, pressionar o botão STEP (⏮). Neste caso, o efeito de pressionar algum interruptor será observado depois de pressionar novamente ao botão STEP. “O botão STEP atua como uma câmara fotográfica, ou seja, cada vez que se pressiona o mesmo, atira-se” uma foto ao estado elétrico do circuito. Na figura 4.9 se observam duas situações da simulação. Os níveis lógicos se representam em cada um dos terminais através das cores (vermelho para 1 lógico, azul para 0 lógico, cinza para estado indeterminável). Para deter a simulação, pressionar o botão STOP (■) situado na parte inferior da tela.

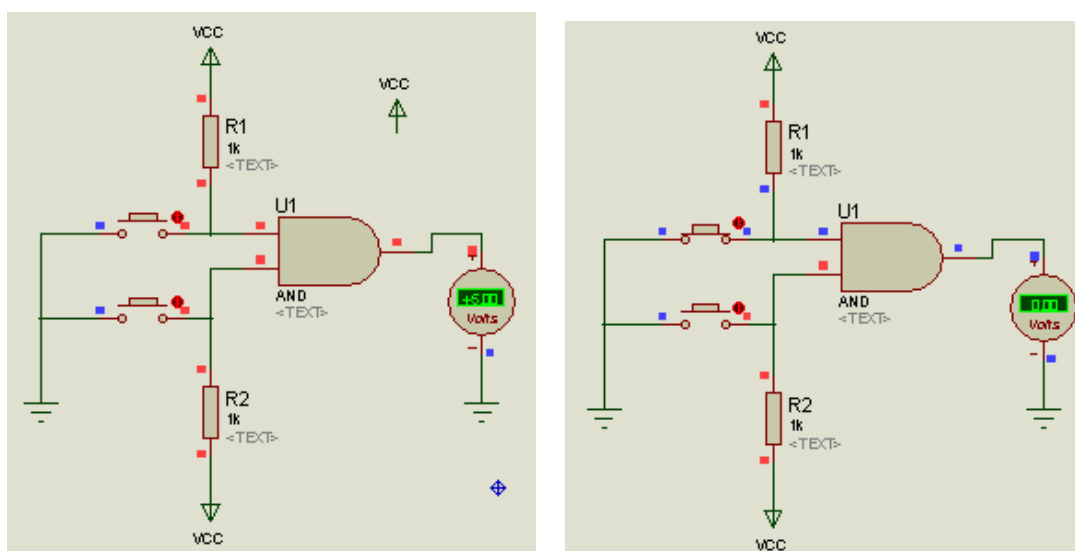



Figura 4.9 Simulação do circuito: (a) os *push-button* estão abertos, (b) um *push-button* é fechado.

Para salvar o circuito em um arquivo, usar as opções SAVE DESIGN ou SAVE DESIGN AS do mesmo jeito que usa qualquer aplicativo de Windows.

4.1.5 Instrumentos virtuais mais usados do PROTEUS ISIS e VSM

O PROTEUS tem vários instrumentos virtuais que oferecem um alto nível de interatividade, com o uso destes é possível fazer um processo de depuração mais rigoroso e obter a posta a ponto do circuito.

Geradores de sinais.

Existem dois tipos de elementos para gerar sinais: (a) o gerador cujo ícone é idêntico o seu símbolo elétrico () e (b) um instrumento virtual cujo aspecto é mais parecido ao instrumento real. O primeiro está na barra de elementos do ISIS, só tem um terminal para ligação (o outro que não aparece fica ligado a GND) e pode gerar vários tipos de sinais (DC ou contínua, senoidal, pulsos, exponencial, etc) como se observa na figura 4.10a. Um tipo de sinal muito interessante é o sinal de áudio que pode estar armazenada em um arquivo com extensão “wave”. Esta permite que um circuito específico possa ser excitado com um sinal real adquirido e armazenado em um arquivo. Neste curso isto não é de muito interesse porque a natureza dos sinais são digitais, por isso, nosso interesse principal é sinal de tipo pulso. Na figura 4.10b se mostra uma modificação de nosso circuito original, agora uma entrada da porta AND está excitada com um gerador de pulsos. Na figura 4.10c se mostram os parâmetros definidos para este tipo de sinal. Cada tipo de sinal tem seus próprios parâmetros dependendo de sua morfologia.

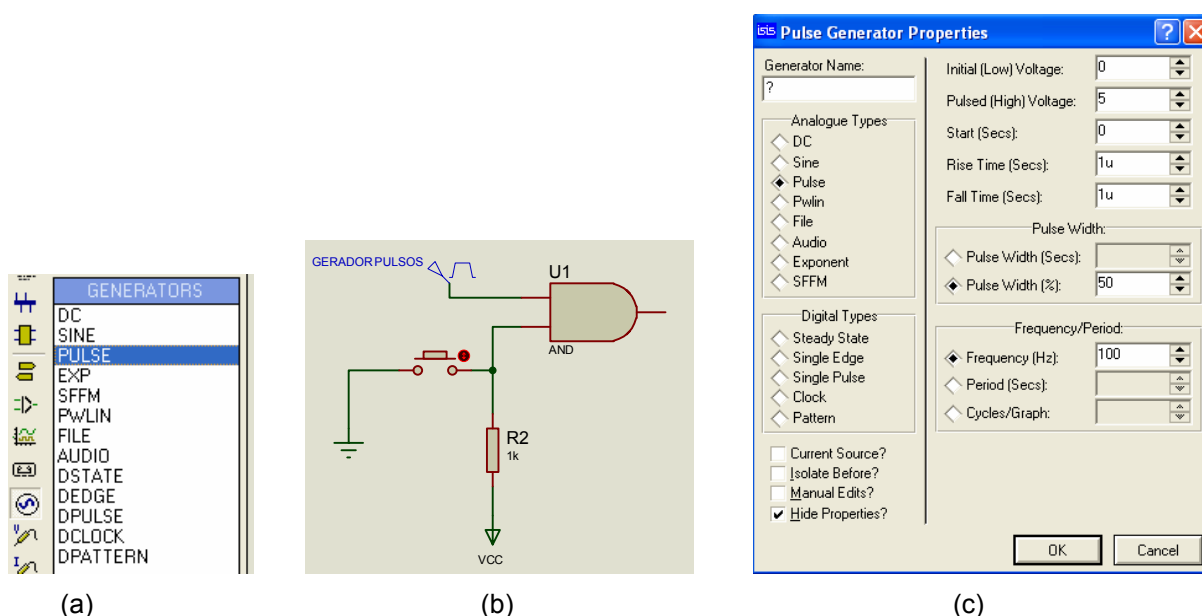
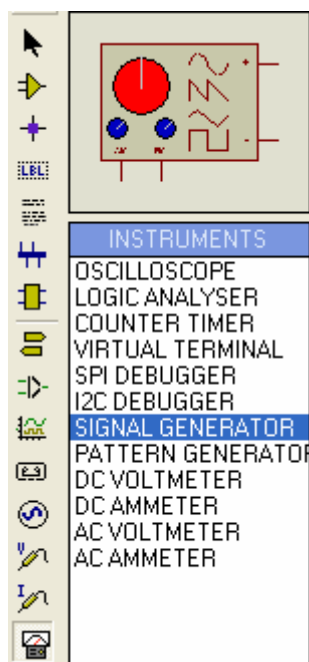


Figura 4.10 Gerador de pulso. (a) Seleção (b) Colocação em um circuito (c) Edição de seus parâmetros: 0 V para nível BAIXO, 5 V para nível ALTO, frequência igual a 100 Hz, com ciclo de trabalho (duty cycle) igual a 50 % (tempos iguais em níveis lógicos zero e um). Não existe atraso a partir do início da simulação.

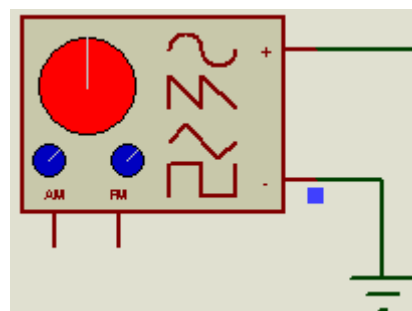
O segundo gerador pode ser obtido dentro dos instrumentos virtuais (figura 4.11a), também permite gerar os sinais típicos (figura 4.11c) que saem pelos terminais + e – (Figura 4.11b).

Para seleccionar o tipo do sinal desejado, pulsar consecutivamente o botão rotulado como *Waveform*. Para seleccionar a polaridade do sinal de saída, pulsar o botão rotulado como *Polarity* para Unipolar (sinais com faixa de tensão desde 0 V até o valor de amplitude seleccionado) ou Bipolar (sinais simétricos respeito à tensão de 0 V, ou seja, sinais com polaridade negativa e positiva). As portas lógicas normalmente operam com sinais unipolares, entre 0 V e 5 V.

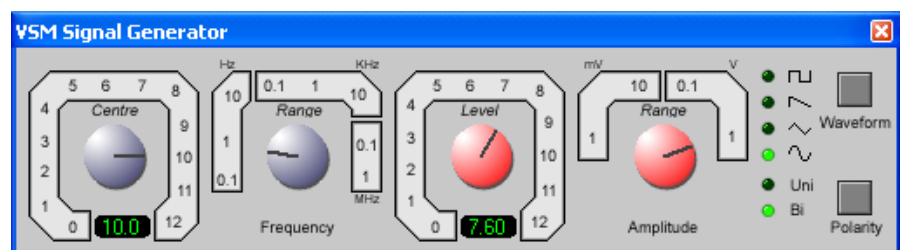
Para seleccionar a frequência (Figura 4.11c) do sinal, usar os 2 controles da esquerda (Controles de frequência) e para seleccionar a amplitude de saída usar os 2 controles da direita (Controles de Amplitude). A filosofia de estes controles é a seguinte: um dos dois controles associados a cada grandeza define a faixa (*Range*) da grandeza (Exemplo: Controle *Frequency* para a Frequência e controle *Amplitud* para a Amplitude) e o outro define o valor específico dentro de cada faixa. Os valores finais seleccionados para cada grandeza aparecem nos indicadores da cor verde que estão embaixo dos controles rotulados como *Centre* (Valor da frequência) e *Level* (Valor da amplitude).



(a)





(b)



(c)

Figura 4.11 Instrumento gerador de sinais. (a) Seleção entre todos os instrumentos virtuais
(b) Observar que o gerador tem duas saídas (c) Controles do gerador.

Por exemplo, para gerar um sinal de pulsos unipolar de amplitude igual a 5 V e frequência igual a 50 Hz, fazer as seguintes operações:

- **Ajuste de forma de onda:** Pressionar o botão *Waveform* até que fique aceso o indicador na posição superior () indicando o sinal de saída vai ser de tipo pulso ou onda quadrada.
- **Ajuste de polaridade:** Pressionar o botão *Polarity* até que fique aceso o indicador na posição rotulada como *Uni* () indicando o sinal de saída vai ser de tipo Unipolar. Se conectar o terminal – do gerador a GND então será positiva, em caso contrario será negativa.
- **Ajuste de amplitude:** Mover o controle *Amplitude-Range* apontando à escala de 1 V (Faixa com valores possíveis desde 0 até 12 V). Mover o controle *Level* apontando ao valor da escala igual a 5. Observar no indicador verde associado que o valor indicado seja de 5.00 (ver figura 4.12)

Ajuste de frequência: Mover o controle *Frequency-Range* apontando à escala de 10 Hz (Faixa com valores possíveis desde 0 Hz até 120 Hz). Mover o controle *Centre* apontando ao valor da escala igual a 5. Observar no indicador verde associado que o valor indicado seja de 50.0 (ver figura 4.12)

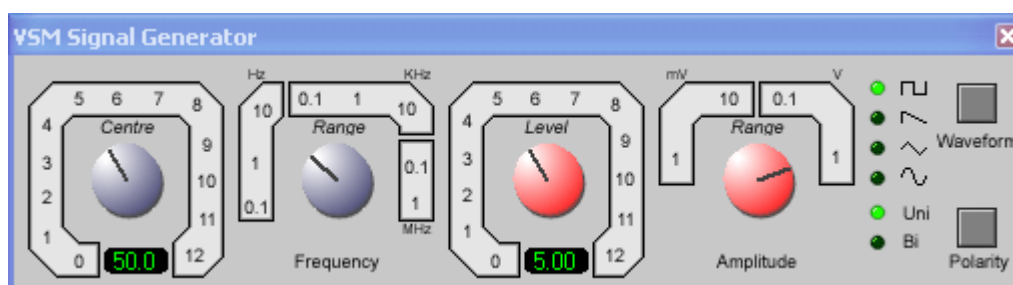



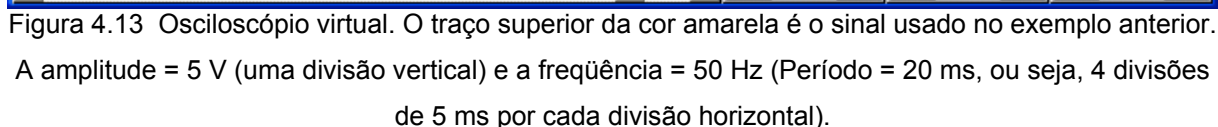
Figura 4.12 Exemplo que mostra o ajuste de um gerador de pulsos virtual com valores muito usados nos circuitos digitais.

Osciloscópio Virtual.

Funciona de maneira similar a um osciloscópio real. As características gerais de este osciloscópio são (ver figura 4.13)


- Quatro canais de entrada verticais, rotulados com as letras A, B, C e D com sensibilidade desde 2 mV/div até 20V/div. Cada traço usa a mesma cor rotulado pelo canal.
- Base de tempo desde 0,5 μ s/div até 200 ms/div. Em este caso, o seletor da Fonte de sinal de varredura horizontal (rotulado como *Source*) deve estar na posição da extrema esquerda indicando o sinal de denta de derra ().

Possibilidade de ocultar qualquer canal na tela (posição *OFF* de cada canal), ajustar a referencia de tensão ou 0V (posição *GND* de cada canal), fazer medidas de tensões de sinais AC sem ter em conta seu nível de tensão contínua (posição *AC* de cada canal), e fazer medidas de tensões de sinais de AC () considerando seu nível de tensão contínua.



Quando quer desenhar um circuito lógico com muitos fios de conexão, é conveniente usar os barramentos. Para fazer um barramento, e tomando o exemplo da figura 4.14 sequência de passos é a seguinte.

- # Sistemas Microprocessados

2. Marcar o começo do barramento fazendo um clique na janela de edição. Riscar o barramentos de forma similar a como se faz um fio de conexão. Cada novo clique permite trocar em 90 graus, a orientação do barramento.
3. Para acabar o barramento, fazer dobro clique.
4. Fazer os fios de conexão da saída de cada uma das portas até o barramento e das entradas do osciloscópio ao ônibus. Note que cada vez que o fio de conexão toque o barramento, aparece uma linha de traços no interior do barramento, indicando conexão.
5. Selecionar o modo de etiquetas (*label*, em inglês) rotulado como na barra de modos (). As etiquetas são um símbolo que permite conectar 2 ou mais fios que converjam no barramento. Isto se realiza lhes atribuindo a estes fios uma mesma etiqueta.

Posicionar-se em cima cada fio, quando este se ilumine ligeiramente em cor vermelha, fazer clique, e digitar o nome do símbolo que representará à etiqueta. Recomenda-se colocar um nome relacionado com a função deste fio. No circuito da figura 4.14 as etiquetas são SAL1 e SAL2. De esta forma, a saída da porta superior (SAL1) visualiza-se no canal D do osciloscópio, e a saída da porta inferior (SAL2) visualiza-se no canal A do osciloscópio.

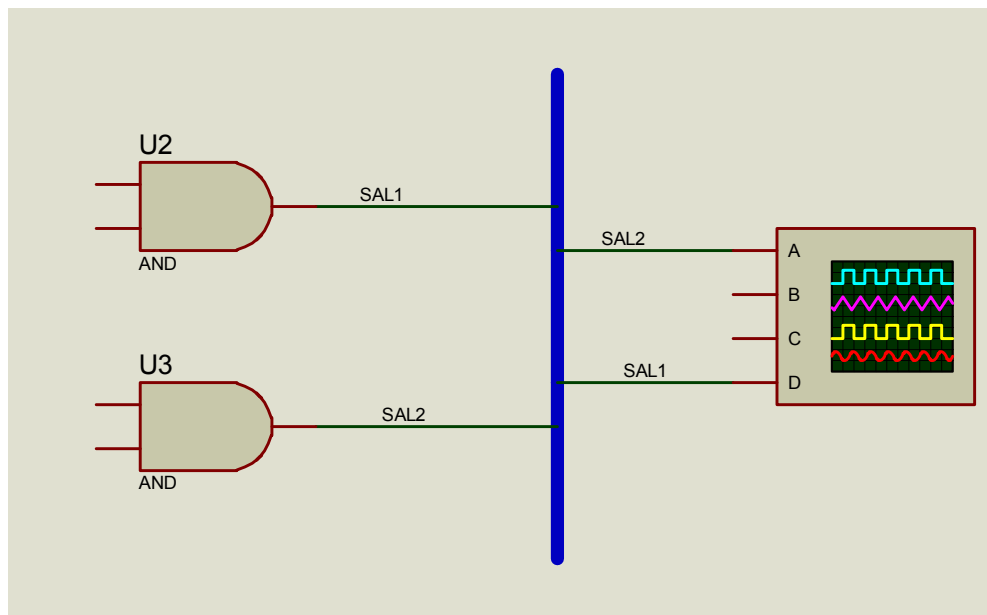


Figura 4.14 Exemplo de uso dos barramentos para conectar as saídas das portas às entradas do osciloscópio.

4.1.7 Elementos da biblioteca úteis para a disciplina

A seguir alguns nomes de elementos da biblioteca de PROTEUS que serão muito usados na presente disciplina: Microprocessador (80C51), Displays de 7 segmentos com codificador BCD (7SEG_BCD) ou sem decodificador (7 SEG-COM-ANODE, 7 SEG-COM-CATHODE), display de cristal líquido (LM016L), teclados matriciais (KEYPAD-PHONE, KEYPAD-SMALLCALC), botões ou teclas isoladas (BUTTON), vários LEDs animados de diversas cores (LED-RED, LED-GREEN, LED-BLUE), Reles e interruptores (RELAY, RLY-SPNO, SW-SPDT-MOM, SWITCH), Bocinha (BUZZER), Abajur (LAMP), Geradores de níveis lógicos (LOGICSTATE, LOGICTOGGLE), prova medidora de níveis lógicos (LOGICPROBE), Motores (MOTOR, MOTOR-DC), Comparador binário (COMPARATOR_4), além dos CI das famílias...(74LS138, 74LS04...), Conversores Analógico-Digitais (ADC0801, ADC0808), Conversor Digital Analógico (DAC0808), transistores NPN (BC546BP, BD135...) e PNP (BC557, BD136...), Memórias EPROM (2732, 2764, 27128...), RAM (6116, 6264, 62256),

Recomenda-se ao estudante que realize uma pesquisa no PROTEUS, de dispositivos e elementos para seu trabalho futuro nesta disciplina.

5 LABORATORIOS

5.1 LABORATORIO 1: KIT DE DESENVOLVIMENTO E MANUSEIO DE LEDS

1. Título: Familiarização com o sistema de desenvolvimento (KIT DE TREINAMENTO) com o microprocessador AT89S8252.

2. OBJETIVOS

Familiarização com o kit de treinamento (ver fotografia em apartado de ANEXOS) a usar na disciplina do ponto de vista de hardware e software.

3. INTRODUÇÃO TEÓRICA

Para desenvolver um projeto com microcontroladores, usualmente se necessitam ferramentas de hardware e software. Dentro das ferramentas de hardware as mais usuais são as seguintes: Computador pessoal (PC), Sistemas de desenvolvimento com o microprocessador a ser usado (também chamados o kits de treinamento), dispositivo programador de microcontroladores, Placa de provas (breadboard) ou algo similar para montagem de circuitos de interface, e fonte de

alimentação. Atualmente muitos microcontroladores têm memória de programa de tipo flash, e não precisam que sejam tirados do circuito final para ser gravados. Este é o caso de nosso microprocessador ATMEL 89S8252.

Dentro das ferramentas de software as mais usuais são as seguintes: Editor de texto, Compilador, Software do dispositivo programador, e simulador. Este último é uma ferramenta que é executada no PC e permite ao programador testar o funcionamento dum programa no PC sem ter que construir nenhum hardware com o microcontrolador. Em nosso caso usaremos os simuladores PROTEUS ISIS/VSM e SDCC. Apesar de que os simuladores são ferramentas muito úteis, apresentam a desvantagem de que o programa não se executa em tempo real e por isso é muito importante testar os programas em sistemas chamados ICD (In Circuit Debugger). Estes sistemas são uma variante dos kits de treinamento que estão conectados a um PC via serial ou paralela, e têm possibilidades de executar programas passo a passo e verificação de registradores, variáveis, e áreas de memória em tempo real. Também permitem a programação sem necessidade de extrair o microcontrolador do circuito de trabalho (In-circuit programming ou In-System Programming system). Os sistemas ICD são relativamente baratos em comparação com outros chamados ICE (In Circuit Emulators) nos quais o microcontrolador do sistema baixo prova é extraído de sua base e é substituído por um conector cujo outro extremo se conecta à caixa do ICE ligado ao PC. O ICE emula em tempo real ao microcontrolador tirado, comportando-se como se este estivesse na base. Embora estes sistemas são caros e por isso são mais usados os sistemas ICD como ocorre em nosso caso.

2.1 Circuito do sistema de desenvolvimento e metodologia geral de trabalho.

O circuito elétrico a usar nos laboratórios de sistemas digitais se mostra na bibliografia 2. A metodologia de trabalho será a seguinte: usar o sistema PROTEUS ISIS e o programa SDCC para simular e depurar os erros dos programas elaborados, e uma vez que estejam pronto, carregar e compilar o programa fonte (*ASM ou *C) com o programa ide8051 (ver próximo apartado).

2.2 Software de controle do sistema de desenvolvimento

O software não precisa ser instalado (ele funciona sobre o sistema operacional DOS) por isso só deverá ser copiado. É muito importante que o estudante não troque os nomes das pastas de trabalho, porque isto provocará erros na execução do software. Isto é devido a que os programas *.BAT (execução por lotes) foram escritos tendo em conta os nomes atuais das pastas. A seguir, se

descrevem os procedimentos que deverão realizar-se para executar programas em assembler e em linguagem C.

- Execução de programas em linguagem assembler
 - a. Fazer dobre click no arquivo C:\SDCC\ASM51\ide8051
 - b. Abrir o arquivo fonte com extensão ASM que deveria estar nesta.
 - c. No menu “Projeto” selecione a opção desejada (a mais usada é Assembla e Carrega)
- Execução de programas em linguagem C
 - a) Fazer dobre click no arquivo C:\SDCC\Exemplo\ide8051
 - b) Abrir o arquivo fonte com extensão ASM
 - c) No menu “Projeto” selecione a opção desejada (a mais usada é Assembla e Carrega)

4. PREPARAÇÃO PREVIA

- Estudar o esquema elétrico do sistema de desenvolvimento com 8051 (ver Anexos de esta Guia)
- Estudar os passos para executar um programa em linguagem assembler usando alguns dos programas explicados: ASM51 embarcado em PROTEUS ISIS (Metalink) ou SDCC.
- Levar ao laboratório um resumo dos seguintes elementos: instruções em assembler e diretivas do assembler (ORG, EQU, DB, DW, DS, SET, etc).

3.1 NOTA SOBRE AVALIAÇÃO DE CADA LABORATORIO

Será realizada uma avaliação de entrada que definirá o 30 % da nota final do laboratório segundo a seguinte formula:

$$NF = (0,3 AE + 0,5 DP + 0,2 RE)/3 \quad \text{onde,}$$

AE é a Avaliação de entrada (ver apartado 4), DP é o Desenvolvimento Prático e RE é a nota do relatório.

5. AVALIAÇÃO DE ENTRADA AO LABORATORIO

Entregar à entrada do laboratório em formato eletrônico (ou enviar por correio eletrônico antes de entrar em laboratório) os seguintes arquivos:

- Circuito elétrico (simplificado) em PROTEUS ISIS da pratica de laboratório (*.DSN)
- Programas em linguagem assembler (*.ASM e *.HEX correspondente) e C (*.C e *.HEX correspondente) da pratica de laboratório.
- Se o professor estima conveniente, realizará uma pergunta escrita à entrada do laboratório relativa aos conteúdos teóricos de base deste laboratório o aos procedimentos experimentais. Neste caso, a nota da avaliação de entrada é a media desta e dos arquivos anteriores.

6. MATERIAL UTILIZADO

- Kit de treinamento e acessórios (fonte de alimentação e cabos de conexão ao computador).
- Computador pessoal e software assembler

7. PROCEDIMENTOS EXPERIMENTAIS

6.1 Familiarização com o sistema de desenvolvimento

Observações importantes:

- a) O conector J20 é empregado para descarregar (usando o protocolo SPI, ver pág. 25 da bibliografia 1) o programa a executar no kit. Os jumpers devem ser colocados somente durante o processo de descarga do programa. Posteriormente devem ser tirados para não afeitar o funcionamento das linhas P1_5, P1_6 e P1_7.
- b) O conector J8 é empregado para RESET/PROGRAMAÇÃO. Na função de programação, os terminais 2 e 3 de J8 devem estar conectados. Na função do RESET manual (com o botão RESET1) os terminais 1 e 2 de J8 devem estar conectados (ver SILKSCREEN do kit). Uma vez que o programa tenha sido descarregado para o kit, pode programar-se na função do RESET manual.

6.1.1 Analisar o esquema elétrico do kit de treinamento da maquete do desenvolvimento com o computador pessoal. Começar observando as

conexões diretas ao microprocessador, detalhando os nomes dos sinais e dos conectores associados a estes sinais. Responder as seguintes perguntas:

- a) Poderia ser expandida conectada alguma memória de programa externa a este sistema? Justificar.
- b) Se o cristal de quartzo usado é de 12 MHz, Quanto dura um ciclo de máquina neste sistema?
- c) Segundo a observação importante a) anterior, detalhe no relatório a explicação e as cartas de tempo do processo de descarrega do programa no microprocessador ATMEL 89S8252.
- d) Que conexão deve ser feita para garantir a piscada dos LEDs D5 a D12 tirando o padrão de bits por a porta P1 do microprocessador (conector J6)? Qual é o nível lógico que garante o aceso dos LEDs ?
- d) Que conexão deve ser feita para garantir a saída dos dados desde a porta P1 do microprocessador até o display LCD?

6.2 Realização do programa para piscada dos LEDs.

6.2.1 Realizar um programa (laço infinito) nos linguagens assembler e C que provoque a piscada dos LEDs D5 a D7 da seguinte forma:

Equipe	Tarefa
1	Deslocamento de LED aceso de D5 até D12 de um em um, ou seja, em cada instante de tempo, apenas existe um LED aceso.
2	Deslocamento de LED aceso de D12 até D5 de um em um, ou seja, em cada instante de tempo, apenas existe um LED aceso.
3	Deslocamento de LED aceso de D5 até D12 ficando aceso o LED anteriormente aceso, ou seja, em cada instante de tempo, vai incrementando o número de LEDs acesos. Depois de acender-se o LED D12 repete-se o laço.
4	Deslocamento de LED aceso de D12 até D12 ficando aceso o LED anteriormente aceso, ou seja, em cada instante de tempo, vai incrementando o número de LEDs acesos. Depois de acender-se o LED D5 repete-se o laço.
5	Deslocamento de LED aceso de D5 até D12 ficando aceso o LED só os dois últimos LEDs anteriormente acesos, ou seja, em cada instante de tempo, apenas ficam acesos 2 LEDs. Depois de acender-se o LED D12 repete-se o laço.

6	Deslocamento de LED aceso de D12 até D5 ficando aceso o LED só os dois últimos LEDs anteriormente acesos, ou seja, em cada instante de tempo, apenas ficam acesos 2 LEDs. Depois de acender-se o LED D5 repete-se o laço.
7	Deslocamento de LED aceso de D5 até D12 ficando aceso o LED só os três últimos LEDs anteriormente acesos, ou seja, em cada instante de tempo, apenas ficam acesos 3 LEDs. Depois de acender-se o LED D12 repete-se o laço.
8	Deslocamento de LED aceso de D12 até D5 ficando aceso o LED só os três últimos LEDs anteriormente acesos, ou seja, em cada instante de tempo, apenas ficam acesos 3 LEDs. Depois de acender-se o LED D5 repete-se o laço.

NOTA: O intervalo de tempo em e entre cada operação é de 1 segundo.

6.3 Interpretação e comparação dos códigos obtidos.

6.3.1 Realizar um programa (laço infinito) nos linguagens assembler e C que provoque a piscada dos LEDs D5 a D7 da seguinte forma:

7. RELATORIO

A estrutura do relatório é a seguinte:

- Portada com dados gerais (Disciplina, Título da pratica, nomes e equipe dos estudantes)
- Resumo da pratica (máximo dois parágrafos)
- Objetivos
- Esquema elétrico simplificado desta pratica elaborado com o programa PROTEUS ISIS mostrando de alguma forma (linhas descontínuas, cartéis de texto, etc.) os diferentes conectores empregados do sistema de desenvolvimento (kit). Calcular os valores das correntes e tensões associados aos circuitos de manuseio dos LEDs (Procurar parâmetros dos dispositivos em internet, por exemplo, hFE transistores).
- Programas em assembler e em linguagem C com seus correspondentes comentários. Se aparecerem demoras, explicar seu cálculo detalhado.
- Resultados obtidos no laboratório e sua discussão (explicação dos mesmos).
- Conclusões

8. BIBLIOGRAFIA DO LABORATORIO

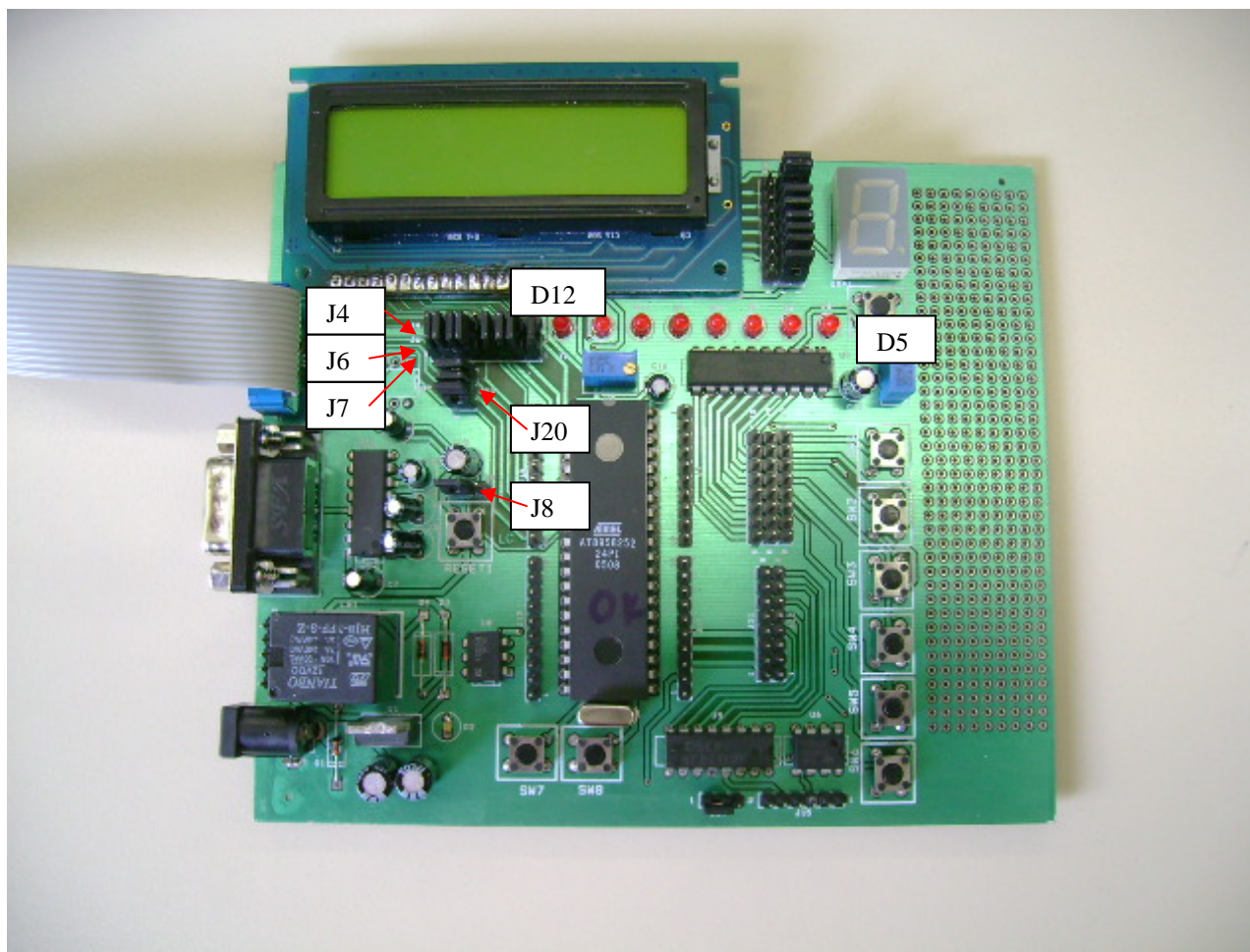
1. Datasheet do Microcontrolador ATMEL 89S8252.
2. Circuitos elétricos do kit de treinamento (ver correio eletrônico e ANEXOS)

9. PREGUNTAS DE REPASO

Estudar o processo de descarrega dos programas usando o protocolo SPI. Explicar funcionamento e mostrar cartas de tempo (ver datasheet do AT89S8252, pág. 25).

10. ANEXOS

10.1 Fotografia do sistema de desenvolvimento



10.2 Silk screen do lado de componentes

deste display (a até g mais o ponto decimal ou dp) estão conectados ao conector J16. O sistema de desenvolvimento está concebido para usar a porta P1 para excitar ao display. Para isto se deve unir cada uma das linhas do conector J13 com as correspondentes do conector J16 y as do conector J7 (bases dos 8 transistores) com as de conector J6. O estudante deve analisar detalhadamente o circuito de excitação composto pelos 8 transistores e tirar conclusões a respeito dos níveis lógicos adequados para acender cada um dos segmentos assim como às correntes que circulam por os segmentos do display quando estejam ligados.

3. PREPARAÇÃO PREVIA

- Estudar detalhadamente na área de recursos visuais do esquema elétrico do sistema de desenvolvimento com 8051 (ver Anexos de esta Guia). Procurar informação em internet do display C-551E (Tensões, correntes, longitude de onda pico, cores).
- Estudar como realizar a conexão dos switches SW1 a SW8 (conector J2) à porta P0 (conector J3).
- Estudar as instruções da linguagem C e assembler e a forma de realizar o interfaceamento entre ambos.

3.1 NOTA SOBRE AVALIAÇÃO DE CADA LABORATORIO

Será realizada uma avaliação de entrada que definirá o 30 % da nota final do laboratório segundo a seguinte formula:

$$NF = (0,3 AE + 0,5 DP + 0,2 RE)/3 \quad \text{onde,}$$

AE é a Avaliação de entrada (ver apartado 4), DP é o Desenvolvimento Prático e RE é a nota do relatório.

4. AVALIAÇÃO DE ENTRADA AO LABORATORIO

Problema de entrada:

Deseja-se implementar um contador reversível de 0 até 9 que seja dirigido por 2 teclas: uma para contar ascendente e outra para contar descendente. Se se pressionassem ambas as teclas, o contador se detém. A contagem deve mostrar-se no display de 7 segmentos. O intervalo de tempo entre cada operação é de 1 segundo.

Realizar:

a) O programa em linguagem assembler

b) O programa em linguagem C

Equipe	Teclas a usar/Forma de contagem
1	SW1/Ascendente, SW2/Descendente
2	SW3/Ascendente, SW4/Descendente
3	SW5/Ascendente, SW6/Descendente
4	SW2/Ascendente, SW3/Descendente
5	SW4/Ascendente, SW5/Descendente
6	SW1/Descendente, SW2/Ascendente
7	SW3/Descendente, SW4/Ascendente
8	SW5/Descendente, SW6/Ascendente

OBS: A equipe corresponde ao número de seu kit de trabalho no laboratório 1.

Entregar à entrada do laboratório em formato eletrônico (ou enviar por correio eletrônico antes de entrar em laboratório) os seguintes arquivos:

- Circuito elétrico (SIMPLIFICADO) em PROTEUS ISIS da pratica de laboratório (*.DSN)
- Programas em linguagem assembler (*.ASM e *.HEX correspondente) e C (*.C e *.HEX correspondente) da pratica de laboratório.
- Se o professor estima conveniente, realizará uma pergunta escrita à entrada do laboratório relativa aos conteúdos teóricos de base deste laboratório o aos procedimentos experimentais. Neste caso, a nota da avaliação de entrada é a media desta e dos arquivos anteriores.

5. MATERIAL UTILIZADO

- Kit de treinamento e acessórios (fonte de alimentação e cabos de conexão ao computador).
- Computador pessoal e software assembler

6. PROCEDIMENTOS EXPERIMENTAIS

6.1 Atenção a teclado e display por o método de espera ou pesquisa

Observações importantes:

- a) O conector J20 é empregado para descarregar (usando o protocolo SPI, ver pág. 25 da bibliografia 1) o programa a executar no kit. Os jumpers devem ser colocados somente durante o processo de descarga do programa. Posteriormente devem ser tirados para não afeitar o funcionamento das linhas P1_5, P1_6 e P1_7.
- b) O conector J8 é empregado para RESET/PROGRAMAÇÃO. Na função de programação, os terminais 2 e 3 de J8 devem estar conectados. Na função do RESET manual (com o botão RESET1) os terminais 1 e 2 de J8 devem estar conectados (ver SILKSCREEN do kit). Uma vez que o programa tenha sido descarregado para o kit, pode programar-se na função do RESET manual.

6.1.1 Realizar as conexões necessárias para resolver o problema orientado na preparação previa. Calcular a correntes que circulam por cada um dos segmentos do display nas condições de seu programa de aplicação (Procure a informação dos transistores BC857).

6.2 Atenção a teclado e display por o método de interrupção

6.2.1 Realizar as conexões necessárias e o programa de contagem ascendente/descendente anterior, mas agora, usando as interrupções externas #INT1 e #INT0, através dos switches SW5 e SW6. A distribuição neste caso é a seguinte:

Equipes pares: SW5/Ascendente e SW6/Descendente

Equipes impares: SW5/Descendente e SW6/Ascendente

11.RELATÓRIO

A estrutura do relatório é a seguinte:

- Portada com dados gerais (Disciplina, Título da pratica, nomes e equipe dos estudantes)
- Resumo da pratica (máximo dois parágrafos)
- Objetivos

- Esquema elétrico simplificado desta pratica elaborado com o programa PROTEUS ISIS mostrando de alguma forma (linhas descontínuas, cartéis de texto, etc.) os diferentes conectores empregados do sistema de desenvolvimento (kit). Calcular os valores das correntes e tensões associados aos circuitos de manuseio dos LEDs (Procurar parâmetros dos dispositivos em internet, por exemplo, hFE transistores).
- Programas em assembler e em linguagem C com seus correspondentes comentários. Se aparecerem demoras, explicar seu cálculo detalhado.
- Resultados obtidos no laboratório e sua discussão (explicação dos mesmos).
- Conclusões

12. BIBLIOGRAFIA

- Microcontroladores. Programação e Projeto com a Família 805. Ricardo Zelenovsky, Alexandre Mendonça, Editora Ltda, 2005, ISBN: 85-87385-12-7.
 - a. Capítulo 13: Programação em C para 8051, detalhar no emprego de códigos em assembly (apartado 13.9.6, pág. 351)
 - b. Capítulo 6: Portas paralelas (ver exercício 6.1, pág. 130)
 - c. Aulas MP9_2.PPT (assembler), MP11.PPT (linguagem C) e MP12.PPT (Portas paralelas e displays de 7 segmentos).
- Circuitos elétricos do kit de treinamento (ver correio eletrônico e ANEXOS)
- Aula MP12. PPT

13. PREGUNTAS DE REPASO

Modificassem-se os valores das resistências do circuito de excitação a transistores, se em lugar de trabalhar com um só display, usam-se 2 displays os quais se excitam de forma multiplexada ?.

14. ANEXOS

10.1 Fotografia do sistema de desenvolvimento (ver guia de pratica 1)

5.3 LABORATORIO 3: ATENÇÃO A DISPLAY LCD.

TÍTULO: Atenção a display LCD.

1. OBJETIVO:

Elaborar programas nas linguagens assembler e C para visualizar informação num display de cristal líquido.

2. INTRODUÇÃO TEÓRICA

Os displays de 7 segmentos usados na pratica anterior tem dos grandes limitações: o consumo e a possibilidade limitada de saída de caracteres alfanuméricos. Atualmente, se tem difundido o uso dos displays de cristal liquida, e especialmente do modulo de Hitachi LM016. Estes módulos têm controladores embarcados que são dedicados somente ao processo de conversão dos códigos de entrada (tipicamente: ASCII) em sinais adequadas para sua visualização nos displays LCD. O programa que se executa no microcontrolador deve enviar uma série de comandos de controle para definir os diferentes modos de trabalho, e posteriormente deve enviar os dados que se desejem mostrar. Além disto, em ocasiões é necessário realizar leituras no registro de estado para conhecer do estado interno do controlador, por exemplo, depois de uma operação de apagado, não se pode realizar nenhuma escritura até depois de determinado tempo. Na prática de hoje, trabalharemos com um módulo LCD LM016. O mesmo está concebido para excitado de forma paralela (ver bibliografias 1 e 2). O sistema de desenvolvimento empregado nesta disciplina leva um display de cristal líquido (ver circuito elétrico na parte de Recursos Visuais). As 8 linhas de dados vão conectadas ao conector J14. O sistema de desenvolvimento está concebido para usar a porta P1 (conector J6) para excitar ao display. Para isto se deve unir cada uma das linhas do conector J4 com as correspondentes do conector J6. Alem de isto, as linhas de controle RS e E estão conectadas diretamente às linhas P3_6 e P3_7 do microcontrolador. O sinal R/#W do display LCD está conectada a terra, por isso somente é possível realizar operações de escrita no display LCD (não e possível ler o registrador de estado do módulo LCD).

3. PREPARAÇÃO PREVIA

Estudar detalhadamente na conexão display LCD-microcontrolador na área de recursos visuais no esquema elétrico do sistema de desenvolvimento com 8051 (ver Anexos de esta Guia).

Estudar como realizar a conexão dos switches SW1 a SW8 (conector J2) à porta P0 (conector J3).

Estudar as instruções da linguagem C e assembler e a forma de realizar o interfaceamento entre ambos.

3.1 NOTA SOBRE AVALIAÇÃO DE CADA LABORATORIO

Será realizada uma avaliação de entrada que definirá o 30 % da nota final do laboratório segundo a seguinte formula:

$$NF = (0,3 AE + 0,5 DP + 0,2 RE)/3 \quad \text{onde,}$$

AE é a Avaliação de entrada (ver apartado 4), DP é o Desenvolvimento Prático e RE é a nota do relatório.

4. AVALIAÇÃO DE ENTRADA AO LABORATORIO

Problema de entrada:

Deseja-se programar um contador reversível de 0 até 9999 que seja dirigido por 2 teclas: uma para contar ascendente e outra para contar descendente. Se se pressionassem ambas as teclas, o contador se detém. A contagem deve mostrar-se na segunda linha do display LCD. Na primeira linha deverá mostrar seu nome e número de CPF. O intervalo de tempo entre cada operação é de 1 segundo.

Realizar:

- a) O programa em linguagem montador
- b) O programa em linguagem C

Equipe	Teclas a usar/Forma de contagem
1	SW1/Ascendente, SW2/Descendente
2	SW3/Ascendente, SW4/Descendente
3	SW5/Ascendente, SW6/Descendente
4	SW2/Ascendente, SW3/Descendente
5	SW4/Ascendente, SW5/Descendente
6	SW1/Descendente, SW2/Ascendente
7	SW3/Descendente, SW4/Ascendente
8	SW5/Descendente, SW6/Ascendente

Entregar à entrada do laboratório em formato eletrônico (ou enviar por correio eletrônico antes de entrar em laboratório) os seguintes arquivos:

Circuito elétrico (SIMPLIFICADO) em PROTEUS ISIS da pratica de laboratório (*.DSN)

Programas em linguagem assembler (*.ASM e *.HEX correspondente) e C (*.C e *.HEX correspondente) da pratica de laboratório.

Se o professor estima conveniente, realizará uma pergunta escrita à entrada do laboratório relativa aos conteúdos teóricos de base deste laboratório o aos procedimentos experimentais. Neste caso, a nota da avaliação de entrada é a media desta e dos arquivos anteriores.

5. MATERIAL UTILIZADO

Kit de treinamento e acessórios (fonte de alimentação e cabos de conexão ao computador).

Computador pessoal e software assembler

6. PROCEDIMENTOS EXPERIMENTAIS

6.1 Atenção a teclado e display por o método de espera ou pesquisa

Observações importantes:

O conector J20 é empregado para descarregar (usando o protocolo SPI, ver pág. 25 da bibliografia 1) o programa a executar no kit. Os jumpers devem ser colocados somente durante o processo de descarga do programa. Posteriormente devem ser tirados para não afeitar o funcionamento das linhas P1_5, P1_6 e P1_7.

O conector J8 é empregado para RESET/PROGRAMAÇÃO. Na função de programação, os terminais 2 e 3 de J8 devem estar conectados. Na função do RESET manual (com o botão RESET1) os terminais 1 e 2 de J8 devem estar conectados (ver SILKSCREEN do kit). Uma vez que o programa tenha sido descarregado para o kit, pode programar-se na função do RESET manual.

6.1.1 Realizar as conexões necessárias para resolver o problema orientado na preparação previa. Calcular a correntes que circulam por cada um dos segmentos do display nas condições de seu programa de aplicação (Procure a informação dos transistores BC857).

7. RELATORIO

A estrutura do relatório é a seguinte:

Portada com dados gerais (Disciplina, Título da pratica, nomes e equipe dos estudantes)

Resumo da pratica (máximo dois parágrafos)

Objetivos

Esquema elétrico simplificado desta pratica elaborado com o programa PROTEUS ISIS mostrando de alguma forma (linhas descontínuas, cartéis de texto, etc.) os diferentes conectores empregados do sistema de desenvolvimento (kit).

Programas em assembler e em linguagem C com seus correspondentes comentários. Se aparecerem demoras, explicar seu cálculo detalhado.

Resultados obtidos no laboratório e sua discussão (explicação dos mesmos).

Conclusões

BIBLIOGRAFÍA

- Circuitos elétricos do kit de treinamento (ver correio eletrônico e ANEXOS)
- Arquivo lcdport.pdf (Informação do LCD)
- Microcontroladores. Programação e Projeto com a Família 805. Ricardo Zelenovsky, Alexandre Mendonça, Editora Ltda, 2005, ISBN: 85-87385-12-7. Capítulo 13: Programação em C para 8051, detalhar no emprego de códigos em assembly (apartado 13.9.6, pág. 351)
- Aula MP13.PPT

PREGUNTAS DE REPASO

Modificassem-se os valores das resistências do circuito de excitação a transistores, se em lugar de trabalhar com um só display, usam-se 2 displays os quais se excitam de forma multiplexada.

ANEXOS

Ver fotografia da pratica 1.

5.4 LABORATORIO 4: USO DAS INTERRUPTÇÕES EXTERNAS.

TÍTULO: Uso das interrupções externas.

1. OBJETIVOS:

Elaborar programas na linguagem C para uso das interrupções externas.

2. INTRODUÇÃO TEÓRICA:

As interrupções são concebidas para atender eventos em tempo real. Elas permitem ganhar em eficiência na execução dos programas, devido a que não é preciso perder tempo em pesquisar o estado de um evento. No microcontrolador 8051 tem 5 fontes de interrupção: 2 externas (INT0# e INT1#) e 3 internas (Timer 0, Timer 1 e Porta Serial). No sistema de desenvolvimento cada um dos pinos INT0# e INT1# estão ligados a dos switches SW7 e SW8 o qual permite ao usuário gerar interrupções de uma forma fácil.

Estas interrupções geram vetores fixos. Ao produzir uma delas, o microcontrolador guarda o conteúdo do Contador de programa (PC) na pilha e salta a uma dos seguintes endereços: 0003H (INT0#) e 0013H (INT1#). O programador deve colocar nestes endereços uma instrução “JMP Endereço” apontando à rotina de atenção à interrupção, ou a própria rotina (se esta ocupar menos de 8 bytes).

Os registradores usados para o controle das interrupções (ver bibliografia 3) são:

IE: Permite habilitar ou desabilitar as interrupções de forma global ou individual.

TCON: Permite definir se as interrupções externas são ativas por nível ou por borda e controlar (leitura/zerar) algumas flags das interrupções do timer e externas.

IP: Permite programar as prioridades das diversas interrupções em um de dois níveis de prioridade (alto e baixo).

O exemplo de uso das interrupções externas desta prática tem um caráter didático e está determinado pela arquitetura do sistema de desenvolvimento. A idéia é gerar manualmente uma interrupção mediante a opressão de um interruptor. Os sinais de solicitude de interrupções são usados para comunicar ao microprocessador sobre algumas situações de alarmes e para gerar temporizações (nos sistemas que não têm timer internos), entre outras aplicações.

3. PREPARAÇÃO PREVIA

Estudar detalhadamente os conceitos gerais sobre interrupções e concretamente o sistema de interrupções do microcontrolador 8051 (consultar palestra MP16_INTERRUPCOES.PPT). Faça um resumo sobre os significados de cada um dos bits dos registradores usados para o controle das interrupções (IE, TCON e IP), as instruções (em assembler e no linguagem C) relacionadas com as interrupções.

Na conexão display LCD-microcontrolador na área de recursos visuais no esquema elétrico do sistema de desenvolvimento com 8051 (ver Anexos de esta Guia).

Estudar como realizar a conexão dos switches SW1 a SW8 (conector J2) à porta P0 (conector J3). Observe que os switches SW7 e SW8 também estão conectados diretamente aos pinos P3_3(pino #INT1) e P3_2 (pino #INT0)

Estudar as instruções da linguagem C e assembler e a forma de realizar o interfaceamento entre ambos.

3.1 NOTA SOBRE AVALIAÇÃO DE CADA LABORATORIO

Será realizada uma avaliação de entrada que definirá o 30 % da nota final do laboratório segundo a seguinte formula:

$$NF = (0,3 AE + 0,5 DP + 0,2 RE)/3 \quad \text{onde,}$$

AE é a Avaliação de entrada (ver apartado 4), DP é o Desenvolvimento Prático e RE é a nota do relatório.

4. AVALIAÇÃO DE ENTRADA AO LABORATORIO

Problema de entrada:

Deseja-se programar um contador reversível de 0 até 999 que seja dirigido por 2 teclas: uma para contar ascendente e outra para contar descendente. Se ambas teclas são pressionadas, o contador será parado. A contagem deve mostrar-se no display LCD segundo o seguinte código (ver tabela embaixo):

0: Linha 0 ou superior, 1: Linha 1 ou inferior,

D: Extrema Direita da linha, C: Centro da linha, E: Esquerda da linha

O intervalo de tempo entre cada operação é de 1 segundo.

Realizar:

a) O programa em linguagem assembler

b) O programa em linguagem C

Equipe	Teclas a usar/Linha/Lugar no display LCD
1	SW7/0/E, SW8/0/C
2	SW7/0/C, SW8/0/D
3	SW7/1/E, SW8/1/C
4	SW7/1/C, SW8/0/D
5	SW7/0/E, SW8/0/D
6	SW7/1/E, SW8/1/D
7	SW7/0/E, SW8/1/E
8	SW7/0/E, SW8/1/D

Entregar à entrada do laboratório em formato eletrônico (ou enviar por correio eletrônico antes de entrar em laboratório) os seguintes arquivos:

Circuito elétrico (SIMPLIFICADO) em PROTEUS ISIS da pratica de laboratório (*.DSN)

Programas em linguagem assembler (*.ASM e *.HEX correspondente) e C (*.C e *.HEX correspondente) da pratica de laboratório.

Se o professor estima conveniente, realizará uma pergunta escrita à entrada do laboratório relativa aos conteúdos teóricos de base deste laboratório o aos procedimentos experimentais. Neste caso, a nota da avaliação de entrada é a media desta e dos arquivos anteriores.

5. MATERIAL UTILIZADO

Kit de treinamento e acessórios (fonte de alimentação e cabos de conexão ao computador).

Computador pessoal e software assembler

6. PROCEDIMENTOS EXPERIMENTAIS

6.1 Emprego das interrupções externas: Contagem

Observações importantes:

O conector J20 é empregado para descarregar (usando o protocolo SPI, ver pág. 25 da bibliografia 1) o programa a executar no kit. Os jumpers devem ser colocados

somente durante o processo de descarga do programa. Posteriormente devem ser tirados para não afeitar o funcionamento das linhas P1_5, P1_6 e P1_7.

O conector J8 é empregado para RESET/PROGRAMAÇÃO. Na função de programação, os terminais 2 e 3 de J8 devem estar conectados. Na função do RESET manual (com o botão RESET1) os terminais 1 e 2 de J8 devem estar conectados (ver SILKSCREEN do kit). Uma vez que o programa tenha sido descarregado para o kit, pode programar-se na função do RESET manual.

6.1.1 Realizar as conexões necessárias para resolver o problema orientado na preparação previa. Carregar os programas (em assembler e C) no sistema de desenvolvimento.

7. RELATORIO

A estrutura do relatório é a seguinte:

Portada com dados gerais (Disciplina, Título da pratica, nomes e equipe dos estudantes)

Resumo da pratica (máximo dois parágrafos)

Objetivos

Esquema elétrico simplificado desta pratica elaborado com o programa PROTEUS ISIS mostrando de alguma forma (linhas descontínuas, cartéis de texto, etc.) os diferentes conectores empregados do sistema de desenvolvimento (kit).

Programas em assembler e em linguagem C com seus correspondentes comentários. Se aparecerem demoras, explicar seu cálculo detalhado.

Resultados obtidos no laboratório e sua discussão (explicação dos mesmos).

Conclusões

8. BIBLIOGRAFÍA

- Circuitos elétricos do kit de treinamento (ver correio eletrônico e ANEXOS)
- Aula MP15_TIMER.PPT
- Aula MP16_INTERRUPTOES.PPT
- Microcontroladores. Programação e Projeto com a Família 8051. Ricardo Zelenovsky, Alexandre Mendonça, Editora Ltda, 2005, ISBN: 85-87385-12-7.
- Capítulo 8: Interrupções, ver exercícios resolvidos.

- Capítulo 13: Programação em C para 8051, epígrafe 13.9.5 (emprego de interrupções com o SDCC), ver exercícios resolvidos 13.1 a 13.3
- Microcontrolador 8051 com Linguagem C. Prático e Didático (Família AT89S8252), Editora Erica Ltda, 2005, Denys Emilio Campion Nicolosi, Rodrigo Barbosa Bronzeri, ISBN: 85-365-0079-4.
- Capítulo 2: O Compilador C, epígrafes 2.17 Funções e 2.17.1 Atendimento às interrupções.

10. PREGUNTAS DE REPASO

10.1 Calcular de forma aproximada quanto tempo demora em executar a rotina de atenção à interrupção e que por cento de tempo representa isto para as tarefas do programa principal. Você acha que este sistema é eficiente?

10.2 Você acha que este kit oferece possibilidades de usar alguns dos dois timers do microcontrolador como contadores? Justifique sua resposta e proponha soluções para um futuro projeto.

5.5 LABORATORIO 5: USO DOS TEMPORIZADORES NO 8051.

TÍTULO: Uso dos temporizadores no 8051

1. OBJETIVOS

Elaborar programas na linguagem C para contagem de eventos externos e para criação de uma base de tempo real para sincronizar tarefas (relógio de tempo real).

2. INTRODUÇÃO TEÓRICA

Os microcontroladores controlam processos que ocorrem em tempo real. Para adquirir informação se realiza uma amostragem cada certo tempo, em dependência da magnitude de medição. Os atuadores de saída trabalham cada certo tempo, por isso precisam de uma variável de contagem do tempo.

O relógio de tempo real permite contar o tempo real empregando uma variável em memória (ou em registros) que é usada como elemento de sincronização das diversas ações a realizar.

Pode-se criar uma base de tempo (relógio de tempo real) no microcontrolador utilizando a interrupção de um dos timers. Pode-se, por exemplo, programar o timer 0 em modo 0 ou 1 e assim obter uma interrupção periódica cada certo número de

milissegundos. Dentro do programa que atende esta interrupção, setar-se uma bandeira que pode ser consultada pelo programa. Se a bandeira estiver em 1 significa que transcorreu o tempo previsto e então se pode executar o evento que se deseja sincronizar. Uma vez realizado o evento se zera a bandeira e continua a consulta.

O seguinte exemplo ilustra esta maneira de proceder.

Exemplo:

Tem-se um 8051 com um cristal de 12 MHz. Deseja-se formar uma base de tempo de 1 seg e sincronizada a ela, alternar o valor do porto P1 entre 0 e FFh.

Pode calcular-se que se o cristal for de 12 MHz e o timer 0 se programa em modo 0, a interrupção ocorre a cada 8,19 mseg (122 Hz) aproximadamente. Para alcançar um segundo terá que contar 122 interrupções (tics) do contador. Para isso se pode utilizar o registro R7. Como bandeira indicadora do segundo se utilizou o bit PSW.5 (F0), a qual é seteada cada um segundo no programa da interrupção do timer e consultada e posta a zero no programa principal.

```
;
; Exemplo: P1 pisca cada 1 seg, sincronizado a uma base de tempo.
; File: RELOJ.ASM
;
    ORG    0
    JMP    INICIO

    ;

    ORG    0Bh                ;Endereço da interrupção TIMER0
    JMP    TIMER_0

    ;

    ; Iniciação do sistema:

    ;

INICIO:

    CLR    EA                ;Inabilitar todas as interrupções

    MOV     TMOD, #00        ;Programar timer 0 como temporizador,
                               ;modo 0, controle por software.

    MOV     TH0, #0
```

```

MOV    TL0, #0           ;Por valor do contador em 0.
MOV    R7, #122          ;Contador dos tics de relógio (122 tics= 1 seg).
CLR    F0                 ;PSW.5 (F0 indicador do segundo)
;
SETB   TR0               ;Habilitar contagem do timer 0
MOV    IE, #82H          ; Habilitar só a interrupção do timer 0
;
; Programa principal:
;
PROG:
    JNB  F0, SIGUE        ;Se consulta F0: si F0=0 ir a SIGUE,
    MOV  A, P1             ;si F0=1, realizar evento sincronizado.
    CPL  A
    MOV  P1, A
    CLR  F0               ;Zerar F0
SIGUE:
    JMP  PROG
;
; Interrupção do timer 0:
;
TIMER_0:
    DJNZ R7, FIN_0        ;Se R7 não é 0, voltar,
    MOV  R7, #122         ;se R7=0, recarregar R7.
    SETB F0               ;Setear bandeira do segundo.
FIN_0:
    RETI                  ;Retornar.
;
END

```

Na prática de hoje, apresentaremos duas aplicações dos temporizadores: relógio de tempo real e um contador de eventos. O estudante deve revisar a conexão do display LCD no kit (ver Guia Metodológica do Laboratório 3).

3. PREPARAÇÃO PREVIA

Estudar detalhadamente na conexão display LCD-microcontrolador na área de recursos visuais no esquema elétrico do sistema de desenvolvimento com 8051 (ver Anexos de esta Guia).

Estudar como realizar a conexão dos *switches* SW1 a SW8 (conector J2) à porta P0 (conector J3). Observe que os *switches* SW7 e SW8 também estão conectados diretamente aos pinos P3_3(pino #INT1) e P3_2 (pino #INT0)

Estudar as instruções da linguagem C e *assembler* e a forma de realizar o interfaceamento entre ambos.

4. AVALIAÇÃO DE ENTRADA AO LABORATORIO

Problema de entrada:

Com o sistema de desenvolvimento, realizar um cronômetro digital que conte (e mostre no display LCD) as centésimas de segundos e que disponha de duas teclas (ver tabela por equipes) cuja opressão deterá e habilitará, respectivamente, a contagem. Não se podem pressionar ambas as teclas ao mesmo tempo.

Realizar:

a) O programa em linguagem *assembler*

b) O programa em linguagem C

Equipe	Teclas a usar/Modo de contagem
1	SW7/Habilitar, SW1/Desabilitar
2	SW8/ Habilitar, SW2/ Desabilitar
3	SW3/ Habilitar, SW7/ Desabilitar
4	SW4/ Habilitar, SW8/ Desabilitar
5	SW7/ Habilitar, SW4/ Desabilitar
6	SW8/ Habilitar, SW3/ Desabilitar
7	SW7/ Habilitar, SW2/ Desabilitar
8	SW8/ Habilitar, SW1/ Desabilitar

Entregar à entrada do laboratório em formato eletrônico (ou enviar por correio eletrônico antes de entrar em laboratório) os seguintes arquivos:

Circuito elétrico (**SIMPLIFICADO**) em PROTEUS ISIS da pratica de laboratório (*.DSN)

Programas em linguagem *assembler* (*.ASM e *.HEX correspondente) e C (*.C e *.HEX correspondente) da prática de laboratório.

Se o professor estima conveniente, realizará uma pergunta escrita à entrada do laboratório relativa aos conteúdos teóricos de base deste laboratório ou aos procedimentos experimentais. Neste caso, a nota da avaliação de entrada é a média desta e dos arquivos anteriores.

5. MATERIAL UTILIZADO

Kit de treinamento e acessórios (fonte de alimentação e cabos de conexão ao computador).

Computador pessoal e software *assembler*

6. PROCEDIMENTOS EXPERIMENTAIS

6.1 Aplicação da temporização: Cronômetro

Observações importantes:

O conector J20 é empregado para descarregar (usando o protocolo SPI, ver pág. 25 da bibliografia 1) o programa a executar no kit. Os *jumpers* devem ser colocados somente durante o processo de descarga do programa. Posteriormente devem ser tirados para não afetar o funcionamento das linhas P1_5, P1_6 e P1_7.

O conector J8 é empregado para RESET/PROGRAMAÇÃO. Na função de programação, os terminais 2 e 3 de J8 devem estar conectados. Na função do RESET manual (com o botão RESET1) os terminais 1 e 2 de J8 devem estar conectados (ver SILKSCREEN do kit). Uma vez que o programa tenha sido descarregado para o kit, pode programar-se na função do RESET manual.

6.1.1 Realizar as conexões necessárias para resolver o problema orientado na preparação previa.

6.1.2 A partir do cronômetro anterior, realize um programa em linguagem C que adicione a possibilidade de contagem de segundos e minutos, e mostrar no display LCD.

7. RELATORIO

A estrutura do relatório é a seguinte:

Portada com dados gerais (Disciplina, Título da pratica, nomes e equipe dos estudantes)

Resumo da pratica (máximo dois parágrafos)

Objetivos

Esquema elétrico simplificado desta pratica elaborado com o programa PROTEUS ISIS mostrando de alguma forma (linhas descontínuas, cartéis de texto, etc.) os diferentes conectores empregados do sistema de desenvolvimento (kit).

Programas em *assembler* e em linguagem C com seus correspondentes comentários. Se aparecerem demoras, explicar seu cálculo detalhado.

Resultados obtidos no laboratório e sua discussão (explicação dos mesmos).

Conclusões

8. BIBLIOGRAFIA

- Circuitos elétricos do kit de treinamento (ver correio eletrônico e ANEXOS)
- Aula MP15_TIMER.PPT e MP16_INTERRUPTOES.PPT
- Microcontroladores. Programação e Projeto com a Família 8051. Ricardo Zelenovsky, Alexandre Mendonça, Editora Ltda, 2005, ISBN: 85-87385-12-7.
- Capítulo 8: Interrupções, ver exercícios resolvidos.
- Capítulo 13: Programação em C para 8051, epígrafe 13.9.5 (emprego de interrupções com o SDCC), ver exercícios resolvidos 13.1 a 13.3
- Microcontrolador 8051 com Linguagem C. Prático e Didático (Família AT89S8252), Editora Erica Ltda, 2005, Denys Emilio Campion Nicolosi, Rodrigo Barbosa Bronzeri, ISBN: 85-365-0079-4.
- Capítulo 2: O Compilador C, epígrafes 2.17 Funções e 2.17.1 Atendimento às interrupções.

9. PREGUNTAS DE REPASO

9.1 Calcular de forma aproximada quanto tempo demora para executar a rotina de atenção à interrupção e que por cento de tempo representa isto para as tarefas do programa principal. Você acha que este sistema é eficiente?

9.2 Você acha que este kit oferece possibilidades de usar alguns dos dois *timers* do microcontrolador como contadores? Justifique sua resposta e proponha soluções para um futuro projeto.

10. ANEXOS

10.1 Formación de una base de tiempo y sincronización de eventos a la misma.

Criação de uma base de tempo.

Pode-se criar uma base de tempo (relógio de tempo real) no microcontrolador utilizando a interrupção de um dos *timers*. Pode-se, por exemplo, programar o timer 0 em modo 0 ou 1 e assim obter uma interrupção periódica (tic do relógio) cada certo número de milissegundos. Emprega-se então uma variável (variável contadora dos tics do relógio) que se incrementa com cada interrupção até chegar à contagem correspondente ao segundo. Com outras variáveis se pode levar a contagem dos segundos, minutos, horas, etc.

A seguir se mostram alguns programas em linguagem C para mostrar algumas idéias sobre o como criar uma base de tempo e como sincronizar eventos. Estes programas foram desenvolvidos para ser compilados com um compilador diferente ao SDCC. Os nomes das funções e as variáveis aparecem em espanhol, já que o mesmo foi realizado neste idioma. Pedimos desculpas e recomendamos ao estudante que realize sua tradução e melhora dos mesmos.

Exemplo 1: Programa para realizar um relógio: (arquivo reloj1.c)

```
#include <8051.h>

#include <intrpt.h>

void interrupt i_T0(void);

void inicializa(void);

unsigned char horas, minutos, segundos, milisegundos;

main()
{
    set_vector(800bh,i_T0); /* APUNTA A LA RUTINA DE SERVICIO */
    inicializa(); /* INICIALIZAR EL SISTEMA */
    for(;;); /* ESPERA INDEFINIDA */
}

void inicializa(void)
```



```

/* RUTINA DE INICIALIZACION */

{
    milisegundos=0; /* TIEMPO INICIAL = 0 */
    segundos=0;
    TMOD=0x01; /* SE SELECCIONA EL T0 EN MODO 1 */
    TL0=0xB0; /* SE INICIALIZAN LOS REGISTROS DEL TIMER */
    TH0=0x3C;
    TCON=0x10; /* SE ACTIVA EL TIMER 0 MEDIANTE TR0 = 1 */
    IP_BITS.B1=1; /* SE CONCEDE MAXIMA PRIORIDAD AL TIMER 0 */
    IE=0x82; /* SE HABILITA LA INTERRUPCION DEL TIMER 0 */
}

void interrupt i_T0(void)

/* RUTINA DE SERVICIO A LA INTERRUPCION DEL TIMER 0 */

{
    TL0=0xB2; /* RECARGA EL VALOR INICIAL DE CUENTA DEL TIMER */
    TH0=0x3C;

    /*
        SE INCREMENTA EL VALOR DE MILISEGUNDOS DE 50 EN 50 COMPROBANDO
        SI SE HA LLEGADO A OTRO SEGUNDO, DESPUES SE COMPRUEBA SI SE
        HA LLEGADO A OTRO MINUTO Y FINALMENTE SE OBSERVA SI SE HA ALCANZADO
        OTRA HORA. SI ES ASI, PONE A CERO EL VALOR CORRESPONDIENTE
    */

    milisegundos++;
    if(milisegundos == (unsigned char)20)
    {
        milisegundos=0;
        segundos++;
        P1=!P1;
    }
}

```

Exemplo 2:

/* Programa (reloj2.c) para mostrar o esquema de programação para a sincronização de eventos ao relógio de tempo real */

```
#include "8051.h"

#include "intrpt.h"

#define TRUE 1

#define FALSE 0

#define T1    3

#define T2    5

#define T3    7

void interrupt Reloj (void);

void FazerEvento (void);

void FazerEvento1 (void);

void FazerEvento2 (void);

void FazerEvento3 (void);

unsigned char miliseg50, seg, seg1, seg2, seg3;

unsigned char flagseg, flagseg1, flagseg2, flagseg3;

main()

{

    set_vector (TIMER0, Reloj); /*Apuntar à rotina do relógio */

    /* Se programam os registradores del microcontrolador: */

    TMOD = 0x01; /* Se selecciona el Timer 0 en modo 1 */

    TLO  = 0xb0; /* Se inicializan los registros del timer 0 */

    TH0  = 0x3c;

    TCON = 0x10; /* Se activa el Timer 0 mediante TR0=1 y se activan las

        interrupciones externas por nivel con IT0 = IT1 =0 */

    IP   = 0x02; /* Alta prioridad a la interrupción del timer 0

        y baja a las demas interrupciones */

    IE   = 0x82; /* Se habilita solamente la interrupcion del timer 0 */

    while (TRUE) /* Lazo infinito */

    {
```

```

if (flagseg)
{
    flagseg = FALSE;
    FazerEvento ();
}
if (flagseg1)
{
    flagseg1 = FALSE;
    FazerEvento1 ();
}
if (flagseg2)
{
    flagseg2 = FALSE;
    FazerEvento2 ();
}
if (flagseg3)
{
    flagseg = FALSE;
    HacerEvento3 ();
}
}
}

void interrupt Reloj (void)
/* Esta funcion es llamada cada 50 mseg (si el cristal
   del microcontrolador es de 12 MHz) por la interrupcion
   del Timer 0. */
{
    miliseg50 ++;
    if (miliseg50 == 20)
    {
        miliseg50 = 0;
    }
}

```

```

    flagseg = TRUE;
    seg1 ++;
    if (seg1 == T1)
    {
        seg1 = 0; flagseg1 = TRUE;
    }
    seg2 ++;
    if (seg2 == T2)
    {
        seg2 = 0; flagseg2 = TRUE;
    }
    seg3 ++;
    if (seg2 == T3)
    {
        seg3 = 0; flagseg3 = TRUE;
    }
}

void FazerEvento (void)
{
}

void FazerEvento1 (void)
{
}

void FazerEvento2 (void)
{
}

void FazerEvento3 (void)
{
}

```

