MICROCONTROLADORES MCS51®

Hugo Vieira Neto, M.Sc.
(hugo@cefetpr.br)

Curitiba, 2002

SUMÁRIO

Sistemas Microprocessados 2
- Hardware 2
- Software 5

Microcontroladores 7

Microcontroladores MCS51 8
- *Parte 1 – Hardware Básico*
  - Disposição dos Terminais 9
  - Descrição Funcional dos Terminais 10
  - Arquitetura Interna 12
  - Organização da Memória 13
  - Bancos de Registros 14
  - Registros Bit Endereçáveis 15
  - Registros de Função Especial 15
  - Clock 17
  - Reset 18
  - Memórias Externas de Programa e de Dados 19
- *Parte 2 - Software*
  - Programação 25
  - Modos de Endereçamento 25
  - Conjunto de Instruções 26
  - Linguagem Assembly 47
  - Tópicos Importantes em Programação 49
  - Linguagem C 51
- *Parte 3 – Hardware Avançado*
  - Ports de Entrada e Saída 52
  - Interrupções 60
  - Temporizadores / Contadores de Eventos 67
  - Interface Serial 80
  - Modos de Redução de Consumo 84

Bibliografia 86

Informações Úteis na Internet 87

Anexos
- Conjunto de Instruções MCS51
- Manual da Placa P51
- Manual do Paulmon

Tutoriais

SISTEMAS MICROPROCESSADOS

Sistemas microprocessados dividem-se basicamente em hardware e software. O hardware é constituído dos componentes físicos do sistema (dispositivos eletrônicos) e o software é constituído dos componentes lógicos (programas e dados). Chama-se de firmware o conjunto de programas gravados em ROM, específicos para o funcionamento de um determinado sistema microprocessado.

HARDWARE

Principais Dispositivos de Hardware

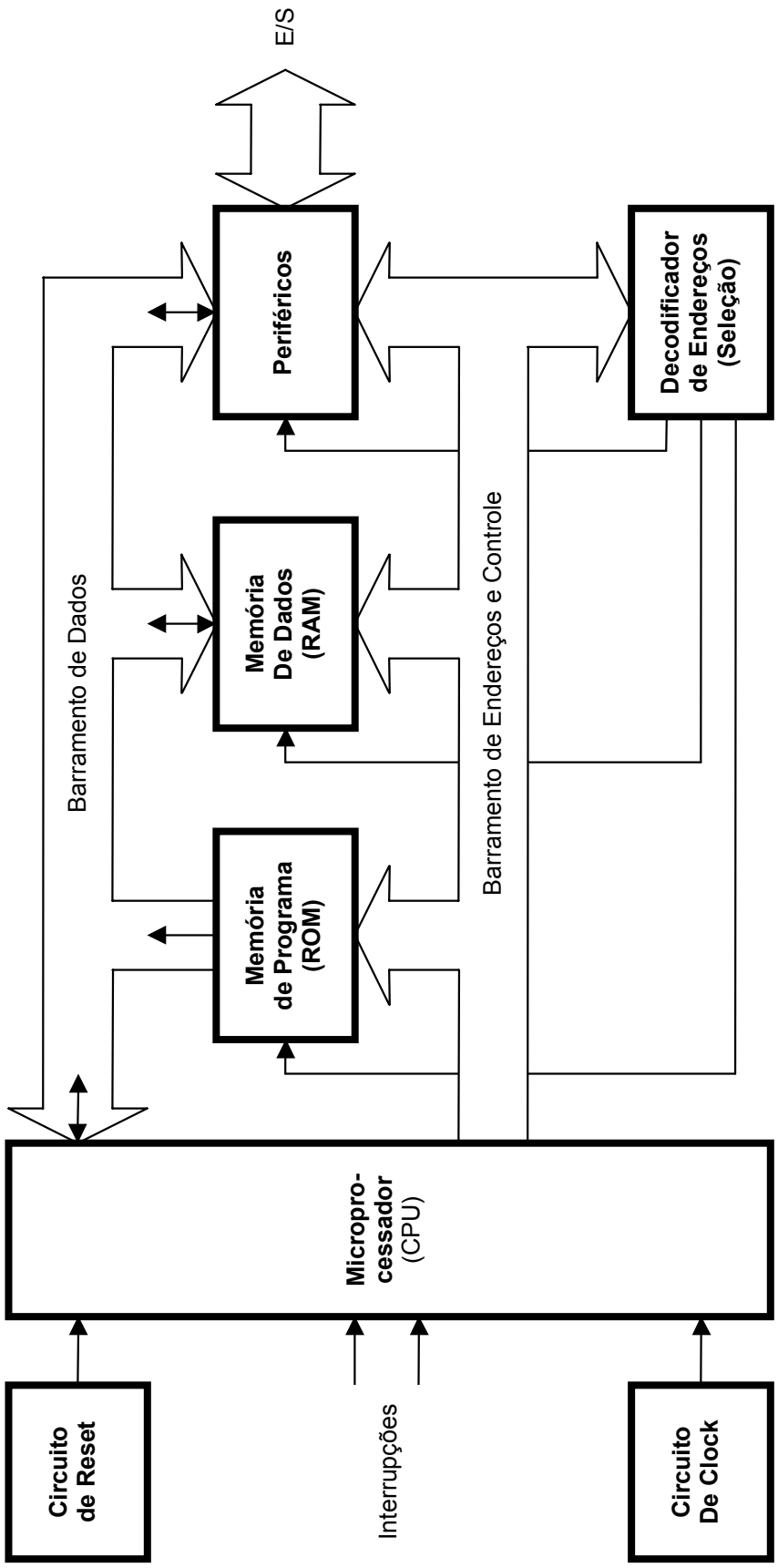
- Microprocessador (CPU): Constitui o bloco "inteligente" do sistema. Segue uma seqüência de instruções previamente armazenadas, chamada de programa. É o responsável pela execução de operações lógicas, aritméticas e de controle.
- Memória não-volátil (ROM, PROM, EPROM): Armazena a seqüência de instruções do programa a ser executado.
- Memória volátil (SRAM, DRAM): Armazena temporariamente os dados relativos ao programa. Também pode armazenar programas de maneira temporária.
- Periféricos (interface paralela, interface serial, temporizadores / contadores de eventos, entre outros): São os responsáveis pela comunicação com o mundo externo ao sistema (entrada e saída de dados).
- Decodificador de endereços: Seleciona o dispositivo a ser acionado pelo microprocessador, auxiliando o microprocessador no gerenciamento do barramento de dados.
- Circuito de reset: É o responsável pela inicialização do sistema.
- Circuito de clock: Fornece a cadência (velocidade) de execução das instruções do programa pelo microprocessador.

Principais Sinais Digitais

- Barramento de Dados (Data Bus): Consiste no conjunto de sinais digitais por onde trafegam dados entre diferentes dispositivos. Trata-se de uma via bidirecional compartilhada entre todos os componentes do sistema microprocessado. Normalmente apenas dois dispositivos fazem uso do barramento de dados em cada instante de tempo (transmissor e receptor), ficando os demais em alta impedância. Quem comanda o barramento de dados é o microprocessador, através dos barramentos de endereços e de controle, exceto durante operações de DMA (Acesso Direto à Memória), quando o controle é cedido a algum periférico.

- Barramento de Endereços (Address Bus): É o conjunto de sinais digitais através do qual são selecionados dispositivos conectados ao barramento de dados. Cada componente do sistema corresponde a um endereço ou faixa de endereços, atendendo quando solicitado pelo microprocessador ou por outro dispositivo, no caso de DMA.
- Barramento de Controle (Control Bus): Conjunto de sinais digitais que auxiliam o endereçamento dos diversos dispositivos de um sistema microprocessado, sinalizando o tipo de operação a ser efetuada. É através do barramento de controle que se definem operações de leitura ou escrita, acesso à memória, acesso aos periféricos ou requisições de DMA. Também é através de sinais especiais do barramento de controle que são realizadas interrupções no processamento do programa para atender a eventos de maior prioridade.

Diagrama em Blocos de um Sistema Microprocessado Genérico



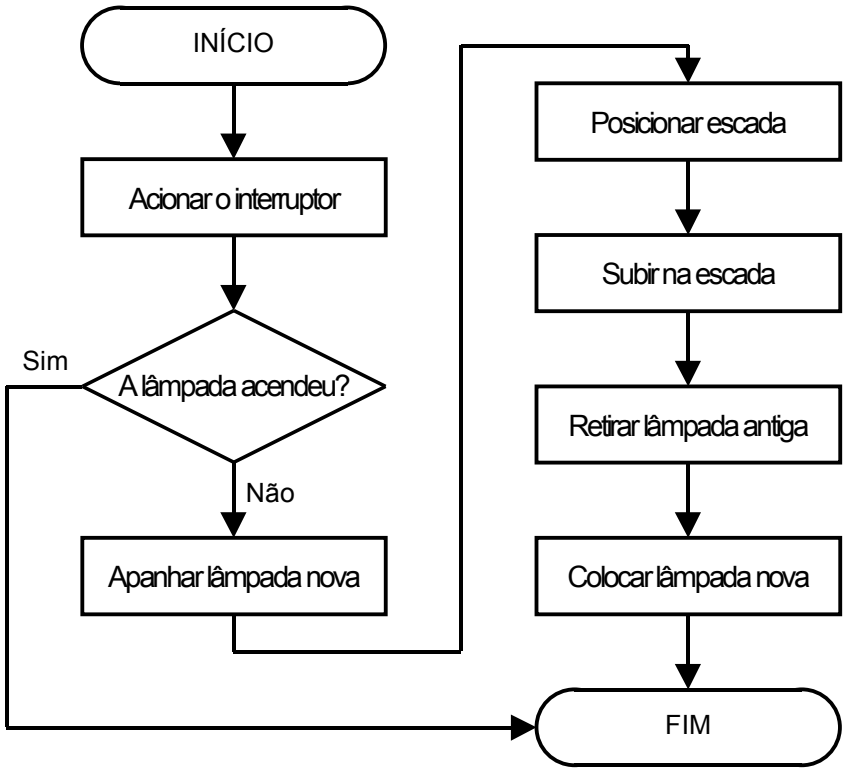
SOFTWARE

Principais Conceitos de Software

Sistemas microprocessados operam segundo a execução seqüencial de instruções e operandos armazenados em memória. A essa lista de instruções dá-se o nome de programa armazenado. Como a execução do programa é seqüencial, apenas uma instrução é executada a cada instante de tempo. Chama-se de algoritmo uma seqüência de operações simples para se realizar uma determinada tarefa mais complexa.

Uma das formas mais comuns e acessíveis de se representar um algoritmo é o fluxograma, uma técnica que consiste em representar na forma de diagrama a seqüência das operações e decisões a serem realizadas para a sua execução. O grau de refinamento das operações representadas em um fluxograma depende em grande parte dos recursos oferecidos pela linguagem de programação a ser utilizada.

Exemplo: Fluxograma para a Troca de uma Lâmpada



O uso de uma linguagem de programação faz-se necessário para a implementação real de um algoritmo na forma de programa executável, a fim de estruturar as instruções e seus respectivos operandos na seqüência a ser seguida pelo microprocessador. Cada microprocessador possui suas próprias instruções, as quais são codificadas de maneira única e constituem a chamada linguagem de máquina ou código de máquina da CPU.

Linguagens de Programação

A principal função das linguagens de programação é proporcionar ao programador uma ferramenta para elaboração de programas que os torne mais inteligíveis do que a linguagem de máquina. Sendo assim, existem linguagens de programação que se aproximam mais da linguagem do microprocessador, chamadas linguagens de baixo nível (como a linguagem Assembly), e existem linguagens que se aproximam mais da linguagem do programador, chamadas linguagens de alto nível (como a linguagem C).

Os programas implementados em linguagens diferentes da linguagem de máquina necessitam ser traduzidas para que possam ser devidamente executadas pelo microprocessador. Aplicativos que realizam a tarefa de codificar os programas para linguagem de máquina recebem os nomes de montador (assembler), no caso da linguagem Assembly, e compilador no caso da linguagem C ou qualquer outra linguagem de alto nível. Existem também os chamados interpretadores, os quais codificam e executam programas em linguagens de alto nível em tempo real. É bastante comum interpretadores para a linguagem BASIC.

Pode-se dividir um programa grande em vários arquivos contendo código-fonte, facilitando a sua manutenção (modularidade). Um outro aplicativo chamado link-editor (linker) é o reponsável pela ligação dos diversos módulos do programa para constituir a sua forma final em linguagem de máquina ou código-objeto. O link-editor é também o responsável pela ligação do código-objeto de bibliotecas de funções utilizadas em linguagens de alto nível.

Atualmente existem ambientes integrados de desenvolvimento de software, constituídos de editor de código-fonte, montador, compilador, link-editor e simulador de programas em um único aplicativo.

Ferramentas de Desenvolvimento

Além dos aplicativos necessários para o desenvolvimento de programas também são necessárias ferramentas para a validação do funcionamento do sistema como um todo (hardware e software). Para essa finalidade existem programas simuladores de hardware e software, programas monitores (debuggers), emuladores de memórias ROM e RAM, e emuladores de microprocessadores e microcontroladores, podendo estes últimos operar em tempo real ou não.

MICROCONTROLADORES

Os microcontroladores, também chamados de "microcomputadores de somente um chip", vêm revolucionando o projeto de sistemas eletrônicos digitais devido à enorme versatilidade de hardware e software que oferecem.

Um microcontrolador reúne em apenas um componente os elementos de um sistema microprocessado completo, antes desempenhados por diversos dispositivos (memória ROM, memória RAM, interface paralela, interface serial, temporizadores / contadores de eventos, controlador de interrupções, entre outros).

Talvez a vantagem mais marcante dos microcontroladores seja a possibilidade de ter seus programas gravados internamente na fabricação do componente, impedindo a engenharia reversa ou cópias não autorizadas.

Famílias de Microcontroladores

- MCS51 – Intel e outros fabricantes
- M68HC11 – Motorola
- Z8 – Zilog
- COP8 – National
- PIC – Microchip
- AVR - Atmel

MICROCONTROLADORES MCS51

Histórico

A família de microcontroladores MCS51 é uma das mais antigas existentes e, talvez por este motivo, é uma das mais conhecidas e utilizadas. Graças a essa característica, a quantidade de ferramentas de desenvolvimento e bibliotecas de software é bastante ampla e variada.

Além da Intel (fabricante original do 8051), diversas outras empresas passaram a comercializar diferentes versões da família MCS51, tais como: Atmel, Dallas Semiconductor, Intregrated Silicon Solutions, Philips, Infineon Technologies, entre outras.

Modelos

Os microcontroladores da família MCS51 possuem internamente ROM (memória de programa) e RAM (memória de dados); temporizadores / contadores de eventos; controlador de interrupções; interfaces de entrada / saída de 8 bits e interface serial síncrona / assíncrona. Os principais modelos são:

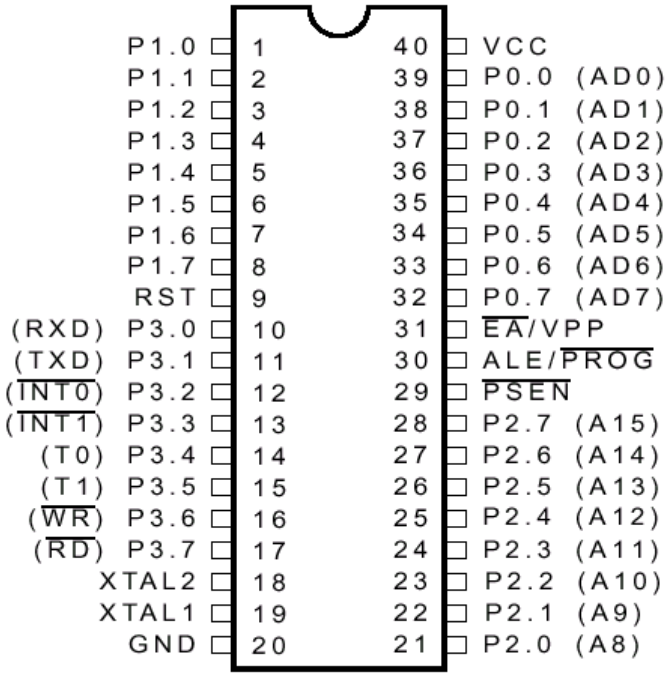
- 8031 – sem ROM (ROMLESS), 128 bytes de RAM e 2 T/C
- 8051 – com 4KB de ROM, 128 bytes de RAM e 2 T/C
- 8751 – com 4KB de EPROM, 128 bytes de RAM e 2 T/C
- 8032 – sem ROM (ROMLESS), 256 bytes de RAM e 3 T/C
- 8052 – com 8KB de ROM, 256 bytes de RAM e 3 T/C
- 8752 - com 8KB de EPROM, 256 bytes de RAM e 3 T/C

Variações

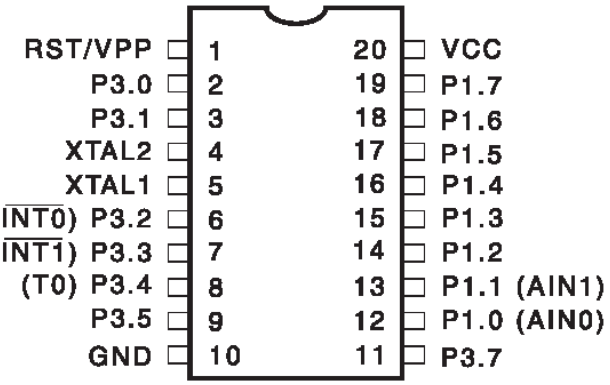
- 80C31, 80C32, 80C51 e 80C52 – versões CMOS, incluindo modos de baixo consumo de energia
- 80LV31, 80LV32, 80LV51 e 80LV52 – versões low-voltage (ISSI)
- 89C51 e 89C52 – versões com memória Flash reprogramável (Atmel, ISSI, Philips)
- 89C1051, 89C2051 e 89C4051– versões com memória Flash reprogramável, comparadores analógicos e invólucro reduzido (Atmel)
- 80C320 – versão com clock otimizado (três vezes mais veloz), capaz de operar em até 33MHz (Dallas Semiconductor)
- C505L – versão com 32KB de ROM, 512 bytes de RAM, conversor A/D de 10 bits e interface para LCD (Infineon Technologies)
- P51XA-G3, P51XA-H3 e P51XAS3 – versões com arquitetura de 16 bits (Philips)

DISPOSIÇÃO DOS TERMINAIS

AT89C51 (DIP40)



AT89C1051 (DIP20)



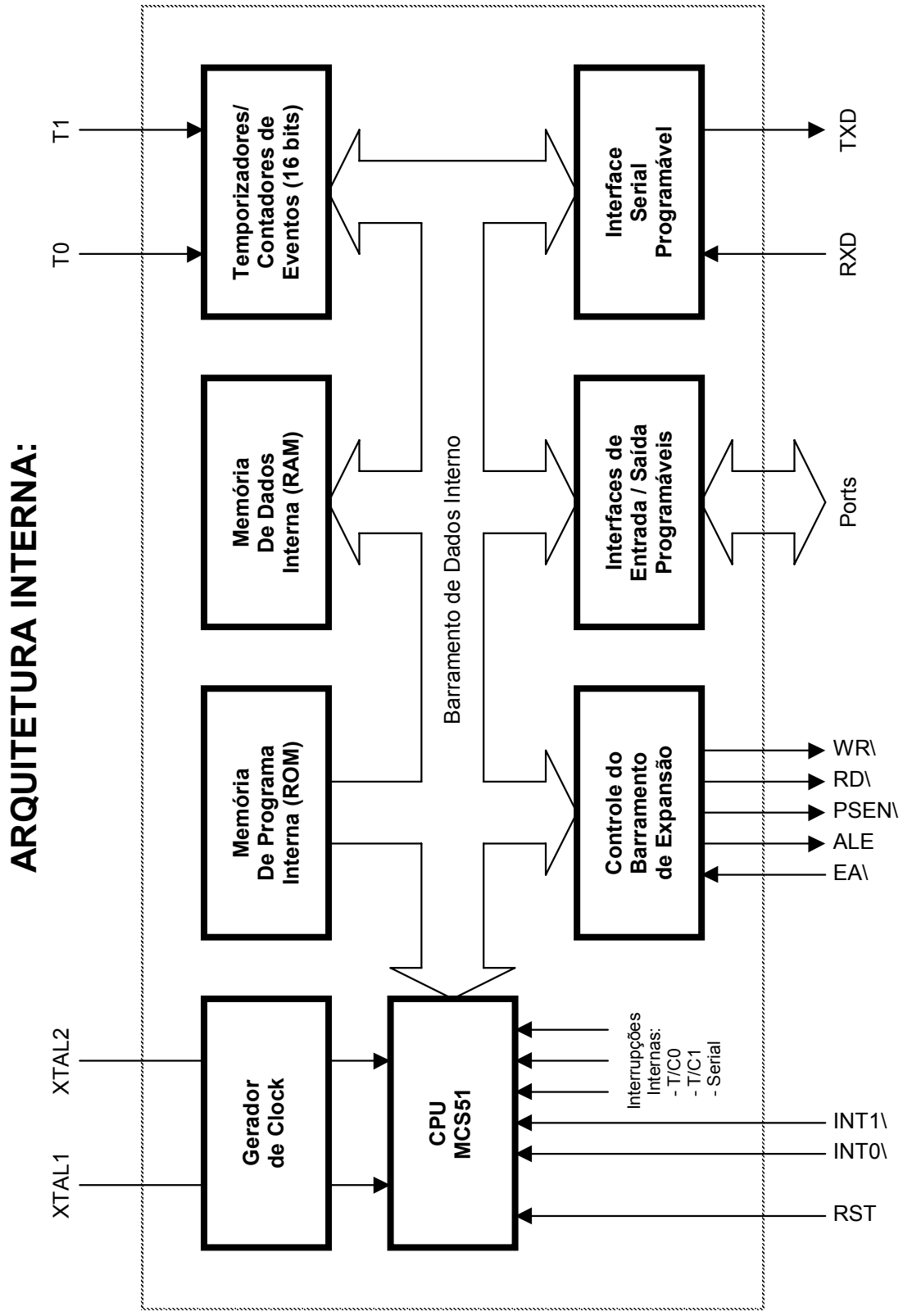
DESCRIÇÃO FUNCIONAL DOS TERMINAIS

Pino	Nome	E/S	Função
1-8	P1.0-P1.7	E/S	O Port 1 é uma interface de E/S bidirecional com 8 bits individualmente endereçáveis e resistores de pull-up internos. Terminais que estejam no estado lógico 1 podem ser utilizados como entradas.
9	RST	E	Quando aplicado nível lógico 1 a este terminal durante 2 ciclos de máquina (com o oscilador operando) ocorre o reset do microcontrolador. Um resistor interno conectado a V_{SS} permite o power-on-reset com apenas um capacitor externo conectado a V_{CC}.
10-17	P3.0-P3.7	E/S	O Port 3 é uma interface de E/S bidirecional com 8 bits individualmente endereçáveis e resistores de pull-up internos. Terminais que estejam no estado lógico 1 podem ser utilizados como entradas. O Port 3 também contém os terminais de interrupções, contadores, interface serial e expansão de memória de dados externa*.
18	XTAL2		Saída do amplificador inversor do oscilador.
19	XTAL1		Entrada do amplificador inversor do oscilador e entrada do gerador de clock interno.
20	V_{SS}		Potencial de referência (terra).
21-28	P2.0-P2.7	E/S	O Port 2 é uma interface de E/S bidirecional com 8 bits individualmente endereçáveis e resistores de pull-up internos. Terminais que estejam no estado lógico 1 podem ser utilizados como entradas. O Port 2 também gera a parte mais significativa dos endereços durante acessos às memórias externas de programa ou dados.
29	PSEN\	S	PROGRAM STORE ENABLE Habilita o acesso à memória de programa externa durante a busca de instruções. Permanece em nível lógico 1 durante o acesso da memória de programa interna.
30	ALE	S	ADDRESS LATCH ENABLE Fornece o sinal para armazenamento da parte menos significativa do endereço durante acessos às memórias externas de programa ou dados.
31	EA\	E	EXTERNAL ACCESS Quando em nível lógico 1, as instruções da memória de programa interna são executadas. Quando em nível lógico 0, todas as instruções são buscadas na memória de programa externa. No caso do 8031 este terminal deve sempre estar em nível lógico 0.
32-39	P0.0-P0.7	E/S	O Port 0 é uma interface de E/S bidirecional com 8 bits individualmente endereçáveis em dreno aberto. Terminais que estejam no estado lógico 1 podem ser utilizados como entradas de alta impedância. O Port 0 também atua como barramento de dados e gera de maneira multiplexada a parte menos significativa dos endereços durante acessos às memórias externas de programa ou dados. No caso de acesso a memórias externas são utilizados resistores de pull-up internos.
40	V_{CC}		Potencial de alimentação (+5V).

*Funções Especiais do Port 3

Pino	Nome	Função
P3.0	RXD/data	Receptor da interface serial assíncrona ou entrada e saída de dados da interface serial síncrona.
P3.1	TXD/clock	Transmissor da interface serial assíncrona ou saída de clock da interface serial síncrona.
P3.2	INT0\	Entrada de interrupção externa 0 ou sinal de controle para o contador 0.
P3.3	INT1\	Entrada de interrupção externa 1 ou sinal de controle para o contador 1.
P3.4	T0	Entrada externa para o contador 0.
P3.5	T1	Entrada externa para o contador 1.
P3.6	WR\	Sinal de escrita na memória externa de dados.
P3.7	RD\	Sinal de leitura na memória externa de dados.

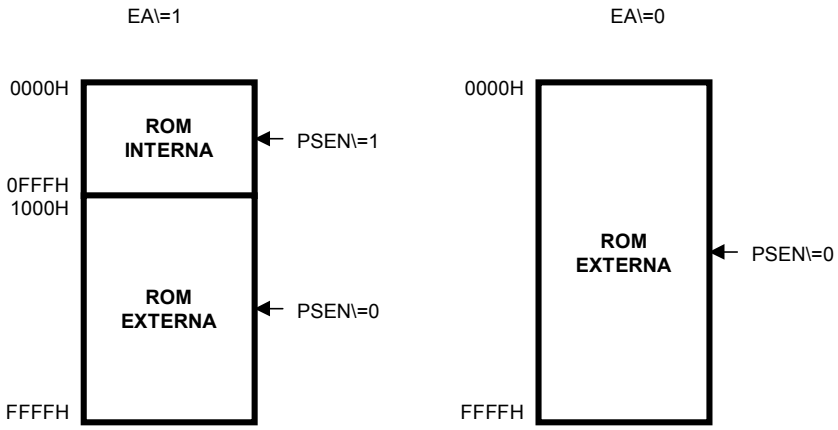
É através das funções especiais dos terminais do Port 3 que se obtém acesso a periféricos internos do microcontrolador (interface serial, contadores de eventos e controlador de interrupções). Os sinais de controle de leitura e escrita da memória de dados externa também são fornecidos através de terminais do Port 3. No entanto, o Port 3 pode ser utilizado apenas como E/S simples.


ARQUITETURA INTERNA:
T1
T0
Temporizadores/ Contadores de Eventos (16 bits)
Interface Serial Programável
TXD
RXD
Memória De Dados Interna (RAM)
Interfaces de Entrada / Saida Programáveis
Ports
Barramento de Dados Interno
Memória De Programa Interna (ROM)
Controle do Barramento de Expansão
WR\
RD\
PSEN\
ALE
EA\
XTAL2
XTAL1
Gerador de Clock
CPU MCS51
Interrupções Internas:
- T/C0
- T/C1
- Serial
INT1\
INT0\
RST

ORGANIZAÇÃO DA MEMÓRIA

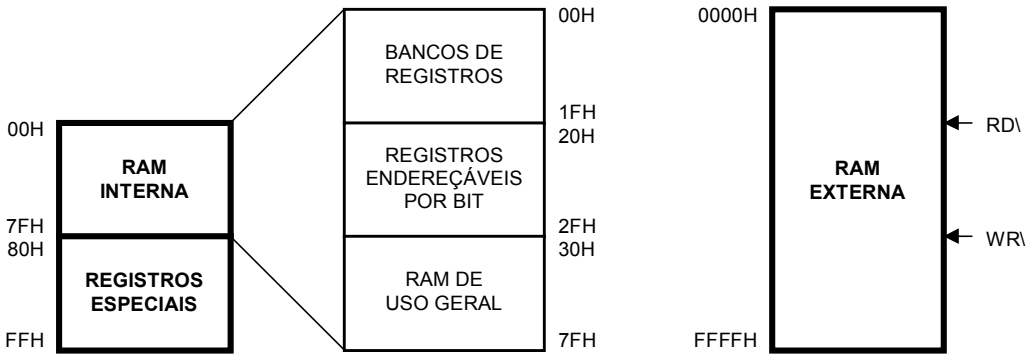
O 8051 acessa as memórias de programa e dados através de sinais de controle diferentes, resultando em mapas de memória separados para programas e dados.

Memória de Programa



A memória de programa pode ser expandida através de barramentos externos. Após o reset, o microcontrolador 8051 irá buscar a primeira instrução no endereço 0000H da memória de programa. O nível lógico presente no terminal EA\ determina se o microcontrolador deverá iniciar a busca das instruções na memória interna ou exclusivamente na memória externa, ignorando a memória interna, se a mesma existir. O sinal PSEN\ habilita o acesso à memória de programa externa.

Memória de Dados



De maneira similar, é possível expandir a memória de dados utilizando barramentos externos. Entretanto, as memórias de dados interna e externa são tratadas pelo 8051 de maneira radicalmente diferente: a interna é acessada através de endereços de 8 bits (MOV) e a externa através de endereços de 16 bits (MOVX). Os sinais RD\ e WR\ são os responsáveis pelos acessos de leitura e escrita, respectivamente, na memória de dados externa.

BANCOS DE REGISTROS

Os microcontroladores da família MCS51 possuem quatro conjuntos de registros, chamados de bancos de registros. Cada banco possui 8 registros, chamados de R0 a R7. Os registros de R0 a R7 normalmente são utilizados como operandos de instruções, armazenando dados temporários na execução do programa.

Pode-se comutar o banco de registros em uso através do estado dos bits de controle RS1 e RS0, existentes no registro de função especial PSW. O uso de diferentes bancos de registros é especialmente útil na implementação de subrotinas e rotinas de atendimento a interrupções, minimizando o uso da pilha para o salvamento de dados.

Quando uma instrução utiliza o modo registro de endereçamento, o endereço físico de memória de dados interna a ser acessado depende do estado dos bits de controle RS1 e RS0, que determinam qual é o banco de registros em uso no momento, como mostra a tabela abaixo.

Banco	PSW:RS1	PSW:RS0	Endereço	Registro
0	0	0	00H	R0
			01H	R1
			02H	R2
			03H	R3
			04H	R4
			05H	R5
			06H	R6
			07H	R7
1	0	1	08H	R0’
			09H	R1’
			0AH	R2’
			0BH	R3’
			0CH	R4’
			0DH	R5’
			0EH	R6’
			0FH	R7’
2	1	0	10H	R0”
			11H	R1”
			12H	R2”
			13H	R3”
			14H	R4”
			15H	R5”
			16H	R6”
			17H	R7”
3	1	1	18H	R0\
			19H	R1\
			1AH	R2\
			1BH	R3\
			1CH	R4\
			1DH	R5\
			1EH	R6\
			1FH	R7\

REGISTROS BIT-ENDEREÇÁVEIS

Nos microcontroladores da família MCS51, a faixa que vai do endereço 20H ao endereço 2FH da memória de dados interna é chamada de região de registros bit-enderecáveis. Estas posições da memória de dados interna possuem uma característica especial que permite a alteração individual de cada bit através de instruções especialmente destinadas a este fim. Desta maneira, cada bit endereçável possui um endereço individual específico, como mostra a tabela abaixo.

	Endereços Individuais dos Bits							
Registro	**bit 7**	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
20H	07H	06H	05H	04H	03H	02H	01H	00H
21H	0FH	0EH	0DH	0CH	0BH	0AH	09H	08H
22H	17H	16H	15H	14H	13H	12H	11H	10H
23H	1FH	1EH	1DH	1CH	1BH	1AH	19H	18H
24H	27H	26H	25H	24H	23H	22H	21H	20H
25H	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
26H	37H	36H	35H	34H	33H	32H	31H	30H
27H	3FH	3EH	3DH	3CH	3BH	3AH	39H	38H
28H	47H	46H	45H	44H	43H	42H	41H	40H
29H	4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
2AH	57H	56H	55H	54H	53H	52H	51H	50H
2BH	5FH	5EH	5DH	4CH	5BH	5AH	59H	58H
2CH	67H	66H	65H	64H	63H	62H	61H	60H
2DH	6FH	6EH	6DH	6CH	6BH	6AH	69H	68H
2EH	77H	76H	75H	74H	73H	72H	71H	70H
2FH	7FH	7EH	7DH	7CH	7BH	7AH	79H	78H

REGISTROS DE FUNÇÃO ESPECIAL

Os registros de função especial (special function registers) são os responsáveis pelo controle e execução das instruções do programa e pela configuração dos periféricos internos, determinando as suas formas de operação. O 8051 possui os seguintes registros de função especial (os endereços na memória de dados interna encontram-se entre parêntesis):

- **PC – Program Counter**

Contador de Programa: indica o endereço da próxima instrução a ser executada (16 bits).

- **PSW (D0H) – Program Status Word**

Contém os sinalizadores (flags) que indicam as ocorrências na execução da última operação lógica ou aritmética. Contém também os bits de controle para a seleção do banco de registros em uso.

- **SP (81H) – Stack Pointer**

Ponteiro da Pilha: indica o topo da pilha (último dado colocado).

- **A (E0H) e B (F0H)**

Trata-se do acumulador (A), empregado nas operações lógicas e aritméticas da CPU, e de um registro secundário (B), empregado apenas nas operações de multiplicação e divisão. São registros intimamente relacionados com a Unidade Lógica e Aritmética da CPU.

- **DPH (83H) e DPL (82H)**

Registros de 8 bits que compõem respectivamente os bytes mais e menos significativos do ponteiro de dados de 16 bits chamado DPTR, utilizado para endereçamento indireto da memória de programa e da memória externa de dados.

- **P0 (80H), P1 (90H), P2(A0H) e P3(B0H)**

Registros que contêm cópias dos estados dos quatro Ports de E/S. A escrita nesses registros altera automaticamente o conteúdo na saída do Port correspondente. A leitura carrega o estado de entrada presente nos terminais do Port no registro correspondente.

- **IE (A8H) – Interrupt Enable e IP (B8H) – Interrupt Priority**

Registros de habilitação / desabilitação das interrupções e de definição da prioridade de atendimento de cada uma delas.

- **PCON (87H) – Power Control**

Presente apenas na versão CMOS, este registro permite colocar o 80C51 em modos de redução de consumo de energia, preservando o conteúdo da memória interna.

- **TCON (88H) – Timer Control e TMOD (89H) – Timer Mode**

Registros de controle e modo de operação dos temporizadores / contadores de eventos.

- **TH1 (8DH), TL1 (8BH), TH0 (8CH) e TL0 (8AH)**

Registros de dados dos temporizadores / contadores de eventos 1 e 0, respectivamente. Contêm os valores das contagens realizadas (16 bits).

- **SCON (98H) – Serial Control e SBUF (99H) – Serial Buffer**

Registros de controle da interface serial e de armazenamento dos dados a serem transmitidos (escrita) ou recebidos (leitura).

Registros de Função Especial Bit-Endereçáveis

Da mesma forma que ocorre na região de registros bit-endereçáveis, existem alguns registros de função especial que também possuem bits individualmente endereçáveis, como mostra a tabela abaixo.

	Endereços Individuais dos Bits							
Registro	**bit 7**	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
P0	87H	86H	85H	84H	83H	82H	81H	80H
P1	97H	96H	95H	94H	93H	92H	91H	90H
P2	A7H	A6H	A5H	A4H	A3H	A2H	A1H	A0H
P3	B7H	B6H	B5H	B4H	B3H	B2H	B1H	B0H
A	E7H	E6H	E5H	E4H	E3H	E2H	E1H	E0H
B	F7H	F6H	F5H	F4H	F3H	F2H	F1H	F0H
TCON	**TF1**	**TR1**	**TF0**	**TR0**	**IE1**	**IT1**	**IE0**	**IT0**
	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
SCON	**SM1**	**SM2**	**SM3**	**REN**	**TB8**	**RB8**	**T1**	**R1**
	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
IE	**EA**			**ES**	**ET1**	**EX1**	**ET0**	**EX0**
	AFH			ACH	ABH	AAH	A9H	A8H
IP				**PS**	**PT1**	**PX1**	**PT0**	**PX0**
				BCH	BBH	BAH	B9H	B8H
PSW	**CY**	**AC**	**F0**	**RS1**	**RS0**	**OV**		**P**
	D7H	D6H	D5H	D4H	D3H	D2H		D0H

CLOCK

O 8051 possui um oscilador interno destinado a gerar o sinal de clock do sistema. Pode-se fazer uso de um cristal oscilador na freqüência de operação desejada, conectado-o aos terminais XTAL1 e XTAL2, juntamente com dois capacitores de realimentação conectados ao terra do circuito.

Para fazer uso de um circuito de clock externo ao 8051, basta conectar o terminal XTAL1 ao potencial de terra do circuito (GND) e aplicar o sinal externo ao terminal XTAL2. Dessa maneira o sinal de clock externo irá diretamente ao circuito de temporização e controle do microcontrolador.

A freqüência de operação do 8051 pode chegar a um máximo de 8, 10 ou 12MHz conforme o modelo utilizado, embora existam atualmente também versões que operam em até 24MHz (89C51) e 33MHz (80C320).

A freqüência de clock é dividida internamente por 12 para a geração dos ciclos de máquina necessários à execução das instruções. Sendo assim, os ciclos de máquina do 8051 têm a duração de 12 ciclos de clock, embora em algumas versões otimizadas (80C320) os ciclos de máquina durem apenas 4 ciclos de clock, sendo portanto 3 vezes mais velozes que a versão original.

RESET

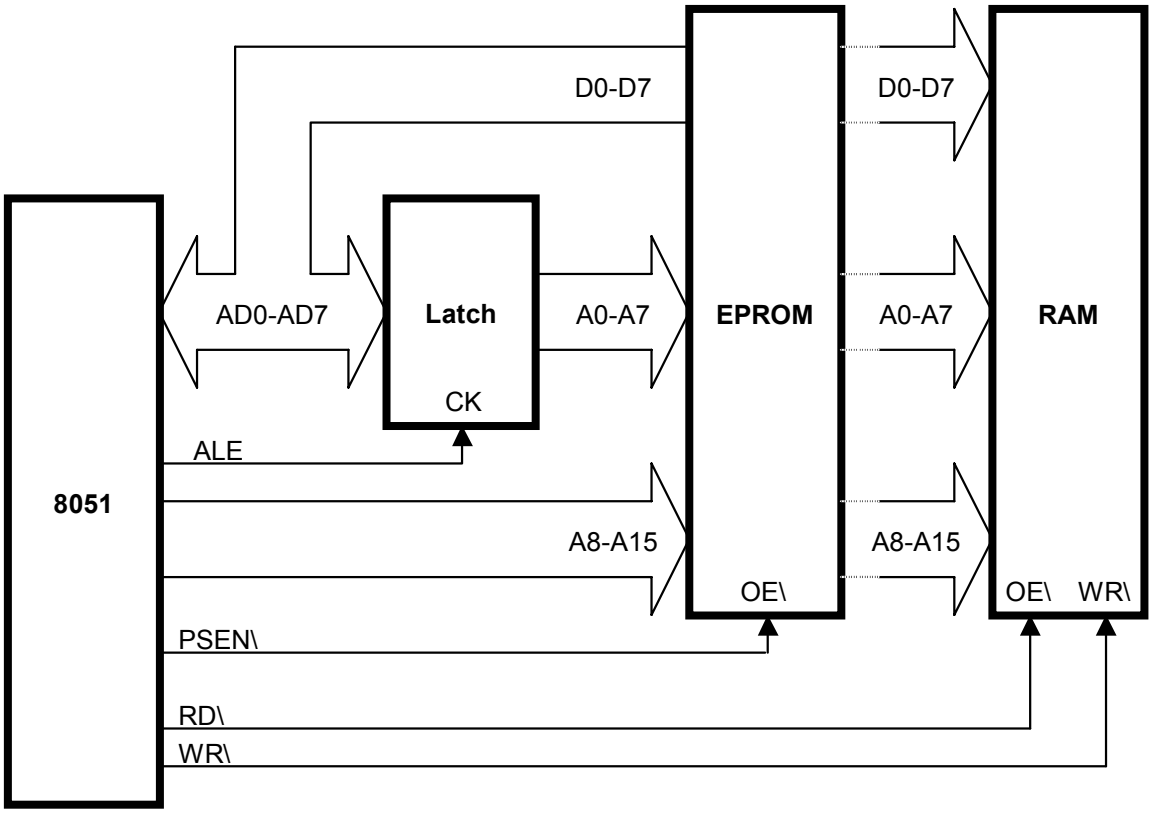
O reset do 8051 é ativado quando o terminal RST é levado a nível lógico 1 por dois ou mais ciclos de máquina. Consiste basicamente na inicialização de alguns registros com valores predeterminados:

- Os registros A, B, PSW, DPTR, PC e todos os registros dos temporizadores / contadores de eventos são zerados.
- O registro SP é carregado com o valor 07H.
- Os Ports P0, P1, P2 e P3 são carregados com FFH. Durante o reset o nível lógico dos terminais é indeterminado, assumindo valor 1 após a execução da seqüência de reset interna.
- O registro SCON é zerado e o registro SBUF fica com valor indeterminado.
- O registro PCON tem apenas o seu bit mais significativo zerado.
- Os registros IE e IP são carregados com o valor XXX00000B, (X=indeterminado).
- A RAM interna não tem o seu conteúdo afetado pelo reset.

Um circuito reset automático ao ligar o sistema (power-on-reset) pode ser implementado com a conexão de um capacitor externo entre o potencial de alimentação e o terminal RST.

MEMÓRIAS EXTERNAS DE PROGRAMA E DE DADOS

Os Ports 0 e 2 podem ser utilizados para acessar memórias externas como função alternativa, possibilitando a expansão da capacidade de memória de programa e de dados do 8051. Nesse modo de operação, o Port 2 atua como a parte mais significativa (A15-A8) e o Port 0 atua como a parte menos significativa (A7-A0) do barramento de endereços externo. Os sinais do barramento de dados externo (D7-D0) também são fornecidos em seguida pelo Port 0, multiplexados com a parte menos significativa dos endereços.



O sinal ALE indica quando a parte menos significativa dos endereços está disponível no Port 0. Este sinal serve para habilitar um latch de 8 bits externo, destinado a armazenar a parte menos significativa do barramento de endereços (A7-A0). Após o armazenamento dos sinais A7-A0, o Port 0 apresenta os sinais D7-D0. É desta forma que ocorre a demultiplexação do barramentos de dados e endereços.

O sinal PSEN\ habilita a leitura da memória de programa externa, tornando-se ativo quando o 8051 busca instruções externamente. O terminal EA\ quando em nível lógico 1 informa ao 8051 que os primeiros 4KB da memória de programa estão na memória de programa interna e o restante na memória externa, se estiver presente. Quando em nível lógico 0, o terminal EA\ determina que o microcontrolador busque todas as instruções de programa na memória externa, ignorando completamente a memória interna, se esta existir.

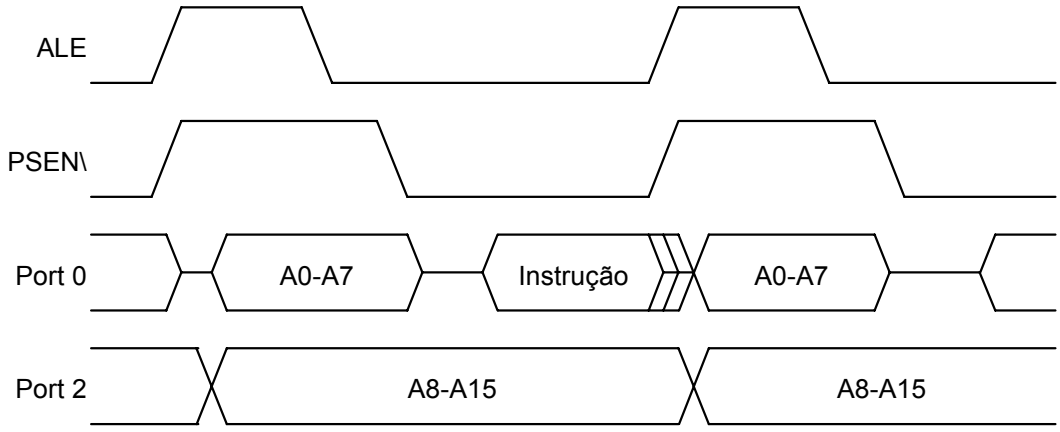
Os sinais RD\ e WR\ somente são ativados no caso de leitura ou escrita, respectivamente, na memória de dados externa (as instruções que acessam a memória de dados externa – MOVX – são distintas das que acessam a memória interna – MOV).

Os sinais RD\, WR\ e PSEN\ nunca são ativados simultaneamente, conforme mostra a tabela abaixo.

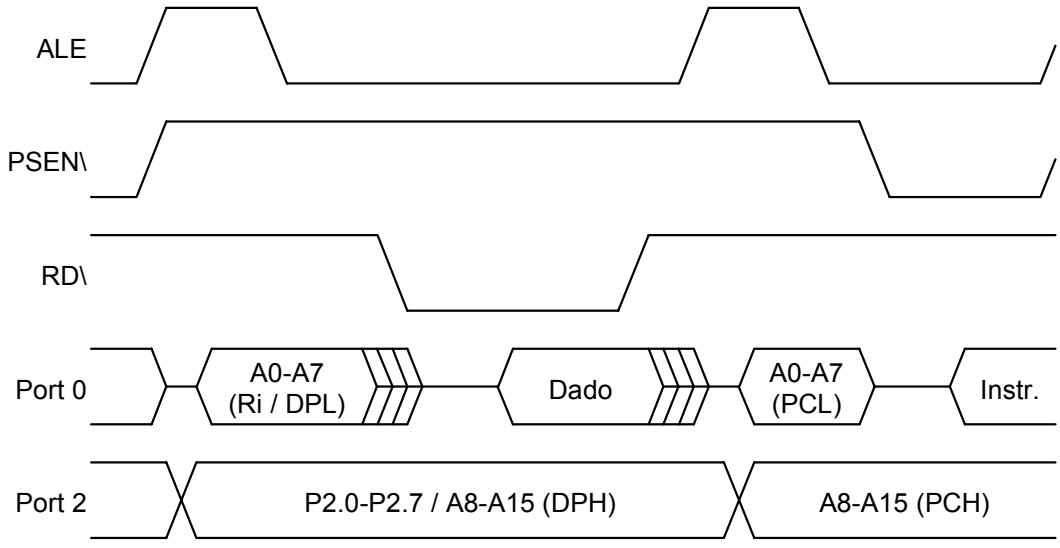
PSEN\	RD\	WR\	Função
1	1	1	Nenhum Acesso Externo
1	1	0	Escrita na Memória de Dados
1	0	1	Leitura da Memória de Dados
0	1	1	Leitura da Memória de Programa

Os sinais de acesso às memórias externas possuem comportamentos bem definidos, apresentados nos diagramas de tempo a seguir.

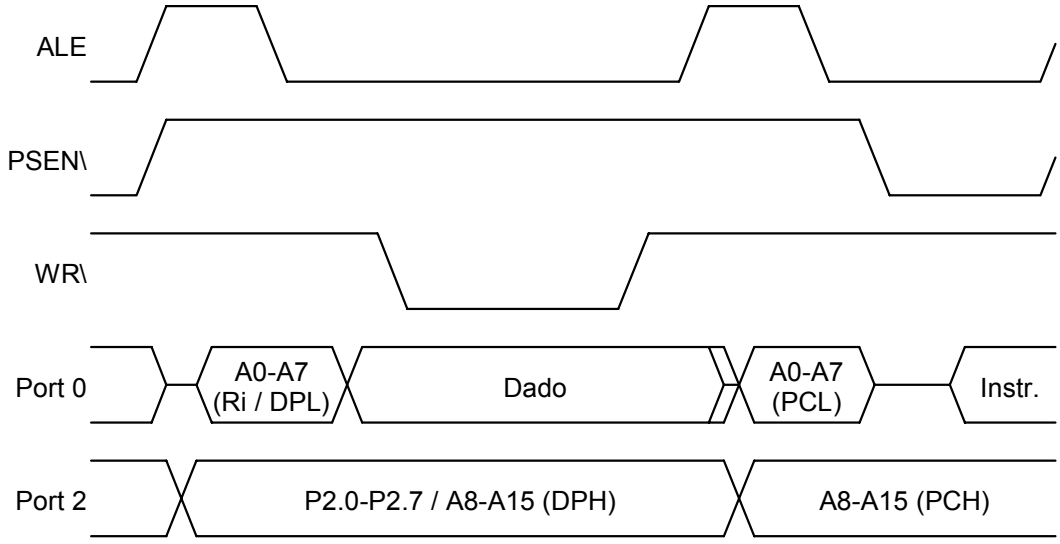
Ciclo de Leitura na Memória de Programa Externa



Ciclo de Leitura na Memória de Dados Externa

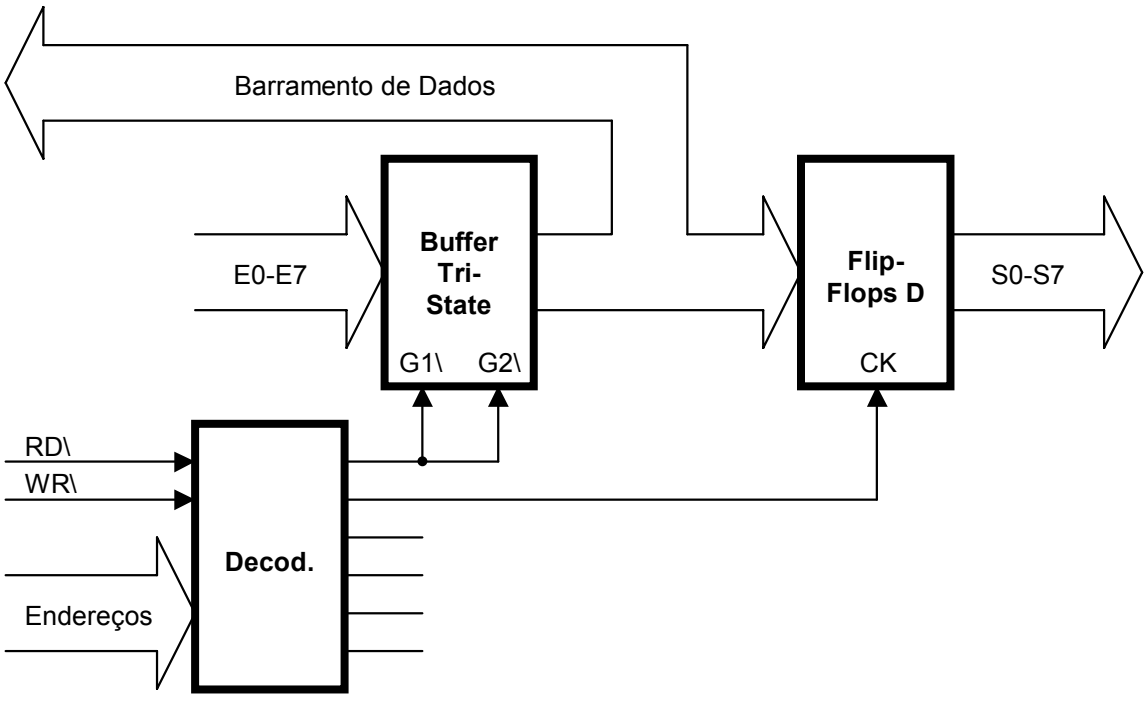


Ciclo de Escrita na Memória de Dados Externa



Expansão das Interfaces de E/S

Pode-se empregar uma técnica conhecida como E/S mapeada em memória (memory mapped I/O) para se expandir as capacidades de E/S do sistema microcontrolado. O método consiste em destinar uma fração da faixa de endereços da memória de dados externa para o acesso a periféricos externos. Sob o ponto de vista do software os periféricos externos são tratados como se fossem posições da memória de dados externa, utilizando as mesmas instruções de acesso (MOVX). A desvantagem está no aumento considerável do tamanho do sistema devido à necessidade componentes externos. Além disso, o acesso a periféricos externos consome o dobro do tempo de acesso aos Ports do microcontrolador.



Nas páginas a seguir são apresentados alguns diagramas esquemáticos de possíveis sistemas baseados em microcontroladores da família MCS51.

O primeiro deles (Sistema Mínimo) utiliza a menor quantidade possível de componentes externos, apenas para implementar os circuitos de clock e power-on-reset. Neste caso, todos os pinos de E/S ficam disponíveis para interconexão com outros dispositivos externos não representados no diagrama.

Em seguida são apresentados diagramas esquemáticos de sistemas que fazem uso de memórias externas. São apresentados sistemas com apenas memória de programa externa (Sistema com ROM Externa), apenas memória de dados externa (Sistema com RAM Externa) e, finalmente, com memórias de programa e de dados externas (Sistema com ROM e RAM Externas). Os Ports 0 e 2 ficam comprometidos com a implementação dos barramentos externos de dados e de endereços. O Port 3 fica parcialmente comprometido com a implementação dos sinais de acesso à memória de dados externa, quando esta for utilizada. Deve-se observar a presença do latch para demultiplexação dos barramentos externos e a forma de conexão dos sinais do barramento de controle (ALE, EA\, PSEN\, RD\ e WR\).

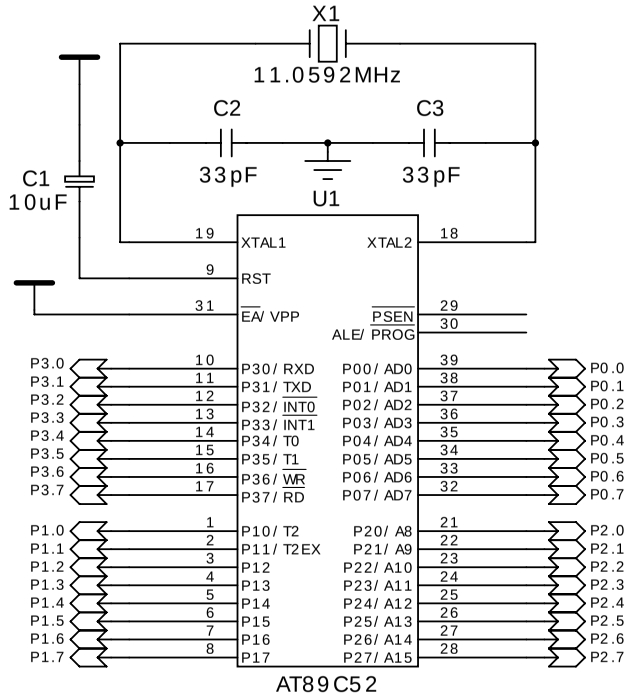
O diagrama esquemático de um sistema que utiliza a técnica de E/S mapeada em memória (Sistema com E/S Mapeada em Memória) também é apresentado. Deve-se notar a presença de um decodificador de endereços para gerar os sinais de habilitação dos dispositivos de E/S, que neste caso são apenas flip-flops D (saídas) e buffers tri-state (entradas).

Por fim, é apresentado o diagrama esquemático de um sistema com memórias externas de programa e de dados, E/S mapeada em memória (interface para display inteligente de cristal líquido) e interface serial padrão RS-232. O Sistema Completo apresentado é bastante similar à placa P51 (ver anexos) e possui algumas cacterísticas especiais quanto à forma de acesso às memórias externas, possibilitando o seu uso com o programa monitor PAULMON (ver anexos).

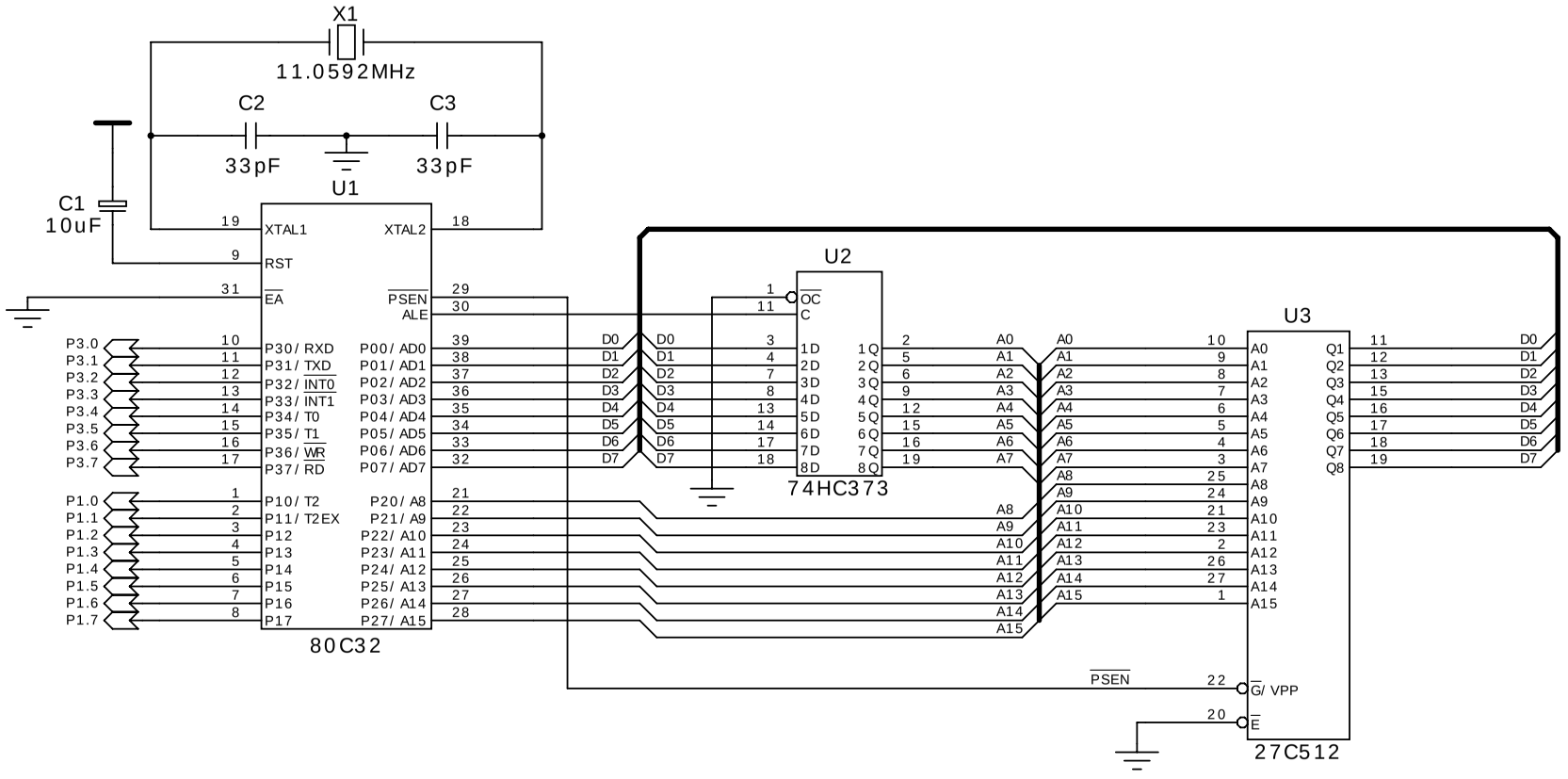
Programas monitores como o PAULMON permitem que se armazene e execute programas na memória RAM externa do sistema. Entretanto, para que isso seja possível, o hardware deve prever alguma forma de acesso à RAM externa como se fosse memória de programa. No diagrama esquemático do Sistema Completo o sinal de leitura da RAM externa é gerado através de uma operação lógica “E” entre os sinais PSEN\ e RD\ (componente U7:A). Desta forma, a RAM externa é acessada tanto como memória de dados externa (quando o sinal RD\ estiver ativo) quanto como memória de programa externa (quando o sinal PSEN\ estiver ativo). Esta característica faz com que os mapas de memória de programa e de dados se sobreponham, sendo necessário, portanto, um decodificador de endereços para atribuir faixas de endereços distintas para a habilitação da ROM e da RAM externas (decodificador composto pelos componentes U3 e U7:B,C,D).

O mesmo decodificador de endereços gera sinais de habilitação para possíveis periféricos mapeados em memória, bem como o sinal de habilitação para um display inteligente de cristal líquido (CN9) mapeado em memória (componentes U3 e U8:A,B,C,D).

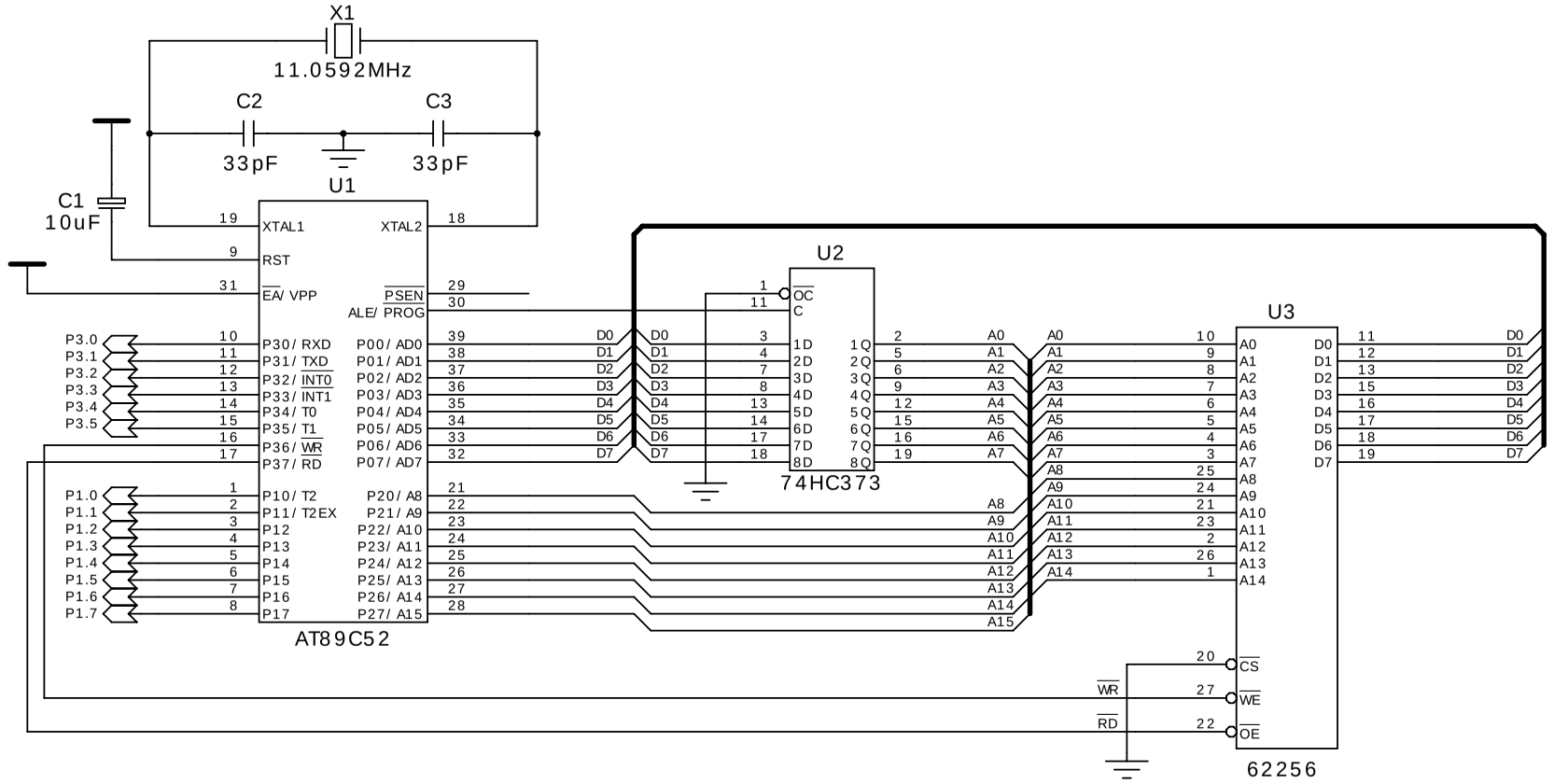
O Sistema Completo apresentado possui diversos jumpers (JP1 a JP10) para a configuração do sistema (ver legenda no canto inferior esquerdo do diagrama esquemático).



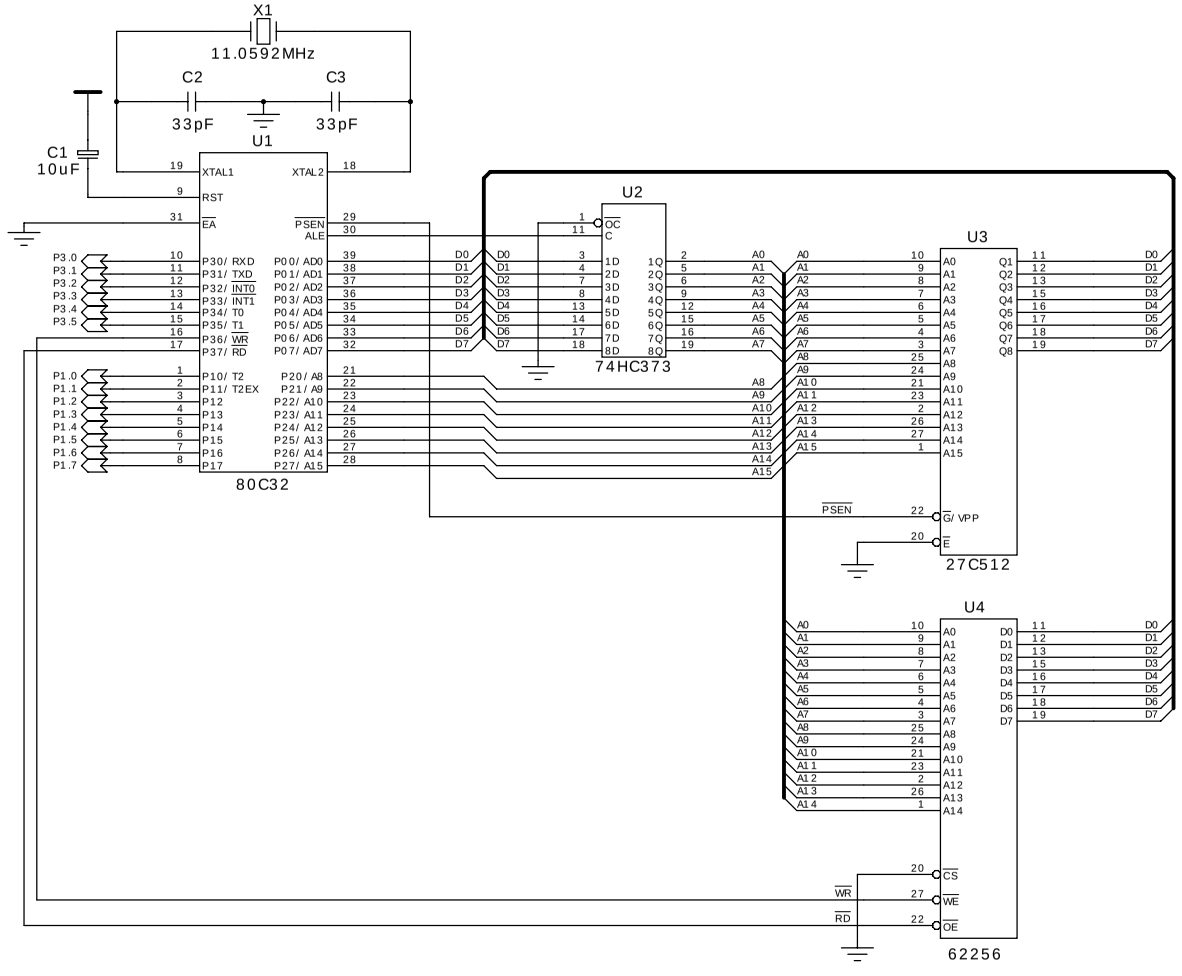
Title	SISTEMA MINIMO	
Size A4	Number	Rev 1.0
Date 03/2002		Drawn by HUGO
Filename MINIMO51.S01		Sheet 1 of 1



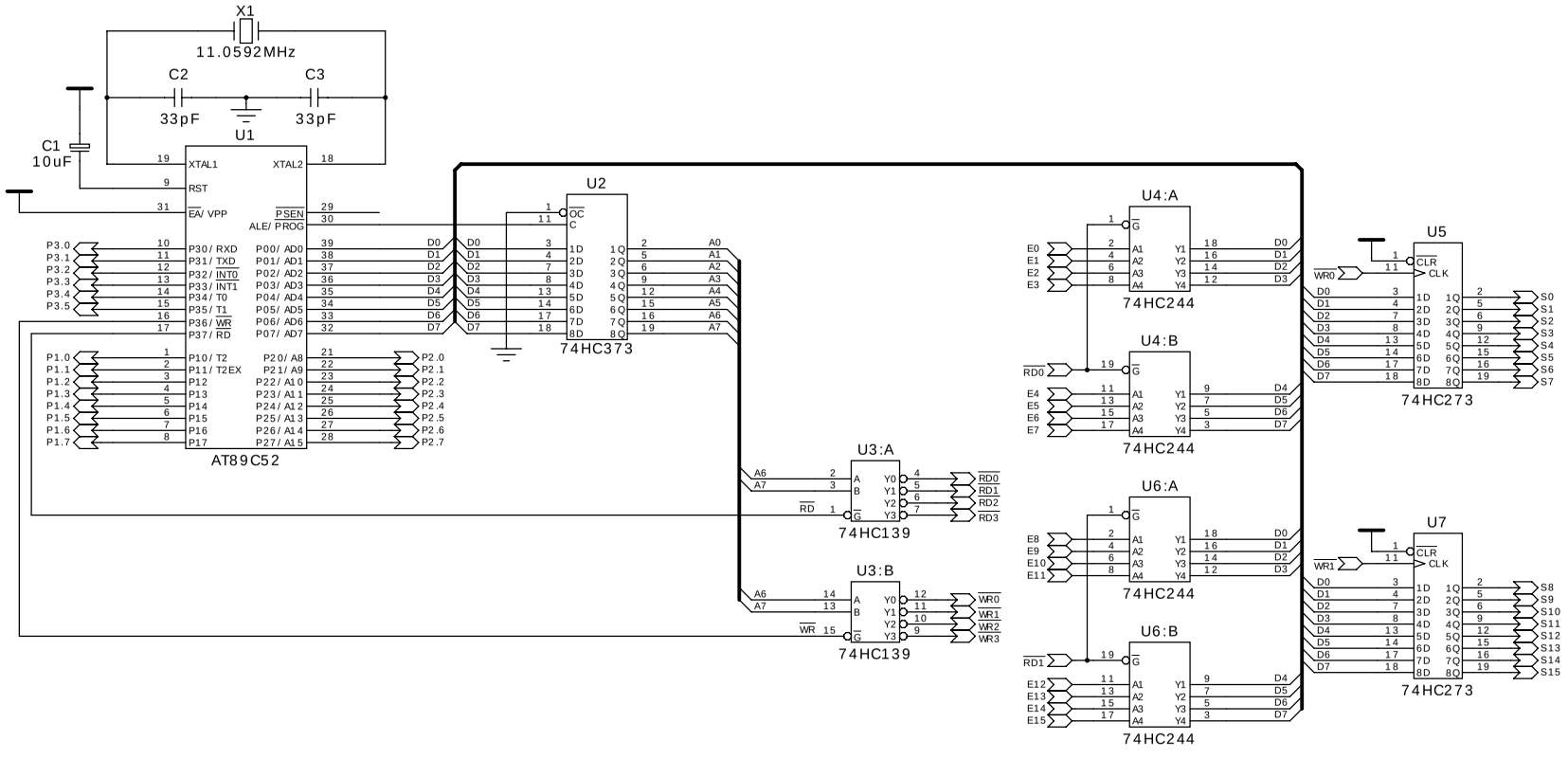
Title	SISTEMA COM ROM EXTERNA	
Size A4	Number	Rev 1.0
Date 03/ 2002		Drawn by HUGO
Filename ROMEXT51.S01		Sheet 1 of 1



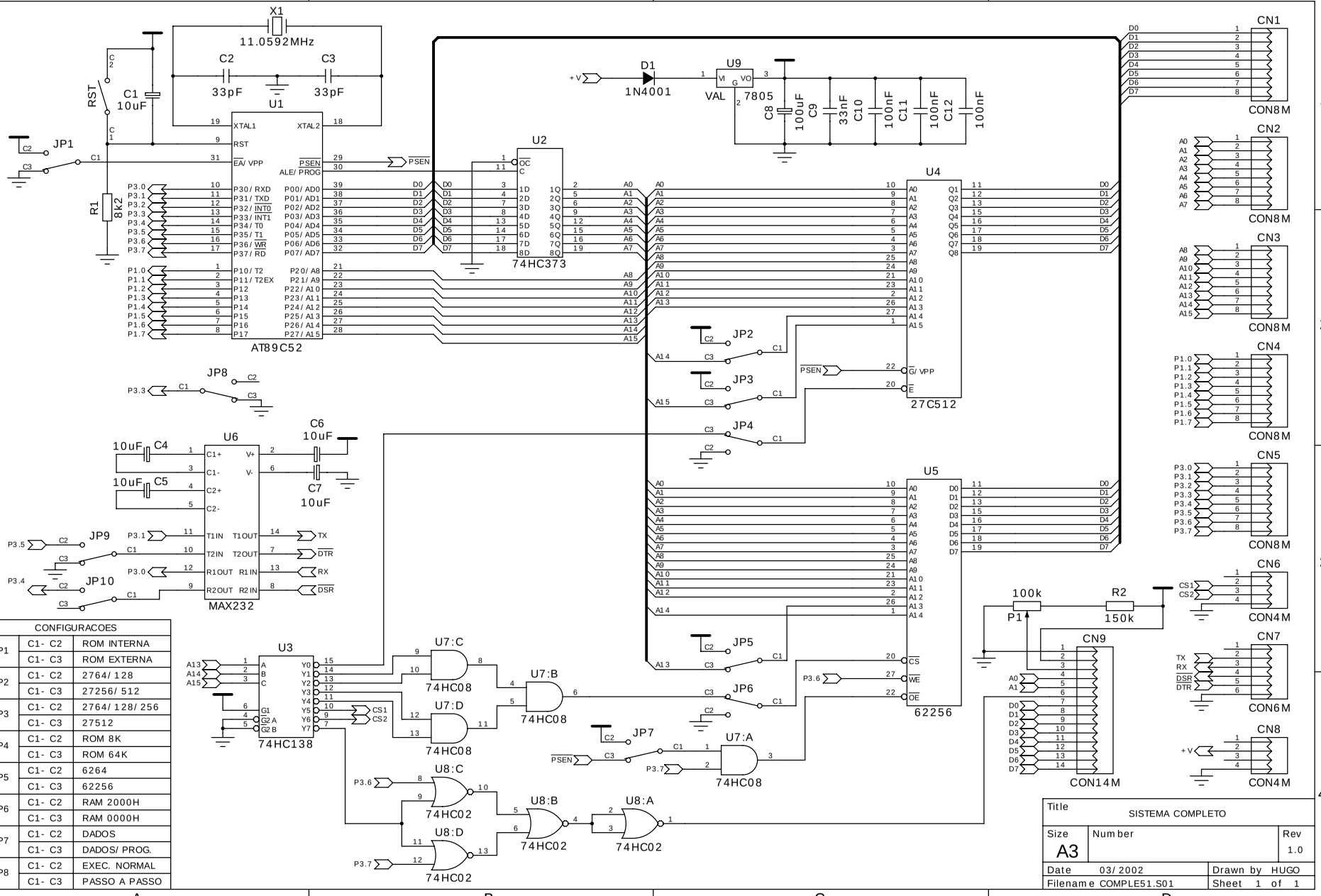
Title	SISTEMA COM RAM EXTERNA	
Size A4	Number	Rev 1.0
Date 03/2002		Drawn by HUGO
Filename RAMEXT51.S01		Sheet 1 of 1



Title	SISTEMA COM ROM E RAM EXTERNAS	
Size A3	Number	Rev 1.0
Date 03/2002		Drawn by HUGO
Filename ROMRAM51.S01		Sheet 1 of 1



Title	SISTEMA COM E/S MAPEADA EM MEMORIA	
Size A3	Number	Rev 1.0
Date 03/2002		Drawn by HUGO
Filename ESMEM51.S01		Sheet 1 of 1



CONFIGURACOES		
JP1	C1- C2	ROM INTERNA
	C1- C3	ROM EXTERNA
JP2	C1- C2	2764/ 128
	C1- C3	27256/ 512
JP3	C1- C2	2764/ 128/ 256
	C1- C3	27512
JP4	C1- C2	ROM 8K
	C1- C3	ROM 64K
JP5	C1- C2	6264
	C1- C3	62256
JP6	C1- C2	RAM 2000H
	C1- C3	RAM 0000H
JP7	C1- C2	DADOS
	C1- C3	DADOS/ PROG.
JP8	C1- C2	EXEC. NORMAL
	C1- C3	PASSO A PASSO

Title	SISTEMA COMPLETO	
Size A3	Number	Rev 1.0
Date 03/ 2002		Drawn by HUGO
Filename COMPLE51.S01		Sheet 1 of 1

EXERCÍCIOS

1) Com base nos diagramas esquemáticos dos sistemas apresentados, responda as seguintes questões:

a) Explique com suas palavras o que ocorreria com o "Sistema Mínimo" se o terminal EA\ do 89C52 fosse conectado a GND em vez de Vcc.

b) Por que o terminal EA\ do 80C32 do "Sistema com ROM Externa" não pode ser conectado a Vcc? E se o 80C32 fosse substituído por um 89C52, seria possível conectar o terminal EA\ a Vcc? Por que?

c) Para que o "Sistema com RAM Externa" funcione corretamente, em qual componente do sistema o programa a ser executado deverá estar gravado? Por que?

d) Determine quantos bytes de memória de programa interna e externa o "Sistema com ROM e RAM Externas" possui. Determine também quantos bytes de memória de dados interna e externa este sistema possui.

e) Determine em que faixa de endereços estão localizados os periféricos representados pelos componentes U4, U5, U6 e U7 no "Sistema com E/S Mapeada em Memória". Quais são os componentes que implementam entradas? E quais são os componentes que implementam saídas? Como seria possível ampliar o número de entradas e saídas mapeadas em memória?

f) Explique como fica o mapa de memória do "Sistema Completo" quando os jumpers de JP1 a JP7 são colocados nas posições C1-C3. E como fica o mapa de memória deste sistema quando os jumpers JP1, JP4, JP6 e JP7 são colocados nas posições C1-C2 (com os jumpers JP2, JP3 e JP5 nas posições C1-C3).

g) Esquematize uma forma de se mapear em memória um conversor A/D ADC0820 no "Sistema Completo". Procure fazer uso ao máximo dos recursos já disponíveis (barramentos externos, decodificador de endereços, etc.). Explique em que faixa de endereços ocorreria o acesso a este conversor A/D.

2) Os seguintes componentes estão disponíveis para projetos de sistemas de hardware: microcontroladores 80C31, 80C32, 89C51 e 89C52; memórias EPROM 27C64, 27C128, 27C256 e 27C512; memórias RAM 6116, 6264 e 62256; e ainda toda a família HC de circuitos integrados TTL. Elabore os seguintes projetos teóricos de hardware, visando o uso da menor quantidade de componentes possível:

a) Projete o hardware de um sistema baseado no microcontrolador 80C32 com 8KB de memória de programa e 32KB de memória de dados. O sistema deve incluir os circuitos de power-on-reset e clock.

b) O levantamento dos requisitos necessários para determinado sistema baseado na arquitetura MCS51 revelou a necessidade de pelo menos 6KB de memória de programa, pelo menos 200 bytes de memória de dados e pelo menos 12 bits de E/S. Com base nessas informações, projete o hardware para tal sistema.

c) Deseja-se um sistema baseado no microcontrolador 89C51 que possua exatamente 2KB de memória de dados externa e pelo menos 6 bits de entrada e 8 bits de saída a mais que os bits disponíveis nos Ports de E/S. Projete o hardware de um sistema que atenda a esses requisitos, utilizando a técnica de E/S mapeada em memória para a implementação dos bits de E/S adicionais.

PROGRAMAÇÃO

O 8051 utiliza códigos de instrução de 8 bits. Entretanto, suas instruções podem ter extensão variável entre um e três bytes.

As instruções de apenas um byte de extensão não requerem operandos ou então já os possuem embutidos nos 8 bits do código da instrução. As instruções de dois ou três bytes de extensão são as que possuem um ou dois bytes adicionais, correspondentes ao(s) operando(s).

Para cada instrução o tempo de execução pode ser de um a quatro ciclos de máquina (cada ciclo de máquina corresponde a 12 ciclos de clock), conforme a complexidade da operação a ser executada e o número de bytes extras a serem buscados na memória de programa.

Na programação em linguagem Assembly (baixo nível) são utilizados símbolos mnemônicos para representar os códigos das instruções, facilitando o entendimento por parte do programador, evitando que seja necessário decorar os códigos de cada instrução. A codificação do programa propriamente dita é realizada por um aplicativo montador (assembler).

No caso do uso de uma linguagem de programação de alto nível como a linguagem C utiliza-se um compilador, que é o responsável pela tradução para o Assembly e posterior montagem do código.

MODOS DE ENDEREÇAMENTO

- **Modo Imediato**

Permite carregar constantes em posições da memória de dados interna. Nas instruções da família MCS51, as constantes são precedidas pelo símbolo #.

- **Modo Direto**

Essa forma de endereçamento utiliza diretamente o endereço da posição da memória de dados interna na qual será efetuada a operação.

- **Modo Indireto**

Nesse caso o endereço da posição de memória onde será efetuada a operação é indicado de forma indireta pelos registros R0 ou R1 do banco de registros em uso ou pelo registro DPTR. Os registros R0, R1 e DPTR atuam como ponteiros nessas instruções, sendo R0 e R1 ponteiros para endereços de 8 bits e DPTR um ponteiro para endereços de 16 bits. Nas instruções da família MCS51, os ponteiros são precedidos pelo símbolo @.

- **Modo Registro**

Permite o acesso aos registros R0 a R7 do banco de registros em uso (working registers), definido pelos bits de controle RS1 e RS0 existentes no registro de função especial PSW.

CONJUNTO DE INSTRUÇÕES

O conjunto das instruções do 8051 pode ser dividido em 5 grupos: transferência de dados, operações lógicas, operações aritméticas, manipulação de variáveis booleanas e controle de programa. A seguir serão analisadas algumas das instruções de cada grupo.

Transferência de Dados

Fazem parte do grupo de transferência de dados as instruções `MOV`, `MOVC`, `MOVX`, `PUSH`, `POP`, `XCH` e `XCHD`.

A instrução `MOV` copia o conteúdo de um endereço da memória de dados interna (ou uma constante) para outro endereço da memória de dados interna, utilizando a seguinte sintaxe:

`MOV <destino>, <fonte>`

O conteúdo do operando `<fonte>` é copiado para o operando `<destino>`, sendo possível utilizar todos os modos de endereçamento (imediato, direto, indireto e registro).

- `MOV A, #dado`

  Carrega no acumulador o valor da constante de 8 bits (dado). O valor da constante pode estar em decimal (sem sufixo), hexadecimal (sufixo H) ou binário (sufixo B).

  Exemplo: `MOV A, #01010101B`

- `MOV A, direto`

  Carrega no acumulador o conteúdo do endereço da memória de dados interna (direto).

  Exemplo: `MOV A, 12H`

- `MOV A, @Ri`

  Carrega no acumulador o conteúdo do endereço da memória de dados interna apontado pelo registro R0 ou R1 (@Ri).

  Exemplo: `MOV A, @R0`

- `MOV A, Rn`

  Carrega no acumulador o conteúdo do registro R0, R1, R2, R3, R4, R5, R6 ou R7 (Rn) do banco de registros em uso.

  Exemplo: `MOV A, R0`

- `MOV DPTR, #dado16`

  Carrega no registro DPTR o valor da constante de 16 bits (dado16).

  Exemplo: `MOV DPTR, #1234H`

A instrução MOVC copia o conteúdo da memória de programa (interna ou externa) para o acumulador, utilizando a seguinte sintaxe:

MOVC A, <fonte>

O operando <fonte> sempre irá empregar o modo de endereçamento indireto através de ponteiros. Trata-se da única forma de leitura de constantes gravadas na memória de programa, como por exemplo, tabelas de dados (look-up tables).

- MOVC A, @A+DPTR

  Adiciona o valor contido no acumulador ao valor contido no registro de DPTR, utilizando o resultado como ponteiro para endereçar uma posição da memória de programa e carrega o seu conteúdo no acumulador.

A instrução MOVX copia o conteúdo de uma posição da memória de dados externa para o acumulador ou vice-versa, utilizando as seguintes sintaxes:

MOVX A, <fonte>
MOVX <destino>, A

Os operandos <fonte> e <destino> sempre irão utilizar o modo de endereçamento indireto através de ponteiros. Trata-se da única forma de leitura ou escrita na memória de dados externa e de periféricos externos mapeados em memória.

- MOVX A, @DPTR

  Carrega o acumulador com o conteúdo da posição de memória de dados externa apontada pelo registro DPTR.

- MOVX @DPTR, A

  Carrega a posição de memória de dados externa apontada pelo registro DPTR com o conteúdo do acumulador.

As instruções PUSH e POP trabalham com o armazenamento temporário de dados, utilizando a seguinte sintaxe:

PUSH <fonte>
POP <destino>

É possível utilizar apenas o modo de endereçamento direto.

- PUSH direto

  O registro SP é incrementado e em seguida o conteúdo do endereço da memória de dados interna (direto) é carregado no topo da pilha.

  Exemplo: PUSH ACC

- `POP direto`

  O topo da pilha é carregado no endereço da memória de dados interna (direto). Em seguida o registro SP é decrementado.

  Exemplo: `POP ACC`

A instrução `XCH` efetua a troca de conteúdos entre o acumulador e um segundo operando, utilizando a seguinte sintaxe:

XCH A, <operando>

É possível utilizar todos os modos de endereçamento (imediato, direto, indireto e registro).

- `XCH A, direto`

  Troca os conteúdos do acumulador e do endereço da memória de dados interna (direto).

  Exemplo: `XCH A, 34H`

A instrução `XCHD` efetua a troca dos nibbles menos significativos entre o acumulador e um segundo operando, utilizando a seguinte sintaxe:

XCHD A, <operando>

É possível utilizar apenas o modo de endereçamento indireto com ponteiros de 8 bits (R0 ou R1).

- `XCHD A, @Ri`

  Troca apenas o nibble menos significativo dos conteúdos do acumulador e do endereço da memória de dados interna apontada pelo registro R0 ou R1 (@Ri).

  Exemplo: `XCHD A, @R0`

Para se ter noção de todas as instruções de transferência de dados existentes para a família MCS51 deve-se consultar a tabela de instruções completa que se encontra nos anexos.

Exercícios

1) Sabendo que a instrução `MOVC` envolve o acesso à memória de programa do microcontrolador 8051, explique o que ocorre com os barramentos externos durante a execução da instrução `MOVC A, @A+DPTR`. Leve em consideração os casos de acesso a endereços existentes na memória de programa interna e externa.

2) Sabendo que a instrução `MOVX` envolve o acesso à memória de dados externa do microcontrolador 8051, explique o que ocorre com os barramentos externos durante a execução das instruções `MOVX A, @DPTR` e `MOVX @DPTR, A`. Qual instrução realiza uma escrita e qual instrução realiza uma leitura na memória de dados externa?

3) Execute o programa dado a seguir no simulador do ambiente Proview e tire conclusões sobre o efeito de cada instrução executada. Para uma melhor noção do que ocorre, sugere-se que o programa seja executado passo a passo, visualizando-se as janelas “Main Registers” e “Data View”.

```
INICIO:   MOV  SP, #20H
          MOV  DPTR, #1234H
          MOV  A, #56
          MOV  01H, #02H
          MOV  R0, A
          MOV  @R1, A
          PUSH DPH
          PUSH DPL
          PUSH ACC
          PUSH PSW
          MOV  PSW, #00011000B
          MOV  A, #78
          MOV  R0, A
          MOV  @R1, A
          POP  DPL
          POP  DPH
          POP  PSW
          POP  ACC
          LJMP INICIO
```

4) Implemente programas em Assembly da família MCS51 para realizar as tarefas a seguir. Sugere-se que os programas implementados sejam simulados no ambiente Proview.

a) Defina o banco de registros 1, carregue o registro R0’ com o valor 32H e o registro R1’ com o valor E5H. Depois efetue a troca dos conteúdos de R0’ e R1’ utilizando apenas instruções `MOV`.

b) Refaça o programa anterior utilizando as instruções `POP` e `PUSH` para efetuar a troca dos conteúdos de R0’ e R1’.

c) Carregue no acumulador o conteúdo do endereço 36H da memória de dados interna utilizando endereçamento direto. Depois carregue o endereço 7FH da memória de dados interna com o valor armazenado no acumulador utilizando endereçamento direto.

d) Refaça o programa anterior utilizando endereçamento indireto.

e) Leia o conteúdo do endereço 0000H da memória de programa no acumulador e depois carregue os endereços 2000H e 3000H da memória de dados externa com o valor armazenado no acumulador.

Operações Lógicas

Fazem parte do grupo de operações lógicas as instruções CLR, CPL, ANL, ORL, XRL, RL, RLC, RR, RRC e SWAP.

A instrução CLR zera o conteúdo do acumulador, possuindo uma única sintaxe:

CLR A

- CLR A
  Zera o conteúdo do acumulador.

A instrução CPL complementa o conteúdo do acumulador bit a bit, possuindo uma única sintaxe:

CPL A

- CPL A
  Complementa o conteúdo do acumulador bit a bit.

As instruções ANL, ORL e XRL realizam as operações lógicas E, OU e OU-EXCLUSIVO, respectivamente, utilizando as seguintes sintaxes:

ANL <operando1>, <operando2>
ORL <operando1>, <operando2>
XRL <operando1>, <operando2>

O resultado da operação é armazenado no <operando1>, que pode ser o acumulador ou uma posição da memória de dados interna endereçada diretamente. O <operando2> pode utilizar todos os modos de endereçamento (imediato, direto, indireto e registro).

- ANL A, #dado
  Efetua a operação lógica E entre o conteúdo do acumulador e a constante (dado) bit a bit. O resultado é armazenado no acumulador.
  Exemplo: ANL A, #0FH

- ORL A, direto
  Efetua a operação lógica OU entre o conteúdo do acumulador e o endereço da memória de dados interna (direto) bit a bit. O resultado é armazenado no acumulador.
  Exemplo: ORL A, 3FH

- XRL A, Rn
  Efetua a operação lógica OU-EXCLUSIVO entre o conteúdo do acumulador e o conteúdo do registro R0, R1, R2, R3, R4, R5, R6 ou R7 (Rn). O resultado é armazenado no acumulador.
  Exemplo: XRL A, R1

As instruções `RL` e `RLC` deslocam o conteúdo do acumulador um bit à esquerda, passando ou não pelo flag de carry. Possuem as seguintes sintaxes únicas:

```
RL A
RLC A
```

- `RL A`

  Desloca o conteúdo do acumulador 1 bit à esquerda. O valor do bit 0 vai para o bit 1, o valor do bit 1 vai para o bit 2 e assim sucessivamente. O valor do bit 7 vai para o bit 0 e nenhum flag é afetado.

- `RLC A`

  Desloca o conteúdo do acumulador 1 bit à esquerda passando pelo flag de carry. O valor do bit 0 vai para o bit 1, o valor do bit 1 vai para o bit 2 e assim sucessivamente. valor do bit 7 vai para o flag de carry e o valor do flag de carry vai para o bit 0.

As instruções `RR` e `RRC` deslocam o conteúdo do acumulador um bit à esquerda, passando ou não pelo flag de carry. Possuem as seguintes sintaxes únicas:

```
RR A
RRC A
```

- `RR A`

  Desloca o conteúdo do acumulador 1 bit à direita. O valor do bit 7 vai para o bit 6, o valor do bit 6 vai para o bit 5 e assim sucessivamente. O valor do bit 0 vai para o bit 7 e nenhum flag é afetado.

- `RRC A`

  Desloca o conteúdo do acumulador 1 bit à direita passando pelo flag de carry. O valor do bit 7 vai para o bit 6, o valor do bit 6 vai para o bit 5 e assim sucessivamente. valor do bit 0 vai para o flag de carry e o valor do flag de carry vai para o bit 7.

A instrução `SWAP` troca o nibble mais significativo pelo nibble menos significativo do acumulador, possuindo a seguinte sintaxe única:

```
SWAP A
```

- `SWAP A`

  Troca o nibble mais significativo pelo nibble menos significativo do acumulador.

Para se ter noção de todas as operações lógicas existentes para a família MCS51 deve-se consultar a tabela de instruções completa que se encontra nos anexos.

Exercícios

1) Analise o programa em Assembly da família MCS51 dado a seguir e forneça a equação lógica do resultado final armazenado no acumulador em função dos registros utilizados:

```
LOGICA:   MOV  A, R0
          CPL  A
          ANL  R1
          MOV  B, A
          MOV  A, R2
          ANL  A, R3
          CPL  A
          XRL  R4
          ORL  A, B
          LJMP LOGICA
```

2) Implemente um programa em Assembly da família MCS51 que realize a seguinte equação lógica: ACC= [(R0+R1).R3]⊕R4\

3) Apresente 2 formas de se zerar o conteúdo do acumulador utilizando instruções lógicas do Assembly da família MCS51.

4) Apresente 2 formas de se complementar o conteúdo do acumulador utilizando instruções lógicas do Assembly da família MCS51.

5) Implemente um programa em Assembly da família MCS51 que seja capaz de setar apenas o bit 7 e de zerar apenas o bit 0 do registro de função especial B, sem alterar o estado dos seus demais bits.

6) Qual é o significado aritmético de uma rotação à esquerda do acumulador?

7) Qual é o significado aritmético de uma rotação à direita do acumulador?

Operações Aritméticas

Fazem parte do grupo de operações aritméticas as instruções `ADD`, `ADDC`, `SUBB`, `INC`, `DEC`, `MUL`, `DIV` e `DA`.

A instrução `ADD` realiza a adição entre o conteúdo do acumulador e o conteúdo de um endereço da memória de dados interna (ou uma constante), utilizando a seguinte sintaxe:

`ADD A, <operando>`

O resultado da adição é armazenado no acumulador, sendo possível utilizar para o `<operando>` todos os modos de endereçamento (imediato, direto, indireto e registro). O flag de carry é afetado conforme o resultado da operação.

- `ADD A, Rn`

Adiciona o conteúdo do acumulador ao conteúdo do registro R0, R1, R2, R3, R4, R5, R6 ou R7 (Rn). O resultado é armazenado no acumulador e o flag de carry é afetado conforme o resultado da operação.

Exemplo: `ADD A, R2`

A instrução `ADDC` realiza a adição entre o conteúdo do acumulador, o conteúdo de uma posição da memória de dados interna (ou uma constante) e o flag de carry, utilizando a seguinte sintaxe:

ADDC A, <operando>

O resultado da adição é armazenado no acumulador, sendo possível utilizar para o `<operando>` todos os modos de endereçamento (imediato, direto, indireto e registro). O flag de carry é afetado conforme o resultado da operação.

Esta instrução permite que adições com mais de 8 bits possam ser implementadas graças ao aproveitamento do flag de carry da operação anterior.

- `ADDC A, @Ri`

Adiciona o conteúdo do acumulador ao conteúdo do endereço da memória de dados interna apontado por R0 ou R1 (@Ri) e ao flag de carry. O resultado é armazenado no acumulador e o flag de carry é afetado conforme o resultado da operação.

Exemplo: `ADDC A, @R0`

A instrução `SUBB` realiza a subtração entre o conteúdo do acumulador, o conteúdo de uma posição da memória de dados interna (ou uma constante) e o flag de carry (borrow), utilizando a seguinte sintaxe:

SUBB A, <operando>

O resultado da subtração é armazenado no acumulador, sendo possível utilizar para o `<operando>` todos os modos de endereçamento (imediato, direto, indireto e registro). O flag de carry (borrow) é afetado conforme o resultado da operação.

Esta instrução permite que subtrações com mais de 8 bits possam ser implementadas graças ao aproveitamento do flag de carry (borrow) da operação anterior.

- `SUBB A, @Ri`

Subtrai do conteúdo do acumulador o conteúdo do endereço da memória de dados interna apontado por R0 ou R1 (@Ri) e o flag de carry (borrow). O resultado é armazenado no acumulador e o flag de carry (borrow) é afetado conforme o resultado da operação.

Exemplo: `SUBB A, @R1`

A instrução INC incrementa o operando de uma unidade, utilizando a seguinte sintaxe:

INC <operando>

É possível utilizar todos os modos de endereçamento (imediato, direto, indireto e registro) e ainda o registro de 16 bits DPTR.

- INC DPTR
  Incrementa o valor do registro de 16 bits DPTR.

A instrução DEC decrementa o operando de uma unidade, utilizando a seguinte sintaxe:

DEC <operando>

É possível utilizar todos os modos de endereçamento (imediato, direto, indireto e registro).

- DEC A
  Decrementa o valor do acumulador.

A instrução MUL realiza a multiplicação entre o acumulador e o registro de função especial B, possuindo uma única sintaxe:

MUL AB

- MUL AB
  Multiplica o conteúdo do acumulador pelo conteúdo do registro de função especial B, armazenando a parte mais significativa do resultado em B e a parte menos significativa no acumulador.

A instrução DIV realiza a divisão entre o acumulador e o registro de função especial B, possuindo uma única sintaxe:

DIV AB

- DIV AB
  Divide o conteúdo do acumulador pelo conteúdo do registro de função especial B, armazenando o quociente no acumulador e o resto em B. Se B for nulo o resultado da operação será indefinido.

A instrução DA realiza o ajuste decimal do acumulador após operações de adição em BCD, possuindo uma única sintaxe:

DA A

- `DA A`

  Realiza o ajuste decimal do acumulador. Deve ser utilizada apenas após operações de adição cujos operandos estejam codificados em BCD. O flag de carry é afetado conforme o resultado do ajuste.

Para se ter noção de todas as operações aritméticas existentes para a família MCS51 deve-se consultar a tabela de instruções completa que se encontra nos anexos.

Exercícios

1) Execute os programas dados a seguir no simulador do ambiente Proview e tire conclusões sobre o efeito de cada instrução executada. Para uma melhor noção do que ocorre, sugere-se que os programas sejam executados passo a passo, visualizando-se as janelas “Main Registers” e “Data View”.

a)
```
SOMA1:  MOV  PSW, #00H      ;adicao de 16 bits:
        MOV  R0, #23H       ; R0R1
        MOV  R1, #90H       ;+R2R3
        MOV  R2, #14H       ;-----
        MOV  R3, #59H       ; R4R5

        MOV  A, R1
        ADD  A, R3          ;adiciona R1 e R3
        MOV  R5, A          ;LSB em R5
        MOV  A, R0
        ADDC A, R2          ;adiciona R0 e R2
        MOV  R4, A          ;MSB em R4
        LJMP SOMA1
```

b)
```
SOMA2:  MOV  PSW, #00H      ;adicao de 16 bits em BCD:
        MOV  R0, #23H       ; R0R1
        MOV  R1, #90H       ;+R2R3
        MOV  R2, #14H       ;-----
        MOV  R3, #59H       ; R4R5

        MOV  A, R1
        ADD  A, R3          ;adiciona R1 e R3
        DA   A              ;ajusta resultado
        MOV  R5, A          ;LSB em R5
        MOV  A, R0
        ADDC A, R2          ;adiciona R0 e R2
        DA   A              ;ajusta resultado
        MOV  R4, A          ;MSB em R4
        LJMP SOMA2
```

c)
```
SUBT:   MOV  PSW, #00H      ;subtracao de 16 bits:
        MOV  R0, #23H       ; R0R1
        MOV  R1, #90H       ;-R2R3
        MOV  R2, #14H       ;-----
```

```
        MOV  R3, #59H       ; R4R5

        MOV  A, R1
        SUBB A, R3          ;subtrai R3 de R1
        MOV  R5, A          ;LSB em R5
        MOV  A, R0
        SUBB A, R2          ;subtrai R2 de R0
        MOV  R4, A          ;MSB em R4
        LJMP SUBT
```

d)
```
MULT:   MOV  PSW, #00H      ;multiplicacao de 16 bits:
        MOV  R0, #23H       ;    R0R1
        MOV  R1, #90H       ;   xR2R3
        MOV  R2, #14H       ;--------
        MOV  R3, #59H       ;R4R5R6R7

        MOV  A, R1
        MOV  B, R3
        MUL  AB             ;multiplica R1 e R3
        MOV  R7, A          ;LSB em R7
        MOV  R6, B          ;MSB em R6R7

        MOV  A, R1
        MOV  B, R2
        MUL  AB             ;multiplica R1 e R2
        ADD  A, R6
        MOV  R6, A          ;adiciona LSB em R6
        MOV  A, B
        ADDC A, #00H
        MOV  R5, A          ;MSB em R5

        MOV  A, R0
        MOV  B, R3
        MUL  AB             ;multiplica R0 e R3
        ADD  A, R6
        MOV  R6, A          ;adiciona LSB em R6
        MOV  A, B
        ADDC A, R5          ;adiciona MSB em R5
        MOV  R5, A

        MOV  A, R0
        MOV  B, R2
        MUL  AB             ;multiplica R0 e R2
        ADD  A, R5
        MOV  R5, A          ;adiciona LSB em R5
        MOV  A, B
        ADDC A, #00H
        MOV  R4, A          ;MSB em R4
        LJMP MULT
```

Manipulação de Variáveis Booleanas (Bits)

Fazem parte do grupo de manipulação de variáveis booleanas as instruções `CLR`, `SETB`, `CPL`, `ANL`, `ORL`, e `MOV`.

A instrução `CLR` reseta uma variável booleana, possuindo a seguinte sintaxe:

`CLR <operando>`

O `<operando>` corresponde ao endereço individual do bit que deseja-se zerar (da região da memória interna de dados endereçável bit-a-bit ou de um registro de função especial endereçável bit-a-bit).

- `CLR bit`
  Reseta o valor da variável booleana endereçada (bit).
  Exemplo: `CLR 35H`

A instrução `SETB` seta uma variável booleana, possuindo a seguinte sintaxe:

`SETB <operando>`

O `<operando>` corresponde ao endereço individual do bit que deseja-se setar (da região da memória interna de dados endereçável bit-a-bit ou de um registro de função especial endereçável bit-a-bit).

- `SETB bit`
  Seta o valor da variável booleana endereçada (bit).
  Exemplo: `SETB TR0`

A instrução `CPL` complementa uma variável booleana, possuindo a seguinte sintaxe:

`CPL <operando>`

O `<operando>` corresponde ao endereço individual do bit que deseja-se complementar (da região da memória interna de dados endereçável bit-a-bit ou de um registro de função especial endereçável bit-a-bit).

- `CPL bit`
  Complementa o valor da variável booleana endereçada (bit).
  Exemplo: `CPL C`

As instruções `ANL`, e `ORL` realizam as operações lógicas E e OU com o flag de carry, respectivamente, utilizando as seguintes sintaxes:

`ANL C, <operando>`
`ORL C, <operando>`

O resultado da operação é armazenado no flag de carry e o `<operando>` pode ser o endereço individual do bit (da região da memória interna de dados endereçável bit-a-bit ou de um registro de função especial endereçável bit-a-bit) ou seu complemento.

- `ANL C, bit`
  Efetua a operação lógica E entre o flag de carry e a variável booleana endereçada (bit).
  Exemplo: `ANL C, 08H`

- `ORL C, /bit`
  Efetua a operação lógica OU entre o flag de carry e o complemento da variável booleana endereçada (bit).
  Exemplo: `ORL C, /EA`

A instrução `MOV` copia o estado de uma variável booleana para o flag de carry ou vice-versa, utilizando as seguintes sintaxes:

`MOV C, <fonte>`
`MOV <destino>, C`

Os operandos `<fonte>` e `<destino>` deve ser o endereço individual do bit (deverá fazer parte da região da memória interna de dados endereçável bit-a-bit ou de um registro de função especial endereçável bit-a-bit).

- `MOV C, bit`
  Copia o estado da variável booleana endereçada (bit) para o flag de carry.
  Exemplo: `MOV C, 90H`

Para se ter noção de todas as instruções de manipulação de variáveis booleanas existentes para a família MCS51 deve-se consultar a tabela de instruções completa que se encontra nos anexos.

Exercícios

1) Apresente 2 formas de setar o estado do flag de carry utilizando instruções de manipulação de variáveis boolenas e operações lógicas do Assembly da família MCS51.

2) De que maneira você acredita que o microcontrolador identifica se os operandos das instruções `ANL`, `ORL` e `MOV` são registros ou bits? (Sugestão: analise a tabela de instruções completa existente nos anexos.)

3) Supondo que as variáveis booleanas X e Y foram atribuídas aos bits de endereço 10H e 20H, implemente um programa em Assembly da família MCS51 que realize a seguinte equação lógica: $C=X \oplus Y$.

Controle de Programa

Fazem parte do grupo de controle de programa as instruções LJMP, AJMP, SJMP, JMP, LCALL, ACALL, RET, RETI, JNZ, JZ, JNC, JC, JNB, JB, JBC, CJNE, DJNZ e NOP.

Este grupo de instruções atua em endereços da memória de programa, causando desvios na seqüência natural de execução do programa (registro de função especial PC).

As instruções LJMP, AJMP e SJMP são instruções de desvio incondicional e possuem as seguintes sintaxes:

```
LJMP <endereço>
AJMP <endereço>
SJMP <endereço>
  JMP @A+DPTR
```

O <endereço> da instrução LJMP é um endereço de 16 bits da memória de programa. Através dessa instrução é possível realizar desvios para qualquer endereço dos 64Kb de memória de programa.

- LJMP end16

  Efetua um desvio incondicional da execução para o endereço (end16) da memória de programa.

  Exemplo: LJMP 2000H

O <endereço> da instrução AJMP é um endereço de 11 bits da memória de programa. Através dessa instrução é possível realizar desvios apenas para a mesma página de 2Kb da memória de programa onde encontra-se a referida instrução.

- AJMP end11

  Efetua um desvio incondicional da execução na mesma página de 2Kb da memória de programa. O endereço de 16 bits do desvio é obtido pela CPU completando os 11 bits (end11) com os 5 bits mais significativos do endereço contido no registro de função especial PC.

  Exemplo: AJMP DESVIO

O <endereço> da instrução SJMP é um valor de 8 bits correspondente ao deslocamento relativo desejado em relação ao atual endereço de execução (registro de função especial PC). Através dessa instrução é possível realizar desvios relativos de –128 a +127 posições na memória de programa, pois o deslocamento é dado em complemento de dois.

- SJMP rel

  Efetua um desvio incondicional relativo da execução do programa. O endereço de 16 bits do desvio é obtido pela CPU somando-se o valor relativo (rel) ao endereço contido no registro de função especial PC.

  Exemplo: SJMP VOLTA

As instruções LCALL e ACALL são instruções de chamadas de subrotinas (funções) e possuem as seguintes sintaxes:

LCALL <endereço>
ACALL <endereço>

O <endereço> da instrução LCALL é um endereço de 16 bits da memória de programa. Através dessa instrução é possível realizar chamadas de subrotinas em qualquer endereço dos 64Kb de memória de programa.

- LCALL end16

Salva o conteúdo do registro de função especial PC na pilha (o registro SP é incrementado 2 vezes, pois trata-se de um registro de 16 bits). Em seguida efetua um desvio para o endereço da subrotina (end16). O valor salvo na pilha constitui o endereço de retorno a ser restaurado pela instrução RET ao final da subrotina.

Exemplo: LCALL 3000H

O <endereço> da instrução ACALL é um endereço de 11 bits da memória de programa. Através dessa instrução é possível realizar chamadas de subrotinas apenas para a mesma página de 2Kb da memória de programa onde encontra-se a referida instrução.

- ACALL end11

Salva o conteúdo do registro de função especial PC na pilha (o registro SP é incrementado 2 vezes, pois trata-se de um registro de 16 bits). Em seguida efetua um desvio para o endereço da subrotina (end11) na mesma página de 2KB da memória de programa. O valor salvo na pilha constitui o endereço de retorno a ser restaurado pela instrução RET ao final da subrotina.

Exemplo: ACALL TEMPO

As instruções RET e RETI são instruções que realizam o retorno de chamadas de subrotinas e interrupções, respectivamente, possuindo as seguintes sintaxes:

RET
RETI

As instruções de retorno recuperam o endereço original contido no registro de função especial PC no momento da chamada da subrotina ou interrupção.

- RET

Retorna de uma subrotina. O endereço de retorno é recuperado da pilha e armazenado no registro de função especial PC (o registro SP é decrementado 2 vezes, pois trata-se de um registro de 16 bits).

Exemplo: RET

- `RETI`

Retorna de uma subrotina. O endereço de retorno é recuperado da pilha e armazenado no registro de função especial PC (o registro SP é decrementado 2 vezes, pois trata-se de um registro de 16 bits). A diferença em relação à instrução `RET` é que a instrução `RETI` habilita novamente interrupções de menor ou igual prioridade, anteriormente desabilitadas na ocorrência da interrupção.

Exemplo: `RETI`

As instruções `JNZ`, `JZ`, `JNC` e `JC` são instruções de desvio condicional e possuem as seguintes sintaxes:

JNZ <endereço>
JZ <endereço>
JNC <endereço>
JC <endereço>

A condição de desvio é dada pelo mnemônico da instrução utilizada: `JNZ` efetua um desvio se o conteúdo do acumulador for diferente de zero, `JZ` efetua um desvio se o conteúdo do acumulador for igual de zero, `JNC` efetua um desvio se o estado do flag de carry for igual a zero e `JC` efetua um desvio se o estado do flag de carry for igual a um. O `<endereço>` de todas essas instruções é sempre um valor de 8 bits correspondente ao deslocamento do desvio relativo desejado em relação ao atual endereço de execução (registro de função especial PC). Através dessa instrução é possível realizar desvios relativos de –128 a +127 posições na memória de programa, pois o deslocamento é dado em complemento de dois.

- `JNZ rel`

Testa o conteúdo do acumulador e efetua um desvio relativo no programa se for diferente de zero. O endereço de 16 bits do desvio é obtido pela CPU somando-se o valor relativo (rel) ao endereço contido no registro de função especial PC, se o resultado do teste for verdadeiro.

Exemplo: `JNZ VOLTA`

- `JZ rel`

Testa o conteúdo do acumulador e efetua um desvio relativo no programa se for igual a zero. O endereço de 16 bits do desvio é obtido pela CPU somando-se o valor relativo (rel) ao endereço contido no registro de função especial PC, se o resultado do teste for verdadeiro.

Exemplo: `JZ ZERO`

- `JNC rel`

Testa o flag de carry e efetua um desvio relativo no programa se for igual a zero. O endereço de 16 bits do desvio é obtido pela CPU somando-se o valor relativo (rel) ao endereço contido no registro de função especial PC, se o resultado do teste for verdadeiro.

Exemplo: `JNC SEM`

- `JC rel`

  Testa o flag de carry e efetua um desvio relativo no programa se for igual a um. O endereço de 16 bits do desvio é obtido pela CPU somando-se o valor relativo (rel) ao endereço contido no registro de função especial PC, se o resultado do teste for verdadeiro.

  Exemplo: `JC CARRY`

As instruções `JNB`, `JB` e `JBC` também são instruções de desvio condicional, porém utilizando variáveis booleanas genéricas para o teste da condição. Possuem as seguintes sintaxes:

`JNB <operando>, <endereço>`
`JB <operando>, <endereço>`
`JBC <operando>, <endereço>`

A condição de desvio é dada pelo mnemônico da instrução utilizada: `JNB` efetua um desvio se o estado da variável booleana for igual a zero, `JB` efetua um desvio se o estado da variável booleana for igual a um e JBC efetua um desvio se o estado da variável booleana for igual a um, posteriormente resetando-a. Em todas essas instruções o `<operando>` é sempre o endereço individual de um bit (da região da memória interna de dados endereçável bit-a-bit ou de um registro de função especial endereçável bit-a-bit) e o `<endereço>` é sempre um valor de 8 bits correspondente ao deslocamento do desvio relativo desejado em relação ao atual endereço de execução (registro de função especial PC). O deslocamento é dado em complemento de dois.

- `JNB bit, rel`

  Testa a variável booleana endereçada (bit) e efetua um desvio relativo no programa se for igual a zero. O endereço de 16 bits do desvio é obtido pela CPU somando-se o valor relativo (rel) ao endereço contido no registro de função especial PC, se o resultado do teste for verdadeiro.

  Exemplo: `JNB EA, INTERR`

- `JB bit, rel`

  Testa a variável booleana endereçada (bit) e efetua um desvio relativo no programa se for igual a um. O endereço de 16 bits do desvio é obtido pela CPU somando-se o valor relativo (rel) ao endereço contido no registro de função especial PC, se o resultado do teste for verdadeiro.

  Exemplo: `JB TF1, TIME`

- `JBC bit, rel`

  Testa a variável booleana endereçada (bit) e efetua um desvio relativo no programa se for igual a um, posteriormente resetando-a. O endereço de 16 bits do desvio é obtido pela CPU somando-se o valor relativo (rel) ao endereço contido no registro de função especial PC, se o resultado do teste for verdadeiro.

  Exemplo: `JBC 20H, SINAL`

A instrução CJNE realiza uma comparação entre 2 operandos e, caso não sejam iguais, efetua um desvio na execução do programa. Possui a seguinte sintaxe:

CJNE <operando1>, <operando2>, <endereço>

O <operando1> pode ser o acumulador, um registro do banco de registros em uso ou um ponteiro, o <operando2> pode ser um endereço da memória de dados interna ou uma constante e o <endereço> é sempre um valor de 8 bits correspondente ao deslocamento do desvio relativo desejado, a ser efetuado no caso do <operando1> e do <operando2> serem diferentes. O deslocamento é dado em complemento de dois.

- CJNE A, direto, rel

Compara o conteúdo do acumulador com o conteúdo da posição de memória de dados interna (direto). Se os valores forem iguais, o programa prossegue normalmente para a próxima instrução. Caso contrário, um desvio é realizado pela CPU somando-se o valor relativo (rel) ao endereço contido no registro de função especial PC.

Exemplo: CJNE A, 1FH, TECLA

A instrução DJNZ realiza uma decremento e, caso o resultado seja diferente de zero, efetua um desvio na execução do programa. Possui a seguinte sintaxe:

DJNZ <operando>, <endereço>

O <operando> pode ser um registro do banco de registros em uso ou um endereço da memória de dados interna ou uma constante e o <endereço> é sempre um valor de 8 bits correspondente ao deslocamento do desvio relativo desejado, a ser efetuado no caso do <operando1> ser diferente de zero. O deslocamento é dado em complemento de dois.

- DJNZ Rn, rel

Decrementa o conteúdo do registro R0 a R7 (Rn) do banco de registros em uso, ocorrendo um desvio no programa caso o resultado seja diferente de zero. Se o resultado após o decremento for nulo o programa prossegue para a próxima instrução. Caso ocorra um desvio, o endereço de desvio é calculado pela CPU somando-se o valor relativo (rel) ao endereço contido no registro de função especial PC.

Exemplo: DJNZ R0, LOOP

A instrução NOP aguarda o tempo de um ciclo de máquina da CPU sem efetuar nenhuma operação. Possui a seguinte sintaxe:

NOP

- NOP
  Sem operação, apenas aguarda o tempo relativo a um ciclo de máquina.
  Exemplo: `NOP`

Para se ter noção de todas as instruções de manipulação de variáveis booleanas existentes para a família MCS51 deve-se consultar a tabela de instruções completa que se encontra nos anexos.

Exercícios

1) Implemente uma subrotina que receba um valor no registro R0 como entrada e retorne esse valor elevado ao quadrado nos registros R1 e R2 (R1 deverá conter o byte mais significativo e R2 o menos significativo).

2) Implemente uma subrotina que receba um valor no acumulador e retorne um valor no registro B, seguindo a correspondência dada na tabela a seguir:

A	10H	20H	30H	40H
B	98H	76H	54H	23H

3) A instrução `NOP` é bastante utilizada em laços de repetição para gerar temporizações através de software, como mostrado na subrotina a seguir. Consultando a tabela de instruções completa para obter os tempos de execução de cada instrução e supondo uma freqüência de clock de 6MHz, calcule o tempo gerado pela subrotina ATRASO.

```
;*********************************************************
; ATRASO - gera um tempo de atraso por software
; ENTRADA: nada
; SAIDA: tempo de atraso
; DESTROI: R1 e R2 do banco de registros em uso
;*********************************************************
ATRASO:   MOV  R1, #80H
A1:       MOV  R2, #90H
A2:       NOP
          NOP
          DJNZ R2, A2
          DJNZ R1, A1
          RET
```

4) Como a instrução DA A somente pode ser usada para o ajuste decimal do acumulador após operações de adição, foi desenvolvida uma subrotina para o ajuste decimal do acumulador após operações de subtração. Analise o funcionamento da subrotina DAASUB, cuja listagem é mostrada a seguir e tire conclusões sobre o seu funcionamento.

```
;*********************************************************
; DAASUB - ajuste decimal do acumulador apos subtracoes
; ENTRADA: A
; SAIDA: A
;*********************************************************
```

```
DAASUB:   PUSH PSW
          PUSH A
          ANL  A, #0FH
          CLR  C
          SUBB A, #0AH
          JC  SEM0        ;inferior precisa de ajuste?
          POP  A
          CLR  C
          SUBB A, #06H    ;ajusta nibble inferior
          PUSH A
SEM0:     CPL  C          ;nao ajusta nibble inferior
          POP  A
          PUSH A
          ANL  A, #F0H
          CLR  C
          SUBB A, #A0H
          JC   SEM1       ;superior precisa de ajuste?
          POP  A
          CLR  C
          SUBB A, #60H    ;ajusta nibble superior
          PUSH A
SEM1:     CPL  C          ;nao ajusta nibble superior
          POP  A
          POP  PSW
          RET
```

5) A subrotina mostrada em seguida realiza a comparação de valores de 24 bits. As variáveis de entrada são ponteiros para os bytes mais significativos dos valores a serem comparados. Analise o funcionamento da subrotina COMP3 e determine como o programador deve modificá-la caso queira ampliar o número de bytes dos valores a serem comparados.

```
;*********************************************************
; COMP3 - compara valores de 3 bytes apontados por R0 e R1
; ENTRADA: R0 e R1 = ponteiros para MSB
; SAÍDA: se  Z = 1 -> R0=R1
;        se CY = 0 -> R0>R1
;        se CY = 1 -> R0<R1
; DESTROI: A, C, R0, R1 e R2 do banco de registros em uso
;*********************************************************
COMP3:    MOV  R2, #03H
DENTRO:   CLR  C
          MOV  A, @R0
          SUBB A, @R1
          JC   FORA
          JNZ  FORA
          INC  R0
          INC  R1
          DJNZ R2, DENTRO
FORA:     RET
```

Exercícios Práticos

Elabore os seguintes programas em Assembly da família MCS51, verificando o seu funcionamento utilizando o monitor PaulMon e a placa P51:

1) Preencha a faixa de 3000h a 3030h da memória de dados externa com a constante CDh.

2) Carregue 50 bytes a partir da posição 20h da memória de dados interna com valores crescentes de 05h a 20h.

3) Copie os conteúdos das posições 40h a 5Fh da memória de dados interna para as posições 4000h a 401Fh da memória de dados externa.

4) Conte o total de bytes iguais a 02h existentes nas primeiras 128 posições da memória de programa (PaulMon). Apresente o resultado em decimal através das subrotinas disponíveis no PaulMon.

5) Conte o total de bytes ímpares existentes nas primeiras 1024 posições da memória de programa (PaulMon). Apresente o resultado em hexadecimal através das subrotinas disponíveis no PaulMon.

6) Encontre o byte de maior valor existente nas primeiras 256 posições da memória de programa (PaulMon). Apresente o resultado em hexadecimal através das subrotinas disponíveis no PaulMon.

LINGUAGEM ASSEMBLY

A linguagem Assembly consiste a grosso modo numa seqüência de mnemônicos que serão posteriormente traduzidos pelo montador (assembler) para os códigos das instruções do microcontrolador.

O assembler possui algumas características interessantes em termos de facilidade no desenvolvimento de programas:

Pseudo-instruções

Trata-se de declarações no programa semelhantes às instruções do microcontrolador, mas que não fazem parte do seu conjunto de instruções. Na verdade são diretivas de montagem do próprio montador. Algumas das pseudo-instruções mais importantes são:

`ORG end`	Define a origem (endereço) do programa, subrotina, tabela, etc.
`DB dado, [...]`	Define constantes (mensagens, tabelas, etc.) a serem gravadas na memória de programa

Definições

São declarações que atribuem nomes a constantes, permitindo referências ao longo do programa.

`rótulo EQU valor`	Atribui o nome do rótulo à constante especificada (valor).

Rótulos

São nomes atribuídos aos endereços de rotinas de tratamento de interrupções, subrotinas, tabelas ou simplesmente locais de desvio do fluxo do programa utilizados em estruturas de programação (decisões e repetições), facilitando referências ao longo do programa.

Comentários

Qualquer texto digitado após ponto-e-vírgula (;) é desprezado pelo assembler durante a montagem do programa. Dessa forma, pode-se fazer uso de comentários úteis para a documentação e melhor entendimento do programa.

Estrutura de um Programa em Assembly

A seguinte formatação deve ser seguida: rótulos na coluna da esquerda, mnemônicos na coluna central e comentários na coluna direita do programa. O montador processa o arquivo contendo o código-fonte em Assembly (extensão .ASM) e gera arquivos contendo a listagem em código de máquina (extensão .LST) e o código-objeto do programa (extensões .OBJ ou .HEX).

Exemplo: Programa Simples em Assembly

```
TESTE     EQU 01010101B           ;CONSTANTE DE TESTE

          ORG 0000H               ;ENDERECO DE RESET
RESET:    LJMP INICIO             ;SALTA PARA O INICIO

          ORG 0100H               ;INICIO DO PROGRAMA
INICIO:   MOV A, #040H            ;INICIALIZA CONTADOR
          MOV R0, #64             ;ENDERECO INICIAL
REPETE:   MOV @R0, #TESTE         ;ESCREVE CONSTANTE
          INC R0                  ;PRÓXIMO ENDEREÇO
          DEC A                   ;DECREMENTA CONTADOR
          CJNE A, #0, REPETE      ;SE NAO TERMINOU REPETE
FINAL:    JMP FINAL
```

Listagem em Código de Máquina Gerada pelo Montador

```
LOC     OBJ           LINE    SOURCE
0055                     1    TESTE     EQU 01010101B
                         2
0000                     3              ORG 0000H
0000    020100           4    RESET:    LJMP INICIO
                         5
0100                     6              ORG 0100H
0100    7440             7    INICIO:   MOV A, #040H
0102    7840             8              MOV R0, #64
0104    7655             9    REPETE:   MOV @ R0 , # 85
0106    08              10              INC R0
0107    14              11              DEC A
0108    B400F9          12              CJNE A, #0, REPETE
010B    80FE            13    FINAL:    JMP FINAL
```

TÓPICOS IMPORTANTES EM PROGRAMAÇÃO

Pilha

É uma estrutura de dados do tipo LIFO (Last-In-First-Out) utilizada para armazenamento temporário de dados e endereços. A denominação de pilha vem do fato do último dado colocado ser o primeiro a ser retirado. O registro SP do 8051 é um ponteiro que indica o topo da pilha (endereço do último dado a colocado na pilha).

É na pilha que são armazenados os endereços de retorno de interrupções e subrotinas. Deve-se tomar extremo cuidado no uso da pilha para não desequilibrá-la (colocar mais dados do que retirar ou vice-versa) e para que não haja estouro da sua capacidade, uma vez que a região de memória destinada a essa finalidade é limitada.

Exemplo

```
          PUSH ACC
          PUSH PSW
          PUSH DPL
          PUSH DPH
          {...}
          POP  DPH
          POP  DPL
          POP  PSW
          POP  ACC
```

Subrotinas

Trata-se de um recurso de programação muito útil quando uma seqüência de ações é bastante utilizada dentro de um programa. Comportam-se de modo semelhante ao das interrupções. A diferença é que as subrotinas são solicitadas por software e não possuem endereços de desvio fixos.

Exemplo

```
INICIO:   MOV  R0, #11H
          CALL ENVIA
          MOV  R0, #22H
          CALL ENVIA
          MOV  R0, #33H
          CALL ENVIA
          SJMP INICIO

ENVIA:    ANL  R0, #0FH
          ORL  R0, #30H
          MOV  SBUF, R0
          RET
```

Decisões

- IF-ELSE (SE-SENÃO)

São estruturas de programação que permitem desviar o fluxo de execução de um programa conforme o estado (falso ou verdadeiro) de determinadas condições. Tais estados são reportados ao software através dos flags existentes no registro PSW do 8051. O flas mais importante existente no 8051 é o C (carry).

Exemplo

```
          CLR  C
          SUBB A, R0
          JZ   IGUAL
          JC   MAIOR
MENOR:    MOV  R1, #2
          SJMP FIM
MAIOR:    MOV  R1, #1
          SJMP FIM
IGUAL:    MOV  R1, #0
          SJMP FIM
```

Repetições

- FOR (DE-ATÉ)
- WHILE (ENQUANTO)
- DO-WHILE (FAÇA-ENQUANTO)

São estruturas de programação que permitem a repetição (loop) de um conjunto de ações enquanto determinada condição for verdadeira. Essas estruturas podem verificar a condição antes da execução das ações ou depois de já terem executado pelo menos uma vez as ações dentro do loop.

Exemplo

```
          MOV  A, #10H
REPETE:   {...}
          DEC  A
          JNZ  REPETE
```

LINGUAGEM C

A programação do 8051 na linguagem C normalmente é mais inteligível ao programador pouco acostumado ao Assembly, por se tratar de uma linguagem estruturada de alto nível. Entretanto o bom conhecimento e entendimento do Assembly do microcontrolador com que se está trabalhando permanece essencial quando se trata de encontrar erros de programação (bugs) e implementar rotinas de alta eficiência, seja em termos de velocidade ou de tamanho.

O papel executado pelo compilador é o de traduzir o programa em C para o Assembly, de modo que este possa ser montado em código de máquina. A grande vantagem de uma linguagem de alto nível é que o programador é poupado de algumas tarefas trabalhosas, como a alocação de memória para variáveis, a passagem de parâmetros para funções, entre outras. A linguagem C oferece ainda a possibilidade de utilização de bibliotecas de funções e também a possibilidade de se portar programas com maior facilidade para outras famílias de microcontrolador.

Exemplo: Programa Simples em Linguagem C

```
void main(void)
{
     unsigned char cont, soma=0;

     for(cont=1; cont<10; cont++)
          soma+=cont;

     while(1);
} /* main*/
```

Listagem em Assembly Gerada pelo Compilador

```
             ; FUNCTION main (BEGIN)
                                                ; SOURCE LINE # 3
0000 750000  R       MOV    soma,#000H
             ; R7 is assigned to cont
                                                ; SOURCE LINE # 5
0003 7F01            MOV    R7,#001H
0005          ?FOR1:
                                                ; SOURCE LINE # 6
0005 E500    R       MOV    A,soma
0007 2F              ADD    A,R7
0008 F500    R       MOV    soma,A
                                                ; SOURCE LINE # 5
000A 0F              INC    R7
000B BF0AF7          CJNE   R7,#00AH,?FOR1
000E          ?WHILE1:
                                                ; SOURCE LINE # 8
000E 80FE            SJMP   ?WHILE1
             ; FUNCTION main (END)
```

PORTS DE ENTRADA E SAÍDA

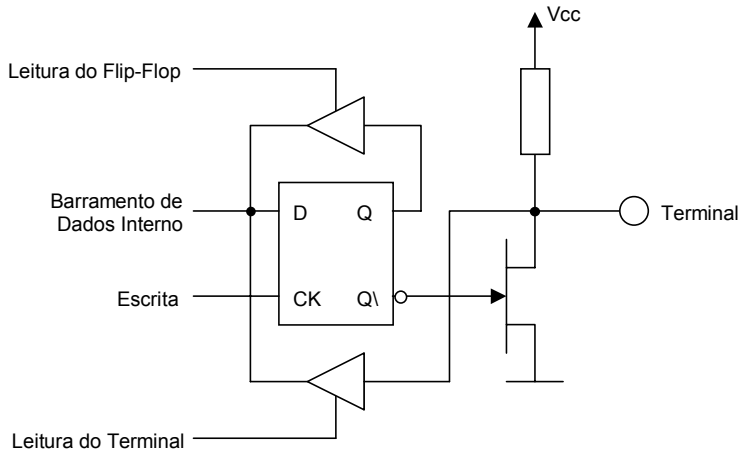
O 8051 possui 32 bits de E/S, acessíveis como Ports (8 bits) ou individualmente. Todos os bits podem ser utilizados como vias de entrada ou saída, em função do projeto de hardware e software do sistema em questão.

O uso dos bits de E/S é bastante simples, pois para cada Port existe um registro de função especial. Basta apenas que o software escreva o valor desejado no registro correspondente (P0, P1, P2 ou P3), no caso de operações de saída, ou efetue uma leitura, no caso de operações de entrada. É possível realizar escritas e leituras individualmente nos bits, pois os registros de função especial dos Ports são bit-endereçáveis.

Exemplos: Para se escrever um byte no Port 1 (8 bits de saída), pode-se utilizar a instrução MOV P1, #byte. Para se ler um byte do port P3 (8 bits de entrada), pode-se utilizar a instrução MOV A, P3.

Para acessar bits individuais dos Ports pode-se fazer uso da instrução MOV bit, C para efetuar operações de saída e a instrução MOV C, bit para efetuar operações de entrada. Além de instruções de transferência de dados, também é possível utilizar instruções lógicas (ANL, ORL, XRL), aritméticas (ADD, ADDC, DEC, INC, SUBB), de manipulação de variáveis booleanas (CLR, CPL, SETB) e de controle de programa (CJNE, DJNZ, JB, JNB, JBC) para operações de E/S.

Estrutura Interna Simplificada de um Terminal do Port 1



No caso de leitura, cada bit comporta-se como um buffer tri-state e no caso de escrita comporta-se como um flip-flop D.

Quando um bit é escrito no Port, este trafega pelo barramento de dados interno do microcontrolador até a entrada do flip-flop D e, após um pulso na entrada de clock, é levado à saída do mesmo. O restante do circuito eletrônico encarrega-se de levar o nível lógico adequado ao terminal externo.

No caso de leitura do Port, algumas das instruções lêem o estado dos terminais, enquanto outras lêem o estado dos flip-flops internos. As instruções que lêem o estado dos flip-flops são aquelas que alteram e escrevem novamente o valor dos bits (INC, DEC, CPL, JBC, DJNZ, ANL, ORL e XRL). As demais instruções lêem o estado presente nos terminais externos.

Os Ports 1, 2 e 3 possuem resistores de pull-up internos e são chamados de "quase bidirecionais". Tal característica faz com que possuam sempre estados lógicos definidos, de forma que, mesmo ao serem utilizados como entradas e em aberto, pode-se sempre medir seus níveis lógicos. O Port 0 não possui essa característica e seus terminais quando em aberto ficarão flutuando. O Port 0 possui condições elétricas de alimentar duas cargas TTL e os Ports 1, 2 e 3 apenas uma carga TTL quando operam como saída.

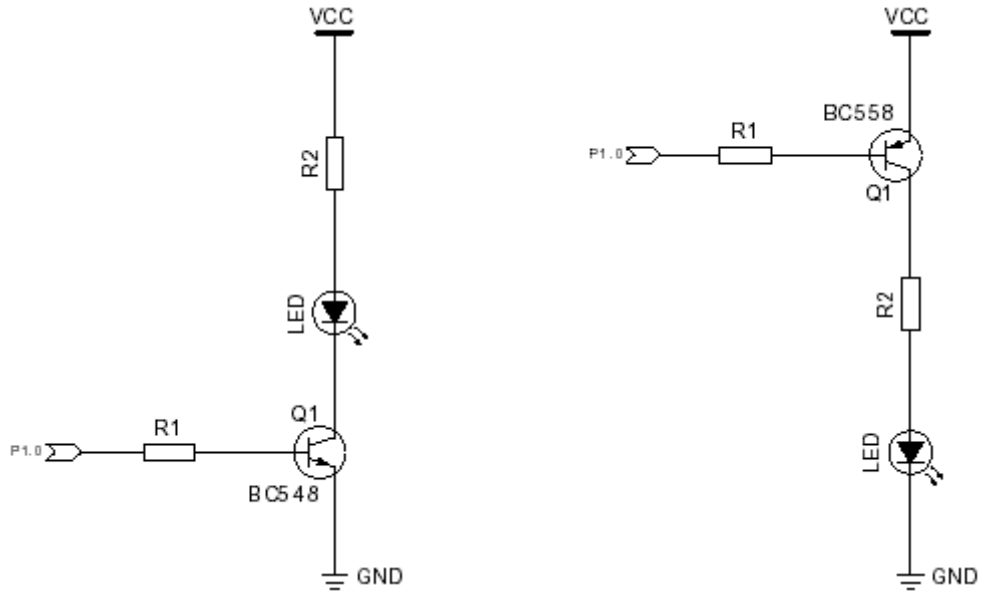
É importante notar que, para que seja possível determinado bit atuar como entrada, é necessário que o seu respectivo flip-flop esteja armazenando nível lógico 1. Nessa situação o transistor de saída não estará conduzindo, permitindo que o circuito externo seja capaz de apresentar tanto nível lógico 0 quanto nível lógico 1 ao microcontrolador. Entretanto, se o flip-flop estiver armazenando nível lógico 0, o transistor de saída estará conduzindo e forçando nível lógico 0 no respectivo terminal, impossibilitando dessa forma, o seu uso como entrada.

Acionamentos de LED

Uma das formas mais simples de saída em um sistema microcontrolado é o acionamento de um LED. Essa aplicação pode ser útil para indicadores visuais e para acionamentos de potência através de opto-acopladores.

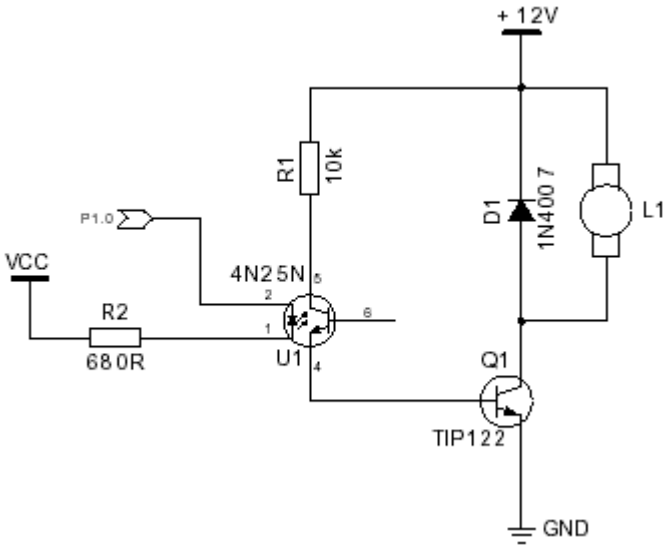
Como os Ports dos microcontroladores da família MCS51 são capazes de fornecer uma corrente muito pequena, devido aos resistores de pull-up internos (com exceção do Port 0), para acender um LED pode-se fazer uso dos circuitos mostrados abaixo.

Os circuitos acima podem ser implementados graças aos transistores de saída dos Ports, que são capazes de conduzir uma corrente razoável, da ordem de 20 a 25 mA (o manual do componente deverá ser consultado para informações mais precisas). Caso a capacidade de condução dos transistores de saída dos Ports do componente não seja suficiente, deve-se recorrer a transistores externos, como mostram os circuitos a seguir.



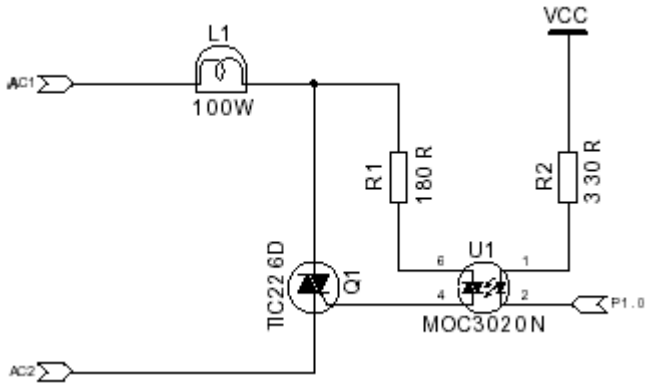
Acionamentos de Potência DC

Para acionamentos de potência DC é recomendável o uso de opto-acopladores para isolamento do sistema microcontrolado. O principal objetivo é evitar que um possível defeito na parte de potência acabe aplicando tensões e correntes elevadas na parte digital do circuito. A figura abaixo mostra uma possível configuração de circuito para o acionamento de um motor DC através do bit P1.0 do 8031.



Acionamentos de Potência AC

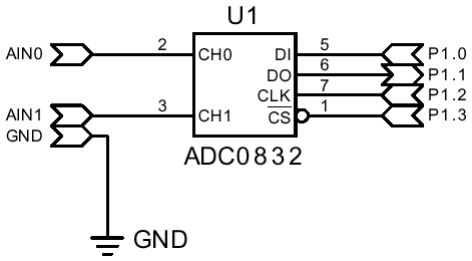
Para acionamentos de potência AC torna-se imprescindível o uso de opto-acopladores para proteção do sistema, pois as tensões e correntes envolvidas tendem a ser elevadas. A figura a seguir apresenta uma possível configuração de circuito para o acionamento de uma lâmpada AC através do bit P1.0 do 8031.



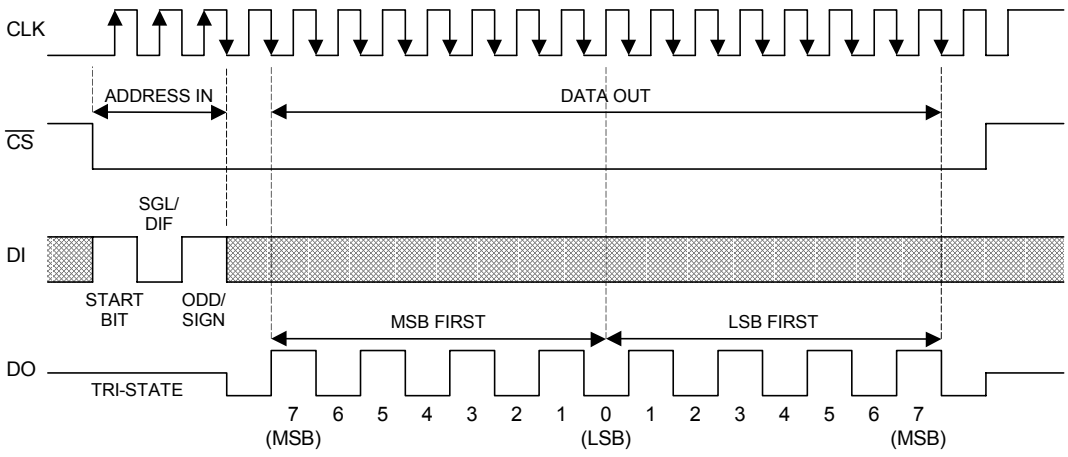
Outras Aplicações

Pode-se utilizar os ports de entrada e saída do 8031 para a conexão com periféricos com interface serial. Existem diferentes padrões de interface serial para periféricos, entre eles o padrão I^2C da Philips e o Microwire da National.

Podemos citar como exemplo de aplicação o interfaceamento de um conversor A/D serial ADC0832 com o port P1 do 8031.



O software envolvido nesta aplicação necessita gerar sinais adequados para o funcionamento do periférico, conforme informações do manual do fabricante. O conversor ADC0832 possui resolução de 8 bits, 2 canais de entrada e interface serial Microwire. O programa deverá controlar 4 dos terminais do port P1 do 8031 para gerar os sinais CLK, CS e DI e para ler o sinal da conversão DO, conforme o diagrama de tempo fornecido abaixo.



A seleção dos canais do conversor é feita conforme a tabela-verdade mostrada a seguir.

SGL/DIF	ODD/SIGN	CANAL
0	0	CH0-CH1 (diferencial)
0	1	CH1-CH0 (diferencial)
1	0	CH0 (entrada simples)
1	1	CH1 (entrada simples)

Abaixo seguem exemplos de subrotinas para a leitura dos canais do conversor ADC0832:

```
;*********************************************************
; Descrição dos terminais
;*********************************************************
DO        EQU  90H          ;DO = P1.0
DI        EQU  91H          ;DI = P1.1
CK        EQU  92H          ;CK = P1.2
CS        EQU  93H          ;CS = P1.3

;*********************************************************
; PULSE - pulso de clock para o conversor A/D ADC0832
;*********************************************************
PULSE:    SETB CK        ;CLK em nivel alto
          NOP
          CLR  CK        ;CLK em nivel baixo
          RET

;*********************************************************
; CONVAD - leitura do conversor A/D ADC0832
; ENTRADA: A = endereco do mux
; SAIDA: A = valor da conversao
; DESTROI: B
;*********************************************************
CONVAD:   CLR  CK
          CLR  CS        ;habilita o ADC0832
          MOV  B, #3          ;3 bits a enviar
LOOPA:    RLC  A
          MOV  DI, C     ;envia bit para DI
          CALL PULSE     ;pulso de clock
          DJNZ B, LOOPA
          CALL PULSE
          MOV  B, #8     ;8 bits a receber
LOOP2:    MOV  C, DO     ;recebe bit de DO
          RLC  A
          CALL PULSE     ;pulso de clock
          DJNZ B, LOOP2
          SETB CS        ;desabilita o ADC0832
          RET
```

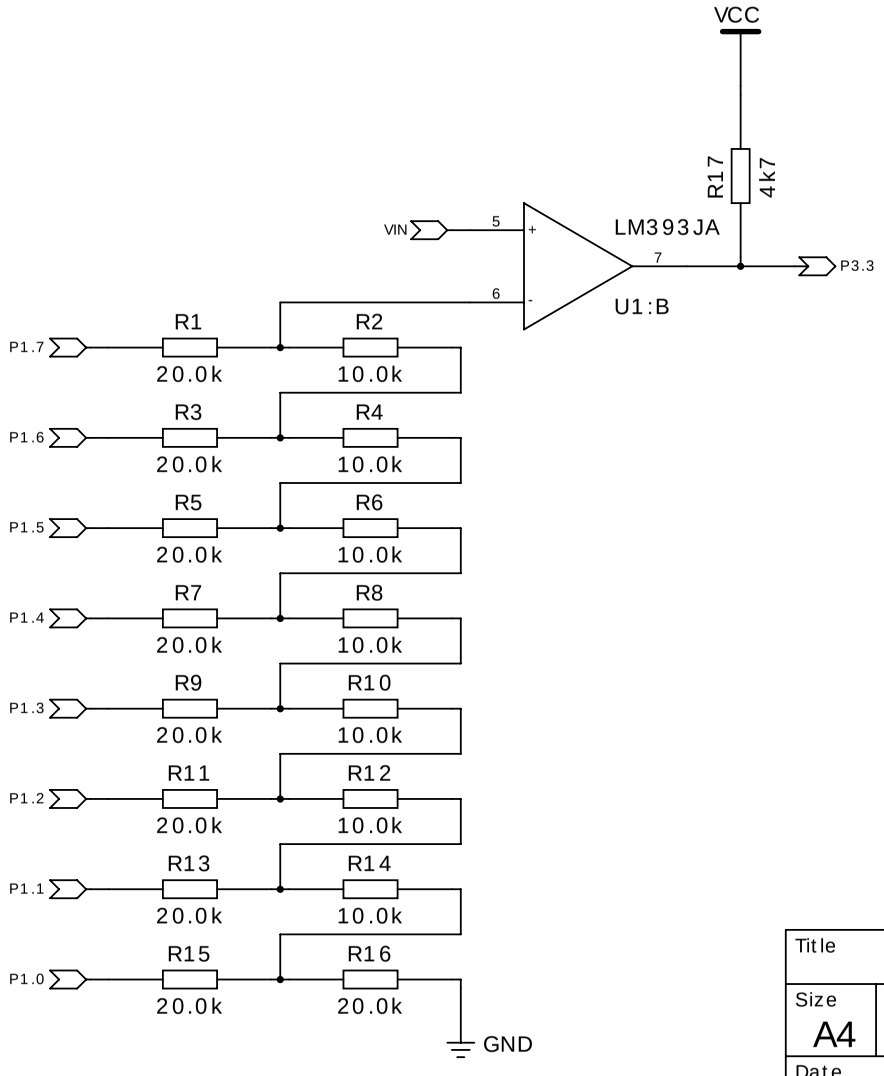
Nas páginas a seguir são apresentados alguns diagramas esquemáticos de outros sistemas de hardware conectados aos ports de entrada e saída do 8031.

O primeiro diagrama esquemático mostra a implementação de um conversor A/D discreto controlado por software. O circuito consiste em um conversor D/A do tipo R2R cujas entradas encontram-se conectadas aos bits do port P1 do 8031 e cuja saída encontra-se conectada à entrada inversora de um comparador. O conversor R2R apresenta em sua saída uma tensão analógica proporcional ao valor digital presente no port P1, de maneira que esta pode ser comparada com a tensão analógica da entrada não-inversora. O sistema é então realimentado através do bit 3 do port 3, onde encontra-se conectada a saída do comparador. O programa a seguir realiza uma conversão A/D por rampa escalonada decrescente:

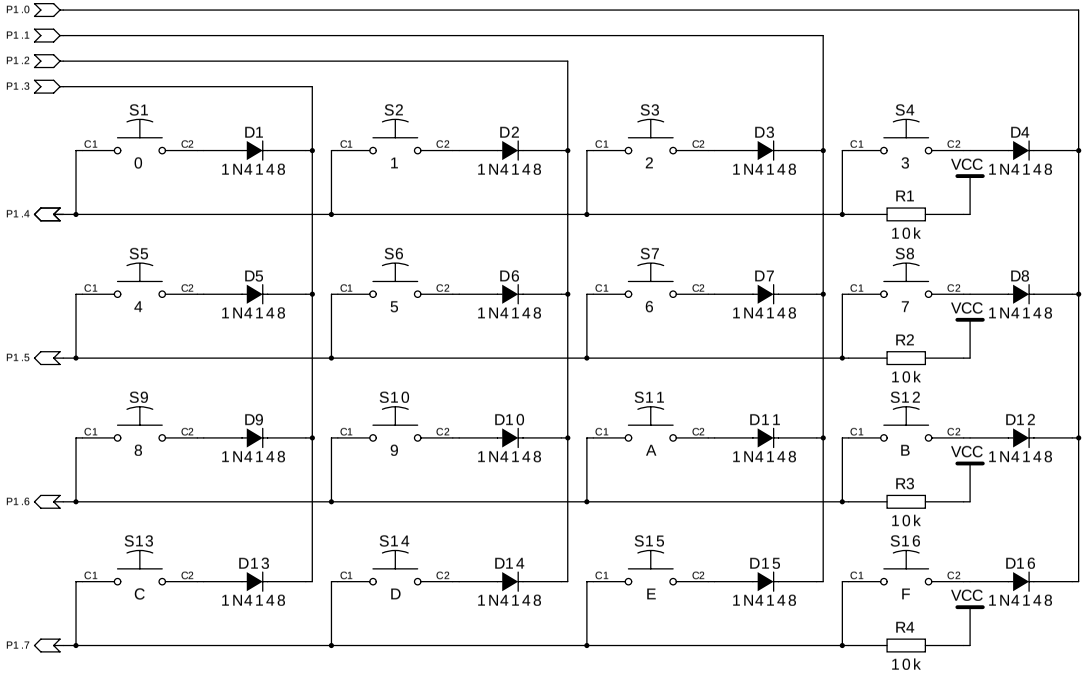
```
;*********************************************************
; Descrição dos terminais
;*********************************************************
CMP     EQU  0B3H         ;P3.3 = saida do comparador

;*********************************************************
; CAD - realiza uma conversao A/D por rampa escalonada
; ENTRADA: tensão analógica
; SAIDA: A
; DESTROI: R0
;*********************************************************
RAMPA:    MOV  R0, #0FFH  ;tensao maxima para comparacao
REPETE:   MOV  P1, R0     ;atualiza a rede R2R
          NOP             ;aguarda estabilizacao
          JNB  CMP, FINAL ;verifica se igualou com entrada
          DJNZ R0, REPETE ;diminui a tensao e repete
          MOV  A, R0      ;A contem o valor da conversao
          RET
```

O outro diagrama esquemático apresentado mostra a implementação de um teclado matricial conectado ao port P1 do 8031. Para o funcionamento desta configuração de teclado, o programa deve realizar uma varredura das colunas em nível lógico 0, verificando nas linhas se existe alguma tecla acionada. Os diodos presentes no circuito asseguram que não exista interferência caso teclas em diferentes colunas sejam acionadas ao mesmo tempo. A seguir é apresentado um exemplo de subrotina de controle para este teclado matricial:

```
;*********************************************************
; ESQUEMA DE LIGACAO DO TECLADO
;*********************************************************
; P1.3   P1.2   P1.1   P1.0
;   1      2      3      4    P1.4
;   5      6      7      8    P1.5
;   9      0      A      B    P1.6
;   C      D      E      F    P1.7
;*********************************************************
```



Title	CONVERSOR A/ D (R2R)	
Size A4	Number	Rev 1.0
Date 03/ 2002		Drawn by HUGO
Filename ADR2R51.S01		Sheet 1 of 1



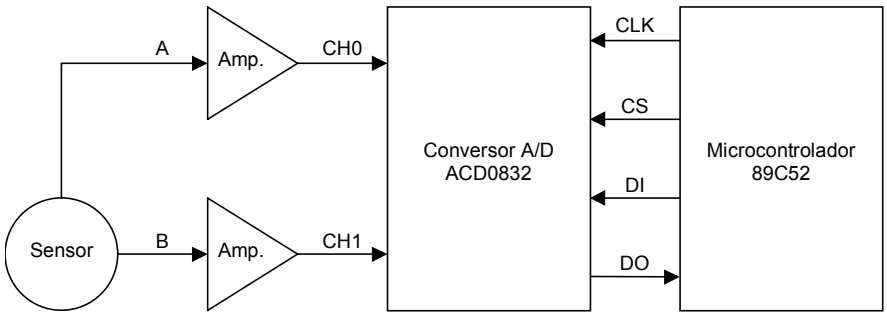
Title	TECLADO MATRICIAL	
Size A3	Number	Rev 1.0
Date 03/2002		Drawn by HUGO
Filename TECMAT51.S01		Sheet 1 of 1

```
;*********************************************************
; TECLA - varre colunas e verifica linhas do teclado
; ENTRADA: nada
; SAIDA: A = codigo ASCII da tecla pressionada
; DESTROI: nada
;*********************************************************
TECLA:
COL0:     MOV  P1, #11110111B      ;coluna 0
          MOV  A, P1
          CJNE A, #11110111B, CL00
COL1:     MOV  P1, #11111011B      ;coluna 1
          MOV  A, P1
          CJNE A, #11111011B, CL10
COL2:     MOV  P1, #11111101B      ;coluna 2
          MOV  A, P1
          CJNE A, #11111101B, CL20
COL3:     MOV  P1, #11111110B      ;coluna 3
          MOV  A, P1
          CJNE A, #11111110B, CL30
          RET

CL00:     CJNE A, #11100111B, CL01 ;coluna 0, linha 0
          MOV  A, #'1'
          RET
CL01:     CJNE A, #11010111B, CL02 ;coluna 0, linha 1
          MOV  A, #'5'
          RET
CL02:     CJNE A, #10110111B, CL03 ;coluna 0, linha 2
          MOV  A, #'9'
          RET
CL03:     CJNE A, #01110111B, VT03 ;coluna 0, linha 3
          MOV  A, #'C'
VT03:     RET

CL10:     CJNE A, #11101011B, CL11 ;coluna 1, linha 0
          MOV  A, #'2'
          RET
CL11:     CJNE A, #11011011B, CL12 ;coluna 1, linha 1
          MOV  A, #'6'
          RET
CL12:     CJNE A, #10111011B, CL13 ;coluna 1, linha 2
          MOV  A, #'0'
          RET
CL13:     CJNE A, #01111011B, VT13 ;coluna 1, linha 3
          MOV  A, #'D'
VT13:     RET

CL20:     CJNE A, #11101101B, CL21 ;coluna 2, linha 0
          MOV  A, #'3'
          RET
```

```
CL21:     CJNE A, #11011101B, CL22 ;coluna 2, linha 1
          MOV  A, #'7'
          RET
CL22:     CJNE A, #10111101B, CL23 ;coluna 2, linha 2
          MOV  A, #'A'
          RET
CL23:     CJNE A, #01111101B, VT23 ;coluna 2, linha 3
          MOV  A, #'E'
VT23:     RET

CL30:     CJNE A, #11101110B, CL31 ;coluna 3, linha 0
          MOV  A, #'4'
          RET
CL31:     CJNE A, #11011110B, CL32 ;coluna 3, linha 1
          MOV  A, #'8'
          RET
CL32:     CJNE A, #10111110B, CL33 ;coluna 3, linha 2
          MOV  A, #'B'
          RET
CL33:     CJNE A, #01111110B, VT33 ;coluna 3, linha 3
          MOV  A, #'F'
VT33:     RET
```

Exercícios

1) Determinado sensor fornece dois sinais analógicos A e B, que variam de 0 a 5V. Deseja-se utilizar o conversor A/D ADC0832 para realizar a aquisição destes sinais por um microcontrolador 89C51, conforme o diagrama em blocos mostrado a seguir. Implemente um programa em Assembly MCS51 capaz de gerar e ler os sinais necessários ao funcionamento adequado do conversor A/D, armazenando o valor da conversão do canal CH0 no registro R0 e o valor da conversão do canal CH1 no registro R1.



2) Dado o diagrama esquemático do conversor A/D (R2R) mostrado anteriormente, implemente um programa que realize uma conversão A/D por aproximação sucessiva.

3) O programa de controle do teclado matricial apresentado trata apenas os casos em que apenas uma tecla é pressionada a cada instante de tempo. Implemente um programa de controle que tenha uma tecla “shift”, que quando pressionada simultaneamente com outra das 15 teclas é capaz de alterar o seu significado.

4) O sensor de temperatura LM74 da National Semiconductor é um periférico que possui interface serial para se comunicar com microcontroladores. Consulte o manual do LM74 e idealize um sistema de hardware e software para o 8031 capaz de ler valores de temperatura fornecidos por este sensor através do port P1.

5) Alguns módulos LCD inteligentes oferecem a possibilidade de operar com um barramento de dados de 4 bits. Consulte o manual de módulos LCD inteligentes (circuito integrado controlador HD44780) e idealize um sistema de hardware e software para o 8031 capaz de acionar este dispositivo através de um barramento de dados de 4 bits e demais sinais de controle necessários, todos implementados no port P1.

INTERRUPÇÕES

Os sinais de interrupção possibilitam a parada da execução do processamento em andamento para o atendimento imediato a eventos internos ou externos de maior prioridade.

Para que uma interrupção seja atendida, a mesma deverá estar devidamente habilitada pelo software e na possibilidade de ocorrência de mais de uma interrupção simultaneamente, existe uma hierarquia de prioridade de atendimento.

São cinco as fontes de interrupção do 8051, sendo duas externas e três internas:

- Interrupção INT0\ (externa)
- Interrupção INT1\ (externa)
- Temporizador / contador de eventos T0 (interna)
- Temporizador / contador de eventos T1 (interna)
- Interface serial (interna)

No Port 3 existem quatro terminais relacionados a interrupções:

- P3.0 – recepção serial RXD
- P3.1 – transmissão serial TXD
- P3.2 – entrada da interrupção externa INT0\
- P3.3 – entrada da interrupção externa INT1\

Se forem usadas quaisquer das funções especiais do Port 3, o mesmo deverá ser acessado apenas através dos seus bits individuais.

No 8051 pode-se habilitar individualmente cada interrupção, cada qual com dois níveis de prioridade definidos por software. Existe um mecanismo interno de prioridade que define a seguinte ordem de atendimento, na remota possibilidade de ocorrência simultânea:

- Interrupção externa 0 (maior prioridade)
- Temporizador / contador de eventos 0
- Interrupção externa 1
- Temporizador / contador de eventos 1
- Interface serial (menor prioridade)

Quando uma interrupção é atendida, o valor do registrador PC é salvo na pilha, de forma a possibilitar o posterior retorno do programa ao ponto em que havia parado. Deve-se atentar para o fato de nenhum outro registro ser salvo na pilha, nem mesmo o acumulador ou o PSW. Fica por conta da rotina de atendimento da interrupção o salvamento dos registros que não podem ser perdidos.

Vetores de Interrupção

Interrupção	Vetor
Interrupção externa 0	0003H
Temporizador/contador de eventos 0	000BH
Interrupção externa 1	0013H
Temporizador/contador de eventos 1	001BH
Interface serial	0023H

Mapa da Faixa Inicial da Memória

0000H	RESET
0002H	
0003H	INT0\
000AH	
000BH	T/C0
0012H	
0013H	INT1\
001AH	
001BH	T/C1
0022H	
0023H	SERIAL

No endereço de reset existem apenas três bytes de memória antes que se comece a invadir o espaço destinado ao vetor da interrupção externa 1. Normalmente utilizam-se estes três bytes para uma instrução de desvio incondicional do programa para outra faixa da memória de programa.

De maneira similar, nos vetores das interrupções INT0\, T/C0, INT1\ e T/C1 estão disponíveis apenas 8 bytes de memória, antes da superposição com o vetor da próxima interrupção. A única exceção corresponde ao vetor da interrupção da interface serial, que por ser o último não se sobrepõe a nenhum outro.

Sempre que uma interrupção é requisitada, um bit de controle relativo a essa interrupção é setado, sendo resetado por hardware quando a interrupção é atendida. A única exceção ocorre com a interrupção da interface serial, cujo bit de controle deve ser resetado pelo software.

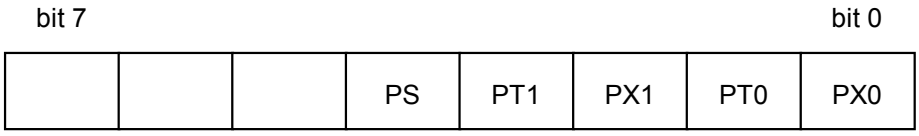
Registros Especiais de Controle das Interrupções

- **Registro IE – Interrupt Enable:** Esse registro tem a função de habilitar ou desabilitar o atendimento das interrupções do 8051.

EA – Enable All
Nível lógico 0 - desabilita todas as interrupções, independentemente de qualquer outro bit de controle.
Nível lógico 1 - permite a habilitação particular de cada interrupção, se o estado do seu respectivo bit de controle individual for igual a 1.

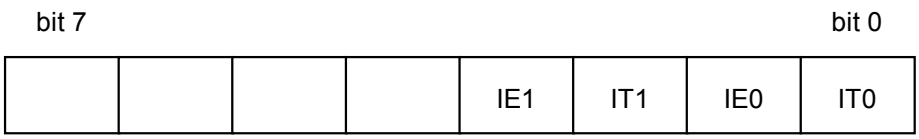
ES – Enable Serial; ET1 – Enable Timer 1; EX1 – Enable External 1; ET0 – Enable Timer 0; EX1 – Enable External 0
Nível lógico 0 - desabilitam as interrupções correspondentes, independente do estado do bit de controle EA.
Nível lógico 1 - habilitam as interrupções correspondentes, se o estado do bit de controle EA também for igual a 1.

- **Registro IP – Interrupt Priority:** Esse registro tem a função de alterar a prioridade de atendimento das interrupções do 8051.



PS – Priority Serial; PT1 – Priority Timer 1; PX1 – Priority External 1; PT0 – Priority Timer 0; PX1 – Priority External 0
Nível lógico 0 – prioridade baixa para a interrupção correspondente.
Nível lógico 1 – prioridade alta para a interrupção da interrupção correspondente.

- **Registro TCON – Timer Control:** Os quatro bits menos significativos desse registro permitem a programação da maneira como as interrupções externas 1 e 0 serão reconhecidas (borda ou nível lógico).



IE1, IE0
Sinalizam internamente as requisições das interrupções externas 1 e 0, respectivamente. São setados quando ocorrem as respectivas interrupções e são zerados por hardware assim que as mesmas são atendidas.

IT1 – Interrupt Transition 1; IT0 – Interrupt Transition 0
Indicam quais os processos de chamada das interrupções externas 1 e 0, respectivamente. Se em nível lógico 1, as interrupções serão aceitas quando ocorrer uma borda de descida nos terminais INT1\ ou INT0\. Se em nível lógico 0, as interrupções serão aceitas apenas pelo nível lógico 0 presente nos terminais INT1\ ou INT0\.

Quando sensíveis a transição, os terminais de interrupção INT1\ e INT0\ são amostrados duas vezes, sendo o período entre as amostragens de 12 ciclos de clock. A requisição de interrupção ocorre quando for detectada uma mudança de nível lógico 1 para nível lógico 0 entre duas amostragens consecutivas.

Quando sensíveis a nível lógico, a amostragem dos terminais de interrupção INT1\ e INT0\ ocorre ao final de cada instrução executada. Uma vez detectado nível lógico 0, este poderá permanecer durante a execução da rotina de atendimento da interrupção, mas deverá retornar a nível lógico 1 ao final desta, caso contrário o sistema atenderá novamente à interrupção. Essa característica permite artifícios para se conseguir a execução de programas passo-a-passo.

É possível a utilização dos bits sinalizadores de interrupção sem que as mesmas estejam habilitadas. Para tanto basta a monitoração do estado desses bits pelo software. À essa técnica de monitoração sem a ocorrência de interrupções dá-se o nome de polling.

Rotinas de Tratamento de Interrupção

As rotinas de tratamento de interrupção não devem alterar o estado dos registradores que estão sendo utilizados no programa principal, a não ser que isso seja explicitamente intencional. Normalmente faz-se uso da pilha para preservar o conteúdo de registros de função especial que eventualmente sejam manipulados durante as rotinas de tratamento de interrupção. No caso dos registros, é comum aproveitar o recurso de chaveamento de bancos de registro que a família MCS51 possui.

Abaixo temos um exemplo de rotina de tratamento para a interrupção externa 1:

```
          ORG  0013H     ;vetor da interrupcao externa 1
EXT1:     PUSH ACC       ;salva acumulador
          PUSH PSW       ;salva estado atual do programa
          MOV  PSW, #08H ;muda para o banco de registros 1
          .
          .              ;detalhes específicos da rotina
          .
          POP  PSW       ;recupera estado anterior do prog.
          POP  ACC       ;recupera acumulador
          RETI
```

As rotinas de tratamento de interrupção são bastante similares às subrotinas. No entanto, não devemos nos esquecer que as rotinas de tratamento de interrupção devem ser terminadas com a instrução `RETI` e que

são chamadas por hardware, enquanto as subrotinas devem ser terminadas com a instrução `RET` e são chamadas por software.

Exercícios

1) Determinado sistema de controle baseado no 80C31 necessita de duas fontes de interrupção: a interrupção da interface serial e a interrupção externa 0. Elabore um trecho de programa em Assembly que habilite essas interrupções, determinando a interrupção da interface serial como mais prioritária. Programe também a interrupção externa 0 para sensibilidade a borda de descida.

2) Dado o seguinte trecho de programa em Assembly para um sistema microcontrolado baseado no 80C32, responda as questões abaixo:

```
          ORG  0003H
EXT0:     PUSH PSW
          SETB RS0
          MOV  A, #20H
          MOV  R0, #40H
          POP  PSW
          RETI

          ORG  0013H
EXT1:     PUSH ACC
          MOV  A, #20H
          MOV  R0, #10H
          POP  ACC
          RETI
          ORG  0100H
PROG:     CLR  RS0              ;o programa começa aqui!
          MOV  TCON, #00000101B
          MOV  IP, #00000100B
          MOV  IE, #10000001B
          MOV  R0, #50H         ;instrução X
          MOV  A, #05H          ;instrução Y
          ADD  A, R0            ;instrução Z
FINAL:    JMP  FINAL
```

a) Supondo que ocorre uma borda de descida no sinal aplicado ao terminal P3.2 (INT0) durante a execução da instrução X, qual será o valor contido no acumulador ao final do programa?

b) Supondo que ocorre uma borda de descida no sinal aplicado ao terminal P3.2 (INT0) durante a execução da instrução Y, qual será o valor contido no acumulador ao final do programa?

c) Supondo que ocorrem bordas de descida nos sinais aplicados aos terminais P3.2 (INT0) e P3.3 (INT1) simultaneamente durante a execução da instrução Z, qual será o valor contido no acumulador ao final do programa?

3) Dado o diagrama esquemático do conversor A/D (R2R) mostrado anteriormente, implemente um programa que realize uma conversão A/D por rampa escalonada decrescente, utilizando a interrupção externa 0 como critério de parada.

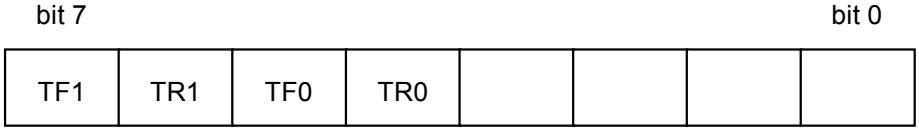
TEMPORIZADORES / CONTADORES DE EVENTOS

Os temporizadores / contadores de eventos (T/C) são periféricos geralmente empregados na geração periódica de pedidos de interrupção, sendo extremamente úteis em sistemas de controle em tempo real. Também são utilizados na contagem ou medição de largura de pulsos externos, contagem de tempo, geração de sinais digitais modulados em largura de pulso (PWM), entre outras aplicações.

O 8051 possui internamente dois T/C programáveis por software e que operam de modo completamente independente dos demais componentes do microcontrolador. O seu funcionamento pode ser habilitado ou desabilitado por software, através de bits de controle em registros especiais, ou hardware, através de terminais externos.

Registros Especiais de Controle dos T/C:

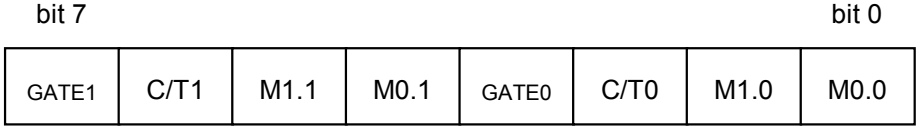
- **Registro TCON – Timer Control:** Esse registro tem a função de habilitar ou desabilitar o funcionamento dos T/C do 8051.



TR1 – Timer Release 1, TR0 – Timer Release 0
Nível lógico 0 – desliga a contagem do T/C correspondente.
Nível lógico 1 – dispara a contagem do T/C correspondente.

TF1 – Timer Flag 1, TF0 – Timer Flag 0
Sempre que ocorrer um estouro da capacidade de contagem (overflow) de um T/C, o bit correspondente será setado, requisitando um pedido de interrupção e sendo zerado novamente ao final da rotina de atendimento.

- **Registro TMOD – Timer Mode:** Esse registro tem a função de programar o comportamento dos T/C do 8051.



GATE1; GATE0
Nível lógico 0 – a contagem do T/C será habilitada se apenas o bit de controle correspondente (TR1 ou TR0) no registro TCON estiver em nível lógico 1.
Nível lógico 1 – a contagem do T/C será habilitada se o bit de controle correspondente (TR1 ou TR0) no registro TCON estiver em nível lógico 1 **e** o terminal de interrupção correspondente (INT1\ ou INT0\) também estiver em nível lógico 1.

Esses bits de controle são úteis para realizar a medição de largura de pulsos externos, colocando-se os sinais de interesse nos terminais de interrupção do 8051. Desse modo, a contagem ocorrerá somente quando o nível lógico do sinal externo for 1.

C/T1 – Counter / Timer 1; C/T0 – Counter / Timer 0
Nível lógico 0 – o T/C correspondente funciona como temporizador (sinal de contagem interno – freqüência de clock dividida por 12).
Nível lógico 1 – o T/C correspondente funciona como contador (sinal de contagem externo – terminal T1 ou T0 do Port3).

M1.1/M0.1 – Mode T/C1; M0.1/M0.0 – Mode T/C0
Permitem a obtenção de quatro modos de operação distintos para cada T/C.

Modos de Operação dos T/C

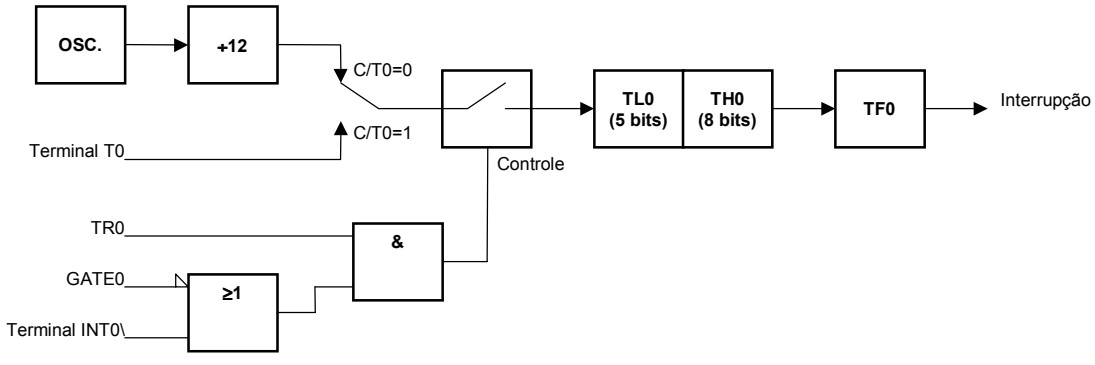
- **Modo 0 (M1.x=0 e M0.x=0):** Temporizador / Contador de 8 bits com divisor de freqüência (prescaler) de 5 bits.

Nesse modo de operação os 5 bits menos significativos dos registros TL1 ou TL0 funcionam como divisor de freqüência por valores que vão de 2 a 32.

Os registros TH1 ou TH0 são programados por software com o valor inicial da contagem. Os valores presentes nesses registros podem ser lidos a qualquer momento. Ao ocorrer estouro na contagem (passagem de FFH para 00H) ocorrerá o pedido da interrupção correspondente, se esta estiver habilitada.

Cabe à rotina de atendimento da interrupção fazer a recarga dos registros TH1 ou TH0 com os valores adequados para o reinício da contagem.

Operação no Modo 0



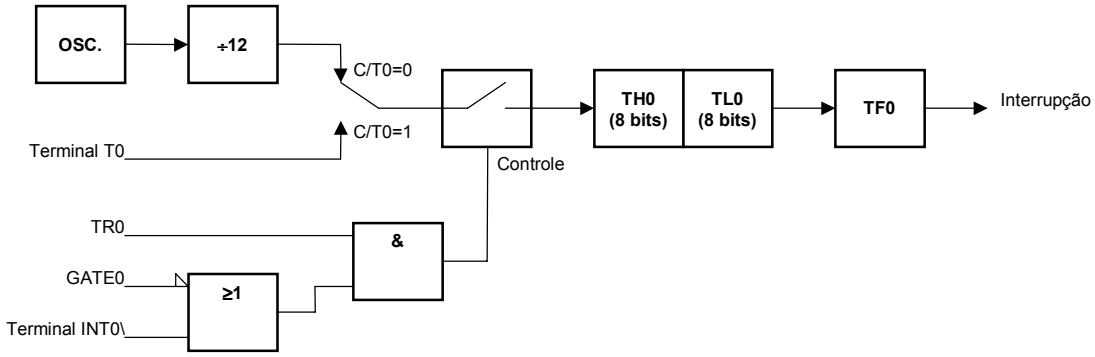
- **Modo 1 (M1.x=0 e M0.x=1):** Temporizador / Contador de 16 bits.

Nesse modo de funcionamento uma contagem de 16 bits é realizada nos pares de registros TH1/TL1 ou TH0/TL0.

De maneira análoga ao Modo 0, o valor inicial da contagem também pode ser programado por software e ao ocorrer estouro na contagem (passagem de FFFFH para 0000H) também ocorre um pedido de interrupção.

Nesse caso também cabe à rotina de atendimento da interrupção fazer a recarga dos registros TH1/TL1 ou TH0/TL0 com os valores adequados para o reinício da contagem.

Operação no Modo 1

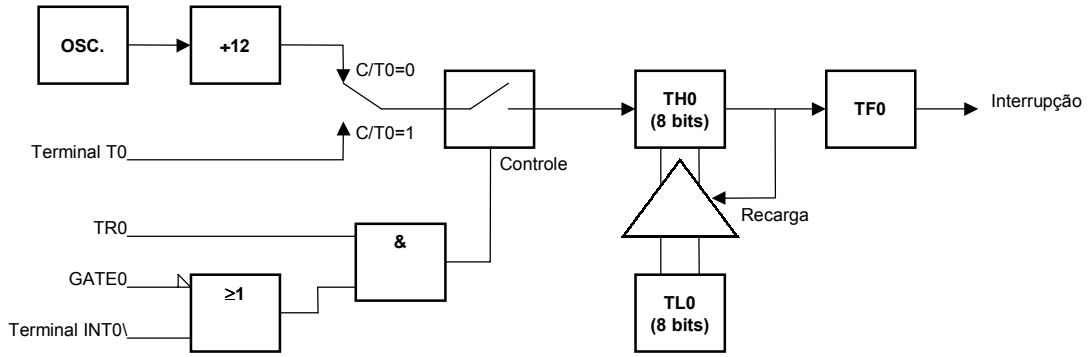


- **Modo 2 (M1.x=1 e M0.x=0):** Temporizador / Contador de 8 bits com recarga automática.

A contagem ocorre nos registros TL1 ou TL0, sendo os registros TH1 ou TH0 utilizados para armazenar os valores de recarga automática de TL1 ou TL0 ao ocorrer estouro na contagem e a conseqüente requisição de interrupção. Nessa forma de operação não há necessidade de rescrever o valor inicial da contagem como no Modo 0.

O T/C1 quando programado no Modo 2 serve para gerar a taxa de transmissão e recepção da interface serial.

Operação no Modo 2



- **Modo 3 (M1.x=1 e M0.x=1):** Duplo Temporizador / Contador de 8 bits.

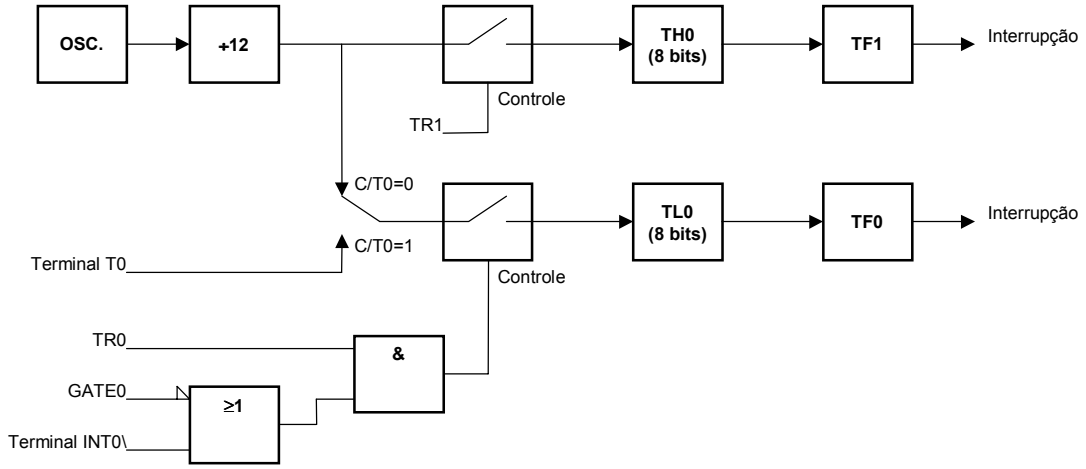
Esse modo de funcionamento é útil somente para o T/C0. Se o T/C1 for programado nesse modo ficará inerte.

No Modo 3 tem-se dois temporizadores / contadores independentes de 8 bits nos registros TH0 e TL0. O controle da contagem em TH0 é feito pelos bits TR1 e TF1, enquanto que o controle da contagem em TL0 é feito pelos bits TR0 e TF0 do registro TCON.

Uma vez programado o T/C0 no Modo 3, pode-se programar o T/C1 em qualquer um dos outros modos, mas este não irá gerar pedidos de interrupção,

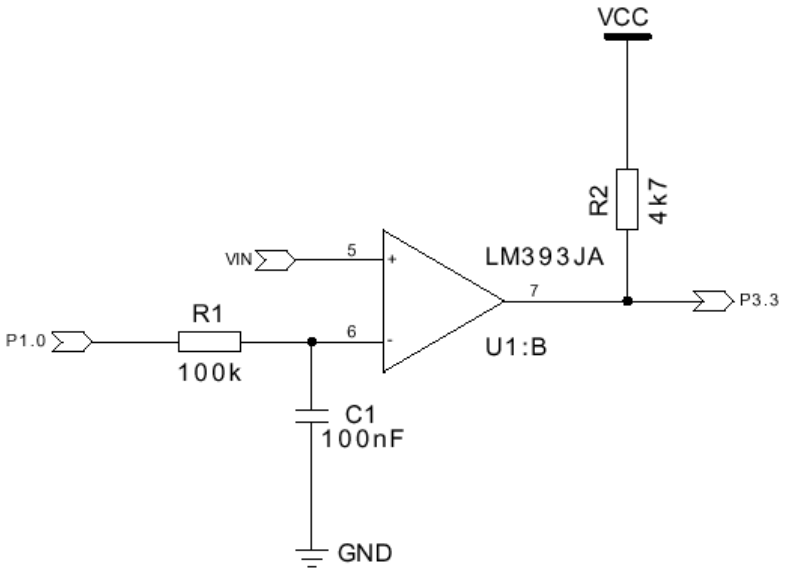
pois o bit TF1 estará sendo utilizado pelo contador em TH0. Entretanto, mesmo assim o T/C1 poderá ser utilizado para a geração da taxa de transmissão e recepção da interface serial do 8051.

Operação no Modo 3



Geração de Sinais Modulados em Largura de Pulso (PWM)

O diagrama esquemático a seguir mostra a implementação de um conversor A/D discreto controlado por software. O princípio de funcionamento consiste na geração de um sinal PWM no bit 0 do port P1. Os componentes R1 e C1 compõem um filtro passa-baixas, cuja função é a de extrair o nível médio do sinal PWM, que é proporcional ao seu ciclo de trabalho. A tensão analógica resultante dessa filtragem será comparada com o sinal de entrada que se deseja medir, possibilitando que o microcontrolador seja realimentado com essa informação através do bit 3 do Port3.



A seguir é apresentado um programa completo em Assembly da família MCS51, capaz de gerar um sinal modulado em largura de pulso no bit 0 do port P1 do 8031. A subrotina SETPWM é a responsável pela determinação do ciclo

de trabalho do sinal gerado, que no caso do exemplo a seguir é fixo em 25%, pois o registro R0 possui o valor 40H.

```
;*********************************************************
; Descricao dos terminais
;*********************************************************
PWM       EQU  090H      ;P1.0 = sinal PWM
CMP       EQU  0B3H      ;P3.3 = saida do comparador

;*********************************************************
; Declaracoes de constantes
;*********************************************************
PRESC     EQU  00001000B ;prescaler do temporizador 0
                         ;determina a frequencia do PWM

;*********************************************************
; Declaracoes de variaveis
;*********************************************************
TALTO     EQU  020H      ;tempo alto do sinal PWM
TBAIXO    EQU  021H      ;tempo baixo do sinal PWM
STACK     EQU  021H      ;topo da pilha

;*********************************************************
; Vetores das interrupcoes
;*********************************************************
          ORG  0003H
EX0:      RETI
          ORG  000BH
TC0:      JMP  LARG      ;geracao do sinal PWM
          ORG  0013H
EX1:      RETI
          ORG  001BH
TC1:      RETI
          ORG  0023H
SER:      RETI
          ORG  002BH
TC2:      RETI

;*********************************************************
; Programa principal
;*********************************************************
          ORG  0100H
INICIO:   MOV  SP, #STACK          ;muda o topo da pilha

          SETB CMP                 ;CMP = entrada
          SETB PWM
          CLR  TR0                 ;desliga T/C0
          MOV  TMOD, #11110000B
          MOV  TL0, #PRESC
          MOV  IE, #10000010B      ;habilita interrup. T/C0
```

```
          MOV R0, #40H
          CALL SETPWM

VOLTA:    JMP VOLTA                 ;laco infinito

;*********************************************************
; Rotinas de tratamento de interrupcao
;*********************************************************
;*********************************************************
; LARG - geracao do sinal PWM
;*********************************************************
LARG:     JNB  PWM, ALTO      ;verifica estado do bit PWM
          MOV  TH0, TALTO     ;se alto carrega TALTO
          CLR  PWM            ;bit PWM = 0
          RETI

ALTO:     MOV  TH0, TBAIXO    ;se baixo carrega TBAIXO
          SETB PWM            ;bit PWM = 1
          RETI

;*********************************************************
; Subrotinas
;*********************************************************
;*********************************************************
; SETPWM - define a largura do sinal PWM
;*********************************************************
SETPWM:   CLR  C
          MOV  A, #0FFH
          SUBB A, R0          ;obtem TBAIXO de TALTO

          JNB  TR0, SINC      ;verifica se T/C0 esta ligado

WAIT0:    JB   PWM, WAIT0
WAIT1:    JNB  PWM, WAIT1     ;aguarda borda de subida

SINC:     CJNE R0, #0FFH, NMAX     ;se R0 = maximo ...
          CLR  TR0                 ;... desliga T/C0 e ...
          SETB PWM                 ;... mantem PWM = 1
          RET

NMAX:     CJNE R0, #00H, NMIN      ;se R0 = minimo ...
          CLR  TR0                 ;... desliga T/C0 e ...
          CLR  PWM                 ;... mantem PWM = 0
          RET

NMIN:     MOV  TALTO, R0           ; carrega TALTO e TBAIXO
          MOV  TBAIXO, A
          SETB TR0                 ;liga T/C0
          RET
```

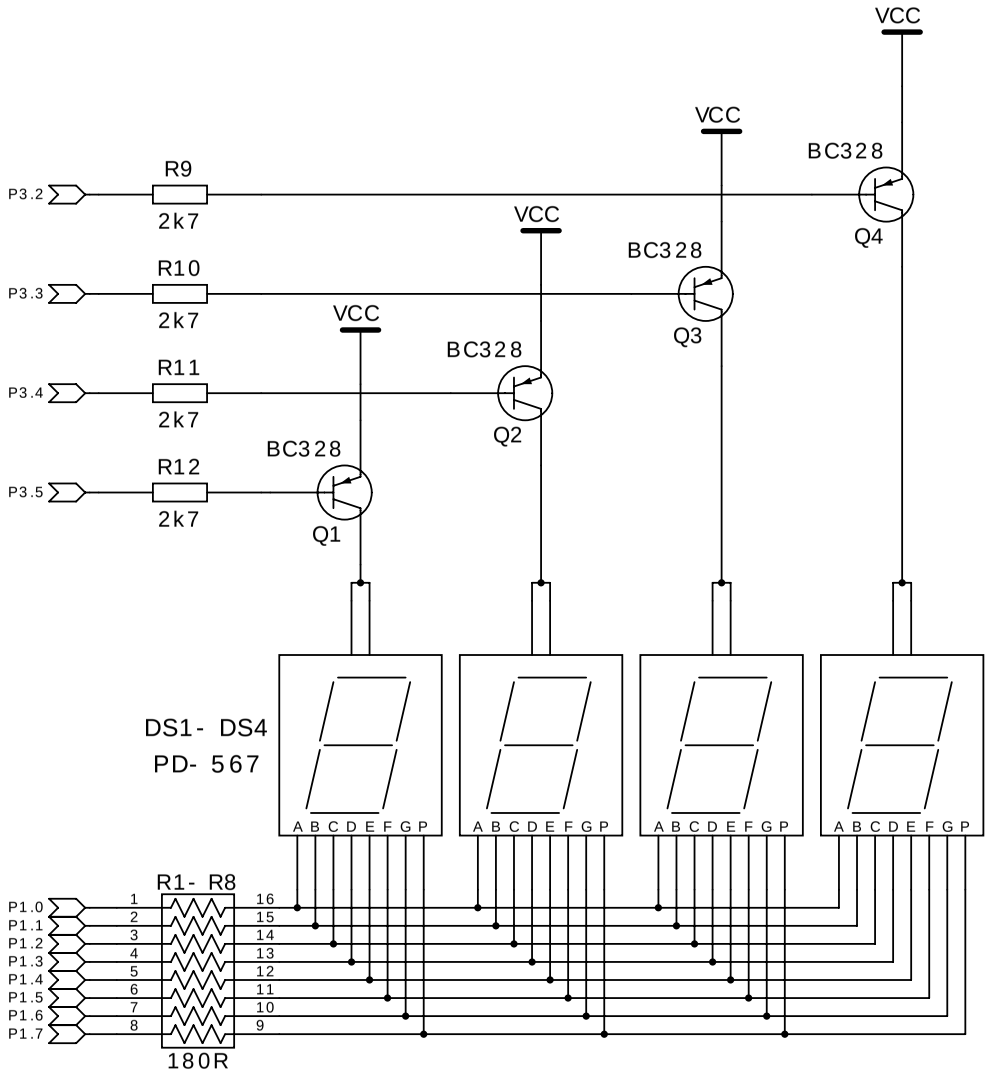
```
;*********************************************************
; ATRASO - Gera atraso de tempo (R1)
; T = (R0 * (36 + R1 * 48)) / CLOCK
;*********************************************************
ATRASO:   MOV  R2, #230          ;12 ciclos de maquina
LOOP:     NOP                    ;12 ciclos de maquina
          NOP                    ;12 ciclos de maquina
          DJNZ R2, LOOP          ;24 ciclos de maquina
          DJNZ R1, ATRASO        ;24 ciclos de maquina
          RET
```

Multiplexação de Displays de Sete Segmentos

Na página seguinte é apresentado o diagrama esquemático de um sistema de multiplexação de displays de sete segmentos.

Como exemplo de aplicação, o programa em Assembly mostrado logo em seguida implementa um cronômetro regressivo no hardware apresentado. A varredura dos displays é feita através de interrupções periódicas do T/C0 e a contagem regressiva do tempo é realizada por interupções periódicas do T/C1 do 8031.

```
;*********************************************************
; Declaracoes de hardware
;*********************************************************
SEGB      EQU  090H          ;P1.0 = segmento b
SEGA      EQU  091H          ;P1.1 = segmento a
SEGF      EQU  092H          ;P1.2 = segmento f
SEGG      EQU  093H          ;P1.3 = segmento g
SEGE      EQU  094H          ;P1.4 = segmento e
SEGD      EQU  095H          ;P1.5 = segmento d
SEGC      EQU  096H          ;P1.6 = segmento c
PD        EQU  097H          ;P1.6 = ponto decimal

DS0       EQU  0B2H          ;P3.2 = display 0
DS1       EQU  0B3H          ;P3.3 = display 1
DS2       EQU  0B4H          ;P3.4 = display 2
DS3       EQU  0B5H          ;P3.5 = display 3

;*********************************************************
; Declaracoes de constantes (CLOCK = 11.0592 MHz)
;*********************************************************
NITC      EQU  15            ;numero de ints para 1 seg
                             ;NITC=INT(CLOCK/(12*65536))+1
TEMPO     EQU  61440         ;tempo de 1/NITC segundos
                             ;TEMPO = CLOCK/(12*NITC)
VARRE     EQU  1200H         ;tempo de varredura dos disp
```



Title	MULTIPLEXACAO DE DISPLAYS DE 7 SEGMENTOS	
Size A4	Number	Rev 1.0
Date 03/2002		Drawn by HUGO
Filename DISMUL51.S01		Sheet 1 of 1

```
;*********************************************************
; Declaracoes de variaveis
;*********************************************************
BUFF0     EQU  020H          ;buffer do display 0
BUFF1     EQU  021H          ;buffer do display 1
BUFF2     EQU  022H          ;buffer do display 2
BUFF3     EQU  023H          ;buffer do display 3
VARDS     EQU  024H          ;estado da varredura dos disp
MIN       EQU  025H          ;contagem dos minutos
SEG       EQU  026H          ;contagem dos segundos
FRAC      EQU  027H          ;contagem das frac de segundo
STACK     EQU  028H          ;topo da pilha

;*********************************************************
; Vetores das interrupcoes
;*********************************************************
          ORG  0003H
EX0:      RETI
          ORG  000BH
TC0:      JMP  SCAND         ;varredura dos displays
          ORG  0013H
EX1:      RETI
          ORG  001BH
TC1:      JMP  COUNT         ;contagem regressiva
          ORG  0023H
SER:      RETI
          ORG  002BH
TC2:      RETI

;*********************************************************
; Programa principal
;*********************************************************
          ORG  0100H
INICIO:   MOV  SP, #STACK          ;muda o topo da pilha
          MOV  PSW, #00000000B     ;banco de registros 0

          MOV  VARDS, #11011101B   ;inicializa varredura

          MOV  MIN, #02H           ;inicializa minutos
          MOV  SEG, #30H           ;inicializa segundos
          MOV  FRAC, #NITC         ;inicializa frac de seg

          CALL DISPM          ;atualiza displays de min
          CALL DISPS          ;atualiza displays de seg

          MOV  TCON, #00000000B    ;desliga temporizadores
          MOV  TMOD, #00010001B    ;T/C0 e T/C1 no modo 1
          MOV  TH0, #HIGH(65536-VARRE)
          MOV  TL0, #LOW (65536-VARRE)
          MOV  TH1, #HIGH(65536-TEMPO)
          MOV  TL1, #LOW (65536-TEMPO)
```

```
          MOV  IE, #10001010B      ;habilita ints dos T/C
          MOV  TCON, #01010000B    ;habilita contagens

VOLTA:    JMP  VOLTA               ;espera interrupcoes

;*********************************************************
; Rotinas de tratamento de interrupcao
;*********************************************************
;*********************************************************
; Temporizador 0 - varredura dos displays multiplexados
;*********************************************************
SCAND:    PUSH ACC
          PUSH PSW
          MOV  PSW, #00001000B     ;banco de registros 1

          MOV  TH0, #HIGH(65536-VARRE)  ;recarrega TH0

          MOV  A, VARDS
          RL   A                   ;passa para proximo disp
          MOV  VARDS, A

          JB   ACC.2, PROX0
          SETB DS3                 ;desabilita display 3
          MOV  P1, #0FFH
          CLR  DS0                 ;habilita display 0
          MOV  P1, BUFF0           ;atualiza display 0
          JMP  FIMT0

PROX0:    JB   ACC.3, PROX1
          SETB DS0                 ;desabilita display 0
          MOV  P1, #0FFH
          CLR  DS1                 ;habilita display 1
          MOV  P1, BUFF1           ;atualiza display 1
          JMP  FIMT0

PROX1:    JB   ACC.4, PROX2
          SETB DS1                 ;desabilita display 1
          MOV  P1, #0FFH
          CLR  DS2                 ;habilita display 2
          MOV  P1, BUFF2           ;atualiza display 2
          JMP  FIMT0

PROX2:    SETB DS2                 ;desabilita display 2
          MOV  P1, #0FFH
          CLR  DS3                 ;habilita display 3
          MOV  P1, BUFF3           ;atualiza display 3

FIMT0:    POP  PSW
          POP  ACC
          RETI
```

```
;*********************************************************
; Temporizador 1 - contagem regressiva
;*********************************************************
COUNT:    PUSH ACC
          PUSH PSW
          MOV  PSW, #00010000B     ;banco de registros 2

          MOV  TH1, #HIGH(65536-TEMPO)  ;recarrega TH1

          DEC  FRAC
          MOV  A, FRAC
          CJNE A, #00H, FIMT1      ;conta NITC (1 segundo)
          MOV  FRAC, #NITC

          MOV  A, SEG
          DEC  A                   ;decrementa segundos
          CALL DAASUB              ;faz o ajuste decimal
          MOV  SEG, A
          CJNE A, #99H, ATSEG      ;se chegou a 99 ...
          MOV  SEG, #59H           ;... corrige para 59

          MOV  A, MIN
          CJNE A, #00H, MINUTO     ;verifica zerou segundos

          MOV  SEG, #00H
          CLR  TR1                 ;para contagem se 00:00
          JMP  FIMT1

MINUTO:   DEC  A                   ;decrementa minutos
          CALL DAASUB              ;faz o ajuste decimal
          MOV  MIN, A

          CALL DISPM               ;atualiza minutos
ATSEG:    CALL DISPS               ;atualiza segundos

FIMT1:    POP  PSW
          POP  ACC
          RETI

;*********************************************************
; Subrotinas
;*********************************************************
;*********************************************************
; DAASUB - ajuste decimal do acumulador apos subtracoes
;*********************************************************
DAASUB:   PUSH PSW
          PUSH A

          ANL  A, #0FH
          CLR  C
          SUBB A, #0AH
```

```
          JC  SEM0       ;inferior precisa de ajuste?
          POP  A
          CLR  C
          SUBB A, #06H   ;ajusta nibble inferior
          PUSH A
SEM0:     CPL  C         ;nao ajusta nibble inferior
          POP  A
          PUSH A
          ANL  A, #F0H
          CLR  C
          SUBB A, #A0H
          JC   SEM1      ;superior precisa de ajuste?
          POP  A
          CLR  C
          SUBB A, #60H   ;ajusta nibble superior
          PUSH A
SEM1:     CPL  C         ;nao ajusta nibble superior

          POP  A
          POP  PSW
          RET

;*********************************************************
; DISPM - atualiza os displays de minutos
;*********************************************************
DISPM:    MOV  DPTR, #TAB7S        ;aponta para a tabela

          MOV  A, MIN              ;recupera minutos
          SWAP A
          ANL  A, #0FH
          MOVC A, @A+DPTR
          MOV  BUFF0, A            ;atualiza dezenas

          MOV  A, MIN              ;recupera minutos
          ANL  A, #0FH
          MOVC A, @A+DPTR
          ANL  A, #01111111B       ;ponto decimal
          MOV  BUFF1, A            ;atualiza unidades

          RET

;*********************************************************
; DISPS - atualiza os displays de segundos
;*********************************************************
DISPS:    MOV  DPTR, #TAB7S        ;aponta para a tabela

          MOV  A, SEG              ;recupera segundos
          SWAP A
          ANL  A, #0FH
          MOVC A, @A+DPTR
          MOV  BUFF2, A            ;atualiza dezenas
```

```
          MOV  A, SEG              ;recupera segundos
          ANL  A, #0FH
          MOVC A, @A+DPTR
          MOV  BUFF3, A            ;atualiza unidades

          RET

;*********************************************************
; Tabela de decodificação para os displays de 7 segmentos
;*********************************************************
;                              0 1 2 3 4 5 6 7 8 9
; P1.0 = segmento b      -     0 0 0 0 0 1 1 0 0 0
; P1.1 = segmento a      -     0 1 0 0 1 0 0 0 0 0
; P1.2 = segmento f      -     0 1 1 1 0 0 0 1 0 0
; P1.3 = segmento g      -     1 1 0 0 0 0 0 1 0 0
; P1.4 = segmento e      -     0 1 0 1 1 1 0 1 0 1
; P1.5 = segmento d      -     0 1 0 0 1 0 0 1 0 0
; P1.6 = segmento c      -     0 0 1 0 0 0 0 0 0 0
; P1.7 = ponto decimal   -     1 1 1 1 1 1 1 1 1 1
;*********************************************************
;              pcdegfab
TAB7S:    DB   10001000B           ;algarismo 0
          DB   10111110B           ;algarismo 1
          DB   11000100B           ;algarismo 2
          DB   10010100B           ;algarismo 3
          DB   10110010B           ;algarismo 4
          DB   10010001B           ;algarismo 5
          DB   10000001B           ;algarismo 6
          DB   10111100B           ;algarismo 7
          DB   10000000B           ;algarismo 8
          DB   10010000B           ;algarismo 9
```

Exercícios

1) Seguindo a idéia de conversão A/D por filtragem de sinal PWM apresentada anteriormente, implemente um programa em Assembly da família MCS51 que utilize a subrotina SETPWM para realizar uma conversão A/D por rampa escalonada decrescente.

2) Implemente um programa em Assembly da família MCS51 que simule um freqüencímetro. Isto deverá ser feito através da contagem de pulsos que o sinal digital de entrada possui em um intervalo de um segundo. Utilize os temporizadores internos do 8031 para temporização e contagem e utilize um dos bits do port P1 como entrada do sinal digital cuja freqüência deseja-se medir.

3) A subrotina dada a seguir gera um tempo de atraso fazendo uso do T/C0 de um 8031. Supondo que a freqüência de clock do microcontrolador é de 12MHz, calcule o tempo de atraso gerado.

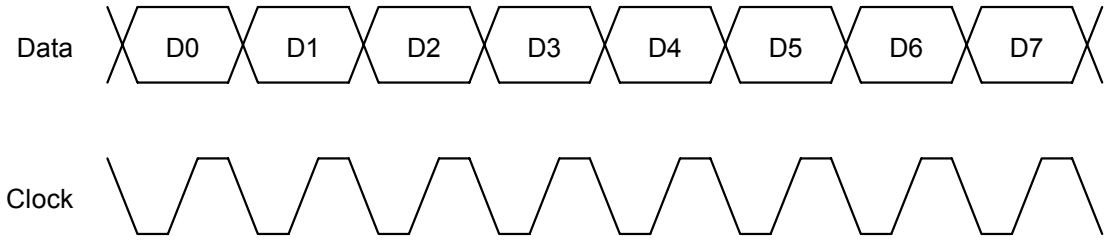
```
;*********************************************************
; ATRASO - gera um tempo de atraso por hardware
; ENTRADA: nada
; SAIDA: tempo de atraso
; DESTROI: nada
;*********************************************************
ATRASO:   MOV  TCON, #00000000B
          MOV  TMOD, #00000001B ;T/C0 no modo 1
          MOV  TH0, #HIGH(1000H);MSB da constante de tempo
          MOV  TL0, #LOW (1000H);LSB da constante de tempo
          SETB TR0              ;dispara contagem do T/C0
ESPERA:   JNB  TF0, ESPERA      ;aguarda estouro
          CLR  TF0              ;para contagem do T/C0
          CLR  TR0              ;reseta estouro
          RET
```

4) Suponha que determinada aplicação empregando o microcontrolador 89C51 exija interrupções periódicas para realizar a varredura de displays de 7 segmentos. Escreva um trecho de programa em Assembly que apenas programe o temporizador 0 no modo 2, de maneira que sejam geradas interrupções a cada 1ms. Esse trecho de programa deve prever também a habilitação da interrupção correspondente e o disparo da contagem do temporizador. A freqüência de clock do microcontrolador é de 6MHz.

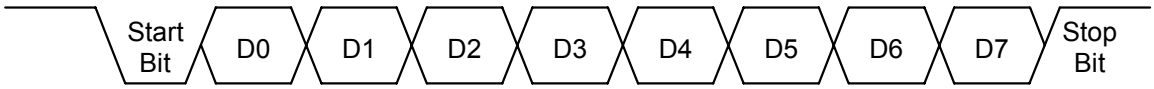
INTERFACE SERIAL

No 8051 pode-se programar a interface serial para operar em modo síncrono half-duplex ou no modo assíncrono full-duplex.

Comunicação Serial Síncrona



Comunicação Serial Assíncrona



A escrita no registro SBUF provoca a transmissão automática do dado escrito através do terminal de transmissão serial (TXD). Por outro lado, dados recebidos através do terminal de recepção serial (RXD) são armazenados no registro SBUF, podendo posteriormente serem lidos pelo software.

Na realidade existem dois registros com o mesmo nome SBUF, sendo um para a transmissão e outro para a recepção. A diferenciação entre eles é feita pelo sistema através das instruções de escrita e leitura (transmissão e recepção, respectivamente) do registro.

Registros Especiais de Controle da Interface Serial

- **Registro SCON – Serial Control:** Esse registro tem a função de controlar e programar o funcionamento da interface serial do 8051.

bit 7 bit 0

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0/SM1 – Serial Mode
Esses bits de controle permitem a obtenção de quatro modos de funcionamento distintos para a interface serial.

SM0	SM1	Modo	Taxa de Transmissão e Recepção
0	0	0	Freqüência de clock dividida por 12
0	1	1	Programável
1	0	2	Freqüência de clock dividida por 32 ou 64
1	1	3	Programável

SM2 – Serial Mode
Possui diferentes funções em cada modo de funcionamento:

- Modo 0: não altera em nada o funcionamento, devendo permanecer no nível lógico 0.
- Modo 1: quando em nível lógico 1 desabilita o pedido de interrupção se for recebido um stop bit inválido.
- Modos 2 e 3: quando em nível lógico 1 habilita a comunicação serial entre vários microcontroladores e desabilita o pedido de interrupção se for recebido um nono bit de dado igual a 0.

REN – Reception Enable
Nível lógico 0 – desabilita a recepção serial.
Nível lógico 1 – habilita a recepção serial.

TB8 – Transmit Bit 8
Nos Modos 2 e 3 indica o estado do nono bit a ser transmitido, programado por software.

RB8 – Receive Bit 8
Não é utilizado no Modo 0 e no Modo 1 indica o estado do stop bit recebido, se o bit SM2 estiver em nível lógico 0. Nos Modos 2 e 3 indica o estado do nono bit recebido.

TI – Transmit interrupt
É o bit de requisição de interrupção da transmissão de dados, sendo setado por hardware após a transmissão do oitavo bit de dados quando no Modo 0 ou ao início do stop bit nos Modos 1,2 e 3. Deve ser zerado por software durante a execução da rotina de atendimento para permitir novas interrupções.

RI – Receive Interrupt
É o bit de requisição de interrupção da recepção de dados, sendo setado por hardware após a recepção do oitavo bit de dados quando no Modo 0 ou a meio tempo da recepção do stop bit nos Modos 1,2 e 3. Deve ser zerado por software durante a execução da rotina de atendimento para permitir novas interrupções.

Modos de Operação da Interface Serial

- **Modo 0 (SM0=0, SM1=0)**

É o único modo de operação onde ocorre comunicação de dados síncrona half-duplex. A taxa de transmissão e recepção é fixa (freqüência de clock do 8051 dividida por 12).

São transmitidos ou recebidos sempre 8 bits de dados, sendo o menos significativo transmitido primeiramente. Os dados (data) são transmitidos ou recebidos pelo terminal RXD e o sinal de sincronismo (clock) pelo terminal TXD.

- **Modo 1 (SM0=0, SM1=1)**

Nesse modo de operação e nos dois modos seguintes ocorre comunicação de dados assíncrona full-duplex com taxa de transmissão e recepção programável. A recepção dos dados é feita pelo terminal RXD e a transmissão pelo terminal TXD.

No Modo 1 cada pacote de dados transmitido ou recebido possui 10 bits, sendo um start bit seguido de 8 bits de dados e um stop bit. Na recepção o stop bit vai para o bit RB8 do registro SCON.

- **Modo 2 (SM0=1, SM1=0)**

No Modo 2 cada pacote de dados transmitido ou recebido possui 11 bits, sendo um start bit seguido de 8 bits de dados, um nono bit e um stop bit.

Na transmissão o nono bit a ser transmitido deverá estar no bit TB8 do registro SCON. Na recepção o nono bit vai para o bit RB8.

A taxa de transmissão e recepção pode ser programada para 1/32 ou 1/64 da freqüência de clock do microcontrolador.

- **Modo 3 (SM0=1, SM1=1)**

O Modo 3 possui funcionamento idêntico ao do Modo 2, exceto pela taxa de transmissão que é programável.

Taxa de Transmissão e Recepção Serial

- Modo 0: a taxa é igual a 1/12 da freqüência de clock (fixa).
- Modo 2: a taxa depende apenas do valor do bit SMOD no registro PCON. Para SMOD=0 a taxa é igual a 1/32 da freqüência de clock e para SMOD=1 a taxa é igual a 1/64 da freqüência de clock.
- Modos 1 e 3: a taxa é fornecida pelo T/C1, de maneira que os dados são transmitidos a cada estouro de contagem. Nesse caso o modo de funcionamento mais comum do T/C1 é o Modo 2 (recarga automática). A interrupção do T/C1 deve ser desabilitada.

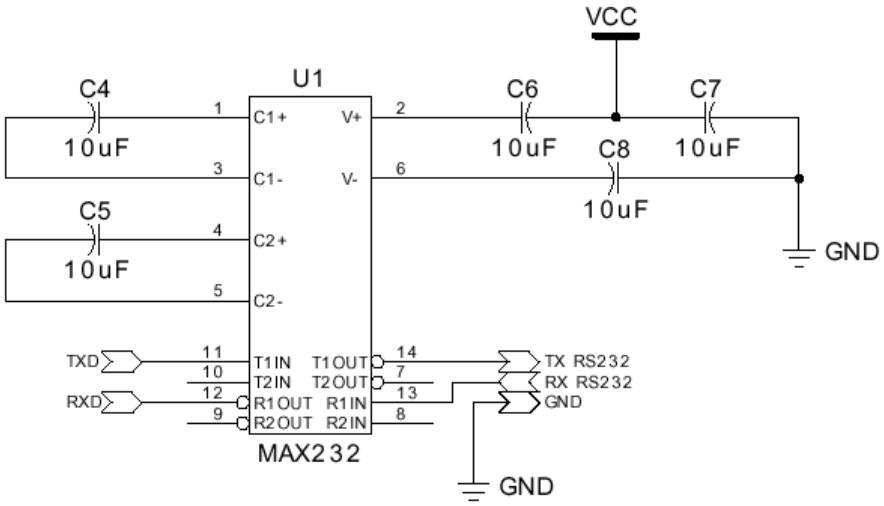
O cálculo da taxa de transmissão e recepção da interface serial com o T/C1 no Modo 2 pode ser feito pela seguinte fórmula:

$$Taxa = \frac{2^{\mathrm{SMOD}}}{32} . \frac{f_{clock}}{12.(256 - \mathrm{TH1})}$$

A seguinte tabela com os valores mais usuais de taxa de transmissão e recepção indica os valores de recarga do T/C1 operando no Modo 2, com SMOD=0 e f_{clock}=11,0592MHz:

Taxa	300 bps	600 bps	1200 bps	2400 bps	4800 bps	9600 bps
TH1	160	208	232	244	250	253

O diagrama esquemático a seguir mostra como é possível implementar uma interface serial assíncrona padrão RS-232, utilizando o circuito integrado MAX232 (conversor de nível TTL/RS232).



Exercícios

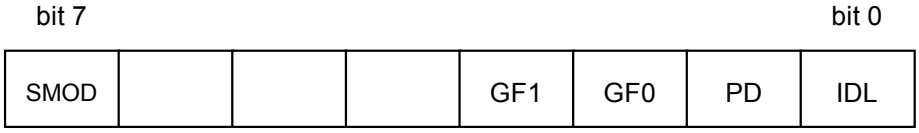
1) Deseja-se programar a interface serial de um microcontrolador 80C31 com clock de 7,3728MHz para operar com comunicação serial assíncrona a 4800 bps. Escreva um programa em Assembly que inicialize a interface serial e o temporizador responsável pela taxa de transmissão e recepção de modo a satisfazer esses requisitos. Uma vez programados os registros especiais necessários, envie o valor B2H pela interface serial.

2) Examine o código-fonte do monitor Paulmon no que diz respeito à forma de implementação das subrotinas CIN, COUT, PHEX, GHEX e PSTRING. Tire suas conclusões a respeito da forma de utilização do registro de função especial SBUF e dos flags TI e RI.

MODOS DE REDUÇÃO DE CONSUMO

Os microcontroladores da família 8051 fabricados com a tecnologia CMOS (80C51) possuem bits de controle que permitem a ativação de modos de operação onde o consumo de energia do dispositivo torna-se reduzido, preservando o conteúdo da memória de dados interna.

- **Registro PCON – Power Control:** Esse registro possui bits de controle dos modos de redução de consumo no 80C51, bits sinalizadores (flags) de uso geral no 80C51 e um bit de controle da taxa de transmissão e recepção da interface serial.



SMOD – Serial Mode
Nos Modos 1,2 e 3 da interface serial a taxa de transmissão e recepção é dobrada quando esse bit está em nível lógico 1.

GF1 – General Flag 1; GF0 – General Flag 0
São bits sinalizadores de uso geral do software.

PD – Power Down; IDL – Idle
Quando em nível lógico alto ativam respectivamente o modo "power down" e o modo "idle". O modo "power down" é prioritário sobre o modo "idle".

Funcionamento no Modo Idle

Esse modo de redução de consumo de energia é conhecido como "modo preguiçoso", pois a CPU é completamente desativada. Entretanto, a memória de dados interna, as interrupções e os demais periféricos do 80C51 permanecem ativos. Nesse caso há uma redução no consumo de aproximadamente 15% em relação ao modo de operação normal.

Para sair do Modo Idle existem duas possibilidades: a requisição de uma interrupção que esteja habilitada ou através de um reset de hardware.

Na ocorrência de uma interrupção habilitada, o bit IDL do registro PCON será zerado por hardware, caracterizando o final da operação no Modo Idle. O fluxo programa será então desviado para a rotina de atendimento da interrupção e ao ser executada a instrução de retorno de interrupção, o programa retornará à instrução seguinte à que causou a entrada no Modo Idle.

Se for utilizado o reset para a saída do Modo Idle, deve-se garantir que o terminal RST permaneça em nível lógico 1 por pelo menos dois ciclos de máquina. Através desse método o bit IDL também será zerado por hardware, porém nesse caso não ocorrerá o desvio do programa para a posição inicial da memória e sim para a instrução posterior à que entrou no Modo Idle.

Funcionamento no Modo Power Down

Nesse caso todas as funções internas do microcontrolador são suspensas, permanecendo inalterado apenas o conteúdo da memória de dados interna. O consumo de corrente do 80C51 nesse modo de operação fica em torno de 10μA. A diferença fundamental em termos de funcionamento é que no Modo Idle o sinal de clock é inibido apenas na CPU e no Modo Power Down é inibido no microcontrolador inteiro.

Existe uma única maneira de sair do Modo Power Down: através o reset de hardware. O sinal de reset no terminal RST deverá ser mantido em nível lógico alto por pelo menos 10ms para permitir o restabelecimento do sinal de clock que estava inativo.

Ao ocorrer o reset todos os registros especiais serão reinicializados e portanto o programa recomeçará do endereço 0000H, mas a memória de dados interna não será afetada.

A grande vantagem desse modo de redução de consumo está na possibilidade de se reduzir a tensão de alimentação do 80C51 até aproximadamente 2V, mantendo os dados armazenados na memória interna. No entanto, a alimentação deverá ser restabelecida a 5V antes do retorno ao modo normal de operação, sob pena de perda do conteúdo da memória de dados interna.

BIBLIOGRAFIA

BARBACENA, I. L., FLEURY, C. A., Kit Didático para Estudo dos Microcontroladores 8051, Revista Saber Eletrônica: dezembro/1997, janeiro/1998, fevereiro/1998, março/1998 e abril/1998.

HALL, D. V., Microprocessors and Interfacing, McGraw-Hill,1986.

INTEL, Embedded Controller Handbook.

MARQUES, L. C., WISINTAINER, M. A., MATIAS JR., R., MAIA, L. F. J., ALVES, J. B. M., Microcontrolador 8051: Laboratório de Experimentação Remota via Internet, Revista Saber Eletrônica: nº 306 – julho/1998.

NICOLOSI, D. E. C., Laboratório de Microcontroladores – Família 8051: Treino de Instruções, Hardware e Software, 1 ed., São Paulo: Érica, 2002.

NICOLOSI, D. E. C., Microcontrolador 8051 Detalhado, 2. ed., São Paulo: Érica: 2001.

SILVA JÚNIOR., V. P., Aplicações Práticas do Microcontrolador 8051, 5 ed. São Paulo: Érica, 1994.

SOUZA, D. J., Desbravando o PIC: Baseado no Microcontrolador PIC16F84, São Paulo: Érica, 2000.

_____, Debugging the 8031 Series, Elektor Electronics Magazine: november 1994.

_____, Microcontroller Survey: News from the World of Bit Crunchers, Elektor Electronics Magazine: february, 1999.

INFORMAÇÕES ÚTEIS NA INTERNET

http://www.atmel.com/ – **Atmel**

http://www.dalsemi.com/ – **Dallas Semiconductor**

http://developer.intel.com/design/ – **Intel**

http://www-us2.semiconductors.philips.com/ – **Philips Semiconductors**

http://www.infineon.com/ – **Infineon Technologies**

http://www.ceibo.com/ – **Ceibo (Embedded C++)**

http://www.fsinc.com/ – **Franklin Software (ProView – C e Assembly)**

http://www.keil.com/ – **Keil Software (μVision – C e Assembly)**

http://www.mcselec.com/ – **MCS Electronics (BasCom51 – BASIC)**

http://www.labcenter.co.uk/ – **Labcenter (Simulador de Hardware)**

http://www.etfgo.br/kitdidatico/ – **Kit Didático - 8051**

http://www.inf.ufsc.br/~jbosco/labvir.htm – **Experimentação Remota - 8051**

http://www.8052.com/ – **Informações e Recursos**

http://www.pjrc.com/tech/8051/ – **Informações e Recursos (PaulMon)**

http://www.vaultbbs.com/sim8052/ – **Simulador**

http://www.microcontroller.com/ - **Informações Gerais**

ANEXOS

CONJUNTO DE INSTRUÇÕES MCS51

CONJUNTO DE INSTRUÇÕES MCS51 (1)

MNEMÔNICO	DESCRIÇÃO	BYTES	CICLOS
Operações Aritméticas:			
ADD A, Rn	Adiciona registro ao acumulador	1	1
ADD A, direto	Adiciona byte direto ao acumulador	2	1
ADD A, @Ri	Adiciona RAM indireta ao acumulador	1	1
ADD A, #dado	Adiciona dado imediato ao acumulador	2	1
ADDC A, Rn	Adiciona registro ao acumulador com carry	1	1
ADDC A, direto	Adiciona byte direto ao acumulador com carry	2	1
ADDC A, @Ri	Adiciona RAM indireta ao acumulador com carry	1	1
ADDC A, #dado	Adiciona dado imediato ao acumulador com carry	2	1
SUBB A, Rn	Subtrai registro do acumulador com borrow	1	1
SUBB A, direto	Subtrai byte direto do acumulador com borrow	2	1
SUBB A, @Ri	Subtrai RAM indireta do acumulador com borrow	1	1
SUBB A, #dado	Subtrai dado imediato do acumulador com borrow	2	1
INC A	Incrementa acumulador	1	1
INC Rn	Incrementa registro	1	1
INC direto	Incrementa byte direto	2	1
INC @Ri	Incrementa RAM indireta	1	1
DEC A	Decrementa acumulador	1	1
DEC Rn	Decrementa registro	1	1
DEC direto	Decrementa byte direto	2	1
DEC @Ri	Decrementa RAM indireta	1	1
INC DPTR	Incrementa ponteiro de dados	1	2
MUL AB	Multiplica A e B	1	4
DIV AB	Divide A por B	1	4
DA A	Ajuste decimal de A	1	1
Operações Lógicas:			
ANL A, Rn	AND entre o acumulador e registro	1	1
ANL A, direto	AND entre o acumulador e byte direto	2	1
ANL A, @Ri	AND entre o acumulador e RAM indireta	1	1
ANL A, #dado	AND entre o acumulador e dado imediato	2	1
ANL direto, A	AND entre byte direto e o acumulador	2	1
ANL direto, #dado	AND entre byte direto e dado imediato	3	2
ORL A, Rn	OR entre o acumulador e registro	1	1
ORL A, direto	OR entre o acumulador e byte direto	2	1
ORL A, @Ri	OR entre o acumulador e RAM indireta	1	1
ORL A, #dado	OR entre o acumulador e dado imediato	2	1
ORL direto, A	OR entre byte direto e o acumulador	2	1
ORL direto, #dado	OR entre byte direto e dado imediato	3	2
XRL A, Rn	XOR entre o acumulador e registro	1	1
XRL A, direto	XOR entre o acumulador e byte direto	2	1
XRL A, @Ri	XOR entre o acumulador e RAM indireta	1	1
XRL A, #dado	XOR entre o acumulador e dado imediato	2	1
XRL direto, A	XOR entre byte direto e o acumulador	2	1
XRL direto, #dado	XOR entre byte direto e dado imediato	3	2
CLR A	Zera o acumulador	1	1

CONJUNTO DE INSTRUÇÕES MCS51 (2)

MNEMÔNICO	DESCRIÇÃO	BYTES	CICLOS
CPL A	Complementa o acumulador	1	1
RL A	Rotaciona acumulador para a esquerda	1	1
RLC A	Rotaciona acumulador para a esquerda através do flag de carry	1	1
RR A	Rotaciona acumulador para a direita	1	1
RRC A	Rotaciona acumulador para a direita através do flag de carry	1	1
SWAP A	Troca nibbles do acumulador	1	1
Transferência de Dados:			
MOV A, Rn	Move registro para o acumulador	1	1
MOV A, direto*	Move byte direto para o acumulador	2	1
MOV A, @Ri	Move RAM indireta para o acumulador	1	1
MOV A, #dado	Move dado imediato para o acumulador	2	1
MOV Rn, A	Move acumulador para registro	1	1
MOV Rn, direto	Move byte direto para registro	2	2
MOV Rn, #dado	Move dado imediato para registro	2	1
MOV direto, A	Move acumulador para byte direto	2	1
MOV direto, Rn	Move registro para byte direto	2	2
MOV direto, direto	Move byte direto para byte direto	3	2
MOV direto, @Ri	Move RAM indireta para byte direto	2	2
MOV direto, #dado	Move dado imediato para byte direto	3	2
MOV @Ri, A	Move acumulador para RAM indireta	1	1
MOV @Ri, direto	Move byte direto para RAM indireta	2	2
MOV @Ri, #dado	Move dado imediato para RAM indireta	2	1
MOV DPTR, #dado16	Carrega ponteiro de dados com constante de 16 bits	3	2
MOVC A, @A+DPTR	Move byte de código relativo ao DPTR para o acumulador	1	2
MOVC A, @A+PC	Move byte de código relativo ao PC para o acumulador	1	2
MOVX A, @Ri	Move RAM externa (end. 8 bits) para o acumulador	1	2
MOVX A, @DPTR	Move RAM externa (end. 16 bits) para o acumulador	1	2
MOVX @Ri, A	Move acumulador para RAM externa (end. 8 bits)	1	2
MOVX @DPTR, A	Move acumulador para RAM externa (end. 16 bits)	1	2
PUSH direto	Coloca byte direto na pilha	2	2
POP direto	Retira byte direto da pilha	2	2
XCH A, Rn	Troca registro com o acumulador	1	1
XCH A, direto	Troca byte direto com o acumulador	2	1
XCH A, @Ri	Troca RAM indireta com o acumulador	1	1
XCHD A, @Ri	Troca dígito menos significativo de RAM indireta com o acumulador	1	1
Manipulação de Variáveis Booleanas:			
CLR C	Reseta o flag de carry	1	1
CLR bit	Reseta bit direto	2	1
SETB C	Seta o flag de carry	1	1
SETB bit	Seta bit direto	2	1
CPL C	Complementa o flag de carry	1	1
CPL bit	Complementa bit direto	2	1
ANL C, bit	AND entre o flag de carry e bit direto	2	2

CONJUNTO DE INSTRUÇÕES MCS51 (3)

MNEMÔNICO	DESCRIÇÃO	BYTES	CICLOS
ANL C, /bit	AND entre o flag de carry e complemento de bit direto	2	2
ORL C, bit	OR entre o flag de carry e bit direto	2	2
ORL C, /bit	OR entre o flag de carry e complemento de bit direto	2	2
MOV C, bit	Move bit direto para o flag de carry	2	1
MOV bit, C	Move o flag de carry para bit direto	2	2
Controle de Programa:			
ACALL end11	Chamada absoluta de subrotina	2	2
LCALL end16	Chamada longa de subrotina	3	2
RET	Retorno de subrotina	1	2
RETI	Retorno de interrupção	1	2
AJMP end11	Desvio absoluto	2	2
LJMP end16	Desvio longo	3	2
SJMP rel	Desvio curto	2	2
JMP @A+DPTR	Desvio indireto relativo ao DPTR	1	2
JZ rel	Desvio se A for igual a zero	2	2
JNZ rel	Desvio se A não for igual a zero	2	2
JC rel	Desvio se o flag de carry for igual a 1	2	2
JNC rel	Desvio se o flag de carry for igual a 0	2	2
JB bit, rel	Desvio se o bit direto for igual a 1	3	2
JNB bit, rel	Desvio se o bit direto for igual a 0	3	2
JBC bit, rel	Desvio se o bit direto for igual a 1, resetando-o	3	2
CJNE A, direto, rel	Comparação entre A e byte direto, desvio se não forem iguais	3	2
CJNE A, #dado, rel	Comparação entre A e dado imediato, desvio se não forem iguais	3	2
CJNE Rn, #dado, rel	Comparação entre registro e dado imediato, desvio se não forem iguais	3	2
CJNE @Ri, #dado, rel	Comparação entre RAM indireta e dado imediato, desvio se não forem iguais	3	2
DJNZ Rn, rel	Decrementa registro, desvio se não for igual a zero	2	2
DJNZ direto, rel	Decrementa byte direto, desvio se não for igual a zero	3	2
NOP	Nenhuma operação	1	1

*MOV A, ACC não é uma instrução válida

Modos de Endereçamento de Dados

Rn - registros de trabalho R0-R7
direto – RAM interna, ports de E/S ou registros especiais
@Ri - RAM interna ou externa endereçada indiretamente por R0 ou R1
#dado - constante de 8 bits incluída na instrução (1 byte)
#dado16- constante de 16 bits incluída na instrução (2 bytes)
bit - flags, bits de ports de E/S ou de registros especiais
A - acumulador

Modos de Endereçamento de Programa

end16 - endereço de destino para LCALL e LJMP (espaço de 64KB)
end11 - endereço de destino para ACALL e AJMP (página de 2KB)
rel - deslocamento de 8 bits (-128 a +127) relativo à próxima instrução para SJMP e todos os saltos condicionais

CÓDIGOS DAS INSTRUÇÕES MCS51 EM ORDEM HEXADECIMAL (1)

CÓDIGO HEXA	NÚMERO DE BYTES	MNEMÔNICO	OPERANDOS
00	1	NOP	
01	2	AJMP	end. código
02	3	LJMP	end. código
03	1	RR	A
04	1	INC	A
05	2	INC	end. dado
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
OC	1	INC	R4
0D	1	INC	R5
OE	1	INC	R6
0F	1	INC	R7
10	3	JBC	end. bit, end. código
11	2	ACALL	end. código
12	3	LCALL	end. código
13	1	RRC	A
14	1	DEC	A
15	2	DEC	end. dado
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7

CÓDIGO HEXA	NÚMERO DE BYTES	MNEMÔNICO	OPERANDOS
20	3	JB	end. bit, end. código
21	2	AJMP	end. código
22	1	RET	
23	1	RL	A
24	2	ADD	A, #dado
25	2	ADD	A, end. dado
26	1	ADD	A, @R0
27	1	ADD	A, @R1
28	1	ADD	A, R0
29	1	ADD	A, R1
2A	1	ADD	A, R2
2B	1	ADD	A, R3
2C	1	ADD	A, R4
2D	1	ADD	A, R5
2E	1	ADD	A, R6
2F	1	ADD	A, R7
30	3	JNB	end. bit, end. código
31	2	ACALL	end. código
32	1	RETI	
33	1	RLC	A
34	2	ADDC	A, #dado
35	2	ADDC	A, end. dado
36	1	ADDC	A, @R0
37	1	ADDC	A, @R1
38	1	ADDC	A, R0
39	1	ADDC	A, R1
3A	1	ADDC	A, R2
3B	1	ADDC	A, R3
3C	1	ADDC	A, R4
3D	1	ADDC	A, R5
3E	1	ADDC	A, R6
3F	1	ADDC	A, R7

CÓDIGOS DAS INSTRUÇÕES MCS51 EM ORDEM HEXADECIMAL (2)

CÓDIGO HEXA	NÚMERO DE BYTES	MNEMÔNICO	OPERANDOS
40	2	JC	end. código
41	2	AJMP	end. código
42	2	ORL	end. dado, A
43	3	ORL	end. dado, #dado
44	2	ORL	A, #dado
45	2	ORL	A, end. dado
46	1	ORL	A, @R0
47	1	ORL	A, @R1
48	1	ORL	A, R0
49	1	ORL	A, R1
4A	1	ORL	A, R2
4B	1	ORL	A, R3
4C	1	ORL	A, R4
4D	1	ORL	A, R5
4E	1	ORL	A, R6
4F	1	ORL	A, R7
50	2	JNC	end. código
51	2	ACALL	end. código
52	2	ANL	end. dado, A
53	3	ANL	end. dado, #dado
54	2	ANL	A, #dado
55	2	ANL	A, end. dado
56	1	ANL	A, @R0
57	1	ANL	A, @R1
58	1	ANL	A, R0
59	1	ANL	A, R1
5A	1	ANL	A, R2
5B	1	ANL	A, R3
5C	1	ANL	A, R4
5D	1	ANL	A, R5
5E	1	ANL	A, R6
5F	1	ANL	A, R7

CÓDIGO HEXA	NÚMERO DE BYTES	MNEMÔNICO	OPERANDOS
60	2	JZ	end. código
61	2	AJMP	end. código
62	2	XRL	end. dado, A
63	3	XRL	end. dado, #dado
64	3	XRL	A, #dado
65	2	XRL	A, end. dado
66	2	XRL	A, @R0
67	1	XRL	A, @R1
68	1	XRL	A, R0
69	1	XRL	A, R1
6A	1	XRL	A, R2
6B	1	XRL	A, R3
6C	1	XRL	A, R4
6D	1	XRL	A, R5
6E	1	XRL	A, R6
6F	1	XRL	A, R7
70	2	JNZ	end. código
71	2	ACALL	end. código
72	2	ORL	C, end. bit
73	1	JMP	@A+DPTR
74	2	MOV	A, #dado
75	3	MOV	end. dado, #dado
76	2	MOV	@R0, #dado
77	2	MOV	@R1, #dado
78	2	MOV	R0, #dado
79	2	MOV	R1, #dado
7A	2	MOV	R2, #dado
7B	2	MOV	R3, #dado
7C	2	MOV	R4, #dado
7D	2	MOV	R5, #dado
7E	2	MOV	R6, #dado
7F	2	MOV	R7, #dado

CÓDIGOS DAS INSTRUÇÕES MCS51 EM ORDEM HEXADECIMAL (3)

CÓDIGO HEXA	NÚMERO DE BYTES	MNEMÔNICO	OPERANDOS
80	2	SJMP	end. código
81	2	AJMP	end. código
82	2	ANL	C, end. bit
83	1	MOVC	@A+PC
84	1	DIV	AB
85	3	MOV	end. dado, end. dado
86	2	MOV	end. dado, @R0
87	2	MOV	end. dado, @R1
88	2	MOV	end. dado, R0
89	2	MOV	end. dado, R1
8A	2	MOV	end. dado, R2
8B	2	MOV	end. dado, R3
8C	2	MOV	end. dado, R4
8D	2	MOV	end. dado, R5
8E	2	MOV	end. dado, R6
8F	2	MOV	end. dado, R7
90	3	MOV	DPTR, #dado16
91	2	ACALL	end. código
92	2	MOV	end. bit, C
93	1	MOVC	A, @A+DPTR
94	2	SUBB	A, #dado
95	2	SUBB	A, end. dado
96	1	SUBB	A, @R0
97	1	SUBB	A, @R1
98	1	SUBB	A, R0
99	1	SUBB	A, R1
9A	1	SUBB	A, R2
9B	1	SUBB	A, R3
9C	1	SUBB	A, R4
9D	1	SUBB	A, R5
9E	1	SUBB	A, R6
9F	1	SUBB	A, R7

CÓDIGO HEXA	NÚMERO DE BYTES	MNEMÔNICO	OPERANDOS
A0	2	ORL	C, /end. bit
A1	2	AJMP	end. código
A2	2	MOV	C, end. bit
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reservado	
A6	2	MOV	@R0, end. dado
A7	2	MOV	@R1, end. dado
A8	2	MOV	R0, end. dado
A9	2	MOV	R1, end. dado
AA	2	MOV	R2, end. dado
AB	2	MOV	R3, end. dado
AC	2	MOV	R4, end. dado
AD	2	MOV	R5, end. dado
AE	2	MOV	R6, end. dado
AF	2	MOV	R7, end. dado
B0	2	ANL	C, /end. bit
B1	2	ACALL	end. código
B2	2	CPL	end. bit
B3	1	CPL	C
B4	3	CJNE	A, #dado, end. código
B5	3	CJNE	A, end. dado, end. código
B6	3	CJNE	@R0, #dado, end. código
B7	3	CJNE	@R1, #dado, end. código
B8	3	CJNE	R0, #dado, end. código
B9	3	CJNE	R1, #dado, end. código
BA	3	CJNE	R2, #dado, end. código
BB	3	CJNE	R3, #dado, end. código
BC	3	CJNE	R4, #dado, end. código
BD	3	CJNE	R5, #dado, end. código
BE	3	CJNE	R6, #dado, end. código
BF	3	CJNE	R7, #dado, end. código

CÓDIGOS DAS INSTRUÇÕES MCS51 EM ORDEM HEXADECIMAL (4)

CÓDIGO HEXA	NÚMERO DE BYTES	MNEMÔNICO	OPERANDOS
C0	2	PUSH	end. dado
C1	2	AJMP	end. código
C2	2	CLR	end. bit
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A, end. dado
C6	1	XCH	A, @R0
C7	1	XCH	A, @R1
C8	1	XCH	A, R0
C9	1	XCH	A, R1
CA	1	XCH	A, R2
CB	1	XCH	A, R3
CC	1	XCH	A, R4
CD	1	XCH	A, R5
CE	1	XCH	A, R6
CF	1	XCH	A, R7
D0	2	POP	end. dado
D1	2	ACALL	end. código
D2	2	SETB	end. bit
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	end. dado, end. código
D6	1	XCHD	A, @R0
D7	1	XCHD	A, @R1
D8	2	DJNZ	R0, end. código
D9	2	DJNZ	R1, end. código
DA	2	DJNZ	R2, end. código
DB	2	DJNZ	R3, end. código
DC	2	DJNZ	R4, end. código
DD	2	DJNZ	R5, end. código
DE	2	DJNZ	R6, end. código
DF	2	DJNZ	R7, end. código

*MOV A, ACC não é uma instrução válida

CÓDIGO HEXA	NÚMERO DE BYTES	MNEMÔNICO	OPERANDOS
E0	1	MOVX	A, @DPTR
E1	2	AJMP	end. código
E2	1	MOVX	A, @R0
E3	1	MOVX	A, @R1
E4	1	CLR	A
E5	2	MOV	A, end. dado*
E6	1	MOV	A, @R0
E7	1	MOV	A, @R1
E8	1	MOV	A, R0
E9	1	MOV	A, R1
EA	1	MOV	A, R2
EB	1	MOV	A, R3
EC	1	MOV	A, R4
ED	1	MOV	A, R5
EE	1	MOV	A, R6
EF	1	MOV	A, R7
F0	1	MOVX	@DPTR, A
F1	2	ACALL	end. código
F2	1	MOVX	@R0, A
F3	1	MOVX	@R1, A
F4	1	CPL	A
F5	2	MOV	end. dado, A
F6	1	MOV	@R0, A
F7	1	MOV	@R1, A
F8	1	MOV	R0, A
F9	1	MOV	R1, A
FA	1	MOV	R2, A
FB	1	MOV	R3, A
FC	1	MOV	R4, A
FD	1	MOV	R5, A
FE	1	MOV	R6, A
FF	1	MOV	R7, A

MANUAL DA PLACA P51

1. INTRODUÇÃO

Este manual descreve como a placa P51 pode ser utilizada na implementação de circuitos baseados no microcontrolador 8031 ou similares (componentes da família 8051 compatíveis pino a pino com o 8031). Contém um histórico do desenvolvimento da placa P51, o diagrama em blocos, o diagrama esquemático e outras informações sobre a configuração e funcionamento do circuito. Este não é um manual sobre o 8031 ou sobre projetos com o mesmo. Portanto, a leitura deste não dispensa a leitura dos manuais específicos dos componentes utilizados.

A placa P51 (figura 1) foi concebida para permitir o desenvolvimento de circuitos baseados no 8031 sem que o projetista necessitasse implementar o protótipo a partir do zero. A placa P51 contém fiação impressa para as conexões do microcontrolador às memórias RAM e EPROM e para um conversor TTL/RS-232. Aproximadamente a metade da área da placa está reservada para prototipação, ou seja, para montar a parte do circuito que é específica para a aplicação em questão.

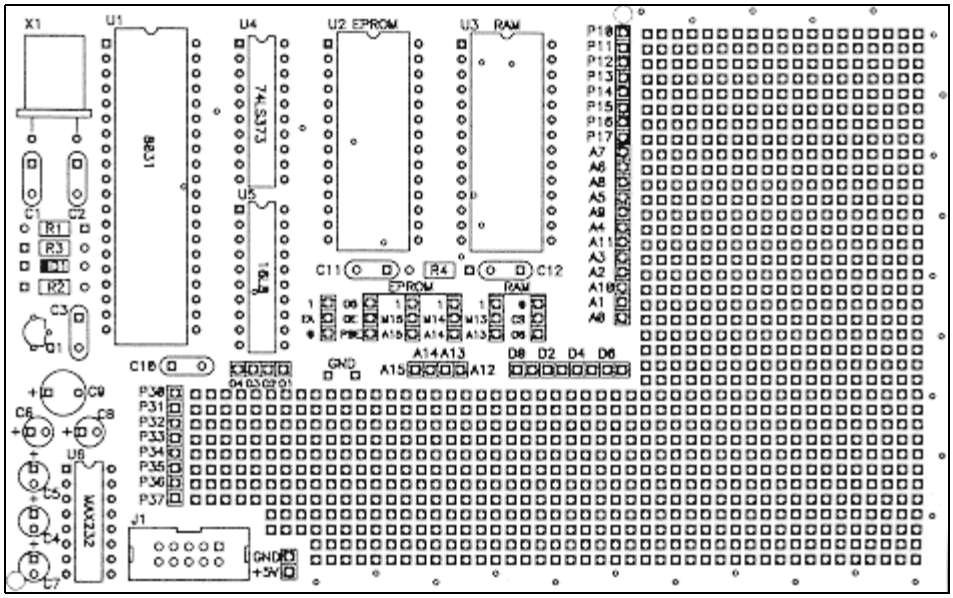


Figura 1: A placa P51 vista pelo lado dos componentes.

A placa P51 foi desenvolvida em janeiro de 1995 no CPGEI (Curso de Pós-Graduação em Engenharia Elétrica e Informática Industrial do CEFET-PR) durante o projeto TD94, realizado no âmbito do convênio CEFET-PR e Bematech. Este projeto foi desenvolvido por Douglas Renaux (coordenador), André Braga (desenvolvimento de software) e Eduardo Bregant (desenvolvimento de hardware, incluindo o desenvolvimento da placa P51). A placa P51 foi utilizada durante a fase de prototipação do projeto TD94, bem como em teses de mestrado do CPGEI, cursos extra-curriculares do CEFET-PR e na disciplina de Sistemas Digitais 2 do Departamento de Eletrônica. O intenso uso desta placa em atividades de ensino e pesquisa no CEFET-PR é uma indicação da importância dos professores do CEFET-PR estarem envolvidos em pesquisas (tanto acadêmicas como contratadas) e demonstra como os resultados destas pesquisas trazem benefícios para a instituição.

A versão 1.1 desta placa inclui as modificações necessárias para o uso do Paulmon, que é um programa monitor para o 8031 desenvolvido por Paul Stoffregen e disponibilizado através da Internet. O Paulmon permite que se carregue um programa em RAM (através da interface serial) e também permite que este

programa seja executado passo a passo. A única modificação necessária na placa P51 foi a inclusão do sinal PSEN\ na PAL. Desta forma a PAL pode gerar os sinais de controle da RAM necessários para a execução de código a partir da RAM. Na versão 1.0 da placa a RAM só podia ser usada para armazenar dados.

2. O CIRCUITO ELÉTRICO

Um diagrama em blocos é apresentado na figura 2 e um diagrama esquemático é apresentado na figura 3. A placa P51 consiste em uma área com fiação impressa (especificada nos diagramas em blocos e esquemático) e uma área de prototipação. A área com fiação impressa contém os componentes básicos do sistema: microcontrolador, demultiplexador do barramento, decodificador de endereços, RAM, EPROM e conversor TTL/RS-232. Estes circuitos normalmente são comuns a todas as aplicações utilizando o 8031. Na área de prototipação é implementada a parte do circuito que é específica para a aplicação em questão.

3. CONFIGURAÇÃO

Uma das características importantes da placa P51 é a possibilidade de configurá-la para uma variedade de aplicações e de componentes. Diversos tipos de EPROM e RAM podem ser utilizados e o circuito decodificador de endereços pode ser projetado de acordo com as necessidades da aplicação.

3.1 EPROM

As seguintes EPROMs podem ser utilizadas na placa P51. A tabela abaixo também mostra como os pinos 1 e 27 são utilizados em cada caso.

EPROM	**2764**	**27128**	**27256**	**27512**
Tamanho	8Kb	16Kb	32Kb	64Kb
Pino 27	PGM\	PGM\	A14	A14
Pino 1	Vpp	Vpp	Vpp	A15

Observação: durante a leitura da EPROM deve-se manter PGM\ e Vpp em +5V.

Para atender aos diferentes tipos de EPROM, os pinos 1 e 27 são conectados a jumpers que devem ser configurados de acordo com a EPROM em uso.

3.2 RAM

As seguintes RAMs podem ser utilizadas na placa P51. A tabela abaixo também mostra como os pinos 1 e 26 são utilizados em cada caso.

RAM	**6264**	**62256**
Tamanho	8Kb	32Kb
Pino 26	CS2	A13
Pino1	Sem conexão	A14

Observação: CS2 é ativo alto (manter em +5V para acessar a RAM).

Para atender aos diferentes tipos de RAM, o pino 26 está conectado a um jumper que deve ser configurado de acordo com a RAM em uso.

3.3 JUMPERS

OEROM é o sinal de seleção da EPROM e é ativo baixo. Pode ser conectado ao PSEN\ do 8031 (ligar J4 a J5) ou pode ser gerado pela PAL (16L8 pino 16 – I/O5, neste caso ligar J4 a J3).

CSRAM é o sinal de habilitação da RAM que pode ser aterrado (ligar J7 a J6) ou gerado pela PAL (ligar J7 a J8). Observação: o tempo de acesso a partir de CSRAM\ é de 85ns na 62256-8 (a mais rápida) e de 150ns na 62256-15.

OERAM é o sinal de habilitação da saída da RAM. Este sinal deve ser gerado pela PAL a partir da decodificação de endereços e do sinal RD\ do 8031. Em algumas aplicações OERAM\ pode ser o sinal RD\ do 8031.

WERAM é o sinal de habilitação de escrita na RAM. Este sinal deve ser gerado pela PAL a partir da decodificação de endereços e do sinal WR\ do 8031. Em algumas aplicações, WERAM\ pode ser o sinal WR\ do 8031.

EA do 8031. Aterrar (ligar J10 a J11) para executar o programa da EPROM externa (8031) e ligar em Vcc (ligar J10 a J9) para executar o programa da ROM interna (8951).

Para ligar o pino 27 da EPROM a um resistor de pull-up, interligar J16 e J15. Para ligar o pino 27 ao sinal A14, interligar J16 a J17. Para ligar o pino 1 da EPROM a um resistor de pull-up, interligar J18 a J19. Pra ligar o pino 1 ao sinal A15, interligar J19 e J20.

Para ligar o pino 26 da RAM a um resistor de pull-up, interligar J12 e J13. Para ligar o pino 26 ao sinal A13, interligar J13 a J14.

Grupo	Sinal	Jumper	Posição	Efeito
RAM	CS	J7	0 O6	Aterra o pino de CS da RAM Liga o CS da RAM ao pino I/O6 da PAL
RAM	M13	J13	1 A13	6264: coloca o pino 26 (CS2) em nível alto 62256: liga o pino 26 (A13) ao sinal A13 da CPU
EPROM	M14	J16	1 A14	27128 e 2764: coloca o pino PGM\ em nível alto 27256 e 275121: liga pino 27 (A14) ao sinal A14 da CPU
EPROM	M15	J19	1 A15	27256, 27128 e 2764: coloca o pino Vpp em nível alto 27512: liga o pino 1 (A15) ao sinal A15 da CPU
EPROM	OE	J4	PSEN OE	Liga o pino OE\ da EPROM ao sinal PSEN\ da CPU Liga o pino OE\ da EPROM ao pino I/O5 da PAL
uC	EA	J10	1 0	Coloca o pino EA\ da CPU em nível alto Coloca o pino EA\ da CPU em nível baixo

4. CONECTOR PARA INTERFACE SERIAL

Sinal	Origem	J1 (placa P51)	DB9	DB25
Data Carrier Detect (DCD)	DCE			8
Receive Data (Rx)	DCE	3	2	3
Transmit Data (Tx)	DTE	5	3	2
Data Terminal Ready (DTR)	DTE			20
Ground (GND)		9	5	7
Data Set Ready (DSR)	DCE			6
Request to Send (RTS)	DTE	4	7	4
Clear to Send (CTS)	DCE	6	8	5
Ring Indicator (RI)	DCE			22

DTE = conector macho, DCE = conector fêmea
RS-232: -12V = Mark = 1 (lógico), +12V = Space = 0 (lógico)

5. CONCLUSÃO

A placa P51 aceita uma grande variedade de componentes, tanto de tipos como de velocidades distintas. Blocos como o decodificador de endereços podem ser eliminados (substituído por conexões diretas) ou implementados com circuitos programáveis. Uma variedade de circuitos pode ser implementada na área de prototipação. Portanto, para cada projeto específico é necessário um projeto detalhado, incluindo diagrama em blocos, esquemático, e diagrama de temporização (timing).

A placa P51 tem servido para uma variedade de implementações. Um circuito mínimo consiste apenas no 8031, circuito de reset (pode ser apenas um capacitor), circuito do cristal, um 74LS373 e mais alguns componentes discretos (resistores, capacitores e jumpers). Este circuito pode ser utilizado em conjunto com um emulador de ROM conectado ao soquete da EPROM. Por outro lado, a placa P51 já foi utilizada em circuitos com display de cristal líquido, teclado, relógio de tempo real, interface serial e interface paralela. Esta variedade de aplicações, aliada ao baixo custo da placa, facilidade de configuração do sistema e dimensões reduzidas e padronizadas (Eurocard) são alguns dos pontos positivos da placa. Sugestões para uma versão melhorada são bem-vindas.

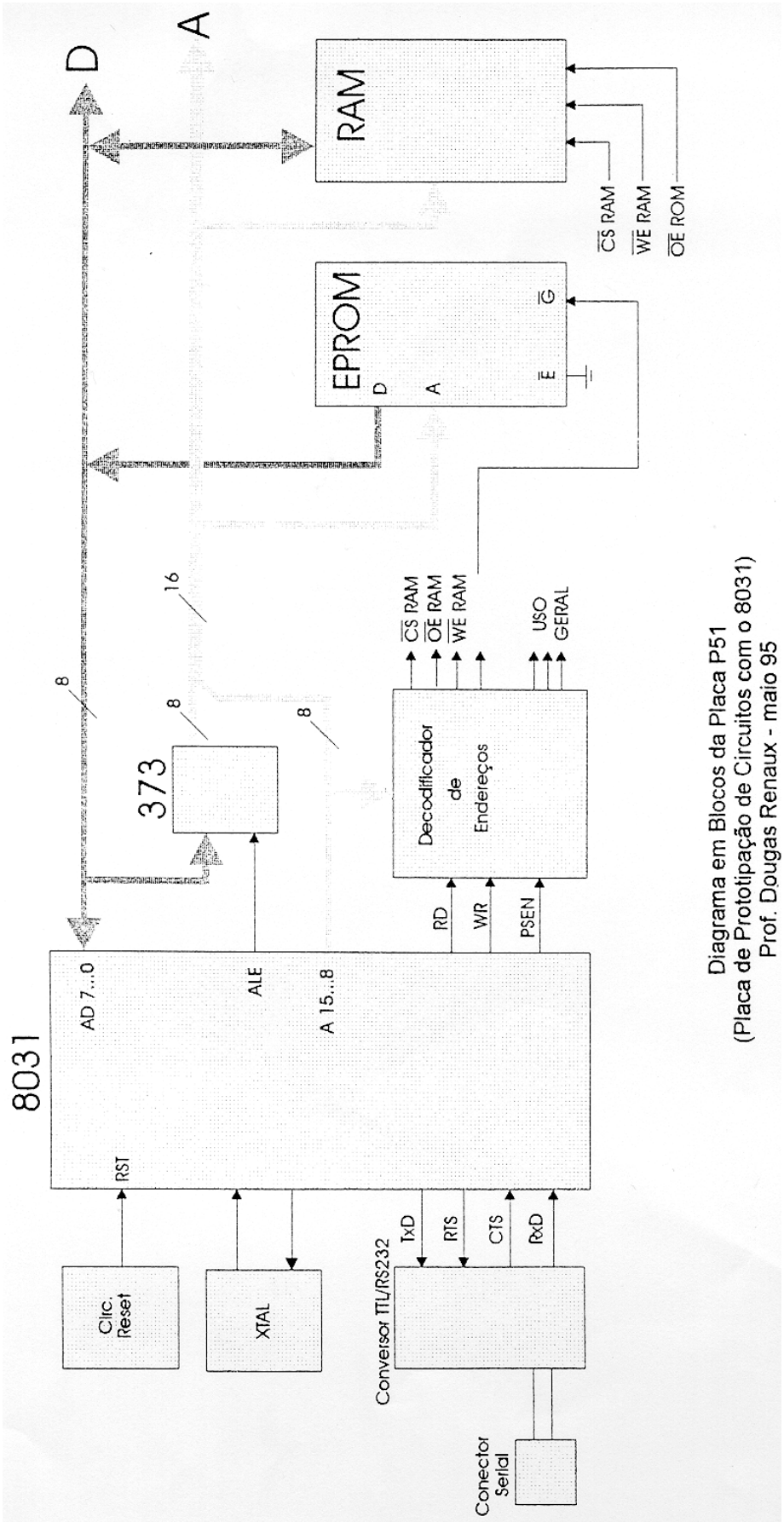


Figura 2: Diagrama em blocos.

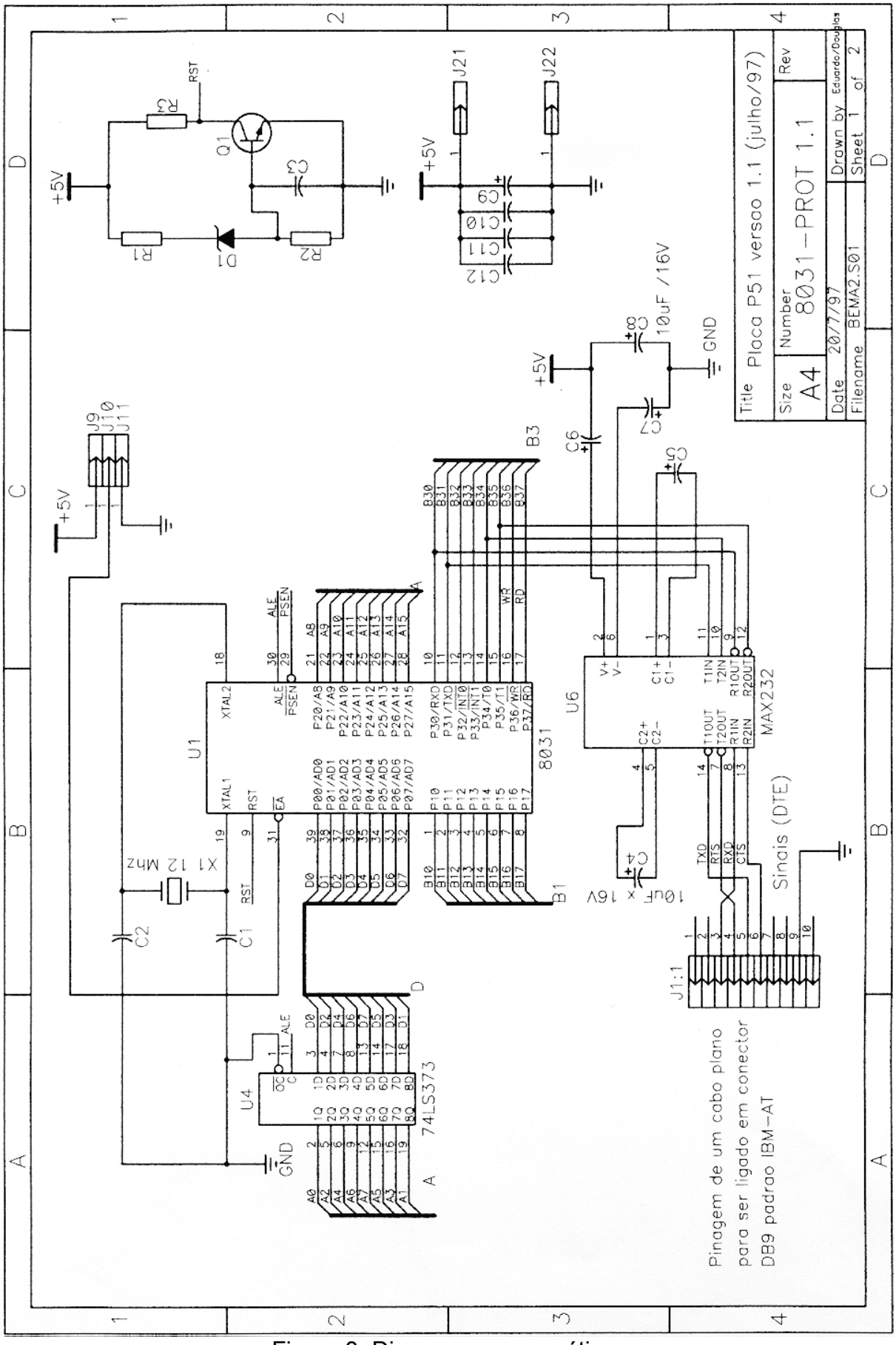
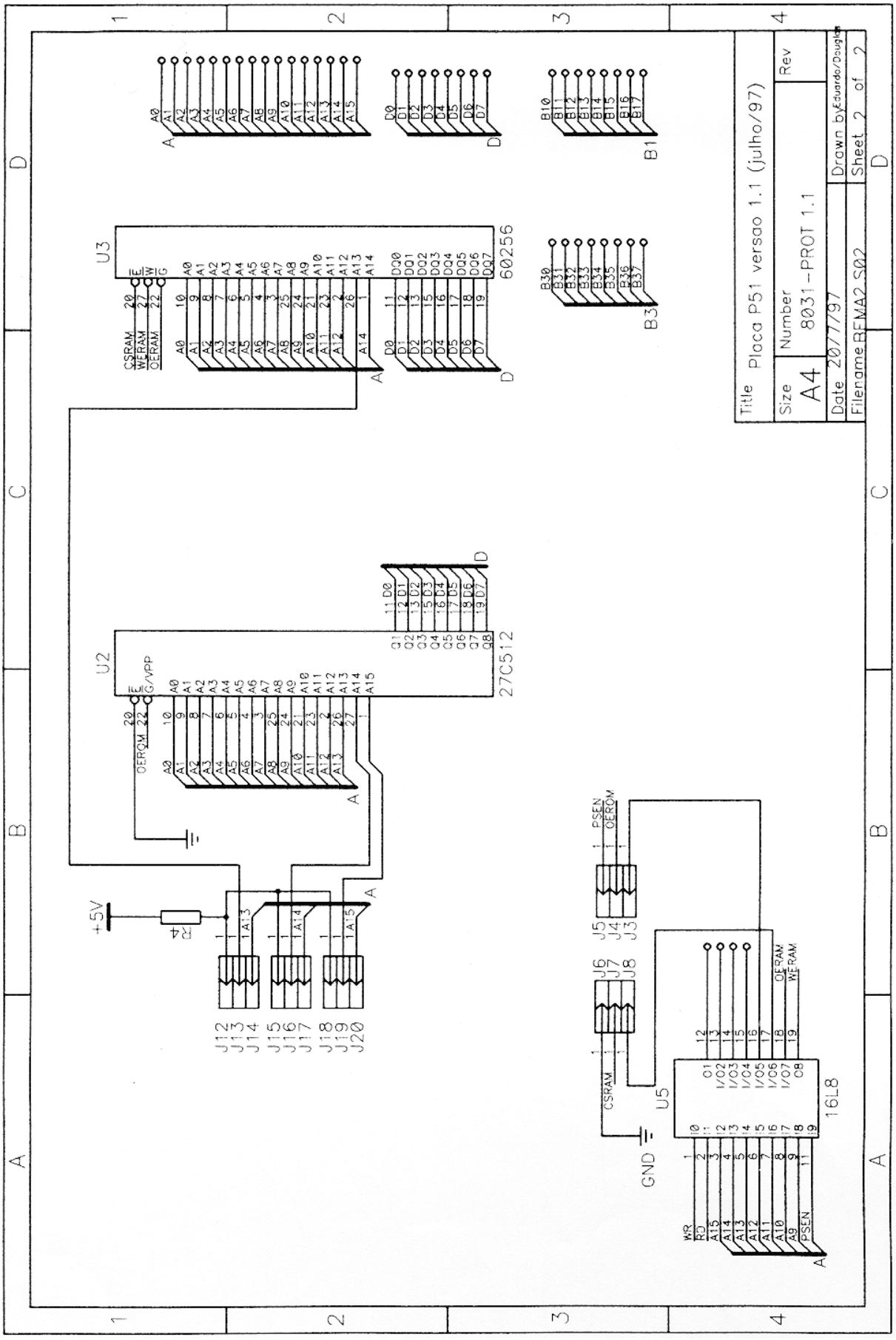


Figura 3: Diagrama esquemático.


Title Placa P51 versao 1.1 (julho/97)
Size A4
Number 8031-PROT 1.1
Rev
Date 20/7/97
Drawn by Eduardo/Douglas
Filename BFMA2.S02
Sheet 2 of 2
U2
27C512
U3
62256
U5
16L8
+5V
R4
GND
CSRAM
WRAM
ORAM
OEROM
PSEN
WR
RD

PROGRAMAÇÃO DA PAL DA PLACA P51

```
;PALASM Design Description

;--------------------------------- Declaration Segment ------------
TITLE    P51LCD.PDS
PATTERN  A
REVISION 1.0
AUTHOR   Hugo Vieira Neto
COMPANY  CEFET-PR
DATE     08/26/99

CHIP  DECODER  PAL16L8

;---------------------------------- PIN Declarations ---------------
PIN  1          WR                    COMBINATORIAL             ; INPUT
PIN  2          RD                    COMBINATORIAL             ; INPUT
PIN  3          A15                   COMBINATORIAL             ; INPUT
PIN  4          A14                   COMBINATORIAL             ; INPUT
PIN  5          A13                   COMBINATORIAL             ; INPUT
PIN  6          A12                   COMBINATORIAL             ; INPUT
PIN  7          A11                   COMBINATORIAL             ; INPUT
PIN  8          A10                   COMBINATORIAL             ; INPUT
PIN  9          A9                    COMBINATORIAL             ; INPUT
PIN  11         PSEN                  COMBINATORIAL             ; INPUT
PIN  10         GND                                             ; INPUT
PIN  12         O1                    COMBINATORIAL             ; OUTPUT
PIN  13         O2                    COMBINATORIAL             ; OUTPUT
PIN  14         O3                    COMBINATORIAL             ; OUTPUT
PIN  15         EDIS                  COMBINATORIAL             ; OUTPUT
PIN  16         OEROM                 COMBINATORIAL             ; OUTPUT
PIN  17         CSRAM                 COMBINATORIAL             ; OUTPUT
PIN  18         OERAM                 COMBINATORIAL             ; OUTPUT
PIN  19         WERAM                 COMBINATORIAL             ; OUTPUT
PIN  20         VCC                                             ; INPUT

;---------------------------------- Boolean Equation Segment ------
EQUATIONS
/CSRAM  = (/A15 * A13) + (/A15 * A14) + (A15 * /A14 * /A13)
/OERAM  = /(RD  * PSEN )
WERAM  = WR
/OEROM  = /PSEN  * /A15 * /A14 * /A13
/O1  = A15 * /A14 * A13
/O2  = A15 * A14 * /A13
/O3  = A15 * A14 * A13 * /A12
EDIS  = A15 * A14 * A13 * A12 * (/RD + /WR)
```

LISTA DE COMPONENTES DA PLACA P51

Circuitos Integrados:

U1 – microcontrolador 80C31 ou 80C32 (89C51 ou 89C52)*
U2 – EPROM 27C64, 27C128, 27C256 ou 27C512*
U3 – RAM 6264 ou 62256
U4 – 74HC373
U5 – PAL16L8
U6 – MAX232

Capacitores:

C1, C2 – 22pF cerâmico
C4, C5. C6, C7, C8 – 10uF x 16V eletrolítico
C9 – 100uF x 16V eletrolítico
C10, C11, C12 – 100nF poliéster

Resistores:

R4 – 1k x1/8W

Diversos:

X1 – cristal oscilador de 11,0592MHz
6 jumpers de configuração
1 soquete torneado de 40 pinos
2 soquetes torneados de 28 pinos
2 soquetes torneados de 20 pinos
1 soquete torneado de 16 pinos
Barra de pinos simples180^{o}
Barra de pinos dupla 180^{o}
Barra de pinos torneados

Extras:

1 capacitor de 10uF x 16V eletrolítico (circuito de reset)
1 resistor de 8k2 x 1/8W (circuito de reset)
1 tact-switch (circuito de reset)
1 regulador de tensão 7805 (fonte de alimentação)
1 jack para eliminador de pilhas (fonte de alimentação)
1 eliminador de pilhas de 9V x 500mA (fonte de alimentação)
1 conector latch fêmea de 10 pinos (cabo de comunicação)
1 conector DB-25 fêmea com capa (cabo de comunicação)
2 metros de flat-cable de 10 vias (cabo de comunicação)

*Observação: Quando forem utilizados os microcontroladores 89C51 ou 89C52 não há necessidade de se utilizar EPROM.

MANUAL DO PAULMON

PAULMON'S DOCUMENTATION

Introduction:

The PAULMON debugger is my attempt to make a user-friendly 8051 debugger, with enough on-line information that it should be unnecessary to read this doc file. PAULMOM is targeted for use by the microprocessor design course at Oregon State, but may be used by anyone (who can figure it out) for projects ranging from research to commercial products. PAULMON is free and may not be distributed for profit whatsoever.

Since I don't expect Prof's or TA's at OSU to make students aware of this documentation nor to provide it nor do I expect students to read much of it, I wrote PAULMON to be very simple and to provide lots of on-line clues about what it can do and how to go about it. I hope that you find PAULMON to be useful and easy to use. Good Luck.

Paul Stoffregen
(paul@ece.orst.edu)

DISCLAIMER: This is free software. As far as warranty is concerned, you get exactly what you pay for! I have tried to make this code as good as possible during the four weeks I worked on it, but nobody is perfect and portions (the single step in particular) were never well tested. USE AT YOUR OWN RISK. The assembly source is provided in case there's something you don't like.

ADDITIONAL DISCLAIMER: This doc file has lots of *tyopes* and other *errorss*, and I really don't care. PAULMON was written to be easy enough that this file ought to be unnecessary, but people ask for it nonetheless, usually before they even try to use the thing.

What you will need to use it:

PAULMON is 8051 assembly code which is intended to be burned into a 2764 EPROM, though a pair of 2732's could be used or a bigger ROM can be used with the rest being empty or filled with other code. The EPROM with PAULMON should be addressed so that it is read from 0000 to 1FFF with the 8051's EA pin wired to make it read all code from external memory.

PAULMON uses the built-in UART in the 8051 to communicate with the user. Typically, a PC computer is used with a terminal program, an 8051 assembler, and a text editor to form a simple, low cost 8051 development system with PAULMON. A serial line receiver and driver should be used (the MAX232 is a good choice, IMHO) to interface the 8051 to the PC's serial port. Only TXD, RXD and ground are used (no handshaking) and PAULMON adapts to use whatever baud rate the computer is using (if it can with the crystal you select, see below).

PAULMON is intended to be used with RAM as well, and the default location for the beginning of the RAM is 2000 (hex), right after the EPROM, though the RAM can be used anywhere in the range of 2000 - FFFF. The read enable signal to the RAM should be the logical OR of the RD and PSEN signals, so that read attempts to external code memory or program memory spaces will read from the RAM. (Use an AND gate to do the logical OR of these signals, since they are active low!) Obviously the write enable of the RAM should be connected to the WR pin of the 8051.

Having a RAM connected in this way will allow the download command in PAULMON to write your program into the RAM (writing into the external data memory space). Then you can run your program, since read attempts from the external program memory space will read from the RAM chip.

How to get it set up:

Design and build your 8051 board. All that is really required is the 8051, an EPROM, a latch (74xx373), some sort of address decoding to enable the EPROM for memory access between 0000-1FFF, and a line receiver to convert the high voltage RS232 to a TTL (or CMOS) compatible signal (or else you'll toast the 8051 before it even has a chance).

To really use PAULMON, a RAM is required as well as the AND gate to allow both program and data read cycles to read the RAM memory, and a reset button to easily get back to PAULMON when your program crashes.

With just the minimal setup, set the computer's baud rate to something slow (like 1200 bps) and power up the board. Press Enter (Return) and hopefully you'll see a screen full of text from PAULMON. PAULMON does not send line feed characters, so the terminal emulator software must be configured to translate CR to CR/LF. (PAULMON ignores LF characters it receives.) If the entire message ends up on one line, then the terminal is not translating CR to CR/LF. After it works, you can try increasing the baud rate and COLD-BOOTING (you must turn the power off, taking the reset line high will not make PAULMON look for the new baud rate... or change the bytes where it stores the old baud rate... see the code if you're interested). If the minimal system shows no signs of life, it's time to check the wiring, usually starting by making sure you didn't swap the TXD and RXD lines.

The Automatic Baud Rate Detection:

This code was borrowed from MDP/51 by Kei-Yong Khoo. It is run immediately after a system reset. It waits for a <RETURN> character, and uses it to calculate the timer #1 reload value. Some modifications have been made to Khoo's code. It requires only one character. It also stores the reload value in four memory locations in internal RAM (78H, 79H, 7AH, and 7BH). These four locations are unlikely to be changed during a user program's execution or while the debugger is running. When another reset occurs (without removing the power) the program looks at those four locations. If all four agree, then it uses that reload value and does not require another key-press. It is interesting to note that occasionally, with crystal values which produce exact reload values (such an 7.3728 MHz), the baud rate detection routine may not correctly calculate the reload value. Garbage will get printed all over the screen. If this happens, just switch off the power and try again. The advantage of crystals such as the 7.3728 MHz is that they allow transmission at speeds of 9600 and 19200 baud! It is highly recommended that you use the highest possible baud rate with this debugger, as is tends to print quite a bit of text to the screen.

On-line Help:

By typing '?' at the main menu, a help screen summarizing the available commands is printed. On-line help is also available regarding the single step run feature. This help is accessed by typing '?' just after using the 'R' command. While in the single step mode, a summary of commands is also available, again by typing '?'.

The <ESC> key:

The <ESC> key is supported extensively. It will abort all commands from any prompt. It will stop the list and hex dump commands in the middle of their printing. It will also interrupt the printing of text to the screen! This is useful at slow baud rates, since a full screen of text can take quite a while to print at 300 baud.

The Download Program command (type 'D')

This allows you to send the object code from the assembler to the external RAM. The object file must be a standard Intel Hex Format file, such as the .OBJ file created by the Pseudo-Assembler, by Pseudo-Corp. The file must be sent as an ASCII transfer. The protocol such as XMODEM is used. Pressing the <ESC> key at any time will abort the transfer. Please note that most communications programs use the <ESC> key to abort their transfer. In this is the case, the first <ESC> will halt the terminal, pressing it again will abort the receive at the 8051/31. Unlike some other debuggers, PAULMON will recognize the <ESC> key anywhere in the middle of the incoming data, not just at the beginning of a line.

The Run Program command (type 'R')

The run command allows you to execute your program. Two types of run are supported, Normal and Single-Step. The single step mode is explained later, as it is fairly complex. During a normal run, the equivalent of an LCALL to your code is given. During the execution of your program, the debugger obviously has no control of the system, unless of course your program calls one of the subroutines offered by the debugger in the jump table at location 0030H. After specifying which run mode you need, the location of your program is prompted, with the current memory pointer value as the default choice. As is the case at all prompts, pressing the <ESC> key will abort the run command. It is interesting to note that the run command leaves timer #1 in auto-baud rate generation mode. If serial communication is desired at the same baud rate as that used for the debugger, timer #1 need not be given a new reload value. It is recommended that the character input and output routines from the debugger be used via the jump table.

The New Memory Location command (type 'N')

The debugger operates with a pointer to the data memory with which you are working. This pointer is used by the list and hex dump command. It is also the default run location. The pointer is incremented as memory is viewed or modified. Just type 'N' to change it.

The List command (type 'L')

This debugger gives you the ability to list your program's code directly from the computer's memory. All the 8051/31 mnemonics are supported, as well as the names of the special function registers. Bit addressable locations are displayed using the standard syntax (e.g. PWS.2 or 20.5), but individual bit location names are not supported (e.g. SCON.0 will print in place of RI). Obviously, the original labels used in the source code cannot be printed, instead the memory locations are displayed. Other special Intel assembly formats, such as $ and CALL are not supported. However, the list command can provide a reassuring look at the program directly from the memory.

The Hex Dump command (type 'H')

By typing 'H', the next 256 bytes of ram are dumped to the screen in hex and ASCII. The <ESC> key may be pressed to abort the printout.

The Edit command (type 'E')

This command allows you to change the values of memory locations in the external RAM. Each location's old value is shown. If <ESC> is pressed, the current location's value is not changed.

The Jump Table:

Despite the use of the word "jump", the user must LCALL to these locations! The individual locations contain jumps to the subroutines, which all terminate with a RET. The table provides the user with a memory location to call to that WILL NOT CHANGE if the debugger is reassembled. The routines available are:

```
0030: Cout     Sends the byte in Acc to the serial port.
0032: Cin      Waits for a character from the serial port, returned in Acc.
0034: pHex     Prints the two digit hex value in Acc to the serial port.
0036: pHex16   Prints the four digit hex value in DPTR to the serial port.
0038: pString  Prints the string in code memory pointed to by DPTR to the
               serial port. The string must terminate with 00H or a high bit
               set.
003A: gHex     Gets a two digit hex value from the serial port, returned in
               Acc.
003C: gHex16   Gets a four digit hex value from the serial port, returned in
               DPTR.
003E: Esc      Checks to see if the <ESC> key is waiting in SBUF. Clears the
               buffer if it is, and returns with the carry flag set.
               Otherwise, leaves SBUF untouched, and returns with C=0.
0040: Upper    Converts character in Acc to uppercase if it is lowercase.
0042: Init     Automatic baud rate detection.
```

The memory location can be placed directly in your code, or an EQU can be used to make your code more readable. For example:

```
gHex16      EQU   003AH             ;this makes the code nice
Program:    MOV   DPTR, #StrLoc     ;load DPTR
            LCALL gHex16            ;print the DPTR
            MOV   A,#13
            LCALL 0030H             ;print a <RET>
            LCALL 0038H             ;print the string
            RET
StrLoc:     DB    "This is my String.", 0
```

Most of these routines leave the registers unchanged, however, it is a good idea to consult the source code just to be sure... In particular, the pHex routine DESTROYS the contents of Acc, so beware. (This has caused some people some grief, as they had assumed the pHex would leave Acc unchanged. If you want it unchanged, the original .ASM file is provided for you to modify.)

The Single-Step Run:

[This part was never written, and the single step run code is somewhat buggy, primarily due to a lack of available beta testers... so docs were never written, but PAULMON ought to give you enough clues to figure it out if you try.]

PAULMON'S ASSEMBLY HEADER:

This is a template for creating program headers that PAULMON2 can recognize. Using this header, you can make your programs appear in the "Run" command's menu and/or make your programs run automatically when the system is reset (useful if you put your code in non-volatile memory). You can also make programs which are plug-in commands to PAULMON2, either adding new functionality or replacing the built-in commands with your own customized versions.

```
EQU   locat 8000h             ;Location for this program

ORG   locat
DB    0A5H,0E5H,0E0H,0A5H     ;signature bytes
DB    35,255,0,0              ;id (35=prog, 253=startup, 254=command)
DB    0,0,0,0                 ;prompt code vector
DB    0,0,0,0                 ;reserved
DB    0,0,0,0                 ;reserved
DB    0,0,0,0                 ;reserved
DB    0,0,0,0                 ;user defined
DB    255,255,255,255         ;length and checksum (255=unused)
DB    "Program Name",0
ORG   locat+64                ;executable code begins here
```

PAULMON2 will only recognize this header if it begin on a 256-byte page boundary. PAULMON2 can be configured to avoid searching certain ranges of memory. If your copy of PAULMON2 is configured this way remember to write your programs/commands in areas where it is allowed to scan for them.

To create ordinary programs (that show up in the R command's list), just use these lines as they are, but change the string to the name of your program.

If your program is stored in non-volatile memory, you can switch the 35 byte to 253 and PAULMON2 will automatically run your program when it starts up. If your program hasn't changed the stack pointer, it can just terminate in a RET instruction and PAULMON2 will start up normally.

To create plug-in commands for PAULMON2, change the 35 to 254. The keystroke that is to run your command must be specified in place of the 255 byte, for example:

```
DB    254,'W',0,0             ;a new command assigned to the 'W' key
```

If the key is a letter, it must be uppercase. If you use a key which conflicts with the built-in commands, your new command will be override by the built-in one... so be careful.

When PAULMON2 runs your plug-in command, R6 & R7 will contain the value of the memory pointer, which you can change if you like. When your command is finished, it should terminate in a RET instruction. If the stack pointer is different from what it what when PAULMON2 called your command, you will almost certainly crash the machine. Apart from SP, R6, and R7, and the return value on the stack, you may use whatever memory you need. If your command needs to store data to be used next time it is run, 08-0F and 20-2F are areas which PAULMON2 (in it's default configuration) will not use.

The "prompt code vector" is a feature where programs or commands in memory have an opportunity to run and add some text to the prompt that

PAULMON2 prints before accepting each new command. The first two bytes must be 165 and 100, and the second two are the actual location PAULMON2 should call. If your prompt modifying code crashes or doesn't return properly, PAULMON2 will not work, so be careful when using this feature, particularly if downloading to non-volatile memory!

If you create nifty plug-in commands, please consider contributing them to other users. Email paul@ece.orst.edu about getting your plug-in commands on the PAULMON2 web page.

A C LANGUAGE PROGRAM TO RUN WITH PAULMON:

```
#include <reg51.h>
#include <stdio.h>

/* A C language program that runs with PaulMon - Hugo Vieira Neto */

/* PaulMon program header */
#define locat 0x3000

at locat+0x00 char code signature[]={0xA5,0xE5,0xE0,0xA5};
at locat+0x04 char code id       []={0x23,0xff,0x00,0x00};
at locat+0x08 char code prompt   []={0x00,0x00,0x00,0x00};
at locat+0x0c char code reserved1[]={0x00,0x00,0x00,0x00};
at locat+0x10 char code reserved2[]={0x00,0x00,0x00,0x00};
at locat+0x14 char code reserved3[]={0x00,0x00,0x00,0x00};
at locat+0x18 char code user     []={0x00,0x00,0x00,0x00};
at locat+0x1c char code length   []={0xff,0xff,0xff,0xff};
at locat+0x20 char code name     []="Hello";

/* Paulmon interrupt vectors */
at 0x2004 int xdata exter0;
at 0x200c int xdata timer0;
at 0x2014 int xdata exter1;
at 0x201c int xdata timer1;
at 0x2024 int xdata serial;
at 0x202c int xdata timer2;

/* Paulmon interrupt vector setting function */
void pmSetIntVect (void code *isr, int xdata *int_vect)
{
      *int_vect=isr;
      int_vect--;
      *int_vect=*int_vect&0xff02|0x0002;
}  /* end of pmSetIntVect */

/* interrupt service routines */
void ext0 (void) interrupt 0 using 1
{
      printf ("External Interrupt 0!\n");
}  /* end of ext0 */

/* "main" program */
at locat+0x40 void program (void)
{
      pmSetIntVect (&ext0, &exter0);
      IT0=1;
      EX0=1;
      EA=1;
      printf ("Hello world!\n");
      while (1);
}     /* end of "main" program */
```

The “Code Starting Address” must be set to the same value of “locat” (“Options” menu – “Project” option – “Linker” dialog box).

TUTORIAIS

CONFIGURAÇÃO E UTILIZAÇÃO DO PROVIEW:

O ProView é um ambiente integrado de desenvolvimento de software para a família MCS51 de microcontroladores, composto de editor de código-fonte, montador Assembly, compilador C, link-editor e simulador. A figura 1 apresenta o aspecto do ProView ao ser iniciado.

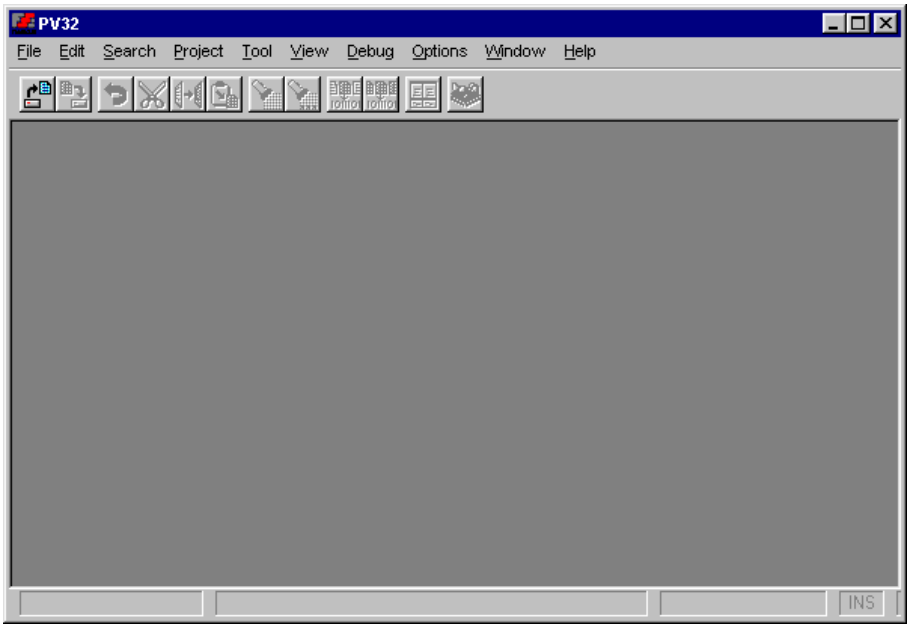


Figura 1: Ambiente Integrado de Desenvolvimento ProView

Para dar início ao desenvolvimento de um projeto, o usuário deverá acessar o menu “Project”, selecionar a opção “New” e definir um nome para o novo projeto (extensão .PRJ), conforme a figura 2.

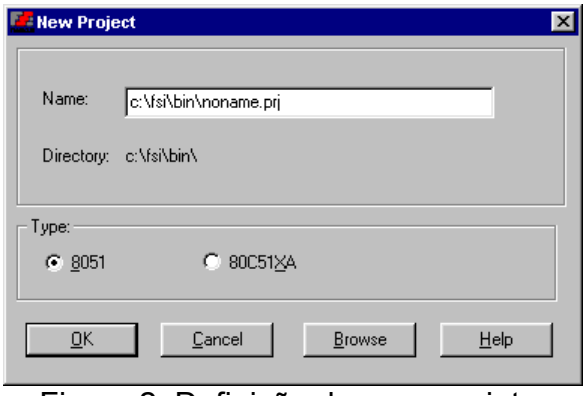


Figura 2: Definição de novo projeto.

Uma vez definido o nome do novo projeto, deve-se acessar novamente o menu “Project” e selecionar a opção “Add file” para definir os arquivos de código-fonte do projeto (figura 3).

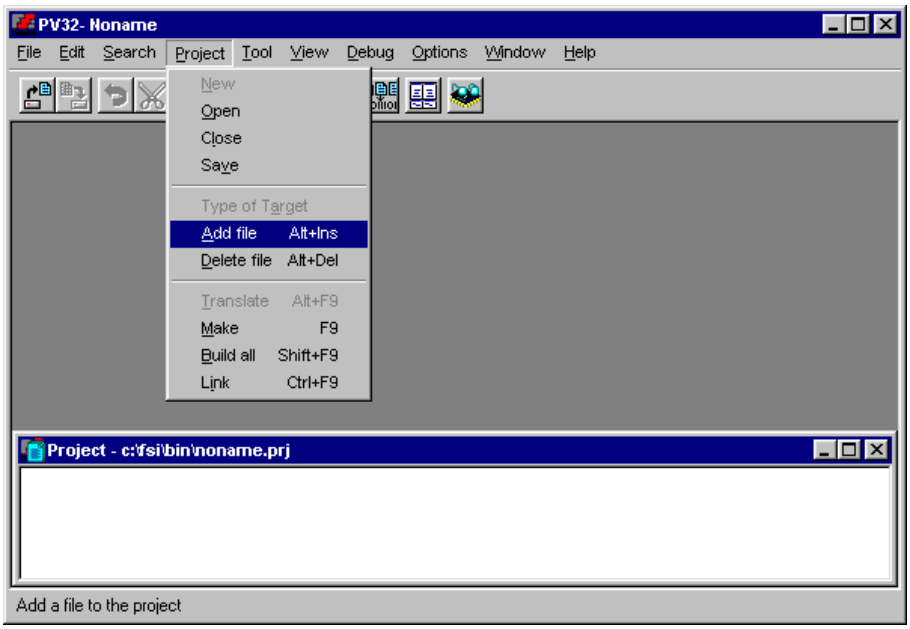


Figura 3: Adição de arquivos de código-fonte ao projeto.

Os arquivos de código-fonte do projeto podem ser já existentes ou não, podendo ser editados no próprio ambiente integrado de desenvolvimento. O ProView possibilita o desenvolvimento de programas na linguagem Assembly da família MCS51 (extensão .ASM) ou na linguagem C (extensão .C). Programas em Assembly são processados pelo montador (A51) e programas em C são processados pelo compilador (C51). A figura 4 apresenta o aspecto da janela do projeto ao serem adicionados arquivos com código-fonte em Assembly e C.

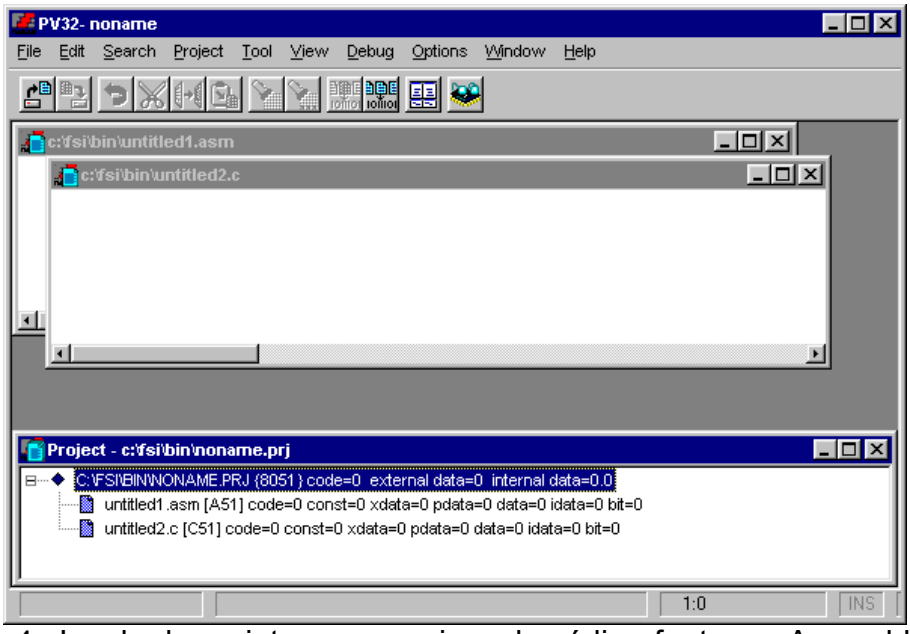


Figura 4: Janela do projeto com arquivos de código-fonte em Assembly e C.

É possível editar os arquivos de código-fonte clicando-se sobre eles na janela do projeto ou acessando o menu “File” e selecionando-se a opção “Open”. Os

detalhes sobre a sintaxe e os recursos do montador e do compilador podem ser verificados através do acesso ao menu “Help”.

Antes de prosseguir com o desenvolvimento do código-fonte, é necessário definir as opções do projeto selecionando o menu “Options” e acessando a opção “Project”. É interessante optar pela geração de arquivos de listagem do código-fonte (extensão .LST), tanto para o compilador C quanto para o montador Assembly, conforme as figuras 5 e 6.

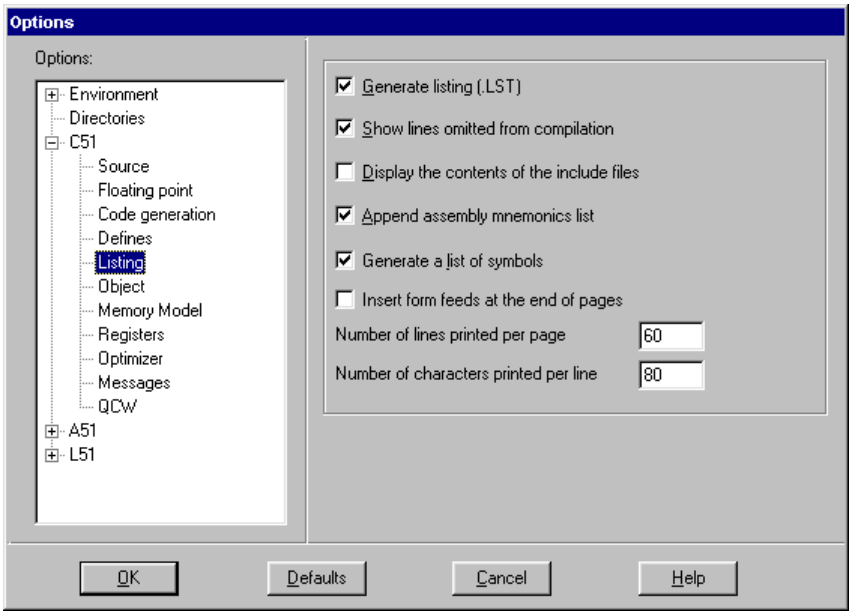


Figura 5: Definição das opções de listagem do compilador.

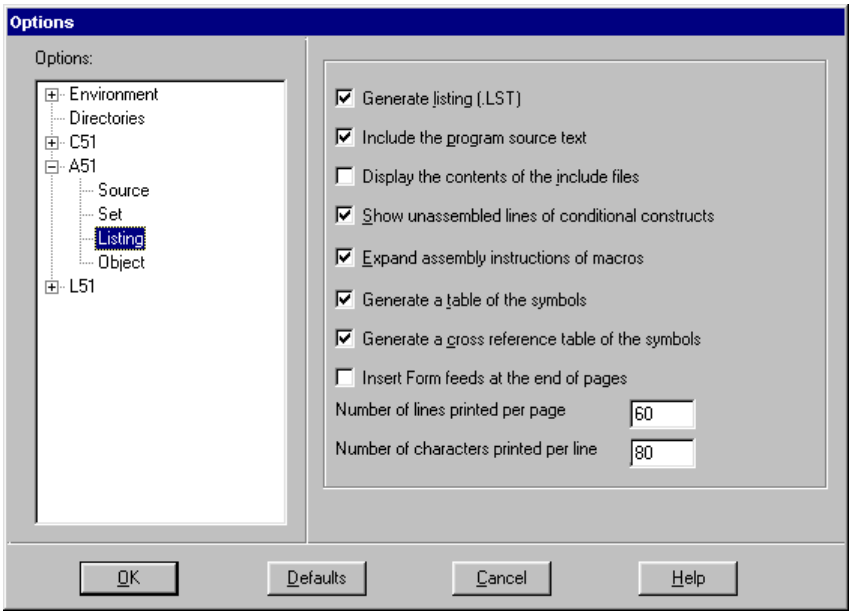


Figura 6: Definição das opções de listagem do montador.

Quando são gerados, os arquivos de listagem contém os resultados da construção do programa pelo compilador e pelo montador, indicando inclusive a

ocorrência de erros e suas localizações. É necessário optar pelo formato de arquivo Intel Hex (extensão .HEX) nas opções do link-editor para que o ProView gere o arquivo final do programa para ser gravado em EPROM (figura 7). O formato Intel Hex é utilizado também para a transferência de programas utilizando o PAULMON.

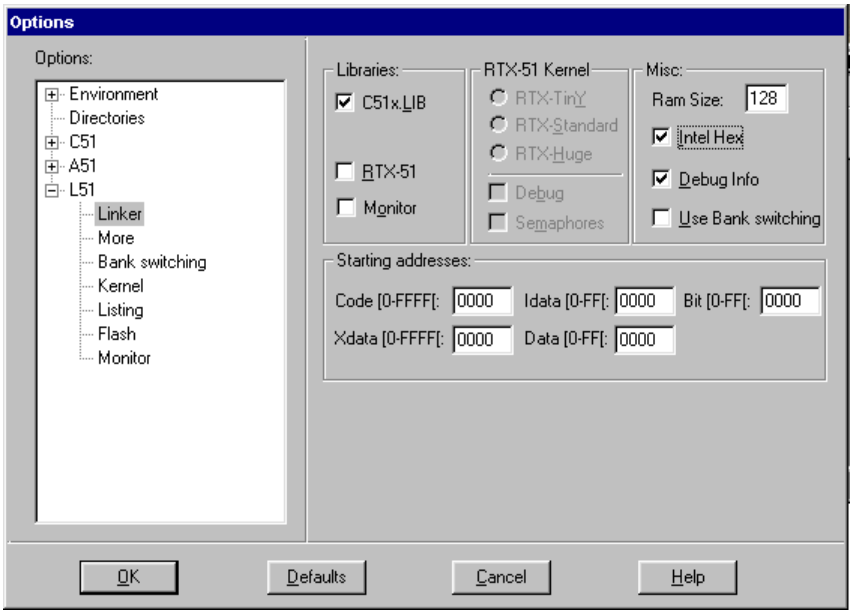


Figura 7: Definição das opções do link-editor.

Uma vez elaborado o código-fonte do projeto, basta construí-lo acessando o menu “Project” e selecionando a opção “Make” (figura 8). O ProView irá indicar eventuais erros e suas localizações para que possam ser corrigidos.

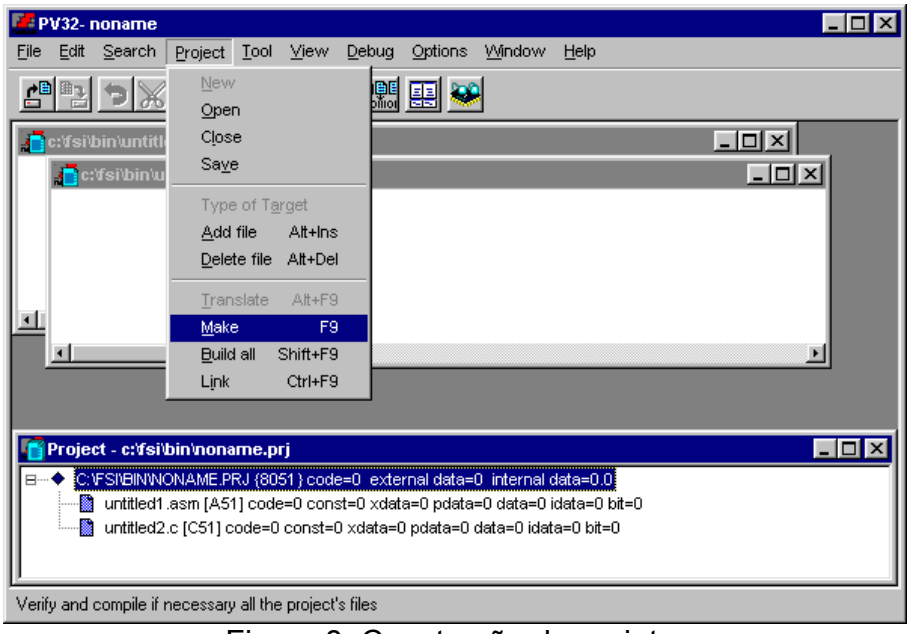


Figura 8: Construção do projeto.

Depois de corrigidos os eventuais erros de sintaxe detectados pelo montador ou compilador, é possível simular o programa construído para verificar possíveis erros de lógica de programação. O simulador pode ser iniciado através do acesso ao menu “Debug” e opção “Start”. A figura 9 apresenta as opções de simulação a serem definidas, tais como o modelo de microcontrolador e a freqüência de clock (cristal oscilador).

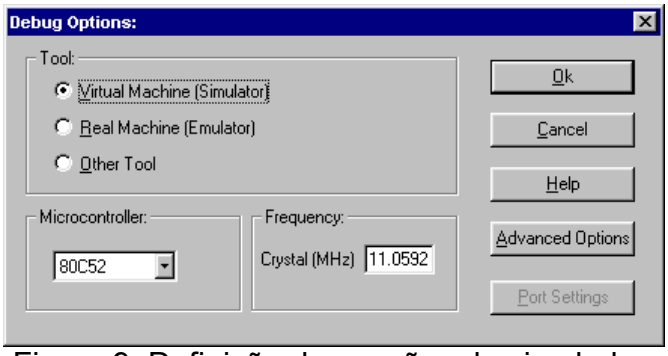


Figura 9: Definição das opções do simulador.

A figura 10 ilustra a simulação de um programa no ProView. Através do menu “View” é possível visualizar o comportamento dos diversos recursos do microcontrolador.

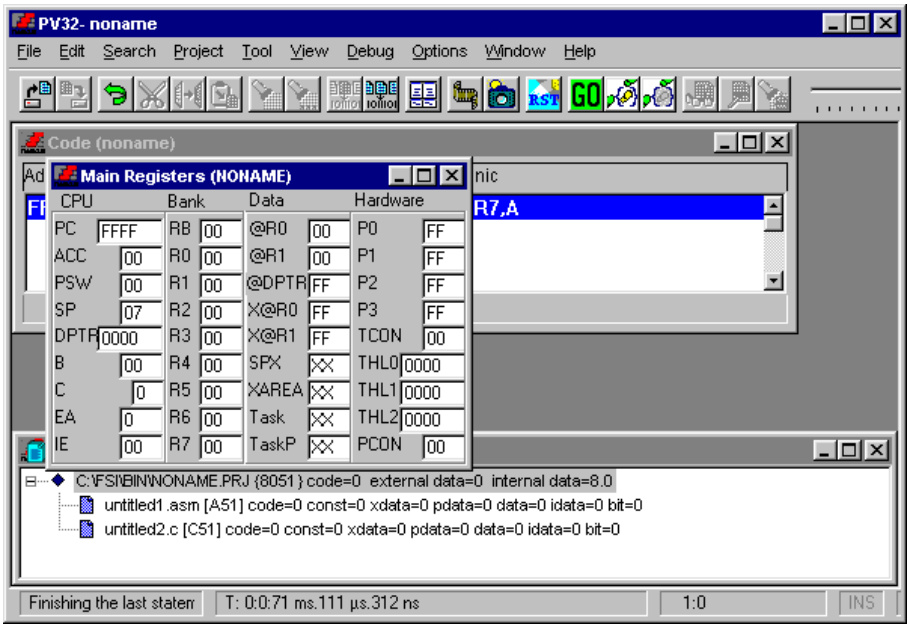


Figura 10: Simulação do projeto.

Através da simulação é possível executar o programa passo-a-passo, observando o efeito de cada instrução e possibilitando, dessa forma, a depuração de eventuais erros no projeto.

CONFIGURAÇÃO E UTILIZAÇÃO DO HYPERTERMINAL:

O PAULMON comunica-se com o usuário através da interface serial assíncrona existente nos microcontroladores MCS51. Para tanto, é necessária a configuração de um aplicativo de comunicações como o HyperTerminal ou outro similar. Ao ser iniciado, o HyperTerminal solicita que seja definido um nome e um ícone para a conexão com o PAULMON (figura 11).

Figura 11: Definição de uma nova conexão.

Em seguida, o HyperTerminal solicita ao usuário que defina a porta de comunicação serial será utilizada na conexão com o PAULMON, conforme ilustrado na figura 12. Uma vez definida a porta de comunicação serial, será necessário configurá-la como mostra a figura 13. Caso a porta de comunicação serial definida já esteja em uso por outro dispositivo, um aviso será apresentado ao usuário (figura 14), sendo necessário reconfigurar novamente o HyperTerminal para outra porta de comunicação serial disponível.

A velocidade máxima de comunicação em bits por segundo depende da freqüência de clock do microcontrolador (19200 bits por segundo é o máximo possível para uma freqüência de 11,0592MHz). O PAULMON possui um recurso de detecção automática da velocidade de comunicação serial, bastando o usuário pressionar a tecla “ENTER” para estabelecer a conexão. Se a conexão for estabelecida com sucesso, a tela de abertura mostrada na figura 15 será apresentada.

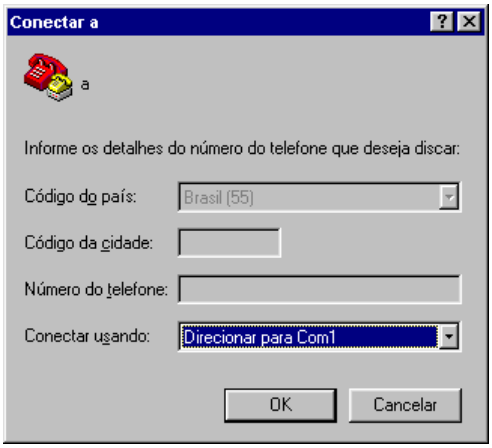


Figura 12: Definição da porta de comunicação serial.

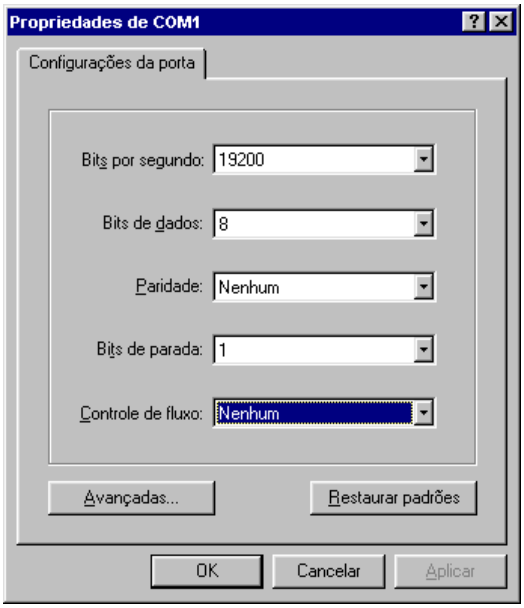


Figura 13: Configurações da porta de comunicação serial.

Figura 14: Mensagem de erro na porta de comunicação serial.

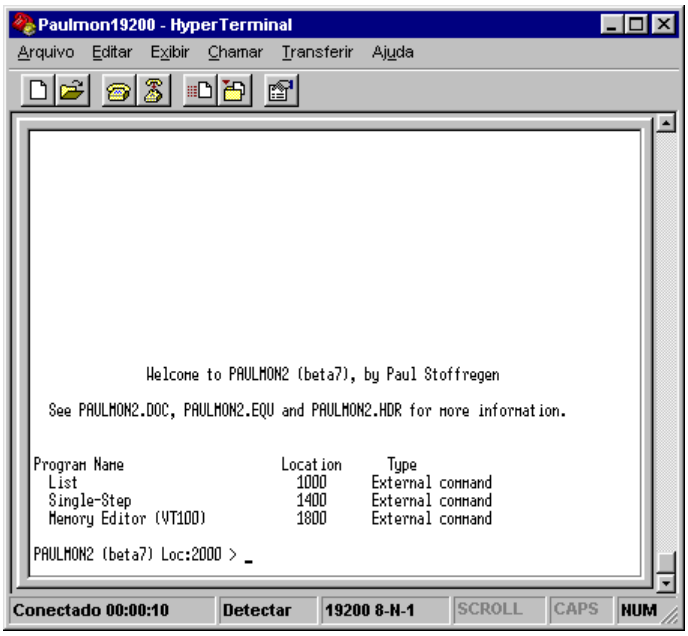


Figura 15: Apresentação do PAULMON.

Caso sejam apresentados caracteres estranhos na janela do HyperTerminal ao se estabelecer a conexão com o PAULMON, deve-se reduzir a velocidade de comunicação e realizar uma nova conexão, desligando e religando a alimentação do microcontrolador. Caso a conexão não se estabeleça, deve-se verificar a integridade da porta de comunicação serial, o cabo de comunicação e a alimentação do microcontrolador.

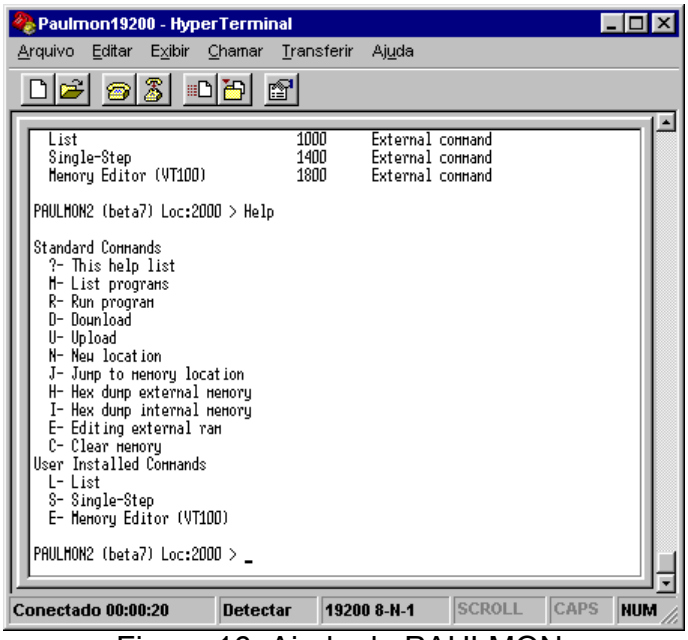


Figura 16: Ajuda do PAULMON.

A figura 16 apresenta a ajuda do PAULMON, obtida a partir do comando “?”. Maiores informações sobre os comandos e funções do PAULMON estão disponíveis documentação fornecida pelo autor.

Para transferir arquivos no formato Intel Hex para o PAULMON é necessário executar o comando “D” e então selecionarar a opção “Enviar arquivo texto” no menu “Transferir” (figura 17). O nome do arquivo a ser transferido será solicitado pelo HyperTerminal, conforme a figura 18.

Figura 17: Comando de transferência de arquivo no formato Intel Hex.

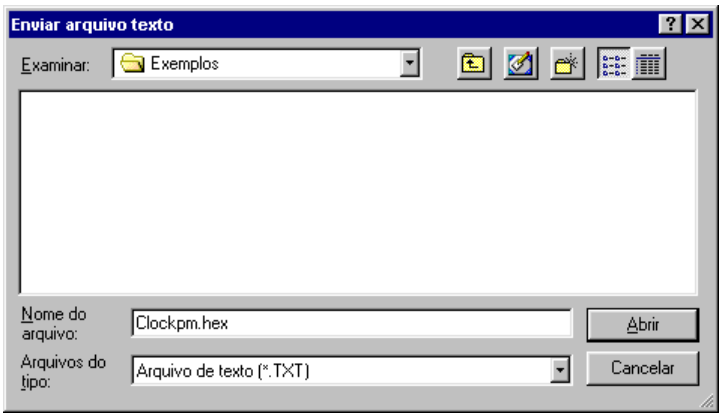


Figura 18: Seleção do arquivo a ser transferido.

O PAULMON mostrará um indicador de progresso da transferência e ao seu término exibirá um sumário das ocorrências, mostrado na figura 19.

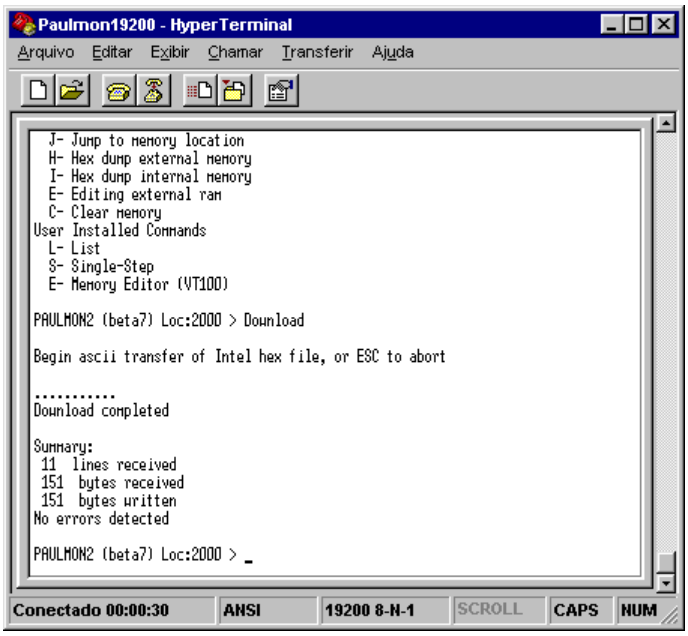


Figura 19: Resultado da transferência de arquivo.

O comando “R” apresenta a lista de programas residentes em memória para execução (figura 20).

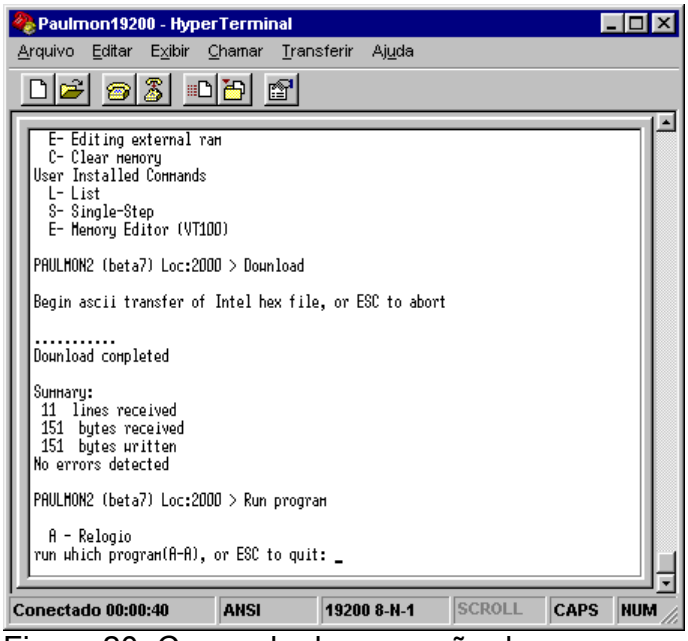


Figura 20: Comando de execução de programa.

Somente programas que contenham o cabeçalho (header) de identificação do PAULMON (consultar documentação) serão exibidos na lista do comando “R”.