

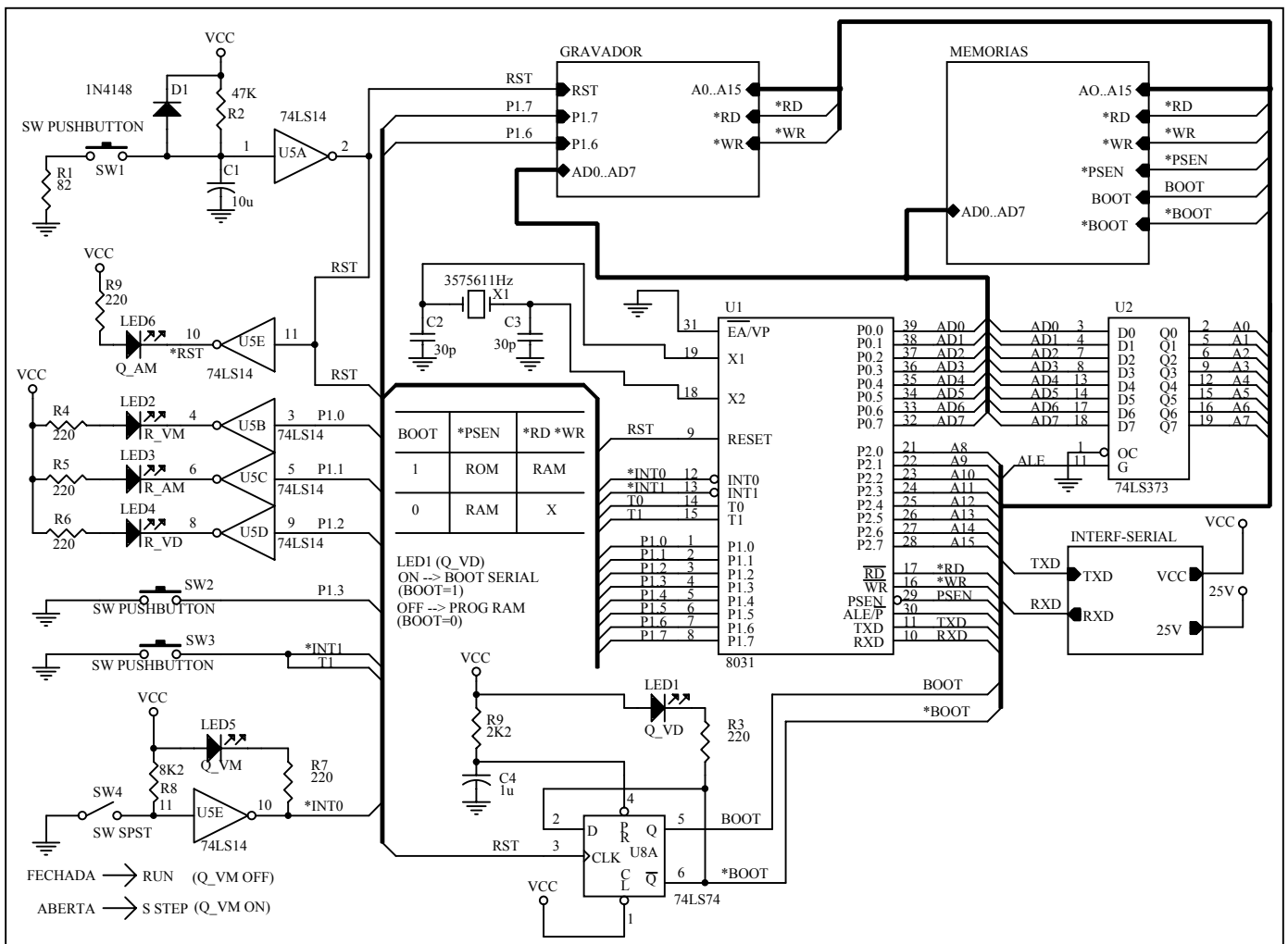
MICRO

CONTROLADORES



MCS-51

RICARDO ZELENOVSKY



MICRO CONTROLADORES

MCS-51[®]

**RICARDO ZELENOVSKY
ASESSOR TÉCNICO ESPE**

DEDICATÓRIA:
a Maria Antonia, Guilherme e Henrique.

ÍNDICE

PREFÁCIO	viii
CAPÍTULO I MICROCONTROLADORES	1-1
1.1. REVISÃO HISTÓRICA	1-1
1.2. POR QUÊ MICROCONTROLADORES ?	1-3
1.3. MICROCONTROLADORES	1-4
1.4. FAMÍLIAS DA INTEL	1-5
CAPÍTULO II FAMÍLIA MCS-51	2-1
2.1. INTRODUÇÃO	2-1
2.2. FAMÍLIA MCS-48	2-1
2.3. FAMÍLIA MCS-51	2-2
2.4. MEMÓRIA EXTERNA	2-3
2.5. MEMÓRIA INTERNA (RAM INTERNA)	2-7
2.6. SFR - SPECIAL FUNCTION REGISTERS	2-9
2.7. MAPA DA RAM INTERNA	2-11
2.8. MAPA DOS BITS	2-13
CAPÍTULO III PINAGEM E TEMPORIZAÇÃO	3-1
3.1. INTRODUÇÃO	3-1
3.2. DESCRIÇÃO DA PINAGEM	3-1
3.3. DIAGRAMAS DE TEMPO	3-3
3.4. RESET	3-6
CAPÍTULO IV CONJUNTO DE INSTRUÇÕES	4-1
4.1. INTRODUÇÃO	4-1
4.2. MODOS DE ENDEREÇAMENTO	4-2
4.3. SOBRE AS INSTRUÇÕES	4-3
4.4. INSTRUÇÕES ARITMÉTICAS	4-3
4.4.1. Soma de 8 Bits	4-3
4.4.2. Soma de 8 Bits com Carry	4-4
4.4.3. Subtração de 8 Bits com Borrow	4-4
4.4.4. Incremento de 8 Bits	4-4
4.4.5. Decremento de 8 Bits	4-4

4.4.6. Incremento de 16 Bits	4-5
4.4.7. Multiplicação e Divisão de 8 Bits	4-5
4.4.8. Ajuste Decimal	4-5
4.5. INSTRUÇÕES LÓGICAS	4-6
4.5.1. AND de 8 Bits	4-6
4.5.2. OR de 8 Bits	4-7
4.5.3. XOR de 8 Bits	4-7
4.5.4. Operações Lógicas com o Acumulador	4-7
4.6. INSTRUÇÕES DE TRANSFERÊNCIA DE DADOS	4-8
4.6.1. Transferência de Dados	4-8
4.6.2. Permutação de Bytes	4-10
4.6.3. Permutação de Nibble	4-10
4.6.4. Operações com a Pilha	4-10
4.6.5. Transferência de Dados com a Memória de Dados Externa	4-11
4.6.6. Leitura da Memória de Programa	4-11
4.7. INSTRUÇÕES BOOLEANAS	4-11
4.7.1. Zerar/Setar/Complementar um Bit	4-12
4.7.2. AND/OR Booleano	4-12
4.7.3. Movimento de Bits	4-12
4.7.4. Desvios Baseados em Bits	4-12
4.8. INSTRUÇÕES DE DESVIO	4-13
4.8.1. Chamadas de subrotinas	4-13
4.8.2. Retorno de Subrotinas	4-13
4.8.3. Desvios	4-14
4.8.4. Desvios Condicionais	4-14
4.8.5. Loops	4-14
4.8.6. Não Operação	4-15
4.9. INSTRUÇÕES E FLAGS	4-15
4.10. OBSERVAÇÕES	4-16
4.10.1. Bancos de Registros	4-16
4.10.2. Registros Especiais	4-17
4.11. CÓDIGOS DE OPERAÇÃO (OP CODES)	4-18
4.11.1. Tabelas de Instruções	4-18
4.11.2. Instruções em Ordem Alfabética com OPCODES	4-21
4.12. JUMP E CALL	4-24
4.12.1. JUMPs Relativos	4-24
4.12.2. JUMPs e CALLs Absolutos	4-26
4.13. EXEMPLOS	4-29

CAPÍTULO V	ASSEMBLER E SIMULADOR	5-1
5.1.	CONCEITOS DO AVMAC51 E DO AVLINK	5-1
5.2.	PSEUDO-INSTRUÇÕES DO ASSEMBLER	5-4
5.3.	O LINKER - AVLINK	5-9
5.4.	O FORMATO INTEL.HEX	5-11
5.5.	PROGRAMAS EXEMPLO	5-12
5.6.	CONCEITOS DO AVSIM 8051	5-17
5.6.1.	Modo Comando	5-17
5.6.2.	Modo Display	5-18
5.7.	COMANDOS DO AVSIM51	5-18
5.7.1.	Comandos de Ambiente	5-19
5.7.2.	Comandos para a Execução de Programas	5-19
5.7.2.1.	Breakpoints	5-19
5.7.2.2.	Condições (Value, Range, Mask, Indirect)	5-20
5.7.2.3.	Passpoints	5-20
5.7.2.4.	Opcode Traps	5-21
5.7.2.5.	Execute Command	5-21
5.7.3.	Comandos de Display	5-21
5.7.4.	Comandos de I/O	5-21
5.7.5.	Comandos de Memória	5-22
5.7.6.	Incremental Cross-Assembler	5-22
5.8.	FLUXOGRAMA DE OPERAÇÃO DO AVSIM51	5-22
5.9.	TELAS DE AJUDA	5-31
5.9.1.	Ajuda para os Comandos	5-31
5.9.2.	Ajuda para o Display	5-31
5.9.3.	Ajuda para a Simulação	5-31
5.9.4.	Ajuda para AVOCET	5-32
5.9.5.	Tela do Simulador	5-32
CAPÍTULO VI	PORTAS PARALELAS	6-1
6.1.	REGISTROS ENVOLVIDOS	6-1
6.2.	DESCRIÇÃO DO FUNCIONAMENTO	6-2
6.2.1.	Porta P1	6-2
6.2.2.	Porta P3	6-3
6.2.3.	Porta P2	6-4
6.2.4.	Porta P0	6-5

6.3. ESCRITA NAS PORTAS	6-6
6.4. EXERCÍCIOS	6-9
 CAPÍTULO VII INTERRUPÇÕES	 7-1
7.1. INTRODUÇÃO	7-1
7.2. REGISTROS ENVOLVIDOS	7-1
7.3. MANEJO DE INTERRUPÇÕES	7-4
7.4. INTERRUPÇÕES EXTERNAS	7-5
7.5. PASSO A PASSO	7-6
7.6. EXERCÍCIOS	7-8
 CAPÍTULO VIII TEMPORIZADORES / CONTADORES	 8-1
8.1. INTRODUÇÃO	8-1
8.2. REGISTROS ENVOLVIDOS	8-1
8.3. MODOS DE OPERAÇÃO	8-4
8.3.1. Modo 0	8-4
8.3.2. Modo 1	8-4
8.3.3. Modo 2	8-5
8.3.4. Modo 3	8-5
8.4. EXERCÍCIOS	8-6
 CAPÍTULO IX PORTA SERIAL	 9-1
9.1. INTRODUÇÃO	9-1
9.2. REGISTROS ENVOLVIDOS	9-1
9.3. MODOS DE OPERAÇÃO	9-3
9.3.1. Modo 0	9-3
9.3.2. Modo 1	9-4
9.3.3. Modo 2	9-5
9.3.4. Modo 3	9-5
9.4. CUIDADOS COM A PORTA SERIAL	9-6
9.5. Comunicação entre vários 8051	9-7
9.6. Comunicação serial entre o 8051 e o PC	9-8
9.7. EXERCÍCIOS	9-9
 CAPÍTULO X ECONOMIA DE ENERGIA E GRAVAÇÃO	 10-1
10.1. INTRODUÇÃO	10-1
10.2. MODO IDLE	10-2
10.3. MODO POWER DOWN	10-3

10.4. PROGRAMAÇÃO DA EPROM (8751)	10-3
10.4.1. Programação	10-4
10.4.2. Verificação	10-5
10.4.3. Bit de Segurança	10-6
10.4.4. Apagamento (8751)	10-6
 CAPÍTULO XI PLACA DE TESTES	 11-1
11.1. INTRODUÇÃO	11-1
11.2. ESQUEMA DA CPU (CPU.SHT)	11-7
11.3. ESQUEMA DA MEMÓRIA	11-11
11.4. ESQUEMA DA SERIAL	11-13
11.5. ESQUEMA DO GRAVADOR	11-17
 ANEXO A USO DOS PROGRAMAS	 A-1
A.1. INTRODUÇÃO	A-1
A.2. TAR_PRU.ASM	A-1
A.3. TUDO.C	A-2
A.4. STEP.ASM / STEP.H	A-3
 BIBLIOGRAFIA	 B-1

PREFÁCIO

Este trabalho surgiu para concretizar os conhecimentos e idéias desenvolvidas nos Seminários de Microcontroladores ministrados na Faculdade de Eletrônica da Escola Politécnica Superior do Exército Equatoriano. Nestes seminários se desenvolveu um conjunto hardware-software que provou ser muito didático para a aprendizagem de microcontroladores da família MCS-51.

Aqui se pretende estudar de forma prática a família MCS-51. Portanto, junto com a exposição da teoria serão provados os conceitos no hardware desenvolvido. Isto permitirá uma sólida aprendizagem por parte dos estudantes. Este hardware recebe os programas pela porta serial de um computador e os executa em seguida. Isso permite uma interação dinâmica por parte dos estudantes que podem escrever programas e testa-los imediatamente. Se houver erros, estes são corrigidos e o novo programa é novamente enviado ao hardware.

Todo o estudo se baseia no 8031, o membro mais simples da família MCS-51. Serão estudadas entretanto as principais diferenças para os demais membros. Neste estudo se supõe que os estudantes possuem conceitos de eletrônica, microprocessadores (8080, 8085 ou Z-80) e programação assembly. Também será útil a disponibilidade de um computador compatível com IBM PC.

O texto está organizado por capítulos e apêndices.

No **Capítulo 1** foi feita uma pequena revisão histórica, se conceitua o que é um microcontrolador e se apresenta uma pequena relação com os principais fabricantes da família MCS-51.

No **Capítulo 2** se conceitua a família MCS-51 e são apresentados os principais membros desta. Aqui se estuda a conexão com as memórias e também a arquitetura da RAM interna.

No **Capítulo 3** são apresentados todos os pinos do microcontrolador e descritas suas funções. Também são estudados os diagramas de tempo. Neste capítulo é apresentado o hardware a ser construído.

O **Capítulo 4** trata do estudo e prática do conjunto de instruções da família MCS-51. Ao final são propostos diversos exercícios de programação, parte deles com as respostas.

No **Capítulo 5** são apresentadas as ferramentas de programação. Basicamente se utilizará o assembler e simulador da Avocet (AVMAC51, AVLINK, AVSIM51). As ferramentas mais importantes serão o assembler e o linker. O simulador é estudado somente para ser usado como um meio auxiliar pois se tem um hardware para testar os programas desenvolvidos.

A partir do **Capítulo 6** serão estudados e praticados os recursos que a família MCS-51 oferece. Neste Capítulo são estudadas as portas paralelas e em seguida se fazem práticas com o hardware.

No **Capítulo 7** são apresentadas as interrupções e também práticas no hardware. Aqui se propõem os princípios para realização de um controle passo a passo.

No **Capítulo 8** são estudados e praticados os contadores/temporizadores.

O **Capítulo 9** aborda a porta serial. Aqui também se propõe um pequeno programa que permite ao hardware receber os programas do PC via porta serial.

No **Capítulo 10** se estuda a economia de energia e a programação da EPROM que há em alguns membros da família. Aqui se propõe um pequeno gravador para essa EPROM.

No **Apêndice 1** se descreve completamente o programa "BOOT SERIAL" que permite ao hardware receber os programas que o PC envia pela porta serial. Aqui está a listagem completa do programa.

No **Apêndice 2** é descrito o programa "SINGLE STEP" que permite ao hardware o controle passo a passo. É apresentada uma listagem completa dos programas.

No **Apêndice 3** se faz um pequeno estudo das portas seriais do PC. Aqui se explica o programa "TODO" e também uma listagem completa em C.

Este trabalho é uma realidade graças à amizade de Brasil e Equador, em particular, se deve à cooperação que há entre os Exércitos destes dois países irmãos. Agradeço o apoio recebido na ESPE, particularmente por parte do Departamento de Ensino e da Faculdade de Eletrônica. Aos professores desta Faculdade se deve o desenvolvimento dos seminários e sugestões aqui presentes. A tradução para espanhol pôde ser feita graças ao empenho dos alunos de Nível X de 1993 e de Nível IX de 1994.

Devo agradecer os engenheiros Ruben León e Vinicio Carrera pelas sugestões e empenho nos trabalhos de correção deste texto, ao engenheiro Fabricio Morales pela colaboração no desenvolvimento dos programas e aos alunos Eduardo Torres e Ivan Cisneros se deve a construção e testes do novo hardware didático.

Agradeço ao aluno Marco Zavala pela ajuda nas apuradas correções.

RICARDO ZELENOSKY

CAPÍTULO I

MICROCONTROLADORES

1.1. REVISÃO HISTÓRICA

A rápida evolução da eletrônica, particularmente na segunda metade do século XX, provocou uma mudança profunda no homem moderno. Os países ficam cada vez mais próximos, graças inicialmente ao rádio, depois à televisão e hoje em dia, com os satélites, pode-se instantaneamente ver qualquer lugar do planeta. Podemos ver o homem pisando pela primeira vez na Lua e também imagens recém enviadas de Marte, Júpiter, Saturno e Plutão.

Além dos meios de comunicação, recordemos os avanços da medicina, meteorologia, transportes, ciência pura e aplicada, prospecção de petróleo e também o controle e melhoria do meio ambiente.

Pelo lado da eletrônica, afirma-se que um grande agente deste progresso foi o computador, que provocou uma realimentação positiva nas pesquisas. O computador permitiu pesquisas mais rápidas e precisas; com isto muitas ciências se beneficiaram, inclusive a tecnologia eletrônica, que passou a fabricar circuitos mais rápidos e eficientes. Com isso pode-se construir computadores mais eficientes, que por sua vez melhoraram ainda mais a eletrônica e assim sucessivamente.

Desde as simples calculadoras de válvulas, programadas por fios, da década do 40 até os supercomputadores dos dias de hoje, foi percorrido um caminho muito grande. No meio deste caminho, no início da década do 70, com o avanço dos LSI vimos surgir os microprocessadores e os computadores pessoais.

É oportuno recordar algumas datas importantes no desenvolvimento dos computadores:

1948 - John Barden, Walter Bratain e William Shockley inventam o Transistor no BELL LABS.

1959 - TEXAS INSTRUMENTS cria o primeiro Circuito Integrado (CI), onde em um mesmo substrato de cristal eram integrados vários transistores.

1964 - DIGITAL começa a vender o PDP-8, o primeiro computador com preço acessível aos laboratórios.

1968 - Surge a INTEL.

1971 - INTEL fabrica o 4004, o primeiro microprocessador. Ele tinha uma arquitetura de 4 bits. Aqui surge a idéia de integrar todo o circuito de controle em um único CI; isso passou a chamar-se microprocessador.

- 1974 - INTEL desenvolve o 4004 e produz o primeiro microprocessador de 8 bits, o 8080. Neste mesmo ano, Kernighan e Ritchie formalizam a linguagem C.
- 1975 - ZILOG começa a vender o Z80 e a MOS TECHNOLOGY começa a vender o MC6501 (US\$20) e o MC6502 (US\$ 25). Nesta época um 8080 custava US\$ 150.
- 1976 - INTEL produz o primeiro microcontrolador, o 8048, e o 8748. Neste mesmo ano INTEL iniciou o projeto do 8086. A TEXAS INSTRUMENTS produz o TMS 9000, primeiro microprocessador de 16 bits. APPLE COMPUTER também surge neste ano.
- 1977 – A APPLE Computer produz o APPLE II (US\$1298) com processador Motorola 6502 (8 bits).
- 1978 – A INTEL começa a produzir o 8086 e também alguns derivativos do 8048, o 8041 e o 8741.
- 1979 – A INTEL distribui o 8088.
- 1980 – A INTEL inicia o que seria a família de microcontroladores de maior sucesso, os 8051 e 8751. Neste mesmo ano também produz o 8087. A APPLE COMPUTER produz o APPLE III, que estava destinado ao fracasso.
- 1981 – A IBM passa a dedicar-se aos sistemas de pequeno porte com o IBM PC, que consistia da CPU 8088, 64 KB RAM, 40 KB ROM, floppy 5,25" (US\$3005).
- 1982 – A INTEL inicia a venda dos 80186, 80188 e 80286. Também começa a vender o primeiro microcontrolador de 16 bits, o 8096 (família MCS-96).
- 1983 - Com os 80C51 e 80C49, a INTEL começa a distribuir microcontroladores CHMOS, de menor consumo de energia. A APPLE COMPUTER produz o LISA, também destinado ao fracasso. A AT&T começa a distribuir o UNIX System V. A IBM anuncia o PC XT (US\$ 4995) e o PC Jr. (US\$1269).
- 1984 - Surge o IBM PC AT com 80286, 256 KB RAM, floppy de 1,2 MB (US\$ 5469). A APPLE COMPUTER produz o que viria a ser um grande sucesso: o MACINTOSH (US\$ 2495).
- 1985 – A INTEL produz o 80386DX (16 MHz, 6 MIPS).
- 1986 – A COMPAQ fabrica o primeiro computador 386, o COMPAQ DESKPRO 386.
- 1988 – A INTEL fabrica o 80386SX (16 MHz, 2,5 MIPS).
- 1989 – A INTEL fabrica o 80486DX (25 MHz, 20 MIPS).
- 1991 – A INTEL fabrica o 80486SX (20 MHz, 16,5 MIPS).
- 1992 – A INTEL fabrica o 80486DX2 (50 MHz, 40 MIPS).
- 1993 – A INTEL fabrica o PENTIUM (60 MHz, 112 MIPS).

O desenvolvimento dos microprocessadores e dos microcontroladores, de acordo com os fabricantes, pode ser esquematizado da seguinte forma:

INTEL

4004 → 8008 → 8080 → 8085 → 8086 (8088) → 80286 → 80386 → 486 → P5 → PII → PIII → PIV

MOTOROLA

6502 → 6509 → 68000 → 68010 → 68020 → 68030 → 68040 → 68060 → PowerPC

ZILOG

Z80 → Z800 → Z8000

1.2. POR QUÊ MICROCONTROLADORES ?

Com o barateamento dos CIs e o surgimento de microprocessadores (CPUs) mais poderosos, começou-se a usar as CPUs mais simples para implementar tarefas dedicadas, tais como controle de impressora, plotter, reguladores de velocidade, acionadores de motores de passo, controladores de elevadores, etc.

Contudo, qualquer controle implicará uma circuitaria muito grande, que muitas vezes encarece o custo do controlador.

Tipicamente:	CPU	→ controle
	ROM	→ programa de controle
	RAM	→ pilha e dados
	Porta Paralela	→ periféricos e I/O
	Porta Serial	→ comunicação
	A/D e D/A	→ sinais analógicos
	Timers	→ temporização

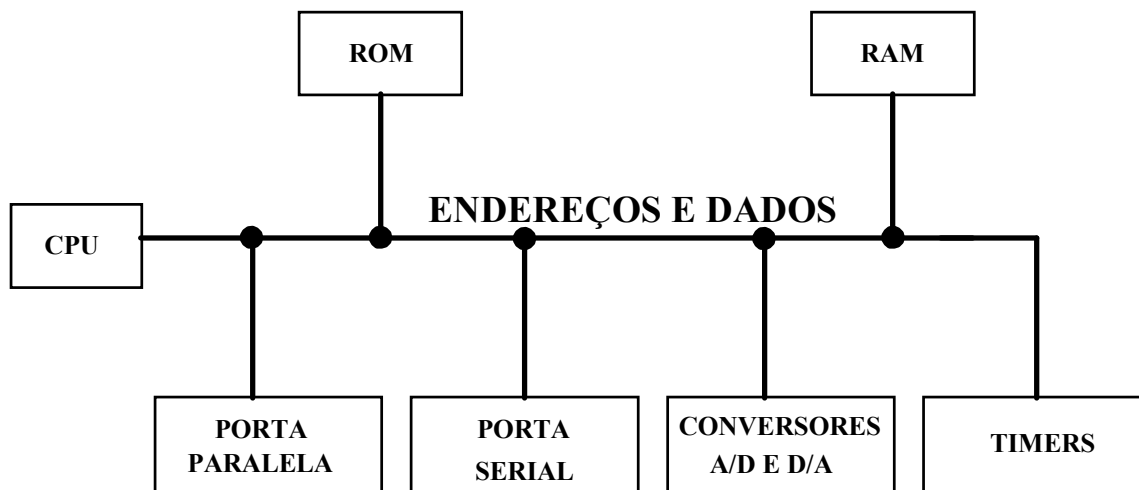


Figura 1.1. Exemplo típico de um microprocessador aplicado em controle.

Estas aplicações tinham o custo dependente do preço da CPU e dos periféricos (ROM, RAM, Portas, A/D, D/A, etc) e também da quantidade de conexões e do tamanho da placa. Para

reduzir o custo, começou a surgir a idéia de colocar todos estes periféricos dentro do chip da CPU. Isso baratearia e diminuiria o tamanho do circuito impresso além de aumentar a confiabilidade. Por outro lado, uma CPU dedicada a um determinado controle não precisa ser muito rápida nem tampouco ter um conjunto de instruções extenso e poderoso. Não são necessárias instruções para trabalhar com ponto flutuante, com strings ou vetores e também os mecanismos de endereçamento devem ser simples. Ou seja, pode-se simplificar a CPU. Assim surgem os microcontroladores, que são simples, baratos e eficientes.

1.3. MICROCONTROLADORES

Os microcontroladores apresentam uma série de recursos incorporados dentro de um único integrado. Estes recursos aumentam com a evolução da eletrônica. Isto permite o desenvolvimento de projetos cada vez mais simples.

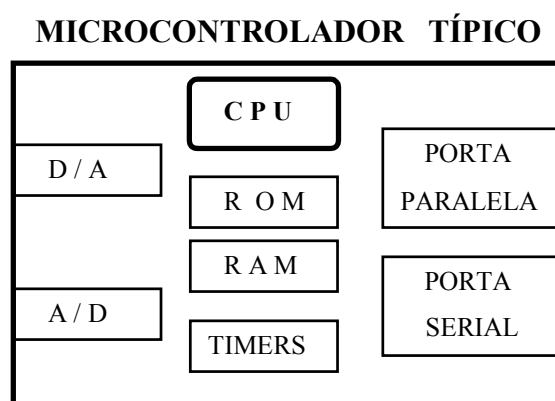


Figura 1.2. Arquitetura básica de um microcontrolador.

Os microcontroladores são específicos para controle, não tem grande capacidade de processamento e por isso nunca haverá um computador pessoal cuja CPU seja um microcontrolador. Eles podem estar presentes em um PC, mas apenas para controlar periféricos.

Usa-se o nome de **Microcontrolador** para designar dispositivos de uso genérico, mas existem vários microcontroladores que têm aplicações específicas, como por exemplo o controlador de teclado 80C51SL-BG e o controlador de comunicações universal 82C152.

Há diversos fabricantes de microcontroladores. Os mais conhecidos são:

- INTEL → 8048, 8049, 8051, 8052, 8096.
- ZILOG → Z8.
- MOTOROLA → 6801, 6804, 6805, 68HC11.
- NATIONAL → COP400, COP800, NS8050.

1.4. FAMÍLIAS DA INTEL

A INTEL possui três famílias de controladores:

8 bits: MCS-48 (obsoleta) e MCS-51

16 bits: MCS-96

Hoje em dia, os manuais mostram uma nova família, MCS-80/85. Agora os antigos 8080 e 8085 estão sendo classificados como controladores. Isto é porque são muito simples quando comparados com um 386 ou um 486. Cada família possui um núcleo básico (arquitetura e instruções) a partir do qual se derivam vários produtos:

MCS-48 → 8048, 8748, 8049, 8749, 8035, 8039, 8050, 8040, etc.

MCS-51 → 8031, 8051, 8751, 8032, 8052, 8752, 8054, 8754, 8058, 8758, 8351FA, 8051FA, 8751FA, 8051GB, 8751GB, 8051SL-BG, 83152, etc.

MCS-96 → 8096, 8098, 80196, 83196, 87196, 80198, 83198, 87198, 80193, 83193, 87193, etc.

São 3 os principais fabricantes do MCS-51:

I N T E L
P H I L I P S
A M D

CAPÍTULO II

FAMÍLIA MCS-51

2.1. INTRODUÇÃO

Este curso abordará somente a família MCS-51. Ela não é tão antiga e limitada como a MCS-48 nem tão cara como a MCS-96. Por isso mesmo é atualmente a família de controladores de maior emprego. É adequada para a grande maioria de aplicações a nível universitário.

2.2. FAMÍLIA MCS-48

A família MCS-51 originou-se a partir da MCS-48, daí o motivo de seu estudo neste curso. Esta foi a primeira família de controladores lançada no mercado pela INTEL. As limitações tecnológicas da época (1976) impuseram uma série de restrições mas, para a época, foi um grande produto. Uma de suas maiores aplicações foram os teclados dos computadores de 16 bits, IBM PC (lançados em 1981).

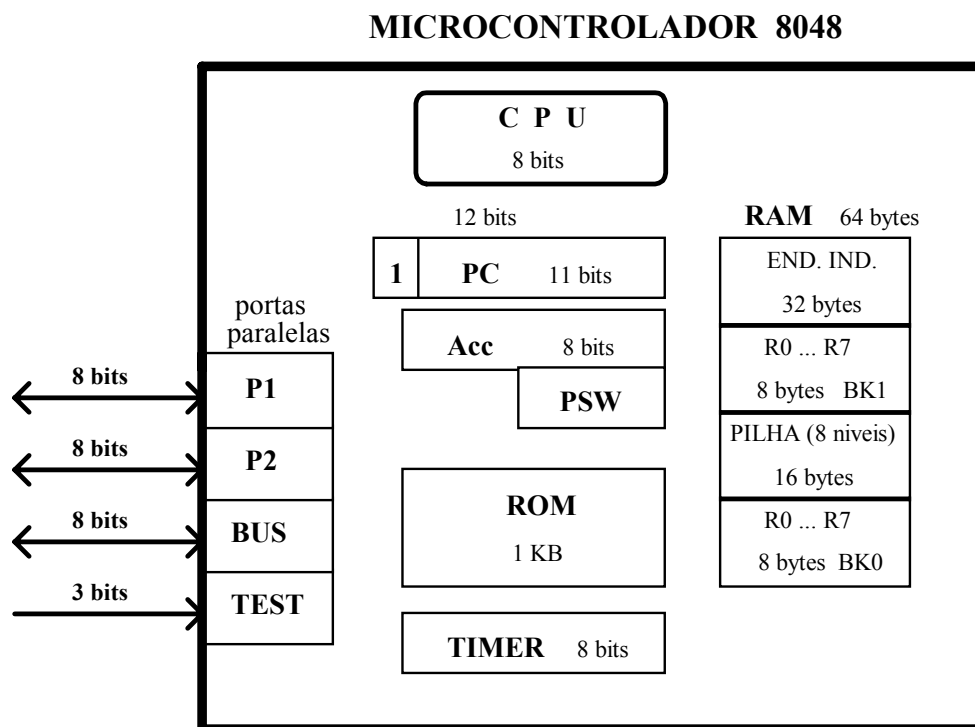


Figura 2.1. Diagrama em blocos do 8048.

O 8048 oferece os seguintes recursos:

- Interrupções (em 1 nível somente),
- Single Step,
- 1K de ROM interna (8748 tinha uma EPROM de 1K),
- Memória de Programa Externa de até 4 KB (PC=12 bits com o bit mais significativo, alterado pela instrução SEL, chaveando os 2 bancos de 2K),
- Aceita outros periféricos (8155,8255,8355,8243,8279),
- 96 instruções, 90% de 1 byte.

2.3. FAMÍLIA MCS-51

O êxito da família MCS-48 e os avanços da tecnologia dos circuitos integrados levaram a INTEL a lançar no mercado a família MCS-51 em 1980. O microcontrolador 8051 é o membro original da família e também o núcleo para toda a MCS-51.

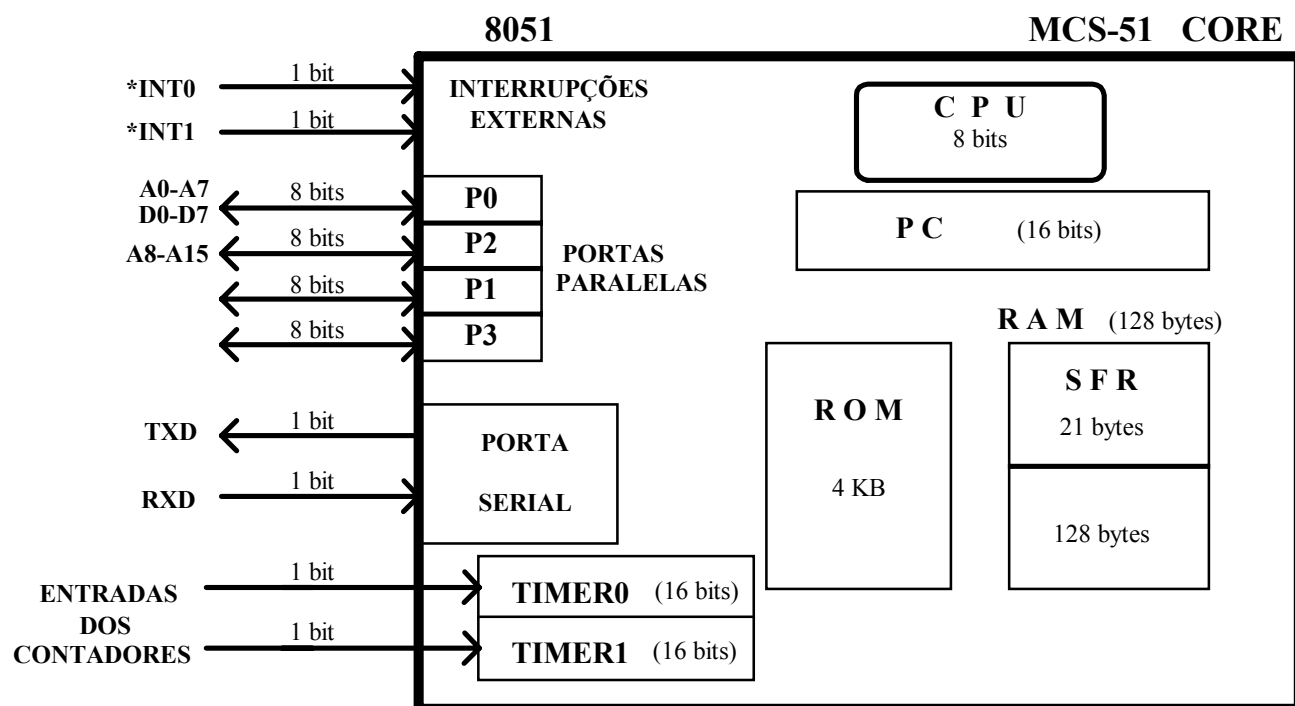


Figura 2.2. Diagrama em blocos do 8051.

O 8051 oferece os seguintes recursos:

- 5 Interrupções (2 externas, 2 dos timers/counters e 1 da porta serial),
- 64 KB de Memória de Programa (PC=16 bits),
- 64 KB de Memória de Dados,

- 111 Instruções:
 - 1 ciclo → 64 → 58%
 - 2 ciclos → 45 → 40%
 - 4 ciclos → 2 → 2%
 - 1 byte → 49 → 44%
 - 2 bytes → 46 → 41%
 - 3 bytes → 16 → 15%
- 98% de 1 ou 2 ciclos → velocidade
- 85% de 1 ou 2 bytes → compacto

2.4. MEMÓRIA EXTERNA

O 8051 não segue a Arquitetura de Von Newman mas sim a chamada Arquitetura de Harward, portanto pode ter programas e dados em memórias distintas.

São chamadas de Memórias Externas:

→ 64 KB Memória de dados (*RD e *WR)

→ 64 KB Memória de programa (*PSEN – Program Store Enable)

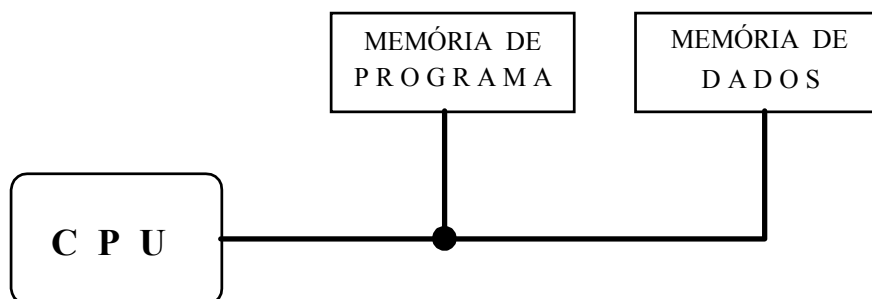


Figura 2.3. Memórias externas do 8051.

Os 4 KB de ROM interna podem ser usados ou não, de acordo com o estado do pino *EA (External Access Enable):

→ se *EA=0 → 64 KB de programa externo

→ se *EA=1 → 4 KB de ROM interna

→ 60 KB de programa externo

Pinos importantes para interface com Memórias externas:

*RD → leitura na memória de dados externa

*WR → escrita na memória de dados externa

*PSEN → leitura na memória de programa

- P0** → multiplexado com endereços (A0-A7) e dados (D0-D7)
P2 → endereços A8-A15
ALE → Address Latch Enable. Sinal para demultiplexar P0
***EA** → External Access Enable. Especifica o uso de memória de programa externa

Para executar um programa a partir de uma memória externa bastam algumas conexões simples:

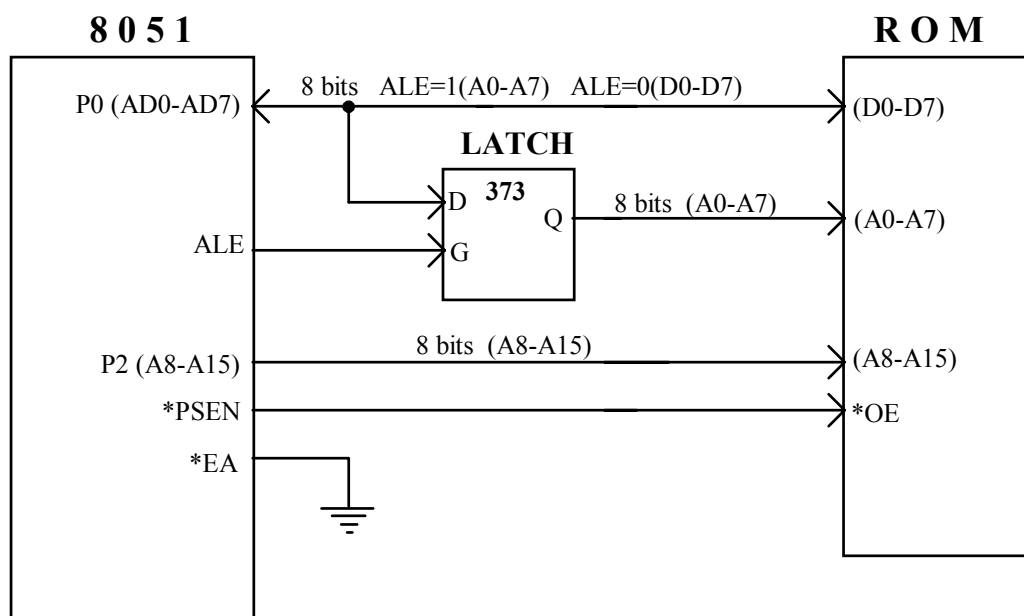


Figura 2.4. Microcontrolador com Memória de Programa externa.

Deve-se notar que a memória de programa só pode ser lida. São sempre emitidos endereços de 16 bits, por isso **as portas P0 e P2 são sacrificadas quando se usa memória de programa externa.**

Um acesso à memória de dados externa tem uma configuração muito semelhante:

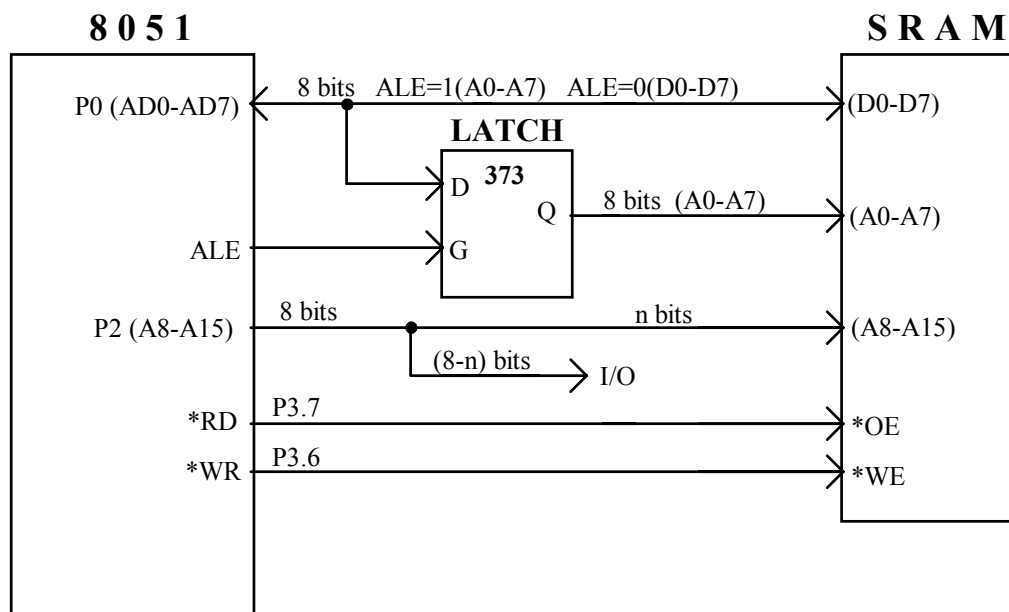


Figura 2.5. Microcontrolador com Memória de Dados externa.

Deve-se notar o uso de dois bits da porta P3. Por outro lado, se todo o programa está na ROM interna, a porta P2 não é totalmente sacrificada; somente são usados os bits necessários para emitir os endereços.

Exemplo:

RAM externa de 2 KB → 8 bits de P0

→ 3 bits de P2

Sobram 5 pinos da porta P2.

OBSERVAÇÃO:

No caso explicado acima, onde são usados somente 3 bits de P2, um acesso à memória de dados não pode ser efetuado com a instrução **MOVX A,@DPTR**. Deve ser trocada por:

```
MOV    P2,# ? ? ? ? ? A10 A9 A8    ;especificar MSB do endereço
MOV    R0,# A7 A6 A5 A4 A3 A2 A1 A0 ;especificar LSB do endereço
MOVX   A,@R0                       ;efetuar a leitura
```

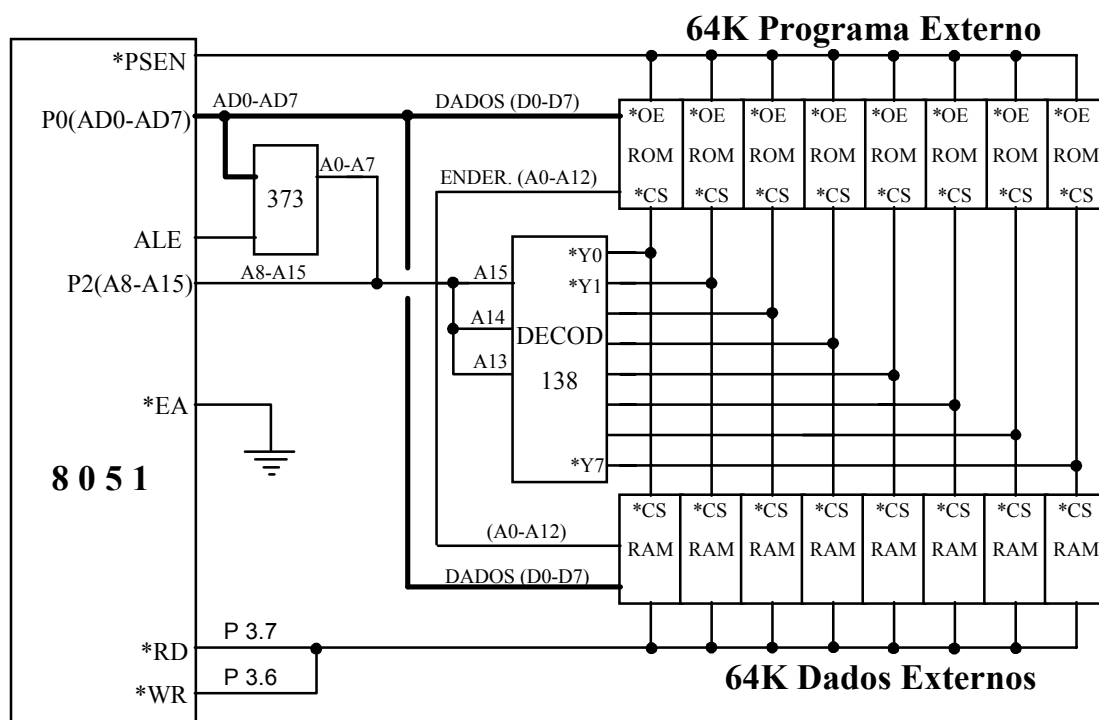


Figura 2.6. O 8051 com 64 KB de Programa e 64 KB de Dados externos.

A figura 2.6 mostra uma utilização exaustiva de memórias de programa e dados. Como os sinais *PSEN e (*RD e *WR) são mutuamente exclusivos, foi possível usar um mesmo decodificador. As portas P0, P2 e os dois bits da porta P3 foram sacrificados. Os buffers foram omitidos para simplificar o desenho.

É possível implementar uma Arquitetura de Von Neumann usando o 8031. Neste caso há um limite de 64 KB de RAM para programa e dados. A figura 2.7 ilustra o circuito.

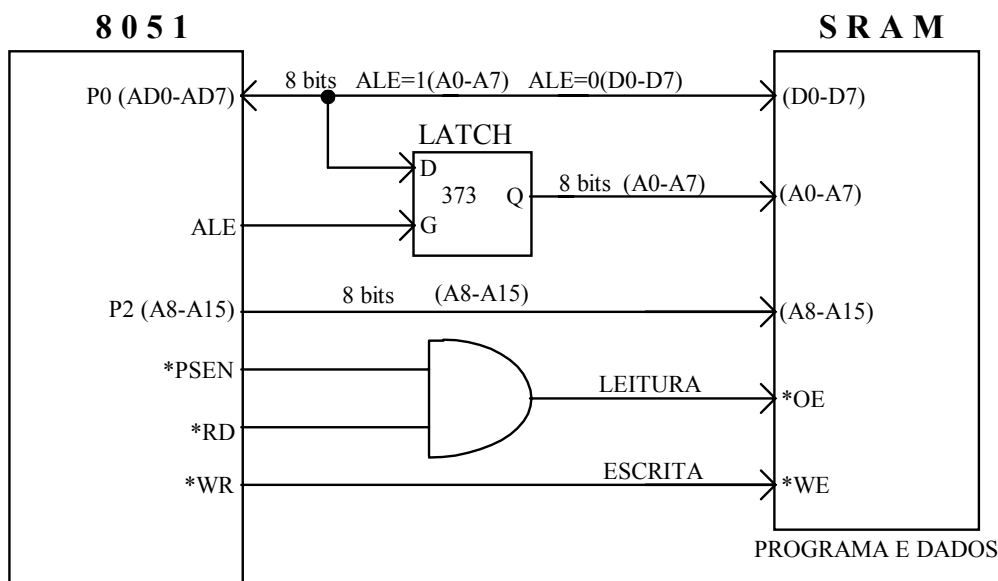


Figura 2.7. Microcontrolador 8051 com arquitetura de Von Newman: uma só Memória de Programa e Dados.

2.5. MEMÓRIA INTERNA (RAM INTERNA)

Uma das grandes vantagens do 8051 é oferecer uma memória de dados interna, com um mínimo de 128 bytes. Essa memória permite um rápido acesso aos dados e, em muitas aplicações, pode eliminar a necessidade da RAM externa, diminuindo portanto o custo do circuito controlador. Além da velocidade, existem áreas de RAM interna que são acessíveis bit a bit, o que é muito útil para operações booleanas.

O espaço de endereçamento reservado para acessar a RAM interna é de 8 bits, o que proporciona um máximo de 256 bytes. Mas com um pequeno artifício é possível colocar mais 128 bytes, resultando num total de 384 bytes na RAM interna (assim é o 8052). Além disso, na RAM interna existem 4 bancos de 8 registros (R0, R1, ... , R7) que podem ser utilizados pelo usuário.

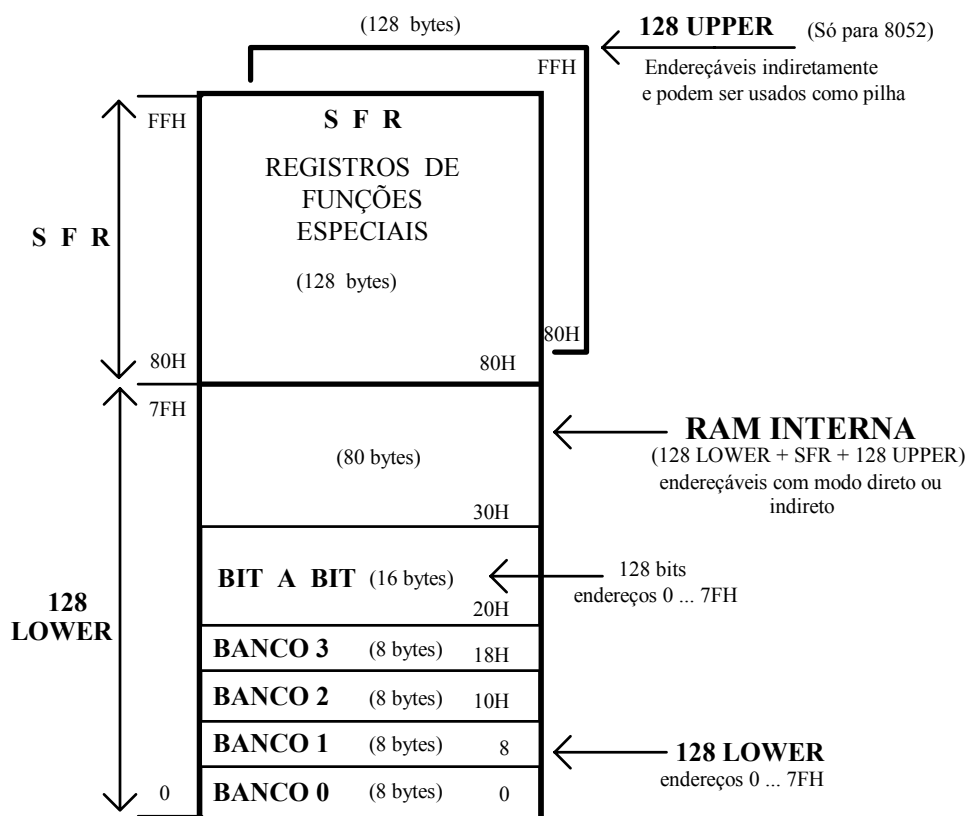


Figura 2.8. RAM interna do MCS-51.

A pilha sempre funciona na RAM interna e a consome muito; por isso nos 8052 o recurso de colocar a pilha nos 128 bytes endereçados indiretamente (128 UPPER) é muito atraente.

Exemplo: seja o caso onde não há RAM externa e se queira utilizar os 4 bancos, os 16 bytes endereçáveis bit a bit e mais 10 variáveis de 1 byte. Isso deixa 70 bytes para a pilha. Se forem usadas interrupções, para cada uma são guardados: PC (2 bytes), PSW, Acc e B ==> 5 bytes. Podem acontecer até 2 interrupções simultâneas, o que consome até 10 bytes. Logo, há 60 bytes

disponíveis para a pilha. Se para cada CALL são guardados PC, PSW, Acc e B (5 bytes), há disponibilidade de até 12 CALL aninhados.

Se a pilha for trabalhar nos 128 UPPER, subtraindo os 10 bytes para as interrupções têm-se 118 bytes, o que resulta em 23 CALL aninhados ($23 \times 5 = 115$ bytes) e ainda sobram 80 bytes para as variáveis nos 128 LOWER.

Os bancos de registros são ótimos para guardar o contexto, principalmente no caso de interrupções. Estes são trocados com uma só instrução. Por exemplo, poder-se-ia usar:

```

BK3    →    interrupção porta serial
BK2    →    interrupção INT1
BK1    →    interrupção INTO
BK0    →    trabalho

```

Só pode haver um banco selecionado de cada vez mas todos os registros podem ser endereçados através do endereço de seu byte.

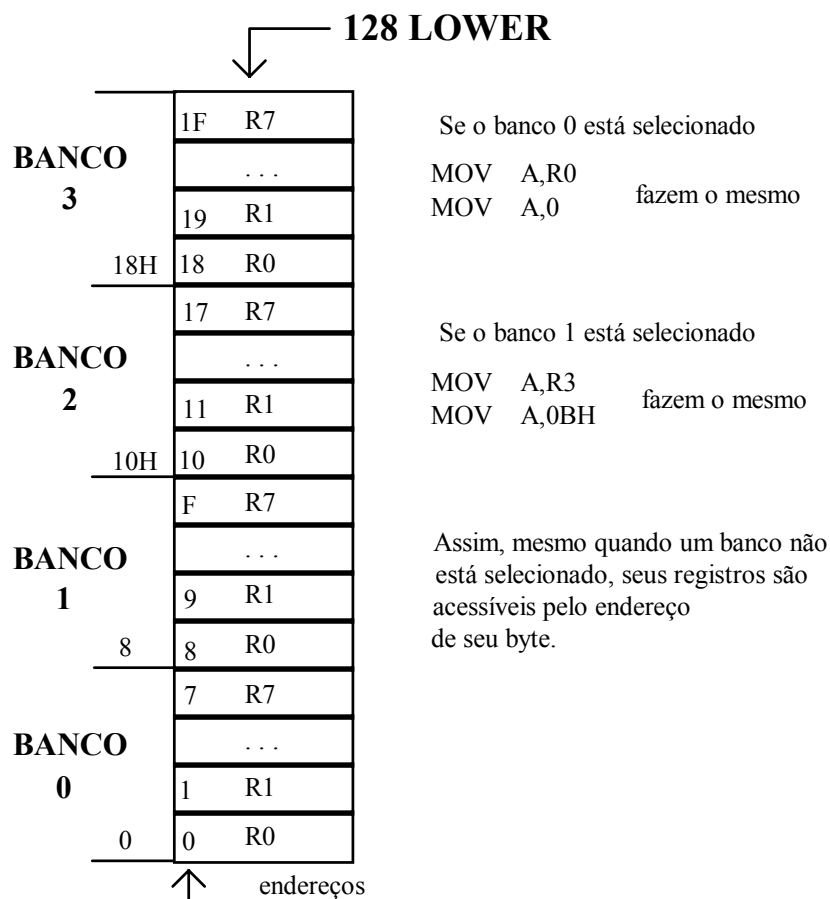


Figura 2.9. Endereço dos registros.

2.6. SFR - SPECIAL FUNCTION REGISTERS

Dos 256 bytes da RAM interna que são endereçados diretamente, 128 (80 FF) são reservados para os Registros de Funções Especiais (SFR). O 8051 apresenta um conjunto mínimo mas pode haver mais SFR em outros membros da MCS-51.

Os SFR não devem ser confundidos com o chamado "128 UPPER", que é uma área de 128 bytes (80H FFH) que só é endereçada indiretamente e só existe no 8051.

A seguir é apresentada uma lista com os SFRs do 8051 e seus endereços.

BIT	SÍMBOLO	NOME	ENDEREÇO
*	Acc	Acumulador	E0
*	B	Registro B	F0
*	PSW	Palavra de Status (Program Status Word)	D0
	SP	Ponteiro da Pilha (Stack Pointer)	81
	DPH	Ponteiro de Dados (high) (Data Pointer High)	83
	DPL	Ponteiro de Dados (low) (Data Pointer Low)	82
*	P3	Porta 3	B0
*	P2	Porta 2	A0
*	P1	Porta 1	90
*	P0	Porta 0	80
*	IP	Prioridade de Interrupção (Interrupt Priority)	B8
*	IE	Habilitação de Interrupção (Interrupt Enable)	A8
	TMOD	Timer/Counter Mode	89
*	TCON	Timer/Counter Control	88
	TH1	Timer/Counter 1 (MSB)	8D
	TL1	Timer/Counter 1 (LSB)	8B
	TH0	Timer/Counter 0 (MSB)	8C
	TL0	Timer/Counter 0 (LSB)	8A
*	SCON	Controle da Porta Serial (Serial Control)	98
	SBUF	Buffer da Porta Serial (Serial Buffer)	99
*	PCON	Controle de Energia (Power Control)	87

Figura 2.10. Lista dos registros SFR.

A seguir é apresentada uma breve descrição de cada registro:

Acc → Acumulador

B → Usado durante as operações de multiplicação e divisão; nos demais casos pode ser usado como auxiliar.

PSW → Program Status Word ou Registro de Flags de Estado

7	6	5	4	3	2	1	0
CY	Ac	F0	RS1	RS0	OV	-	P

PSW.7 → **CY** → Carry Flag

PSW.6 → **Ac** → Carry auxiliar (operação com BCD – Binary Coded Decimal)

PSW.5 → **F0** → Flag 0, uso geral

PSW.4 → **RS1** → Seleccionador de Banco (Range Selector), bit 1

PSW.3 → **RS0** → Seleccionador de Banco (Range Selector), bit 0

RS1	RS0	BANCO SELEC.
0	0	Banco 0
0	1	Banco 1
1	0	Banco 2
1	1	Banco 3

PSW.2 → **OV** → Overflow

PSW.1 → **-** → Flag definível pelo usuário

PSW.0 → **P** → Paridade (ímpar)

P=1 → quantidade ímpar de 1s

P=0 → quantidade par de 1s

SP → Ponteiro da Pilha (Stack Pointer), incrementado antes de PUSH e CALL

DPTR → Ponteiro para memória de dados externa (Data Pointer).

É um registro de 16 bits formado por **DPH** e **DPL**

P0, P1, P2, P3 → Latches das portas paralelas

IP → Prioridade das interrupções (Interrupt Priority)

IE → Habilitação das interrupções (Interrupt Enable)

TMOD → Modo de operação dos Timers/Counters (Timer/Counter Mode)

TCON → Controle dos Timers/Counters (Timer/Counter Control)

SCON → Controle da porta serial (Serial Control)

SBUF → Dois registros, um para leitura e outro para escrita (Serial Buffer)

Leitura → receber o dado da porta serial

Escrita → enviar o dado para a porta serial

PCON → Controle de energia (Power Control); coloca no modo "Power Down" ou "Idle"

2.7. MAPA DA RAM INTERNA

Aqui é apresentado um mapa completo da RAM Interna da MCS-51. Os endereços dos bytes vão de 0H até FFH. Existem dois grupos de bytes:

- 128 LOWER → de 00H até 7FH.
- SFR → de 80H até FFH.

Observações:

- 1) Os registros ou endereços em **negrito** são endereçáveis bit a bit.
- 2) Usam-se as letras "BK" para abreviar a palavra "banco".
- 3) As células marcadas com "-" são inexistentes.
- 4) A seqüência de endereços é da direita para a esquerda e de baixo para cima.

S F R										
	FFH	-	-	-	-	-	-	-	-	F8H
	F7H	-	-	-	-	-	-	-	B	F0H
	EFH	-	-	-	-	-	-	-	-	E8H
	E7H	-	-	-	-	-	-	-	Acc	E0H
	DFH	-	-	-	-	-	-	-	-	D8H
	D7H	-	-	-	-	-	-	-	PSW	D0H
	CFH	-	-	-	-	-	-	-	-	C8H
	C7H	-	-	-	-	-	-	-	-	C0H
	BFH	-	-	-	-	-	-	-	IP	B8H
	B7H	-	-	-	-	-	-	-	P3	B0H
	AFH	-	-	-	-	-	-	-	IE	A8H
	A7H	-	-	-	-	-	-	-	P2	A0H
	9FH	-	-	-	-	-	-	SBUF	SCON	98H
	97H	-	-	-	-	-	-	-	P1	90H
	8FH	-	-	TH1	TH0	TL1	TL0	TMOD	TCON	88H
	87H	PCON	-	-	-	DPH	DPL	SP	P0	80H
128 LOWER										
	7FH	7FH	7EH	...						78H
	77H									70H
	6FH									68H
	67H									60H
	5FH									58H
	57H									50H
	4FH									48H
	47H									40H
	3FH									38H
	37H						...	31H	30H	30H
	2FH	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H	28H
	27H	27H	26H	25H	24H	23H	22H	21H	20H	20H
BK3	1FH	R7	R6	R5	R4	R3	R2	R1	R0	18H
BK2	17H	R7	R6	R5	R4	R3	R2	R1	R0	10H
BK1	0FH	R7	R6	R5	R4	R3	R2	R1	R0	08H
BK0	07H	R7	R6	R5	R4	R3	R2	R1	R0	00H

Figura 2.11. Mapa completo da RAM Interna.

2.8. MAPA DOS BITS

A seguir, na figura 2.12 é apresentado um mapa com todos os bits do MCS-51. São 256 bits e, portanto, ocupam 32 bytes. Estão marcados o endereço dos bits, o nome e o byte ao qual pertencem. Existem duas classes de bits: os que pertencem ao 128 LOWER e têm os endereços de 00H a 7FH e os que pertencem aos SFR e têm os endereços de 80H até FFH. Alguns bits não existem ou serão utilizados em futuras versões e estão marcados com "-".

Como será abordado mais adiante, os bits podem ser endereçados de até 4 formas diferentes. Como exemplo, uma instrução para habilitar a recepção serial (setar o bit REN), poderia ser:

- 1) SETB REN
- 2) SETB SCON.4
- 3) SETB 98H.4
- 4) SETB 9CH

MAPA DOS 256 BITS											
END BIT	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	END BIT	END BYTE	NOM BYTE
FFH	-	-	-	-	-	-	-	-	F8H	F8H	
F7H	B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0	F0H	F0H	B
EFH	-	-	-	-	-	-	-	-	E8H	E8H	
E7H	Acc.7	Acc.6	Acc.5	Acc.4	Acc.3	Acc.2	Acc.1	Acc.0	E0H	E0H	Acc
DFH	-	-	-	-	-	-	-	-	D8H	D8H	-
D7H	CY	AC	F0	RS1	RS0	OV		P	D0H	D0H	PSW
CFH	-	-	-	-	-	-	-	-	C8H	C8H	-
C7H	-	-	-	-	-	-	-	-	C0H	C0H	-
BFH	-	-	-	PS	PT1	PX1	PT0	PX0	B8H	B8H	IP
B7H	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	B0H	B0H	P3
AFH	EA	-	-	ES	ET1	EX1	ET0	EX0	A8H	A8H	IE
A7H	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	A0H	A0H	P2
9FH	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	98H	98H	SCON
97H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	90H	90H	P1
8FH	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	88H	88H	TCON
87H	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	80H	80H	P0
7FH	7FH	7EH	7DH	7CH	...				78H	2FH	
77H									70H	2EH	
6FH									68H	2DH	
67H									60H	2CH	
5FH									58H	2BH	
57H									50H	2AH	
4FH									48H	29H	
47H									40H	28H	
3FH									38H	27H	
37H									30H	26H	
2FH									28H	25H	
27H									20H	24H	
1FH									18H	23H	
17H									10H	22H	
0FH				...	0BH	0AH	09H	08H	08H	21H	
07H	07H	06H	05H	04H	03H	02H	01H	00H	00H	20H	

Figura 2.12. Mapa completo dos bits do 8051.

CAPÍTULO III

PINAGEM E TEMPORIZAÇÃO

3.1. INTRODUÇÃO

O encapsulamento dos microcontroladores varia muito, de acordo com as funções desempenhadas. Basicamente são usados 3 tipos de encapsulamento :

- DIP → Dual in Pack,
- QFP → Quad Flat Pack,
- PLCC → Leadless Chip Carrier.

Os 8051 e 8052 estão disponíveis em 40 DIP, 44 PLCC e 44 QFP. Alguns exemplos:

- 8031 (mais simples) → DIP40
- 8051GB (AD + timer, etc) → 100 QFP
- 8051SL (Keyboard Controller) → 100 QFP
- 83152 (Universal Com. Contr.) → DIP48 E PLCC68

3.2. DESCRIÇÃO DA PINAGEM

Estudar-se-á a pinagem do 8051 no encapsulamento DIP40 (HMOS) → 8051 AH

(40) **Vcc** → Alimentação de +5V. Consumo: $I_{cc} = 125 \text{ mA}$, com todas as saídas desconectadas.

(20) **Vss** → Terra.

(32-39) **P0** → Porta 0 (AD0...AD7). Além de porta paralela, está multiplexada com o byte menos significativo (LSB) dos endereços e dos dados. Admite 8 cargas LS TTL.

(21-28) **P2** → Porta 2 (A8...A15). Além de porta paralela, está multiplexada com o byte mais significativo (MSB) dos endereços. Admite 4 cargas LS TTL.

(1-8) **P1** → Porta 1. Admite 4 cargas LS TTL.

(10-17) **P3** → Porta 3. Compartilhada com uma série de recursos Admite 4 cargas LS TTL.

P3.0 → **RXD**, entrada serial

P3.1 → **TXD**, saída serial

P3.2 → ***INT0**, interrupção externa 0

P3.3 → ***INT1**, interrupção externa 1

P3.4 → **T0**, entrada para o timer 0 (contador neste caso)

P3.5 → **T1**, entrada para o timer 1 (contador neste caso)

P3.6 → ***WR**, escrita na memória de dados externa

P3.7 → *RD, leitura na memória de dados externa

- (9) **RST** → Reset. Com o oscilador funcionando, deve ser mantido um nível alto durante 24 períodos.
- (30) **ALE/PROG** → Address Latch Enable. Pulso para acionar o latch que captura o LSB do endereço (com sua borda ascendente). Ele é emitido à razão de 1/6 da frequência do oscilador e pode ser usado para acionar entradas externas. Um ALE é omitido durante o acesso à Memória de Dados Externa. Também é usada na gravação da ROM interna.
- (29) ***PSEN** → Program Store Enable. Pulso de leitura para a Memória de Programa Externa. Quando o programa está sendo executado na memória de programa externa ele aparece como 1/6 da frequência de clock. Quando há acesso à memória de dados externa, 2 PSEN são perdidos.
- (31) ***EA/VPP** → External Access Enable. Informa à CPU se o programa está na Memória de Programa Externa ou na ROM Interna. Também usado para gravação da ROM Interna.
- *EA = 1 (Vcc) → (0000H - 0FFFH) ROM Interna
 → (1000H - FFFFH) Memória de Programa Externa
- *EA = 0 (Vss) → (0000H - FFFFH) Memória de Programa Externa
- (19) **XTAL1** → Entrada do amplificador inversor do oscilador interno. Deve ser conectado à terra se for usado um clock externo (HMOS) ou ao clock externo (CHMOS).
- (20) **XTAL2** → Saída do amplificador inversor do oscilador interno. Se for usado clock externo, serve como entrada para o mesmo (HMOS) ou não é conectado (CHMOS).

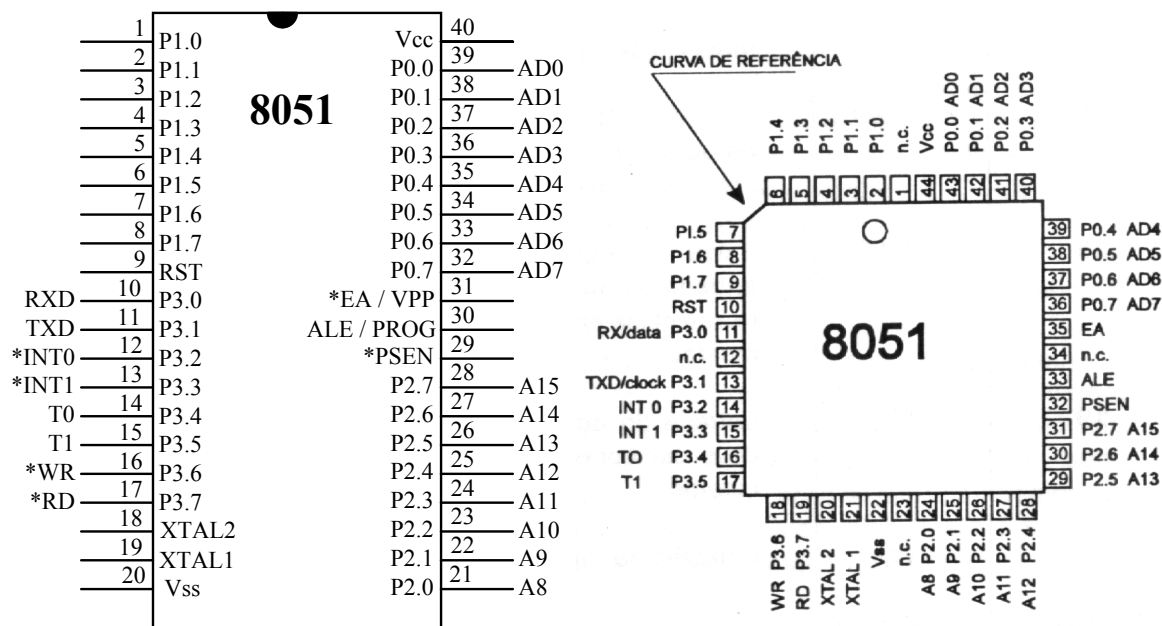
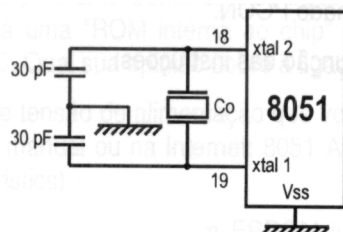


Figura 3.1. Pinagem do 8051.

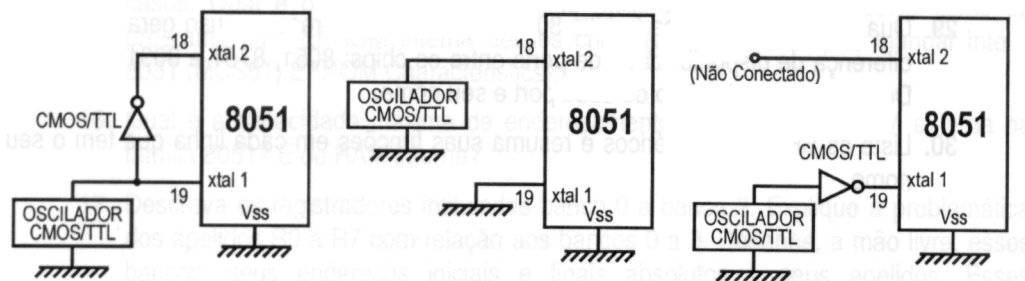
OBS: A figura a seguir ilustra as possíveis ligações dos pinos XTAL1 e XTAL2:

Opções de ligação:

a) Cristal



b) Oscilador Externo



Observação: Os capacitores podem ser variados de ± 10 pF.

3.3. DIAGRAMAS DE TEMPO

Aqui serão abordados alguns diagramas de tempo, os quais foram simplificados em benefício da compreensão. Eles **não** mostrarão os tempos de subida e descida dos pulsos. Os retardos de propagação entre o CLK e os demais sinais foram omitidos. Esses detalhes devem ser consultados no "datasheet" da CPU a ser usada.

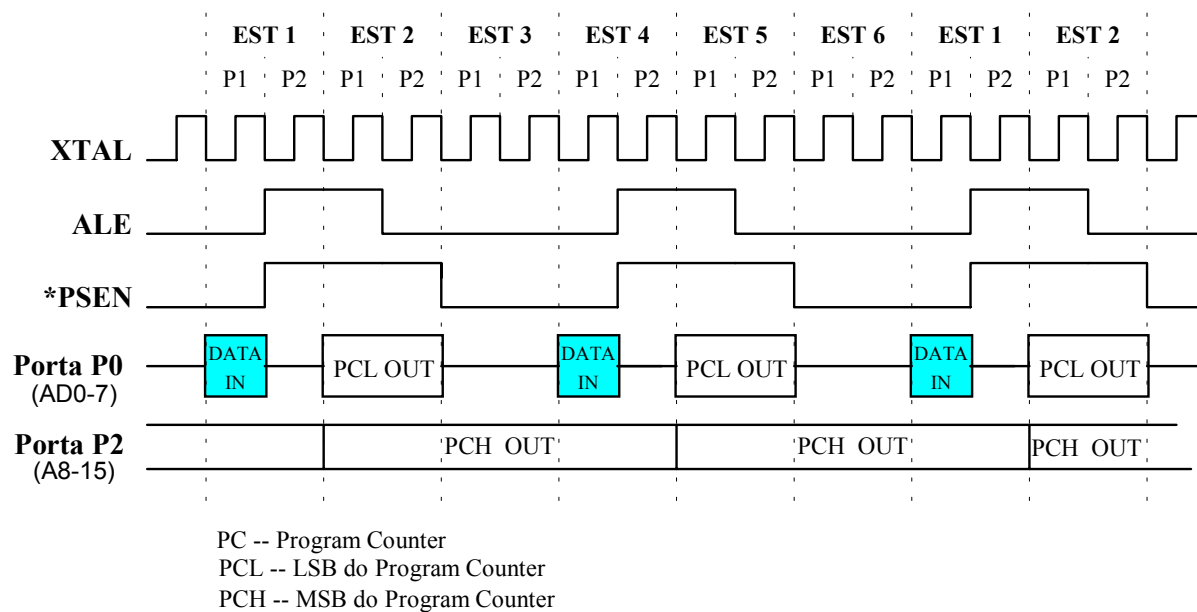


Figura 3.2. Fetch de programa, Memória Externa.

Cada período de Clock recebe o nome de Fase (Phase). Cada estado é composto por 2 períodos de Clock ou 2 fases, denominadas P1 e P2. Observar que EST1=EST4, EST2=EST5, etc. O Program Counter (PC) é composto por dois registros de 8 bits: $PC = PCH + PCL$.

As frequências de ALE e *PSEN são iguais a 1/6 da frequência do cristal, mas o "Duty Cycle (DC)" é diferente para cada uma. ALE tem o DC=33% (1/3) enquanto que *PSEN tem o DC=50% (1/2).

$$f(ALE) = f(*PSEN) = f(XTAL)/6$$

$$DC(ALE) = 33\% \Rightarrow 33\% \text{ alto e } 67\% \text{ baixo}$$

$$DC(*PSEN) = 50\% \Rightarrow 50\% \text{ alto e } 50\% \text{ baixo}$$

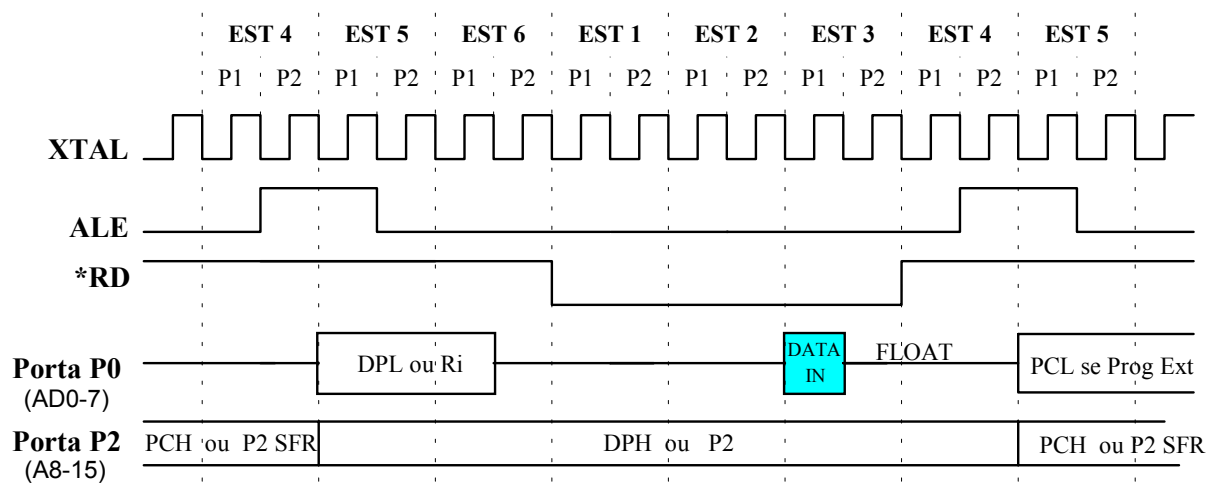


Figura 3.3. Leitura na Memória de Dados Externa.

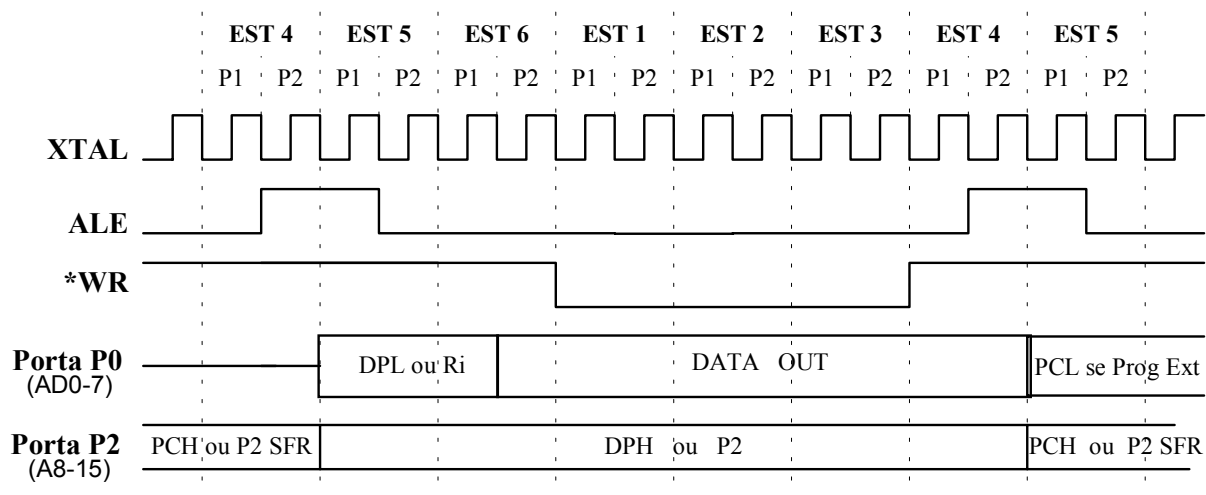


Figura 3.4. Escrita na Memória de Dados Externa.

A leitura e a escrita na Memória de Dados Externa usam o dobro de tempo quando são comparadas com um acesso à memória de programa. Notar que $f(\text{ALE})$ não tem $f(\text{XTAL})/6$ pela ausência de um pulso de ALE e que não existe *PSEN. Perde-se um pulso de ALE e dois pulsos de *PSEN. Então,

$f(\text{ALE}) = f(*\text{PSEN}) = f(\text{XTAL})/6$ mas, quando há acessos à Memória de Dados Externa, se perdem 1 pulso de ALE e 2 pulsos de *PSEN.

Nos acessos à Memória de Dados Externa é usado um endereço de 16 bits. Esse endereço pode ser formado por um registro ponteiro de dados de 16 bits, DPTR (DPTR = DPH + DPL) ou por R0 ou R1 e o conteúdo do SFR P2.

		MSB		LSB
DPTR	→	DPH	e	DPL
P2 e Ri	→	SFR P2	e	R0 ou R1

Exemplos:

	leitura	escrita
Com DPTR	→ MOVX A,@DPTR	ou MOVX @DPTR,A
Com P2 e Ri	→ MOVX A,@Ri	ou MOVX @Ri,A (onde i=0 ou i=1)

Notar que quando se usa DPTR para endereçar a memória externa, os 16 bits dos endereços são emitidos por P0 e P2 mas ao se usar Ri, os 16 bits são formados pelo conteúdo de Ri (LSB) e o que estava no SFR P2, ou seja, não se altera o conteúdo de P2.

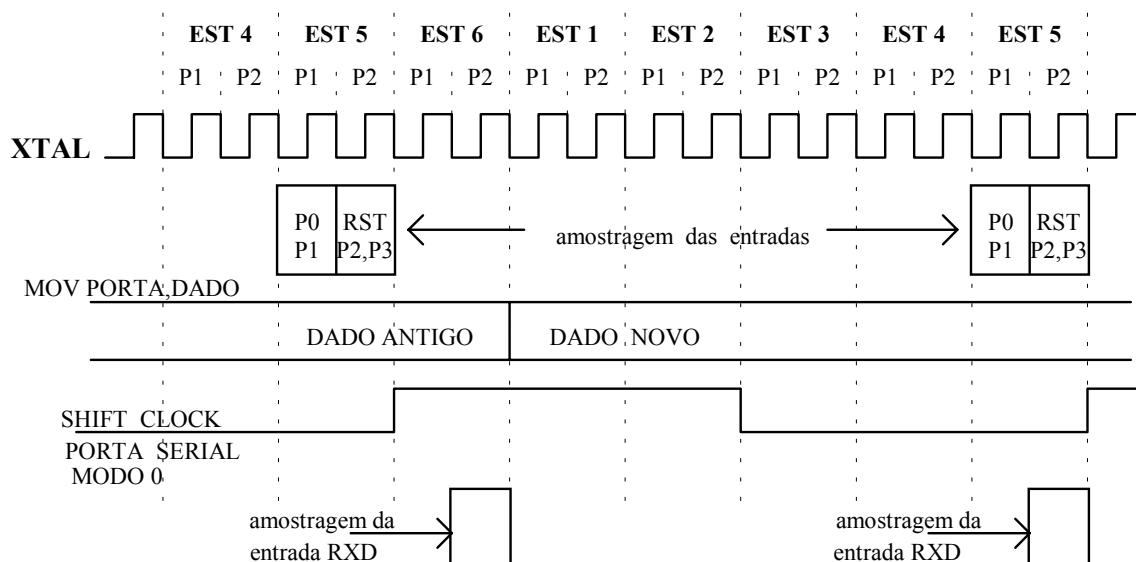


Figura 3.5. Operação das portas.

3.4. RESET

A entrada de Reset é o pino RST que possui um Schmitt Triger na entrada. O Reset é realizado quando este pino se mantém em nível alto pelo menos por 2 ciclos de máquina (24 clocks) enquanto o oscilador estiver funcionando. A CPU responde gerando um reset interno de acordo com a temporização que indicada na figura 3.6.

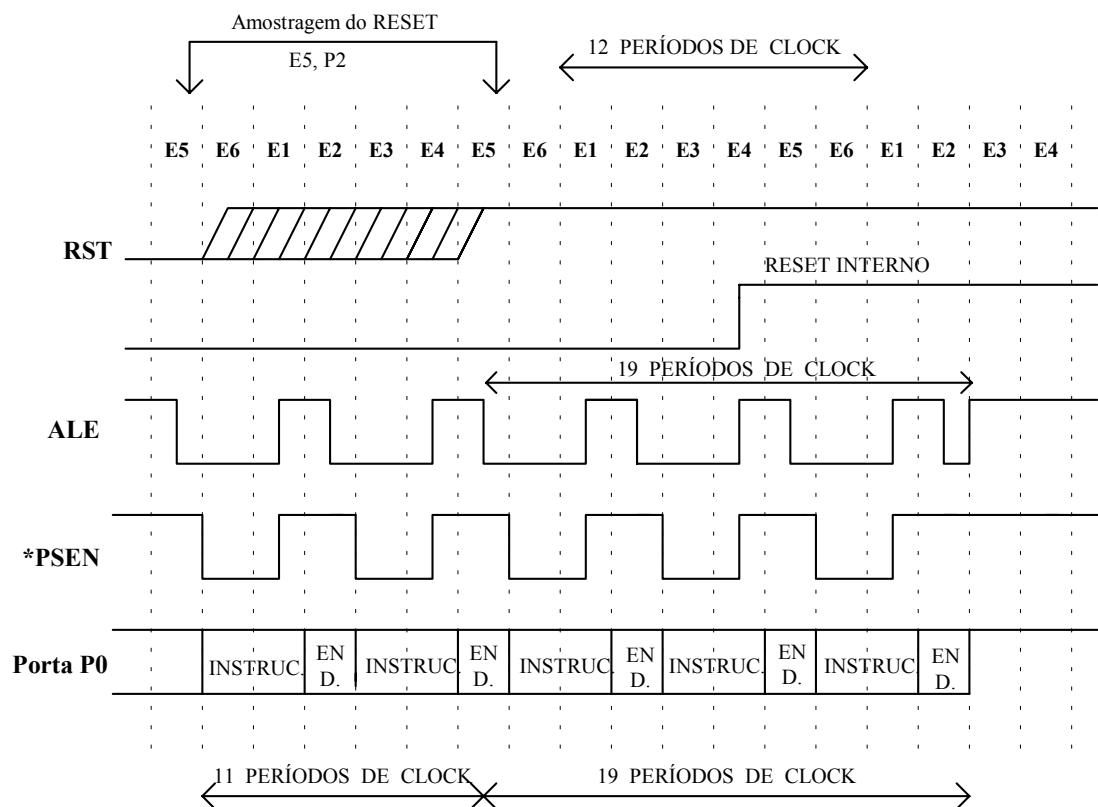


Figura 3.6. Temporização do RESET.

A entrada RST é assíncrona e é amostrada durante a fase 2 (P2) do estado 5 (E5). Depois do RST ser amostrado como alto, ainda se mantém atividade nos pinos durante 19 períodos de clock. Os pinos só cessarão a atividade 19 a 31 períodos de clock depois de RST ser ativada.

Após o RST, os pinos ALE e PSEN vão para 1. Depois de RST=0 (desativada), são necessários 1 a 2 períodos de clock para que ALE e PSEN comecem. Por isso, outros dispositivos externos não podem ser sincronizados com o tempo interno do Reset do 8051.

A RAM Interna não é afetada pelo reset. Depois do RESET, a CPU é inicializada com os seguintes dados:

SP	= 7	ESTADO DOS REGISTROS
SBUF	= ?	APÓS O RESET
P0,P1,P2,P3	= FFH	
OUTROS SFR	= 0	

Uma maneira fácil de realizar o "Power on Reset" e o "Reset Manual" é mostrada na figura 3.7.

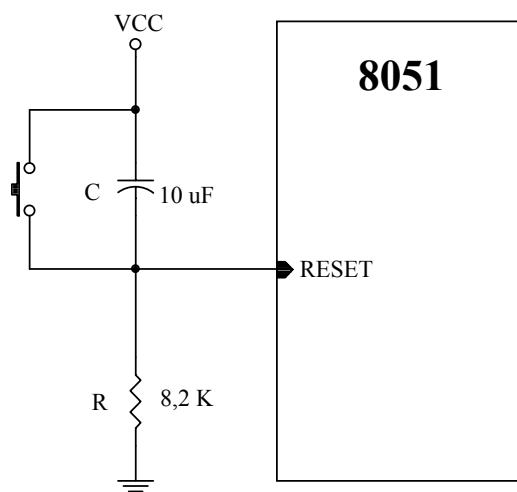


Figura 3.7. Circuito para RESET do 8051 (HMOS).

O esquema da figura 3.7 é válido para HMOS. Nas famílias CHMOS o resistor 8,2 K Ω pode ser omitido pois há um "pull-down" interno; além disso, o capacitor deve ser reduzido a 1 μ F.

Quando é energizado, o circuito (RC) assegura no pino RST um nível alto durante uma quantidade de tempo que depende dos valores de R e C. Para ter um "Power On Reset" seguro é necessário que o pino RESET seja mantido em alto durante o tempo necessário para que o oscilador inicie e a isto se adicionam os 24 períodos de clock para o RESET efetivo.

Tempos que serão computados no power up:

- Vcc se estabiliza \rightarrow 10ms
- Partida do oscilador \rightarrow 1ms (10MHz)
10ms (1MHz)

Com este circuito de Reset Vcc cai rapidamente a 0. Com isto, a tensão do pino RST pode cair momentaneamente abaixo de 0 mas não causa danos. Com a chave em paralelo com um capacitor pode-se gerar o RESET manual. Para que se possa projetar um RESET eficiente e preciso é necessário analisar o circuito RC de primeira ordem.

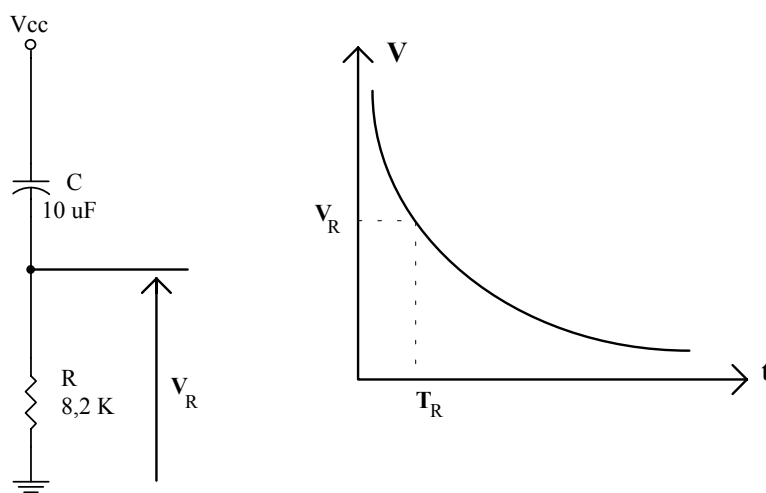


Figura 3.8. Análise do circuito para RESET do 8051.

Da análise do circuito se pode escrever a equação:

$$V_R = V_{CC} e^{-t/RC}$$

A pergunta, quando se calcula um circuito de RESET, é: Qual RC me garante que o RESET fique ativado por um tempo maior que o especificado ? Para que se possa responder a esta pergunta é necessário saber qual a tensão de comutação do Schmitt-Trigger do 8051 e qual é o modelo elétrico da entrada RESET (ou seja, quanta corrente consome). Essas informações não estão facilmente disponíveis nem nos Data Sheets.

Exemplo: com o 8051 usando cristal de 1 MHz:

10 ms → estabilizar Vcc

10 ms → partida do oscilador

24 μs → necessários para o Reset (24 períodos de clock)

20024 μs → Tempo do Reset = 20,024 ms =(aprox.) 20 ms

Supondo que o Schmitt Trigger comute a 3,5 V (na transição de alto para baixo; valores abaixo de 3,5 V na entrada geram 0 na saída) e usando os valores de R e C da figura 3.7, calcula-se.

$$\ln(3,5) - \ln(5) = -(t/RC)$$

$$t = RC (\ln(3,5) - \ln(5)) = 29 \text{ ms} > 20 \text{ ms} \quad \text{OK !}$$

Pelo que parece, o circuito de RESET sugerido deve funcionar.

Apresenta-se um circuito alternativo que, apesar do maior gasto de componentes, permite o conhecimento da tensão do Schmitt Trigger, saber seu consumo e calcular precisamente o circuito de RESET. Neste circuito o diodo é usado para garantir uma rápida descarga do capacitor quando se desliga a fonte de alimentação.

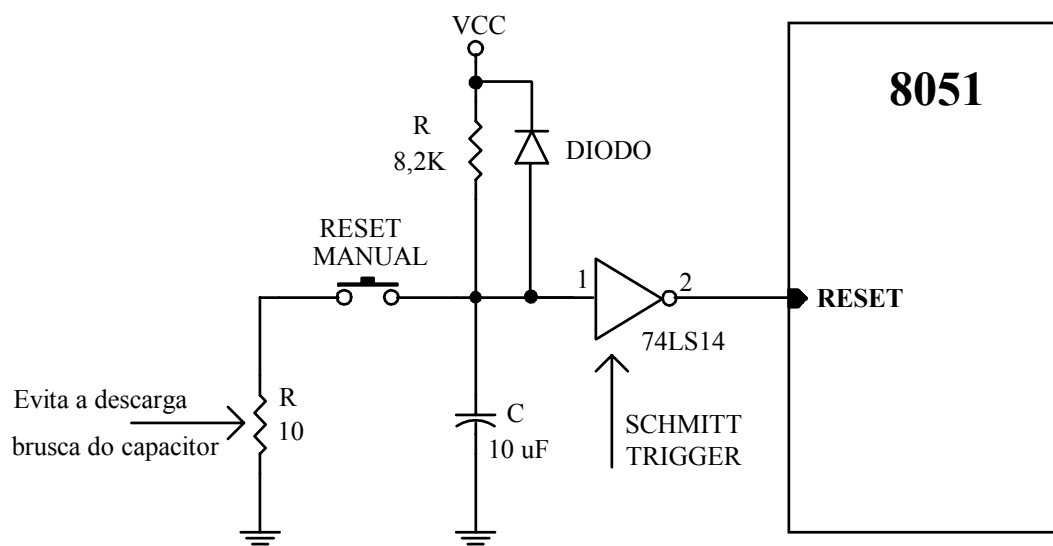


Figura 3.9. Circuito de RESET com Schmitt-Trigger TTL.

CAPÍTULO 4

CONJUNTO DE INSTRUÇÕES

4.1. INTRODUÇÃO

Todos os membros da família MCS-51 executam o mesmo conjunto de instruções. As instruções são otimizadas para aplicações de controle de 8 bits. Elas permitem um rápido endereçamento da RAM interna, facilitando operações byte a byte em estruturas de dados pequenas. O conjunto de instruções também oferece um grande suporte para manipulações e operações de variáveis de um bit, facilitando os sistemas lógicos que necessitam de operações booleanas.

A seguir serão mostradas todas as instruções da família MCS-51. Para uma descrição mais detalhada dessas instruções deve-se consultar o manual do MCS-51. (Aqui se supõe que o leitor tenha uma pequena experiência em programação assembly).

Para ajudar os programadores, os programas montadores para o MCS-51 têm os endereços dos SFR e dos bits previamente declarados. É como se em todo programa já existisse as pseudo-instruções:

Acc	EQU	0E0H
B	EQU	0F0H
PSW	EQU	0D0H
etc.		

Para a exposição das instruções, utilizar-se-ão as seguintes abreviaturas:

Rn	→	qualquer registro R0, R1, ..., R7
direto, dir	→	endereço da RAM interna (8 bits)
@Ri	→	@R0 ou @R1, usado para endereçamento indireto
#data, #dt	→	constante de 8 bits (byte)
#data16, #dt16	→	constante de 16 bits
adr16	→	endereço de 16 bits (endereço 64 KB)
adr11	→	endereço de 11 bits (endereço 2 KB)
rel	→	deslocamento relativo (complemento a 2: -128 a +127)
bit	→	endereço de um bit da RAM interna
A	→	acumulador (registro)
Acc	→	endereço do acumulador.

4.2. MODOS DE ENDEREÇAMENTO

De acordo com o "Data Sheet" do MCS-51, existem 6 modos de endereçamento com a seguinte nomenclatura :

- Imediato
- Direto
- Indireto
- Registrador
- Registrador Específico
- Indexado

Imediato → O valor da constante é colocado no opcode.

MOV A, #100

Carrega 100 no acumulador (Acc=100). O byte 100 é um dado imediato. Deve-se notar a presença do sinal # que indica operação imediata.

Direto → O operando especifica um endereço de 8 bits da RAM interna.

MOV A, 20

Transfere para o acumulador o conteúdo do endereço 20 da RAM Interna. Todo endereçamento direto usa a RAM Interna.

Indireto → Aqui se especifica um registro onde está o endereço do operando. Só pode ser usado para endereçamento indireto: R0, R1 ou DPTR.

MOV A, @R0

Coloca no acumulador o conteúdo do endereço que está em R0

Registro → No código de operação da instrução existe um campo de 3 bits (pois são 8 registradores, de R0 a R7) onde é especificado o registro a ser utilizado. Essa forma é eficiente e evita utilizar um byte adicional para indicar o registro.

MOV A,R0

Coloca no acumulador o conteúdo de R0. É uma instrução de um só byte.

Registro Específico → Algumas operações são específicas para certos registros. Por exemplo, algumas instruções sempre operam com Acc ou DPTR e não necessitam de espaço no opcode para especificar isto.

MOVX A,@DPTR

Esta é uma instrução para leitura da Memória de Dados Externa. Coloca no acumulador o conteúdo do endereço da RAM Externa que está no DPTR. Como sempre são usados

Acc e DPTR, não é necessário especificá-los, o que faz com que a instrução empregue apenas 1 byte.

Indexado → O endereço do operando é formado pela soma de um endereço base com um registro de indexação. Somente a Memória de Programa pode ser endereçada deste modo.

MOVC A,@A+DPTR

A → índice,

DPTR → endereço base.

A soma do DPTR com o acumulador forma um endereço da Memória de Programa e o conteúdo deste endereço é transferido para o acumulador. Essa instrução é ótima para "look up table".

Observação: também existe um desvio indexado

4.3. SOBRE AS INSTRUÇÕES

Agora serão mostradas as instruções do MCS-51. Estas não serão muito detalhadas. Em cada instrução são apresentados o número de bytes do opcode e o número de ciclos de máquina (MC=12 clocks) necessários para sua execução.

Existem 111 instruções na família MCS-51. Elas estão distribuídas da seguinte forma:

- Aritméticas 24 → 22%
- Lógicas 25 → 23%
- Transferência de dados 28 → 25%
- Booleanas 17 → 15%
- Desvios 17 → 15%

4.4. INSTRUÇÕES ARITMÉTICAS

4.4.1. Soma de 8 Bits:

O acumulador é um dos operandos e também armazena o resultado. Existem 4 instruções de soma:

		bytes	MC	CY	AC	OV
ADD A,	Rn	1	1	X	X	X
	direto	2	1			
	@Ri	1	1			
	#data	2	1			

4.4.1. Soma de 8 Bits com Carry:

O acumulador é um dos operandos e também armazena o resultado. São 4 instruções de soma com carry:

		bytes	MC	CY	AC	OV
ADDC A,	Rn	1	1	X	X	X
	direto	2	1			
	@Ri	1	1			
	#data	2	1			

4.4.3. Subtração de 8 Bits com Borrow:

O acumulador é um dos operandos e também recebe o resultado. Deve-se lembrar que a subtração sempre usa o borrow. São 4 instruções de subtração com borrow:

		bytes	MC	CY	AC	OV
SUBB A,	Rn	1	1	X	X	X
	direto	2	1			
	@Ri	1	1			
	#data	2	1			

4.4.4. Incremento de 8 Bits:

Existem 4 instruções para incremento de 8 bits :

		bytes	MC	CY	AC	OV
INC	A	1	1	-	-	-
	Rn	1	1			
	direto	2	1			
	@Ri	1	1			

4.4.5. Decremento de 8 Bits:

Há 4 instruções de decremento de 8 bits e muito se assemelham às instruções de incremento:

		bytes	MC	CY	AC	OV
DEC	A	1	1	-	-	-
	Rn	1	1			
	direto	2	1			
	@Ri	1	1			

4.4.6. Incremento de 16 Bits:

Existe apenas um incremento de 16 bits (não há decrementos de 16 bits) :

		bytes	MC	CY	AC	OV
INC	DPTR	1	2	-	-	-

4.4.7. Multiplicação e Divisão de 8 Bits:

Na multiplicação e na divisão sempre são usados os registros A e B. Estas são as instruções que necessitam mais tempo de execução.

Multiplicação: $A * B \rightarrow$ Resultado: A=LSB B=MSB

Divisão: $A / B \rightarrow$ Resultado: A=quociente B=resto

		bytes	MC	CY	AC	OV
MUL	AB	1	4	0	-	X
DIV	AB			0	X	0

4.4.8. Ajuste Decimal:

Esta instrução existe para permitir operações com representação BCD. O ajuste é valido apenas para as somas (ADD e ADDC). Existe uma única instrução:

		bytes	MC	CY	AC	OV
DA	A	1	1	X	X	-

OBS 1: DA A não pode simplesmente converter um número hexadecimal no Acumulador para BCD nem se aplica à subtração.

Exemplo: A = 56H (0101 0110B) representando 56 decimal
 R3 = 67H (0110 0111B) representando 67 decimal
 Carry = 1

ADDC A, R3 → A = 0BEH e C (carry) e AC (auxiliar carry) iguais a 0

DA A → A = 24H (= 0010 0100B) e Carry=1

Algoritmo de ajuste da soma pela instrução DAA:

- 1- Se os bits 3-0 de A forem maiores que 9 ou se AC=1 → 6 é somado a A para gerar o dígito BCD menos significativo. Esta soma interna pode ligar o carry se o carry da nibble menos significativa propagar por toda a nibble mais significativa mas não apagará o carry.
- 2- Se o carry está agora em 1 ou se os bits 7-4 excederem 9, esta nibble é incrementada de 6 para produzir o dígito BCD mais significativo. O carry indicará se a soma é maior do que 99. O OV (overflow) não é afetado.

Na realidade, soma-se 00H, 06H, 60H ou 66H conforme o valor inicial de A e PSW.

OBS 2: Números BCD podem ser incrementados somando 01H ou decrementados somando 99H e usando o ajuste decimal DA A.

Exemplo: Se A = 30H (representando 30 decimal), as instruções

ADD A,#99H

DA A

resultarão em A = 29H (representando 29 decimal) e C (Carry) = 1 pois $30 + 29 = 129$

4.5 INSTRUÇÕES LÓGICAS

As instruções lógicas são as que realizam as operações de AND, OR e XOR. Deve-se notar que essas instruções são muito semelhantes. O resultado da operação não obrigatoriamente deve ser colocado no acumulador; também se pode indicar um endereço para receber o resultado.

4.5.1. AND de 8 Bits:

Usa-se o símbolo ANL (Logical AND) para representar esta instrução. Existem 6 instruções reunidas em dois grupos de acordo com o destino do resultado:

		bytes	MC	CY	AC	OV
ANL A,	Rn	1	1	-	-	-
	direto	2	1			
	@Ri	1	1			
	#data	2	1			
ANL direto	A	2	1			
	#data	3	2			

4.5.2. OR de 8 Bits:

Emprega-se o símbolo ORL (Logical OR) para representar esta instrução. Existem 6 instruções, reunidas em dois grupos de acordo com o destino do resultado:

		bytes	MC	CY	AC	OV
ORL A,	Rn	1	1	-	-	-
	direto	2	1			
	@Ri	1	1			
	#data	2	1			
ORL direto	A	2	1			
	#data	3	2			

4.5.3. XOR de 8 Bits:

Emprega-se o símbolo XRL (Logical Exclusive OR) para representar esta instrução. Existem 6 instruções, reunidas em dois grupos de acordo com o destino do resultado:

		bytes	MC	CY	AC	OV
XRL A,	Rn	1	1	-	-	-
	direto	2	1			
	@Ri	1	1			
	#data	2	1			
XRL direto	A	2	1			
	#data	3	2			

4.5.4. Operações Lógicas com o Acumulador:

Pode-se inicializar, complementar ou rodar o acumulador. As rotações podem se realizar para a direita ou para a esquerda utilizando ou não o bit de carry. Também se pode trocar de posição as nibbles do acumulador (uma nibble é formada por 4 bits, ou seja, um byte possui 2 nibbles). Todas estas instruções são classificadas como de registro específico pois sempre operam com o acumulador (não utilizam bytes de opcode para especificar isto). As instruções são:

- **CLR A** → inicializa com zero o acumulador
- **CPL A** → complementa de 1 o acumulador (inverte os bits)
- **RL A** → roda o acumulador à esquerda
- **RLC A** → roda o acumulador à esquerda com carry
- **RR A** → roda o acumulador à direita
- **RRC A** → roda o acumulador à direita com carry
- **SWAP A** → intercambiar as nibbles do acumulador

		bytes	MC	CY	AC	OV
CLR	A	1	1	-	-	-
CPL				-	-	-
RL				-	-	-
RLC				X	-	-
RR				-	-	-
RRC				X	-	-
SWAP				-	-	-

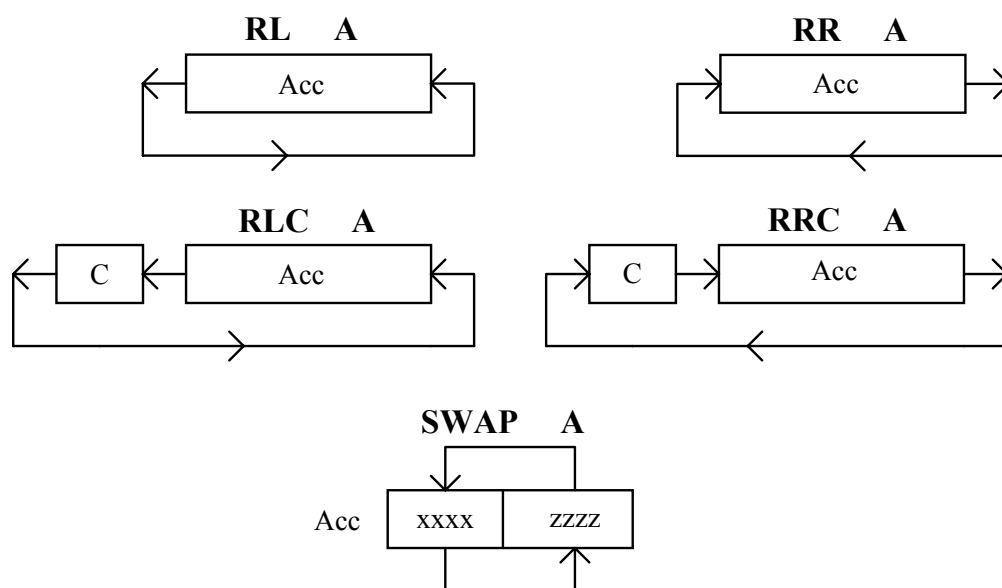


Figura 4.1. Gráfico das rotações e do swap.

4.6. INSTRUÇÕES DE TRANSFERÊNCIA DE DADOS

Existem 3 tipos de instruções de transferência de dados:

- as que trabalham com a RAM Interna (22),
- as que trabalham com a Memória de Dados Externa (4) e
- as que trabalham com a Memória de Programa (2).

Em geral, as transferências de dados operam na RAM Interna, a menos que se indique o contrário.

4.6.1. Transferência de Dados:

Todas as operações se realizam na RAM Interna. Existem quase todas as transferências que se possa imaginar.

		bytes	MC	CY	AC	OV
MOV A,	Rn	1	1	-	-	-
	direto	2	1			
	@Ri	1	1			
	#data	2	1			
MOV Rn,	A	1	1			
	direto	2	2			
	#data	2	1			
MOV direto,	A	2	1			
	Rn	2	2			
	direto	3	2			
	@Ri	2	2			
	#data	3	2			
MOV @Ri	A	1	1			
	direto	2	2			
	#data	2	1			

Pode-se pensar que existem todas as combinações possíveis entre **A**, **Rn**, **direto**, **@Ri** e **#data**, mas existem algumas exceções:

Todas as combinações possíveis:

MOV	A	,	A
	Rn		Rn
	direto		Direto
	@Ri		@Ri
			#data

Instruções que não existem:

MOV	A	,	A
	Rn		Rn
	Rn		@Ri
	@Ri		@Ri
	@Ri		Rn

4.6.2. Permutação de Bytes:

Toda permutação opera na RAM Interna. Estas instruções são muito úteis pois permitem a troca de conteúdo entre dois registros sem a necessidade de usar um outro auxiliar. Existem 3 permutações e todas usam o acumulador como um dos operandos.

		bytes	MC	CY	AC	OV
XCH A,	Rn	1	1	-	-	-
	direto	2	1			
	@Ri	1	1			

4.6.3. Permutação de Nibbles:

Existe só uma instrução e esta opera com a RAM Interna. Nesta instrução sempre se usa o acumulador e um operando (por endereçamento indireto). É útil para trabalhar com BCD e em conversões de hexadecimal para código ASCII.

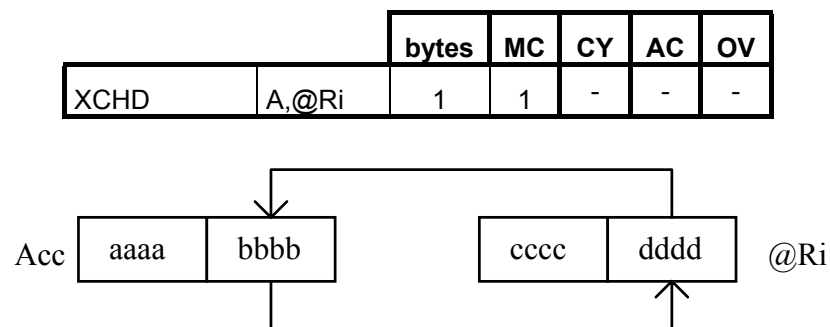


Figura 4.2. Gráfico da permutação de nibbles.

4.6.4. Operações com a Pilha:

Como já foi estudada, a pilha opera exclusivamente com a RAM Interna. Existem apenas duas instruções de pilha e trabalham com endereçamento direto. Quer dizer, sempre usam dois bytes para o opcode e não se pode usar a instrução PUSH A, mas sim PUSH Acc. Deve-se lembrar que Acc é o label para o endereço do acumulador. Na instrução PUSH, primeiro incrementa-se o ponteiro (SP) e em seguida escreve-se a informação. Exemplo: se SP=7, PUSH Acc colocará o conteúdo do acumulador no endereço 8. A instrução POP trabalha ao contrário.

		bytes	MC	CY	AC	OV
PUSH	direto	2	2	-	-	-
POP	direto					

4.6.5. Transferências de Dados com a Memória de Dados Externa:

Existem instruções para leitura e escrita, as quais sempre usam o acumulador como fonte ou destino do dado. É possível fazer dois acessos, um com endereçamento de 16 bits usando o DPTR e outro com endereçamento de 8 bits usando @Ri (@R0 ou @R1). Deve-se lembrar que quando se usa endereçamento de 8 bits, o conteúdo de P2 permanece inalterado e representa os 8 bits mais significativos do endereço. A instrução usada é MOVX, onde X indica que se trabalhará com Memória de Dados Externa.

LEITURA		bytes	MC	CY	AC	OV
MOVX A,	@Ri	1	2	-	-	-
	DPTR					

ESCRITA		bytes	MC	CY	AC	OV
MOVX @Ri,	A	1	2	-	-	-
MOVX @DPTR,						

4.6.6. Leitura da Memória de Programa:

Quando se trabalha com Memória de Programa separada da Memória de Dados Externa, a Memória de Programa só pode ser lida (sinal *PSEN). Existem duas instruções para realizar esta operação e também são as únicas que usam endereçamento indexado. São muito convenientes para construção de tabelas de consulta. O mnemônico empregado muda para MOVC onde o C indica Memória de Códigos.

LEITURA		bytes	MC	CY	AC	OV
MOVC A,	@A+DPTR	1	2	-	-	-
	@A+PC					

4.7. INSTRUÇÕES BOOLEANAS

São chamadas de instruções booleanas as que trabalham com bits. Todas essas instruções operam com um só bit por vez e o CARRY (C) funciona como se fosse o acumulador, ou seja, é um dos operandos e normalmente armazena o resultado. Os bits são endereçados diretamente, mas quando se usa o CARRY este é endereçado implicitamente. Existem também desvios baseados no estado dos bits; estes são relativos à posição atual do PC e se pode saltar 127 bytes adiante ou 128 bytes para trás.

4.7.1. Zerar/ Setar /Complementar um Bit:

Para carregar zero: usa-se a instrução → **CLR**

Para carregar um: usa-se a instrução → **SETB**

Para complementar: usa-se a instrução → **CPL**

		bytes	MC	CY	AC	OV
CLR	C	1	1	0	-	-
	bit	2	1	-	-	-
SETB	C	1	1	1	-	-
	bit	2	1	-	-	-
CPL	C	1	1	X	-	-
	bit	2	1	-	-	-

4.7.2. AND/OR Booleano:

Há dois AND e dois OR. Nestas instruções o carry (C) trabalha como o acumulador, quer dizer, é um dos operandos e também recebe o resultado. Exemplo: $ANL\ C, bit \rightarrow C = C \text{ AND } bit$. Empregam-se os mnemônicos ANL para Logical AND e ORL para Logical OR. Nas instruções em que um operando é "/bit", indica que a operação se realiza com o complemento deste bit. Essas operações são muito convenientes quando se implementam operações booleanas.

		bytes	MC	CY	AC	OV
ANL C,	bit	2	1	X	-	-
	/bit					
ORL C,	bit	2	1	X	-	-
	/bit					

4.7.3. Movimento de Bits:

São duas as instruções para movimento de bits e sempre envolvem o CARRY.

	bytes	MC	CY	AC	OV
MOV C , bit	2	1	X	-	-
MOV bit , C	2	2	-	-	-

4.7.4. Desvios Baseados em Bits:

São 5 as instruções de desvio: duas se baseiam no CARRY e as outras três operam com qualquer bit. Os mnemônicos **JB** e **JC** são usados para desviar se o bit está ativado (em um) e

JNB e **JNC** para desviar se o bit está desativado (em zero). Existe o mnemônico **JBC** que desvia se o bit está ativado e depois complementa este bit.

		bytes	MC	CY	AC	OV
JC	rel	2	2	-	-	-
JNC						
JB	bit,rel	3	2			
JNB						
JBC						

4.8. INSTRUÇÕES DE DESVIO

Nas instruções de desvio estão incluídas as chamadas de subrotinas (**CALL**) e os desvios (**JMP**); estes podem ser realizados com 11 ou 16 bits. Também estão incluídos os desvios condicionais (**JZ**, **JNZ**, **CJNE**), os loops (**DJNZ**) e as instruções de retorno (**RET**, **RETI**).

4.8.1. Chamadas de Subrotinas:

São duas as instruções de chamada de subrotinas: a instrução **LCALL**, que usa um endereço de 16 bits e por isso permite acesso a qualquer posição nos 64 KB de programa, e a instrução **ACALL**, que usa apenas 11 bits, ou seja, permite acesso dentro de um bloco de 2 KB da memória de programa; os 11 bits menos significativos do PC são trocados pelos bits especificados na instrução. A vantagem é que **ACALL** necessita apenas de 2 bytes para o opcode enquanto que **LCALL** precisa de 3 bytes.

		bytes	MC	CY	AC	OV
LCALL	adr16	3	2	-	-	-
ACALL	adr11	2	2			

4.8.2. Retorno de Subrotinas:

São 2 as instruções de retorno de subrotinas. A instrução **RET** é o retorno usual. A instrução **RETI** é usada para retorno das subrotinas que atendem as interrupções, indicando o fim de uma rotina de interrupção. Mais adiante, no capítulo de interrupções, será abordado este tema com maior detalhe.

	bytes	MC	CY	AC	OV
RET	1	2	-	-	-
RETI					

4.8.3. Desvios:

São quatro as instruções de desvios. Há dois desvios que são idênticos às chamadas de subrotinas e que também permitem saltar numa faixa de 2 KB (AJMP) ou de 64 KB (LJMP). Os outros dois desvios são relativos.

		bytes	MC	CY	AC	OV
AJMP	adr11	2	2	-	-	-
LJMP	adr16	3	2			
SJMP	Rel	1	2			
JMP	@A+DPTR	1	2			

4.8.4. Desvios Condicionais:

Há dois desvios que são feitos com os valores do acumulador: **JZ**, que salta se o conteúdo do acumulador é zero, e **JNZ**, que salta se o conteúdo do acumulador não é zero. Deve-se notar que não existe o flag zero (como nos Z80 e 8085); a decisão para o desvio é feita com o conteúdo atual do acumulador. Há outras 4 instruções que usam o mnemônico **CJNE**, que significa "compare dois elementos e desvie se forem diferentes". Todos estes desvios são relativos e por isso têm um alcance de 127 bytes para adiante e 128 bytes para trás.

		bytes	MC	CONDIÇÃO	CY	AC	OV
JZ	rel	2	2	se Acc = 0	-	-	-
JNZ				se Acc 0			
CJNE	A ,direto ,rel	3	2	se Acc Direto			
CJNE	A	3	2	se Acc (#data)			
	Rn ,#data ,rel			se Rn (#data)			
	@Ri			se @Ri (#data)			

4.8.5. Loops :

As instruções de loops são do tipo "decremente e desvie se for diferente de zero" (DJNZ). Há duas instruções, as quais são muito úteis para a repetição de determinadas partes do programa. Utilizam como contador um registro ou um byte da RAM Interna e por isso têm um limite de até 256 repetições. Como são baseadas em desvios relativos, têm um alcance de 127 bytes para adiante e de 128 bytes para trás.

			bytes	MC	CY	AC	OV
DJNZ	Rn	,rel	2	2	-	-	-
	direto						

4.8.6. Não Operação :

Há uma instrução que não faz nada, consumindo apenas ciclos da CPU. É muito útil para reservar espaços na memória de programa quando se trabalha com EPROM.

	bytes	MC	CY	AC	OV
NOP	1	1	-	-	-

4.9. INSTRUÇÕES E FLAGS

Para a perfeita utilização das instruções é necessário conhecer como estas alteram os flags. Em seguida apresenta-se uma pequena tabela com um resumo dessas informações. Na tabela, 0 (zero) indica que o flag é apagado, 1 (um) indica que é ativado, X (don't care) significa que o flag será ativado ou apagado de acordo com o resultado da instrução e "-" indica que o flag não é alterado.

INSTR/FLAG	C	OV	AC
ADD	X	X	X
ADDC	X	X	X
SUBB	X	X	X
MUL	0	X	-
DIV	0	X	-
DA	X	-	-
RRC	X	-	-
RLC	X	-	-
SETB C	1	-	-
CLR C	0	-	-
CPL C	X	-	-
ANL C,bit	X	-	-
ANL C,/bit	X	-	-
ORL C,bit	X	-	-
ORL C,/bit	X	-	-
MOV C,bit	X	-	-
CJNE	X	-	-

Figura 4.3. Comportamento dos flags.

4.10. OBSERVAÇÕES

Aqui convém fazer uma pausa para alguns comentários e exemplos sobre endereçamento e uso das instruções.

4.10.1. Bancos de Registros :

Existem quatro bancos, selecionados por RS1 e RS0. Estes bancos podem ser acessados como registros, com endereçamento direto ou com endereçamento indireto.

	BANCO 0	BANCO 1	BANCO 2	BANCO 3
	R0 R1...R7	R0 R1...R7	R0 R1...R7	R0 R1...R7
Endereço:	00 01 07	08 09 15	16 17 23	24 25 31

Supondo que o banco 1 esteja selecionado, as instruções seguintes dão como resultado a mesma operação mas algumas consomem menos bytes que outras.

MOV	A,R1	1 Byte	1MC	(A,Rn)
MOV	A,9	2 Bytes	1MC	(A,direto)

Também se pode fazer uma sequência para endereçamento indireto:

MOV	R0,#9	2 Bytes	1MC	
MOV	A,@R0	1 Byte	1MC	
	Total	3 Bytes	2MC	

Os registros de outros bancos são acessados empregando endereçamento direto ou indireto :

MOV	A,23	2 bytes	1 MC	(23 e/ou R7 do banco 2)
-----	------	---------	------	-------------------------

Aqui são apresentadas algumas instruções interessantes para realizar transferências entre registros (supõe-se que o banco 3 esteja selecionado) :

MOV	R1,R6	(não existe)
MOV	R1,30	(equivale a MOV R1,R6)
MOV	25,R6	(equivale a MOV R1,R6)
MOV	R1,R1	(não existe)
MOV	R1,25	(equivale a MOV R1,R1)
MOV	25,R1	(equivale a MOV R1,R1)

4.10.2. Registros Especiais (SFR) :

Deve-se notar que todos os SFR, exceto o Acumulador, são acessados empregando endereçamento direto. Para que exista compatibilidade com o 8052, não se permite endereçamento indireto para os SFR. No 8052, quando se usa endereçamento indireto com endereços maiores que 07FH, acessa-se a região chamada 128 UPPER. As duas instruções a seguir dão o mesmo resultado; na verdade são idênticas e geram o mesmo opcode.

MOV	B,#10	2 bytes	1 MC
MOV	0F0H,#10	2 bytes	1 MC

O "B" que se usa aqui é na verdade um label para o endereço 0F0H da RAM Interna. Todos os programas montadores que trabalham com a família MCS-51 trazem os endereços dos SFR já definidos, quer dizer, é como se existissem as seguintes declarações (isto já havia sido afirmado no início deste capítulo) :

B	EQU	0F0H
Acc	EQU	0E0H
PSW	EQU	0D0H
	...	
SP	EQU	081H
P0	EQU	080H

Deve-se ter cuidado com o acumulador pois existem dois símbolos para o mesmo: A e Acc:

A	→	para as instruções com registro específico
Acc	→	para usar acumulador por endereçamento direto

Esses dois símbolos permitem construir instruções interessantes que, se não forem levadas em conta, podem consumir muita memória. Exemplos :

MOV	R7,A	1 byte	1 MC (MOV Rn,A)
MOV	R7,Acc	2 bytes	1 MC (MOV Rn,direto)
MOV	A,Acc	2 bytes	1 MC (MOV A,direto)
MOV	A,B	2 bytes	1 MC (MOV A,direto)
MOV	Acc,B	3 bytes	2 MC (MOV direto,direto)

4.11. CÓDIGOS DE OPERAÇÃO (OPCODE)

As instruções da família MCS-51, para que possam ser executadas pelo microcontrolador, devem ser traduzidas em códigos de operação (opcode). São estes opcodes que a CPU compreende e executa. A cada instrução é associado somente um opcode.

4.11.1. Tabelas de Instruções :

As seguintes tabelas mostram a distribuição dos opcodes da família MCS-51. As tabelas devem ser vistas por colunas. Notar que o opcode "A5" não tem instrução.

	0	1	2	3	4	5	6	7
0	NOP	JBC bit,rel	JB bit,rel	JNB bit,rel	JC rel	JNC rel	IJZ rel	JNZ rel
1	AJMP adr11	ACALL adr11	AJMP adr11	ACALL adr11	AJMP adr11	ACALL adr11	AJMP adr11	ACALL adr11
2	LJMP adr16	LCALL adr16	RET	RETI	ORL dir,A	ANL dir,A	XRL dir,A	ORL C,bit
3	RR A	RRC A	RL A	RLC A	ORL dir,#dt	ANL dir,#dt	XRL dir,#dt	JMP @A+DPTR
4	INC A	DEC A	ADD A,#dt	ADDC A,#dt	ORL A,#dt	ANL A,#dt	XRL A,#dt	MOV A,#dt
5	INC dir	DEC dir	ADD A,dir	ADDC A,dir	ORL A,dir	ANL A,dir	XRL A,dir	MOV dir,#dt
6	INC @R0	DEC @R0	ADD A,@R0	ADDC A,@R0	ORL A,@R0	ANL A,@R0	XRL A,@R0	MOV @R0,#dt
7	INC @R1	DEC @R1	ADD A,@R1	ADDC A,@R1	ORL A,@R1	ANL A,@R1	XRL A,@R1	MOV @R1,#dt
8	INC R0	DEC R0	ADD A,R0	ADDC A,R0	ORL A,R0	ANL A,R0	XRL A,R0	MOV R0,#dt
9	INC R1	DEC R1	ADD A,R1	ADDC A,R1	ORL A,R1	ANL A,R1	XRL A,R1	MOV R1,#dt
A	INC R2	DEC R2	ADD A,R2	ADDC A,R2	ORL A,R2	ANL A,R2	XRL A,R2	MOV R2,#dt
B	INC R3	DEC R3	ADD A,R3	ADDC A,R3	ORL A,R3	ANL A,R3	XRL A,R3	MOV R3,#dt
C	INC R4	DEC R4	ADD A,R4	ADDC A,R4	ORL A,R4	ANL A,R4	XRL A,R4	MOV R4,#dt
D	INC R5	DEC R5	ADD A,R5	ADDC A,R5	ORL A,R5	ANL A,R5	XRL A,R5	MOV R5,#dt
E	INC R6	DEC R6	ADD A,R6	ADDC A,R6	ORL A,R6	ANL A,R6	XRL A,R6	MOV R6,#dt
F	INC R7	DEC R7	ADD A,R7	ADDC A,R7	ORL A,R7	ANL A,R7	XRL A,R7	MOV R7,#dt

	8	9	A	B	C	D	E	F
0	SIMP rel	MOV DPTR,#dt16	ORL C,/bit	ANL C,/bit	PUSH dir	POP dir	MOVX A,@DPTR	MOVX @DPTR,A
1	AJMP adr11	ACALL adr11	AJMP adr11	ACALL adr11	AJMP adr11	ACALL adr11	AJMP adr11	ACALL adr11
2	ANL C,bit	MOV bit,C	MOV C,bit	CPL bit	CLR bit	SETB bit	MOVX A,@R0	MOVX @R0,A
3	MOVC A,@A+PC	MOVC A,@A+DPTR	INC DPTR	CPL C	CLR C	SETB C	MOVX A,@R1	MOVX @R1,A
4	DIV AB	SUBB A,#dt	MUL AB	CJNE A,#dt,rel	SWAP A	DA A	CLR A	CPL A
5	MOV dir,dir	SUBB A,dir	-----	CJNE A,dir,rel	XCH A,dir	DJNZ dir,rel	MOV A,dir	MOV dir,A
6	MOV dir,@R0	SUBB A,@R0	MOV @R0,dir	CJNE @R0,#dt,rel	XCH A,@R0	XCHD A,@R0	MOV A,@R0	MOV @R0,A
7	MOV dir,@R1	SUBB A,@R1	MOV @R1,dir	CJNE @R1,#dt,rel	XCH A,@R1	XCHD A,@R1	MOV A,@R1	MOV @R1,A
8	MOV dir,R0	SUBB A,R0	MOV R0,dir	CJNE R0,#dt,rel	XCH A,R0	DJNZ R0,rel	MOV A,R0	MOV R0,A
9	MOV dir,R1	SUBB A,R1	MOV R1,dir	CJNE R1,#dt,rel	XCH A,R1	DJNZ R1,rel	MOV A,R1	MOV R1,A
A	MOV dir,R2	SUBB A,R2	MOV R2,dir	CJNE R2,#dt,rel	XCH A,R2	DJNZ R2,rel	MOV A,R2	MOV R2,A
B	MOV dir,R3	SUBB A,R3	MOV R3,dir	CJNE R3,#dt,rel	XCH A,R3	DJNZ R3,rel	MOV A,R3	MOV R3,A
C	MOV dir,R4	SUBB A,R4	MOV R4,dir	CJNE R4,#dt,rel	XCH A,R4	DJNZ R4,rel	MOV A,R4	MOV R4,A
D	MOV dir,R5	SUBB A,R5	MOV R5,dir	CJNE R5,#dt,rel	XCH A,R5	DJNZ R5,rel	MOV A,R5	MOV R5,A
E	MOV dir,R6	SUBB A,R6	MOV R6,dir	CJNE R6,#dt,rel	XCH A,R6	DJNZ R6,rel	MOV A,R6	MOV R6,A
F	MOV dir,R7	SUBB A,R7	MOV R7,dir	CJNE R7,#dt,rel	XCH A,R7	DJNZ R7,rel	MOV A,R7	MOV R7,A

4.11.2. Instruções em Ordem Alfabética com OPCODES :

A seguir serão apresentadas todas as instruções em ordem alfabética. Esta lista é para a construção dos opcodes. As tabelas também apresentam essas mesmas informações, mas aqui elas estão mais detalhadas.

OBSERVAÇÃO: Empregar-se-á a seguinte notação :

n = 0,1,2,3,4,5,6,7

i = 0,1

data = palavra de 8 bits (byte)

data16 = palavra de 16 bits

adr16 = um endereço de 16 bits (Memória Externa)

adr11 = um endereço de 11 bits (Memória Externa)

direct ou **dir** = um endereço da RAM Interna (8 bits)

rel = um deslocamento relativo

MSB = byte mais significativo de uma palavra de 16 bits

LSB = byte menos significativo de uma palavra de 16 bits

1)	ACALL	adr11	<table border="1"><tr><td>A10 A9 A8 1</td><td>0 0 0 0</td><td>A7 A6 A5 A4</td><td>A3 A2 A1 A0</td></tr></table>	A10 A9 A8 1	0 0 0 0	A7 A6 A5 A4	A3 A2 A1 A0
A10 A9 A8 1	0 0 0 0	A7 A6 A5 A4	A3 A2 A1 A0				
2)	ADD	A,Rn	28+n				
3)	ADD	A,direct	25 direct				
4)	ADD	A,@Ri	26+i				
5)	ADD	A,#data	24 data				
6)	ADDC	A,Rn	38+n				
7)	ADDC	A,direct	35 direct				
8)	ADDC	A,@Ri	36+i				
9)	ADDC	A,#data	34 data				
10)	AJMP	adr11	<table border="1"><tr><td>A10 A9 A8 0</td><td>0 0 0 0</td><td>A7 A6 A5 A4</td><td>A3 A2 A1 A0</td></tr></table>	A10 A9 A8 0	0 0 0 0	A7 A6 A5 A4	A3 A2 A1 A0
A10 A9 A8 0	0 0 0 0	A7 A6 A5 A4	A3 A2 A1 A0				
11)	ANL	A,Rn	58+n				
12)	ANL	A,direct	55 direct				
13)	ANL	A,@Ri	56+i				
14)	ANL	A,#data	54 data				
15)	ANL	direct,A	52 direct				
16)	ANL	direct,#data	53 direct data				
17)	ANL	C,bit	82 bit				
18)	ANL	A,/bit	B0 bit				
19)	CJNE	A,direct,rel	B5 direct relativo				
20)	CJNE	A,#data,rel	B4 data direct				
21)	CJNE	Rn,#data,rel	B8+n data relativo				

22)	CJNE	@Ri,#data,rel	B6+i data	relativo	
23)	CLR	A	E4		
24)	CLR	bit	C2	bit	
25)	CLR	C	C3		
26)	CPL	A	F4		
27)	CPL	bit	B2	bit	
28)	CPL	C	B3		
29)	DA	A	D4		
30)	DEC	A	14		
31)	DEC	Rn	18+n		
32)	DEC	direct	15	direct	
33)	DEC	@Ri	16+i		
34)	DIV	AB	84		
35)	DJNZ	Rn,rel	D8+n	relativo	
36)	DJNZ	direct,rel	D5	direct	relativo
37)	INC	A	04		
38)	INC	Rn	08+n		
39)	INC	direct	05	direct	
40)	INC	@Ri	06+i		
41)	INC	DPTR	A3		
42)	JB	bit,rel	20	bit	relativo
43)	JBC	bit,rel	10	bit	relativo
44)	JC	rel	40	relativo	
45)	JMP	@A+DPTR	73		
46)	JNB	bit,rel	30	bit	relativo
47)	JNC	rel	50	relativo	
48)	JNZ	rel	70	relativo	
49)	JZ	rel	60	relativo	
50)	LCALL	adr16	12	MSB(adr16)	LSB(adr16)
51)	LJMP	adr16	02	MSB(adr16)	LSB(adr16)
52)	MOV	A,Rn	E8+n		
53)	MOV	A,direct	E5	direct	
54)	MOV	A,@Ri	E6+i		
55)	MOV	A,#data	74	data	
56)	MOV	Rn,A	F8+n		
57)	MOV	Rn,direct	A8+n	direct	
58)	MOV	Rn,#data	78+n	data	
59)	MOV	direct,A	F5	direct	

60)	MOV	direct,Rn	88+n	direct	
61)	MOV	direct,direct	85	dir(fonte)	dir(destino)
62)	MOV	direct,@Ri	86+i	direct	
63)	MOV	direct,#data	75	direct	data
64)	MOV	@Ri,A	F6+i		
65)	MOV	@Ri,direct	A6+i	direct	
66)	MOV	@Ri,#data	76+i	data	
67)	MOV	bit,C	92	bit	
68)	MOV	C,bit	A2	bit	
69)	MOV	DPTR,#data16	90	MSB(data16)	LSB(data16)
70)	MOVC	A,@A+DPTR	93		
71)	MOVC	A,@A+PC	83		
72)	MOVX	A,@Ri	E2+i		
73)	MOVX	A,@DPTR	E0		
74)	MOVX	@Ri,A	F2+i		
75)	MOVX	DPTR,A	F0		
76)	MUL	AB	A4		
77)	NOP		00		
78)	ORL	A,Rn	48+n		
79)	ORL	A,direct	45	direct	
80)	ORL	A,@Ri	46+i		
81)	ORL	A,#data	44	data	
82)	ORL	direct,A	42	direct	
83)	ORL	direct,#data	44	data	
84)	ORL	C,bit	72	bit	
85)	ORL	C,/bit	A0	bit	
86)	POP	direct	D0	direct	
87)	PUS	direct	C0	direct	
88)	RET		22		
89)	RETI		32		
90)	RL	A	23		
91)	RLC	A	33		
92)	RR	A	03		
93)	RRC	A	13		
94)	SETB	bit	D2	bit	
95)	SETB	C	D3		
96)	SJMP	rel	80	relative	
97)	SUBB	A,Rn	98+n		

98)	SUBB	A,direct	95	direct	
99)	SUBB	A,@Ri	96+i		
100)	SUBB	A,#data	94	data	
101)	SWAP	A	C4		
102)	XCH	A,Rn	C8+n		
103)	XCH	A,direct	C5	direct	
104)	XCH	A,@Ri	C6+i		
105)	XCHD	A,@Ri	D6+i		
106)	XRL	A,Rn	68+i		
107)	XRL	A,direct	65	direct	
108)	XRL	A,@Ri	66+i		
109)	XRL	A,#data	64	data	
110)	XRL	direct,A	62	direct	
111)	XRL	direct,#data	63	direct	data

4.12. JUMP E CALL

Segundo o modo de formação do endereço pode-se identificar 3 tipos de JUMPs e dois tipos de CALLs:

Os 3 JUMPs são: relativo, absoluto e longo.

Os 2 CALLs são: absoluto e longo.

Neste item serão esclarecidas estas instruções para evitar erros que são muito freqüentes quando se começa a trabalhar com a família MCS-51.

4.12.1. JUMPs Relativos :

Existem dois tipos de JUMPs relativos: os incondicionais e os condicionais. Estes últimos (os condicionais) realizam um desvio de acordo com o estado de um registro, de um flag ou de um bit, enquanto o outro tipo (incondicional) sempre realiza o desvio. Exemplo:

- JUMP incondicional: SJMP rel
- JUMP condicional: JNC rel

Nos JUMPs relativos especifica-se quantos bytes (deslocamento) se deseja desviar, para frente ou para trás. Este deslocamento é codificado em complemento a 2. Como há apenas um byte para este deslocamento, pode-se saltar 127 bytes para frente ou 128 bytes para trás.

Agora serão apresentados alguns exemplos com JUMPs relativos. Os programas, por simplicidade, sempre iniciarão no endereço 200H. Ao lado de cada instrução estarão os opcodes.

EXEMPLO 1 : DESVIO CONDICIONAL PARA FRENTE

	ORG	200H
	ADD	A,#30H
	JC	LABEL
	ADD	A,#20H
	DA	A
LABEL	MOV	B,A

O programa apresenta um desvio condicional para frente. A seguir tem-se este mesmo programa com os endereços de memória e seu conteúdo.

endereço	conteúdo	instrução	deslocamento
200	24	ADD A,#30H	-4
201	30		-3
202	40	JC LABEL	-2
203	03		-1
204	24	ADD A,#20H	+0
205	20		+1
206	D4	DA A	+2
207(LABEL)	F5	MOV B,A	+3
208	F0		+4

É conveniente ressaltar que quando a instrução "JC LABEL" está sendo executada, o Program Counter (PC) já está apontando para a instrução seguinte; por isso, o deslocamento +0 está na instrução "ADD A,#20H".

EXEMPLO 2 : DESVIO CONDICIONAL PARA TRÁS

	ORG	200H
	INC	A
LABEL	MOV	A,B
	ADD	A,#20H
	DA	A
	JC	LABEL
	MOV	R0,A

O programa apresenta um desvio condicional para trás e a seguir pode-se observar os endereços de memória e seu conteúdo.

endereço	conteúdo	instrução		deslocamento
200	04	INC	A	-8 (F8)
201(LABEL)	E5	MOV	A,B	-7 (F9)
202	F0			-6 (FA)
203	24	ADD	A,#20H	5 (FB)
204	20			-4 (FC)
205	D4	DA	A	-3 (FD)
206	40	JC	LABEL	-2 (FE)
207	F9			-1 (FF)
208	F8	MOV	R0,A	+0
209	75	MOV	B,#50H	+1
20A	F0			+2
20B	50			+3

4.12.2. JUMPs e CALLs Absolutos :

Os JUMPs e CALLs absolutos são sempre incondicionais. Neste tipo de instrução apenas se trocam alguns bits do PC pelos bits especificados na instrução (são trocados os 11 bits menos significativos do PC). Isto significa que os JUMPs e CALLs estão limitados a um alcance de 2 KB, quer dizer, é como se a memória estivesse dividida em páginas de 2 KB e se pudesse saltar dentro dos limites de cada página. Estas instruções permitem JUMPs e CALLs que consomem poucos bytes. Deve-se ter cuidado ao escrever os opcodes.

EXEMPLO 3 : DESVIO ABSOLUTO

```

                ORG      200H
                ADD      A,#30H
                JNC      LABEL
                AJMP     LB1
LABEL          ADD      A,#20H
                ...
                ORG      765H
LB1            DA       A
                ...

```

O programa acima apresenta um desvio condicional para frente e abaixo se tem o programa com os endereços de memória e seu conteúdo.

endereço	conteúdo	instrução		deslocamento
200	24	ADD	A,#30H	-4
201	30			-3
202	50	JNC	LABEL	-2
203	02			-1
204	E0	AJMP	LB1	+0
205	65			+1
206(LABEL)	24	ADD	A,#20H	+2
207	20			+3
...				
765(LB1)	04	DA	A	--

OBSERVAÇÃO:

AJMP	LB1 →	A10 A9 A8 0	0 0 0 0	A7 A6 A5 A4	A3 A2 A1 A0
E0 65		1 1 1 0	0 0 0 0	0 1 1 0	0 1 0 1

EXEMPLO 4 : CALL ABSOLUTO

```

        ORG    400H
        ADD    A,#30H
        JNC    LABEL
        ACALL  LB1
LABEL    ADD    A,#20H
        ...
        ORG    234H
LB1      DA     A
        ...

```

O programa é semelhante ao anterior mas o AJMP foi trocado por um ACALL. Notar que a instrução ACALL está chamando uma rotina que está atrás, mas dentro do bloco de 2 KB.

endereço	conteúdo	instrução		deslocamento
400	24	ADD	A,#30H	-4
401	30			-3
402	50	JNC	LABEL	-2
403	02			-1
404	50	ACALL	LB1	+0
405	34			+1
406(LABEL)	24	ADD	A,#20H	+2

...

ACALL	LB1 →	A10 A9 A8 1	0 0 0 0	A7 A6 A5 A4	A3 A2 A1 A0
50 34		0 1 0 1	0 0 0 0	0 0 1 1	0 1 0 0

EXEMPLO 5 : DESVIO ABSOLUTO (COM ERRO)

	ORG	700H
	ADD	A,#30H
	JNC	LABEL
	AJMP	LB1
LABEL	ADD	A,#20H
	...	
	ORG	900H
LB1	DA	A
	...	

Este programa vai apresentar um erro do tipo "illegal range" devido ao fato que o AJUMP está no primeiro bloco de 2 KB (0 até 7FF) e o label está no segundo bloco (800 até FFF).

EXEMPLO 6 : DESVIO ABSOLUTO (COM ERRO)

	ORG	7FFH
	AJMP	LB1
	...	
	ORG	700H
LB1	DA	A
	...	

Aparentemente não há erro pois o AJMP e o label estão no mesmo bloco de 2 KB. Na verdade, há um erro porque quando se inicia a execução de uma instrução o PC já está apontando para a instrução seguinte, ou seja, quando a CPU inicia a execução do AJMP, o PC está em 801H (AJMP usa dois bytes). O PC e o label estão em dois blocos (de 2 KB) distintos. Os programas montadores devem ser suficientemente inteligentes para detetar este tipo de erro.

4.13. EXEMPLOS

A seguir é apresentada uma sequência de exemplos que vão ilustrar a utilização das instruções do MCS-51. Supõe-se que a CPU é o 8031, que os programas serão usados como subrotinas e que sempre iniciam no endereço 200H. Nos primeiros exemplos serão apresentadas também as listagens em hexadecimal dos opcodes.

EXEMPLO 1. SUBROTINA SUM:

$R7 = R1 + R0$.

Colocar em R7 o resultado da soma de R0 e R1.

```
                ORG    200H
SUM:            MOV    A,R0
                ADD    A,R1
                MOV    R7,A
                RET
```

Listagem em hexadecimal com os opcodes:

200	E8	MOV	A,R0
201	29	ADD	A,R1
202	FF	MOV	R7,A
203	22	RET	

Deve-se ter cuidado para que a soma ($R0+R1$) não seja maior que 256. A melhor solução é ter uma subrotina geral e por isso o resultado é armazenado em 2 registros: R7 (MSB) R6 (LSB). O resultado será maior que 256 quando existir um carry.

A seguir está a nova subrotina SUM ($R7 R6 = R1 + R0$).

```
                ORG    200H
SUM:            MOV    A,R0
                ADD    A,R1
                MOV    R6,A    ;R6 guarda o LSB
                CLR    A      ;zera o acumulador
                ADDC   A,#0    ;acrescenta o carry
                MOV    R7,A    ;R7 guarda o MSB
                RET
```

Listagem em hexadecimal com os opcodes:

200	E8	MOV	A,R0
201	29	ADD	A,R1
202	FE	MOV	R6,A
203	E4	CLR	A
204	34	ADDC	A,#0
205	00		
206	FF	MOV	R7,A
207	22	RET	

A instrução "ADDC A,#0" pode ser substituída pela instrução "RLC A", com a vantagem de se economizar um byte. Isto está na listagem hexadecimal a seguir.

Listagem em hexadecimal com os opcodes:

200	E8	MOV	A,R0
201	29	ADD	A,R1
202	FE	MOV	R6,A
203	E4	CLR	A
204	33	RLC	A
205	FF	MOV	R7,A
206	22	RET	

EXEMPLO 2. SUBROTINA SUB:

$R7 = R1 - R0$.

Colocar em R7 o resultado da subtração de R1 e R0

	ORG	200H	
SUB:	MOV	A,R1	
	CLR	C	;zera o carry (necessario na subtracao)
	SUBB	A,R0	
	MOV	R7,A	
	RET		

Listagem em hexadecimal com os opcodes:

200	E9	MOV	A,R1
201	C3	CLR	C
202	98	SUBB	A,R0
203	FF	MOV	R7,A

204 22 RET

Um cuidado que se deve ter ao usar a instrução SUBB é o de zerar o carry pois o SUBB sempre o utiliza.

EXEMPLO 3. SUBROTINA MUL:

R7 R6 = R1 * R0.

Colocar em R7 (MSB) e em R6 (LSB) o produto de R0 e R1

```

                ORG      200H
MUL:           MOV      A,R0
                MOV      B,R1
                MUL      AB
                MOV      R6,A      ;guarda o LSB
                MOV      R7,B      ;guarda o MSB
                RET

```

Listagem em hexadecimal com os opcodes:

200	E8	MOV	A,R0
201	89	MOV	B,R1
202	F0		
203	A4	MUL	AB
204	FE	MOV	R6,A
205	AF	MOV	R7,A
206	F0	MOV	R7,B
207	22	RET	

EXEMPLO 4. SUBROTINA DIV:

R7(quoc.) R6(resto) R1/R0.

Dividir R1 por R0 e guardar o resultado em: R7(quociente) e R6(resto)

```

                ORG      200H
DIV:           MOV      A,R1      ;dividendo em A
                MOV      B,R0      ;divisor em B
                DIV      AB
                MOV      R7,A      ;guarda o quociente
                MOV      R6,B      ;guarda o resto
                RET

```

Listagem em hexadecimal com os opcodes:

200	E9	MOV	A,R1
201	88	MOV	B,R0
202	F0		
203	84	DIV	AB
204	FF	MOV	R7,A
205	AE	MOV	R6,B
206	F0		
207	22	RET	

EXEMPLO 5. SUBROTINA DIV_7:

Verificar se o byte que está em R3 é múltiplo de 7 e , se for, ativar o bit 0 da porta P1.

	ORG	200H	
DIV_7:	MOV	A,R3	
	MOV	B,#7	
	DIV	AB	;dividir por 7
	MOV	A,B	;coloca o resto em A
	JZ	LB	;ha resto ?
	CLR	P1.0	;zera bit P1.0
	RET		
LB:	SETB	P1.0	;seta bit P1.0
	RET		

Listagem em hexadecimal com os opcodes:

200	EB	MOV	A,R3
201	75	MOV	B,#7
202	F0		
203	07		
204	84	DIV	AB
205	E5	MOV	A,B
206	F0		
207	60	JZ	LB
208	03		
209	C2	CLR	P1.0
20A	90		
20B	22	RET	

20C (LB)	D2	SETB	P1.0
20D	90		
20E	22	RET	

Há uma outra solução que ocupa um byte a menos. Inicia-se com a hipótese de que o número não é divisível por 7 (zerar P1.0) e em seguida se realiza o teste; caso o número for divisível inverte-se P1.0. Economiza-se um RET.

200	C2	CLR	P1.0
201	90		
202	EB	MOV	A,R3
203	75	MOV	B,#7
204	90		
205	07		
206	84	DIV	AB
207	E5	MOV	A,B
208	F0		
209	70	JNZ	LB
20A	02		
20B(LB)	B2	CPL	P1.0
20C	90		
20D	22	RET	

EXEMPLO 6. SUBROTINA INICIALIZAR:

Inicializar com zero toda a RAM interna, ou seja, os endereços de 0 a 127

	ORG	200H	
ZERAR:	CLR	RS1	;seleciona banco 0
	CLR	RS0	;seleciona banco 0
	CLR	A	
	MOV	R0,#127	;maior endereço
LB:	MOV	@R0,A	
	DJNZ	R0,LB	
	RET		

Listagem em hexadecimal com os opcodes:

200	C2	CLR	RS1
201	D4		

202	C2	CLR	RS0
203	D3		
204	E4	CLR	A
205	78	MOV	R0,#127
206	7F		
207(LB)	F6	MOV	@R0,A
208	D8	DJNZ	R0,LB
209	FD		
20A	22	RET	

EXEMPLO 7. SUBROTINA MUL16:

$R7\ R6\ R5\ R4 = (R3\ R2) * (R1\ R0)$

Em R1 e R0 existe um número de 16 bits e em R3 e R2 existe um outro. Multiplicar estes dois números de 16 bits e guardar o resultado em R7, R6, R5 e R4.

A solução é muito simples pois esta multiplicação pode ser vista como 4 multiplicações de 8 bits. A equação a seguir ilustra este efeito.

$$(2^8 \cdot R3 + R2) * (2^8 \cdot R1 + R0) = 2^{16} (R3 \cdot R1) + 2^8 (R3 \cdot R0 + R2 \cdot R1) + (R2 \cdot R0)$$

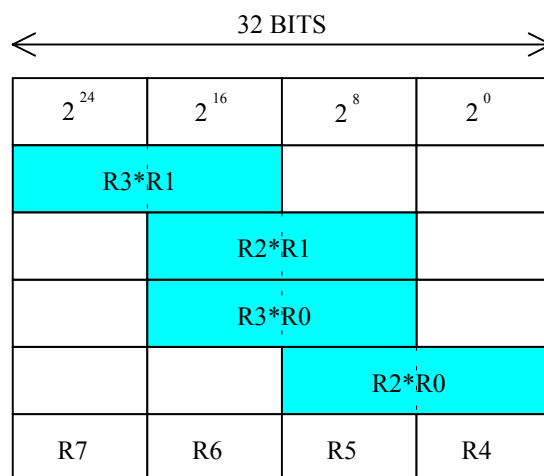


Figura 4.4. Explicação da rotina de multiplicação.

```

MUL16:  MOV    A,R2      ;
        MOV    B,R0      ;
        MUL    AB        ;R2*R0
        MOV    R4,A      ;R4 contem LSB(R2*R0)
        MOV    R5,B      ;R5 contem MSB(R2*R0)
        MOV    A,R3      ;;
        MOV    B,R0      ;;
        MUL    AB        ;R3*R0

```


ADD	A,R5	;A contem $LSB(R3 \cdot R0) + MSB(R2 \cdot R0)$
MOV	R5,A	;coloca em R5 o valor de A
CLR	A	;zera A
ADDC	A,B	;A contem Carry + 0 + $MSB(R3 \cdot R0)$
MOV	R6,A	;coloca em R6 o valor de A
MOV	A,R2	;
MOV	B,R1	;
MUL	AB	;R2*R1
ADD	A,R5	;A contem $LSB(R2 \cdot R1) + LSB(R3 \cdot R0) + MSB(R2 \cdot R0)$
MOV	R5,A	;coloca em R5 o valor de A
MOV	A,B	;A contem $MSB(R2 \cdot R1)$
ADDC	A,R6	;A contem Carry+ $MSB(R3 \cdot R0) + MSB(R2 \cdot R1)$
MOV	R6,A	;coloca em R6 o valor de A
MOV	A,R3	::
MOV	B,R1	::
MUL	AB	;R3*R1
ADD	A,R6	;A contem $LSB(R3 \cdot R1) + MSB(R2 \cdot R1) + MSB(R3 \cdot R0)$
MOV	R6,A	;coloca em R6 o valor de A
CLR	A	;zera A
ADDC	A,B	;A contem Carry + 0 + $MSB(R3 \cdot R1)$
MOV	R7,A	;coloca em R7 o valor de A
RET		

EXEMPLO 8. SUBROTINA SUMBCD:

Em R1 e R0 existem 2 números BCD que deverão ser somados e o resultado deve ser armazenado em R7 e R6

SUMBCD:	MOV	A,R1	
	ADD	A,R0	;A contem R0 + R1
	DA	A	;ajuste decimal depois de uma soma BCD
	MOV	R6,A	;coloca em R6
	CLR	A	;zera A
	ADDC	A,#0	;A contem 0 + Carry + 0
	MOV	R7,A	;coloca em R7
	RET		

EXEMPLO 9. SUBROTINA SUBBCD:

Em R1 e R0 encontra-se um número em BCD de quatro dígitos. Subtrair 86 deste número e colocar o resultado em R7 e R6.

Como o "decimal adjust" (DA A) só funciona em somas, será necessário realizar a subtração utilizando complemento a 10. Deve-se notar que 99 funciona como -1 e que 98 como -2. O complemento a 10 de 86 é 9914.

$$\begin{array}{r} 76 \\ + \quad 99 \text{ (-1)} \\ \hline 1 \quad 75 \\ \uparrow \\ \text{ignorar este 1} \end{array}$$

$$\begin{array}{r} 76 \\ + \quad 98 \text{ (-2)} \\ \hline 1 \quad 74 \\ \uparrow \\ \text{ignorar este 1} \end{array}$$

$$\begin{array}{r} 10000 \\ - \quad 86 \\ \hline 9914 \end{array}$$

complemento a 10 de 86

```

SUBBCD:  MOV     A,R0
         ADD     A,#14H ;R0 + LSB de -86
         DA      A      ;ajuste decimal da soma
         MOV     R6,A    ;coloca em R6
         MOV     A,R1
         ADDC    A,#99   ;R1 + MSB de -86
         DA      A      ;ajuste decimal da soma
         MOV     R7,A    ;coloca em R7
         RET

```

EXEMPLO 10. SUBROTINA BINBCD:

Converter um número binário para BCD.

Existem vários casos de acordo com o valor do número a ser convertido:

- caso a: 0 a 99 (2 algarismos BCD)
- caso b: 0 a 255 (1 byte)
- caso c: 0 a 999 (3 algarismos BCD)
- caso d: 0 a 9999 (4 algarismos BCD)

CASO a: Em R4 existe um número de 0 a 99. Convertê-lo para BCD e colocar o resultado em R6. Ao dividir o número por 10, a unidade termina em Acc e a dezena em B.

```

BINBCD_99: MOV     A,R4
           MOV     B,#10
           DIV     AB      ; A=0X  B=0Y

```

SWAP	A	; A=X0 B=0Y
ADD	A,B	; A=XY B=0Y
MOV	R6,A	;coloca em R6 o valor de A
RET		

CASO b: Em R4 existe um número de 0 a 255. Convertê-lo para BCD e colocar o resultado em R7 e R6. Divide-se o número por 100 para separar a centena e depois usa-se a solução do caso a.

```

BINBCD_255:  MOV    A,R4
              MOV    B,#100
              DIV    AB      ;separar centena (divide por 100)
              MOV    R7,A    ;R7 guarda o quociente (digito BCD mais sigf.)
              MOV    A,B    ;igual a BINBCD_99
              MOV    B,#10
              DIV    AB
              SWAP   A
              ADD    A,B
              MOV    R6,A
              RET

```

CASO c: Em R5 e R4 existe um número de 0 a 999. Convertê-lo para BCD e colocar o resultado em R7 e R6. A solução usada aqui é a de somar 256 em R7 e R6 e decrementar R5 até que este seja 0; depois se usa uma rotina idêntica à BINBCD_255.

```

BINBCD_999:  CLR     A      ;zera A
              MOV    R7,A    ;poe zero em R7 e R6 para
              MOV    R6,A    ;receber os resultados
              MOV    A,R5
              JZ     LB1     ;se R5=0 (0<= num <=255), seguir adiante
              ;
LB2:          MOV    A,R6
              ADD    A,#56H   ;somar 256 BCD e decrementar
              DA     A        ;R5 ate que este chegue
              MOV    R6,A    ;a zero
              MOV    A,R7
              ADDC   A,#2
              DA     A
              MOV    R7,A

```

```

                DJNZ      R5, LB2      ;decrementar R5
                ;
LB1:            MOV      A, R4          ;parecido com BINBCD_255 (0<= num <=255)
                MOV      B, #100
                DIV      AB
                ADD      A, R7
                DA        A
                MOV      R7, A
                MOV      A, B
                MOV      B, #10
                DIV      AB
                SWAP     A
                ADD      A, B
                ADD      A, R6
                DA        A
                MOV      R6, A
                MOV      A, R7          ;ha possibilidade de que se
                ADDC     A, #0          ;necessite passar um carry adiante
                DA        A
                MOV      R7, A
                RET

```

CASO d: Em R5 e R4 existe um número do 0 a 9999. Convertê-lo para BCD e colocar seu resultado em R7 e R6.

A solução adotada no Caso c sugere um loop para zerar R5. Mas para números muito grandes esse loop pode ser excessivo em termos de tempo e por isso a melhor solução seria usar divisões por 1000, 100 e 10. O problema agora será como realizar divisões de 2 números de 16 bits usando a divisão de 8 bits que se tem disponível.

Uma maneira de solucionar este problema é converter o LSB para BCD e depois converter o MSB para BCD, multiplicá-lo por 256 e somar com o anterior. Uma multiplicação por 256 é fácil: multiplique por 200 (multiplique por 2 e desloque 2 posições BCD para a esquerda); multiplique por 50 (multiplique por 5 e desloque 1), multiplique por 6 e finalmente some os três resultados.

Com esta solução pode-se pensar em um caso e mais geral:

CASO e: Em R4 e R3 existe um número binário de 16 bits (0 a 9999). Convertê-lo para BCD e colocar seu resultado em R7, R6 e R5. Na solução serão utilizadas algumas subrotinas auxiliares.

Converte um byte do ACC para BCD e coloca o resultado em AUXH e AUXL

```
BCD8:      MOV      B,#100      ;parecido com BINBCD_255
           DIV      AB
           MOV      AUXH,A
           MOV      A,B
           MOV      B,#10
           DIV      A,B
           SWAP     A
           ADD      A,B
           MOV      AUXL,A
           RET
```

Recebe um número BCD em R2 e R1 e o incrementa o número de vezes especificado em R0 (R0 > 0).

```
ROT_INC:   MOV      R1AUX,R1
           MOV      R2AUX,R2
           DEC      R0
R_INC:     MOV      A,R1AUX
           ADD      A,R1
           DA       A
           MOV      R1,A
           MOV      A,R2AUX
           ADDC     A,R2
           DA       A
           MOV      R2,A
           DJNZ     R0,R_INC
           RET
```

Somar dois números BCD, um número está em R2 e R1, enquanto o outro está indicado pelo ponteiro R0. O resultado deve ser guardado no endereço apontado por R0.

$[@(R0+1) \text{ e } @R0] \text{ } [@(R0+1) \text{ e } @R0] + [R2 \text{ e } R1]$

```
SOMA:      MOV      A,@R0
           ADD      A,R1      ;soma os LSBs
           DA       A        ;ajuste decimal da soma
           MOV      @R0,A     ;guarda a soma em @R0
```

INC	R0	;vai para proximo endereco
MOV	A,@R0	
ADDC	A,R2	;soma os MSBs com Carry
DA	A	;ajuste decimal da soma
MOV	@R0,A	;guarda o MSB da soma
RET		

Aqui está a subrotina principal:

(R7, R6, R5) ← BCD(R4, R3)

BCD16:	MOV	A,R3	;coloca em A o LSB
	ACALL	BCD8	;converte o LSB p/ BCD e coloca
	;		;o resultado em AUXH e AUXL
	MOV	R5,AUXL	;o LSB em BCD vai para R5 e R6
	MOV	R6,AUXH	
	MOV	A,R4	;coloca em A o MSB
	ACALL	BCD8	;converte o LSB p/ BCD e coloca
	;		;o resultado em AUXH e AUXL
	MOV	R2,AUXH	;o MSB em BCD vai para R1 e R2
	MOV	R1,AUXL	
	MOV	R0,#6	
	ACALL	ROT_INC	;Calcula BCD(MSB)*6
	MOV	R0,#5	;Para que a soma seja colocada em R5 e R6
	ACALL	SOMA	;(R6,R5) <- (R6,R5) + 6*BCD(MSB)
	CLR	A	;zera A
	RLC	A	;houve carry? (por seguranca)
	MOV	R7,A	;coloca carry em R7
	;		
	MOV	R2,AUXH	;AUXH e AUXL ainda guardam o BCD(MSB)
	MOV	R1,AUXL	
	MOV	R0,#5	
	ACALL	ROT_INC	;Calcula BCD(MSB)*5
	;		; A R2 R1 deslocar
	MOV	A,R2	; 0X 0X YZ uma
	MOV	R0,#1	
	XCHD	A,@R0	; 0Z 0X YX posicao BCD
	SWAP	A	; Z0 0X YX para a
	XCH	A,R1	; YX 0X Z0 esquerda

```

SWAP      A          ; XY 0X Z0 (multipl. por 10)
XCH       A,R2       ; 0X XY Z0
MOV       R0,#5      ;Para que a soma seja colocada em R5 e R6
ACALL     SOMA        ;(R6,R5) <- (R6,R5)+ 6*BCD(MSB)+50*BCD(MSB)
MOV       A,R7
ADDC      A,#0        ;acrescenta o carry a R7
DA        A
MOV       R7,A
;
MOV       R2,AUXH     ;AUXH e AUXL ainda guardam o BCD(MSB)
MOV       R1,AUXL
MOV       R0,#2
ACALL     ROT_INC     ;Calcula BCD(MSB)*2
MOV       R0,#6       ;Para que a soma seja colocada em R6 e R7
ACALL     SOMA
RET

```

Note que podem ser introduzidas algumas melhorias nas subrotinas de multiplicação: a multiplicação por 2 se resume a uma soma.

$$\begin{aligned}
 2n &= n + n \\
 5n &= 2n + 2n + n \\
 6n &= 5n + n
 \end{aligned}$$

Deve-se notar que existem muitas outras rotinas para converter BIN para BCD.

EXEMPLO 11. SUBROTINA NIB_ASC:

Converter uma nibble que está no Acc para seu correspondente ASCII

Exemplos: 0110 (6) 36H
 1010 (A) 41H

A solução é simples:

Se $n < 10$ → somar 30H
 Se $n \geq 10$ → somar 37H

```

NIB_ASC:  PUSH      Acc          ;coloca o conteudo de A na pilha
          CLR       C            ;zera Carry
          SUBB      A,#10        ;A-10 -> se CY=1 => A<10
          POP       Acc          ;      -> se CY=0 => A>=10
          JNC       NIB_ASC1     ;

```

```

      ADD      A,#30H      ;de 30 a 39 (algarismos)
      RET
NIB_ASC1: ADD      A,#37H      ;de 41 a 46 (letras)
      RET

```

EXEMPLO 12 SUBROTINA HEX_ASC:

Converter o byte que está em Acc em seus 2 correspondentes ASCII, guardando o mais significativo em Acc e o menos significativo em B.

Exemplo: Acc = 0110 1010 Acc = 36H e B = 41H

```

HEX_ASC: MOV      B,A      ;coloca em B uma copia de A
      ANL      A,#0FH      ;zera nibble mais sigf. de A
      ACALL     NIB_ASC
      XCH      A,B      ;troca os valores de A e B
      ANL      A,#0F0H      ;zera a nibble menos sigf. de A
      SWAP     A      ;troca as nibbles de A
      ACALL     NIB_ASC
      RET

```

EXEMPLO 13. SUBROTINA ASC_HEX:

Converter os 2 caracteres ASCII que se encontram em Acc (mais significativo) e B (menos significativo) em seu binário correspondente

Exemplo: Acc = 43H e B = 38H => Acc = 1100 1000

C 8

```

ASC_HEX: ACALL     ASCNIB
      SWAP     A      ;troca as nibbles em A
      XCH      A,B      ;troca valores entre A e B
      ACALL     ASCNIB
      ADD      A,B      ;coloca as 2 nibbles em A
      RET

```

Subrotina auxiliar, recebe um caracter ASCII no Acc e retorna sua nibble correspondente.

```

ASCNIB:  CLR      C      ;zera Carry
      SUBB     A,#30H      ;A-30H -> se CY=1 => A<30 (ERRO)
      JC       ERRO      ;      -> se CY=0 => A>=30
      PUSH     Acc      ;guarda A-30H na pilha
      SUBB     A,#10      ;A-10 -> se CY=1 => A<10

```


	JNC	ASCNIB1	; -> se CY=0 => A>=10
	POP	Acc	;restaura o valor A-30H
	RET		
	;		
ASCNIB1:	SUBB	A,#6	;A-6 -> se CY=1 => A<6
	POP	Acc	; -> se CY=0 => A>=6
	JNC	ERRO	
	SUBB	A,#6	;Notar que CY=1 (ou seja, A-37H)
	RET		
	;		
ERRO:	CPL	P1.7	;indica erro
	SJMP	ERRO	;fica em loop

EXEMPLO 14. UMA SUGESTÃO:

Uma solução para divisões quando o dividendo é fixo e tem mais de um byte: seja o caso em que é necessário dividir um número N por 836; não se pode fazer diretamente $N/836$ mas $836 \approx (\text{aprox.}) 65536/78$ e então se usa o seguinte truque: $(N*78)/65536$, ou seja, multiplica-se por 78 e se deixam de fora os dois últimos bytes.

No caso de conversão de binário para BCD de um número de 0 até 9999 é necessário dividi-lo por 1000. Esta divisão pode ser feita através de uma multiplicação seguida de uma divisão por um número que seja uma potencia de dois. Deve-se lembrar que dividir um número por 2 corresponde a 1 shift right, por 4 corresponde a 2 shift right, por 8 corresponde a 3 shift right, ...

A divisão por 1000 pode ser feita usando vários valores de N e n; a tabela a seguir ilustra o erro cometido.

$$1000 = \frac{N}{n}$$

N	n	erro %	delta %	bits fora
1024	1	97,656	-2,344	10
2048	2	97,656	-2,344	11
4096	3	97,656	-2,344	12
8192	4	97,656	-2,344	13
16384	16	97,656	-2,344	14
32768	33	100,708	+0,708	15
65536	66	100,708	+0,708	16
131072	131	99,945	-0,055	17
262144	262	99,945	-0,055	18
524288	524	99,945	-0,055	19
1048576	1049	100,040	+0,044	20
2097152	2097	99,993	-0,007	21
4194304	4194	99,993	-0,007	22

Um bom exercício é desenvolver uma rotina para converter de binário para BCD usando este truque para a divisão por 1000. Verificar também a precisão (deve-se usar valores de N e n que resultem em delta % < 0 e, para obter o resto, multiplica-se o resultado por 1000 e subtrai-se do original).

CAPÍTULO V

ASSEMBLER E SIMULADOR

5.1. CONCEITOS DO AVMAC51 E DO AVLINK

O AVMAC 8051 é um assembler (realocável), com recursos de macros para a família do MCS-51. O AVMAC51 recebe como entrada um arquivo com instruções em assembly e procede da seguinte forma:

- Envia o arquivo a um pré-processador para validar as macros e as diretivas para assemblagem condicional.
- O pré-processador produz um segundo arquivo que é enviado ao assembler propriamente dito.
- O assembler produz um arquivo objeto e um arquivo com a listagem do código-fonte.
- O objeto produzido pelo assembler é enviado (pelo usuário) ao linker, o qual produz um arquivo absoluto em formato .HEX.



Figura 5.1. Fluxograma para utilização do assembler e linker.

Para ativar o assembler usa-se o comando que está ilustrado na figura 5.2.

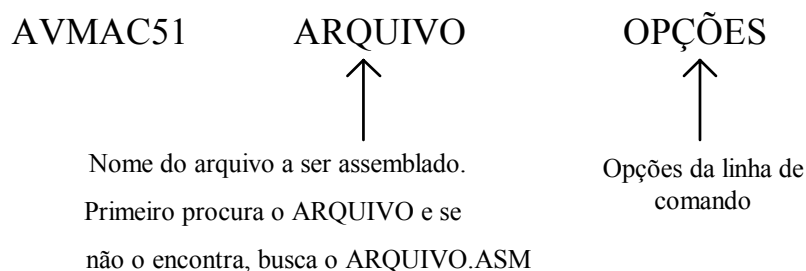


Figura 5.2. Linha de comando para a ativação do assembler.

O linker é usado da forma ilustrada na figura 5.3.

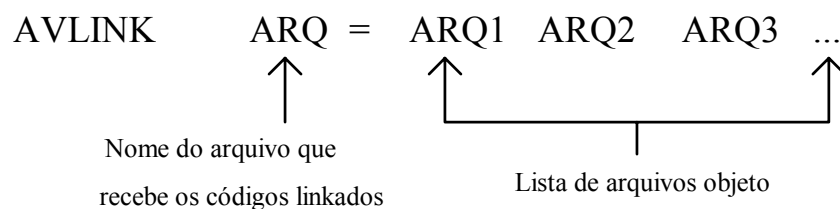


Figura 5.3. Linha de comando para a ativação do linker.

Por default, os arquivos absolutos serão feitos em formato INTEL HEX.

Um programa em linguagem assembly consiste de uma seqüência de "sentenças", cada uma ocupando uma linha do arquivo. Cada sentença pode ter até 4 campos. Os campos são separados por espaços e tabulações.

Label	Operação	Operando	Comentário
-------	----------	----------	------------

Existem sentenças de 2 tipos:

- Instruções → do conjunto de instruções do microcontrolador
- Pseudo-instruções → que orientam o assembler.

LABEL: é um identificador que opcionalmente pode ser seguido por : (dois pontos). Ao usá-lo, é definido como um símbolo de usuário e atribui-se o endereço do primeiro byte da instrução. Os labels devem ser colocados na primeira coluna do texto.

OPERAÇÃO: contém uma instrução do microcontrolador (em linguagem assembly) ou uma pseudo-instrução (diretiva para o assembler).

OPERANDO: Pode haver nenhum, um, dois ou mais operandos; no caso de 2 ou mais operandos estes são separados por vírgulas.

COMENTÁRIO: Qualquer seqüência de caracteres precedida por ; (ponto e vírgula).

IMPORTANTE:

Labels → devem começar na primeira coluna.
 Códigos → não devem começar na primeira coluna.
 Comentários → pode começar na primeira coluna.

Os **símbolos** (labels) podem ter quantos caracteres se deseje mas somente os primeiros 32 são significativos. Podem ser: A → Z, 0 → 9, \$.

Os **números** podem estar em outras bases diferentes da decimal, isto é, podem estar nas seguintes bases: 2, 8 ou 16; neste caso devem estar precedidos ou seguidos por:

BASE	PRECEDIDO	SEGUIDO
2	%	B
8	@	O ou Q
16	\$	H

Exemplo: 254 → 0FEH ou \$FE

127 → 1111111B ou %1111111

Para definir as **constantes de caracteres** usa-se ' ou "; estas são calculadas como inteiros de 16 bits. Em caso de uma letra, o byte de ordem alta é zero.

O símbolo \$ é o contador de endereços do assembler (assembly-time counter). O valor de \$ é o endereço do primeiro byte da instrução atual.

Nas **expressões aritméticas** os símbolos, números, caracteres e \$ são referenciados como valores inteiros de 16 bits. Deve-se ter cuidado com os valores negativos, que são expressados em complemento a 2. Por exemplo, -1 é igual a FFFFH, então:

$$-1 + 5 = 4$$

-1 LT 0 => Falso, pois -1(FFFFH) não é menor que zero.

(Comparações não usam sinais).

OPERADOR	USO	RESULTADO
+	$x+y$	$x+y$
+	$+x$	$0+x$
-	$x-y$	$x-y$
-	$-x$	$0-x$
*	$x*y$	$x*y$ (sem sinal)
/	x/y	$x\div y$ (sem sinal)
MOD	$x \text{ MOD } y$	resto de $x\div y$ (sem sinal)
SHL	$x \text{ SHL } y$	x deslocado à esquerda y vezes
SHR	$x \text{ SHR } y$	x deslocado à direita y vezes
HIGH	HIGH x	MSB de x
LOW	LOW x	LSB de x
.	data.bit	endereço do bit do byte data
NOT	NOT x	negação de x (complemento 1)
AND	$x \text{ AND } y$	AND lógico bit a bit (bitwise)
OR	$x \text{ OR } y$	OR lógico bit a bit (bitwise)
XOR	$x \text{ XOR } y$	XOR lógico bit a bit (bitwise)
LT	$x \text{ LT } y$	1 se $x < y$, senão 0
LE	$x \text{ LE } y$	1 se $x \leq y$, senão 0
EQ	$x \text{ EQ } y$	1 se $x = y$, senão 0
NE	$x \text{ NE } y$	1 se $x \neq y$, senão 0
GE	$x \text{ GE } y$	1 se $x \geq y$, senão 0
GT	$x \text{ GT } y$	1 se $x > y$, senão 0

Figura 5.1. Lista com os operadores aritméticos.

5.2. PSEUDO-INSTRUÇÕES DO ASSEMBLER

Sempre que o assembler é chamado, este primeiro envia o arquivo fonte para o pré-processador. Pode-se evitar este passo com o seguinte comando (se não houver instruções para o pré-processador):

```
AVMAC51   ARQ.ASM   NOMACEVAL
```

Normalmente definir TRUE = 0FFFFH e FALSE = 0 evita problemas com relações.

END → deve estar presente em todo arquivo, se faltar, será gerado um.

EQU → atribui valores numéricos aos símbolos (os valores atribuídos são permanentes).

TEQ → atribui valores numéricos aos símbolos (os valores atribuídos são temporários).

OBSERVAÇÃO: é opcional a definição do tipo para os símbolos, os quais são:

CODE → memória de programa,
DATA → memória de dados interna,
XDATA → memória de dados externa,
BIT → endereços do espaço de bits,
NUMBER → qualquer coisa, é válido em qualquer lugar.

Exemplos: LB EQU 5, CODE → LB é um endereço de programa
 FLAG EQU P1.3,BIT → FLAG é um endereço de bit

DS → (Define Space) reserva espaço na RAM. Dentro do segmento de bits, DS reserva espaço em bits. Ex: DS 8 → reserva 8 bytes da RAM no segmento onde estiver, geralmente CODE.

DB → (Define Byte) reserva e inicializa espaço

Exemplos:

TXT DB "Isto é uma string", 0DH,0AH,0
 TXT DB "Isto é uma string\r\n\0"
 L1 DB 8 ;reserva 1 byte e o inicializa com 8
 L2 DB 1, 2, 3 ;reserva 3 bytes e os inicializa com estes valores

SÍMBOLO	SIGNIFICADO
\n	nova linha
\r	retorno de carro
\t	tabulação
\0	NULL ou 0
\x	precede um hexadecimal de 2 dígitos (\x0A=line feed)

DW → (Define Word) é como DB, mas trabalha com words (16 bits). Ex: L3 DW 123H

PROC ,ENDPROC → labels globais, permitem programas modulares.

Símbolo local → começa com "L?XXXX" e só tem sentido entre PROC.....ENPROC, quando se encontra ENPROC, todos os símbolos locais são destruídos.

Exemplo:

```

BLOCO1 PROC
        MOV    A,#7
L?TOP:  CPL    P1.7        ;L?TOP é uma variável local
        DJNZ   A,L?TOP
        ENDPROC
  
```

INCLUDE → igual à linguagem C,

"ARQ" " " → busca arquivo no diretório corrente

<ARQ> < > → busca arquivo no diretório indicado pela variável MS/DOS "INCLUDE"

SEG → existem 4 segmentos já definidos **CODE**, **DATA**, **XDATA** e **BIT**. Se não foi indicado nada, usa-se o segmento CODE. A pseudo-instrução SEG permite trocar de segmentos

Exemplo:

	MOV	B,#20	;Depois da montagem e linkagem
	SEG	DATA	;não haverá nada entre as instruções
MARCA	DW	45H	;MOV B,#20 e INC A. O segmento
	SEG	CODE	;DATA será colocado em
	INC	A	;outra parte

DEFSEG → permite a definição de outros segmentos, tais como: RAMDATA, ROMDATA, PILHA, etc.

Na definição de um segmento pode-se informar:

ABSOLUT → não passa pela linkagem, deve ser dado o endereço inicial.

RELOCABLE → o linker define seu endereço.

ALIGNMENT → indica o tamanho do bloco para alinhamento; se for pedido um alinhamento de 100H, o linker tentará colocar o segmento em um endereço múltiplo de 100H.

BLOCK → indica ao linker que o segmento deve residir em uma determinada área. Se BLOCK=PAGE então o segmento deve estar dentro de uma página de 2 KB.

OVERLAID → indica ao linker que o mesmo segmento em módulos diferentes deve estar no mesmo endereço.

CLASS → define a classe do segmento; existem 4 classes: CODE, DATA, XDATA, BIT

Exemplos:

DEFSEG STACK, CLASS=DATA	;stack é um segmento redirecionável
DEFSEG KUMQUATS	;KUMQUATS é CODE por default
DEFSEG BLOCK, START=200H	;BLOCK é segmento CODE, iniciando em 200H.
DEFSEG TRW, ALIGN=PAGE	;TRW é segmento tipo CODE e deve começar ;com endereço múltiplo de 800H (dentro de páginas ; de 2k)
DEFSEG NEWIO, START = 50H, CLASS=BIT	;NEWIO é um segmento na área de ; bits, começando em 50H.

OBSERVAÇÃO: DEFSEG só define o segmento; para entrar em um segmento qualquer é necessário usar a instrução SEG.

ORG → origem.

PUBLIC, **EXTERN** → permite compartilhar símbolos definidos em arquivos (módulos) distintos. Ex: PUBLIC LABEL e EXTERN LABEL_EXT

O PRÉ-PROCESSADOR (ADP)

O pré-processador é chamado ADP. O assembler necessita encontrá-lo pois é a primeira ação a ser realizada. Todas as pseudo-operações relativas ao pré-processador começam com o carácter %.

%IF - %ELSE - %ELSEIF - %ENDIF → sem comentários

%SWITCH - %CASE - %DEFAULT - %ENDSW → idêntico à linguagem C.

Ex: %SWITCH PARAM
 %CASE 1
 ...
 %CASE 9

 %DEFAULT
 ...
 %ENDSW

%REPT - %ENDREPT → repetir uma seção do programa.

Ex: %REPT 5
 ...
 %ENDREPT

%FOR - %ENDFOR → sem comentários.

Ex: %FOR I = 1 TO 10
 ...
 %ENDFOR

%MACRO → permite definir trechos de programa e representá-los por um símbolo.

Exemplo:

Definir um macro que represente 3 rotações para a esquerda:

```
RL3       %MACRO               ;exemplo de macro sem parametros
          RL       A
          RL       A
          RL       A
          %ENDM
```

Utilização desta macro:

```
ADD       A,#5
RL3
```

```
SUBB    A,B
```

Definição de uma macro para representar a soma de três variáveis:

```
SOMA    %MACRO      x,y,z      ;exemplo de macro com parametros
        MOV          A,z
        ADD          A,y
        MOV          x,A
        %ENDM
```

Utilização da macro recém definida:

```
MOV      B,A
SOMA     R7,R6,R5
MOV      R3,A
```

%IFB - %IFNB → (If Blank - If Not Blank) se foi atribuído ou não valor a um parâmetro (blank significa que não foi atribuído nenhum valor); geralmente usado nas macros.

%IFEQ - %IFNEQ → compara dois parâmetros.

%GENSYM → gerar símbolos automaticamente

Exemplo: A utilização do FOR pode resultar em repetição de um label:

```
% FOR      I          IN "P1.7","P1.6","P1.5"
AQUI       JNB        I,AQUI
% ENDFOR
```

Vai resultar em:

```
AQUI    JNB        P1.7,AQUI
AQUI    JNB        P1.6,AQUI
AQUI    JNB        P1.5,AQUI
```

O label AQUI foi definido três vezes, o que é um erro. A solução é usar GENSYM:

```
% FOR      I          IN "P1.7","P1.6","P1.5"
% GENSYM   AQUI
AQUI       JNB        I,AQUI
% ENDFOR
```

Vai resultar em:

```
??0000    JNB        P1.7,??0000
??0001    JNB        P1.6,??0001
??0002    JNB        P1.5,??0002
```

Notar que se evitou a repetição do mesmo label.

OPÇÕES DO ASSEMBLER AVMAC51

As opções podem ser colocadas na linha de comando quando o assembler é invocado ou em um arquivo fonte, usando a linha chamada "linha de opção". Ela deve iniciar na primeira coluna.

\$option opções,....

OBSERVAÇÕES:

- Duas instruções genéricas JMP e CALL são fornecidas. O assembler decide qual instrução usar em função da distância, de forma que obtenha o menor tamanho. Isso é válido para referências para trás. Em referências para adiante é obrigatório o uso de LJMP, AJMP ou LCALL, ACALL.
- Todos os nomes dos SFR e dos bits já estão definidos.
- Também já estão definidos os seguintes endereços de interrupções

RESET	→ 0	;endereço a ser executado depois do RESET
EXTI0	→ 3	;interrupção EXTERNA 0
TIMER0	→ 08H	;interrupção do TIMER 0
EXTI1	→ 13H	;interrupção EXTERNA 1
TIMER1	→ 1BH	;interrupção do TIMER 1
SINT	→ 23H	;interrupção da PORTA SERIAL

- Na listagem do arquivo com extensão .PRN gerado, o campo de endereços apresentará os seguintes símbolos:

&	→ segmento definido pelo usuário	@	→ segmento PAGE0 (se houver)
'	→ segmento CODE	*	→ label externo
"	→ segmento DATA		

5.3. O LINKER - AVLINK

O AVLINK é um sistema interativo para linkar (ligar) objetos. Este programa é invocado usando:

AVLINK ARQ = ARQ START (100H)

A partir do ARQ.OBJ é gerado o ARQ.HEX, onde o programa começará a partir do endereço absoluto 100H.

Outros arquivos podem ser gerados:

ARQ.SYM → arquivo de símbolos.

ARQ.MAP → indica onde cada parte do programa está localizada.

AVLINK → invoca linker em modo interativo

AVLINK @ARQ → lê arquivo e aguarda comandos

OPÇÕES (listadas com a opção **-h**) :

MAP = mapfile → redireciona o arquivo .MAP (MAP = COM). (MA=mapfile)

NoMAP → não gera .MAP. (-NM)

ORDER (seg nombre, seg nombre...) → força que os segmentos sejam carregados na sequência especificada. (-OC seg1, seg2, ...)

HEX = arquivo imagem → redireciona a saída .HEX. (HX = filename)

NoOut → não gera .HEX. (-NO)

Out Format = format_type → formato do arquivo absoluto: (OF= mot | mot0 | tek)

HEX → Intel HEX.

TEK → Tektronix.

MOT → Motorola.

SYmbols = symfile → redireciona a saída .SYM. (SY=filename)

NoSymbols → não gera arquivo .SYM (default). (-NS)

SYmbols → gera arquivo .SYM. (-SY)

PlainDump → tabela de símbolos. (-PD)

QUIet → não gera mensagens, incluindo os warnings. (-QU)

STart (clase,endereço) → endereço de início para cada classe.

ToP (clase,endereço) → endereço limite para cada classe.

RecLen= tamanho → tamanho do record nos arquivos .HEX (default = 32). (-RL=tam)

PutSeg (Segname, endereço) → realoca o segmento especificado em Segname. (-PS(seg,addr))

DefSym (symname, endereço) → define um símbolo público. (-DS(name,val))

MaxLen (segname, tamanho) → tamanho máximo para um segmento. (-ML(seg,len))

ShowsMods → coloca no arquivo .MAP a informação de cada arquivo .OBJ. (-SM)

ShowPublics → mostra no .MAP os labels públicos de cada arquivo. (-SP)

SortbyName → ordena a lista gerada por ShowPublics. (-SN)

RumsAt(segname, addr) → ?.

HEXFORM → é um programa que ordena os arquivos .HEX de forma que os endereços mais baixos apareçam primeiro. Modelo para a utilização:

HEXFORM	ARQ_IN.HEX	ARQ_OUT.HEX
	(não ordenado)	(ordenado)

Serve também para converter entre .HEX e o formato binário com a sintaxe: hexform ARQ=ARQ

AVLIB → permite a criação de bibliotecas. Modelo para a utilização:

AVLIB	ART.LIB = ARQ1.OBJ, ARQ2.OBJ
-------	------------------------------

Se um programa que usa a biblioteca ART.LIB, mas chama somente ARQ1, então ARQ2 não será linkado.

AVREF → gera a referência cruzada. Modelo para a utilização:

AVREF TESTE = TESTE.SYM, ARQ1.XRF, ARQ2.XRF, ARQ3.XRF

(Os arquivos ARQ1.XRF, ARQ2.XRF, ARQ3.XRF foram gerados por AVLINK).

(O arquivo TESTE.HEX foi produzido pelos arquivos ARQ1.OBJ, ARQ2.OBJ, ARQ3.OBJ).

5.4. O FORMATO INTEL.HEX

O formato INTEL.HEX representa informações binárias através de um texto ASCII imprimível. Ele é organizado em uma série de records. Dentro de cada record o dado binário é representado por dígitos (hexadecimal) ASCII, 2 dígitos para cada byte.

Cada record tem um endereço de carregamento a partir do qual os bytes devem ser carregados. Para cada record também há um checksum de 8 bits que é um complemento a 2 da soma dos bytes de dados, contador, tipo de record e endereço. Notar que os valores binários dos bytes, e não os códigos ASCII, são usados no cálculo do checksum.

O checksum obtido somado ao que está no record deve resultar em zero.

RECORD DE DADOS

:	COUNT 2 char	END. CARGA 4 char	TIPO 00	DADOS (2*count) char	CHECKSUM 2 char	CR	LF
---	-----------------	----------------------	------------	-------------------------	--------------------	----	----

RECORD FINAL

:	COUNT 00	END. INÍCIO 4 char	TIPO 01	CHECKSUM 2 char	CR	LF
---	-------------	-----------------------	------------	--------------------	----	----

CR - Carriage Return LF - Line Feed

Figura 5.4. Descrição do formato INTEL.HEX.

Exemplo: Record com os dados 1,2,3,4,5 colocados a partir do endereço 1234H e com endereço de início em 0000H.

: 05 1234 00 01 02 03 04 05 A6

: 00 0000 01 FF

(foram deixados espaços para clareza)

Checksum para os records:

$$05 + 12H + 34H + 00 + 01 + 02 + 03 + 04 + 05 = 5AH \rightarrow 0 - 5AH = A6$$

$$00 + 00 + 00 + 01 = 01 \rightarrow 0 - 01 = 0FFH$$

5.5. PROGRAMAS EXEMPLO

A seguir são apresentados três programas fonte (.ASM) com a listagem (.PRN) gerada pelo assembler. Por simplicidade, todos os programas iniciam no endereço 0.

PROGRAMA FONTE EXEMPLO 1:

```
;EXEMPLO1.ASM
;
;Somar dois numeros BCD de 4 digitos : R1R2R3 = R4R5 + R6R7
```

	SEG	CODE	
INICIO	MOV	A,R7	; R7 + R5
	ADD	A,R5	
	DA	A	
	MOV	R3,A	; R3 = R7 + R5
	;		
	MOV	A,R6	; R6 + R4 + Carry
	ADDC	A,R4	
	DA	A	
	MOV	R2,A	; R2 = R6 + R4 + Carry
	CLR	A	
	ADDC	A,#0	; usar o Carry
	MOV	R1,A	; R1 = Carry
	END		

LISTAGEM DO ARQUIVO EXEMPLO1.PRN GERADO PELO ASSEMBLER:

	1	;EXEMPLO1.ASM		
	2	;		
	3	;Somar dois numeros BCD de 4 digitos : R1R2R3 = R4R5 + R6R7		
	4			
	5	SEG	CODE	
0000' EF	6	INICIO	MOV	A,R7 ; R7 + R5
0001' 2D	7		ADD	A,R5
0002' D4	8		DA	A
0003' FB	9		MOV	R3,A ; R3 = R7 + R5
	10		;	
0004' EE	11		MOV	A,R6 ; R6 + R4 + Carry
0005' 3C	12		ADDC	A,R4
0006' D4	13		DA	A
0007' FA	14		MOV	R2,A ; R2 = R6 + R4 + Carry
0008' E4	15		CLR	A
0009' 34 00	16		ADDC	A,#0 ; usar o Carry
000B' F9	17		MOV	R1,A ; R1 = Carry
	18		END	

PROGRAMA FONTE EXEMPLO 2:

```

; EXEMPLO2.ASM
;
; Converter para BCD um numero binario entre 0 e 999
; R7R6(BCD) ← R5R4(Bin)
; Esta rotina esta no exemplo E10 caso c

BB39      SEG      CODE
          CLR      A
          MOV      R7,A      ;ZERAR R7 E R6
          MOV      R6,A

          MOV      A,R5
          JZ       LB1      ;SE R5=0, SALTAR ESSA FASE
LB2        MOV      A,R6      ;SOMAR 256 BCD TANTAS VEZES
          ADD      A,#56H     ;QUANTO EH O VALOR DE R5
          DA       A
          MOV      R6,A
          MOV      A,R7
          ADDC     A,#2
          DA       A
          MOV      R7,A
          DJNZ     R5,LB2     ;ZERO R5

LB1        MOV      A,R5      ;CONVERTE R4 PARA BCD
          MOV      B,#100
          DIV      AB         ;SEPARA CENTENAS
          ADD      A,R7
          DA       A
          MOV      R7,A
          MOV      A,B
          MOV      B,#10
          DIV      AB         ;SEPARA DEZENAS
          SWAP     A
          ADD      A,B
          ADD      A,R6      ;SOMA COM O QUE EXISTE
          DA       A
          MOV      R6,A
          MOV      A,R7
          ADDC     A,#0
          DA       A
          MOV      R7,A
          SJMP     $
          END

```

LISTAGEM DO ARQUIVO EXEMPLO2.PRN GERADO PELO ASSEMBLER:

```

1      ; EXEMPLO2.ASM
2
3      ; Converter para BCD um numero binario entre 0 e 999
4      ; R7R6(BCD) <-- R5R4(Bin)
5      ; Esta rotina está no exemplo E10 C
6
7      SEG      CODE
0000' E4      8      BB39      CLR      A
0001' FF      9              MOV      R7,A      ;ZERAR R7 E R6
0002' FE      10             MOV      R6,A
11
0003' ED      12             MOV      A,R5
0004' 60 0C    13             JZ       LB1      ;SE R5=0, PULAR ESSA FASE
0006' EE      14      LB2      MOV      A,R6      ;SOMAR 256 BCD TANTAS VEZES
0007' 24 56    15             ADD      A,#56H    ;QUANTO EH O VALOR DE R5
0009' D4      16             DA       A
000A' FE      17             MOV      R6,A
000B' EF      18             MOV      A,R7
000C' 34 02    19             ADDC     A,#2
000E' D4      20             DA       A
000F' FF      21             MOV      R7,A
0010' DD F4    22             DJNZ     R5,LB2    ;ZERO R5
23
0012' EC      24      LB1      MOV      A,R5      ;CONVERTE R4 PARA BCD
0013' 75 F0 64 25      MOV      B,#100
0016' 84      26             DIV      AB      ;SEPARA CENTENAS
0017' 2F      27             ADD      A,R7
0018' D4      28             DA       A
0019' FF      29             MOV      R7,A
001A' E5 F0    30             MOV      A,B
001C' 75 F0 0A 31      MOV      B,#10
001F' 84      32             DIV      AB      ;SEPARA DEZENAS
0020' C4      33             SWAP     A
0021' 25 F0    34             ADD      A,B
0023' 2E      35             ADD      A,R6      ;SOMA COM O QUE EXISTE
0024' D4      36             DA       A
0025' FE      37             MOV      R6,A
0026' EF      38             MOV      A,R7
0027' 34 00    39             ADDC     A,#0
0029' D4      40             DA       A
002A' FF      41             MOV      R7,A
002B' 80 FE    42             SJMP     $      ;LOOP ETERNO
43      END

```


PROGRAMA FONTE EXEMPLO 3:

;EXEMPLO3.ASM

;Converter um byte nos 2 caracteres HEXA ASCII correspondentes

; AB (HEXA ASCII) ← A

;Exemplo: se Acc=0110 1010 → Acc=36H e B=41H

;Observacao: ASCII(6)=36H e ASCII(A)=41H

;Corresponde ao exemplo E12 do capitulo 4

;Para ilustrar este programa sera criado o segmento "PROGRAMA"

;do tipo CODE, realocavel.

	DEFSEG	PROGRAMA, CLASS=CODE	;CRIAR SEGMENTO
	SEG	PROGRAMA	;ENTRAR NO SEGMENTO
HEXASC	MOV	B,A	;GUARDAR UM BYTE
	ANL	A,#0FH	;SEPARAR LSNIBBLE
	ACALL	NIBASC	;CONVERTER NIBBLE → ASCII
	XCH	A,B	;RECUPERAR UM BYTE
	ANL	A,#0FOH	;SEPARAR MSNIBBLE
	SWAP	A	
	ACALL	NIBASC	
	SJMP	\$;PARAR EXECUÇÃO

;Rotina para converter uma nibble no ASCII correspondente

NIBASC	PUSH	ACC	;GUARDAR NIBBLE
	CLR	C	
	SUBB	A, #10	;A-10 → CY=1 ==> A<10
	POP	ACC	; → CY=0 ==> A<=10
	JNC	NIBASC1	
	ADD	A, #30H	;30H ... 39H
	RET		
NIBASC1	ADD	A, #37H	;44H ... 46H
	RET		
	END		

LISTAGEM DO ARQUIVO EXEMPLO3.PRN GERADO PELO ASSEMBLER:

```

1  ;EXEMPLO3.ASM
2
3  ;Converter um byte nos 2 caracteres HEXA ASCII correspondentes
4  ;AB (HEXA ASCII) <-- A
5  ;Exemplo: se Acc=0110 1010 → Acc=36H e B=41H
6  ;Observacao: ASCII(6)=36H e ASCII(A)=41H
7  ;Corresponde ao exemplo E12
8
9  ;Para ilustrar este programa sera criado o segmento "PROGRAMA",
10 ;do tipo CODE, realocavel.
11
12          DEFSEG          PROGRAMA, CLASS=CODE      ;CRIAR
SEGMENTO
13          SEG             PROGRAMA ;ENTRAR EM SEGMENTO
0000& F5 F0 14  HEXASC     MOV             B,A          ;GUARDAR UM BYTE
0002& 54 0F 15              ANL             A,#0FH      ;SEPARAR LSNIBBLE
0004& ....X 16              ACALL          NIBASC       ;CONVERTER NIBBLE → ASCII
0006& C5 F0 17              XCH             A,B          ;RECUPERAR UM BYTE
0008& 54 F0 18              ANL             A,#0F0H     ;SEPARAR MSNIBBLE
000A& C4      19              SWAP          A
000B& ....X 20              ACALL          NIBASC
000D& 80 FE 21              SJMP           $             ;PARAR EXECUÇÃO
22
23 ;Rotina para converter uma nibble em ASCII correspondente
000F& C0 E0 24  NIBASC     PUSH          ACC          ;GUARDAR NIBBLE
0011& C3      25              CLR             C
0012& 94 0A 26              SUBB          A,#10        ;A-10 → CY=1 ==> A<10
0014& D0 E0 27              POP             ACC        ;      → CY=0 ==> A<=10
0016& 50 03 28              JNC            NIBASC1
0018& 24 30 29              ADD             A,#30H     ;30H ... 39H
001A& 22      30              RET
001B& 24 37 31  NIBASC1    ADD             A,#37H     ;41H ... 46H
001D& 22      32              RET
23              END

```

5.6. CONCEITOS DO AVSIM 8051

O AVSIM51 é feito para ser usado com o AVMAC51. Este simulador oferece muitos recursos:

- Uma "CPU visual" na tela,
- Todos os registros, flags, stack, portas e memória,
- Desassembler (Inverse assembler),
- Single step,
- Undo (back trace),
- Rodar todo o programa,
- Simular I/O através de arquivos,
- Pode-se aplicar patches diretamente usando mnemônicos,
- Contador de ciclos para medir tempo de execução,
- Breakpoints.

Chama-se o programa com: AVSIM51 [keystrokes]. A versão 1.0 suporta apenas um argumento.

AVSIM51 <CPU tipo> FL <command file_name>

Para que o AVSIM51 rode eficientemente devem estar presentes três arquivos:

- AVSIM51.OVR
- AVSIM51.REG
- AVSIM51.HLP

Se não foi especificado na linha de comando, o AVSIM51, ao ser invocado, pode especificar o tipo de CPU.

Existem dois modos de operação (troca-se de modo usando **ESC**):

- Command mode → Executar comando
- Display mode → Acesso dentro da CPU

5.6.1. Modo Comando

Neste modo o cursor sempre está em uma linha de comando - próximo na parte inferior do monitor. Dá assim a possibilidade de executar todos os comandos.

O modo comando é feito por menus. Se o menu é muito extenso, usa-se a barra de espaço para mostrar o resto. Os comandos são selecionados por:

- barra highlighted
- uma letra.

Alguns comandos pedem uma linha para complementá-lo. A sequência **CRTL + C** é usada para abortar qualquer comando ou menu e retornar ao menu principal.

5.6.2. Modo Display

Neste modo o cursor está em um dos campos da CPU que aparece na tela. Tudo o que pode ser acessado pelo cursor está em highlight.

Cada campo na janela pertence a um dos tipos:

- HEX
- ASCII
- BINARY
- CODE

5.7. COMANDOS DO AVSIM51

Antes de iniciar o estudo dos comandos, é necessário uma série de observações.

Sempre que um valor é necessário pode-se usar uma expressão aritmética. Uma expressão é composta por uma ou mais constantes conectadas por operadores.

uma expressão:	20H - (BF_BASE-1)
múltiplas expressões:	MAIN , "AB", \$-1, '0a'
sintaxe de endereço:	[adr space:] expressão → D:BF_BASE

Os valores numéricos podem estar na base 2, 8, 16 ou 10.

@ → indica base 8 quando usado com os mnemônicos

@ → em expressões indica o operador indireto.

Há diversos outros operadores:

- . → um ponto é um símbolo onde está o resultado da última expressão
' +100' → expressão anterior +100.
- + → soma.
- → subtração ou negação.
- @ → operador indireto.
- () → até 4 níveis.

Quando um endereço é solicitado, o espaço de endereçamento pode ser explicitamente especificado como:

Code	C	Ex: C: Main
Data	D	Ex: D: BUFFER
External	X	Ex: X: 0A

5.7.1. Comandos de Ambiente:

-Carregar arquivos:

Load Prog	→	atribui memória e carrega o programa
Load Data	→	atribui memória e carrega os dados
Load symbols	→	carrega a tabela de .símbolos .do Cross.Assembler
Load Avocet	→	Load Symbols (.PRN)
Load Avocet	→	Load Program (.HEX ou .MIK)

-Arquivo de comando: evita a repetição dos comandos freqüentes.

Load	→	executa arquivo de comando.
Open	→	abrir arquivo de comando
Close	→	fechar arquivo de comando
Restart	→	reiniciar arquivo de comando

-Atribuir memória: Set memory_map: → atribui espaço de dados externo.
(como ROM ou RAM e não pode ser liberada)

5.7.2. Comandos para a Execução de Programas:

5.7.2.1. Breakpoints:

Existem diversos tipos de breakpoints:

- De acordo com o acesso: read/write
write only
- De acordo com o endereço: endereço de memória
faixa de endereço
registrador
- De acordo com a duração: sticky → somente é desativado pelo usuário
dynamic → se auto destrói quando ocorre

O número de breakpoints que podem estar ativos simultaneamente depende somente da memória disponível no PC.

Dois breakpoints tipo "sticky" são instalados automaticamente:

- todo endereço de memória não definido
- toda faixa de memória definida como ROM tem o tipo "write only break point".

Breakpoints incondicionais: sempre interrompem a execução quando há acesso a um endereço.

Set sticky_bkpt	→	só é desativado pelo usuário.
Set dynamic_bkpt	→	se auto apaga quando ocorre.

Acesso: read/write ou write only

Endereço: valor ou faixa de valores

Para retardar uma interrupção até n passos a partir do breakpoint, entre com um valor decimal antes da seleção de acesso/endereço.

Breakpoint dinâmicos incondicionais podem ser ativadas usando "break point cursor keys" (F2, F3, F4).

Quando um breakpoint ocorre, a execução do programa se detém antes da instrução que ativa o breakpoint.

- Se é dinâmico, ele é apagado e a execução do programa pode prosseguir.
- Se é do tipo "sticky" é necessário usar "single step".

Os comandos que operam com os breakpoints são:

- View breakpoints.
- Reset breakpoints.
- Reset all.
- Reset trap list.

5.7.2.2. Condições (Value, Range, Mask, Indirect)

- VALUE → deve ser fornecido um valor de 8 bits o qual será comparado com o conteúdo, endereço ou registro sujeito a breakpoint. Exemplo: interrompe quando o conteúdo de R0 seja 0F0H.
- RANGE → deve-se definir um limite inferior e um superior; quando o conteúdo estiver nesta faixa (inclusive) o breakpoint é ativado.
- MASK → máscara de 8 bits com 0, 1 e X (don't care). Quando os bits da máscara coincidem com o conteúdo do endereço do registro é produzido um trap. Exemplo: mask = 1XXX XXX0.
- INDIRECT → agora o conteúdo do registro é interpretado como uma endereço e é usado para buscar um dado na memória, o qual é comparado com o valor. Um segundo valor de 16 bits (offset) deve ser fornecido para ser somado com o conteúdo do registro (index) e assim produzir um endereço alvo.

5.7.2.3. Pass Points:

São idênticos aos breakpoints, mas em vez de gerar um trap, eles incrementam um contador de 32 bits.

5.7.2.4. Opcode Traps:

O PC é usado para apontar um dado (um opcode neste caso), que será comparado com um dado especificado. A opção pede um mnemônico completo ou, pelo menos, o primeiro byte é usado na comparação.

Exemplos: JNZ \$ interrompe em toda instrução que tenha JNZ,
 MOV R0,#7 provocar um trap em todos MOV R0, #X.

5.7.2.5. Execute Command:

Permite executar um comando como se fosse um interpretador. Pode ser usado para desviar para qualquer lugar (JMP adr) ou para forçar o retorno de uma subrotina.

5.7.3. Comandos de Display:

DUMP 1 / 2 Absolute → endereço de início da janela que mostra a área de dados. Com o cursor dentro da janela, basta usar <SCROLL UP> ou <SCROLL DOWN>.

DUMP 1 / 2 Indirect → deve ser fornecido uma endereço ou registro e também o valor de offset. O endereço indireto aparece em highlight. Exemplo: usa-se para monitorar transferências com @R1 ou @R0.

5.7.4. Comandos de I/O:

Permite que se unam arquivos aos pinos do microcontrolador e assim se simulem as entradas e saídas. Em um simulador não existem dispositivos externos para interagir com os pinos do microcontrolador. São fornecidos dois recursos para simular atividade externa:

- I/O Interativo → o usuário manualmente gera os sinais nos pinos de interesse.
- I/O por Arquivo → as entradas e saídas vão e vem dos arquivos.

Um arquivo de I/O é uma cadeia de bytes, onde cada bit está conectado a um pino externo do microcontrolador. Também deve-se indicar a velocidade (em ciclos de máquina) que será usada para varrer o arquivo. A recepção de caracteres ASCII pela porta serial é simulada ao unir os 7 bits menos significativos do arquivo de I/O ao SBUF e o bit mais significativo ao bit RI do SCON.

Podem ser gerados arquivos de entrada e arquivos de saída. O comando UNDO não somente retrocede a instrução como também retrocede os arquivos de I/O corretamente.

I/O, OPEN, (Y or N) rewind, IO rate, Mapbit:

I/O rate n → a cada n ciclos de máquina será feita uma transferência.

Mapbit → <I/O bit (0-7)> <memo bit(0-7)>, <endereço> <IN/OUT>:

Exemplo: 7, 3, P1, OUT → irá direcionar o bit P1.3 para o bit 7 do arquivo de saída.

7,RI, IN → o valor do bit RI (do SCON) virá do bit 7 do arquivo de entrada.

View I/O files → mostra todos os arquivos de I/O abertos, a designação dos bits e o valor do "rate counter".

5.7.5. Comandos de Memória:

Set Memory_map → Define memória adicional RAM ou ROM, a qual é inicializada com zeros.

View Memory_map → Ver os espaços de endereços de cada área.

EXPRESSION → Coloca o valor da expressão na posição do cursor.

Memory Clear → Preencher com zeros a área especificada.

Memory Fill → Preencher com um valor a área especificada.

Memory Move → Deslocar áreas de memória, mesmo quando há superposição.

Memory Search → Buscar determinado caracter na memória.

5.7.6. Incremental Cross-Assembler:

Patch code → Permite entrar com os mnemônicos em vez de dados hexadecimais (o endereço usado é o que está no PC). Para alterar um byte de uma instrução pode-se usar a área que está à direita do PC.

Open Output File → Grava todas as instruções que foram digitadas para o patch.

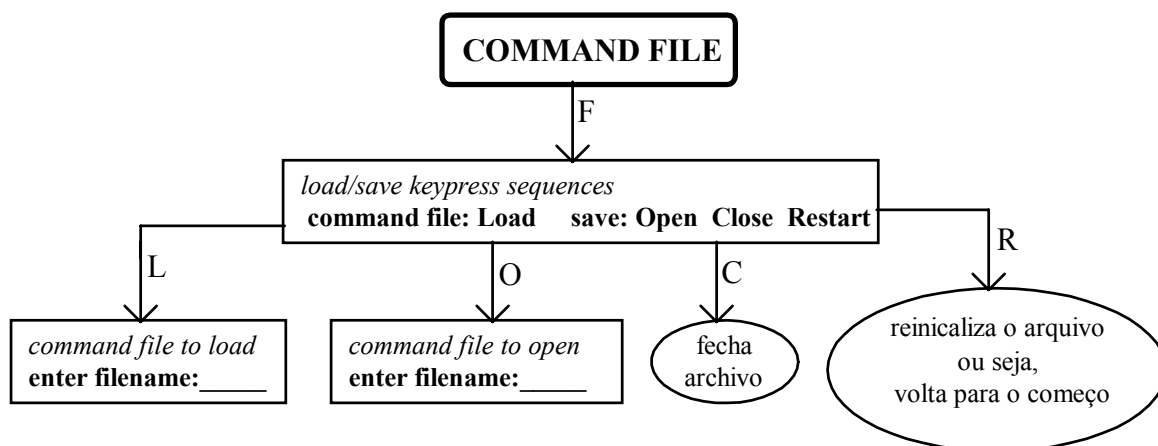
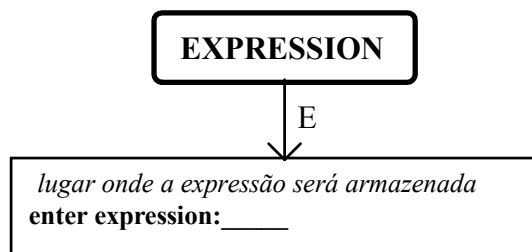
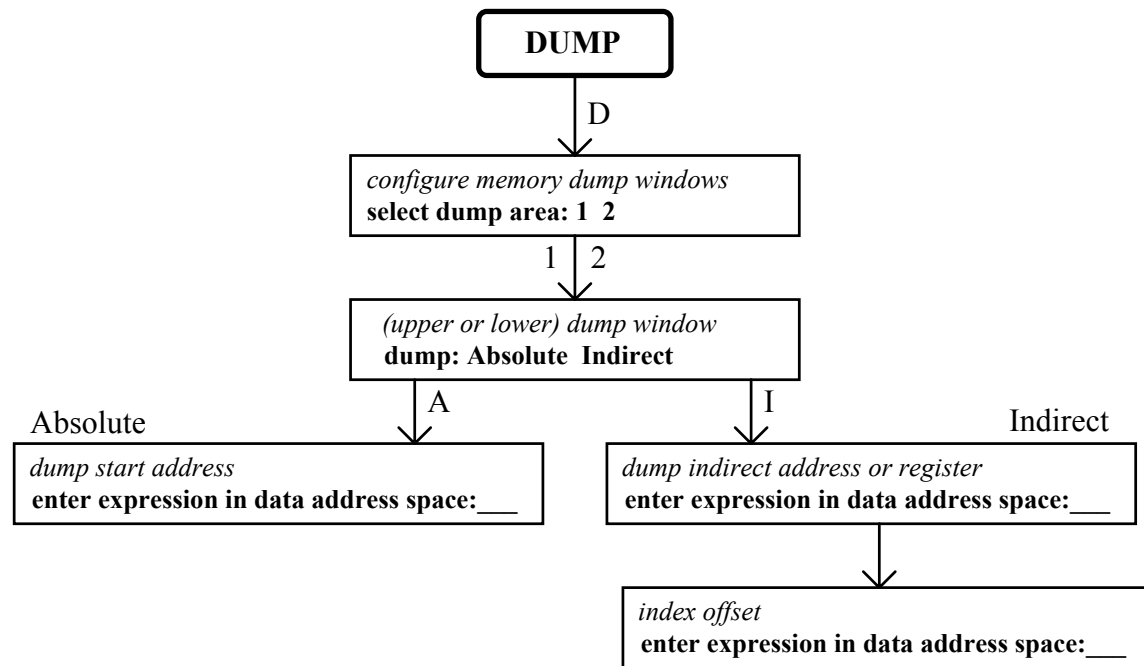
5.8. FLUXOGRAMA DE OPERAÇÃO DO AVSIM51

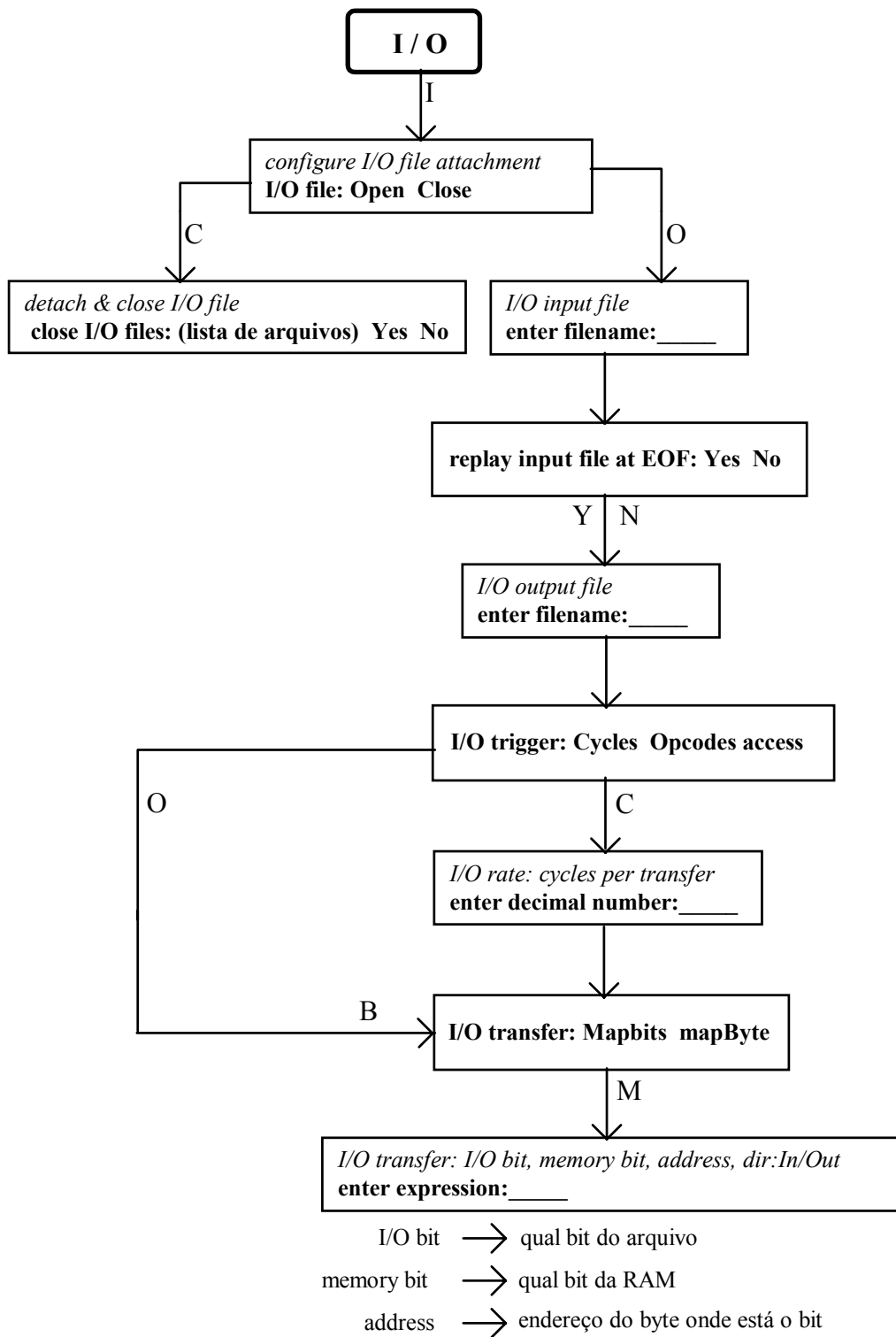
A seguir será apresentado o fluxograma de todas as opções que existem no AVSIM51.

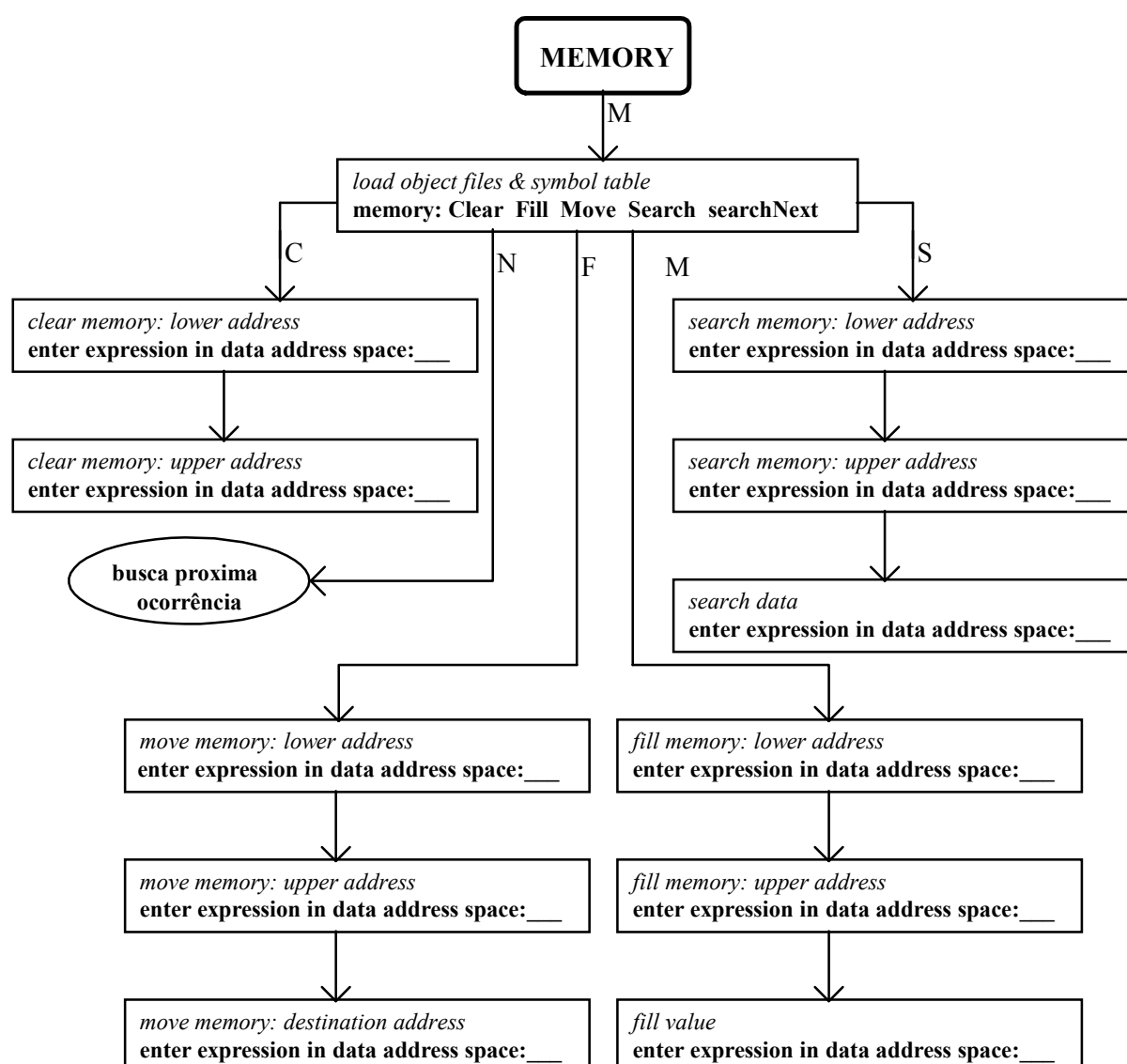
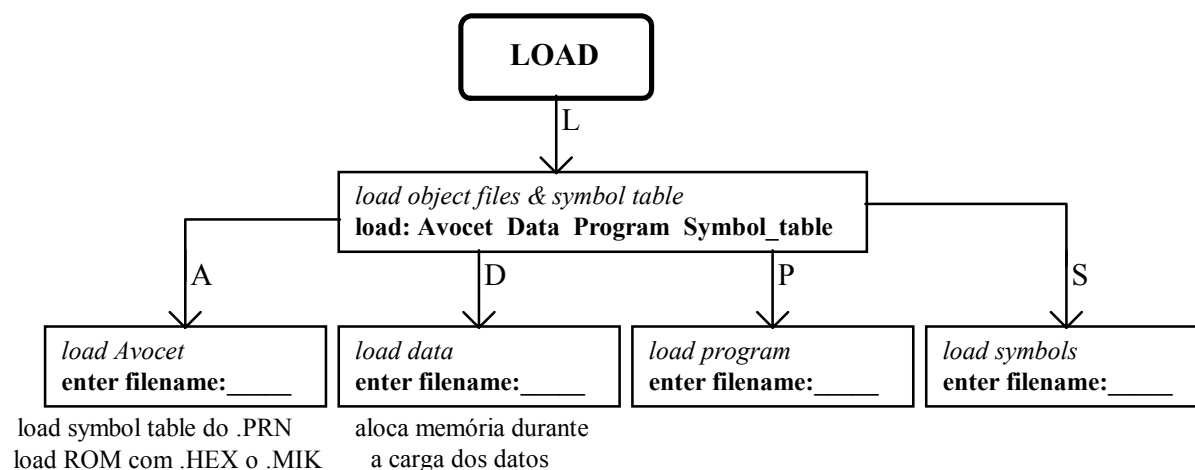
Usar-se-á a seguinte convenção:

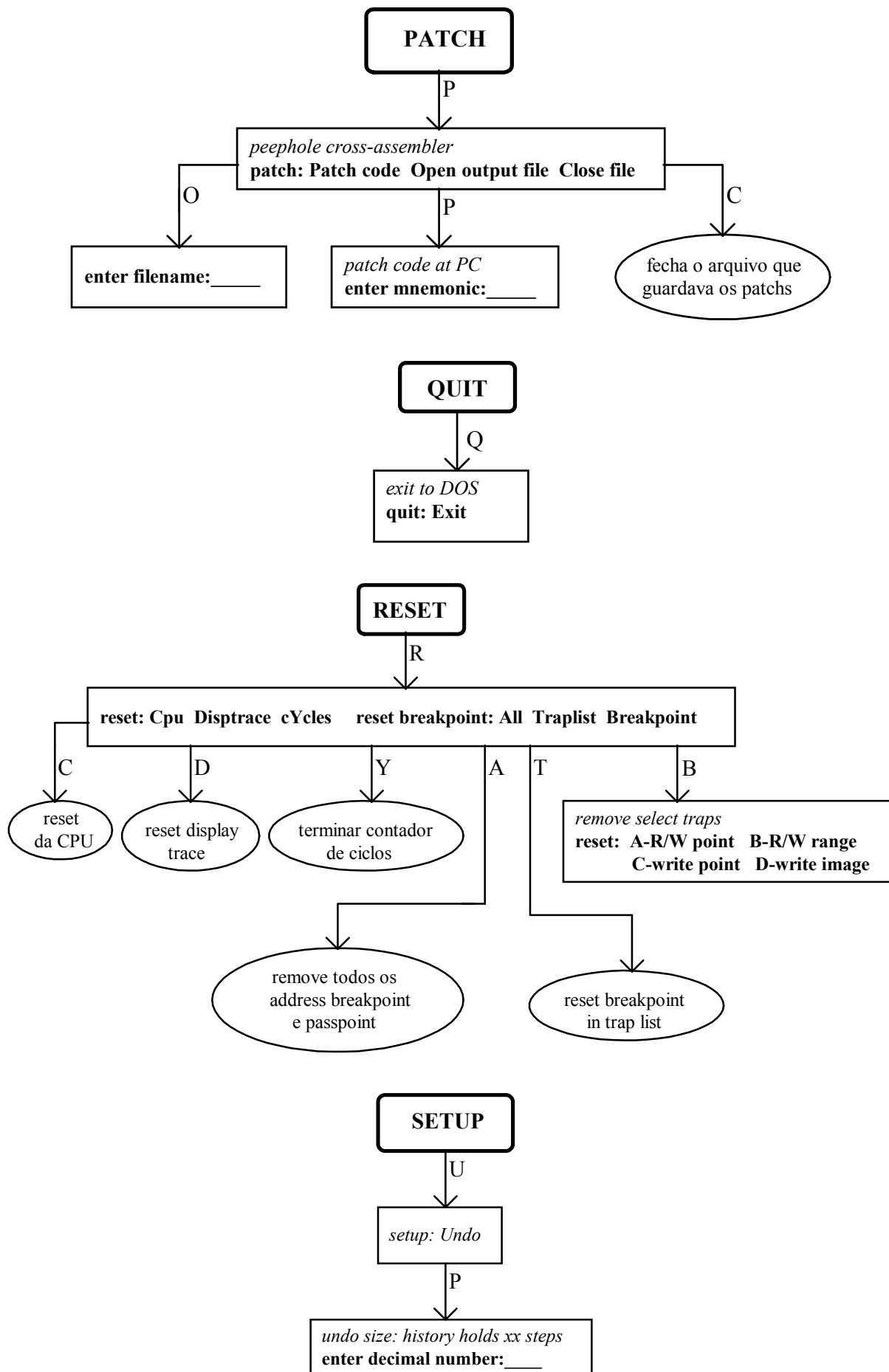
em itálicos → as informações exibidas pelo AVSIM.

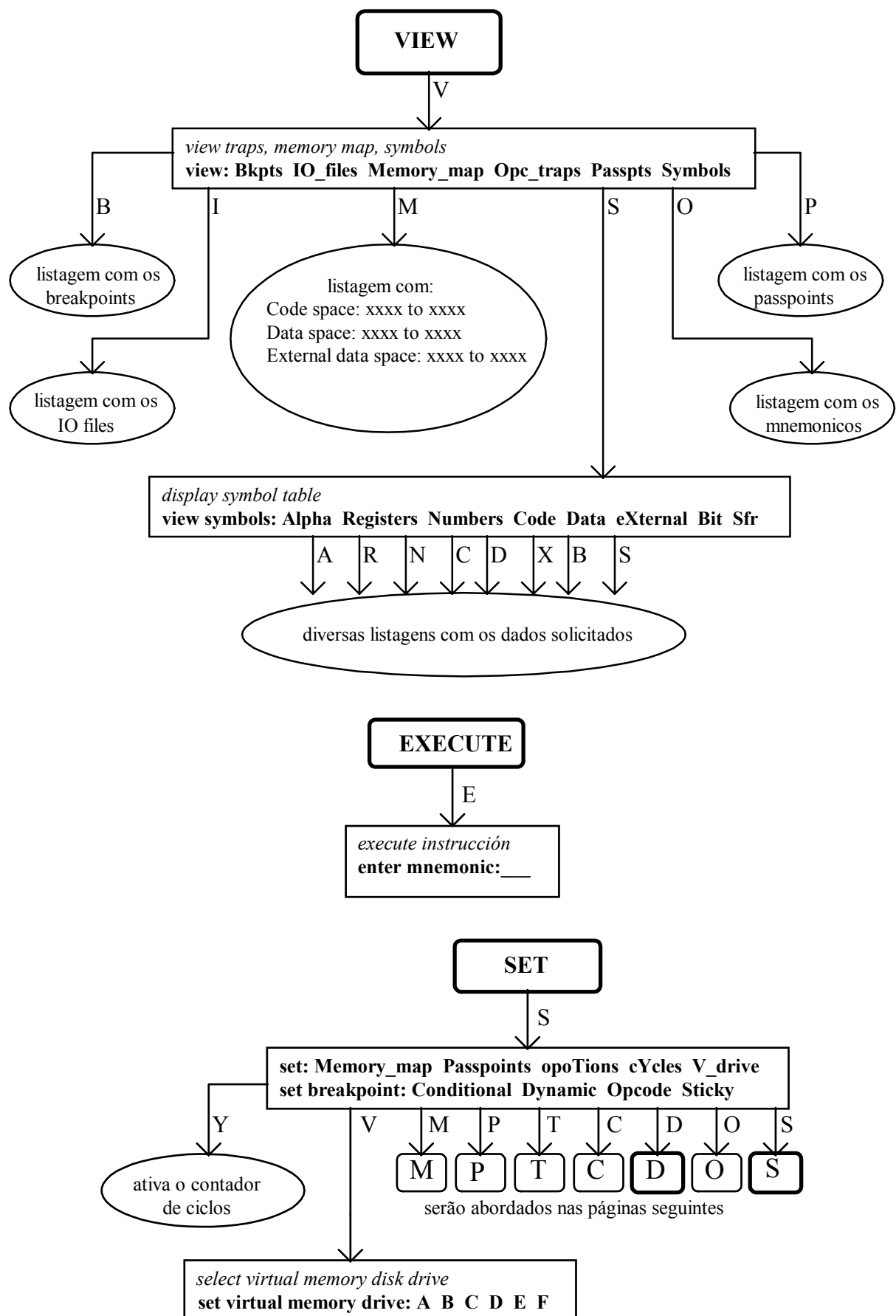
em negrito → as solicitações de entradas, seja através de letra, highlight ou expressão.

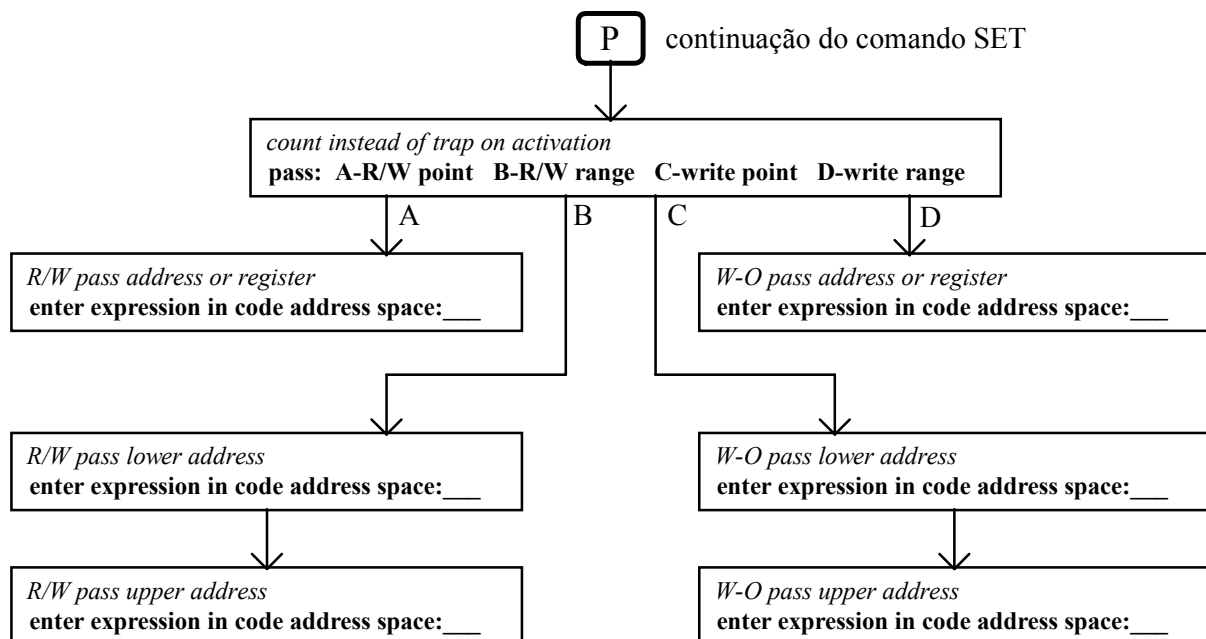
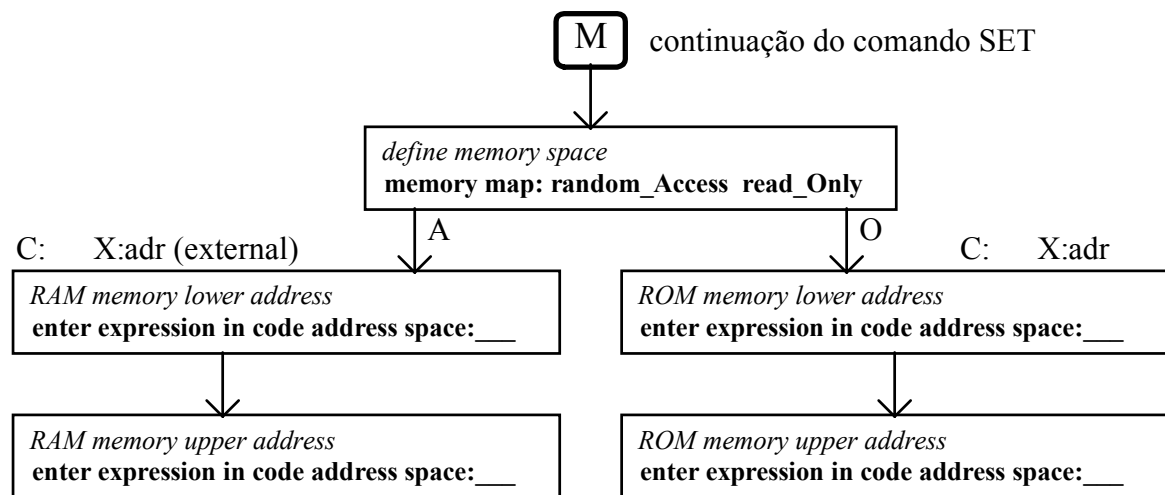


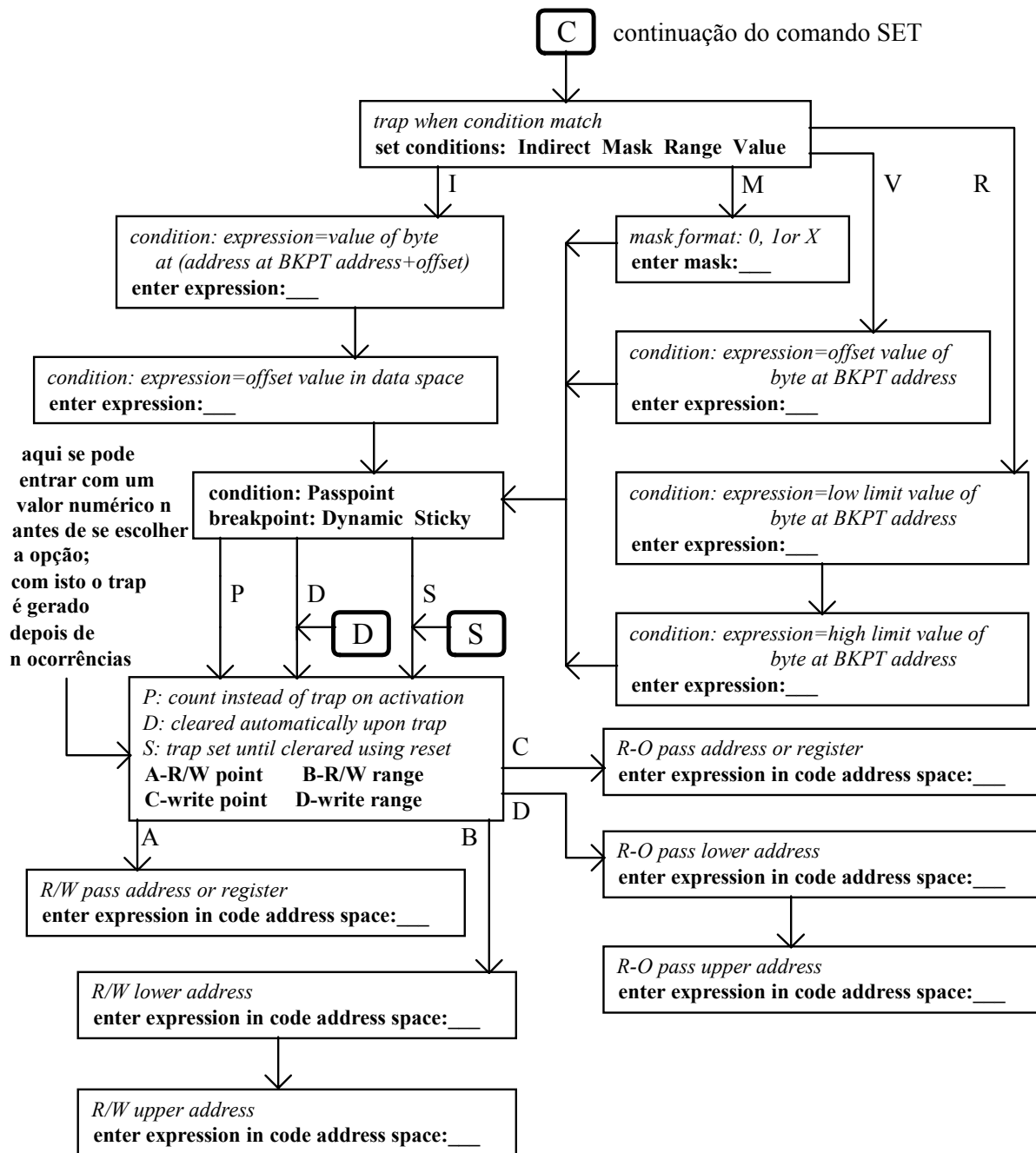


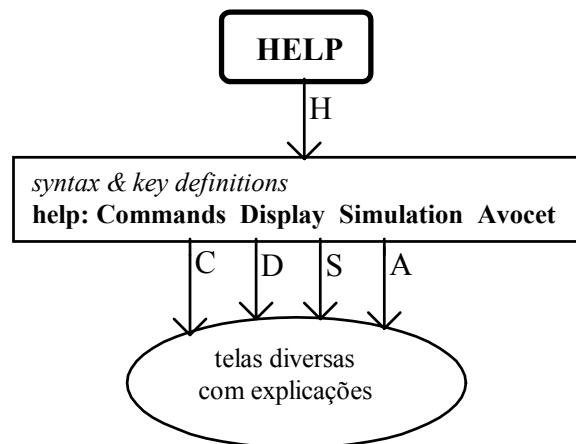
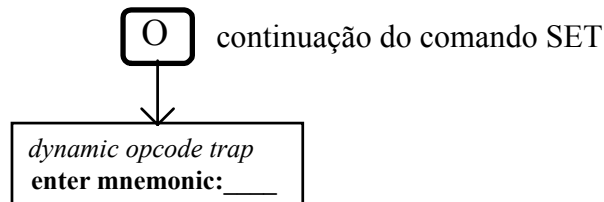
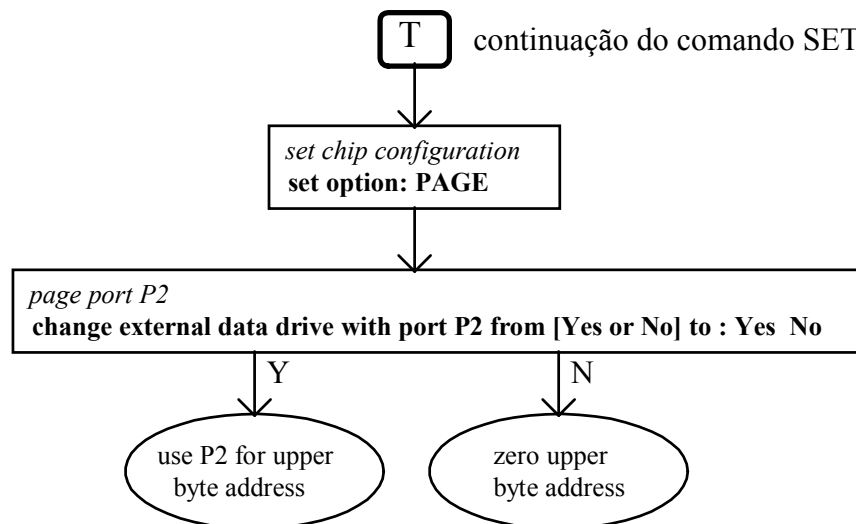












5.9. TELAS DE AJUDA

5.9.1. Ajuda para os Comandos

5.9.2. Ajuda para o Display

5.9.3. Ajuda para a Simulação

RUN:

- F1 → Go Toggle : roda simulação
- F10 → Single step : roda passo a passo
- F9 → Undo : desfaz última instrução

BKPT:

- F2 → move o cursor do breakpoint para cima
- F4 → move o cursor do breakpoint para baixo
- F3 → BKP SET: seta um breakpoint dinâmico no cursor

F5 → Speed: velocidade de simulação

ALT+F5 → Label Toggle “Label”: endereços e operandos simbolicamente
 “ADDR”: sem símbolos na desassemblagem

F6 → Display Toggle “ON”: a tela é atualizada a cada instrução
 “OFF”: somente janelas de trace são atualizadas até que um trap ocorra

ALT+F6 → Trace Toggle “ON”: a janela é atualizada mesmo quando o display está OFF
 “OFF”: a janela é atualizada quando o display está ON

F8 → Skip/Toggle: sai ou entra passo a passo em instruções CALL

Teclas de controle de modo:

- ESC → alterna modo display / comando
- F7 → modo do cursor: Hex, ASCII, BIN
- CTRL+PgUp → alterna o modo de scroll (rolamento)

Movimento do cursor:

- setas do teclado → movem 1 posição em cada sentido

HOME/END → primeiro/último caracter da janela

PgUp/PgDn → rola uma janela para cima/baixo

Movimento entre janelas:

RETURN → vai para a última posição alterada

CTRL + → : move para a janela à direita

CTRL + ← : move para a janela à esquerda

Teclas de edição de objetos:

+/- → incrementa/decrementa byte/word/flag

Ins → alterna byte/nibble/bit (dependendo da posição)

CTRL-END → apaga da posição para o final do objeto

CTRL-HOME → apaga o objeto inteiro

5.9.4. Ajuda para o AVOCET

5.9.5. Tela do Simulador

CAPÍTULO VI

PORTAS PARALELAS

A família MCS-51 oferece 4 portas paralelas, denominadas P0, P1, P2 e P3 e para cada porta existe um SFR. De acordo com a configuração do hardware, uma ou mais portas estarão totalmente ou parcialmente disponíveis.

6.1. REGISTROS ENVOLVIDOS

Os 4 registros, P0, P1, P2 e P3 são na realidade os latches das portas e não os pinos da CPU. Algumas instruções operam com o conteúdo destes latches e outras com os valores dos pinos. As portas paralelas são utilizadas pela CPU para efetuar várias tarefas:

P0 → byte inferior de endereços e dados ==> BUS

P2 → byte superior de endereços

P3 →	0 → RXD	- entrada serial
	1 → TXD	- saída serial
	2 → *INT0	- interrupção externa 0
	3 → *INT1	- interrupção externa 1
	4 → T0	- entrada externa para o contador 0
	5 → T1	- entrada externa para o contador 1
	6 → *WR	- strobe para escrita na memória de dados externa
	7 → *RD	- strobe para leitura da memória de dados externa

Durante uma operação de escrita ou leitura de memória de dados, por exemplo, os dados das portas P0 e P2 são removidos e por eles se emitem os endereços e dados. Terminadas as operações, o conteúdo do latch reaparece nos pinos da CPU.

Existem detalhes sobre a utilização de cada porta como entrada ou saída mas, de uma forma geral, pode-se dizer que:

Saída: Basta escrever 0 ou 1 na porta que o nível lógico aparece nos pinos corretos.

Entrada: todas as portas (exceto P0) possuem um pull-up interno; quando se escreve 1 em um bit da porta, diz-se que este bit está programado como entrada pois dispositivos externos podem colocar (forçar) este 1 em 0. Assim, para ter uma porta como entrada, escreve-se 1 e depois com a leitura será lido 1 ou 0 de acordo com o nível que externamente está aplicado no pino. Devido a isto, as portas são chamadas **quasi-bidirecionais**.

6.2. DESCRIÇÃO DO FUNCIONAMENTO

Cada porta paralela é constituída por três partes:

- um registro latch (SFR ==> P0,P1,P2,P3)
- um driver de saída
- Um buffer de entrada

Os drivers de saída de P0 e P2 e o buffer de entrada de P0 são usados para acessar a memória externa de programa ou de dados.

6.2.1. Porta P1

A figura 6.1 ilustra o esquema elétrico da porta 1.

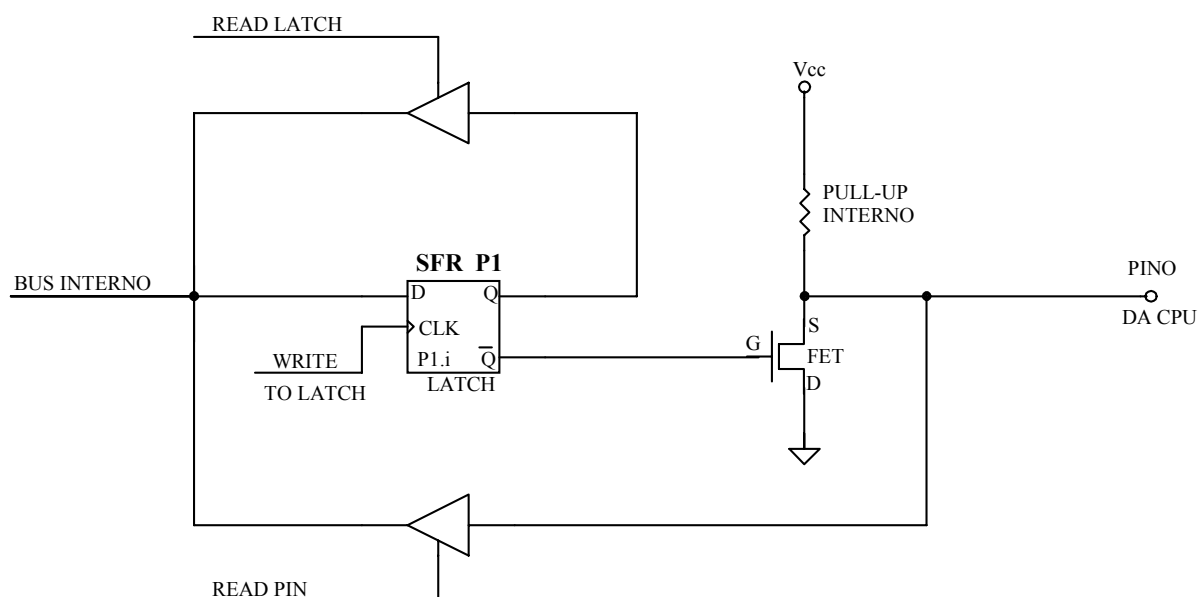


Figura 6.1. Esquema para um bit da porta P1.

O latch de um bit da porta P1 (um bit do SFR P1) é representado por um flip-flop D, no qual se escreve um valor do bus interno através de um pulso Write to Latch gerado pela CPU. A saída do latch é colocada no bus interno através de um sinal Read Latch gerado pela CPU. O nível do pino da porta é colocado no bus interno através do sinal Read Pin, também gerado pela CPU. Algumas instruções que lêem o estado da porta operam com Read Latch enquanto outras operam com Read Pin. Isto será visto com mais detalhes.

Quando escreve na porta:

- 0 → *Q=1 → FET ON → saída=0
- 1 → *Q=0 → FET OFF → saída=1 (pull-up)

Para ser usado como entrada, o latch da porta deve estar em 1; isso desconecta o driver FET da saída. Assim, o pino da porta vai para um nível alto, levado pelo pull-up. Esse pino poderá ser

levado para o nível baixo por qualquer elemento externo. Devido ao pull-up interno, essa porta é denominada "quasi-bidirecional". Quando está configurado para entrada, este vai para 1 e, se externamente é levado para baixo, então fornece corrente. Se a porta estivesse em alta impedância quando configurado como entrada, então seria classificado como "bidirecional verdadeiro".

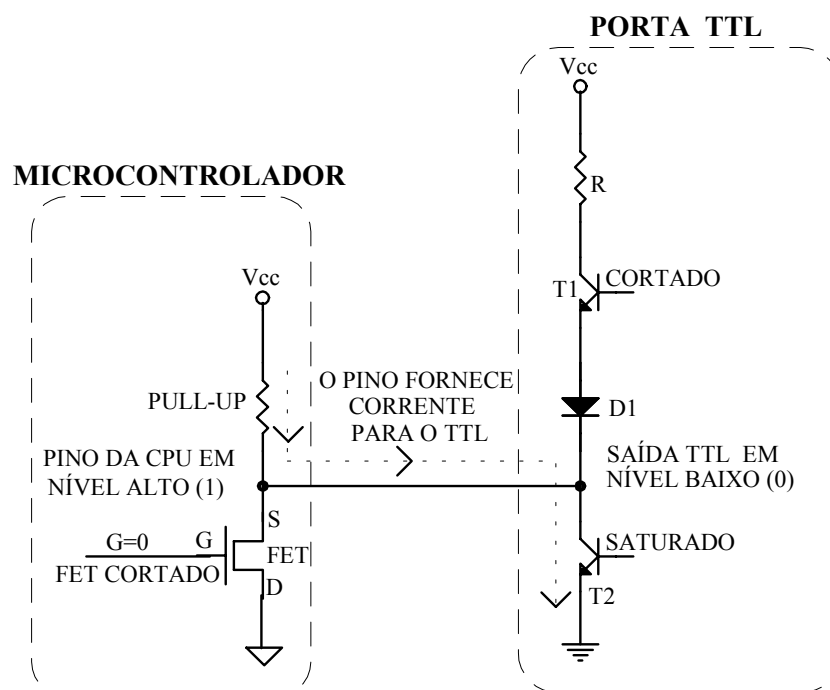


Figura 6.2. Pino do microcontrolador sendo levado a 0 por uma saída TTL.

6.2.2. Porta P3

Na figura 6.3 está o esquema elétrico da porta P3. Pela porta P3 têm-se diversas funções alternativas: *RD, *WR, T0, T1, TXD, RXD, *INT0, *INT1.

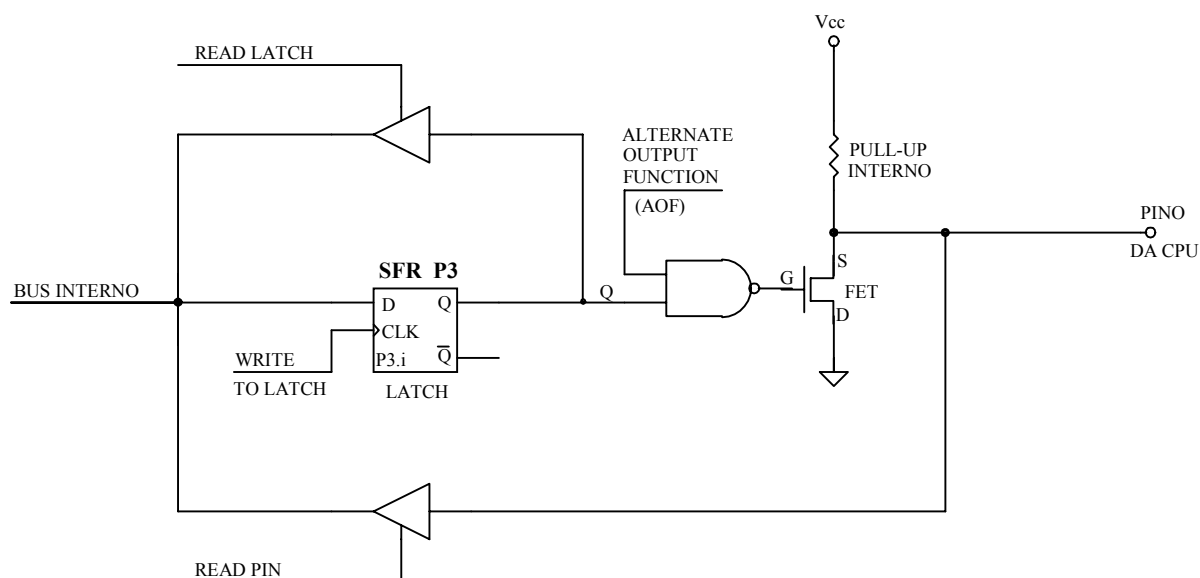


Figura 6.3. Esquema para um bit da porta P3.

Quando AOF=1, tem-se na saída a porta P3. Assim, se AOF=1 e for escrito:

$0 \rightarrow Q=0 \rightarrow G=1 \rightarrow \text{FET ON} \rightarrow \text{saída}=0$
 $1 \rightarrow Q=1 \rightarrow G=0 \rightarrow \text{FET OFF} \rightarrow \text{saída}=1 \text{ (pull-up)}$

Quando o latch de P3 contém 1, a saída pode ser controlada pelo sinal "Alternate Output Function - AOF". Já foram estudadas as funções alternativas geradas através de P3. Se o latch de P3 está em 0, o pino da porta estará em zero e as diversas funções alternativas não estarão disponíveis. P3 também é uma porta quasi-bidirecional.

6.2.3. Porta P2

O esquema da figura 6.4 ilustra a porta P2. Por esta porta também sai o byte mais significativo dos endereços.

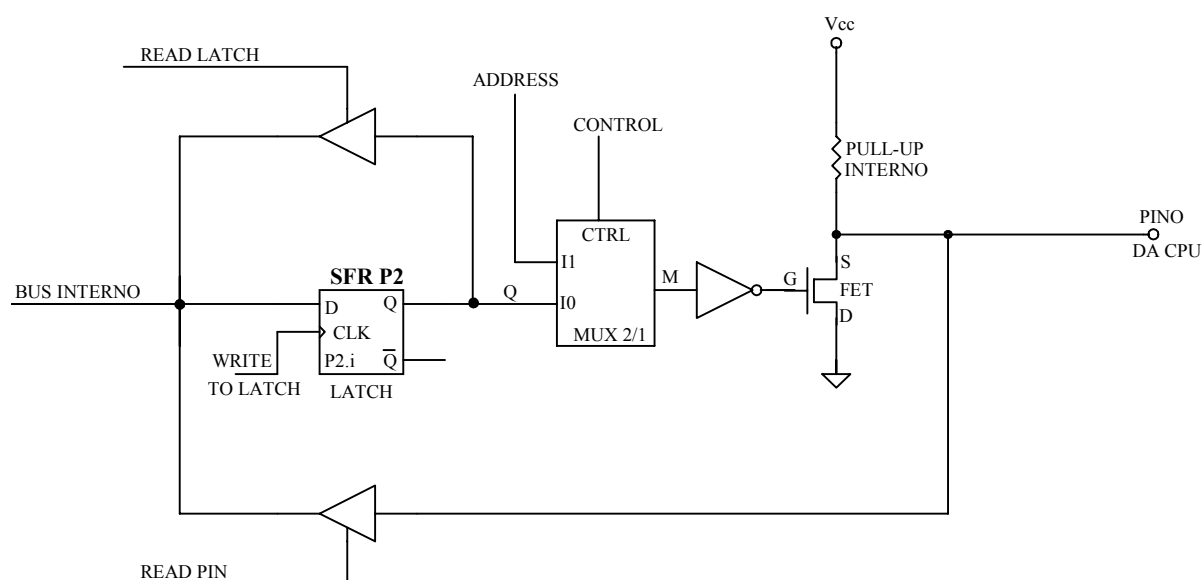


Figura 6.4. Esquema para um bit da porta P2.

O funcionamento é muito semelhante aos casos anteriores.

Se o CONTROL = 0 e for escrito:

$0 \rightarrow Q=0 \rightarrow M=0 \rightarrow G=1 \rightarrow \text{FET ON} \rightarrow \text{saída}=0$
 $1 \rightarrow Q=1 \rightarrow M=1 \rightarrow G=0 \rightarrow \text{FET OFF} \rightarrow \text{saída}=1 \text{ (pull-up)}$

Se o CONTROL = 1, ADDRESS controla o nível no pino. Assim, com CONTROL = 1:

$\text{ADDR}=0 \rightarrow M=0 \rightarrow G=1 \rightarrow \text{FET ON} \rightarrow \text{saída}=0$
 $\text{ADDR}=1 \rightarrow M=1 \rightarrow G=0 \rightarrow \text{FET OFF} \rightarrow \text{saída}=1 \text{ (pull-up)}$

De acordo com o nível lógico do sinal de CONTROL, permite-se a saída ao latch de P2 ou ao byte alto de endereço.

6.2.4. Porta P0

A figura 6.5 ilustra o esquema elétrico da porta P0. Esta porta é a mais complexa pois atende a três funções:

- bus de dados da CPU
- byte menos significativo dos endereços
- porta paralela (bidirecional verdadeira)

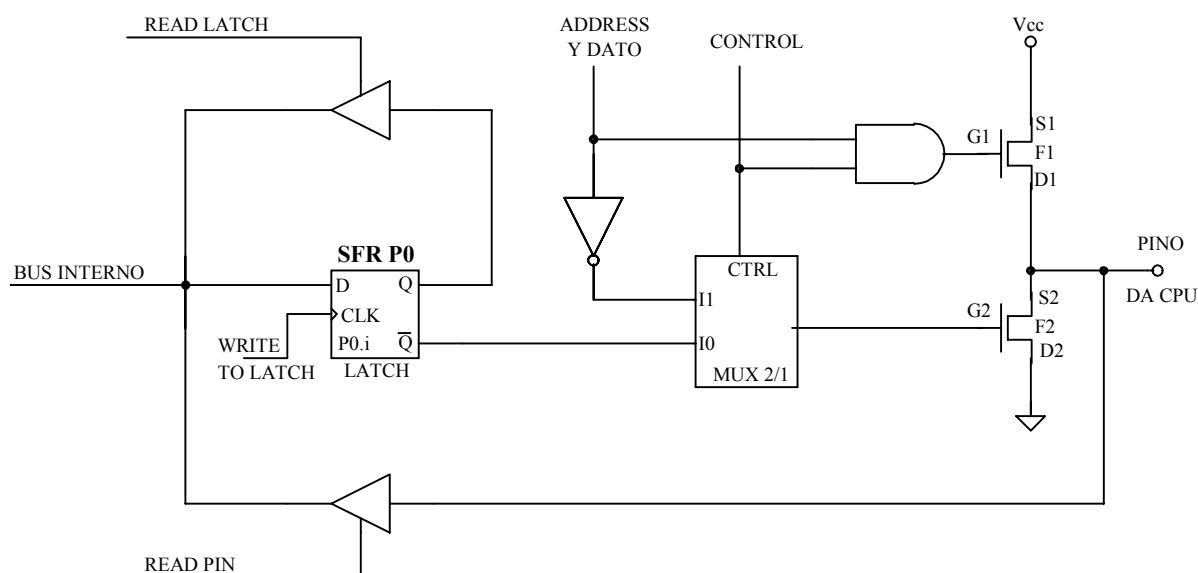


Figura 6.5. Esquema para um bit da porta P0.

A porta P0 difere das demais por ser utilizada para transportar dados durante as operações com a memória e portanto necessita ser bidirecional verdadeira. Se ela está sendo utilizada como porta paralela, então $\text{CONTROL} = 0$. Com isso $G1=0$ e F1 está sempre cortado (não existe pull-up).

Se $\text{CONTROL}=0$ e for escrito :

$0 \rightarrow *Q=1 \rightarrow G2=1 \rightarrow F2 \text{ ON} \rightarrow \text{saída}=0$

$1 \rightarrow *Q=0 \rightarrow G2=0 \rightarrow F2 \text{ OFF} \rightarrow \text{a saída flutua (não há pull-up)}.$

Note que, quando se escreve 1, a porta pode ser utilizada como uma entrada de alta impedância. Para transformá-la em uma porta quase-bidirecional basta colocar externamente resistores de pull-up.

Quando a porta é utilizada para enviar endereços ou dados, $\text{CONTROL} = 1$ e a saída depende de ADDRESS/DATA (ADR/DT):

$\text{ADR/DT}=0 \rightarrow G1=0 \rightarrow F1 \text{ OFF e } G2=1 \rightarrow F2 \text{ ON} \rightarrow \text{saída}=0$

$\text{ADR/DT}=1 \rightarrow G1=1 \rightarrow F1 \text{ ON e } G2=0 \rightarrow F2 \text{ OFF} \rightarrow \text{saída}=1$

(nunca se terá F1 ON e F2 ON ==> curto-circuito).

Para que a porta possa ser usada na leitura de memória é necessária alta impedância; nesse caso $\text{CONTROL}=0$ e se força uma escrita (com 1) no latch de P0.

Se $\text{CONTROL}=0 \rightarrow G1=0 \rightarrow F1 \text{ OFF}$

Se $Q=1 \rightarrow *Q=0 \rightarrow G2=0 \rightarrow F2 \text{ OFF}$

Se F1 e F2 estão em OFF \rightarrow alta impedância.

A leitura da memória é feita com o sinal READ PIN.

6.3. ESCRITA NAS PORTAS

Na execução de qualquer instrução que altere o latch de uma porta, o novo valor chega ao latch durante S6P2, que é o final do ciclo de instrução. Entretanto, os latches são na realidade enviados até os buffers de saída durante a fase P2 de qualquer período de clock (durante P1 se mantém o mesmo valor). Assim, o novo valor aparecerá no pino durante P1 do próximo ciclo (que é S1P1).

Se for necessária uma mudança de 0 para 1 nas portas P1, P2 e P3, o PULL-UP deverá entregar bastante corrente para que essa transição seja rápida. Nas transições de 0 para 1, durante S1P1 e S1P2, entra em atividade um PULL-UP adicional que pode fornecer 100 vezes mais corrente que o PULL-UP normal. Note que os PULL-UP são FETs e não transistores bipolares.

O PULL-UP normal é um FET (depletion-mode) com a porta (G) conectada à fonte (S). Quando está conectado à terra, circula por ele uma corrente de 0,25 mA.

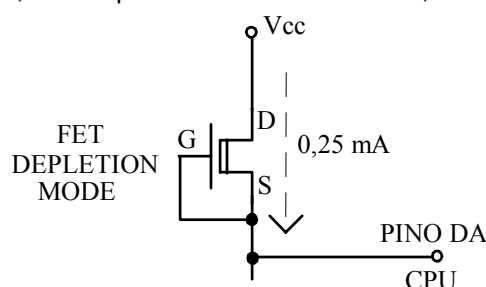


Figura 6.6. O FET (depletion mode) usado como PULL-UP para as portas paralelas. (HMOS)

Na figura 6.7. está o PULL-UP adicional que acelera as transições de 0 para 1. Note que quando $A=B=0$, a saída S vai para 1 e com isto o transistor entra no circuito fornecendo até 30 mA.

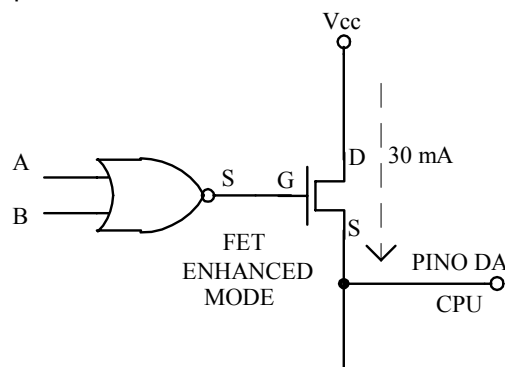


Figura 6.7. Controle para o FET (enhanced mode). (HMOS)

Na figura 6.8 está o esquema completo da saída da porta paralela.

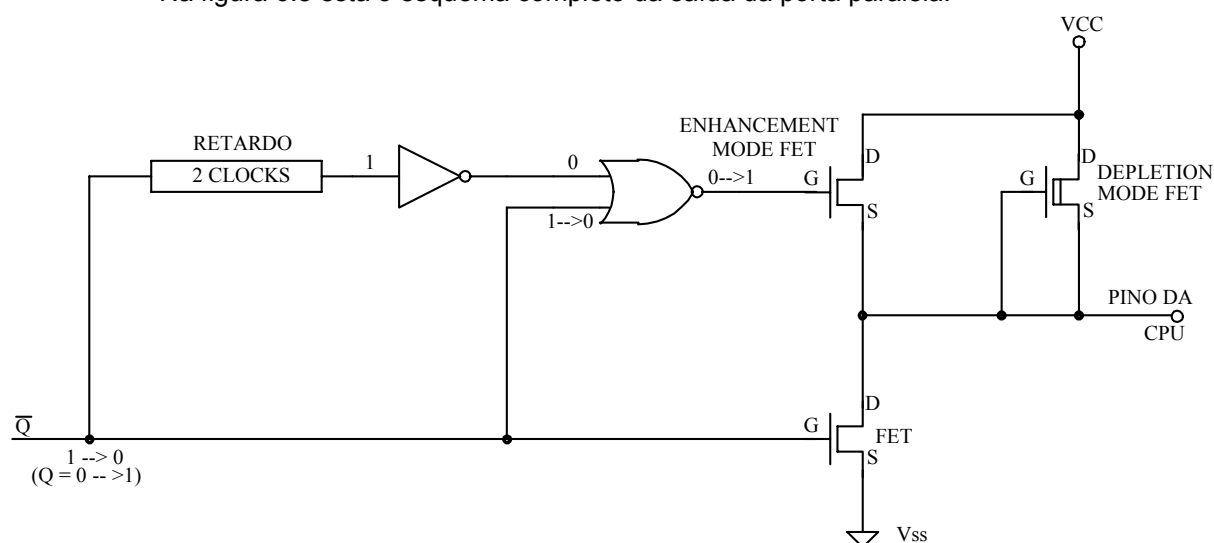


Figura 6.8. Acionamento do FET (enhanced mode) nas transições de 0→1. (HMOS)

Note que se \bar{Q} vai de 1→0, durante dois períodos de clock a saída (S) da porta NOR será 1, permitindo que o enhancement mode FET entre em paralelo com o FET (pull-up) normal, dando maior capacidade de corrente. Este esquema é utilizado em HMOS. Nas famílias CHMOS o esquema varia um pouco mas a idéia é a mesma.

Os buffers de saída das portas P1, P2 e P3 podem acionar até 4 cargas LS TTL. Eles podem ser acionados (trabalhando como entrada) por circuitos TTL e NMOS. Como possuem pull-up interno, essas portas também podem ser acionadas por TTL de coletor aberto mas as transições de 0→1 não serão rápidas porque o pull-up tem baixa capacidade de corrente.

A porta P0 pode acionar até 8 LS TTL (modo BUS). Quando opera como porta paralela, é necessário pull-up externo para acionar outras entradas.

Algumas instruções de leitura utilizam o dado armazenado no latch, enquanto outras usam o estado do pino. As instruções que usam o dado do latch são aquelas que lêem o valor, (possivelmente) o alteram e o escrevem de novo (read-modify-write). Quando o destino do operando é uma porta ou um bit da porta, é utilizado o dado do latch e não o valor do pino. A seguir está uma lista destas instruções que operam com o dado do latch:

Instrução	Exemplo
ANL	ANL P1,A
ORL	ORL P2,A
XRL	XRL P3,A
JBC	JBC P1.1,LB (pula se bit=1 e zera o bit)
CPL	CPL P3.0
INC	INC P2
DEC	DEC P2

DJNZ		DJNZ	P3,label
MOV	PX.Y,C	MOV	P1.0,C
CLR	PX.Y	CLR	P1.2
SETB	PX.Y	SETB	P1.3

Pode parecer que as 3 últimas instruções não são do tipo "read-modify-write". Na realidade, é lido todo o byte da porta, o bit considerado é alterado e o novo byte é devolvido para a porta.

Uma razão para usar o dado do latch e não o valor do pino é evitar um equívoco na interpretação do nível de tensão do pino, como por exemplo, quando um bit de uma porta está sendo usado para acionar a base de um transistor.

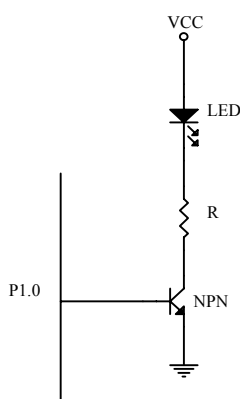


Figura 6.9. Bit 0 da porta 1 sendo usado para acender um led através de um transistor.

Quando P1.0=1 → LED aceso

Quando P1.0=0 → LED apagado

Sem dúvida quando P1.0=1 a tensão no pino vai estar em 0,7 V, o que é um 0 lógico em nível TTL. Espera-se que a instrução CPL P1.0 inverta o estado do led. Supondo que o led esteja aceso (P1.0=1), se a CPU usa o valor do pino, ela vai obter 0 e isto fará P1.0=1, quer dizer, não vai mudar o estado do led. O melhor é usar o dado do latch, que está em 1.

6.4. EXERCÍCIOS

Todos as exercícios e exemplos estão baseados no circuito de práticas.

EXERCÍCIO 6.1. PISCA1 E PISCA2

Acender os 3 leds em seqüência. Não esquecer de colocar um retardo. O led se acende quando se põe nível alto em um bit da porta.

```
;PISCA1.ASM
;
RTD      EQU          60000          ; 65536>RTD>256
VERMELHO EQU          P1.0
AMARELO  EQU          P1.1
VERDE    EQU          P1.2
;
                DEFSEG PROG, CLASS=CODE, START=0
                SEG      PROG
                ORG      RESET
INIC        MOV      P1,#0          ;APAGA OS LEDS
AQUI        SETB     VERMELHO
                LCALL    RETARD
                CLR      VERMELHO
                SETB     AMARELO
                LCALL    RETARD
                CLR      AMARELO
                SETB     VERDE
                LCALL    RETARD
                CLR      VERDE
                SJMP     AQUI
;
;ROTINA PARA GERAR UM RETARDO
RETARD      MOV      R7,# HIGH RTD
L1          MOV      R6,# LOW  RTD
L2          DJNZ     R6,L2
                DJNZ     R7,L1
                RET
                END
```

Uma outra solução, muito mais simples, pode ser utilizada. Como não são usados os demais pinos da porta P1, pode-se escrever qualquer coisa neles. Usa-se então o Acc e o CY para obter um total de 9 bits e, através de rotações, consegue-se gerar os códigos para acender os leds em seqüência. A figura 6.10 ilustra a idéia.

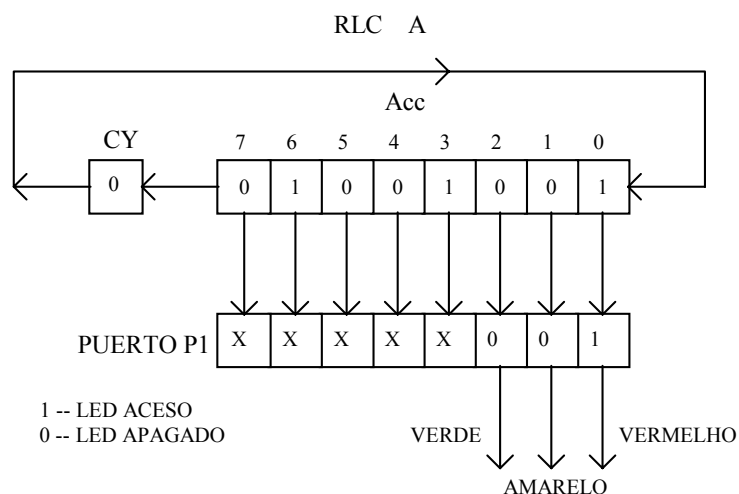


Figura 6.10. Uso do Acc e CY para acender leds em seqüência.

Se Acc e CY são inicializados com estes valores, basta uma rotação com CY (RLC A) para mudar o led que deve ficar aceso. A cada rotação escreve-se o conteúdo de Acc em P1.

```
;PISCA2.ASM
;
RTD      EQU          60000          ; 65536>RTD>256
VERMELHO EQU          P1.0
AMARELO  EQU          P1.1
VERDE    EQU          P1.2
;
                DEFSEG PROG, CLASS=CODE, START=0
                SEG      PROG
                ORG      RESET
INIC         MOV      A, #01001001B    ; 01 001 001
                CLR      C
AQUI         MOV      P1, A
                ACALL    RETARD
                RLC      A
                SJMP     AQUI
;
;ROTINA PARA GERAR UM RETARDO
RETARD       MOV      R7, # HIGH RTD
L1           MOV      R6, # LOW  RTD
L2           DJNZ     R6, L2
                DJNZ     R7, L1
                RET
                END
```

EXERCÍCIO 6.2. CHAVE1, CHAVE2 E CHAVE3

Usando os três leds como um contador de três bits, construir um programa que conte os acionamentos da chave SW2 (a contagem será observada nos três leds). Note que a chave não necessita de pull-up porque usa o pull-up interno da porta. A figura 6.11 ilustra as conexões.

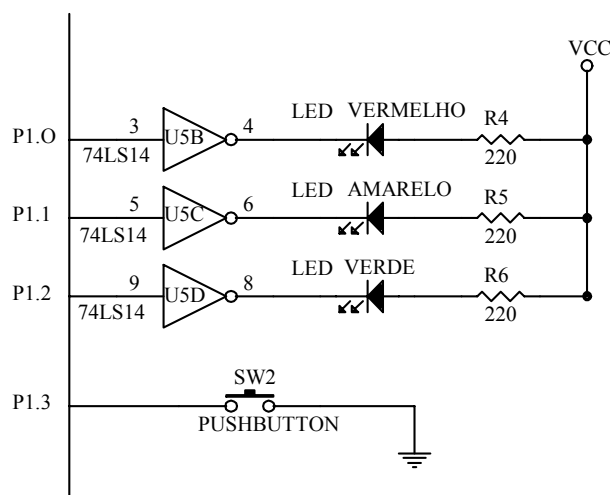


Figura 6.11. Conexões de leds e chave no circuito de práticas.

No primeiro programa será usada uma solução simples, que tem o problema de bouncing.

```

;CHAVE1.ASM
;
SW2          EQU          P1.3
;
                DEFSEG PROG, CLASS=CODE, START=0
                SEG        PROG
                ORG        RESET
INIC          CLR         A           ;ZERAR CONTADOR
                MOV        P1,A       ;APAGAR LEDS
                SETB       SW2        ;PROGRAMAR P1.3 COMO ENTRADA
AQUI1        JB          SW2,AQUI1    ;AGUARDAR ACIONAMENTO
                INC        A
                MOV        P1,A       ;DAR SAÍDA AO CONTADOR
                SETB       SW2        ;GARANTIR P1.3 COMO ENTRADA
AQUI2        JNB        SW2,AQUI2     ;AGUARDAR LIBERAR CHAVE
                SJMP       AQUI1
                END

```

No segundo programa é apresentada uma solução para eliminar o bouncing utilizando retardos. Ao detectar uma transição na chave, o programa aguarda um tempo para que se extinga o bouncing. O tempo que se deve aguardar é determinado de forma empírica.

```

;CHAVE2.ASM
;
; NESTE PROGRAMA SE PRETENDE ELIMINAR O BOUNCING COM RETARDOS
; O VALOR DO RETARDO É PARTICULAR PARA CADA CIRCUITO
; (DEPENDENTE DO TAMANHO DOS CABOS, QUALIDADE DA CHAVE, ETC)
; RECOMENDA-SE: DETERMINAR EMPÍRICAMENTE O MELHOR RETARDO PARA CADA CIRCUITO
;
RTD          EQU          500          ;RTD>256
SW2          EQU          P1.3
;
                DEFSEG PROG, CLASS=CODE, START=0
                SEG        PROG
                ORG        RESET
INIC          CLR         A           ;ZERAR CONTADOR
                MOV        P1,A       ;APAGAR LEDS
                SETB       SW2        ;PROGRAMAR P1.3 COMO ENTRADA
AQUI1        JB          SW2,AQUI1    ;AGUARDAR ACIONAMENTO
                INC        A
                MOV        P1,A       ;DAR SAÍDA AO CONTADOR
                SETB       SW2        ;GARANTIR P1.3 COMO ENTRADA
                ACALL      RTD         ;ELIMINAR BOUNCING
AQUI2        JNB        SW2,AQUI2     ;AGUARDAR LIBERAR CHAVE
                ACALL      RTD         ;ELIMINAR BOUNCING
                SJMP       AQUI1
;
;ROTINA PARA GERAR UM RETARDO
RETARD       MOV         R7,# HIGH RTD
L1           MOV         R6,# LOW  RTD
L2           DJNZ        R6,L2
                DJNZ      R7,L1
                RET
                END

```

A solução de aguardar um tempo depois de detectar uma transição na chave funciona bem, mas oferece duas principais desvantagens. A primeira é que se aguarda um intervalo de tempo fixo, ou seja, se a chave de baixa qualidade é trocada por uma de melhor qualidade e que possua pouco bouncing, tem-se que mudar o programa. A segunda desvantagem é sua vulnerabilidade a ruídos pois estes podem provocar acionamentos indevidos. O programa chave 3, apesar de ser simples, soluciona estes problemas. Usam-se duas rotinas que detectam transições de 0→1 e de 1→0; o que se especifica é quanto tempo a chave deve estar em 0 ou em 1 para que se considere o acionamento válido (para que se considere o fim do bouncing).

```
;CHAVE3.ASM
;
; SOLUCAO MAIS EFICIENTE PARA ELIMINAR O BOUNCING COM RETARDOS
; USAM-SE ROTINAS QUE AGUARDAM A ESTABILIZACAO DA CHAVE
; HA UMA PARA AS TRANSIÇÕES DE 0 PARA 1 E OUTRA DE 1 PARA 0
; ESTA TECNICA SE ADAPTA MELHOR AS VARIACOES ENTRE OS CIRCUITOS
;
RTD          EQU          100
SW2          EQU          P1.3
;
                DEFSEG PROG, CLASS=CODE, START=0
                SEG        PROG
                ORG        RESET
INIC          CLR          A                ;ZERAR CONTADOR
                MOV        P1,A            ;APAGAR LEDS
                SETB       SW2            ;PROGRAMAR P1.3 COMO ENTRADA
AQUI          ACALL        RBT_1_0        ;TRANSICAO LIMPA DE 1 PARA 0
                INC        A
                MOV        P1,A            ;DAR SAÍDA AO CONTADOR
                ACALL        RBT_0_1        ;TRANSICAO LIMPA DE 0 PARA 1
                SJMP        AQUI
;
;ELIMINAR BOUNCING NAS TRANSIÇÕES DE 0 PARA 1
RBT_0_1       MOV        R7,#RTD
LB1           JNB         SW2,RBT_0_1
                DJNZ       R7,LB1
                RET
;
;ELIMINAR BOUNCING NAS TRANSIÇÕES DE 1 PARA 0
RBT_1_0       MOV        R7,#RTD
LB2           JB          SW2,RBT_1_0
                DJNZ       R7,LB2
                RET
                END
```

CAPÍTULO VII

INTERRUPÇÕES

7.1. INTRODUÇÃO

O 8051 apresenta 5 tipos de interrupções:

- 2 externas
- 2 timers
- 1 serial

Alguns outros membros da família MCS-51 podem apresentar outras interrupções; por exemplo, o 8052 tem uma interrupção adicional dedicada ao timer 2.

7.2. REGISTROS ENVOLVIDOS

Os registros dedicados às interrupções permitem um controle total sobre as mesmas. Cada interrupção pode ser habilitada ou desabilitada individualmente. Também é possível desabilitar todas as interrupções de uma só vez. O registro **IE** (Interrupt Enable) controla a habilitação das interrupções e é ilustrado na figura 7.1.

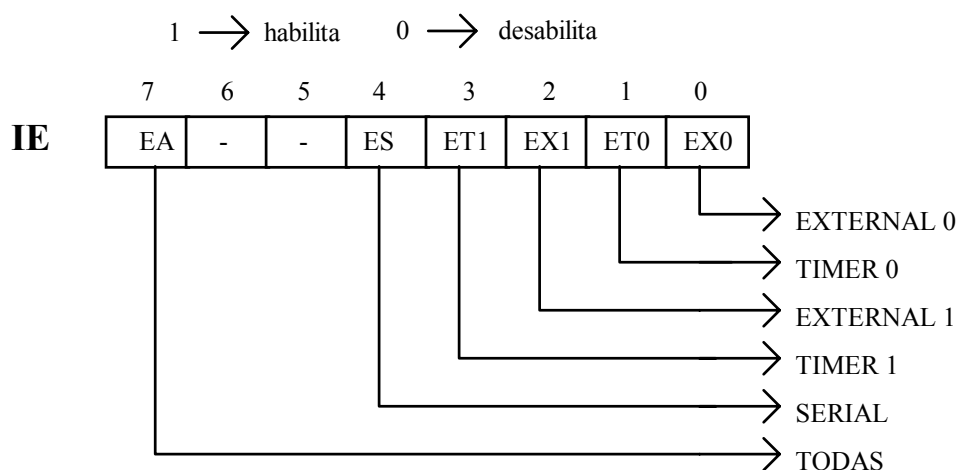


Figura 7.1. Registro IE - habilitação de interrupções.

Cada interrupção pode ter dois níveis de prioridade: prioridade alta ou prioridade baixa. Uma interrupção de alta prioridade pode interromper uma de baixa prioridade mas não acontece o contrário. Uma interrupção não pode interromper uma outra de mesma prioridade. Se forem

recebidas interrupções de diferentes prioridades, a de alta prioridade é atendida primeiro. Se forem recebidas duas interrupções de igual prioridade, determina-se por uma seqüência interna de polling (consulta) qual será atendida primeiro. Assim, dentro de um mesmo nível de prioridade, existe uma seqüência de atendimento (a seqüência do polling). O conteúdo do registro **IP** define as prioridades e é ilustrado na figura 7.2.

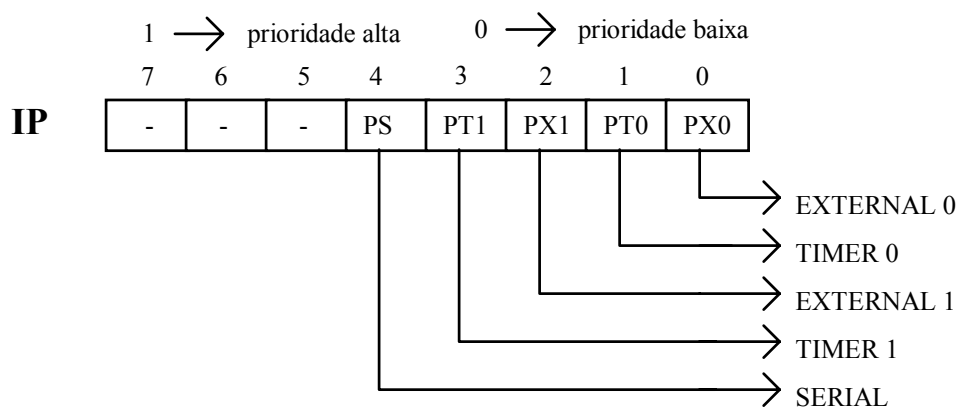


Figura 7.2. Registro IP - prioridade das interrupções.

As duas interrupções externas (INT0 e INT1) podem ser acionadas por nível ou por borda de descida (\downarrow). Isto é definido através de dois bits do registro **TCON**. Este registro também tem outra finalidade pois cada interrupção indica sua ativação usando um bit do registro **TCON**.

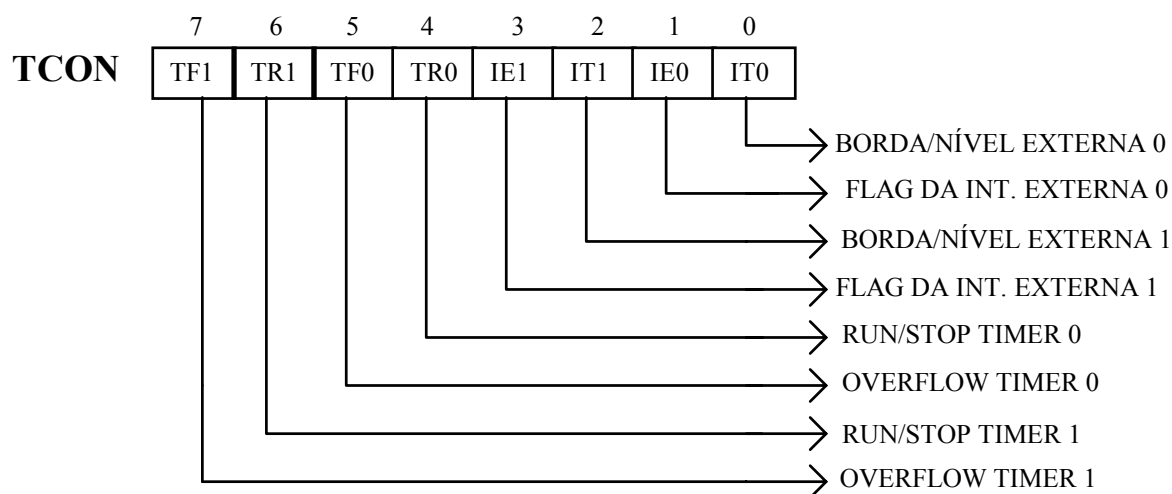


Figura 7.3. Registro TCON - diversas funções para interrupções e timers.

TF1 → flag de transbordamento (overflow) do contador/temporizador 1. Ativado por hardware quando há transbordamento no contador do timer 1 (timer/counter 1). É apagado por hardware quando o processador é desviado para a rotina de atendimento da interrupção.

TR1 → bit de partida/parada (run/stop) do contador/temporizador 1.

- TF0 → flag de transbordamento (overflow) do contador/temporizador 0. Ativado por hardware quando há transbordamento no contador do timer 0 (timer/counter 0). É apagado por hardware quando o processador é desviado para a rotina de atendimento da interrupção.
- TR0 → bit de partida/parada (run/stop) do contador/temporizador 0.
- IE1 → flag da interrupção externa 1. É ativado (colocado em um) por hardware quando se detecta uma interrupção externa 1. É apagado (colocado em zero) por hardware (só no modo borda) quando o processador é desviado para a rotina de atendimento da interrupção.
- IT1 → indica se a interrupção externa 1 opera por borda ou por nível:
 1 → borda de descida (↓),
 0 → nível baixo.
- IE0 → flag da interrupção externa 0. É ativado (colocado em um) por hardware quando se detecta uma interrupção externa 0. É apagado (colocado em zero) por hardware (só em modo borda) quando o processador é desviado para a rotina de atendimento da interrupção.
- IT0 → indica se a interrupção externa 0 opera por borda ou por nível:
 1 → borda de descida (↓),
 0 → nível baixo.

O flag de interrupção da porta serial está em outro registro (SCON).

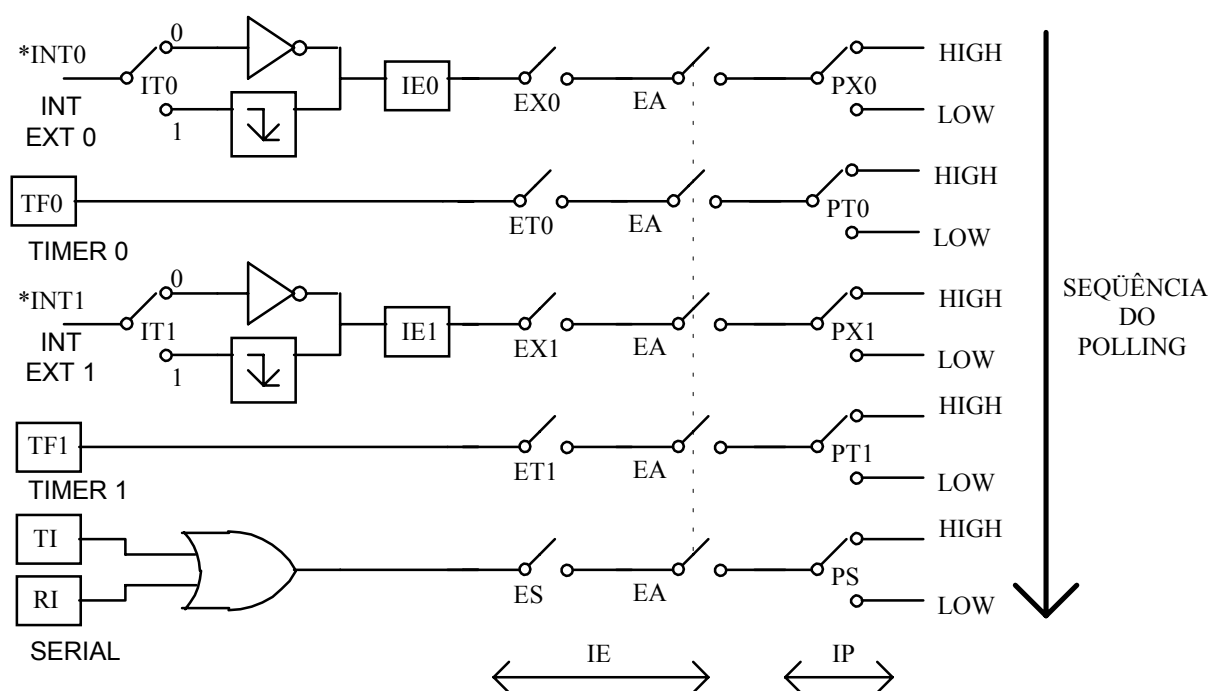


Figura 7.4. Esquema das interrupções na família MCS-51.

Todos os bits que geram interrupção podem ser ativados por software com os mesmos resultados como se tivessem sido ativados por hardware. Isto quer dizer que uma interrupção pode ser ativada por software e também que as interrupções pendentes podem ser canceladas.

Cada interrupção é vetorizada em um endereço pré-definido:

RESET		00H
Externa 0	IE0	03H
Timer 0	TF0	0BH
Externa 1	IE1	13H
Timer 1	TF1	1BH
Serial	RI+TI	23H (o + indica OU)

No endereço de vetorização das interrupções há pouco espaço, o suficiente para colocar umas poucas instruções. Portanto é normal colocar nestes endereços um desvio (jump) para um outro local, onde está a rotina que atende à interrupção.

7.3. MANEJO DE INTERRUPÇÕES

Os flags das interrupções são amostrados na segunda fase do quinto estado (S5P2) de cada ciclo de máquina. As amostras são submetidas a polling (consulta) durante o próximo ciclo de máquina. A figura 7.5 ilustra a seqüência para atendimento de uma interrupção.

Se um dos flags (de interrupção) está ativo durante S5P2, o próximo ciclo de polling vai detectar e identificar a interrupção a ser atendida e será gerada uma instrução LCALL sempre que não seja bloqueada por:

- uma interrupção de prioridade igual ou mais alta que a que está sendo atendida,
- não tenha finalizado a instrução que se está processando no ciclo de polling corrente,
- a instrução em progresso é um RETI ou uma escrita em IP ou IE.

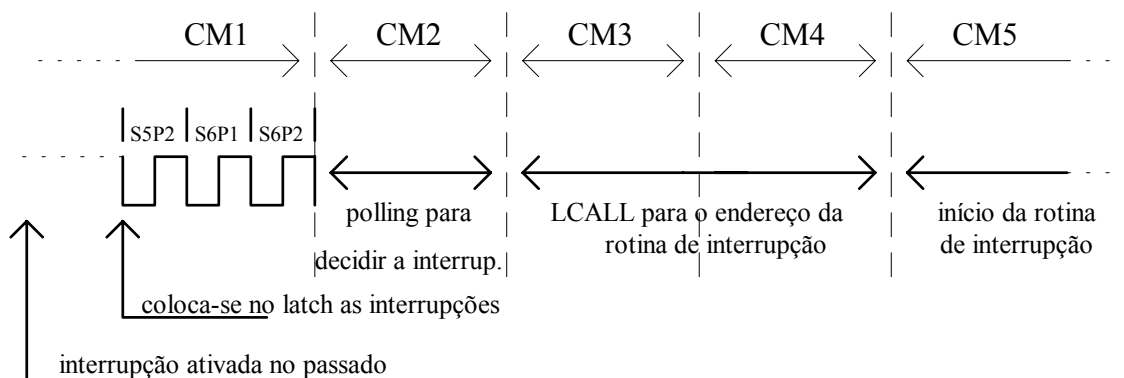


Figura 7.5. Seqüência para atendimento a uma interrupção.

Na figura 7.5 é ilustrada a resposta mais rápida a uma interrupção. Neste caso CM2 é o final de uma instrução e não é um RETI nem uma escrita em IP ou IE. Quando CM2 é uma instrução RETI ou uma escrita em IP ou IE, uma instrução a mais será executada antes que a interrupção seja vetorizada.

A sequência de polling se repete a cada ciclo de máquina e os valores processados são aqueles que estarão presentes no instante S5P2 do ciclo de máquina anterior. Deve-se observar que se um flag de interrupção foi ativado mas não pôde ser atendido (por uma das condições de bloqueio) e é apagado antes da condição de bloqueio ser removida, a interrupção não será atendida. Quer dizer, o fato de um flag de interrupção estar ativo e não ser atendido, não será gravado.

O processador reconhece um pedido de interrupção através da execução de um LCALL gerado por hardware. Normalmente o flag que gerou o pedido é zerado por hardware, exceto para:

- TI, RI da porta serial,
- IE0, IE1 quando ativado por nível

A instrução RETI é usada para finalizar uma rotina de atendimento a interrupções. Uma instrução RET funciona mas o sistema de controle de interrupções não sabe que a rotina terminou, ou seja, serão bloqueadas todas as demais interrupções de prioridade igual ou inferior.

7.4. INTERRUPÇÕES EXTERNAS

As duas interrupções externas (0 e 1) podem ser programadas para funcionar por nível (ITX=0) ou por borda de descida (↓) (ITX=1). Os pinos das interrupções são amostrados em cada ciclo de máquina (1 ciclo de máquina=12 períodos de clock); assim, uma interrupção externa deve permanecer constante por pelo menos 12 períodos de clock, caso contrário ela pode ser ignorada.

Quando a interrupção opera por borda de descida (↓), o pino deve permanecer em alto por pelo menos 12 períodos de clock e depois em baixo por pelo menos 12 períodos de clock; isto garante a ativação do flag de interrupção. Os flags de interrupções externas (IEX0 ou IEX1) serão automaticamente zerados pela CPU quando uma rotina de serviço é chamada.

Quando a interrupção opera por nível, o pino deve permanecer ativado (em baixo) até que a interrupção seja atendida; depois disso, o pedido deve ser removido antes que a rotina de serviço de interrupção termine caso contrário uma nova interrupção será gerada.

O tempo de latência de uma interrupção depende, entre outras coisas, das condições de bloqueio já vistas. No caso destas condições de bloqueio existirem, leva-se entre 5 a 9 ciclos de máquina.

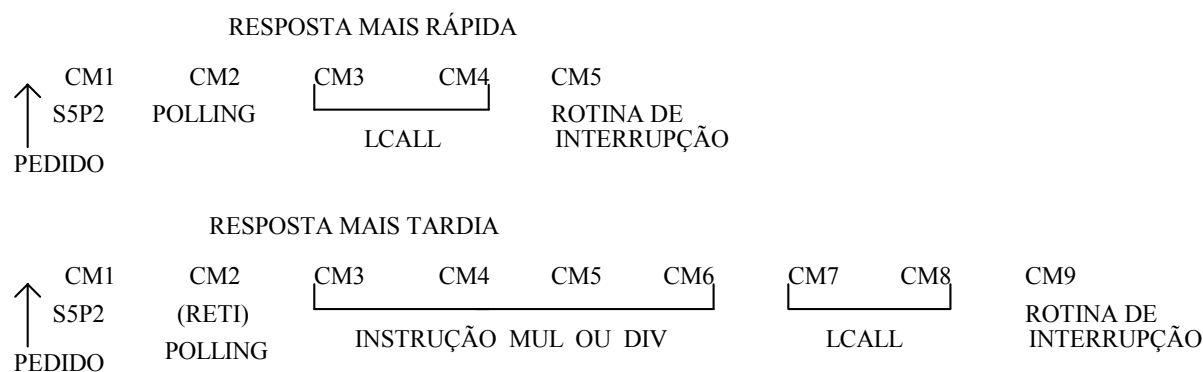
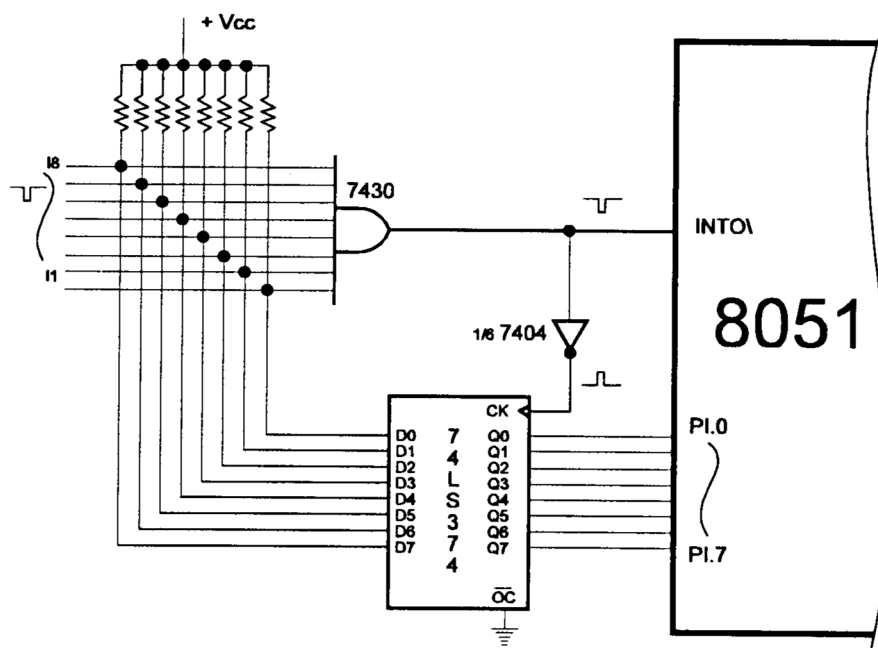


Figura 7.6. Latência para as interrupções.

Como os flags de interrupção serial e dos temporizadores/contadores são ativados em S5P2, o que foi estudado é válido para qualquer interrupção.

Caso seja necessário, pode-se usar um esquema como o da figura a seguir para expandir as interrupções externas:



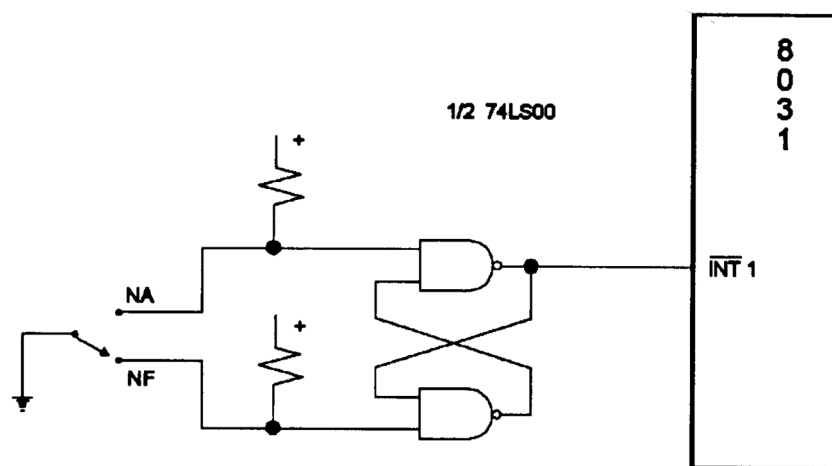
7.5. PASSO A PASSO

A estrutura de interrupção do MCS-51 permite, de forma muito simples, uma implementação para a execução de programas passo a passo (single step). Como já foi visto, uma interrupção não será atendida quando uma interrupção de igual prioridade esteja sendo atendida e, quando esta interrupção termina com um RETI, obrigatoriamente uma instrução será executada antes que se aceite a nova interrupção.

Uma interrupção não será re-atendida antes de terminar sua rotina. Essa rotina deve terminar com RETI e, depois dessa instrução, uma outra deverá ser executada antes de ser atendida a interrupção.

Imagine um sistema onde a interrupção 1 está programada para trabalhar por nível e que a entrada INT1 esteja conectada a terra, ou seja, a interrupção está sempre sendo invocada. Se forem adicionadas a esse sistema uma rotina principal e uma rotina de serviço para a interrupção 1, cria-se o ambiente para executar passo a passo o programa principal. Cada vez que voltar da rotina de interrupção (com um RETI), será executada uma única instrução do programa principal e, em seguida, desviará novamente para a rotina de interrupção. Pode-se usar uma chave para marcar o instante de retorno da rotina de interrupção. O exercício 7.4 ilustra a utilização do recurso de passo a passo.

A figura 11.5 (capítulo 11) mostra uma implementação possível para a execução passo a passo. Outra possibilidade, mais complexa mas sem bouncing na chave é mostrada na figura a seguir:



7.6. EXERCÍCIOS

EXERCÍCIO 7.1. LED_INT1, LED_INT2 E LED_INT3

No circuito de práticas existe um pushbutton (SW3) para a interrupção 1 que será usado para fazer acender os leds (vermelho, amarelo e verde) em sequência. Cada vez que a chave for acionada, uma interrupção será gerada e a rotina de atendimento à interrupção deverá trocar o led aceso.

```
;LED_INT1.ASM
                DEFSEG PROG, CLASS=CODE, START=0
                SEG  PROG
;
                ORG          RESET
                AJMP         INICIO
;
                ORG          EXTI1
                AJMP         EXT1
;
                ORG          50H
INICIO          MOV          A,#01001001B
                CLR          C
                MOV          P1,A
                SETB         EX1          ;HABILITAR INT. EXT. 1
                SETB         EA          ;HABILITAR FLAG GERAL
                SJMP         $           ;LOOP INFINITO
;
;ROTINA PARA ATENDER A INTERRUPCAO EXTERNA 1
EXT1            RLC          A           ;ACENDER LEDS EM SEQUENCIA
                MOV          P1,A
                RETI
                END
```

Nota-se um problema: os 3 leds se acendem quando a chave é acionada e só um led fica aceso quando se libera a chave, mas nunca se sabe qual. Este problema acontece porque a rotina de interrupção é muito mais rápida que a chave. Ao retornar da interrupção, a chave ainda está acionada e uma nova interrupção é gerada. Para solucionar isso, pode-se monitorar o sinal INT1 (P3.3) e só permitir o retorno depois de que a chave esteja liberada (vai para um).

Para ilustrar esta solução, o programa LED_INT2.ASM foi implementado. O programa é idêntico ao anterior exceto por uma instrução antes do retorno da interrupção.

```
;LED_INT2.ASM
                DEFSEG PROG, CLASS=CODE, START=0
                SEG  PROG
SW3             EQU          P3.3
;
                ORG          RESET
                AJMP         INICIO
;
                ORG          EXTI1
                AJMP         EXT1
;
                ORG          50H
INICIO          MOV          A,#01001001B
                CLR          C
                MOV          P1,A
                SETB         EX1          ;HABILITAR INT. EXT. 1
                SETB         EA          ;HABILITAR FLAG GERAL
                SJMP         $           ;LOOP INFINITO
;
;ROTINA PARA ATENDER A INTERRUPCAO EXTERNA 1
EXT1            RLC          A           ;ACENDER LEDS EM SEQUENCIA
                MOV          P1,A
```

```

AQUI          JNB          SW3,AQUI          ;AGUARDAR LIBERAR SW2
              CLR          IE1
              RETI
              END

```

Na verdade, o que se fez foi obrigar que a interrupção que está programada para operar por nível opere por borda (aguarda-se a liberação da chave). O que acontece quando se usa o primeiro programa mas com a interrupção trabalhando por borda (↓) ? Supõe-se que os leds deverão acender na seqüência correta.

```

;LED_INT3.ASM
                DEFSEG PROG, CLASS=CODE, START=0
                SEG PROG
;
                ORG          RESET
                AJMP         INICIO
;
                ORG          EXT11
                AJMP         EXT1
;
                ORG          50H
INICIO          MOV          A,#01001001B
                CLR          C
                MOV          P1,A
                SETB         IT1          ;EXT 1 OPERA POR BORDA
                SETB         EX1          ;HABILITAR INT. EXT. 1
                SETB         EA          ;HABILITAR FLAG GERAL
                SJMP         $           ;LOOP INFINITO
;
;ROTINA PARA ATENDER A INTERRUPCAO EXTERNA 1
EXT1            RLC          A           ;ACENDER LEDS EM SEQUENCIA
                MOV          P1,A
                RETI
                END

```

Talvez se note um acionamento indevido ao liberar a chave porque a chave é ruidosa e, ao ser liberada, aparece o bouncing (seqüências de zeros e uns). Uma solução definitiva é usar a rotina de eliminação de bouncing para controlar a transição de 1 para 0.

EXERCÍCIO 7.2. PASSO_EXE.ASM

Escrever um programa que faça acender os leds em seqüência (vermelho, amarelo e verde). Executar esse programa com um controle passo a passo usando a interrupção 1. A solução é muito simples. No programa principal estará somente uma seqüência de instruções que acende os leds segundo a ordem especificada (sem retardo). A interrupção externa 1 estará programada para operar por nível. A rotina de interrupção aguarda que a chave SW3 seja acionada (vá a zero); quando isto ocorre, retorna-se da rotina de interrupção. Ao regressar, a entrada INT1 estará em zero (pela chave SW3) e uma nova interrupção é gerada. Ou seja, a cada acionamento da chave SW3, é executada uma instrução do programa principal. A primeira interrupção, para que o programa entre em controle passo a passo, é provocada por software ao ativar o bit IE1.

```

;PASSO_EXE.ASM
;
; PROGRAMA PARA ILUSTRAR A EXECUCAO PASSO A PASSO
; SERA USADA A CHAVE SW3 PARA EXECUTAR PASSO A PASSO O LOOP PRINCIPAL
; CADA VEZ QUE SE ACIONA SW3, UMA INSTRUCAO É EXECUTADA
;
RTD            EQU          100

```

```

SW3      EQU      P3.3
;
          DEFSEG PROG, CLASS=CODE, START=0
          SEG      PROG
;
          ORG      RESET
          AJMP     INIC
;
          ORG      EXTI1
          AJMP     EXTI
;
          ORG      50H
INIC      MOV      A,#01001001B    ;CODIGO PARA ACENDER
          CLR      C                ;LEDS EM SEQUENCIA
          MOV      P1,A
          SETB     EX1              ;HABILITAR INT 1
          SETB     EA              ;HABILITAR GERAL
          SETB     IE1             ;PROVOCAR INT 1
          NOP                     ;AQUI O PROG INTERROMPE
PRINCIPAL RLC      A                ;RODAR CODIGO
          MOV      P1,A            ;ACENDER LEDS
          SJMP     PRINCIPAL       ;RETORNAR
;
          ORG      100H
EXTI1     ACALL    RBT_0_1          ;TRANSICAO DE 0 A 1
          ACALL    RBT_1_0          ;TRANSICAO DE 1 A 0
          RETI
;
;ELIMINAR BOUNCING NAS TRANSICOES DE 0 PARA 1
RBT_0_1   MOV      R7,#RTD
LB1       JNB      SW3,RBT_0_1
          DJNZ     R7,LB1
          RET
;
;ELIMINAR BOUNCING NAS TRANSICOES DE 1 PARA 0
RBT_1_0   MOV      R7,#RTD
LB2       JB       SW3,RBT_1_0
          DJNZ     R7,LB2
          RET
          END

```


CAPÍTULO VIII

TEMPORIZADORES / CONTADORES

8.1. INTRODUÇÃO

O 8051 tem dois registros contadores de 16 bits, denominados TIMER 0 e TIMER 1, dedicados às funções de contagem e temporização (counter/timer-contador/temporizador). Há uma importante distinção entre os conceitos de contador e temporizador.

Quando opera como **temporizador**, o registro é incrementado a cada ciclo de máquina (usa como base o cristal da CPU). O sinal de contagem aparece com 1/12 da frequência do clock.

Quando opera como **contador**, o registro é incrementado de acordo com o sinal que se coloca nas entradas T1 e T0, ou seja, o contador opera a cada transição de 1 para 0 (borda de descida ↓) na entrada T0 ou T1.

Deve-se ter um cuidado quando em operações em modo contador: já se sabe que as entradas são amostradas durante S5P2 de cada ciclo de máquina (figura 3.5). Quando em um ciclo de máquina a entrada é detectada em 1 e depois em 0, no próximo ciclo o contador é incrementado. O incremento acontece em S3P1 do ciclo seguinte ao que foi detectada a transição. Para garantir que o nível correto tenha sido amostrado, é necessário que o sinal de entrada (o sinal que vai acionar os contadores) permaneça pelo menos um ciclo de máquina em nível alto e pelo menos outro ciclo de máquina em nível baixo, quer dizer, a máxima frequência que responderá o contador é de 1/24 da frequência de clock. O sinal de entrada pode ter qualquer "duty cycle", desde que se respeite a restrição antes mencionada.

8.2. REGISTROS ENVOLVIDOS

Dois registros SFR são utilizados para controlar as funções e operações do temporizador/contador: em **TMOD** especifica-se o modo de operação e em **TCON** controla-se a operação. A figura 8.1 ilustra o registro **TMOD** e a figura 8.2 ilustra o registro **TCON**.

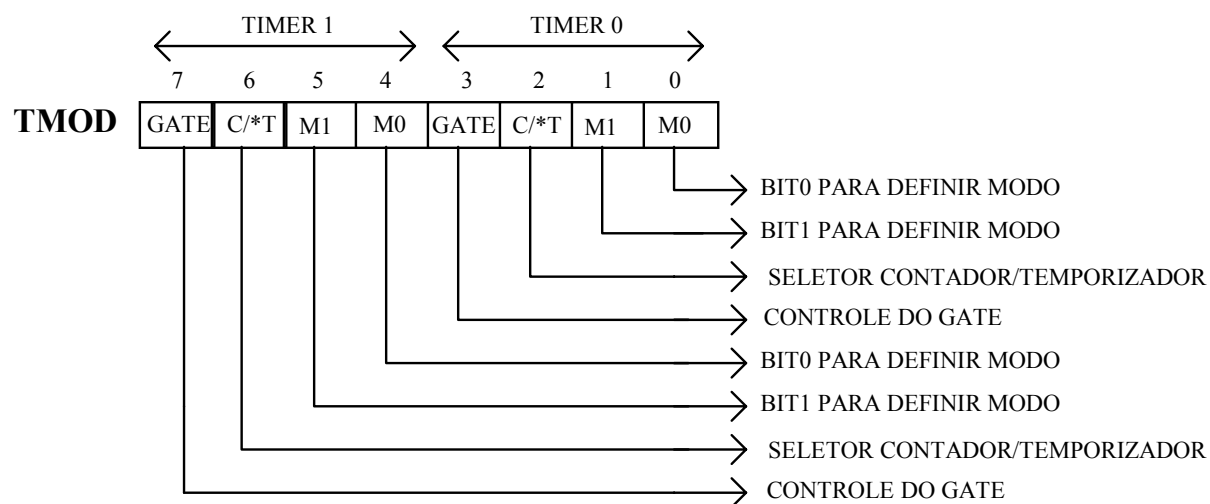


Figura 8.1. Descrição do registro TMOD.

GATE → especifica como será feito o controle:

se GATE = 1 → conta somente se TR1=1 e INT1=high, (idem para TR0 e INT0)

se GATE = 0 → conta somente se TR1=1 (controle somente por software)

(GATE especifica se INT1 será usado para controlar o funcionamento do

contador/temporizador → pode ser usado para medir a largura de pulsos externos ligados a INT0 ou INT1)

C/*T → seleciona modo contador ou temporizador:

se C/*T = 1 → modo contador (conta usando a entrada T1)

se C/*T = 0 → modo temporizador (conta a cada ciclo de máquina)

M1 M0 → seleciona o modo de operação:

0 0 → THi é temporizador/contador de 8 bits e TLi é um pre-scaler de 5 bits,

0 1 → THi e TLi formam um temporizador/contador de 16 bits,

1 0 → contador/temporizador de 8 bits com auto-recarga (TLi conta e THi valor para recarga),

1 1 → TL0 contador/temporizador de 8 bits (usando TR0, *INT0 e TF0)

→ TH0 contador/temporizador de 8 bits (usando TR1, *INT1 e TF1)

→ TH1 e TL1 parado (mas pode operar em outros modos)

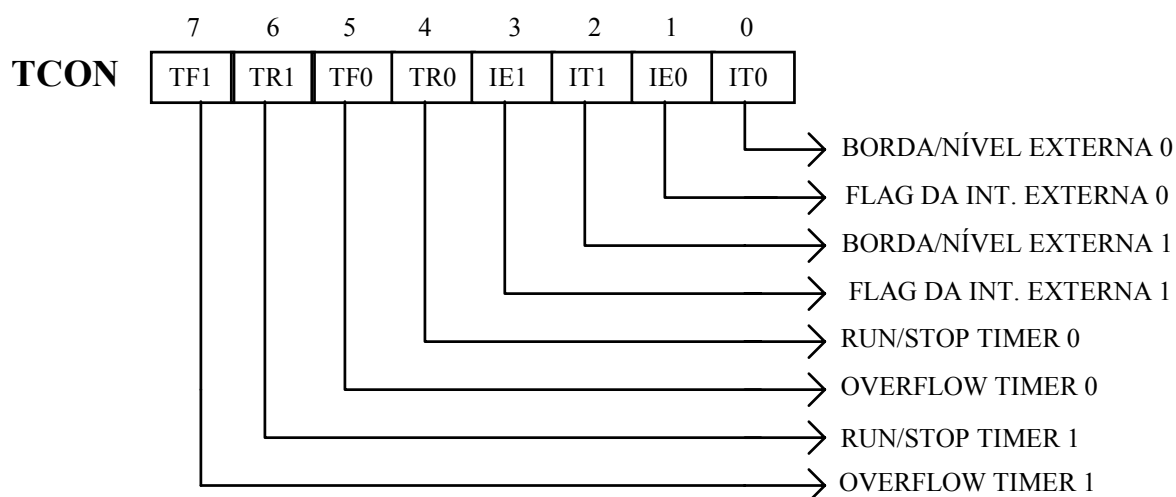


Figura 8.2. Descrição do registro TCON.

TF1 → flag de transbordamento (overflow) do contador/temporizador 1. Ativado por hardware quando há transbordamento no timer 1. É apagado por hardware quando o processamento é desviado para a rotina de interrupção.

TR1 → bit de partida/parada (run/stop) do contador/temporizador 1.

TF0 → flag de transbordamento (overflow) do contador/temporizador 0. Ativado por hardware quando há transbordamento no timer 0. É apagado por hardware quando o processamento é desviado para a rotina de interrupção.

TR0 → bit de partida/parada (run/stop) do contador/temporizador 0.

IE1 → flag da interrupção externa 1. Ativada por hardware quando é detectada uma interrupção. Apagado por hardware (somente se for modo borda) quando a interrupção é processada, ou seja, quando se desvia para a rotina de interrupção.

IT1 → indica se a interrupção externa 1 opera por borda ou por nível:

1 → borda de descida (↓),

0 → nível baixo.

IE0 → flag da interrupção externa 0. Ativada por hardware quando é detectada uma interrupção. Apagado por hardware (somente se for modo borda) quando a interrupção é processada, ou seja, quando se desvia para a rotina de interrupção.

IT0 → indica se a interrupção externa 0 opera por borda ou por nível:

1 → borda de descida (↓),

0 → nível baixo.

8.3. Modos de operação

Os dois contadores/temporizadores (timer1 e timer0) podem trabalhar em 4 modos de operação que são selecionados empregando os bits M1 e M0 do registro TMOD. Os modos 0, 1 e 2 são iguais para os 2 contadores/temporizadores mas o modo 3 é diferente.

8.3.1. Modo 0

Este modo é idêntico para os dois contadores/temporizadores. Neste modo tem-se um contador de 8 bits com um divisor (pre-scaler) de 5 bits. Resulta então em um contador/temporizador de 13 bits, compatível com o que havia no MCS-48. Os 13 bits são formados pelos 8 bits do registro TH1 e pelos 5 bits menos significativos do registro TL1. Os outros 3 bits do TL1 são indeterminados. O transbordamento (overflow) é gerador quando a contagem faz a transição de 1FFFh para 0000; neste instante ativa-se o bit de overflow (TF1 ou TF0).

O controle da contagem é simples:

- se GATE = 0, TR1 controla o contador/temporizador (controle por software)
- se GATE = 1, TR1 e INT1 controlam o contador/temporizador
(permite também um controle externo por hardware)

A figura 8.3 apresenta um diagrama em blocos do contador/temporizador 1 operando em modo 0. A mesma figura é válida para o contador/temporizador 0.

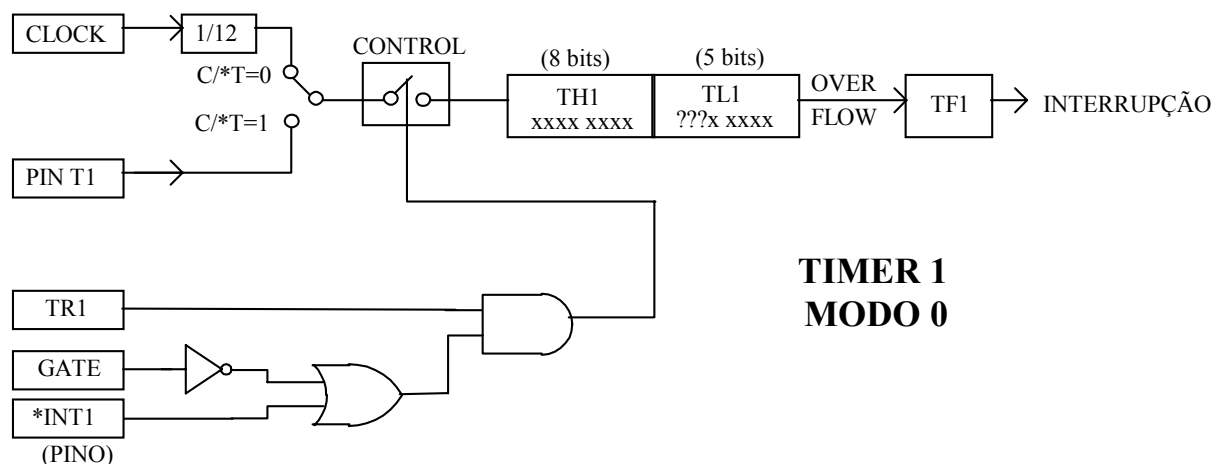


Figura 8.3. Diagrama em blocos para o TIMER 1 em MODO 0.

8.3.2. Modo 1

Este modo de operação é o mais simples e por isto é muito utilizado. É idêntico ao modo 0, mas os contadores são de 16 bits. A figura 8.4. ilustra o diagrama em blocos para este modo, que é idêntico para os dois timers.

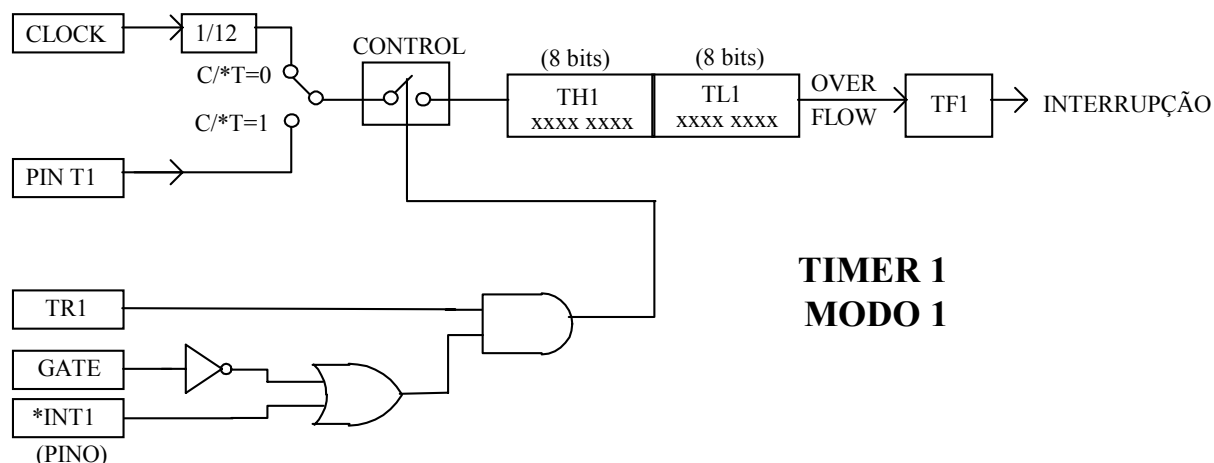


Figura 8.4. Diagrama em blocos para o TIMER 1 em MODO 1.

8.3.3. Modo 2

Neste modo tem-se um contador/temporizador de 8 bits (TLi) com registro de recarga (THi) para quando ocorre o transbordamento. Neste modo os dois contadores/temporizadores operam de forma idêntica. O transbordamento (overflow) não somente ativa TF_i como também recarrega TL_i com o valor guardado em TH_i (este permanece inalterado). O valor de recarga deve ser fornecido por software.

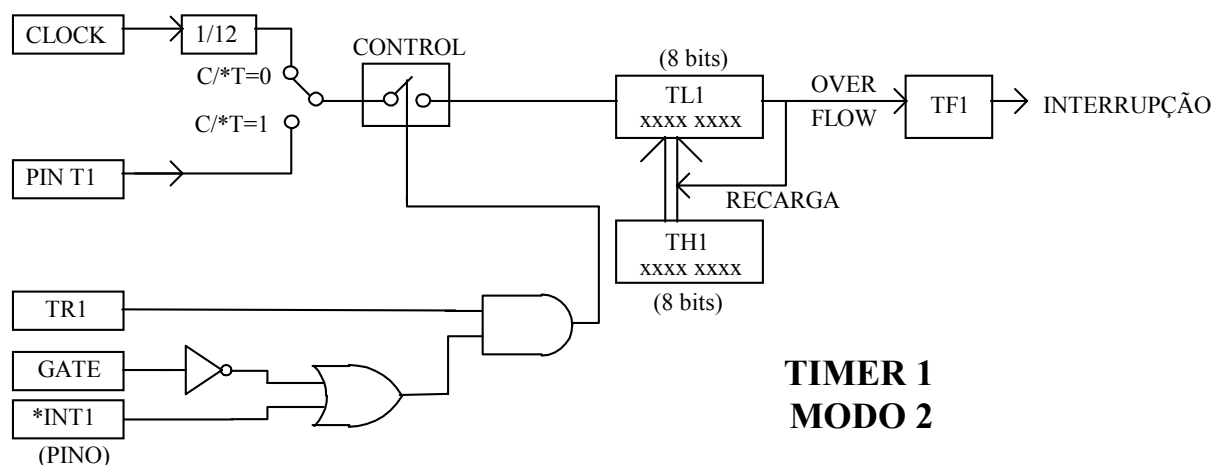


Figura 8.5. Diagrama em blocos para o TIMER 1 em MODO 2.

8.3.4. Modo 3

Este é o único modo onde os contadores/temporizadores têm um comportamento diferente. Neste modo, o contador/temporizador 1 simplesmente suspende a contagem (é como se estivesse com TR₁=0), enquanto o contador/temporizador 0 se divide em dois contadores de 8 bits:

TH₀ → contador/temporizador de 8 bits usando C/*T, GATE, TR₀, *INT₀ e TF₀,

TL₀ → contador/temporizador de 8 bits usando TR₁ e TF₁ (ou seja, provoca a interrupção do timer 1).

É como se existem 2 contadores/temporizadores de 8 bits (TH0 e TL0) e outro de 16 bits (timer 1). O timer 1, entretanto, pode ser usado para operar em qualquer outro modo (0, 1 ou 2) e usa-se o modo 3 para detê-lo. Não há problemas porque os modos são independentes. Também se pode usar o timer 1 (em modo 2) para gerar o "baud rate" da porta serial.

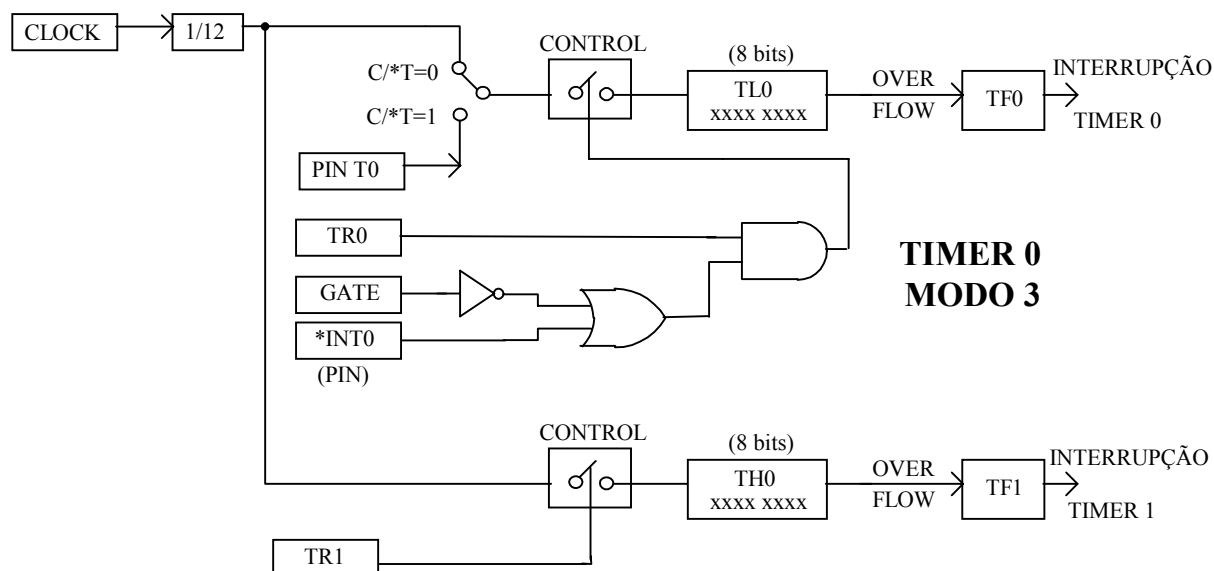
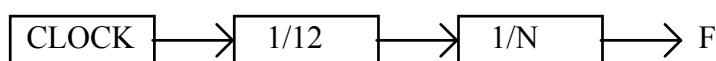


Figura 8.6. Diagrama em blocos para o TIMER 0 em MODO 3.

8.4. EXERCÍCIOS

EXERCÍCIO 8.1. LED_10HZ

Acender os leds vermelho, amarelo e verde a uma frequência de 10 Hz. Para solucionar esse exercício é necessário gerar um retardo equivalente ao período de 10 Hz. A figura 8.7 ilustra o cálculo do valor do contador para gerar esse retardo.



$$F = \frac{\text{CLOCK}}{12 \cdot N} \quad N = \frac{\text{CLOCK}}{12 \cdot F}$$

Figura 8.7. Cálculo de N para gerar uma determinada frequência F.

Se $F=10$ Hz e $\text{CLOCK}=3,575611$ Hz, então $N=29797$. Como este é um contador que conta para cima, temos que subtrair este valor de 65536. Com isto, $N=65536-29797=35739$ ou 8B9BH. Na realidade, a frequência gerada não será exatamente 10 Hz mas de um valor muito próximo:

$$F = 3,575611 / 12 \cdot 29797 = 9,99 \text{ Hz}$$

Será utilizado o timer 0 no modo 1; a cada interrupção a rotina acende um novo led e recarrega o timer.

TMOD	GATE	C/*T	M1	M0	GATE	C/*T	M1	M0
	0	0	0	0	0	0	0	1

Figura 8.8. Valor a ser programado em TMOD.

IE	EA	-	-	ES	ET1	EX1	ET0	EX0
	1	0	0	0	0	0	1	0

Figura 8.9. Valor a ser programado em IE.

```

;LED_10HZ.ASM
;
                DEFSEG PROG, CLASS=CODE, START=0
                SEG          PROG
;
DEZ_HZ          EQU          35739
;
                ORG          RESET
                AJMP         INIC
;
                ORG          TIMER0
                AJMP         TIM0
;
                ORG          50H
INIC             MOV         TL0,#LOW  DEZ_HZ
                MOV         TH0,#HIGH DEZ_HZ
                MOV         TMOD,#1    ;TIMER 0 EM MODO 1
                MOV         IE,#82H
                MOV         P1,#0      ;APAGAR TODOS OS LEDS
                MOV         A,#01001001B
                CLR         C
                SETB        TR0
                SJMP        $
;
                ORG          100H
TIM0             MOV         TL0,#LOW  DEZ_HZ ;REINICALIZAR
                MOV         TH0,#HIGH DEZ_HZ ;O CONTADOR
                RLC         A
                MOV         P1,A
                RETI
                END

```

Notar que na rotina de interrupção está a recarga dos temporizadores que é feita com a instrução "MOV direto,#data" mas que consome 2 ciclos de máquina, ou seja, que há um retardo que se acumula. Para evitar ou corrigir isso, pode-se adicionar ao contador o retardo da instrução; é mais preciso usar "MOV TL0,# LOW (DEZ_HZ+2).

Quando existem várias interrupções, não se sabe com certeza quando uma interrupção do temporizador pode ser aceita; isso implica um erro maior que se acumula. Há uma solução simples porque depois do transbordamento o temporizador segue contando e esta contagem é exatamente o tempo que atrasou o serviço da interrupção; basta usar esse valor na recarga. As instruções a seguir ilustram a idéia.

```

MOV      A, #LOW  (DEZ_HZ+2)

ADD      A, TL0                      ;1 ciclo

MOV      TL0, A                      ;1 ciclo

MOV      TH0, #HIGH (DEZ_HZ+2)

```

EXERCÍCIO 8.2. LED_1HZ

Acender os leds vermelho, amarelo e verde a uma frequência de 1 Hz. A solução é idêntica ao exercício anterior, mas vai surgir uma dificuldade ao calcular os valores de recarga:

$$\text{CLOCK}=3575611 \text{ Hz e } F=1 \text{ Hz}$$

$$N=3575611 / 12*1 = 297968$$

Como se pode ver, o valor de recarga é muito grande ($297\ 968 > 65\ 536$) e impede a utilização do temporizador. Uma boa saída é utilizar o esquema do exercício anterior (com 10 Hz) e adicionar, por software, um divisor por 10.

```

;LED_1HZ.ASM
;
                DEFSEG PROG, CLASS=CODE, START=0
                SEG          PROG
;
DEZ_HZ          EQU          32203
DIVISOR         EQU          10
;
                ORG          RESET
                AJMP         INIC
;
                ORG          TIMER0
                AJMP         TIM0
;
INIC            ORG          50H
                MOV          TL0, #LOW  DEZ_HZ
                MOV          TH0, #HIGH DEZ_HZ
                MOV          R7, #DIVISOR ;PREPARAR DIVISAO POR 10
                MOV          TMOD, #1    ;TIMER 0 EM MODO 1
                MOV          IE, #82H
                MOV          P1, #0      ;APAGAR TODOS OS LEDS
                MOV          A, #01001001B
                CLR          C
                SETB         TR0
                SJMP         $
;
TIM0            ORG          100H
                MOV          TL0, #LOW  DEZ_HZ ;REINICALIZAR
                MOV          TH0, #HIGH DEZ_HZ ;O CONTADOR
                DJNZ         R7, FIM        ;DIVIDIR POR 10
                MOV          R7, #DIVISOR
                RLC          A
                MOV          P1, A
FIM             RETI
                END

```


EXERCÍCIO 8.3. ONDA1

Gerar através de P1.7 uma onda com o seguinte formato:

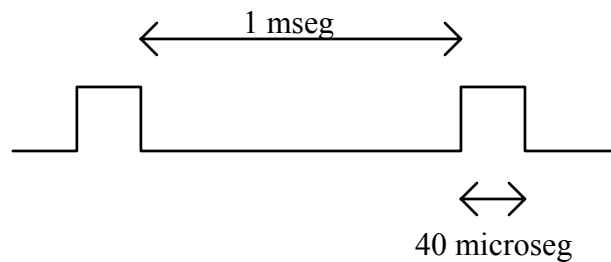


Figura 8.10. Sinal a ser gerado pela saída P1.7.

Usando $N = (t \cdot \text{CLOCK})/12$, calcula-se o valor de recarga para o temporizador. Serão programados dois retardos alternadamente de forma a gerar o sinal pedido.

Para $t=1 \text{ ms} \rightarrow N=298 \rightarrow 65536-298=65238$

Para $t=40 \mu\text{s} \rightarrow N=12 \rightarrow 65536-12=65524$

```
;ONDA1.ASM
                DEFSEG PROG, CLASS=CODE, START=0
                SEG  PROG
;
SAIDA           EQU            P1.7
R_1MS           EQU            65238          ;RETARDO DE 1 MILISEG
R_40MICRO       EQU            65524          ;RETARDO DE 40 MICROSEG
;
                ORG            RESET
                AJMP           INICIO
;
                ORG            TIMER0
                AJMP           TIM0
;
INICIO          ORG            50H
                MOV            TL0,#LOW  R_1MS
                MOV            TH0,#HIGH R_1MS
                MOV            TMOD,#1      ;TIMER 0 EM MODO 1
                MOV            IE,#82H      ;EA=1 E ET0=1
                CLR            SAIDA
                SETB            TR0          ;PARTIDA DO TIMER 0
                SJMP           $            ;LOOP INFINITO
;
TIM0            JB            SAIDA,LB1      ;2 CICLOS
                MOV            TH0,#LOW  (R_40MICRO+4) ;2 CICLOS
                MOV            TL0,#HIGH (R_40MICRO+4) ;2 CICLOS
                SETB            SAIDA        ;1 CICLO
                RETI                    ;2 CICLOS
LB1             MOV            TH0,#LOW  (R_1MS+4)
                MOV            TL0,#HIGH (R_1MS+4)
                CLR            SAIDA
                RETI
                END
```

Notar que a rotina que atualiza o temporizador para gerar o retardo de $40 \mu\text{s}$ consome 9 ciclos de máquina, quer dizer, consome $30,2 \mu\text{s}$. Pode parecer que se o tempo de $40 \mu\text{s}$ fosse mudado para $25 \mu\text{s}$ não haveria solução.

EXERCÍCIO 8.4. ONDA2

Gerar através de P1.7 uma onda com o seguinte formato:

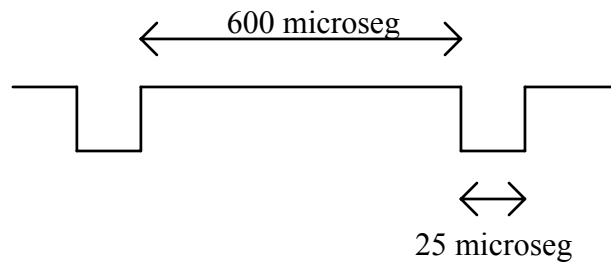


Figura 8.11. Sinal a ser gerado pela saída P1.7.

Usando $N = (t \cdot \text{CLOCK})/12$, calcula-se o valor de recarga para o temporizador. O segredo será trabalhar com o temporizador em modo 2 (os dois valores de recarga são menores que 256) e a rotina de interrupção apenas muda o valor da recarga (que está em TH0).

Para $t=600 \mu\text{seg} \rightarrow N=179 \rightarrow 256-179=77$

Para $t=25 \mu\text{seg} \rightarrow N=7 \rightarrow 256-7 =249$

```
;ONDA2.ASM
                DEFSEG PROG, CLASS=CODE, START=0
                SEG  PROG

;
SAIDA           EQU            P1.7
R_600MICRO     EQU            77           ;RETARDO DE 600 MICROSEG
R_25MICRO      EQU            249          ;RETARDO DE 25 MICROSEG
;
                ORG            RESET
                AJMP           INICIO

;
                ORG            TIMER0
                AJMP           TIM0

;
INICIO          ORG            50H
                MOV            TL0,#R_600MICRO ;PRIMEIRA CONTAGEM
                MOV            TH0,#R_25MICRO ;CONTAGEM SEGUINTE
                MOV            TMOD,#2         ;TIMER 0 EM MODO 2
                MOV            IE,#82H         ;EA=1 E ET0=1
                CLR            SAIDA
                SETB            TR0             ;PARTIDA DO TIMER 0
                SJMP           $               ;LOOP INFINITO

;
TIM0            JB            SAIDA,LB1         ;2 CICLOS
                MOV            TH0,#R_25MICRO ;2 CICLOS
                SETB            SAIDA           ;1 CICLOS
                RETI                     ;2 CICLOS
LB1             MOV            TH0,#R_600MICRO
                CLR            SAIDA
                RETI
                END
```

EXERCÍCIO 8.5. LEDS1

Acender os leds vermelho, amarelo e verde em seqüência, mudando a cada 10 pulsos em T1. Será usado o contador/temporizador 1 como contador e programado para operar em modo 2, com um valor de recarga igual a 246 (256-10). O acumulador e carry serão usados para acender os leds na seqüência correta

TMOD	GATE	C/*T	M1	M0	GATE	C/*T	M1	M0
	0	1	1	0	0	0	0	01

Figura 8.12. Valor a ser programado em TMOD (contador 1, modo 2).

IE	EA	-	-	ES	ET1	EX1	ET0	EX0
	1	0	0	0	1	0	0	0

Figura 8.9. Valor a ser programado em IE.

```

;LEDS1.ASM
DEFSEG PROG, CLASS=CODE, START=0
SEG PROG

;
CONTA_10 EQU 246
;
ORG RESET
AJMP INICIO

;
ORG TIMER1
AJMP TIM1

;
ORG 50H
INICIO MOV A, #01001001B
CLR C
MOV P1, A
MOV TL1, #CONTA_10 ;CARREGAR CONTADOR
MOV TH1, #CONTA_10 ;VALOR DE RECARGA
MOV TMOD, #60H ;CONTADOR 1, MODO 2
MOV IE, #88H ;EA=1, ET1=1
SETB TR1 ;LIGAR CONTADOR
SJMP $ ;LOOP INFINITO

;
TIM1 RLC A
MOV P1, A
RETI
END

```

A chave SW3 aciona diretamente a entrada T1, mas há bouncing e por isso serão notadas mudanças nos leds antes de 10 acionamentos. Para esse caso específico o bouncing deverá ser eliminado por hardware.

CAPÍTULO IX

PORTA SERIAL

9.1. INTRODUÇÃO

A porta serial existente na família MCS-51 é "full duplex", quer dizer, pode transmitir e receber dados simultaneamente. Tem um buffer que permite receber um segundo byte antes que o byte previamente recebido tenha sido retirado (lido) do registro de recepção. Mas, se o primeiro byte não tiver sido lido no tempo em que o segundo byte se completa, um dos dois será perdido.

9.2. REGISTROS ENVOLVIDOS

A porta serial possui um registrador chamado SBUF, o mesmo que se usa para enviar ou receber dados pela porta serial. Na realidade, o nome SBUF se refere a dois registros, um somente para leitura por onde se recebem os dados que chegam pela porta serial e outro somente para escrita por onde se transmitem dados pela porta serial.

O modo de operação da porta serial é controlado pelo registro **SCON**, que é ilustrado na figura 9.1. A figura 9.2 apresenta um resumo dos modos de operação.

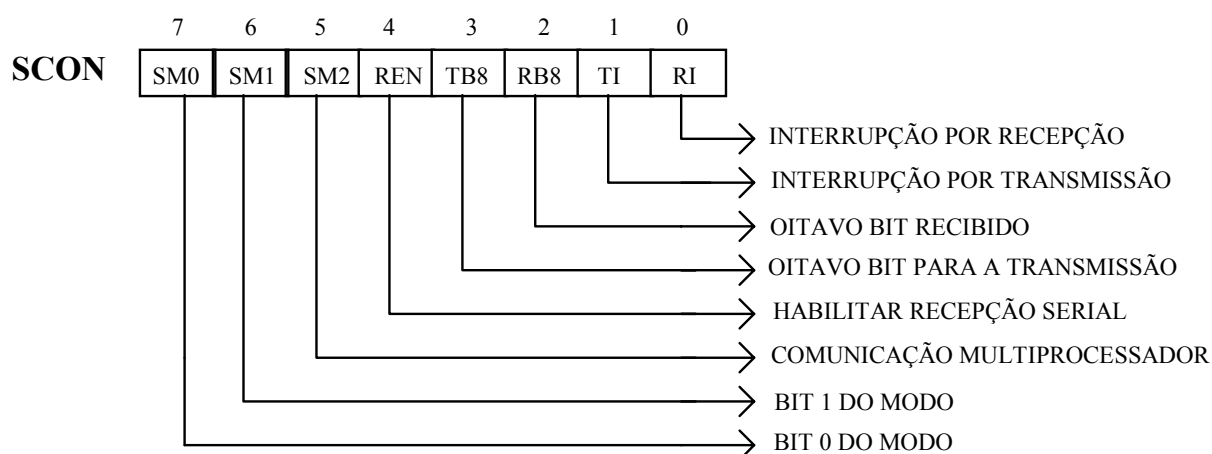


Figura 9.1. Descrição do registro SCON.

SM0	SM1	MODO	DESCRIÇÃO	FREQÜÊNCIA
0	0	0	registro de deslocamento	clock/12
0	1	1	UART de 8 bits	variável
1	0	2	UART de 9 bits	clock/12 ou clock/64
1	1	3	UART de 9 bits	variável

Figura 9.2. Modos de operação da porta serial.

SM2 → comunicação multiprocessador (habilitada com SM2=1):

SM2=1 e em modo 1 ==> interrupção (RI=1) com o bit de parada (stop bit) igual a 1

SM2=1 e em modo 2 ou 3 ==> interrupção (RI=1) se for recebido RB8=1.

REN → habilita a recepção serial (Reception Enable)

TB8 → oitavo bit a ser transmitido nos modos 2 e 3.

RB8 → oitavo bit recebido nos modos 2 e 3.

TI → flag de interrupção por término de transmissão pela porta serial:

MODO 0 → ativado no final do oitavo bit

DEMAIS → ativado no começo do bit de parada

RI → flag de interrupção por recepção pela porta serial

MODO 0 → ativado no final do oitavo bit

DEMAIS → ativado na metade do bit de parada

Observação: os flags TI e RI não são apagados por hardware e por isso a rotina de interrupção deve fazê-lo, ou seja, estes flags devem ser zerados por software.

O registro PCON também toma parte na geração do baud rate da porta serial. Este registro é ilustrado na figura 9.2.

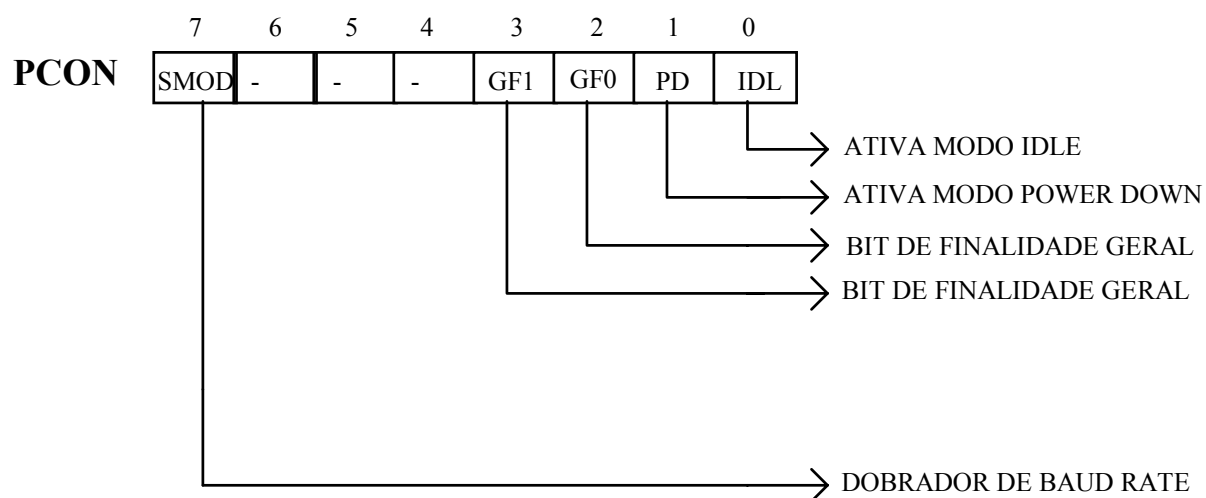


Figura 9.3. Descrição do registro PCON.

9.3. MODOS DE OPERAÇÃO

Existem quatro modos de operação da porta serial, cada um com uma finalidade. Será apresentado um resumo de cada modo:

9.3.1. Modo 0 (síncrono, 8 bits)

Neste modo os dados entram e saem pelo pino RXD. O pino TXD fornece o clock para o deslocamento (shift). São transmitidos e recebidos dados de 8 bits (LSB primeiro). O baud rate é de 1/12 da frequência do clock (igual a 1 ciclo de máquina).

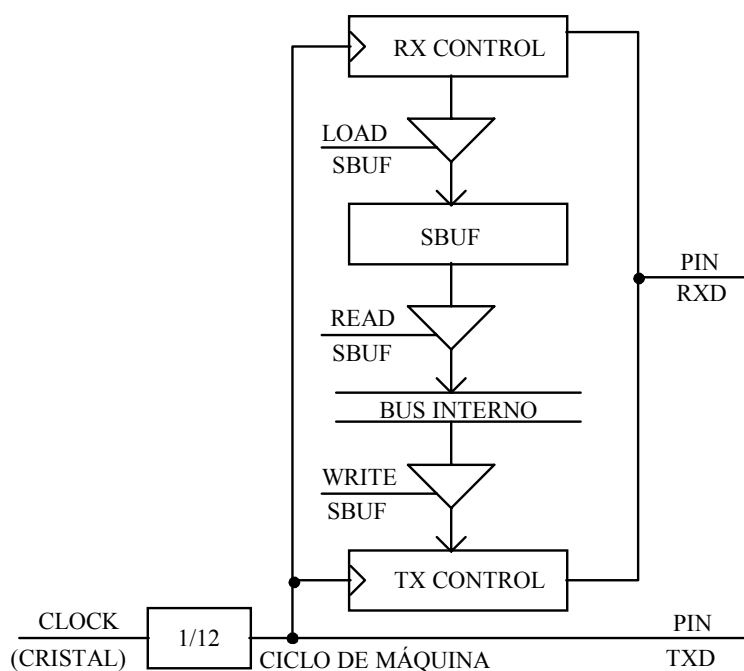
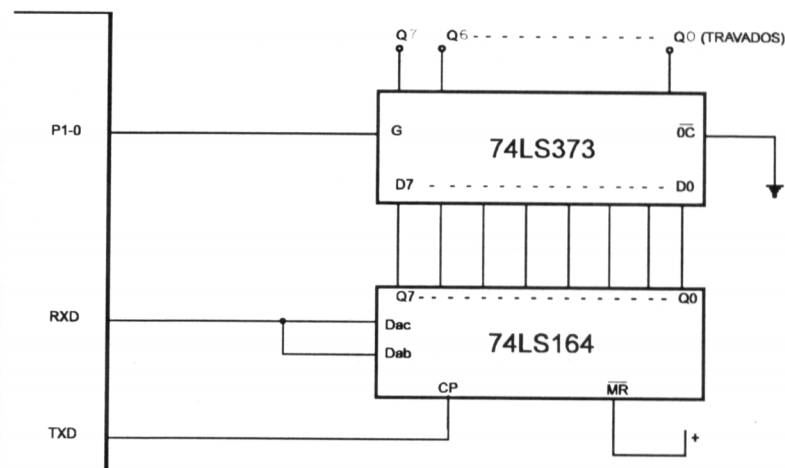


Figura 9.4. Esquema da porta serial em modo 0.

Este modo pode ser usado para expandir as portas de I/O do microcontrolador usando um esquema como o da figura a seguir:



9.3.2. Modo 1 (assíncrono, 8 bits)

No Modo 1 são operados 10 bits com um baud rate programável. Recebe-se pelo pino RXD e transmite-se pelo pino TXD. A figura 9.5 ilustra o frame de bits serial. O bit de partida (START) é sempre zero e o bit de parada (STOP) é sempre um na transmissão. Na recepção o bit de parada é colocado no bit RB8.

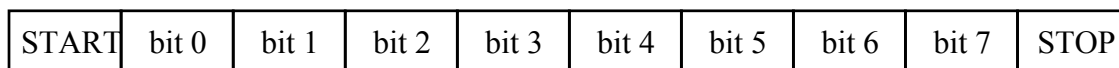


Figura 9.5. Frame de bits gerado pela porta serial em modo 1.

Neste modo o baud rate é gerado pelo contador/temporizador 1. A cada 16 (SMOD=1) ou 32 (SMOD=0) transbordamentos (overflows) é enviado um pulso para o circuito serial. A figura 9.6 ilustra o esquema do gerador do baud rate e também apresenta algumas fórmulas para o cálculo do valor a ser programado no contador/temporizador.

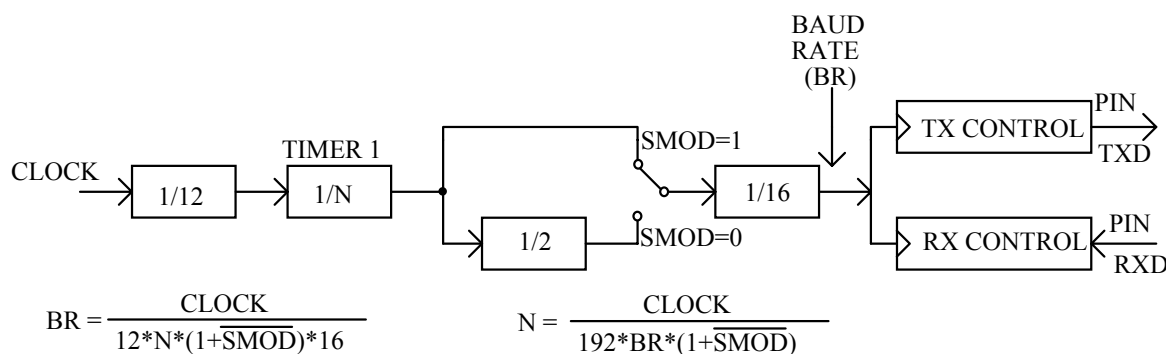


Figura 9.6. Esquema para a geração do baud rate (BR) no modo 1.

Usando o contador/temporizador 1 programa-se qualquer baud rate. Na grande maioria dos casos tem-se o baud rate e é buscado o valor N a ser programado no contador/temporizador 1 (o mais cômodo é usar o modo 2 - auto recarga). Um problema que sempre existe é o erro gerado pela aproximação do valor a ser programado no contador/temporizador. Isto provoca a geração de baud rates ligeiramente diferentes.

Exemplo: Usando um cristal de 3,575611 MHz, qual é o valor a ser programado no timer 1 para operar a 9600 bauds (usar SMOD=0) ?

$$N = 3575611 / (384 \cdot 9600) = 0,9699 \text{ (aproximadamente 1)}$$

Usando timer 1 no modo 2 tem-se: $256 - 1 = 255$ (TH1=TL1=255).

Observação: a transferência do dado recebido para o registro SBUF (e RB8, que guarda o stop bit) e a ativação do flag RI somente acontecerá se:

RI = 0 e (SM2= 0 ou o bit de parada =1).

9.3.3. Modo 2 (assíncrono, 9 bits, baud rate fixo)

Este modo opera com 11 bits e o baud rate pode ser de 1/32 ou 1/64 do clock. O bit TB8/RB8 serve para transmitir a paridade ou gerar um segundo bit de parada (TB8=1). O frame de bits usado neste modo é ilustrado na figura 9.7.

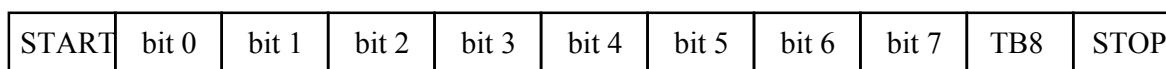


Figura 9.7. Frame de bits gerado pela porta serial no modo 2.

O único controle que se tem sobre o baud rate é através do uso do bit SMOD. Se SMOD=0, trabalha-se com BR=clock/32 e se SMOD=1, trabalha-se com BR=clock/64. A figura 9.8 ilustra o esquema para geração do baud rate no modo 2.

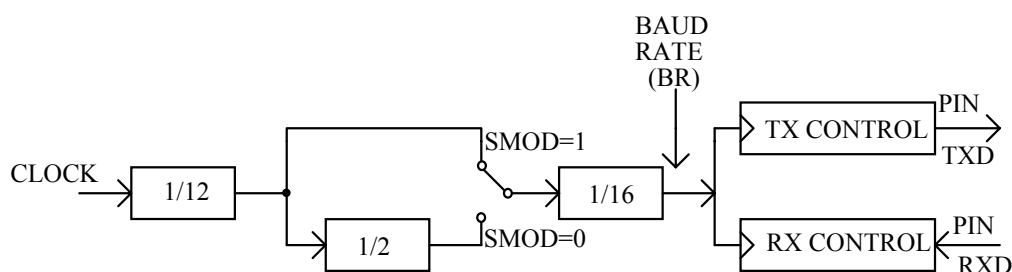


Figura 9.8. Esquema para a geração do baud rate (BR) no modo 2.

Na transmissão o bit 8 é copiado do bit TB8. Na recepção o bit 8 é copiado para o bit RB8. Os bits TB8 e RB8 estão no registro SCON.

O dado recebido somente é carregado no SBUF (e RB8) se:

RI=0 e (SM2=0 ou bit 9 (stop bit) = 1).

9.3.4. Modo 3 (assíncrono, 9bits, baud rate variável)

É idêntico ao modo 2, exceto que a geração do baud rate é idêntico ao modo1. A figura 9.9 ilustra o frame de bits e a geração do baud rate.

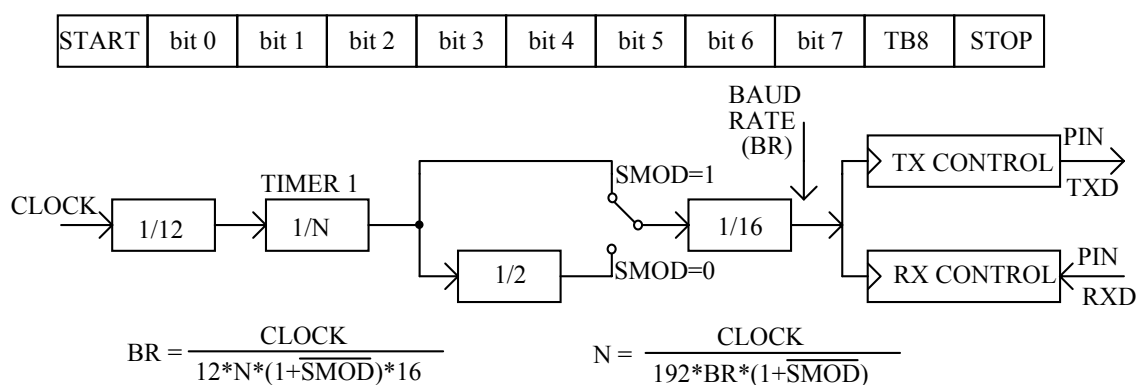


Figura 9.9. Frame de bits e geração do baud rate para a porta serial no modo 3.

9.4. CUIDADOS COM A PORTA SERIAL

Nos quatro modos a transmissão se inicia quando é escrito um byte em SBUF. A recepção é habilitada quando REN=1 (no MODO 0 exige-se também RI=0). Não se esqueça de que somente existe uma única interrupção dedicada à porta serial, portanto ela é gerada por duas condições:

- Término da transmissão de um byte → flag TI
- Término da recepção de um byte → flag RI

Deve-se testar os flags RI e TI para determinar se a interrupção foi por transmissão ou por recepção. Os flags RI e TI devem ser apagados por software.

Um outro cuidado muito importante é com a geração do baud rate. Muitas vezes não se consegue a comunicação serial devido à imprecisão do baud rate. Portanto o problema a seguir é proposto: Programar o contador/temporizador 1 para gerar 9600 bauds para a porta serial operando em modo 3, sendo que o cristal é de 4 MHz.

$$\text{Solução: } N = 4000000 / (384 * 9600) = 1,085 \quad (\text{SMOD}=0).$$

Na solução calculou-se que $N=1,085$ mas somente podem ser programados números inteiros; assim, caso se aproxime para 1, o baud rate gerado será de $\text{BR}=4000000/384=10416,7$. Será que irá funcionar bem com esse baud rate ?

Para responder esta pergunta é necessário compreender o que acontece com a transmissão serial. Esta transmissão é assíncrona, quer dizer, pode iniciar em qualquer instante; o início é caracterizado pela presença de um bit de partida (START). Uma vez iniciada, deve-se garantir a duração de cada bit. Para cada byte é transmitido um bit de partida, os bits de dados e um ou dois bits de parada. O que se deve buscar é garantir que não haja um erro muito grande neste frame de bits. A figura 9.10 ilustra o caso da porta serial operando em modo 3.

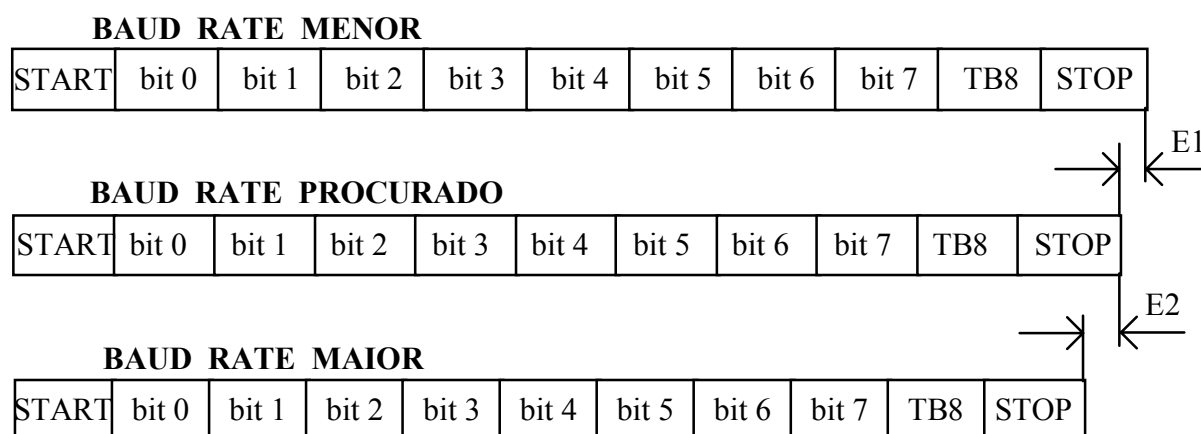


Figura 9.10. Erros provocados por diferenças no baud rate.

A fase mais crítica é a recepção do último bit porque poderá apresentar uma defasagem muito grande (rotuladas de E1 e E2 na figura). Não se pode especificar um valor limite para esses

erros pois diversos fatores, particulares para cada caso, precisam ser levados em consideração. Como um valor prático e que funciona na grande maioria dos casos arbitra-se que o valor do erro, para o último bit, deve ser menor que 40% da duração de um bit (usando BR exato).

$$\left| \frac{11}{BRp} - \frac{11}{BR} \right| \leq \frac{40}{100} * \frac{1}{BR}$$

BRp → baud rate programado

BR → baud rate exato

A equação pode ser simplificada para :

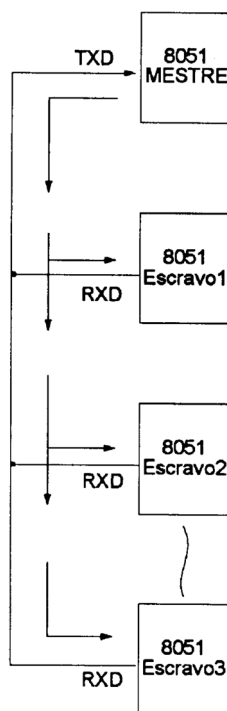
$$\left| \frac{1}{BRp} - \frac{1}{BR} \right| \leq \frac{1}{27,5 * BR}$$

Como pode ser verificado, com um cristal de 4 MHz não se conseguirá transmitir a 9600 pois o erro do baud rate será muito grande.

$$\left| \frac{1}{10416,7} - \frac{1}{9600} \right| \leq \frac{1}{27,5 * 9600}$$

8,167 μs ≤ 3,788 μs → falso.

9.5. Comunicação entre vários 8051



Os modos 2 e 3 permitem interligar vários 8051, sendo um mestre e vários escravos. Nestes modos temos:

- 1 start bit;
- 8 bits de dados;
- um nono bit que vai para o bit RB8 (na recepção) ou pode ser escolhido 0 ou 1 na transmissão escrevendo-se em TB8;
- 1 stop bit

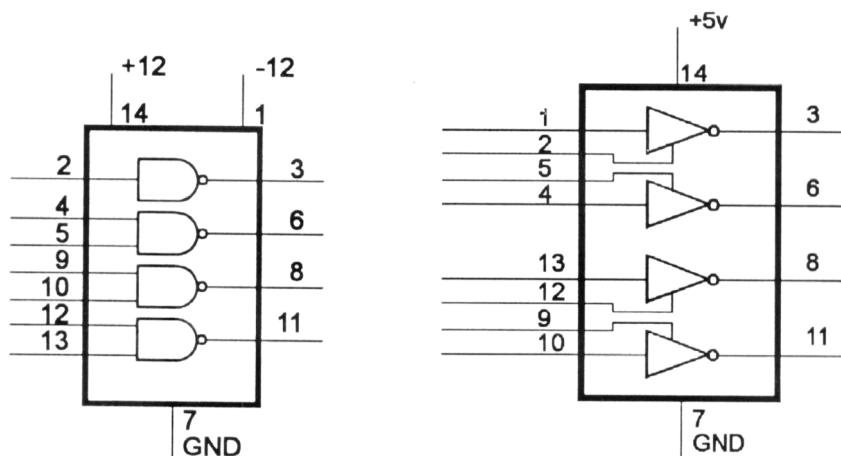
Note que se SM2 = 1 e RB8 = 1, a interrupção da serial será atendida.

O algoritmo de comunicação consiste em:

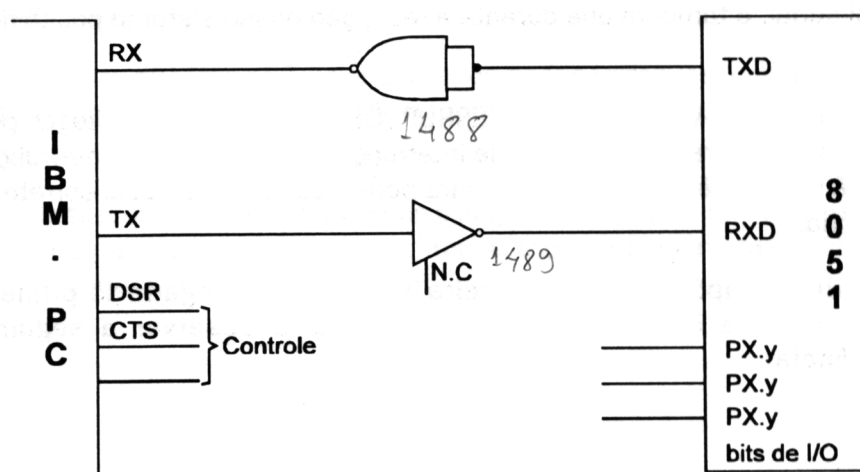
- 1) No início, todos os escravos estão com SM2 = 1
- 2) Quando o mestre quiser enviar dados para algum escravo, ele escreverá 1 em seu bit TB8 e então enviará serialmente o endereço do escravo desejado, e como teremos todos os bits RB8 em 1, **todos** escravos serão interrompidos para verificar se é seu o endereço enviado.
- 3) O escravo selecionado zerará o seu bit SM2 e estará preparado para receber os dados, os quais terão agora o nono bit (RB8) em 0.
- 4) Os demais escravos permanecerão com SM2 em 1 e, dessa forma, não serão mais interrompidos pois os dados têm RB8 = 0.
- 5) Em resumo, o mestre envia endereços com o nono bit em 1 e os dados com o nono bit em 0, portanto, se o mestre desejar se comunicar com outro escravo basta enviar o novo endereço com o nono bit em 1.

9.6. Comunicação serial entre o 8051 e o PC

Devido à diferença de tensão entre os níveis TTL e os usados no padrão RS-232C (PC), é necessário usar circuitos conversores para interfacear o microcontrolador com a porta serial do PC. A conversão pode ser feita através de circuitos transistorizados discretos (veja as figuras 11.10, 11.11 e 11.12 e leia a seção 11.4) ou através de circuitos integrados como o 1488 (conversor de TTL para RS-232C) e o 1489 (conversor de RS-232C para TTL). A seguir estão os esquemas dos dois integrados e o esquema básico de interligação entre o 8051 e a serial do PC.



CIs 1488 (esquerda) e 1489 (direita)



9.7. EXERCÍCIOS

Exercício 9.1. TX_SER

Usando um loop infinito, transmitir pela porta serial todos os caracteres ASCII de "0" a "Z". Usar 9600 bauds, 8 bits de dados, 1 bit de partida e 2 bits de parada.

Para este caso o modo 3 é o mais adequado e TB8=1 será usado para gerar um bit de parada. A transmissão será feita por interrupção, quer dizer, a cada byte transmitido haverá uma interrupção. O contador/temporizador 1 será programado para modo 2 (auto-recarga).

SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
	1	1	0	0	1	0	0	0

TMOD	GATE	C/*T	M1	M0	GATE	C/*T	M1	M0
	0	0	1	0	0	0	0	0

IE	EA	-	-	ES	ET1	EX1	ET0	EX0
	1	-	-	1	0	0	0	0

Figura 9.11. Inicialização de os registros para o exercício 9.1.

Cálculo do divisor formado pelo contador/temporizador 1:

$$N = 3575611/(384*9600) = 0,9699 \rightarrow 1. \quad (TH1 = TL1 = 256 - 1 = 255).$$

```

;TX_SER.ASM
DEFSEG PROG, CLASS=CODE, START=0
SEG          PROG
;
BR_9600      EQU          255
;
ORG          RESET
AJMP         INIC
;
ORG          SINT
AJMP         SERIAL
;
INIC          ORG          50H
MOV          TMOD,#20H      ;TIMER 1 EM MODO 2
MOV          TH1,#BR_9600   ;PROGRAMAR BAUD RATE
MOV          TL1,#BR_9600
SETB         TR1           ;INICIAR TIMER 1
MOV          SCON,#0C8H     ;MODO 3 COM TB8=1
MOV          IE,#90H        ;HAB INTERRUPT SERIAL
MOV          A,#"0"         ;PRIMEIRO ASCII
SETB         TI            ;TX PRIMEIRO ASCII
SJMP         $             ;LOOP INFINITO
;
SERIAL        ORG          100H
CLR          TI            ;APAGAR FLAG
MOV          SBUF,A         ;TRANSMITIR
INC          A
CJNE        A,#"Z"+1,SER1   ;FOI O ULTIMO ?
MOV          A,#"0"         ;REINICIALIZAR
SER1          ACALL        RETARDO ;ATRASAR TRANSMISSAO
RETI
;
RETARDO       MOV          R7,#0
AQUI          DJNZ         R7,AQUI
RET
END

```

Exercício 9.2. RX_SER

O circuito deverá responder aos seguintes comandos que chegam pela porta serial:

- | | |
|---------------------------|--------------------------|
| 1 → acender todos os leds | 0 → apagar todos os leds |
| R → acender led vermelho | r → apagar led vermelho |
| A → acender led amarelo | a → apagar led amarelo |
| V → acender led verde | v → apagar led verde |

Estes comandos vão chegar pela porta serial usando o formato: 1 bit de partida, 8 bits de dados, 2 bits de parada, com um baud rate de 9600.

```
;RX_SER.ASM
                DEFSEG PROG, CLASS=CODE, START=0
                SEG          PROG

;
BR_9600         EQU          255
LED_VERMELHO   EQU          P1.0
LED_AMAR       EQU          P1.1
LED_VERDE      EQU          P1.2
CHEGOU         EQU          32.0
;
                ORG          RESET
                AJMP         INIC
;
                ORG          SINT
                AJMP         SERIAL
;
INIC            ORG          50H
                MOV          TMOD,#20H      ;TIMER 1 EM MODO 2
                MOV          TH1,#BR_9600   ;PROGRAMAR BAUD RATE
                MOV          TL1,#BR_9600
                SETB         TR1            ;INICIAR TIMER 1
                MOV          SCON,#0D0H     ;MODO 3 COM REN=1
                MOV          IE,#90H        ;HAB INTERRUPT SERIAL
                CLR          CHEGOU         ;APAGAR FLAG
                JNB          CHEGOU,ESPERA   ;AGUARDAR UM COMANDO
                CLR          CHEGOU
;
                CJNE         A,#"0",LB1
                CLR          LED_VERMELHO   ;CHEGOU 0
                CLR          LED_AMAR
                CLR          LED_VERDE
                SJMP         ESPERA
;
LB1             CJNE         A,#"1",LB2
                SETB         LED_VERMELHO   ;CHEGOU 1
                SETB         LED_AMAR
                SETB         LED_VERDE
                SJMP         ESPERA
;
LB2             CJNE         A,#"R",LB3
                SETB         LED_VERMELHO   ;CHEGOU R
                SJMP         ESPERA
LB3             CJNE         A,#"r",LB4
                CLR          LED_VERMELHO   ;CHEGOU r
                SJMP         ESPERA
;
LB4             CJNE         A,#"A",LB5
                SETB         LED_AMAR        ;CHEGOU A
                SJMP         ESPERA
LB5             CJNE         A,#"a",LB6
                CLR          LED_AMAR        ;CHEGOU a
                SJMP         ESPERA
;
LB6             CJNE         A,#"V",LB7
                SETB         LED_VERDE      ;CHEGOU V
                SJMP         ESPERA
LB7             CLR          LED_AMAR        ;CHEGOU v
                SJMP         ESPERA
;
SERIAL          CLR          RI              ;APAGAR FLAG DE INTERRUPT
                MOV          A,SBUF          ;COLOCAR DADO NO Acc
                SETB         CHEGOU
                RETI
                END
```

CAPÍTULO X

ECONOMIA DE ENERGIA E GRAVAÇÃO

10.1. INTRODUÇÃO

Para aplicações onde o consumo de potência é crítico, as versões CHMOS oferecem modos de redução de potência como recurso padrão. Há dois modos básicos para economia de energia (redução de consumo):

- Modo Idle
- Modo Power Down

A figura 10.1 apresenta um diagrama de blocos onde se pode ver o funcionamento e a distinção entre os dois modos de economia de energia.

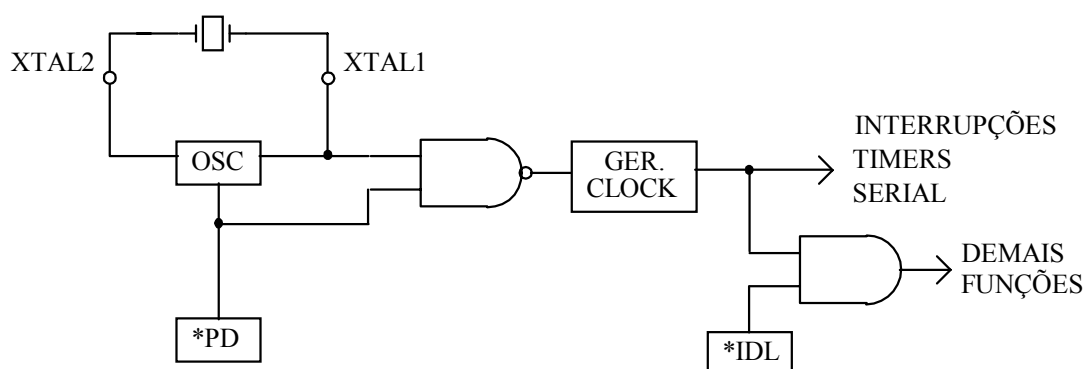


Figura 10.1. Controle executado pelos flags (*PD e *IDL) que habilitam a economia de energia.

No **MODO IDLE** (IDL=1) o oscilador continua a trabalhar para três funções, mas é removido do resto da CPU. As três funções que operam em modo Idle são:

- interrupções
- timers
- porta serial

No **MODO POWER DOWN** (PD=1) o oscilador é paralisado e com isto toda CPU fica congelada.

Esses dois modos são ativados pelo registrador **PCON**, que é descrito na figura 10.2.

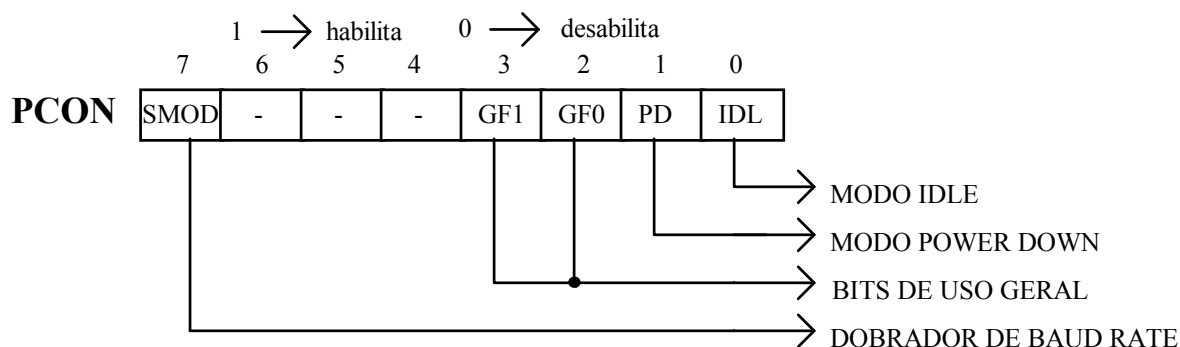


Figura 10.2. Descrição do registro PCON.

Nas versões HMOS o registrador PCON só contém o bit SMOD; os demais não devem ser utilizados.

10.2. MODO IDLE

A instrução que ativa o bit IDL é a última a ser executada antes que a CPU entre no modo Idle. Neste modo só funciona a interrupção, os timers e a porta serial. A CPU fica congelada sem o clock. O consumo é cerca de 85% do consumo normal.

Todo o status é preservado:

- SP, PC, PSW, Acc e todos os registradores,
- os pinos das portas mantêm os mesmos estados,
- ALE = PSEN = High.

Há duas maneiras de terminar o modo Idle:

- por uma interrupção que esteja habilitada,
- por reset.

Se uma interrupção termina o modo Idle, a instrução a ser executada depois do RETI é a que vem em seguida à que ativou o bit IDL. Na figura 10.3 há um esquema que ilustra este funcionamento.

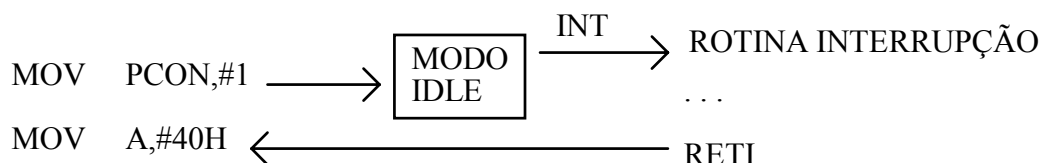


Figura 10.3. Retorno do modo Idle através de uma interrupção.

Os Flags GF0 e GF1 podem ser usados para dar indicação se a interrupção aconteceu no modo normal ou no modo Idle. A rotina que ativa o bit IDL deve antes ativar um dos dois flags (GF0 ou GF1) para indicar que a CPU entrou no modo Idle.

O Reset é a outra forma de terminar o modo Idle. Como o oscilador está funcionando, são necessários apenas 24 períodos de clock. O Reset coloca em zero o bit IDL de forma assíncrona e a CPU reassume a execução a partir da instrução que ativou o IDL. Podem acontecer 2 a 3 ciclos de máquina antes que a CPU retome o controle.

O hardware interno inibe o acesso à RAM interna durante o período de modo Idle mas não os acessos aos pinos das portas. Para evitar erros, a instrução que vem em seguida à ativação do IDL não deve escrever nas portas ou na memória externa.

10.3. MODO POWER DOWN

A instrução que ativa o bit PD é a última a ser executada antes que a CPU entre no modo Power Down. Nesse modo o oscilador fica parado e assim todas as funções se congelam mas o conteúdo da RAM interna e dos SFR são mantidos. O consumo é de aproximadamente 10 μ A.

Em modo power down são mantidos:

- RAM interna e SFR
- Pinos das portas
- ALE = PSEN = LOW

A única maneira de sair do Power Down é por Reset, que vai alterar o conteúdo dos SFR mas não alterará o conteúdo da RAM interna. Assim, os valores importantes deverão estar armazenados na RAM interna. Esse Reset deve esperar a partida do oscilador (pelo menos 10 ms). O programa recomeçará do início (RESET), ao contrário do modo IDLE.

Em Power Down pode-se baixar o Vcc até 2V; alguns cuidados devem ser observados:

- Não baixar o Vcc antes do Power Down
- Levantar o Vcc antes de sair do Power Down.

Uma possível utilização do modo power down pode ser realizada usando um sensor que capte a redução de Vcc (p.ex. 4,75V) e ative uma interrupção externa com prioridade alta para entrar no modo power down.

10.4. PROGRAMAÇÃO DA EPROM (8751)

Algumas versões da família MCS-51 possuem uma EPROM interna. A tabela da figura 10.4 relaciona algumas destas CPUs.

CPU	TAMANHO	TIPO	VPP	TEMPO
8751	4 KB	HMOS	21 V	4 min
8751H	4 KB	HMOS	21 V	4 min
87C51	4 KB	CHMOS	12,75 V	13 s
8752B	8 KB	HMOS	12,75 V	26 s

Figura 10.4. Algumas versões do MCS-51 com EPROM interna.

A CPU 8751H é programada com 21 V, usando 50 ms para cada byte, o que dá cerca de 4 minutos ($4K * 50 \text{ ms} = 200 \text{ s}$). As CPUs 87C51 e 8752BH usam um modo de programação chamado de "Quick Pulse" que é feito com 12,75 volts e 25 pulsos de 100 μs para cada byte, resultando em um menor tempo de programação.

Durante a programação existem 3 configurações usadas:

- Programação
- Verificação
- Programação do bit de segurança

A figura 10.5 ilustra a habilitação desses modos.



MODO	RST	*PSEN	ALE	*EA	P2.7	P2.6	P2.5	P2.4
PROGRAMAÇÃO	1	0		VPP	1	0	X	X
INIBIDO	1	0	1	X	1	0	X	X
VERIFICAÇÃO	1	0	1	1	0	0	X	X
SEGURANÇA	1	0		VPP	1	1	X	X

Figura 10.5. Habilitação dos diversos modos de programação (X → don't care).

10.4.1. Programação

Para a programação é necessário que o oscilador esteja funcionando com uma frequência de 4 a 6 MHz. A figura 10.6 ilustra a operação em modo programação.

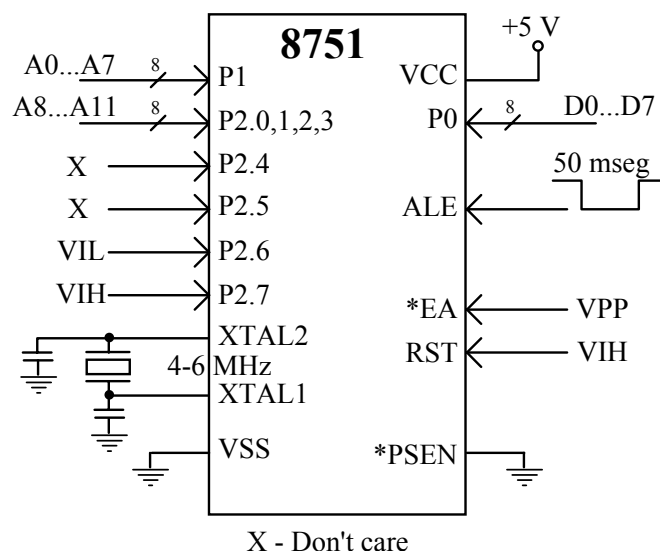


Figura 10.6. A CPU 8751 em modo programação.

Normalmente EA é mantido em nível alto (VIH) até antes do pulso em ALE. Um pouco antes deste pulso, coloca-se EA = VPP e em seguida envia-se o pulso ALE; depois dos 50 ms faz-se EA = VIH.

10.4.2. Verificação

Se o bit de segurança não for programado, a memória EPROM pode ser lida com finalidades de verificação. Isso pode ser feito depois da programação de cada byte. A única alteração é que P2.7 é colocado em nível baixo para habilitar o buffer de saída por P0. Como P0 não possui pull up interno, um pull up externo de 10 K Ω deverá ser colocado em cada linha de dados. A figura 10.7 ilustra a operação do 8751 em modo verificação.

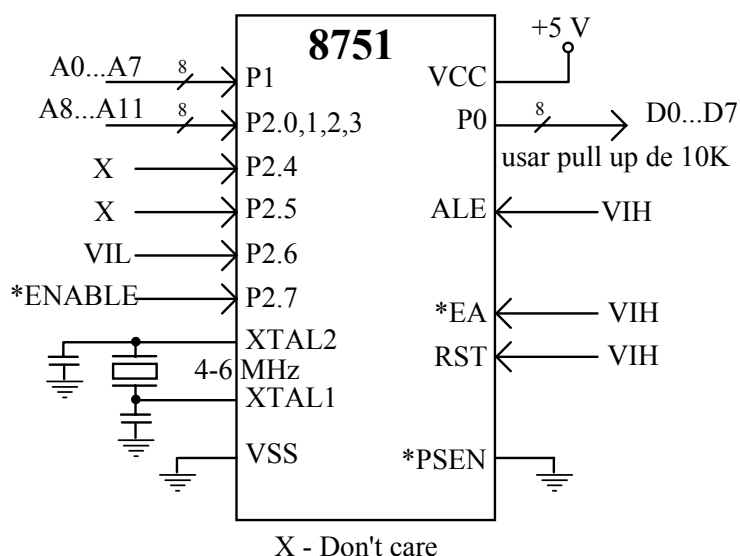


Figura 10.7. O CPU 8751 em modo verificação.

10.4.3. Bit de Segurança

O bit de segurança é uma trava que, quando programada, impede a leitura da EPROM interna. Também impede que a CPU execute programas a partir de uma memória externa. Ao apagar a EPROM se apaga também o bit de segurança. A figura 10.8 ilustra a operação da CPU em modo programação do bit de segurança.

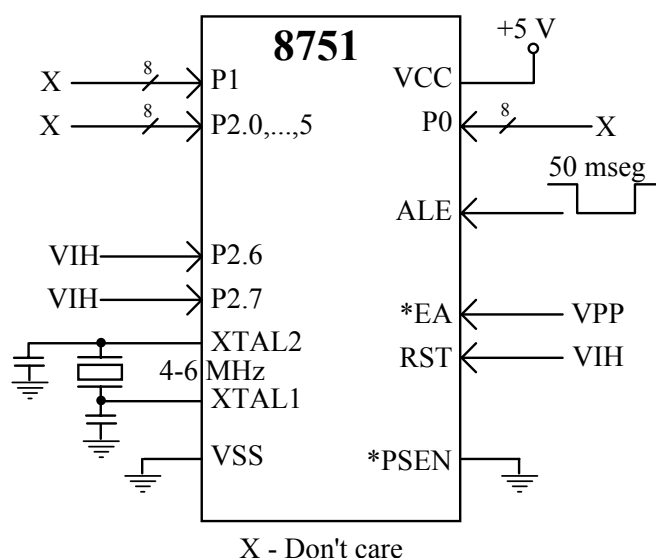


Figura 10.6. A CPU 8751 em modo programação do bit de segurança.

Outros membros da família oferecem técnicas mais sofisticadas de segurança. Por exemplo, o 8751BH oferece um array de 32 bytes onde se pode colocar chaves de criptografia. A cada endereço de EPROM que é lido, os 5 bits de endereço são usados para acessar um dos 32 bytes e com ele é feito um *XOR (NXOR). Se estes 32 bytes estão apagados (em 1), a operação NXOR não produz nenhuma alteração.

10.4.4. Apagamento (8751)

O apagamento da memória EPROM do 8751 é feito por luz ultravioleta de alta densidade (2537 Angstroms e 15 Watts/cm²) durante 20 a 30 minutos a uma distância de 2,5 cm. A EPROM ficará com todos os bits em 1 após o processo.

CAPÍTULO XI

PLACA DE TESTES

11.1. INTRODUÇÃO

Neste capítulo descreve-se o projeto e a operação da Placa de Testes. No capítulo XII estão os detalhes do software que gerencia esta placa. A finalidade da placa é permitir flexibilidade no ensino de microcontroladores. O esquema possibilita a utilização de um computador para assemblear e linkar os programas e depois enviá-los por porta serial à Placa de Testes para que sejam executados.

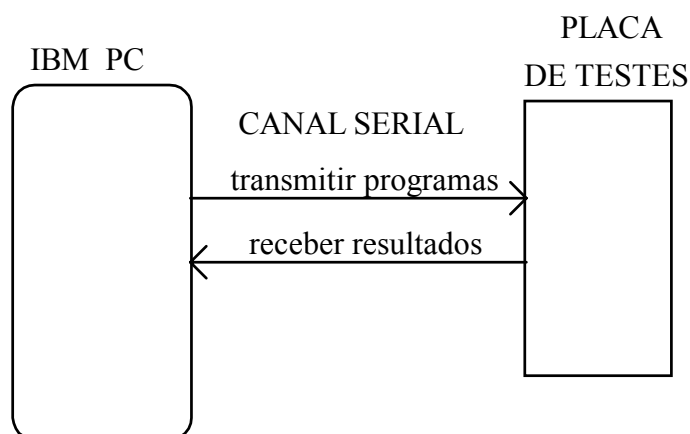


Figura 11.1. Conexão da Placa de Testes com o IBM PC.

O mais usual quando se trabalha em ensino de microcontroladores é montar uma pequena placa onde são montados o microcontrolador, chaves para se interagir, leds para sinalizar alguns resultados e uma memória EPROM onde se gravam os programas. Isto tem uma séria desvantagem: apagar e gravar a EPROM cada vez que se muda o programa. Para cada correção que se faz, repete-se o processo de apagar e gravar. Além de ser um processo lento, significa que se deve ter disponibilidade de um gravador de EPROM.

Com a Placa de Testes não se pretende utilizar a EPROM pois os programas serão transmitidos através da porta serial do PC, escritos na memória de programa do 8031 e executados. A velocidade aumenta bastante pois os erros podem ser rapidamente corrigidos no PC e em seguida transmitidos para a placa.

A placa tem dois modos de operação:

- modo Boot Serial
- modo Execução

Para o funcionamento correto existem duas memórias: uma EPROM e uma RAM estática (SRAM). Quando a placa está em modo Boot Serial, a EPROM trabalha como memória de programa e a memória estática trabalha como memória de dados. Na EPROM está um programa muito simples que tem como função receber o programa que chega pela porta serial e escrevê-lo na RAM estática. Ao terminar a transmissão do programa, o usuário põe a placa em modo Execução e com isto a RAM estática, que era memória de dados, se transforma em memória de programa e a EPROM é desabilitada; com isto o programa recém transmitido é executado. Se for preciso transmitir um novo programa, simplesmente basta colocar a placa em modo Boot Serial e transmitir o novo programa. Há um programa escrito em C (TAR_PRU.C) que ajuda na comunicação com a Placa de Testes.

Para que o circuito seja completo foram colocados recursos para gravar e ler os membros da família MCS-51 que têm EPROM interna: 8751H, 8751BH, 87C51, 87C51FA, 8752BH e outros modelos. Há recursos para gravação usando o modo standard (pulsos de 50 ms) como também usando o modo "quick pulse programming" (pulsos de 100 μ s).

A seguir há uma descrição detalhada de cada item da placa de testes. Os esquemas estão no final deste capítulo e têm os nomes:

- esquema da CPU (cpu.sht)
- esquema da MEMÓRIA (memo.sht)
- esquema da SERIAL (serial.sht)
- esquema do GRAVADOR (grav.sht)

A seguir são apresentados os quatro esquemas:









11.2. ESQUEMA DA CPU (CPU.SHT)

Este esquema é o de maior hierarquia. Aqui está a CPU e também os blocos que representam os demais esquemas: MEMÓRIAS, INTERF-SERIAL e GRAVADOR. A seguir há uma descrição completa de todos os itens que compõem esta placa.

O **CRISTAL** utilizado é de 3,575611 MHz e basta conectá-lo aos pinos (X1 e X2) do oscilador da CPU. Utilizou-se este cristal porque já estava disponível e também porque esta frequência permite que a porta serial opere a 9600 bauds com boa precisão. Este cristal é muito comum no Brasil pois é utilizado em todas as televisões para gerar a frequência da portadora de cor do sistema PAL-M. Um cristal muito comum no Equador é o de 3,58 MHz, que é a frequência da portadora de cor do sistema NTSC e que também pode ser utilizado na Placa de Testes. Um cristal ideal para a geração de um baud rate preciso é um de 3,6864 MHz; um cristal de 11,0592 também é interessante pois permite um baud rate preciso além de permitir um clock maior para a CPU. Os dois capacitores C2 e C3 estão colocados por recomendação do fabricante da CPU e o motivo da utilização é garantir que o cristal oscile na frequência apropriada durante a energização do circuito.

O circuito de **RESET** já foi estudado no capítulo III. Notar que o resistor R2 é de 47 K e que junto com o capacitor de 10 μ F, deve garantir um tempo mais que suficiente para o reset da CPU. O diodo D1 permite uma rápida descarga do capacitor C1 quando se desliga o circuito. Também é possível um reset manual através do pushbutton SW1 e o resistor R1 de 82 Ω é usado para impedir a descarga abrupta do capacitor C1. Deve-se tomar cuidado com os resistores R1 e R2 pois quando o pushbutton SW1 é acionado (para provocar reset) esses dois resistores formam um divisor resistivo que define o valor da tensão na entrada do Schimit Trigger 74LS14. Se R1 não for muito pequeno quando comparado com R2, a tensão na entrada do 74LS14 poderá ser interpretada como "1 lógico" e a CPU não será resetada. Neste circuito, quando se aciona SW1 a tensão do divisor é de: $(5 \cdot 82) / (47000 + 82) = 9 \text{ mV}$ (muito próxima de zero). A figura 11.2 ilustra o circuito de reset e os dispositivos conectados a este sinal. Nesta figura pode-se ver o LED6 (led quadrado de cor amarela) que indica quando o reset manual está acionado; um inversor (74LS14) foi usado para que o led se acenda quando a chave for pressionada.

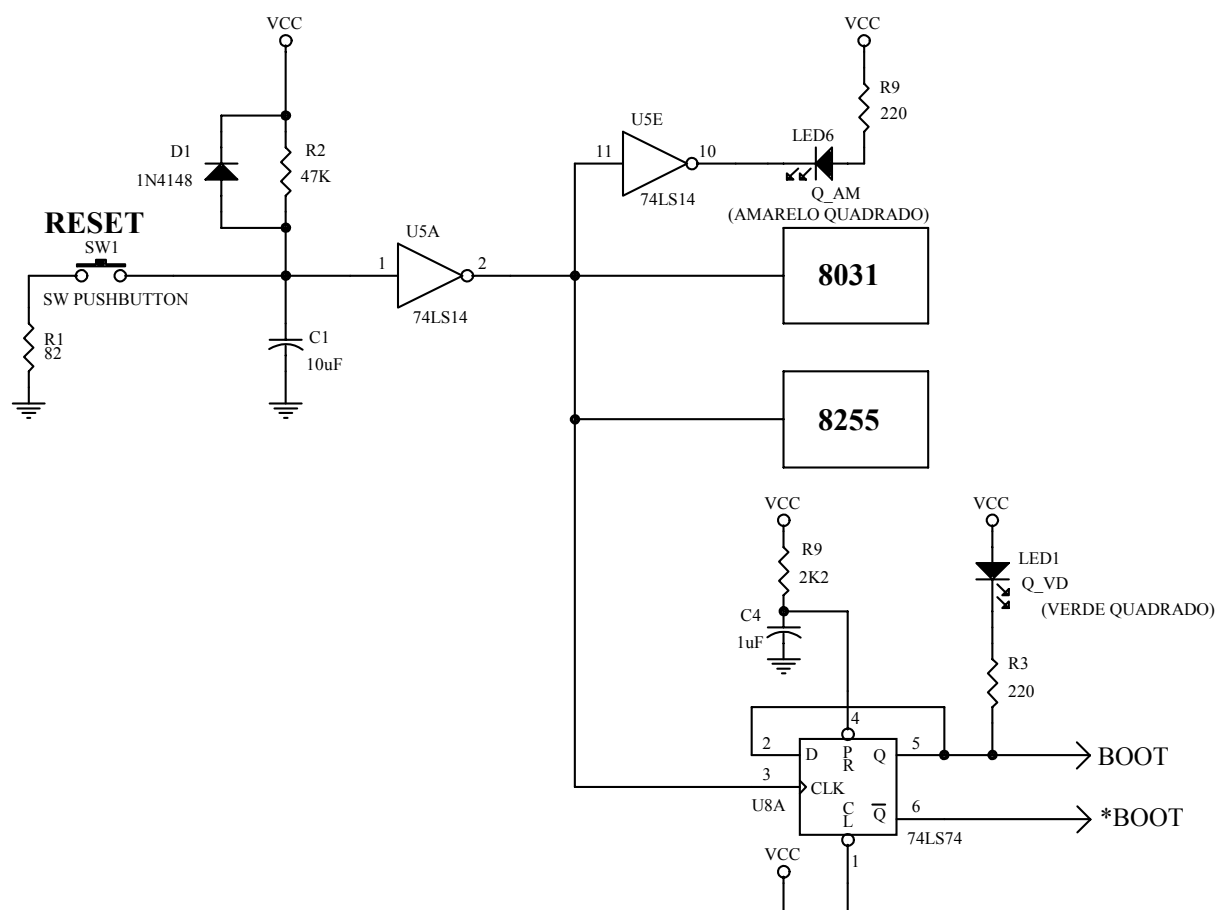


Figura 11.2. Esquema do circuito de "power on" reset.

Na figura 11.2 também se pode notar que o sinal de reset é utilizado pelo 8031 e pelo 8255. Além disto, este sinal também aciona um flip flop tipo D que está conectado em modo "toggled", ou seja, a cada reset o flip flop muda de estado. Este circuito será explicado quando for abordado o item do boot serial.

Existem **TRÊS LEDS SINALIZADORES** (LED2, LED3 e LED4) que servem de saída aos programas dos usuários. Estes leds são acionados por 3 bits da porta P1 (P1.0, P1.1 e P1.2). Para evitar a conexão dos leds diretamente aos pinos da CPU (na verdade não há grandes problemas nisto) e também para que se acendam quando se escrever 1 nos bits da porta P1, foram usados inversores 74LS14 (que estavam sobrando). A figura 11.3 ilustra em detalhes a conexão dos leds com os bits da porta P1.

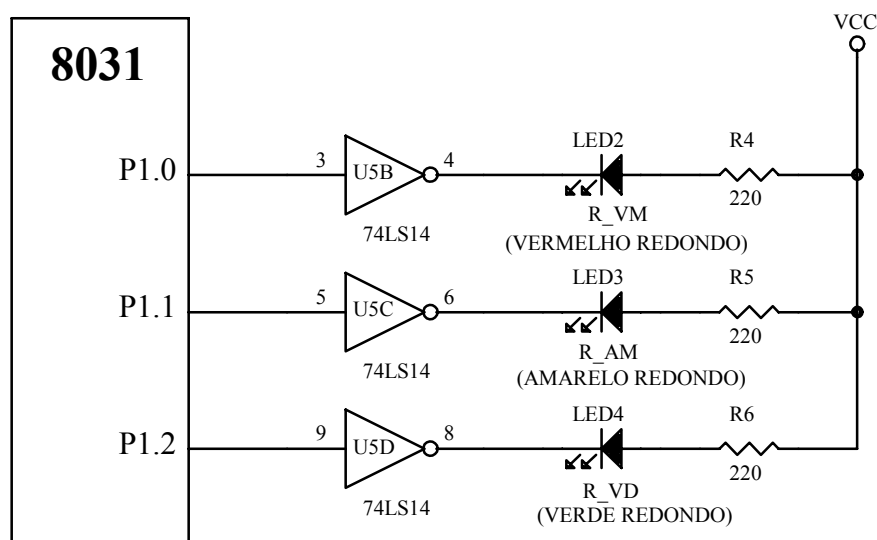


Figura 11.3. Conexão dos leds de sinalização.

Os **DOIS PUSHBUTTONS** (SW2 e SW3) permitem que os usuários interajam com a CPU. Pode-se escrever programas que respondam ao acionamento desses pushbuttons. Notar que os pushbuttons não possuem resistores de "pull up" porque aproveitam o pull up interno que há na porta P1. Para que se possa ler o estado do pushbutton SW2 é necessário programar 1 no bit P1.3, enquanto que para trabalhar com o pushbutton SW3 é necessário programar 1 em *INT1(P3.3) e em T1(P3.5). Quando a chave está aberta, lê-se 1 e quando acionada, é lido 0. Deve-se notar que não há nenhum recurso por hardware para reduzir o bouncing. O pushbutton SW3 pode provocar interrupções e também acionar a entrada do contador/temporizador 1. A figura 11.4 ilustra as conexões dos pushbuttons com a CPU.

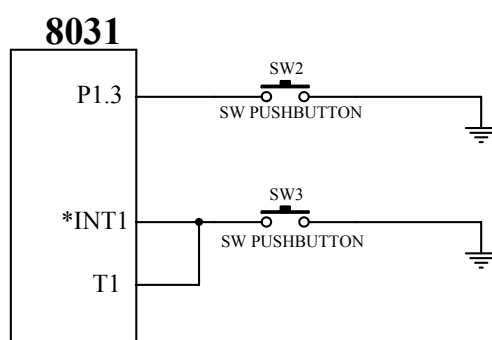


Figura 11.4. Conexão dos pushbuttons com a CPU.

A CHAVE PASSO A PASSO (SW4) está conectada através de um inversor à entrada de interrupção 0. Há um led quadrado de cor vermelha que se acende quando a chave está aberta. Quando a chave está fechada, na saída do inversor há um nível alto e com isto não se ativa a interrupção, mas quando a chave está aberta (LED5 aceso) a saída do inversor vai para um nível baixo, acionando a interrupção 0. Esta interrupção 0 (ou qualquer outra interrupção) pode ser

aproveitada para controlar a execução de programa no modo passo a passo. A figura 11.5 ilustra a conexão da chave passo a passo.

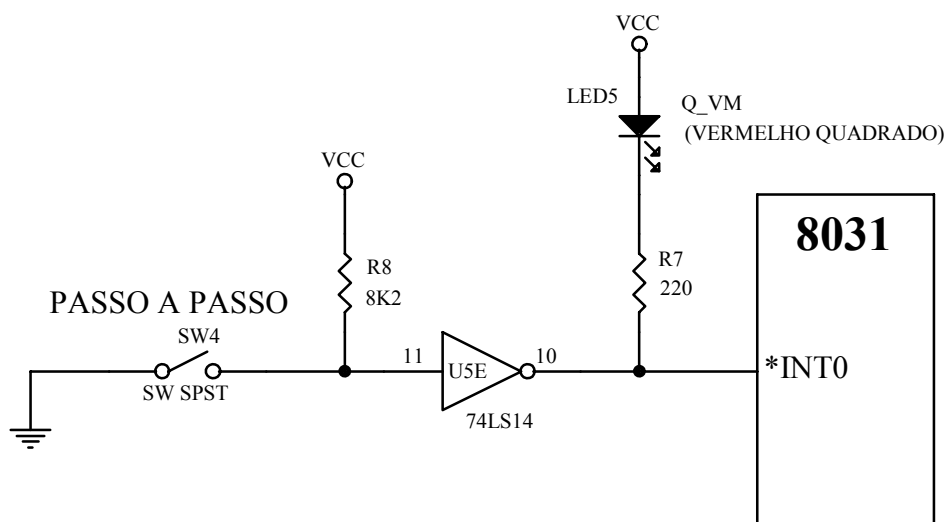


Figura 11.5. Chave para o modo passo a passo.

Como último item do esquema da CPU está a demultiplexação do barramento de endereços. Esta tarefa é realizada por um 74LS373, que congela em sua saída os endereços que estavam presentes na entrada quando ALE fez a transição de alto para baixo (↓). O barramento de endereços é totalmente demultiplexado e entregue aos demais dispositivos, enquanto o barramento de dados é multiplexado. Desde que os dispositivos leiam ou escrevam dados no momento correto, não há nenhum problema em usar o barramento de dados multiplexado (deve-se tomar cuidado com a carga que este barramento pode suportar).

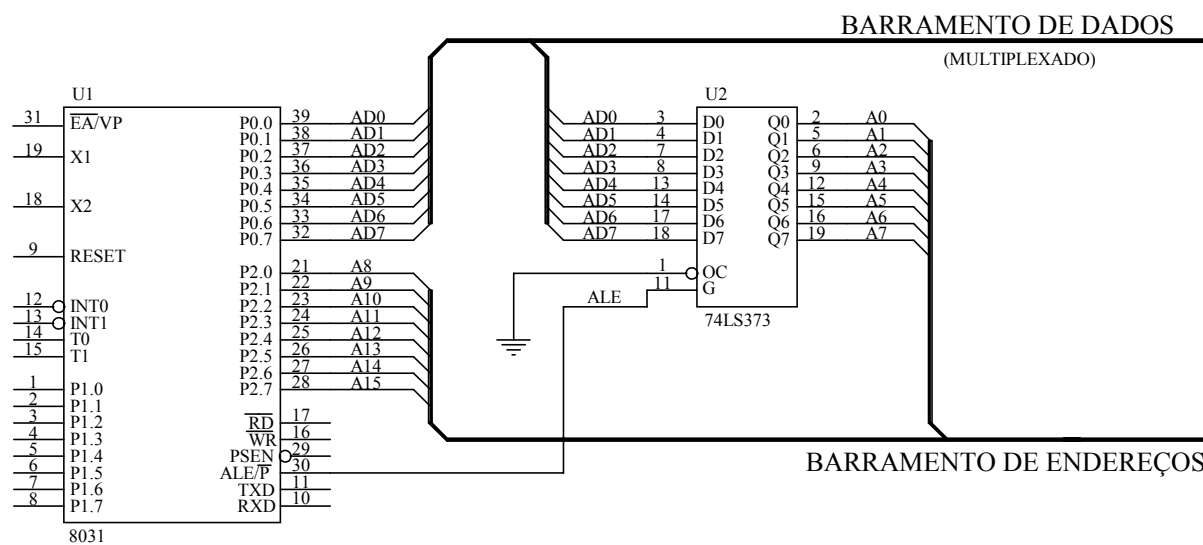


Figura 11.6. Separação dos barramentos de endereços e de dados.

11.3. ESQUEMA DA MEMÓRIA

Para a memória da placa foi utilizada uma EPROM (2732) e uma memória estática (SRAM-6116). A primeira idéia é que a EPROM trabalhe como memória de programa (*PSEN) e a SRAM trabalhe como memória de dados (*RD e *WR). Isto significaria gravar os programas na EPROM e, a cada novo programa, seria necessário apagar e regravar a EPROM. Para evitar este trabalho, há dois modos em que a placa pode operar:

- **MODO BOOT** → A EPROM funciona como memória de programa e a SRAM como memória de dados. Na EPROM está um programa muito simples que recebe dados pela porta serial e os escreve na SRAM (envia-se o programa pela porta serial).
- **MODO EXECUÇÃO** → a EPROM é desabilitada e a SRAM trabalha como memória de programa, executando o programa recém transmitido.

A especificação do modo de operação é controlada por um flip-flop (U8A) conectado em modo toggled. A cada acionamento do RESET este flip-flop muda de um modo para outro. A operação é simples:

1. Coloca-se a placa em modo BOOT (aciona-se o RESET até que o led verde quadrado esteja aceso). Com isto a EPROM será a memória de programa e a SRAM será a memória de dados.
2. Transmite-se o programa a ser executado pela porta serial. O programa da EPROM é responsável por receber o que chega pela porta serial (o programa em formato INTEL.HEX) e escrever na SRAM.
3. Aciona-se o RESET e com isto o flip-flop U8A muda de estado, ou seja, a placa é colocada em modo EXECUÇÃO. Neste modo a EPROM é desabilitada e a SRAM é tomada como memória de programa, executando o que aí está escrito.
4. Se for necessário enviar um novo programa, basta acionar o RESET e a placa vai novamente para o modo BOOT.

A tabela da figura 11.7 ilustra o comportamento das memórias segundo o sinal de BOOT.

BOOT	EPROM	SRAM
0	*PSEN	*RD e *WR
1	X	*PSEN

Figura 11.7. O sinal BOOT controlando a EPROM e a SRAM.

Na figura 11.9 estão as memórias, o circuito responsável por torná-las memória de programa ou dados (U6A, U6B, U6C, U7A) e o flip-flop de BOOT acionado pelo sinal de RESET. A EPROM somente estará operante quando BOOT=1 (*BOOT=0) porque este sinal vai diretamente

para o *CE deste CI; se BOOT=0 a EPROM está desabilitada. Quando BOOT=1, a entrada *WE da SRAM é acionada pelo sinal de *WR e a entrada *OE é acionada por *RD; quando BOOT=0, a entrada *WE está em 1 (não há como escrever) e a entrada *OE é acionada por *PSEN, ou seja, trabalha como memória de programa. Notar que no flip-flop há um circuito (R9 e C4) que garante o preset, ou seja, que sempre se inicie em 1 (BOOT=1, pronto para receber um programa). O fato de existir este circuito de preset necessitou que se aumentasse o tempo do RESET da CPU pois a CPU só deve funcionar depois que o sinal BOOT esteja totalmente estabilizado.

A tabela da figura 11.8 ilustra como está o pino *CE da EPROM e os pinos *WE e *OE da SRAM.

BOOT	EPROM	SRAM	
	*CE	*WE	*OE
0	*PSEN	*WR	*RD
1	1	1	*PSEN

Figura 11.8. BOOT controlando os pinos de leitura e escrita da SRAM e da EPROM.

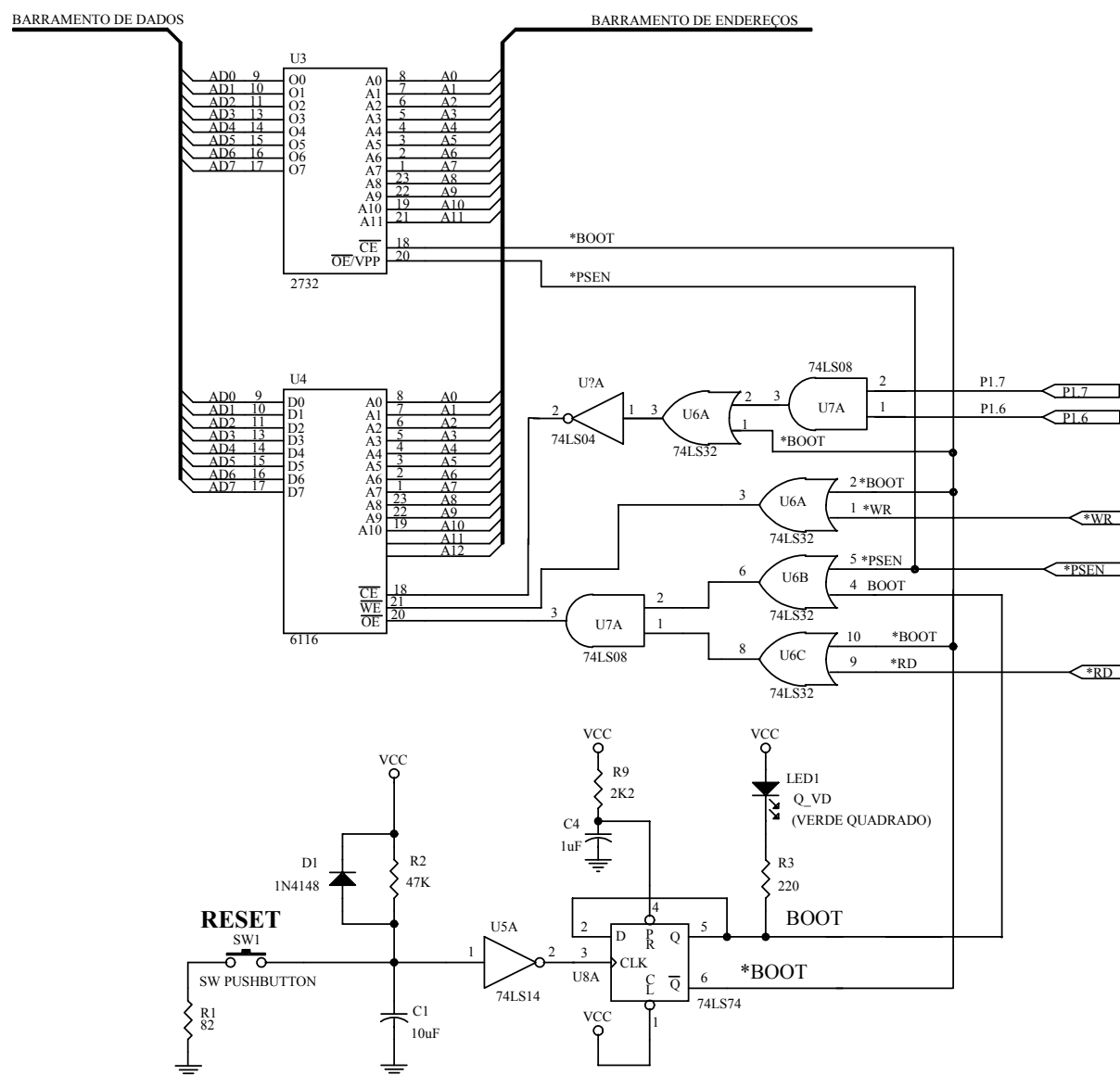


Figura 11.9. Circuito de memória controlado pelo sinal BOOT.

11.4. ESQUEMA DA SERIAL

O microcontrolador 8031 oferece uma porta serial bidirecional que será utilizada para enviar os programas que serão executados pela placa de testes. Pelo pino TXD a CPU transmite os dados e pelo pino RXD os dados são recebidos. O conector serial de 9 pinos dos PCs transmite pelo pino 2 (PC_RX) e recebe pelo pino 3 (PC_TX). A solução parece ser simples pois bastaria conectar as entradas com as saídas. O problema é que no conector serial dos PCs os sinais estão com os níveis de tensão do padrão RS 232 (+/- 12 V) e nos pinos da CPU têm-se níveis TTL. Os conversores RS 232/TTL necessitam de fontes de alimentação de +12 V e -12 V; os mais consagrados são: 1488 e 1489.

Pareceria que para poder operar com a porta serial conectada ao PC deveríamos ligar a Placa de Testes a duas fontes extras (+12 V e -12 V). Mas com isto a placa se tornaria grande porque seriam necessários dois transformadores extras. Mas o que se busca é uma placa simples que permita receber programas do PC e não uma porta serial perfeita.

Converter para TTL os sinais RS 232 gerados por um PC é fácil com a utilização de um transistor. A figura 11.10 ilustra um circuito que converte RS 232 em TTL. Se a entrada PC_TX está em +12 V, o transistor se satura e a saída RXD vai a zero; por outro lado, se a entrada PC_TX está em -12 V, o transistor estará cortado e, devido ao resistor de pull up (R14), a saída vai para VCC. O diodo D12 aumenta a velocidade de transistor quando este faz a transição da saturação para o corte.

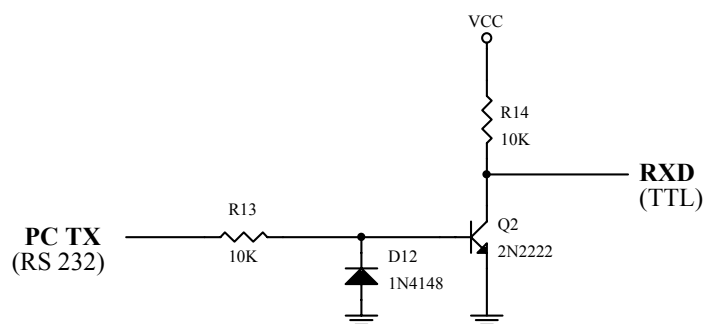


Figura 4.10. Conversor de RS 232 para TTL.

Há algumas tolerâncias na porta serial do PC, as quais poderão ser aproveitadas para simplificar o conversor TTL/RS 232. O primeiro tópico é que a porta serial do PC interpreta como tensão positiva qualquer valor entre +3 V e +12 V. Com isto, podemos utilizar os +5 V da placa em substituição à fonte de +12 V. O mesmo é válido para as tensões negativas, ou seja, a porta serial do PC interpreta como tensão negativa qualquer valor entre -12 V e -3 V. A figura 4.8 ilustra estas faixas de tensão.

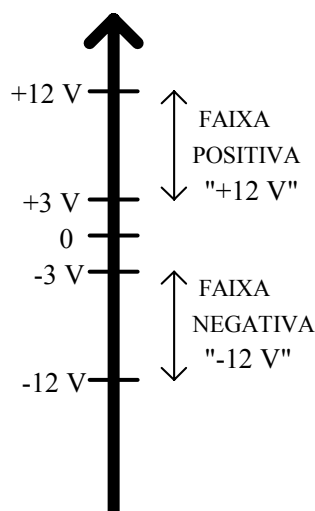


Figura 11.10. Faixas de tensão toleradas pela porta serial do PC.

Se houvesse uma fonte de -12 V disponível a conversão de TTL para RS 232 seria muito simples. A figura 11.11 ilustra o emprego de um transistor para converter os níveis. Quando o sinal TXD do 8031 está em 1, o transistor estará cortado e a saída PC_RX vai para -12 V; quando o sinal TXD está em zero, o transistor se satura e a saída PC_RX vai para +5 V.

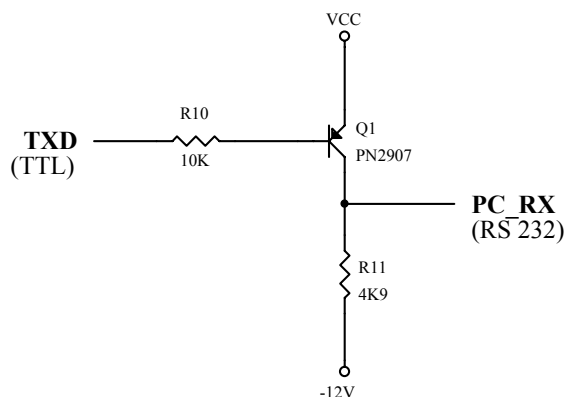


Figura 11.11. Circuito para converter níveis TTL para níveis RS 232.

O problema agora é: onde conseguir uma fonte de -12 V ? Há um truque que se pode usar: uma saída RS 232 quando inativa está com uma tensão negativa (usualmente -12 V), ou seja, o sinal PC_TX quando não usado estará apresentando uma tensão de -12 V. Esta mesma saída, quando começa a transmitir apresenta pulsos de +12 V e de -12 V. Se for usado um circuito capaz de separar as tensões negativas, será obtida uma fonte que pode substituir a fonte de -12 V. A figura 11.12 ilustra este circuito. Notar que o diodo D11 conectado a PC_TX separa as tensões negativas, que são armazenadas no capacitor C10. Quando há tensões positivas, esse diodo estará cortado. O diodo D10 está em paralelo com o capacitor para garantir que as tensões nunca cheguem a um valor positivo. Um circuito deste tipo consegue fornecer uma tensão próxima de -6 V, mas que é suficiente para que o conversor TTL/RS 232 funcione.

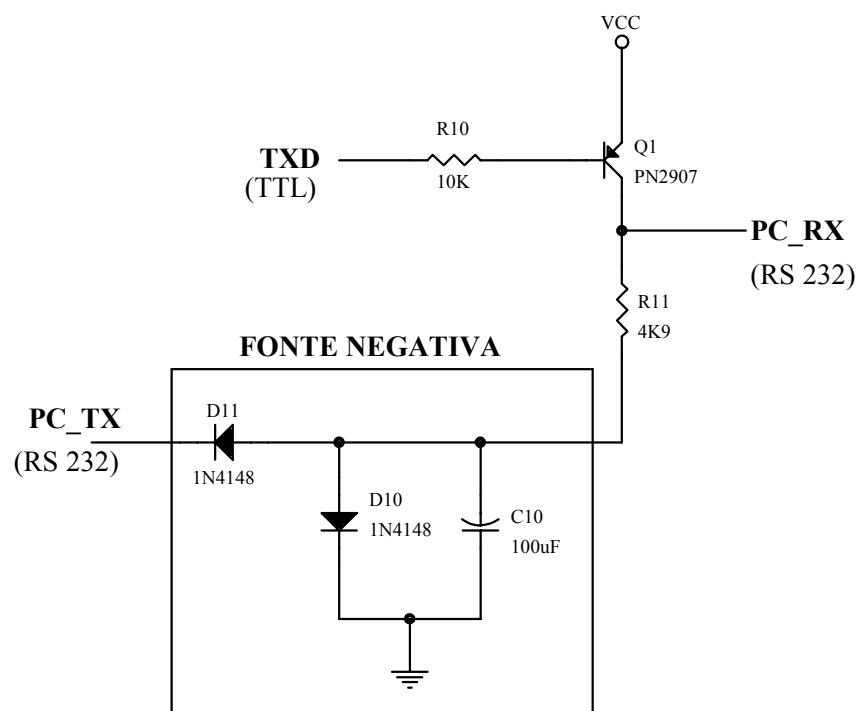


Figura 11.12. Conversor TTL/RS 232 com a fonte negativa.

Agora é possível fazer as conversões RS 232 e TTL que são necessárias para que a porta serial opere. Cabe lembrar que este conversor é muito limitado e que funciona apenas a distâncias pequenas. Deve-se testar quando for empregado a distâncias maiores e com velocidades mais altas. O esquema completo da interface serial (serial.sht) está no item 11.2.

Para complementar o presente texto, é apresentada na figura 11.13 uma tabela com as funções dos pinos para os conectores de 9 pinos e 25 pinos.

NOME	DIR.	25 PINOS	9 PINOS	DESCRIÇÃO
TD	→	2	3	Transmitir dados
RD	←	3	2	Receber dados
DSR	←	6	6	Conjunto de dados pronto
DTR	→	20	4	Terminal de dados pronto
RTS	→	4	7	Solicitação para enviar
CTS	←	5	8	Livre para enviar
DCD	←	8	1	Deteção de portadora
RI	←	22	9	Indicador de chamada
TERRA	—	7	5	Terra de sinal
TERRA P.	—	1	—	Terra de proteção

Figura 11.13. Descrição dos sinais dos conectores seriais.

Para economizar espaço na placa, usa-se um adaptador como fonte de alimentação. O adaptador é responsável por gerar uma tensão retificada, mas não regulada, na faixa de 7,5 a 10 V. O regulador da placa (LM7805) é responsável por gerar os 5 volts regulados. Normalmente os adaptadores são de má qualidade e geram uma tensão que depende da corrente; quanto mais corrente se necessita, menor é a tensão. Para evitar uma dissipação de potência muito grande sobre o regulador, recomenda-se testar com diversas tensões, iniciando por uma bem baixa como, por exemplo, 3V. Os capacitores C17 e C18 são usados para garantir a regulação.

Para a fonte de alimentação de 25 V, que é usada somente quando se grava a EPROM das CPUs, usa-se uma fonte externa que deve fornecer 25 V. O capacitor C19 é usado, por segurança, para garantir a regulação. Se for usada uma fonte boa, esse capacitor não será necessário. A figura 11.14 ilustra as duas fontes de alimentação.

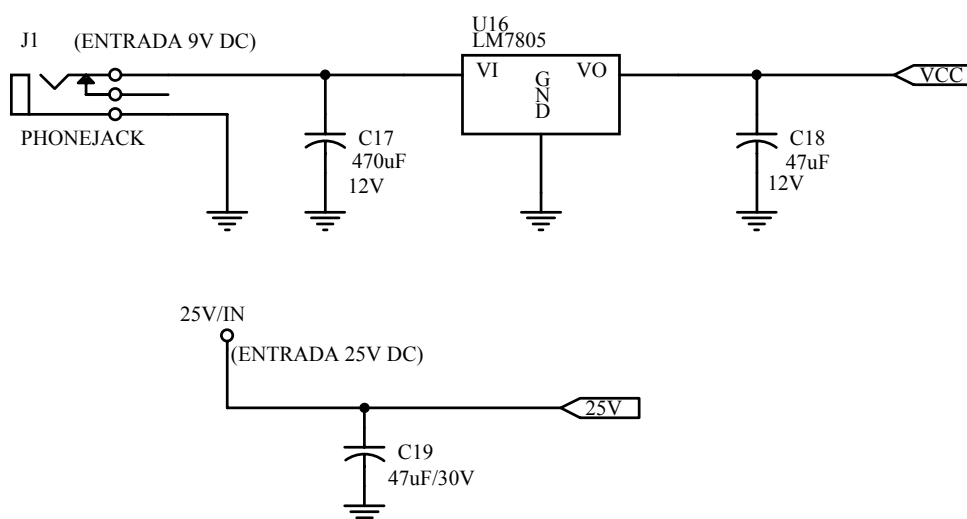

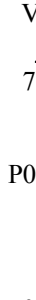







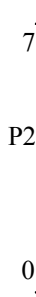





Figura 11.14. Fontes de alimentação de 5 V e 25 V.

11.5. ESQUEMA DO GRAVADOR

Para o gravador de EPROM se buscou um projeto que permitisse gravar todas as CPUs 8751 e 8752. Portanto, foi feita uma análise de que sinais seriam necessários para cada pino. A figura 11.15 apresenta uma tabela com todos os pinos e os sinais necessários. Por essa figura se pode concluir que o pino *EA/VPP é o mais difícil pois, de acordo com a CPU, deve fornecer 3 níveis de tensão: 21 / 12,75 / 5 V. Pelo pino ALE/*PROG enviam-se pulsos de 50 ms para a gravação standard ou pulsos de 100 μ s para a gravação usando "quick pulse programming".

			SOCKET ZERO FORCE							
8752BH	87C51 8751BH	8751H				8751H	87C51 8751BH	8752BH		
A0	A0	A0	1	 P1	VCC	40	5V	5V	5V	
A1	A1	A1	2		 P0		39	D0	D0	D0
A2	A2	A2	3				38	D1	D1	D1
A3	A3	A3	4				37	D2	D2	D2
A4	A4	A4	5				36	D3	D3	D3
A5	A5	A5	6				35	D4	D4	D4
A6	A6	A6	7				34	D5	D5	D5
A7	A7	A7	8			0	33	D6	D6	D6
H	H	H	9	RST		32	D7	D7	D7	
X	X	X	10	 P3	*EA/VPP	31	5/21 V	5/12,75 V	5/12,75 V	
X	X	X	11		ALE/*PROG	30	 P1	 P2	 P2	
X	X	X	12		*PSEN	29				
X	X	X	13		 P2		28	L/H	L/H	L/H
X	X	X	14				27	L/H	L/H	L/H
X	X	X	15				26	X	X	X
L/H	L/H	X	16				25	X	X	A12
L/H	L/H	X	17			24	A11	A11	A11	
4-6 MHz	4-6 MHz	4-6 MHz	18	XTAL2			23	A10	A10	A10
4-6 MHz	4-6 MHz	4-6 MHz	19	XTAL1			22	A9	A9	A9
			20	GND		21	A8	A8	A8	

P1 é um pulso de 50 milissegundos

P2 é um pulso de 100 microsegundos

Figura 11.15. Sinais necessários para gravar os diferentes tipos de microcontroladores com EPROM.

A parte mais difícil deste gravador é a geração das tensões de programação (5 V, 12,75 V e 21 V). Além disso, deve-se ter um controle digital sobre essas tensões. A solução é obtida com o uso do LM317 (regulador ajustável), que tem a capacidade de manejar tensões de 1,2 V até 37 V. A figura 11.16 ilustra o emprego deste CI.

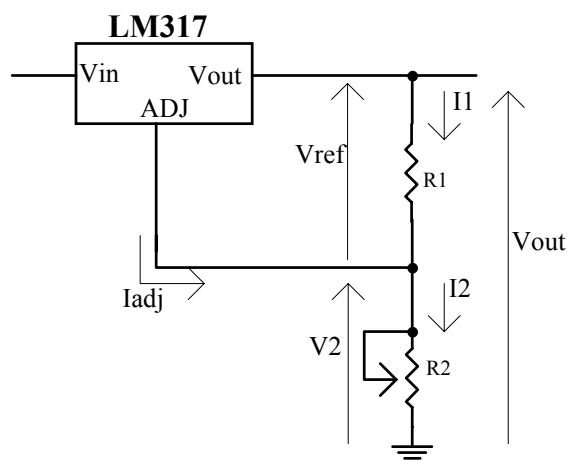


Figura 11.16. Uma sugestão de emprego do LM317.

Quando em operação, o LM317 apresenta uma tensão nominal de referência de 1,25 V entre a saída e o pino de ajuste. Essa tensão de referência é gerada através do resistor R1 e como a tensão é constante, a corrente I2 flui pelo resistor R2 gerando a tensão de saída Vout. A corrente Iadj representa um erro e o dispositivo está preparado para minimizá-la; esta corrente tem um valor típico de 100 µA. O equacionamento é muito simples e é feito a seguir:

$$V_{out} = V_{ref} + V_2$$

$$I_2 = I_1 + I_{adj} = \frac{V_{ref}}{R_1} + I_{adj}$$

$$V_2 = R_2 I_2 = R_2 \left(\frac{V_{ref}}{R_1} + I_{adj} \right)$$

$$V_{out} = V_{ref} + R_2 \left(\frac{V_{ref}}{R_1} + I_{adj} \right)$$

$$V_{out} = V_{ref} \left(1 + \frac{R_2}{R_1} \right) + R_2 I_{adj}$$

O que se pretende é descobrir que valores de R2 se deve ter para as tensões de 5 V, 12,75 V e 21 V. O equacionamento a seguir apresenta uma expressão genérica para R2.

$$R_2 = \frac{V_{out} - V_{ref}}{\left(\frac{V_{ref}}{R_1} + I_{adj} \right)}$$

Quando são usados os valores típicos do fabricante (lista abaixo), tem-se:

$$(V_{ref}=1,25 \text{ V} \quad I_{adj} = 100 \text{ µA} \quad R_1 = 220 \text{ Ω})$$

$$R_2 = \frac{V_{out} - 1,25}{5,782} \text{ KΩ}$$

A tabela da figura 11.17 apresenta os valores de resistência necessários para gerar as tensões de interesse.

TENSÃO	R2
21 V	3,416 kΩ
12,75 V	1,989 KΩ
5 V	649 Ω

Figura 11.17. Valores de R2 para gerar as diversas tensões.

A maneira mais simples de gerar essas 3 tensões é conectar ao LM317 esses três valores de resistores e controlar seu caminho para terra com transistores. Pode-se usar buffers não inversores com coletor aberto. A figura 11.18 ilustra este circuito. Deve-se notar que o resistor responsável pelos 21 V está permanentemente conectado à terra, ou seja, quando se desabilita os dois transistores a saída vai a 21 V. É preciso saber os valores de resistores que, em paralelo com o resistor de R16, permitam gerar as outras tensões. Na verdade o calculo é simples.

Para 12 V: $[1/3416 + 1/R18 = 1/1989] \rightarrow R18 = 4,761 \text{ K}\Omega$.

Para 5 V : $[1/3416 + 1/R17 = 1/649] \rightarrow R17 = 800 \Omega$.

Para permitir um ajuste preciso das tensões usam-se potenciômetros para cada resistor. É claro que primeiro se ajusta o potenciômetro dos 21 V e em seguida os outros dois.

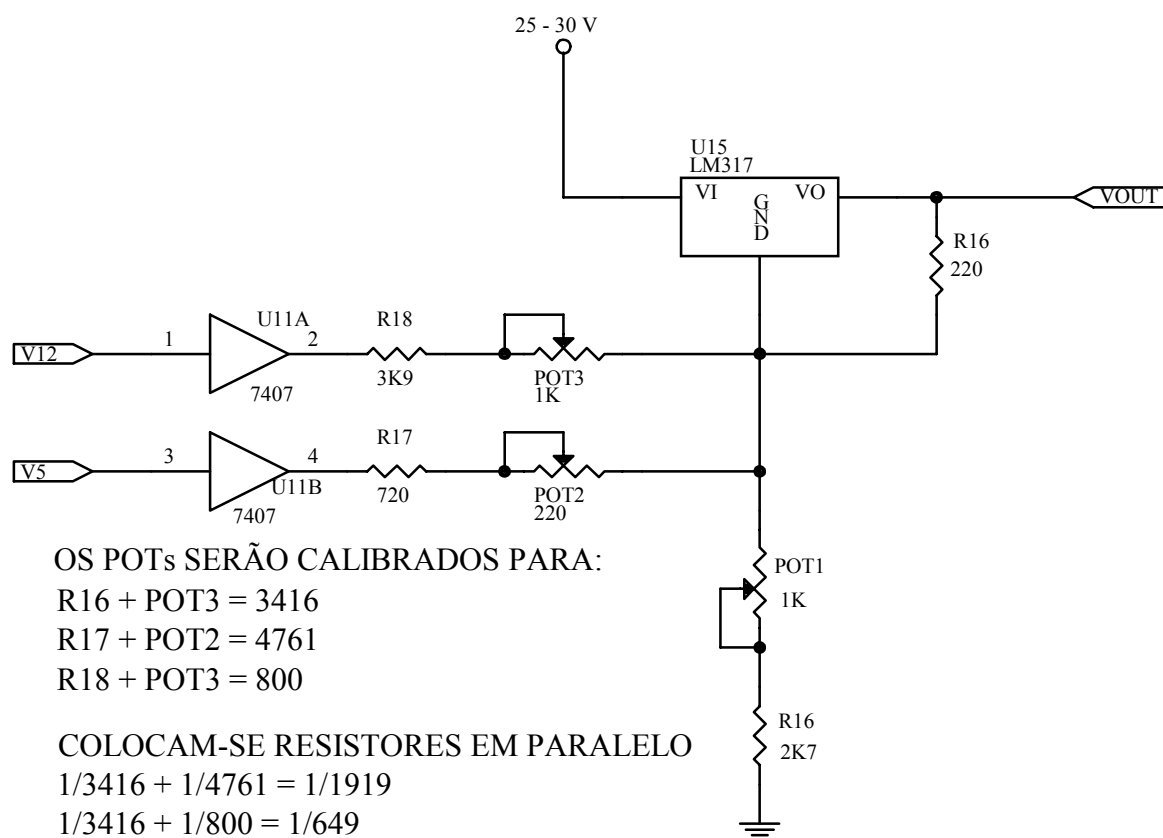


Figura 11.18. Geração das tensões de 21 V; 12,75 V e 5V.

No item 11.2 está o esquema completo do gravador de EPROM (GRAV.SHT). O 8255 é responsável pela conexão com os dados da EPROM, a geração dos endereços e alguns sinais de controle; o 8255 (PC7) vai também gerar o pulso de programação. O 74LS373, além de alguns sinais de controle, também controla o regulador LM317, gerando as diversas tensões. Como já foi estudado, a porta P0 não tem pull up interno e portanto tornam-se necessários os resistores de 10 K Ω para realizar o pull up externo. O bit P1.7, quando em zero, permite que se escreva no 8255 e o bit P1.6, quando em zero, permite que se escreva no 75LS373. Logo depois do RESET, P1.7 e P1.6 estarão em nível 1, quer dizer, os dois CIs (8255 e 373) estarão desabilitados. Nunca se deve operar com os dois CIs habilitados ao mesmo tempo pois uma escrita vai escrever nos dois simultaneamente.

BIBLIOGRAFIA

Almoyna, Julio Martínez. "**Dicionário de Português-Espanhol**", Ed Porto, Portugal.

Becker, Idel "**Dicionário Espanhol-Português e Português-Espanhol**", décima segunda edição, Ed. Nobel, São Paulo, SP, 1992.

Braga, Nilton. "**RS-232 Técnicas de Interface**", terceira tiragem, Ed. EBRAS, Rio de Janeiro, RJ, 1990.

Kernighan, Brian W. & Ritchie, Dennis M. "**The C Programming Language**", second edition, Ed. Prentice Hall - Software Series, Englewood Cliffs, New Jersey, 1988.

Manual da INTEL. "**8 Bit Embedded Controller Handbook**".

Purdum, Jack. "**C Programmer's Toolkit**", Ed. QUE Corporation, Carmel, Indiana, 1989.

Silva Jr, Vidal P. "Aplicações práticas do microcontrolador 8051", 9ª ed., Ed. Érica, São Paulo, 1998.

Nicolosi, Denys E. C. "Microcontrolador 8051 detalhado", Ed. Érica, São Paulo, 2000.